PhD-FSTM-2020-53
The Faculty of Science, Technology and Medicine

DISSERTATION

Presented on 24/09/2020 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Oyebade Kayode OYEDOTUN
Born on 08 March 1989 in Oyo (Nigeria)

ANALYZING AND IMPROVING VERY DEEP NEURAL NETWORKS: FROM OPTIMIZATION, GENERALIZATION TO COMPRESSION

Dissertation defence committee

Assist. Prof. Djamila Aouada, dissertation supervisor
Senior Research Scientist, Université du Luxembourg

Assist. Prof. Bissyande Tegawendé, Chairman
Senior Research Scientist, Université du Luxembourg

Prof. Dr. Björn Ottersten, Vice Chairman
Professor, Université du Luxembourg

Prof. Dr. David Fofi
Professor, University of Burgundy, France

Assist. Prof. Marwan Hassani
Eindhoven University of Technology, Eindhoven, The Netherlands

"*Hang on to your hat. Hang on to your hope. And wind the clock, for tomorrow is another day.*" – *E.B. White*

To my beloved family, friends and late Prof. Dr. Adnan Khashman

Acknowledgements

My experience at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) at the University of Luxembourg has been extraordinary and thrilling. First, my profound gratitude goes to my advisors Dr. Djamila Aouada and Prof. Björn Ottersten for their guidance and mentorship throughout the period of my programme. Their insightful feedbacks were extremely useful in improving the outcome of the different projects that we embarked on. Importantly, I cannot but acknowledge the impressive work rate and commitment of Dr. Djamila to the success of this programme, and my all-around growth.

Second, I would like to appreciate my immediate collaborators, Abdelrahman Shabayek and Kassem Al Ismaeil. Their good spirit, contributions and excitement over various projects, which we worked on together were very helpful. In addition, my sincere appreciation goes to all the team members of the Computer Vision, Imaging and Machine Intelligence (CVI^2) Research Group including Konstantinos Papadopoulos, Renato Baptista, Alexandre Saint, Anis Kacem, Enjie Ghorbel and Adel Mohamed Ali.

Finally, this work wouldn't have been possible without the financial support of the Fonds National de la Recherche (FNR - National Research Fund, Luxembourg) with reference number R-AGR-0424-0605-D/Björn Ottersten, which was obtained through the University of Luxembourg.

Index

1 Introduction **1**

1.1 Motivation and Scope 3

1.2 Challenges in Training Very Deep Neural Network 5

1.2.1 Alleviating training problems of Very Deep Neural Networks 5

1.2.2 Explaining the Role of Skip Connections in Very Deep Neural Networks 5

1.3 Challenges in Deep Neural Network Compression 6

1.3.1 Joint Knowledge Transfer for Model Compression 6

1.3.2 Sparsity Enforcing Penalty for Model Compression 6

1.4 Challenges in Multimodal Learning for Deep Neural Network 7

1.4.1 DNN architecture for Multimodal Learning 7

1.4.2 Imbalanced Information Quality in Multimodal Data 7

1.5 Objectives and Contributions 7

1.5.1 Residual Network with Improved Generalization Performance 7

1.5.2 Reformulated Highway Neural Network for Improved Training 8

1.5.3 Training Very Deep Neural Networks without Skip Connections 8

1.5.4 Understanding Very Deep Neural Networks with Skip Connections . . 9

1.5.5 Deep Neural Network Compression 9

1.5.6 Multimodal Learning with Deep Neural Network 10

1.6 Publications 10

1.7 Thesis Outline 12

2 Background **15**
2.1 Optimizing Very Deep Neural Networks 15
2.1.1 Modifying model architecture 15
2.1.2 Modifying the Training Scheme for the Models 17
2.2 Model compression 18
2.2.1 Model Pruning 18
2.2.2 Knowledge Distillation 18
2.3 Deep Neural Network for Multimodal Learning 21

3 Improving the Generalization of Very Deep Neural Network with Skip Connections **22**
3.1 Introduction 23
3.2 Related work 25
3.3 Problem Statement 26
3.4 Residual Learning with Stochastic Input Skip Connections 28
3.5 Maxout S-ResNet with Elastic Net Regularization 30
3.5.1 Learning units activation function via maxout 30
3.5.2 Elastic Net Regularization (ENR) 31
3.5.3 Feature standardization 31
3.6 Experiments and discussion 33
3.7 Conclusion 36

4 Reformulated Highway Neural Network for Improved Training **38**
4.1 Introduction 39
4.2 Related work 43
4.2.1 Theoretical motivation for deeper models 43
4.2.2 Challenges of training deeper models 43
4.2.3 Model performance with depth 44
4.2.4 Approaches for training very deep networks 45
4.3 Background and problem statement 46

4.3.1 Background: highway network 47
4.3.2 Problem statement 48
4.4 Proposed approach 51
4.4.1 Highway network with gate constraints 51
4.4.2 Highway network with gate constraints, and concatenated and compressed features 55
4.5 Experiments 57
4.5.1 Model training settings and evaluation criteria 58
4.5.2 CIFAR-10 and CIFAR-100 datasets 60
4.5.3 Fashion-MNIST dataset 62
4.5.4 SVHN dataset 63
4.5.5 Imagenet-2012 (ILSVRC) dataset 65
4.5.6 Training and inference time 65
4.5.7 GPU memory requirement 68
4.5.8 Qualitative evaluation of gating units activations 69
4.5.9 Ablation studies 71
4.6 Conclusion 71

5 Training Very Deep Neural Network without Skip Connections 73
5.1 Introduction 74
5.2 Related work 76
5.3 Background: the problem of training very deep PlainNets 77
5.3.1 Vanishing and exploding gradients 77
5.3.2 Near singularities 81
5.4 Proposed approach: training very deep PlainNets 83
5.4.1 Proposed PlainNet 83
5.4.2 Preliminary experiments 85
5.4.3 Comparison with ResNet 88
5.5 Experiments 90

5.6 Conclusion 92

6 Explaining Deep Neural Networks with Skip Connections and Summed Hidden Representations 93

6.1 Introduction 94
6.2 Related work 95
6.3 Background and preliminaries 97
6.3.1 Background: importance of studying linear models 97
6.3.2 Preliminaries: PlainNet, ResNet and ResNeXt 98
6.4 Theoretical analysis of the role of skip connections for optimization 100
6.4.1 Forward-pass: loss of basis in hidden representations 101
6.4.2 Foward-pass: Without the Layer Weights Joint Diagonalization Assumption 105
6.4.3 Backpropagation: error gradients singularity 113
6.5 Theoretical Analysis of Skip connections for Model Generalization 115
6.5.1 PlainNet 116
6.5.2 ResNet and ResNeXt 116
6.6 Experiments 118
6.6.1 Datasets, settings and evaluations 118
6.6.2 Results and discussion 119
6.6.3 Results for linear models 123
6.6.4 Skip connections in hindsight 125
6.7 Conclusion 126

7 Understanding Deep Neural Networks with Skip Connections and Concatenated Hidden Representations 127

7.1 Introduction 128
7.2 Related Work 130
7.3 Preliminaries 131
7.3.1 Plain network (PlainNet) 131

7.3.2 Concatened Network of hidden representations (ConcNeXt) 132
7.4 Theoretical Study of DNNs with Skip Connections and Concatenated Hidden Represenations 133
7.4.1 PlainNet (PlainNet) 134
7.4.2 Concatened Network of aggregated hidden representations (ConcNeXt) 135
7.5 Experiments 137
7.5.1 Experimental settings 137
7.5.2 Results for models with non-linear linear activation 138
7.5.3 Results for models with the linear activation function 142
7.6 Cardinality increase in ConcNeXt and model performance 144
7.6.1 Main insights into models concatenated hidden representations 144
7.7 Conclusion 145

8 Structured Compression of Deep Neural Networks with Debiased Elastic Group Lasso 146
8.1 Introduction 147
8.2 Related work 150
8.3 Background and problem statement 151
8.3.1 Background 151
8.3.2 Problem statement 152
8.4 Proposed approach 153
8.4.1 Debiased Elastic Group LASSO (DEGL) 153
8.4.2 Evaluation metrics 155
8.5 Experiments 157
8.5.1 Main experiments 157
8.5.2 Results 158
8.5.3 Ablation studies 166
8.6 Conclusion 170

9 Deep Neural Network Compression with Teacher Latent Subspace Learning and LASSO 171

9.1 Introduction 172

9.2 Related work 175

9.3 Background and problem statement 177

9.3.1 Background: KD and FitNet 177

9.3.2 Problem statement 178

9.4 Proposed latent subspace learning for compression 180

9.4.1 Latent subspace reparameterization 180

9.4.2 Parameters assembly, fine-tuning and pruning 183

9.4.3 Subspace learning, mutual information and convergence 185

9.5 Experiments 186

9.5.1 General settings 186

9.5.2 Results and discussion 187

9.5.3 Ablation studies 198

9.6 Conclusion 201

10 Facial Expression Recognition with Multimodal Learning of RGB-depth Data 203

10.1 Introduction 204

10.2 Literature review 206

10.3 Problem formulation 207

10.4 Proposed facial expression framework 209

10.4.1 Dataset 209

10.4.2 Data pre-processing 209

10.4.3 Learning pipeline 211

10.5 Experiments 215

10.5.1 Pipeline training 215

10.5.2 Experimental setup 215

10.5.3 Results and discussion 216

10.6 Conclusion 218

11 Learning to Fuse Latent Representations for Multimodal Data 220

11.1 Introduction 221

11.2 Background and Problem Statement 223

11.2.1 Background on learning with multimodal data 223

11.2.2 Problem statement 225

11.3 Learning to Fuse RGB-Depth Latent representations 225

11.4 Experiments 227

11.4.1 BU-3DFE facial expression RGB-D dataset 227

11.4.2 Washington object RGB-D dataset 228

11.5 Conclusion 230

12 Conclusions 231

12.1 Summary 231

12.2 Future directions 232

12.2.1 Neural architecture search for deep neural networks 233

12.2.2 Analysing non-linear deep models with skip connections 233

12.2.3 Optimal structure for model compression 233

12.2.4 Fusing Several Multimodal data for Improved Learning 234

A1 Proof of Proposition 6.1 235

A2 Expanded derivation of (6.15) 235

A3 Proof of Theorem 6.3 237

A4 Empirical Proof of Corollary 7.2 238

A5 Proof of Lemma 1 240

A6 Proof of Lemma 2 241

A7 Proof of Proposition 2 241

List of Abbreviations

CMF ..... Computational Memory Footprint

CNN ..... Convolutional Neural Network

ConcNeXt ..... Concatenated Network of Hidden Representations

DEGL ..... Debiased Elastic Group LASSO

DenseNet ..... Densely Connected Network

DNN ..... Deep Neural Network

DPN ..... Dual Path Network

EGL ..... Elastic Group LASSO

ENR ..... Elastic Net Regularization

FAST ..... Features from Accelerated Segment Test

FLOPS ..... Floating Point Operations Per Second

FS ..... Feature Standardization

GPU ..... Graphics Processing Unit

HOG ..... Histogram of Oriented Gradients

HOSVD ..... Higher Order Singular Value Decomposition

HWCC ..... Highway network with Concatenated and Compressed Features

HWGC ..... Highway network with Gate Constraints

Init ..... Initialization of model parameters

KB ..... Kilo Bytes

KD ..... Knowledge Distillation

LASSO ..... Least Absolute Shrinkage and Selection Operator

LDA ..... Linear Discriminant Analysis

LReLU ..... Leaky Rectified Linear Unit

MB ..... Mega Bytes

MLP ..... Multilayer Perceptron

MMS ..... Model Memory Size

MSE ..... Mean Squared Error

NBC ..... Naïve Bayes Classifier

PlainNet ..... Plain Network

ReLU ..... Rectified Linear Unit

ResNet ..... Residual Network

RoR ..... Residual network of Residual network

SIFT ..... Scale-Invariant Feature Transform

SL ..... Subspace Learning

S-ResNet ..... Stochastic Residual Network

SURF ..... Speeded Up Robust Features

SVM ..... Support Vector Machine

SVD ..... Singular Value Decomposition

List of Notations

$\boldsymbol{A}$ ..... matrix $\boldsymbol{A}$

$\boldsymbol{a}$ ..... vector $\boldsymbol{a}$

$\boldsymbol{A}^T$ ..... transpose of matrix $\boldsymbol{A}$

$\boldsymbol{A}^{-1}$ ..... inverse of matrix $\boldsymbol{A}$

$\boldsymbol{A}^{\dagger}$ ..... pseudoinverse of matrix $\boldsymbol{A}$

$\Delta\boldsymbol{A}$ ..... change in matrix $\boldsymbol{A}$

$\frac{\delta f}{\delta z}$ ..... partial derivative of f with respect to z

$\mathbb{E}(a)$ ..... expectation of a

H ..... cross-entropy loss function

$\mathbb{H}(\boldsymbol{A})$ ..... entropy of $\boldsymbol{A}$

$h(x)_j^l$ ..... output of unit j in layer l

$\boldsymbol{H}(\boldsymbol{X})^l$ ..... feature representation in layer l

$\kappa(\boldsymbol{A})$ ..... condition number of matrix $\boldsymbol{A}$

diag($\boldsymbol{a}$) ..... diagonal matrix with diagonal entries $\boldsymbol{a}$

$\odot$ ..... concatenation operation

$\mathbf{I}$	identity matrix
log (a)	logarithm of a
$I(\boldsymbol{A};\boldsymbol{B})$	mutual information between $\boldsymbol{A}$ and $\boldsymbol{B}$
$A \subseteq B$	set A is a subset of set B
$P(A)$	probability of A
t_{th}	pruning threshold for model parameters
$\Upsilon_{k=1}^{s}\boldsymbol{A}^k$	successive vertical concatenations of the matrices $\boldsymbol{A}^k$
$U(a,b)$	uniform distribution in the range a to b
$\mathcal{N}(\mu,\,\beta)$	normal distribution with mean μ and standard deviation β
w_{ji}^{l}	weight connecting unit j in layer l to unit i in layer $l-1$
$\widetilde{\boldsymbol{W}}$	reparameterized weight matrix $\boldsymbol{W}$
$\boldsymbol{W}^l$	weight matrix in layer l
$(\boldsymbol{W})^m$	weight matrix $\boldsymbol{W}$ raised to the power of m

List of Figures

1.1 Top-5 error rate with depth on the ILSVRC [158] 3

2.1 Very deep neural network. Left: model with no skip connection. Right: model with skip connection that adds of concatenates the outputs of the previous block, $\boldsymbol{H}(\boldsymbol{X})^{l-1}$, to the output of the current block, $\boldsymbol{H}(\boldsymbol{X})^{l}$. 16

2.2 Model pruning in DNN. The units are shown as circles, and weights as connections. The discontinuous connections show the weights to be removed, while the continuous connnections show the unaffected weights. 19

2.3 Knowledge distillation in DNN. The final outputs the Student are constrained to follow the final outputs of Teacher. 20

3.1 Performance of deep architectures with depth. Left: Training error on USPS dataset. Right: Training error on COIL-20 dataset. It is seen that optimization becomes more difficult with depth 27

3.2 (a) Proposed model with shortcut connections from the input to hidden layers (b) Closer view of the proposed residual learning with a hypothetical stack of two hidden layers 29

3.3 Performance of deep architectures with depth on the USPS dataset. Left: Test error rate with depth. Right: Test error rate for different dropout probabilities of input shortcut connections 34

3.4 Performance of deep architectures with depth on the MNIST dataset. Left: Test error rate with depth. Right: Test error rate for different dropout probabilities of input shortcut connections 35

4.1 Highway network block [207] 47
4.2 Original highway block learning attributes. Top row: 50-layer model trained on MNIST dataset. Bottom row: 50-layer model trained on CIFAR-100 dataset. First column: transform gate biases. Second column: mean of the transform gate outputs over the whole training set. Third column: transform gate output using a random training sample [207] 49
4.3 Highway network block with gate constraints (HWGC block) [158] 52
4.4 Gate remapping 53
4.5 Proposed highway network block with gate constraints and concatenated features (HWCC block) 56
4.6 Error rate for HWCC-19. Left: CIFAR-10 dataset. Right: CIFAR-100 dataset . 62
4.7 Error rate for HWCC-19. Left: Fashion-MNIST dataset. Right: SVHN dataset 64
4.8 Gating units' activations (responses) for the first highway block in HWCC-19 trained on CIFAR-10 dataset. The first column shows the input images for the 10 different classes; other columns show units' activations for different convolution maps 70
4.9 Convergence rate for HWCC-19. Left: CIFAR-10 dataset. Right: CIFAR-100 dataset 70

5.1 Error rates for very deep PlainNets. PlainNet-BN: PlainNet trained with only batch normalization; Proposed-PlainNet: PlainNet proposed in this chapter. PlainNet-BN is hard to optimize; batch normalization is insufficient for successful training 75
5.2 Normalized mean layer activations for a 100 layer PlainNet over COIL-20 dataset 78
5.3 Normalized mean layer activations for a 100 layer PlainNet over USPS dataset 78

5.4 Maximum absolute units' activations with depth 79
5.5 Infinity-norm based condition number for model weights 79
5.6 50th layer activations in a 100 layer PlainNet for the entire COIL-20 training set 80
5.7 99th layer activations in a 100 layer PlainNet for the entire USPS training set 80
5.8 % of units' unnormalized absolute activations above threshold for the USPS dataset 81
5.9 Top: PlainNet error rates on training set for COIL-20 (left) and USPS (right) datasets. Bottom: Error rates on test set for COIL-20 (left) and USPS (right) datasets [162] 85
5.10 Error rates with proposed PlainNet using MNIST dataset 86
5.11 Test-error based ablation study of the different components for improving PlainNet training using COIL-20 dataset 87

6.1 Models with skip connections considered in this chapter. $\oplus$ denotes addition operation. Left: PlainNet block. Middle: ResNet block. Right: ResNeXt block. Full DNNs are constructed by stacking several blocks 97
6.2 Training curves for models. Left: Training loss. Right: Training accuracy . . . 119
6.3 Testing curves for models. Left: Testing loss. Right: Testing accuracy 120
6.4 Units' activations for 110 layer models trained on the CIFAR-100 dataset. Left column: PlainNet. Middle column: ResNet. Right column: ResNeXt 120
6.5 Units' weights for 110 layer models trained on the CIFAR-100 dataset. Left column: PlainNet. Middle column: ResNet. Right column: ResNeXt 121
6.6 Units' gradients for 110 layer models trained on CIFAR-100 dataset. Left column: PlainNet. Middle column: ResNet. Right column: ResNeXt 121
6.7 Model weights condition number. Left: CIFAR-10 dataset. Right: CIFAR-100 dataset 122
6.8 Model weights condition number. Left: MNIST dataset. Right: ImageNet dataset122
6.9 Model hidden layer condition number; layers with missing condition numbers have infinite values. Left: CIFAR-10 dataset. Right: CIFAR-100 dataset . . . 122

6.10 Condition number for layer weights in the models trained with the linear activation function 124
6.11 Condition number for hidden layer representations in the models trained with the linear activation function; layers with missing condition numbers have infinite values 124

7.1 Generic DNN block used for analysis, where $\odot$ denotes the vertical concatenation operation. Left: PlainNet block. Right: ConcNeXt block. 129
7.2 Hidden representations for the 110 layers MLPs using randomly chosen batch of input data from the MNIST dataset. Top row: PlainNet-110. Bottom row: ConcNeXt-110 ($s = 1$). 139
7.3 Weights distribution for the 110 layers MLPs using randomly chosen batch of input data from the MNIST dataset. Top row: PlainNet-110. Bottom row: ConcNeXt-110 ($s = 1$). 139
7.4 Units' outputs distribution for the 110 layers MLPs using randomly chosen batch of input data from the MNIST dataset. Top row: PlainNet-110. Bottom row: ConcNeXt-110 ($s = 1$). 140
7.5 Condition of the layer weights in the MLPs. Left: condition number of the weights in the different layers *plotted to log-scale*. Right: smallest singular value of the weights in the different layers. 140
7.6 Condition number of the hidden layer representations in the MLPs. The layers for which values are not shown have infinite condition numbers. Left: condition number of the hidden representations in the different layers *plotted to log-scale*. Right: condition number of the hidden representations in the different layers of the ConcNeXt ($s = 1$) and ConcNeXt ($s = 2$). 141
7.7 Condition of the layer weights in the CNNs. Left: condition number of the weights in the different layers. Right: smallest singular value of the weights in the different layers. 142

7.8 Condition number of the hidden layer representations in the 110 layers CNN models. The layers for which values are not shown have infinite condition numbers. 143
7.9 Condition of the layer weights in the 110 layers MLPs with the *linear activation function*. Left: condition number of the weights in the different layers *plotted to log-scale*. Right: smallest singular value of the weights in the different layers. 143
7.10 The distribution of the condition number of random matrices of different sizes with entries sampled from Gaussian distributions obtained using He method [68]; the experiment is repeated 1000 times. Top row: distribution of all condition numbers. Bottom row: distribution of condition numbers greater than 10^3, which improves the clarity of condition number increase with matrix size. . . . 145

8.1 Overall framework for the proposed approach in Section 8.4. Top left: DNN is trained using the proposed elastic group LASSO cost function, $J_{egl}(W)$, in Section 8.4.1 (a). Top right: the non-selected filters after training translate to the black convolution channels. Bottom right: Non-selected filters are pruned, as in Section 8.4.1 (b). Bottom left: Pruned model is retrained without group LASSO, as in Section 8.4.1 (c) 151
8.2 LeNet-5 CMF and MMS results on MNIST 159
8.3 VGG-16 CMF and MMS results on CIFAR-10 160
8.4 DEGL2 compression results for VGG-16 on CIFAR-10 160
8.5 ResNet-56 CMF and MMS results on CIFAR-10 161
8.6 VGG-16 CMF and MMS results on CIFAR-100 162
8.7 DEGL3 compression results for VGG-16 on CIFAR-100 162
8.8 AlexNet CMF and MMS results on ImageNet 163
8.9 DEGL4 compression results for AlexNet on ImageNet 163
8.10 ResNet50 CMF and MMS results on ImageNet 164
8.11 Pruning threshold values and AlexNet performance loss on ImageNet dataset 166
8.12 Pruning threshold values and ResNet-50 performance loss on ImageNet dataset 166

8.13 Group LASSO penalty weight and AlexNet performance loss 168
8.14 Group LASSO penalty weight and ResNet-50 performance loss 168
8.15 Training time for compression methods on imageNet 169
8.16 First convolution layer filters in AlexNet trained with DEGL. Filters '3', '20', '40', '49', '60'and '64' are selected for pruning 169

9.1 Overall framework for the proposed subspace learning. It is assumed that the teacher model, T, is already trained to convergence and then its parameters are frozen. The Student model, S, is trained in a layer-wise fashion. In each stage, the hidden layer of the student model, H_s^l, is used to learn a subspace of the corresponding hidden layer of teacher model, H_t^l; a cost function $E(H_t^l, o_s^l)$ that minimizes the reconstruction error of the hidden repersentation of the teacher model H_t^l and the student model's output o_s^l is defined to achieve this. The output of each layer is the variable given in the layer's block. o_t^{sm} and o_s^{sm} represent the softmax outputs of the teacher model and student model, respectively. 173
9.2 Overall framework for compressing a Teacher deep model to obtain a Student deep model via subspace learning 181
9.3 Compressed VGG models convergence rate for the first few epochs 188
9.4 Convolution maps activations for the first layer of VGG-16 models trained on CIFAR-10 dataset; models have been simulated using an image from the 'horse' class. Compression enforces increased units activities in stud-m and stud-s relative to the teacher model 188
9.5 Average units' activations over CIFAR-10 dataset; the third convolution layer of trained VGG-16 models are shown. Most units in the teacher models operate with small activations; most units' activations in stud-m and stud-s have been redistributed to have larger values to compensate for compression . . . 189

9.6 Convolution maps activations for the first layer of VGG-16 models trained on CIFAR-100 dataset; models have been simulated using an image from the 'elephant' class. It is observed that compression imposes increased units activities in stud-m and stud-s relative to the teacher model 191
9.7 Average units' activations over CIFAR-100 dataset; the first convolution layer of trained VGG-16 models are shown. Most units in the teacher models operate with small activations; most units' activations in stud-m and stud-s have been redistributed to have larger values to compensate for compression . . . 192
9.8 Model inference time for test set of dataset 193
9.9 Convolution maps activations for the first layer of models trained on MNIST dataset; models have been simulated using an image from the 'five' class. Compression enforces increased units activities in stud-m and stud-s relative to the teacher model 194
9.10 Average units' activations over MNIST dataset; the first convolution layer of trained models are shown. It is observed that most units in the teacher models operate with small activations, while most units' activations in stud-m and stud-s have been redistributed to have larger values to compensate for compression 195
9.11 Student model test classification accuracy with number of training samples for the SVHN dataset 197
9.12 Sparsity ratios for different values of $L1$-norm penalty term, λ 198
9.13 CIFAR-10 test accuracy for different pruning threshold, T_h, and L1-norm penalty, λ 199
9.14 CIFAR-100 test accuracy for different pruning threshold, T_h, and $L1$-norm penalty, λ 200
9.15 MNIST test accuracy for different pruning threshold, T_h, and $L1$-norm penalty, λ200

10.1 BU-3DFE dataset facial expressions [4] 210
10.2 BU-3DFE depth map pre-processing 210

10.3 Learning pipeline for depth map modality: PL-depth map 212
10.4 Learning pipeline for RGB modality: PL-RGB 213
10.5 Depth map and RGB latent representation fusion pipeline: PL-fusion 214

11.1 Proposed data dependent fusion of latent representations, F^A and F^B extracted using DNN-A and DNN-B respectively. A modality gate is used for filtering F^A and F^B to realize improved fusion 226

A1 The distribution of singular values of random matrices with entries sampled from uniform distribution obtained using the He method [68]; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.4711. Middle: 0.4728. Right: 0.4734. 240
A2 The distribution of singular values of random matrices with entries sampled from Gaussian distribution obtained using the He method [68]; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.4450. Middle: 0.4887. Right: 0.4914. 240
A3 The distribution of singular values of random matrices with entries sampled from uniform distribution in the range -0.05 to 0.05; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.0531. Middle: 0.2336. Right: 0.4537. 240
A4 The distribution of singular values of random matrices with entries sampled from Gaussian distribution with $\mu = 0$ and $\beta = 0.01$; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.0173. Middle: 0.0806. Right: 0.1806. 241
A5 The distribution of singular values of random matrices with entries sampled from Gaussian distribution with $\mu = 0$ and $\beta = 0.05$; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.0856. Middle: 0.4027. Right: 0.4863. 241

List of Tables

3.1 Error rate (%) on the USPS dataset 35
3.2 Error rate (%) on the MNIST dataset 36

4.1 Proposed 19-layer HWCC architecture 58
4.2 Test error rate (%) on CIFAR-10 and CIFAR-100 dataset 61
4.3 Test error rate (%) on Fashion-MNIST dataset 63
4.4 Test error rate (%) on SVHN dataset 64
4.5 Test error rate (%) on imagenet-2012 (ILSVRC) dataset 66
4.6 Training time per epoch on CIFAR-10 dataset 66
4.7 Inference time per frame for CIFAR-10 dataset 66
4.8 Inference time per frame for imagenet dataset 67
4.9 GPU memory resource for CIFAR-10 dataset 69
4.10 GPU memory resource for Imagenet dataset 69

5.1 Test error rate (%) on the CIFAR-10, CIFAR-100 and SVHN datasets 88
5.2 Error rate (%) on ImageNet dataset using a single model with single crop . . 88

6.1 Results after training on the MNIST dataset 118
6.2 Results after training on the CIFAR-10 dataset 119
6.3 Results after training on CIFAR-100 dataset 119
6.4 Results (i.e. top-1 accuracy) after training on the ImageNet dataset 119

7.1 110 layers MLP details and results on the MNIST dataset. 139

7.2 Model details and results of the CNN models with 110 layers trained on CIFAR-10 dataset. 141

8.1 LeNet-5 compression results on MNIST dataset 159
8.2 VGG-16 compression results on CIFAR-10 dataset 159
8.3 ResNet-56 compression results on CIFAR-10 dataset 160
8.4 VGG-16 compression results on CIFAR-100 dataset 162
8.5 AlexNet compression results on ImageNet dataset 163
8.6 ResNet-50 compression results on ImageNet dataset 164
8.7 ImageNet pre-trained ResNet-50 compression results 165

9.1 Compression results on CIFAR-10 dataset 187
9.2 Compression results for proposed stud-m model trained with LASSO and then pruned using different T_h (pruning threshold value) values on CIFAR-10 dataset; P% is in reference to the VGG teacher model used for Table 9.1 . . . 190
9.3 Compression results on CIFAR-100 dataset 191
9.4 Compression results for proposed stud-m model trained with LASSO and then pruned using different T_h (pruning threshold value) values on CIFAR-100 dataset; P% is in reference to the VGG teacher model used for Table 9.3 193
9.5 Compression results on MNIST dataset 194
9.6 Compression results for proposed stud-m model trained with LASSO and then pruned using different T_h (pruning threshold value) values on MNIST dataset; P% is in reference to the teacher model used for Table 9.5 195
9.7 Compression results on SVHN dataset 196

10.1 Experimental results for the different pipelines 216
10.2 Our best experimental result via depth map and RGB fusion along with state-of-the-art results for comparison 218

11.1 BU-3DFE RGB-D dataset results with 10-fold CV 228

11.2 Results comparison on BU-3DFE dataset 228
11.3 Washington RGB-D dataset results with 10-fold CV 229
11.4 Washington RGB-D dataset results with 5-fold CV 229

Abstract

Learning-based approaches have recently become popular for various computer vision tasks such as facial expression recognition, action recognition, banknote identification, image captioning, medical image segmentation, etc. The learning-based approach allows the constructed model to learn features, which result in high performance. Recently, the backbone of most learning-based approaches are deep neural networks (DNNs). Importantly, it is believed that increasing the depth of DNNs invariably leads to improved generalization performance. Thus, many state-of-the-art DNNs have over 30 layers of feature representations. In fact, it is not uncommon to find DNNs with over 100 layers in the literature. However, training very DNNs that have over 15 layers is not trivial. On one hand, such very DNNs generally suffer optimization problems. On the other hand, very DNNs are often overparameterized such that they overfit the training data, and hence incur generalization loss. Moreover, overparameterized DNNs are impractical for applications that require low latency, small Graphic Processing Unit (GPU) memory for operation and small memory for storage. Interestingly, skip connections of various forms have been shown to alleviate the difficulty of optimizing very DNNs.

In this thesis, we propose to improve the optimization and generalization of very DNNs with and without skip connections by reformulating their training schemes. Specifically, the different modifications proposed allow the DNNs to achieve state-of-the-art results on several benchmarking datasets.

The second part of the thesis presents the theoretical analyses of DNNs without and with skip connections based on several concepts from linear algebra and random matrix theory.

The theoretical results obtained provide new insights into why DNNs with skip connections are easy to optimize, and generalize better than DNNs without skip connections. Ultimately, the theoretical results are shown to agree with practical DNNs via extensive experiments.

The third part of the thesis addresses the problem of compressing large DNNs into smaller models. Following the identified drawbacks of the conventional group LASSO for compressing large DNNs, the debiased elastic group least absolute shrinkage and selection operator (DEGL) is employed. Furthermore, the layer-wise subspace learning (SL) of latent representations in large DNNs is proposed. The objective of SL is learning a compressed latent space for large DNNs. In addition, it is observed that SL improves the performance of LASSO, which is popularly known not to work well for compressing large DNNs. Extensive experiments are reported to validate the effectiveness of the different model compression approaches proposed in this thesis.

Finally, the thesis addresses the problem of multimodal learning using DNNs, where data from different modalities are combined into useful representations for improved learning results. Different interesting multimodal learning frameworks are applied to the problems of facial expression and object recogntion. We show that under the right scenarios, the complementary information from multimodal data leads to better model performance.

Chapter 1

Introduction

In this recent era of explosion in internet speed, high resolution measurements, data collection and storage capacity, relying on humans to analyze data has become a very daunting task. Moreso, there has been a consistent growth in the complexity of data for analysis. On one hand, there are problems for which formulating them in precise mathematical terms is painstacking. On the other hand, for more complicated problems, our limited knowledge can make formalizing the relationships among varibales outrightly impossible.

The alternative approach is employing intelligent machines for understanding and analysing these vast and complicated data. Generally, intelligent machines rely on different algorithms that learn patterns in data for achieving various tasks; the explicit formalization of different problems is eliminated with this approach. Early intelligent machines invariably relied on extensive handcrafted feature extraction algorithms, and a simple classification algorithm. The extensive and careful feature extraction stage ensures that features, which are crucial for discrimination in the classification stage are extracted that results. For uncomplicated tasks, common image processing operations such as filtering, edge detection, erosion, dialation and skeletionization, are applied to extract features, which are fed into simple classifiers such as Multi-Layer Perceptron (MLP) with one to two layers, Linear Support Vector Machine (SVM), Linear Discriminant Analysis (LDA) classifer and Naïve Bayes Classifier (NBC). For complicated tasks, advanced feature extraction are techniques such as the Scale-Invariant Feature Transform (SIFT) [124, 123], Speeded-Up Robust Features

(SURF) [8], Features from Accelerated Segment Test (FAST) [178] and Histogram of Orientated Gradients (HOG) [38] are employed. The challenge with this approach is that the. image processing techniques or feature descriptors typically vary from task to task. Furthermore, determining the suitable image processing techniques or feature descriptors to apply for different tasks is not trivial. Applying an inappropriate feature descriptor results in features that are not indeed descriminative, which subsequently lead to poor results for classification.

However, in the last decade, there has been a shift from the aformentioned approach to end-to-end learning algorithms, where important features (i.e. optimal features) in the training data that allow good discrimination are learnt during model optimization; this eliminates applying advanced image processing operations or handcrafting features before classification. In this direction, DNNs are arguably the most popular and successful learning models. The capacity of DNNs to approximate complicated mapping functions have earned them a high peformance reputation for machine learning and computer vision tasks. Very interesting results have been reported on tasks ranging from gesture recognition [153], image segmentation [61], facial expression recognition [156], lip reading [132] and image captioning [36]. It is observed that many successful DNNs [67, 254, 236] are typically large with several millions of parameters so that they are practically unuseful where there are tight computational constraints.

Following the discussed prospects of DNNs for computer vision tasks, this thesis addresses the construction and training of DNN models that perform better than previous state-of-the-art models. Particularly, due to performance, we are interested in DNNs with several layers that are referred to as very deep neural networks (i.e. very DNNs). Furthermore, we seek to provide theoretical explainations for the failure or success of very DNNs with different architectures. In addition, the problem of compressing large DNNs into compact models that are efficient is tackled. Finally, different DNN architectures and learning algorithms for improving multimodal learning results on the problem of facial expression and object recognition are investigated.

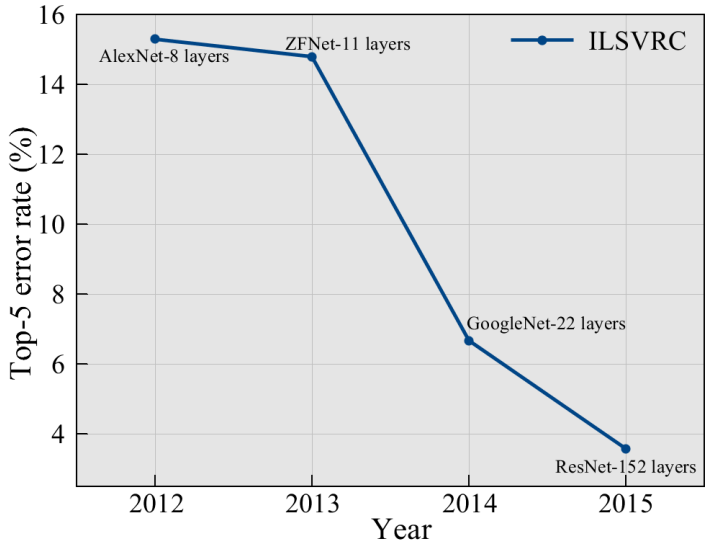


Figure 1.1: Top-5 error rate with depth on the ILSVRC [158]

1.1 Motivation and Scope

State-of-the-art results on several tasks in computer vision are achieved using DNNs for learning difficult mapping functions. Over a decade ago, most DNNs employed two to four hidden layers. However, there has been a gradual and consistent desire for the training of much deeper models. This direction was first motivated by improved empirical results obtained by simply extending the depth of previous state-of-the-art models. For example, network-in-network [119], maxout networks [58] and stacked convolutional auto-encoder [133] all have over four layers. Perhaps the biggest breakthrough that reinforced the importance of depth was the proposed AlexNet [100] with 8 layers, which shattered the previous best results on the popular and benchmarking Imagenet-2012 dataset, which were obtained using classical computer vision techniques such as handcrafted feature descriptors [182]. In recent times, it is not uncommon to find in the literature DNNs with over fifty layers [207, 67, 236, 216]. For the aforementioned reasons, the training of DNNs with more layers to realize improved generalization performance has become an interesting research direction. Furthermore, understanding the operation of very DNNs would facilitate the construction and training of new models with better performance. Figure 1.1 summarizes the

impact of model depth on reported results on the ILSVRC dataset. From the evolution of best results reported on ILSVRC dataset, the impact of depth for improving model generalization on hard datasets becomes apparent. It is noted that similar observations have been reported on other hard benchmarking datasets such as CIFAR10, CIFAR-100 and SVHN. For example, most of the models [67, 77, 101] that have approached an error rate of 5% and 23% on CIFAR-10 and CIFAR-100 datasets, respectively, employed many layers of feature representations.

Noting the observation that most state-of-the-art DNNs are cumbersome with several millions of parameters [100, 194, 67, 254], the compression of large DNNs is being actively studied. The main problems associated with large models include large storage memory, large graphics processing unit (GPU) memory for training, high inference time and high electrical power consumption. The aforementioned problems can limit their practical usefulness in portable electronic devices with tight computational budget. Subsequently, deep neural network compression aims to obtain a compact model from the original model that is considered cumbersome.

Most learning problems involve using data from a single modality (i.e. source). However, there are many instances, where the data from a single modality may not be consistently rich in crucial features for the problem being addressed. As such, the latent representations extracted by the DNN that is trained on such data may not be robust. For tackling this problem, different works [257, 90, 113] have relied on multimodal learning, where data from different modalities within the problem domain are employed for learning. The core assumption in multimodal learning is that the data from different modalities provide complementary information, which can improve the latent representation of the problem being addressed. In this thesis, the learning system is taken as a DNN that accepts as inputs data from different modalities, and learns in the hidden layers robust representations that capture the important factors of variations in the data. Subsequently, the latent representations from the different data modalities are combined via concatenation, after which higher latent representations are learned towards the task being solved.

1.2 Challenges in Training Very Deep Neural Network

1.2.1 Alleviating training problems of Very Deep Neural Networks

Very DNNs employ several layers of latent representation for learning the heirarchical compositions of the features in the training data. However, training DNNs with many layers is not a trivial task due to optimization problems; many works [67, 207, 162] have reported training problems when the number of layers exceed 15 [207, 162]. It is well known that the optimization problem becomes aggravated with further depth increase [67, 207, 162]. Specifically, it is observed that the training problem can be so severe that not only does the model fail to generalize well, but even the training data cannot be successfully learned. Going forward, the addition of skip connections has been proposed to alleviate the training problem [67, 207, 216, 211, 236]. However, the drawbacks of the most successful very DNNs with skip connections [67, 254] include (i) excessive (GPU) memory requirement (ii) long training and (iii) inference time.

1.2.2 Explaining the Role of Skip Connections in Very Deep Neural Networks

Furthermore, the criticism of DNNs being black boxes has increased in recent times given that the literature is lacking of a concrete explanation of why the proposed solutions that rely on skip connections of various forms alleviate the optimization problem, and on top of this results in improved generalization performance. We observe that explaining the classical-vanilla DNNs without skip connections, which has been around for over a decade is already challenging. As such, it is not surprising that understanding the operation of recent DNNs with skip connections, which are complicated acrhitectures is currently an active research area.

1.3 Challenges in Deep Neural Network Compression

1.3.1 Joint Knowledge Transfer for Model Compression

For knowledge transfer based methods, compressing a large DNN into a smaller one requires the transfer of important information in the large model to the small one. In this direction, the large and small models are commonly referred to as the teacher and student models, respectively. Although interesting results have been reported in earlier works such as in [73, 175], the problem with the aforementioned works is that the joint transfer of knowledge from the teacher model to the student model is hard. The several layers of compositional functions seen in DNNs means that the space of possible functions that the student model can explore for mimicing the exact knowledge in the teacher model is large. That is, the degree of freedom allowed by the several hidden layers invaribaly results in the student model failing to precisely capture the knowledge in the teacher model [148].

1.3.2 Sparsity Enforcing Penalty for Model Compression

The pruning method basically assumes that the large model is overparameterized so that some of the model weights are redundant or have small values, and can be removed without hurting model performance [1]. Generally, most works in this direction enforce the weights of large models to have small values by using a modified cost function for training. In fact, the ideal scenario for pruning is that many weights in the model have zero values; the removal of such weights has no effect on the model performance. The modified cost function typically incorporates regularization terms such as the $L1$-norm [218] and $L2$-norm [275] on model weights; $L1$-norm is the popular least absolute shrinkage and selection operator (LASSO). It is known that the $L2$-norm penalty term does not enforce model weights to have zero values, as seen in the LASSO penalty term [218]. As such, the LASSO penalty is more desirable for model compression than the $L2$-norm penalty. Unfortunately, enforcing sparsity in DNNs that have several layers with the LASSO penalty term is challenging due to the following: (i) feature saturation, where the number of selected features cannot exceed the number of training data points [275, 264] (ii) random selection of a feature from a group of

correlation features [275, 264] (iii) entangled feature selection and shrinkage [218, 275] that can obfuscate model interpretability, and cause model underfitting [264].

1.4 Challenges in Multimodal Learning for Deep Neural Network

1.4.1 DNN architecture for Multimodal Learning

Multimodal data can improve the performance of DNNs on various learning tasks. However, determining the suitable DNN architectures that allow the realization of the benefit of multimodal learning can be tricky. Common challenges include choosing the base DNN models to employ, and the stage to combine the latent representations from the different modalities.

1.4.2 Imbalanced Information Quality in Multimodal Data

When the core assumption that the data from different modalities are almost equally rich in information and noise-free is violated, multimodal learning can lead to catastrophic learning. The problem is that the decent latent representation from the good data modality is corrupted by the noisy latent representation from the other data modality that is problematic.

1.5 Objectives and Contributions

In this thesis, we address the challenges of training very deep neural networks, understanding their operations, compressing them into compact sizes mentioned, and their application for multimodal learning in Section 1.2, Section 1.3 and Section 1.4, respectively. The different proposals that tackle the identified problems are discussed in the following sections.

1.5.1 Residual Network with Improved Generalization Performance

In order to improve generalization, stochastic shortcut connections between the input data and the hidden layers were added to residual networks (ResNet). These shortcut connections are combined with the conventional shortcut connections from preceding layers to the

higher ones as seen in the conventional ResNet. This new method, referred to as S-ResNet, has been evaluated on the United States Postal Service (USPS) and Modified National Institute of Standards and Technology (MNIST) benchmark datasets showing an improved regularization on very deep neural networks as compared to the conventional ResNet approach. Furthermore, an extension of the S-ResNet approach has been proposed. In very deep networks, rectified linear units (ReLUs) can die out during training and thus impact the representation capacity of the network. As a solution, Maxout units have been proposed to replace ReLUs, in addition to using elastic regularization to alleviate the increased model parameters. This improved S-ResNet has been validated experimentally, showing better results as compared to state-of-the-art and the S-ResNet model.
This work has been published in [162] and [160].

1.5.2 Reformulated Highway Neural Network for Improved Training

State-of-the-art deep neural network (DNNs) typically use various forms of skip connections for successful training. Generally, skip connections of identity mappings are used. e.g. residual networks (ResNets). However, another interesting approach is using gating mechanisms for guiding the information flow in the different hidden layers for successful training. e.g. highway networks. In our work, we identify the drawbacks of the gating mechanism in the HighwayNet. Subsequently, we propose a modified highway network with gate constraints to mitigate the identified problems of the HighwayNet. Experimental results show that the proposed model outperforms or matches state-of-the-art models. Importantly, we show that the proposed model requires less training time, inference time and GPU memory for operation, as compared to the state-of-the-art models that yield similar performance.
This work has been published in [158] and an extended journal version accepted in [159].

1.5.3 Training Very Deep Neural Networks without Skip Connections

Training DNNs with several hidden layers is difficult due to optimization problem. Generally, DNNs with over 15 hidden layers require skip connections for successful training. However,

explanations for the operation of DNNs that employ skip connections is lacking in the literature. Therefore, we investigate in our work, the origin of the optimization problems observed in DNNs with several hidden layers that use no skip connections. Following the investigation results, an approach for the successful training of DNNs with several hidden layers without using skip connections is proposed. The proposed approach relies on the combination of different components that directly address the training problems identified. Using our proposed approach, we demonstrate the successful training of DNNs having up to 100 layers without employing skip connections. To the best of our knowledge, these models are the deepest DNNs without skip connections that have been successfully trained.
This work has been published in [157] and [161].

1.5.4 Understanding Very Deep Neural Networks with Skip Connections

DNN models with employ skip connections that the sum hidden representations of different layers achieve remarkable results on benchmarking datasets, however, a concrete explanation for their successful operation is still lacking. Furthermore DNN architectures with skip connections and concatenated hidden representations perform extremely well on various learning assignments. However, their unique characteristics that make them very successful have not been theoretically studied in the literature. Specifically, the role of skip connections for training DNNs is investigated using various aspects of linear algebra and random matrix theory. We study theoretically various forms of the aforementioned DNN architectures. Our results reveals why such DNN models with several hidden layers are trainable, and generalize well on several tasks.
This work has been published in [149], and under review in [151] and [150].

1.5.5 Deep Neural Network Compression

For model compression, latent subspace learning (SL) and least absolute shrinkage and selection operator (LASSO) is proposed to improve the transfer of knowledge from already trained large models to small models. The goal was to show that better model compression

could be obtained in comparison to current approaches. Furthermore, for the convention LASSO, we identify the problems of selection saturation, random selection of correlated features and entangled feature selection and shrinkage. Subsequently, the debiased elastic group LASSO (DEGL) is proposed to improve the structured compression of large models. This work has been published in [155] and [148].

1.5.6 Multimodal Learning with Deep Neural Network

The application of DNN to multimodal learning is considered where the goal is to allow constructed models learn latent representations of different data modalities such that robust and concise representations are relied on for performing the task at hand. Specifically, a gating mechanism is proposed to address the problem. Interesting results are obtained in the work using RGB and depth data as multimodal data for facial expression recognition and objects classification.

1.6 Publications

JOURNALS

1. **Oyedotun, O.K.**, Shabayek, A.E.R, Aouada, D. & Ottersten, B. (2020). "Deep Network Compression with Teacher Latent Subspace Learning and LASSO", Applied Intelligence.

2. **Oyedotun, O.K.**, Shabayek, A.E.R, Aouada, D. & Ottersten, B. (2020). "Improved Highway Block for Training Very Deep Neural Network", IEEE Access.

3. **Oyedotun, O.K.**, Al Ismaeil, K. & Aouada, D. (2020). "Why is Everyone Training Very Deep Neural Network with Skip Connections?", IEEE Transactions on Neural Networks and Learning Systems. (Under review).

4. **Oyedotun, O.K.**, Al Ismaeil, K. & Aouada, D. (2020). "Training Very Deep Neural Networks: Rethinking the Role of Skip Connections", Neurocomputing. (Under review).

CONFERENCES

1. **Oyedotun, O.K.** & Aouada, D. (2020). “Why do Deep Neural Networks with Skip Connections and Concatenated Hidden Representations Work?”, International Conference on Neural Information Processing, Bangkok, Thailand.

2. **Oyedotun, O.K.**, Shabayek, A.E.R, Aouada, D. & Ottersten, B. (2020). “Revisiting the Training of Very Deep Neural Networks without Skip Connections”, IEEE International Conference Pattern Recognition. (Accepted with revision).

3. **Oyedotun, O.K.**, Shabayek, A.E.R, Aouada, D. & Ottersten, B. (2020). “Going Deeper with Neural Networks without Skip Connections”, IEEE International Conference on Image Processing (ICIP), Abu Dhabi, UAE.

4. **Oyedotun, O.K.**, Aouada, D. & Ottersten, B. (2020). “Structured compression of deep neural networks with debiased elastic group lasso”, IEEE Winter Conference on Applications of Computer Vision (WACV), CO, USA. pp. 2277-2286.

5. **Oyedotun, O.K.**, Aouada, D. & Ottersten, B. (2019). “Learning to Fuse Latent Representations for Multimodal Data”, In 44th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK. pp. 3122-3126.

6. **Oyedotun, O.K.**, Shabayek, A.E.R, Aouada, D. & Ottersten, B. (2018). “Highway Network Block with Gates Constraints for Training Very Deep Networks”, In IEEE International Conference on Computer Vision and Pattern Recognition (CVPR) Workshop, Salt Lake City, Utah, US. pp. 1658-1667.

7. **Oyedotun, O.K.**, Shabayek, A.E.R, Aouada, D. & Ottersten, B. (2018). “Improving the Capacity of Very Deep Networks with Maxout units”, In 43rd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, Canada. pp. 2971-2975.

8. **Oyedotun, O.K.**, Shabayek, A. E. R., Aouada, D., & Ottersten, B. (2017). “Training Very Deep Networks via Residual Learning with Stochastic Input Shortcut Connec-

tions", In 25th International Conference on Neural Information Processing (ICONIP), Guangzhou, China, 10635, pp. 23-33.

9. **Oyedotun, O.K.**, Demisse, G., Shabayek, A.E.R, Aouada D., & Ottersten B. (2017). "Facial Expression Recognition via Joint Deep Learning of RGB-Depth Map Latent Representations", 2017 IEEE International Conference on Computer Vision (ICCV) Workshop, Venice, Italy, pp. 3161-3168.

PUBLICATIONS NOT INCLUDED IN THE THESIS

1. **Oyedotun, O.K.**, Saint A., Papadopoulos, K., & Aouada D. (2021). "Understanding Generalization Gap of Deep Neural Networks Trained with Large Batch Sizes", IEEE Winter Conference on Applications of Computer Vision (WACV). (To be submitted).

2. Papadopoulos, K., Ghorbel, E., **Oyedotun, O.K.**, Aouada, D., & Ottersten, B. (2020). "DeepVI: A Novel Framework for Learning Deep View-invariant Human Action Representations using a Single RGB Camera". IEEE International Conference on Automatic Face and Gesture Recognition, Buenos Aires. pp. 18-22.

1.7 Thesis Outline

The organisation of this thesis is enumerated as follows.

- **Chapter 2:** This chapter discusses the background works for very deep neural networks for improved generalization performance, and the compression of large deep neural networks for compact models with smaller inference time.

- **Chapter 3:** In this chapter, the different approaches such as stochastic input skip connections, and the combination of maxout units, feature standardization and a linear support vector machine (SVM) for improving the generalization performance of residual networks are presented.

- **Chapter 4:** In this chapter, the highway network blocks, as originally defined, is shown to lose in effectiveness as the network depth increases. Subsequently, a new formulation using gate constraints is proposed to ensure a faster optimization reaching convergence with less epochs, and at the same time improving generalization while preserving the network depth.
- **Chapter 5:** This chapter investigates the problem of training very deep neural networks with skip connections. Following the identified training problems, the proposed approach that results into successful model training is presented.
- **Chapter 6:** In this chapter, we study the role of skip connections that sum the hidden representations of different layers so that the optimization of very deep neural networks is successful with improved generalization performance.
- **Chapter 7:** In this chapter, we study how the introduction of skip connections that concatenate the hidden representations of different layers allow the successful training of very deep neural networks.
- **Chapter 8:** This chapter discusses the proposed debiased elastic group LASSO (DEGL) that alleviates the problems of convention group LASSO for structured compression.
- **Chapter 9:** The compression of DNNs using teacher latent subspace learning and LASSO for further compression is presented in this chapter. On top of this, it is shown that subspace learning improves compression results of LASSO.
- **Chapter 10:** The application of multimodal data and DNN for the recognition of facial expressions is proposed. Multimodal learning is shown to outperform learning that is based on a single data modality.
- **Chapter 11:** This chapter tackles the problem of imbalanced information richness in multimodal data for learning by proposing a gating mechanism that performs the dynamic selection of decent latent representations from the different multimodal data.

- **Chapter 12:** The thesis is concluded by summarizing the contributions of this research, along with various interesting future directions.

Chapter 2

Background

This chapter introduces the foundational works for the problems addressed in this thesis. Namely, the major works in the optimization and compression of very DNNs are discussed in the following sections.

2.1 Optimizing Very Deep Neural Networks

Considering the success of very DNNs discussed in Chapter 1, very DNNs have found application in various computer vision applications tasks. However, training very DNNs is not trivial. In Sections 2.1.1 and 2.1.2, the two major approaches for tackling the optimization problems of very DNNs are discussed.

2.1.1 Modifying model architecture

Given an input $\boldsymbol{H}(\boldsymbol{X})^{l-1}$ (i.e. batch data), from layer l-1 feeding into a stack of specified number of hidden layers with output $\boldsymbol{H}(\boldsymbol{X})^{l}$; in the conventional training scheme, the stack of hidden layers learns a mapping function of the form

$$\boldsymbol{H}(\boldsymbol{X})^{l} = F^{l}(\boldsymbol{H}(\boldsymbol{X})^{l-1}). \tag{2.1}$$

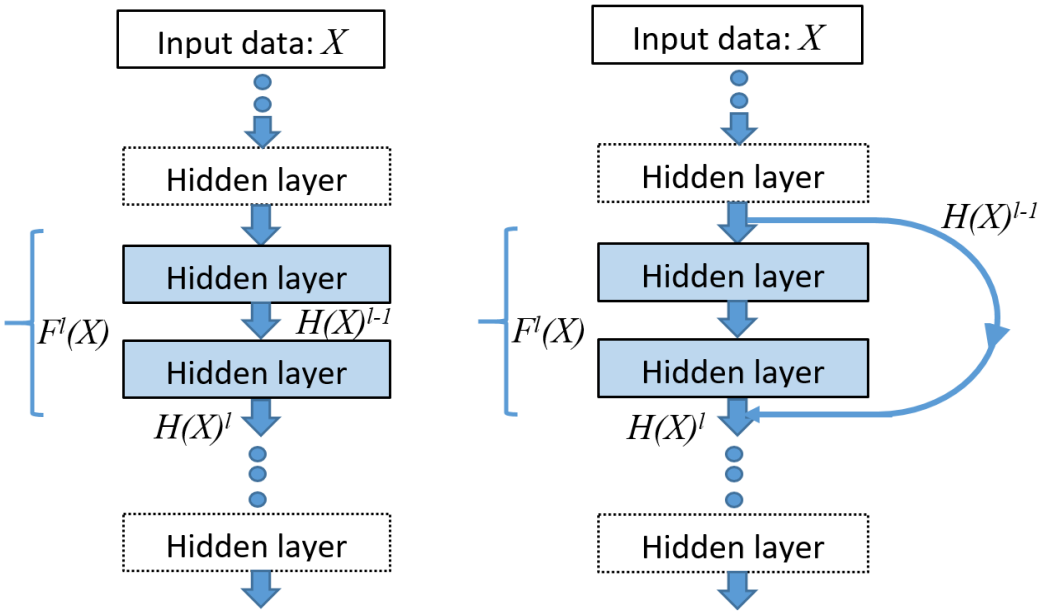


Figure 2.1: Very deep neural network. Left: model with no skip connection. Right: model with skip connection that adds of concatenates the outputs of the previous block, $\boldsymbol{H}(\boldsymbol{X})^{l-1}$, to the output of the current block, $\boldsymbol{H}(\boldsymbol{X})^{l}$.

The problem of optimizing very DNNs can be alleviated by modifying the architectures of the models as in [207, 67, 236, 216]. Generally, the output of the hidden layers are connected to the output of the earlier layers via skip (or shortcut) connections. In this fashion, the combination of the outputs of different layers is peformed by addition or concatenation. These methods are discussed in turn as follows.

Adding the Outputs of Different Hidden Layers

In this scenario, the outputs of the different layers in the models are transported via skip connections and subsequently added. For such addition operations to be valid, the dimensions of the different layer outputs being added must correspond. Generally, the model layers are divided into blocks, where each block is composed of two or three layers. For simplicity and performance, the outputs of each block are added. It is noted that this approach does not increase the dimensions of the outcome obtained from adding the outputs of different layers. We briefly discuss residual learning as a building block for the different models that are ex-

tensively studied in this thesis. In [67], residual learning was achieved by employing shortcut connections from preceding hidden layers to the higher ones. The residual learning proposed in [67] uses shortcut connections such that the stack of hidden layers learns a mapping function of the form

$$\boldsymbol{H}(\boldsymbol{X})^l = F^l(\boldsymbol{H}(\boldsymbol{X})^{l-1}) + \boldsymbol{H}(\boldsymbol{X})^{l-1}, \tag{2.2}$$

where $\boldsymbol{H}(\boldsymbol{X})^{l-1}$ is the output of the previous block carried over via the skip connection. The actual transformation function learned by the stack of hidden layers can be written as follows

$$F^l(\boldsymbol{H}(\boldsymbol{X})^{l-1}) = \boldsymbol{H}(\boldsymbol{X})^l - \boldsymbol{H}(\boldsymbol{X})^{l-1}, \tag{2.3}$$

where $1 \leq l \leq L$ and $\boldsymbol{H}(\boldsymbol{X})^0$ is the input data, $\boldsymbol{X}$; L is the depth of the network.
This training setup was found very effective in training very deep networks, achieving state-of-the-art results on some benchmarking datasets [67].

Concatenating the Outputs of Different Hidden Layers

The alternative approach to adding outputs of hidden layers is to concatenate the outputs of the different layers in the model. Again, for this conctenation operation to be valid, the dimensions of the outputs of the different layers being concatenated must agree. In this thesis, we denote the concatenation operation by $\odot$. As such, following Figure 2.1 (right), the output of the transformation in block l is given as

$$\boldsymbol{H}(\boldsymbol{X})^l = F^l(\boldsymbol{H}(\boldsymbol{X})^{l-1}) \odot \boldsymbol{H}(\boldsymbol{X})^{l-1}. \tag{2.4}$$

2.1.2 Modifying the Training Scheme for the Models

In this section, we discuss a different direction for alleviating the problem of training very DNNs. This direction is based on special initialization techniques that place model parameters in basins of attractions, which favours optimization. One popular method is the

DiracNet [253] initialization that mimics the role of skip connections at the start of training. Basically, the DiracNet initialization parameterizes the weight of the l-th layer in the model as follows

$$\widetilde{\boldsymbol{W}}^l = \boldsymbol{W}^l + \mathbf{I}, \tag{2.5}$$

where $\boldsymbol{W}^l$ is a random matrix with the entries sampled from a uniform or Gaussian distribution, and $\mathbf{I}$ is the identity matrix. In the following chapters, it is shown that this approach is not as interesting as modifying model architectures that is discussed in Section 2.1.1.

2.2 Model compression

In this section, the DNN compression methods, which are closely related to this thesis are discussed. These methods are model pruning and knowledge distillation approaches.

2.2.1 Model Pruning

In model pruning, the objective is to identify unuseful or redundant weights or units in large DNNs. The basic concept behind model pruning is shown in Figure 2.2. In the Multilayer Perceptrons (MLP), this achieved by observing and removing the weights,w^l_{ji} (i.e. as in Figure 2.2.), that are below a certain threshold, t_{th}. That is, we set $w^l_{ji} = 0$ for $w^l_{ji} < t_{th}$. Ideally, we would want $t_{th} = 0$. However, in practice, t_{th} is commonly chosen to be a sufficiently small number in the range 10^{-3} to 10^{-4}.

2.2.2 Knowledge Distillation

Knowledge distillation (KD) [73] is a popular method for compressing large DNNs. For the KD approach, there are usually two DNNs. The large DNN that referred to as the *Teacher* that we want to compress, and the small DNN referred to as the *Student* that acquires the concise knowledge from the *Teacher*. Generally, in KD, the unit's output of the *Student* model are matched with the units' output of the *Teacher* model using a modified training objective referred to as the KD loss. The overall framework for KD is shown Figure 2.3. This can

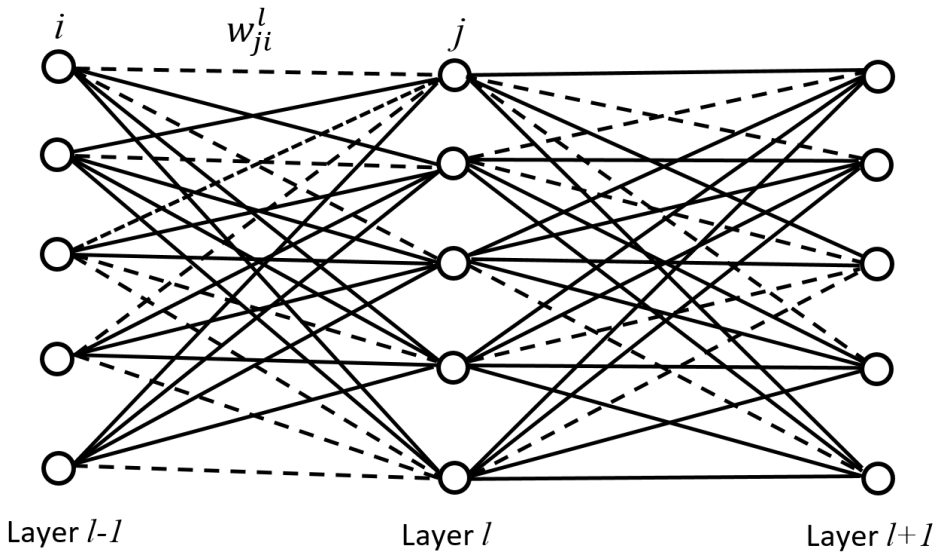


Figure 2.2: Model pruning in DNN. The units are shown as circles, and weights as connections. The discontinuous connections show the weights to be removed, while the continuous connnections show the unaffected weights.

be seen as regularizing the *Student* model using the knowledge of the *Teacher* model. As opposed to model pruning, KD does not remove weights or parameters from the *Teacher*; the *Teacher* parameters and structure are totally unaffected by the compression procedure. Rather, KD simply transfers knowledge to the *Student* by constraining the units in its final layer to learn distributions that are similar to the output units of the *Teacher*. In this fashion, the expectation is that the output units in the *Student* responds to novel input data in a way that is comparable to the output units of the *Teacher*. Let the parameters of the *Student* be denoted by $\boldsymbol{\theta}_S$, so that the KD loss, $L_{KD}(\boldsymbol{\theta}_S)$, is

$$L_{KD}(\boldsymbol{\theta}_S) = \lambda H(\boldsymbol{y}, \boldsymbol{P}_S) + H(\boldsymbol{P}_T^{\tau}, \boldsymbol{P}_S^{\tau}), \tag{2.6}$$

where $\boldsymbol{P}_S$ is the final output of the *Student*, $\boldsymbol{P}_S^{\tau}$ is the softened $\boldsymbol{P}_S$, $\boldsymbol{P}_T^{\tau}$ is the softened final output of *Teacher* $\boldsymbol{P}_T$, τ the hyperparameter for softening $\boldsymbol{P}_S$ and $\boldsymbol{P}_T$, H is the cross-entropy loss function, λ is a hyperparameter for controlling the impact of the hard target cross-entropy and y is the ground truth label for the training data. The work in [73, 175] notes that softening the final outputs of the *Teacher* and *Student* improves entropy, and thus

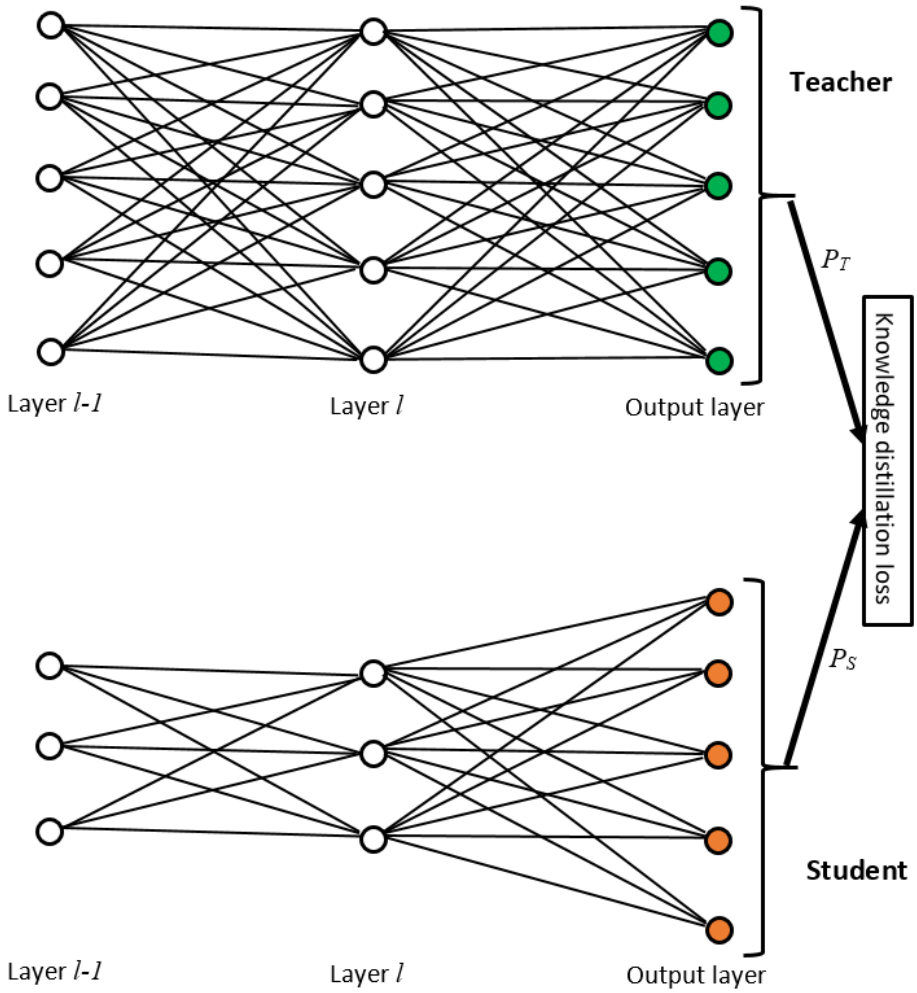


Figure 2.3: Knowledge distillation in DNN. The final outputs the Student are constrained to follow the final outputs of Teacher.

knowledge transfer that leads to better results. The final outputs of the *Teacher* and *Student* are softened using the following relations

$$\boldsymbol{P}_T^\tau = softmax(\frac{\boldsymbol{P}_T}{\tau}), \tag{2.7}$$

and

$$\boldsymbol{P}_S^\tau = softmax(\frac{\boldsymbol{P}_S}{\tau}). \tag{2.8}$$

2.3 Deep Neural Network for Multimodal Learning

DNNs are mostly trained using data from a single source (i.e. modality) [153, 121]. For instance, speech recognition systems usually take as input the processed audio recordings of the speaker. The DNN recognition system then learns good latent representations, which encode the crucial features in the input data. The latent representations are expected to capture the factors of variations in the input data. However, there are scenarios where there are access to data from other sources for the same task; these data are referred to multimodal data. Subsequently, multimodal learning [257, 90, 113] relies on the concept of learning by using multimodal data within the problem domain. For instance, the lips movements of the speaker can be recorded as videos; these video recordings can be processed, and then used in conjunction with the aforementioned audio recordings for improved learning. The objective is to use the audio and video data to provide complementary information for training the DNN-based speech recognition system. That is, during learning, where the audio recording is unclear, the video data may provide interesting concepts that still allows a decent representation to be captured. Similarly, where the video data is unclear, the audio data provides useful concepts that allow a reasonable latent representation to be learned.
Another interesting example, which is relatively popular, is multimodal data that is based on RGB and depth images. In the absence of illumination variations, RGB images provide rich information about a scene than depth maps. However, RGB data suffer considerable information quality degradation than depth maps with illumination changes [273]. Consequently, RGB and depth map images can be complementary, and thus suitable for multimodal learning.

Chapter 3

Improving the Generalization of Very Deep Neural Network with Skip Connections

Many works have posited the benefit of depth in deep networks. However, one of the problems encountered in the training of very deep networks is feature reuse. That is, features are 'diluted' as they are forward propagated through the model. Hence, later network layers receive less informative signals about the input data, consequently making training less effective. In this work, we address the problem of feature reuse by taking inspiration from an earlier work which employed residual learning for alleviating the problem of feature reuse. We propose a modification of residual learning for training very deep networks to realize improved generalization performance. For this, we allow stochastic shortcut connections of identity mappings from the input to hidden layers. We perform extensive experiments using two datasets. We observe interesting results without employing any form of data augmentation or explicit regularization technique.

3.1 Introduction

Neural networks have been extremely useful for learning complex tasks such as gesture recognition [153] and banknote recognition [152]. More recently, as against shallow networks with one layer of feature abstraction, there has been massive interest in deep networks which compose many layers of features abstractions. There are many earlier works [75, 53] which established that given a sufficiently large number of hidden units, a shallow network is a universal function approximator. Interestingly, many works addressing the benefit of depth in neural networks have also emerged. For example, using the concept of sum-product networks, Delalleau & Bengio [40] posited that deep networks can efficiently represent some family of functions with lesser number of hidden units as compared to shallow networks. In addition, Mhaskar et al. [135] provided proofs in their work that deep networks are capable of operating with lower Vapnik-Chervonenkis (VC) dimensions. Bianchini & Scarselli [15] employing some architectural constraints, derived upper and lower bounds for some shallow and deep architectures; they concluded that using the same resources (computation units), deep networks are capable of representing more complex functions than shallow networks. In practice, the success of deep networks have corroborated the position that deep networks have a better representational capability as compared to shallow networks; many state-of-the-art results on benchmarking datasets are currently held by deep networks [223, 59, 34].

In recent times, the aforementioned theoretical proofs, practical results and new works [194, 210] now suggest that employing *even deeper* networks could be quite promising for learning even *more complex or highly varying functions*. However, it has been observed that the training of models beyond some few layers results in optimization difficulty [207, 67]. In this work, for the sake of clear terms, we refer to models with 2-10 hidden layers as 'deep networks', models with more than 10 hidden layers as 'very deep networks' and use the term 'deep architecture' to refer interchangeably to a deep network or very deep network. We consider the effective training of very deep networks; that is, simultaneously overcoming optimization problems associated with model depth increase and more importantly improv-

ing generalization performance. We take inspiration from an earlier work which employed residual learning for training very deep networks [67]. However, training very deep models with millions of parameters come with the price of over-fitting. On one hand, various explicit regularization schemes such as $L1$-norm, $L2$-norm and max-norm can be employed for alleviating this problem. On the other hand, a more appealing approach is to explore some form of implicit regularization such as reducing the co-adaptation of model units on one another for feature learning (or activations) [206] and encouraging stochasticity during optimization [223].

Furthermore, many works on very deep networks [67, 207, 76] relied on rectified linear units (ReLUs) for tackling the problem of units saturation and vanishing gradients in order to improve optimization. Nevertheless, ReLUs can die out during learning, consequently blocking error gradients and learning nothing [261]. As such, dead ReLUs impact the representation capacity of very deep networks which rely on the backpropagation of error gradients through several layers. In addition, approximating model parameters at test time of very deep ReLUs based networks trained with dropout can be quite inexact due to several layers of nonlinearities [206]. Consequently, we propose to modify the learning of very deep networks such that the model size truly reflects on representation capacity and therefore improves performance.

In this work, we advance in this direction with some modifications on the form of residual learning that we propose for implicitly improving model regularization by *emphasizing stochasticity* during training. Our contribution is that we propose to modify residual learning for training very deep networks where we allow shortcut connections of identity mappings from the input to the hidden layers; such shortcut connections are stochastically removed during training. Particularly, the proposed training scheme is shown to improve the implicit regularization of very deep networks as compared to the conventional residual learning. Furthermore, we preserve the representation capacity of very deep networks due to dead ReLUs by learning the activation functions of units via maxout [58]. In addition, it is shown that the features learned are quite linearly separable such that a linear SVM can successfully replace the fully connected layers of the model. Particularly, it is observed that using

maxout units and the linear SVM, improved results in comparison to [67] are obtained. We employ our proposed approach for performing extensive experiments using the USPS and MNIST datasets; results obtained are quite promising and competitive with respect to state-of-the-art results.

The rest of this chapter is organized as follows. Section 2 discusses related works. Section 3 serves as background and introduction of residual learning. Section 4 gives the description of the proposed model. Section 5 contains experiments, results and discussion on benchmark datasets. In section 6, we conclude the work with our key findings.

3.2 Related work

The optimization difficulty observed in training very deep networks can be attributed to the fact that input features get diluted from the input layer through the many compositional hidden layers to the output layer; this is evident in that each layer in the model performs some transformation on the input received from the preceding layer. The several transformations with model depth may make features not reusable. Here, one can conjecture that the signals (data features) which reach the output layer for error computation may be significantly less informative for effective weights update (or correction).

Many works have provided interesting approaches for alleviating the problem of training deep architectures. In [68, 56], carefully guided initializations were considered for specific activation functions; these initializations were found useful for improving model optimization and the rates of convergence. In another interesting work [84], batch normalization was proposed for tackling the problem of internal covariate shift which arises from non-zero mean hidden activations. Nevertheless, the problem of training (optimizing) very deep networks commonly arises when the number of hidden layers exceeds 15 [207]. For example, the work in [207] employed *transform* gates for routing data through very deep networks; they refer to their model as a *highway network*. The concept is that the transform gates are either closed or open. When the transform gates are closed, input data are routed through the hidden layers without transformations; in fact, each hidden layer essentially copies the features

from the preceding layer. However, when the transform gates are open, the hidden layers perform the conventional features transformations using layer weights, biases and activation functions. Inasmuch as the highway network was shown to allow for the optimization of very deep networks and improving classification accuracies on benchmark datasets, it comes with a price of learning additional model parameters for the transform gates. Another work, He et al. [67] has addressed the problem of feature reuse by using residual learning for alleviating the dilution (or attenuation) of features during forward propagation through very deep networks; they refer to their model as a *ResNet*. The ResNet was also shown to alleviate optimization difficulty in training very deep networks. In [76], identity shortcut connections were used for bypassing a subset of layers to facilitate training very deep networks. In a following work [69], dropping out the shortcut connections from preceding hidden layers was experimented with; however, convergence problems and unpromising results were reported.

3.3 Problem Statement

First, we emphasize the problem of training very deep networks using the USPS dataset. Figure 3.1 (left) shows the performance of plain deep architectures with a different number of hidden layers. Particularly, it will be seen that the performance of the models significantly dips from over 10 hidden layers. We further emphasize this problem by going beyond the typical uniform initialization (i.e. Unit_init in Figure 3.1) scheme for neural network models; we employ other initialization and training techniques which have been proposed for more effective training of deep models; these techniques include Glorot [68] initialization, He [56] initialization and batch normalization [84] which are shown as Glorot_init, He_init and BN in Figure 3.1.

In addition, we investigate this problem using the COIL-20 dataset[1] which composes 1,440 samples of different objects of 20 classes. The concepts which we follow in using the COIL-20 dataset as sanity check are in two folds: (1) it is a small dataset, hence it is expected that deep architectures would easily overfit such training data (2) the dataset is of much higher di-

[1] http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html

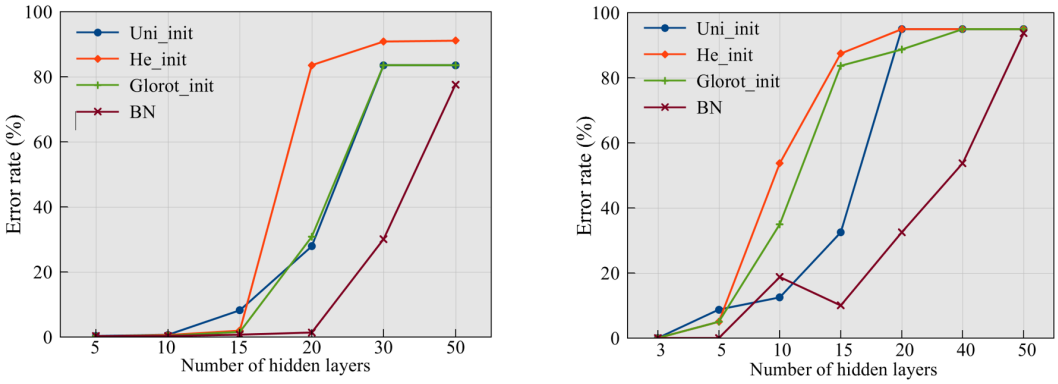


Figure 3.1: Performance of deep architectures with depth. Left: Training error on USPS dataset. Right: Training error on COIL-20 dataset. It is seen that optimization becomes more difficult with depth

mensionality. Obviously, this training scenario can be seen as an extreme one which *indeed* favours deep models with enormous parameters for overfitting the training data. This follows directly from the concept of model complexity and curse of dimensionality with high dimensional input data as against the number of training data points. However, our experimental results do not support the overfitting intuition; instead, the difficulty of model optimization is observed when the number of hidden layers is increased beyond 10; see Figure 3.1. It will be seen that for both USPS and COIL-20 datasets, training with batch normalization improved model optimization with depth increase. Nevertheless, model optimization remains a problem with depth increase. However, residual learning [67] has been employed in recent times for successfully training very deep networks. The idea is to scheme model training such that stacks of hidden layers learn residual mapping functions rather than the conventional transformation functions.

Second, very deep networks have large expressive power for representing complex and compositional target functions. However, very deep networks learned with ReLUs can suffer reduction of representation capability as a result of dead ReLUs that go into saturation and blocking error gradients for updating associated model parameters during training. As such, these dead units fail to learn and therefore impact the learning capability of the model. This problem can be more severe in very deep networks (i.e. over 20 layers) where layer

units much earlier in the model (i.e closer to the input) rely on the back propagation of error gradients over several hidden layers away. Furthermore, a deep network trained with dropout can be seen as an implicit ensemble model with shared parameters. At test time, model parameters are usually approximated by scaling them [206]. In [58], such approximation was shown to be more inexact for nonlinear networks. For very deep networks that compose ReLUs and trained with dropout, implicit model averaging is more significant and therefore the approximation of model parameters at test time could be more inexact than in shallower networks. Consequently, the aforementioned problems can work together to hurt model performance at test time.

3.4 Residual Learning with Stochastic Input Skip Connections

For improving the training of very deep models, we take inspiration from residual learning. Our proposed model incorporates some simple modifications to further improve on optimization and generalization capability as compared to the conventional ResNet. We refer to the proposed model as stochastic residual network (*S-ResNet*). The proposed training scheme is described below:

(i) There are identity skip (i.e. shortcut) connections of identity mappings from the input to hidden layers of the model; this is in addition to the shortcut connections from preceding hidden layers to the higher ones as seen in the conventional ResNets.

(ii) The identity shortcut connections from the input to the hidden layers are stochastically removed during training. Here, hidden layer units do not always have access to the untransformed input data provided via shortcut connections.

(iii) At test time, all the shortcut connections are present. The shortcut connections are not parameterized and therefore do not require rescaling at test time as in [223, 76].

The proposed scheme for training very deep models is shown in Figure 3.2 (a); conventional shortcut connections from preceding hidden layers, with shortcut connections from the input

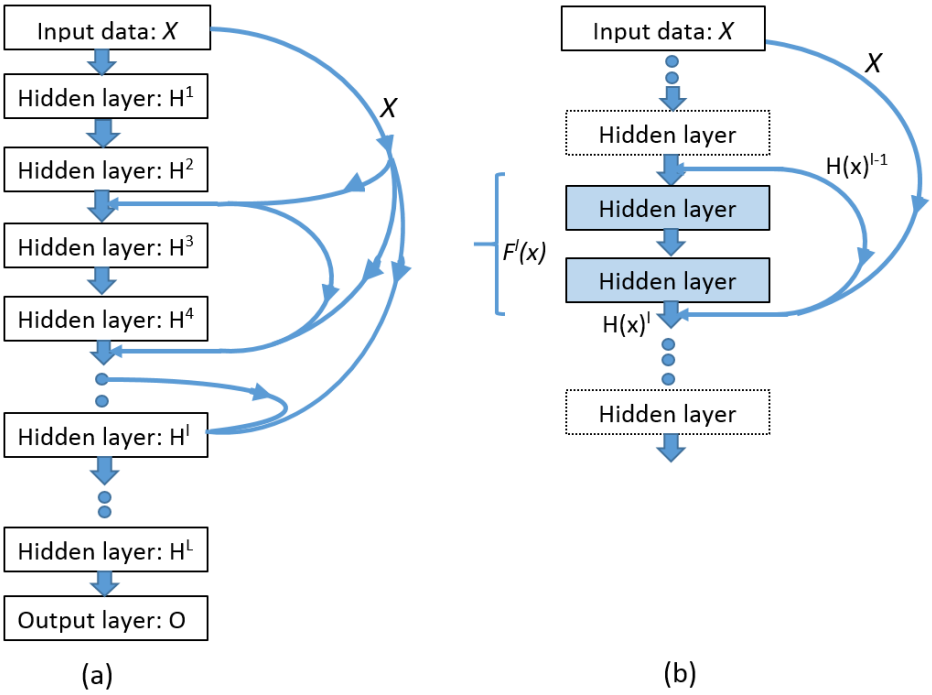


Figure 3.2: (a) Proposed model with shortcut connections from the input to hidden layers (b) Closer view of the proposed residual learning with a hypothetical stack of two hidden layers

to the different hidden layers are shown. For the modification that we propose in this work, the transformed output of a stack of hidden layers denoted, l, with shortcut connection from the preceding stack of hidden layers, $\boldsymbol{H}(\boldsymbol{X})^{l-1}$, and shortcut connection from the input $\boldsymbol{X}$ can be written as follows

$$\boldsymbol{H}(\boldsymbol{X})^{l} = F^{l}(\boldsymbol{H}(\boldsymbol{X})^{l-1}) + \boldsymbol{H}(\boldsymbol{X})^{l-1} + \boldsymbol{X}. \tag{3.1}$$

where $1 \leq l \leq L \mid \boldsymbol{X} = 0$ for $l = 1 \because \exists\ \boldsymbol{H}(\boldsymbol{X})^{0} = x$; $\boldsymbol{H}(\boldsymbol{X})^{l}$, $F^{l}(\boldsymbol{H}(\boldsymbol{X})^{l-1})$, $\boldsymbol{H}(\boldsymbol{X})^{l-1}$ and x are of the same dimension. In this work, every stack of residual learning block composes two hidden layers. For a clearer conception of our proposed model, a single residual learning block of two hidden layers is shown in Figure 3.2 (b). From Figure 3.2 (b), assume that the underlying target function to be learned by a hypothetical residual learning block is $F^{l}(\boldsymbol{H}(\boldsymbol{X})^{l-1})$, then using the aforementioned constraints on l, it learns a residual function of the form

$$F^l(\boldsymbol{H}(\boldsymbol{X})^{l-1}) = \boldsymbol{H}(\boldsymbol{X})^l - \boldsymbol{H}(\boldsymbol{X})^{l-1} - \boldsymbol{X}. \tag{3.2}$$

For dropout of shortcut connections from the input layer to the stack of hidden layers l, we can write

$$F^l(\boldsymbol{H}(\boldsymbol{X})^{l-1}) = \boldsymbol{H}(\boldsymbol{X})^l - \boldsymbol{H}(\boldsymbol{X})^{l-1} - D * \boldsymbol{X}, \tag{3.3}$$

where $D \in \{0, 1\}$ and $D \sim Bernoulli(p_s)$ determines that $\boldsymbol{X}$ (shortcut connection from input) is connected to the stack of hidden layers l with probability p_s; that is, $P(D = 1) = p_s$ and $P(D = 0) = 1 - p_s$ for $0 \leq p_s \leq 1$; and $*$ defines an operator that performs the shortcut connection, given the value of D. The conventional dropout probability for hidden units is denoted p_h.

3.5 Maxout S-ResNet with Elastic Net Regularization

In this section, we present the details of our proposal for addressing the problems mentioned in Section 3.2. Particularly, our approach relies on learning maxout units, elastic net regularization and feature standardization as discussed below.

3.5.1 Learning units activation function via maxout

Conventional units in neural networks use 'hand-crafted' activation functions such as the rectified linear function, log-sigmoid function, tan-sigmoid, exponential linear function, etc. Maxout is an approach proposed in [58], where units activation functions are learned via the combination of piecewise linear functions. Moreover, [58] showed that maxout units allow a better approximation at test time of weights learned using the dropout technique [206], since model training with dropout can be considered as some of sort of ensemble model that implicitly share model parameters. The output of a maxout unit k at layer l, $h(x)^l_k$, can be written as follows

$$h(x)_k^l = \max_{j \in [1,c]} o(x)_{kj}^l, \tag{3.4}$$

where $o(x)_{kj}^l$ is the output of a linear regressor j at layer l, and c is the number of feature extractors or channels across which we max pool.

3.5.2 Elastic Net Regularization (ENR)

Elastic Net Regulation (ENR) allows the explicit regularization of neural network models by penalizing model parameters. The elastic net regularization can be seen as a linear combination of $L1$-norm and $L2$-norm regularizations [275]. While the $L1$-norm (or LASSO) regularization results in strictly sparse solutions performing what can be seen as feature selection, the $L2$-norm (weight decay) regularization only encourages solutions with small values without indeed setting them to zero. The $L1$-norm regularization has the problem of discarding features when employed for correlated features [275]; the ENR aims to overcome this shortcoming by adding $L2$-norm to counteract the aforementioned problem with $L1$-norm regularization. For training neural networks with ENR, we can minimize the negative conditional log-likelihood of data given the model parameters along with the $L1$-norm and $L2$-norm penalties as follows

$$J(\boldsymbol{W}) = -\arg\min_{w} \sum_{n=1}^{N} \log P(\boldsymbol{y}^{(n)} | \boldsymbol{x}^{(n)}; \boldsymbol{W}) + \lambda_1 \|\boldsymbol{W}\|_1 + \lambda_2 \|\boldsymbol{W}\|_2^2, \tag{3.5}$$

where $J(\boldsymbol{W})$ is the cost function, $\boldsymbol{x} \in \mathbb{R}^d$ is the input data, $\boldsymbol{y} \in \mathbb{R}^k$ is the output, $\boldsymbol{W} \in \mathbb{R}^{d \times k}$ is the model parameter, n is the index of training samples, λ_1 and λ_2 control the magnitude of the $L1$-norm and the $L2$-norm penalties, respectively.

3.5.3 Feature standardization

Feature standardization (FS) (or Z-score normalization) in machine learning has been shown to improve generalization [16]. Features are rescaled to realize the characteristics of a standard normal distribution; that is, centering data to have a mean of zero and standard

deviation of one. An explanation for the impact of feature standardization is that all features are transformed to the same scale such that features which vary less are not dominated by features that vary more. Feature standardization can be obtained by using

$$x_z^{(i)} = \frac{x^{(i)} - \bar{x}^{(i)}}{\sqrt{Var[x^{(i)}]}}, \tag{3.6}$$

where $x^{(i)}$ is an input data feature with index *i*, $\bar{x}^{(i)}$ is the mean and $Var[x^{(i)}]$ is the variance. In this chapter, we propose S-ResNet with maxout units for: (1) preserving the representation capacity of the S-ResNet, since ReLUs can die and learn nothing during training (2) improving parameters approximation of dropout trained S-ResNet at test time, since maxout units result in piecewise linear components for constructing units activation functions (3) tackle model overfitting with ENR as a result of the increase in model parameters introduced by the maxout units. Interestingly, we show that the features learned by the modified model using our training approach are very linearly separable such that a linear support vector machine (SVM) can replace the fully connected layers of the S-ResNet and achieve state-of-the-art results. Finally, we are able to improve experiment results by standardizing the features learned from the convolution layers stage of the maxout S-ResNet for training the linear SVM.

The training approach for the model that we propose herein is given as algorithm 3.1, where P_s, P_h, λ_1, λ_2, η, α and N_e are the dropout rate for the identity stochastic input shortcut connections, drop rate of hidden units, weight for $L1$-norm penalty, weight for $L2$-norm penalty, learning rate, momentum rate and maximum number of training epochs, respectively; SGD is stochastic gradient descent and BN is batch normalization. The aforementioned parameters are usually chosen using a validation set. The model trained using this algorithm is referred to as 'Maxout S-ResNet+ENR+FS+SVM'.

Algorithm 3.1: Maxout S-ResNet+ENR+FS+SVM

1. Set P_s, P_h, λ_1, λ_2, η, α & N_e using a validation set
2. Train S-ResNet via SGD+BN: feed data in mini-batches
3. If stopping condition for (2) is reached, discard the fully connected layers and do (4), else do (2)
4. Save model parameters
5. Forward propagate input data through model
6. Standardize the features obtained from the model's last layer
7. Train a linear SVM on the standardized features

3.6 Experiments and discussion

For demonstrating the effectiveness of our proposed model, we train very deep networks and observe their optimization characteristics over various training settings using the USPS and MNIST datasets. The USPS dataset[1] composes handwritten digits 0-9 (10 classes) of 7,291 training and 2,007 testing samples; while the MNIST dataset[2] composes handwritten digits 0-9 of 60,000 training and 10,000 testing samples. For the USPS dataset, we use 2×2 convolutional filters, 2×2 max pooling windows and 2 fully connected layers of 300 ReLUs. For the MNIST dataset, we use 3×3 convolutional filters, 2×2 max pooling windows and 2 fully connected layers of 500 ReLUs. For both datasets, models have output layers of 10 softmax units. Our 54-layer S-ResNet model composes 50 convolution layers, 2 max pooling layers and 2 fully connected layers; we apply batch normalization only in the fully connected layers. For the USPS dataset, we use $P_s = 0.8$, $P_h = 0.7$, $\lambda_1 = 10^{-5}$, $\lambda_2 = 10^{-3}$, $\eta = 0.1$, $\alpha = 0.9$ and $N_e = 100$. The learning rate was annealed from its initial value to the final value of 5×10^{-5} for the last 20 epochs. For the MNIST dataset, we use $P_s = 0.8$, $P_h = 0.8$, $\lambda_1 = 10^{-6}$, $\lambda_2 = 10^{-4}$, $\eta = 0.1$, $\alpha = 0.9$ and $N_e = 550$; the learning rate was annealed from its initial value to the final value of 5×10^{-5} for the last 270 epochs.

Figure 3.3 (left) shows the performance of our proposed 54-layer S-ResNet on the USPS dataset with different number of hidden layers at a dropout probability of $p_s = 0.8$ for the input shortcut connections to the hidden layers; for the conventional dropout of hidden units, a dropout probability of ($p_h = 0.6$) is used. In addition, Figure 3.3 (right) shows the performance of the best proposed model (54 hidden layer S-ResNet) with different dropout

[1] http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html
[2] http://yann.lecun.com/exdb/mnist/

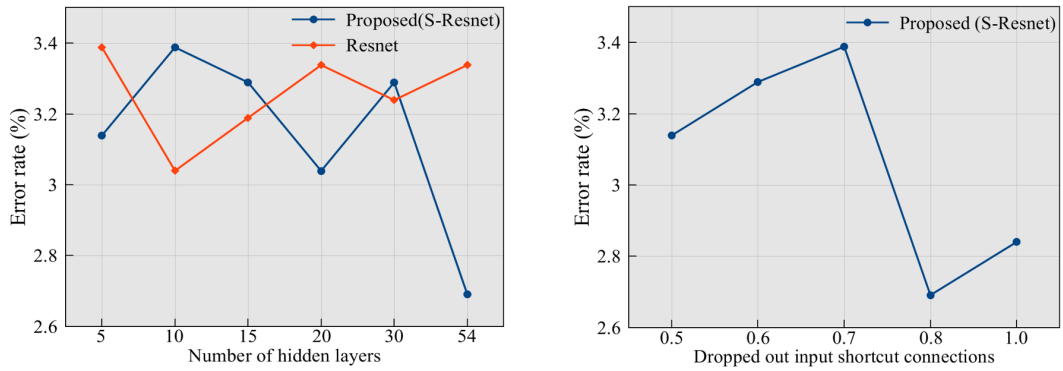


Figure 3.3: Performance of deep architectures with depth on the USPS dataset. Left: Test error rate with depth. Right: Test error rate for different dropout probabilities of input shortcut connections

probabilities for input shortcut connections to the hidden layers. Table 3.1 shows the error rates obtained on the test data for the USPS dataset along with the state-of-the-arts results. It will be seen that with 54-hidden layers, our S-ResNet achieves a state-of-the-art performance; that is, an error rate of 2.69%, surpassing the conventional ResNet (baseline model). Interestingly, it is seen that the shallower 20-layer S-ResNet achieves a higher test error rate of 3.04%. Furthermore, it is seen that the proposed 54-layer maxout S-ResNet+ENR+FS+SVM achieves an error rate of 2.19%. We observe that the models with dagger (i.e †) employed some form of data augmentation (or manipulation). For example, [241, 243] extended the training dataset with 2,400 machine-printed digits; while [22] employed virtual data in addition to the original training data. However, our proposed models employs no such data augmentation tricks.

We repeat similar experiments on the MNIST dataset. Figure 3.4 (left) shows the error rates of the S-ResNets and the conventional ResNets with different number of hidden layers. It is observed that the S-ResNets are better regularized as compared to the ResNets for all the different model depths. Particularly, with 54 hidden layers, the S-ResNet achieved a result competitive with the state-of-the-art results; we reach an error rate of 0.52%. Figure 3.4 (right) shows the error rates of the 54-hidden layer S-ResNet with different dropout probabilities for the input shortcut connections to the hidden layers. In Table 3.2, we report the obtained error rates for our experiments, along with the best results reported in recent

Table 3.1: Error rate (%) on the USPS dataset

Models	Test error (%)
Invariant vector supports [188]	3.00
Neural network (LetNet) [192]	4.20
Sparse Large Margin Classifiers (SLMC) [232]	4.90
Incrementally Built Dictionary Learning (IBDL-C) [219]	3.99
Neural network + boosting [192]	†2.60
Tangent distance [191]	†2.50
Human performance [191]	2.50
Kernel density + virtual data [92]	†2.40
Kernel density + virtual data + classifier combination [92]	†2.20
Nearest neighbour [92]	5.60
Baseline: ResNet - 54 hidden layers	3.34
Proposed: S-ResNet - 20 hidden layers	**3.04**
Proposed: S-ResNet - 54 hidden layers	**2.69**
Proposed: Maxout S-ResNet+ENR+FS+SVM - 54 layers	**2.19**

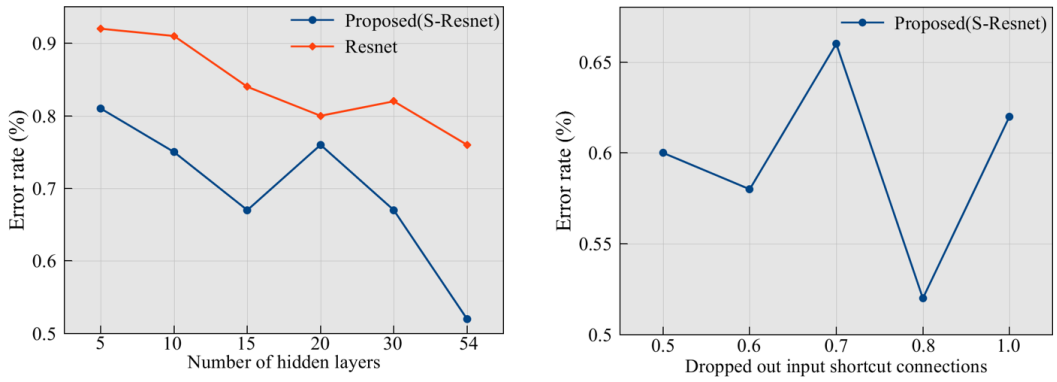


Figure 3.4: Performance of deep architectures with depth on the MNIST dataset. Left: Test error rate with depth. Right: Test error rate for different dropout probabilities of input shortcut connections

works. Also, for the MNIST dataset, we found that dropping out input shortcut connections to the hidden layers with a probability of 0.8 yielded the best result as given in Table 3.2. For both datasets, the S-ResNets employed no explicit regularization technique for improving generalization capability; we relied on the implicit regularization of the models via dropout of input shortcut connections and hidden units for the S-ResNet, and dropout of hidden units only for ResNet. The shallower 15-layer S-ResNet achieves a higher error rate of 0.64%. Again, it is observed that the proposed maxout S-ResNet+ENR+FS+SVM model achieves a smaller error rate of 0.36%. It is interesting to note that the proposed models do not

Table 3.2: Error rate (%) on the MNIST dataset

Models	Test error (%)
Highway Net-16 [207]	0.57
Highway Net-32 [207]	0.45
Supervised Sparse Coding + linear SVM [26]	0.84
Deep Fried Convet [243]	0.71
PCANet [22]	0.62
Network in Network (NIN) [119]	0.45
Deeply Supervised Network (DSN) [107]	0.39
ConvNet + L-BFGS [103]	0.69
Neural network + adversarial examples [57]	0.78
Neural network ensemble + DropConnect [223]	0.52
Baseline: Residual network (Resnet) - 54 hidden layers	0.76
Proposed model (S-Resnet) - 15 hidden layers	**0.64**
Proposed model (S-Resnet) - 54 hidden layers	**0.52**
Proposed: Maxout S-ResNet+ENR+FS+SVM - 54 layers	**0.36**

suffer from convergence problem as reported in an earlier work which experimented with a similar training scheme [69]. In addition, the experimental results given in Tables 3.1 & 3.2 suggest that the proposed training schemes improve the implicit regularization of very deep networks; that is, lower test errors are achieved for the proposed models as compared to the ResNets. We conjecture that the simple modification employed for the proposed model helps to reduce the reliance of model units in one layer over others for feature learning. We observe that [223] also reported an error rate of 0.21%, however [223] employed some form of data augmentation using an ensemble of 5 neural networks; without data augmentation, they obtained a test error rate of 0.52%. Conversely, we employ no data augmentation and model ensemble.

3.7 Conclusion

Very deep networks suffer optimization problems even in situations that indeed favour over-fitting. Furthermore, assuming that we are able to optimize very deep networks, over-fitting is almost always inevitable due to large model capacity. We address the aforementioned problems by taking inspiration from residual learning. Our proposed model, stochastic residual network (*S-ResNet*), employs stochastic shortcut connections from the input to the hidden layers for essentially improving the implicit regularization of very deep models.

Furthermore, very deep networks have large representational capacity; hence, they require appropriate regularization to alleviate overfitting the training data. We build on S-ResNet for improving generalization of very deep networks. As such, we propose to learn the activation function of units in the S-ResNet via maxout units since ReLU units can die out in training and impact representation capacity of very deep networks. We employ Elastic Net Regularization (ENR) and Feature Standardization (FS) for further regularization of the S-ResNet model due to increased parameterization based on the maxout units. Our experimental results show that this method performs better than S-ResNet and earlier methods, including human performance on the USPS dataset. On the MNIST dataset, we obtain very competitive results.

Chapter 4

Reformulated Highway Neural Network for Improved Training

In this chapter, we take a step further in the training of very DNNs by propose a highway network with gate constraints that improves the optimization and generalization of very DNNs. Namely, we propose two variants of highway network, *HWGC* and *HWCC*, employing feature summation and concatenation respectively. The proposed highway networks, besides being more computationally efficient, are shown to have more interesting learning characteristics such as natural learning of hierarchical and robust representations due to a more effective usage of model depth, fewer gates for successful learning, better generalization capacity and faster convergence than the original highway network. Experimental results show that our models outperform the original highway network and many state-of-the-art models. Importantly, we observe that our second model with feature concatenation and compression consistently outperforms our model with feature summation of similar depth, the original highway network, many state-of-the-art models and even ResNets on benchmarking datasets. Furthermore, the second proposed model is more computationally efficient than the state-of-the-art in view of training, inference time and GPU memory resource, which strongly supports real-time applications.

4.1 Introduction

Deep neural networks have found applications in many real-life tasks; their successes for learning different difficult problems are well documented. In particular, the field of computer vision has hugely benefited from deep neural networks for various applications ranging from pose estimation [41], segmentation [6], action recognition [163], face recognition [154], etc. In recent times, there is a growing trend of using deep networks for learning directly from raw data (i.e. without feature decorrelation, feature selection, image restoration and denoising, etc.) or with very minimal data processing. In essence, deep networks are burdened to cope with various data irregularities, rather than carefully handcrafting the features that are thought to be important for the task at hand. A motivation for this line of application is that lesser human involvement is required; this translates to shorter time for building models and a reduction in data processing costs. Moreover, the difficulty of the problems that we intend to solve over time has consistently increased. For example, the MNIST dataset[1] that was considered very difficult many years ago, and thus used for benchmarking, is no longer considered as a hard dataset; many works [130, 160] have reported error rates in the range [0.5%, 0.21%]. However, tackling harder learning tasks has motivated a new direction for deep neural networks with many layers of feature representations. Such models are referred to as very deep networks[2] [207]. Interestingly, empirical [207, 67] and theoretical [40, 135, 15] evidences show that increasing the depth of deep models increases the complexity of the functions that they can capture. The hope is that with the availability of massive amount of data, we can learn a robust classifier that can generalize well on unseen data. Consequently, deep networks with over 15 layers have been reported to produce state-of-the-art results in many works [67, 77]; models with hundreds of layers can be found in [67, 189].

Although very deep networks give very promising results, they are hard to train [207, 67]. In fact, using very simple and small datasets, [162] showed that different initialization schemes [56, 68] and batch normalization [84] do not alleviate the problem of optimizing very deep networks beyond a certain depth. A major problem associated with training very

[1] http://yann.lecun.com/exdb/mnist/

[2] Deep models with 15 or more layers

deep networks is that learning the identity function is hard; this problem compounds as depth increases [67]. Consequently, many works [207, 67, 77] that successfully trained very deep networks relied on some means of routing untransformed features from the earlier layers through the model; that is, learning near identity functions for the different layers. One of the earliest models with over 20 layers that was successfully trained is referred to as a *highway network* [207]. The highway network is essentially a stack of highway blocks that is trained end-to-end using the gradient descent optimization method. Particularly, the highway block relies on a gating mechanism for routing untransformed lower layer features through the model to alleviate the problem of model optimization. In another work [67], Residual Network (ResNet) with skip connections of identity mappings was proposed for training very deep networks; the ResNet is essentially a stack of residual blocks with *skip connections*, also known as *shortcut connections*. Skip connections allow the direct routing of lower layer transformations to higher layers, and therefore ResNet can learn near identity functions while alleviating the difficulty of model optimization.

Although very impressive results were reported for the original highway network [207], there are learning shortcomings that are discussed as follows. Indeed, the highway blocks used for constructing the model may make it difficult to learn new feature transformations (or latent representations) which are important for generalization as training progresses. This is a result of how highway blocks are learned. First, the gating units use the sigmoid activation function that limits output range to 0-1. At the start of training, the biases of the gating units of all highway blocks are initialized to negative values. As a result, this situates the gating units close to saturation. While this setup alleviates the difficultly of model optimization by mostly routing untransformed features via the highway block, the gating units further go into saturation as training progresses. This learning setup favours learning essentially untransformed features, and therefore may impact model generalization that typically relies on learning new feature representations in the different hidden layers. Thus, very deep networks learned using highway blocks of the described form may suffer a reduction in model effective depth due to lack of new latent representations for training data, since emphasis is put on learning untransformed features. Alternatively, the same problem can be considered

as learning too close to the identity function regime at the expense of learning new transformations; as such, this can hurt generalization performance since the model becomes *inherently* much shallower than the topology implies. Furthermore, employing a gate for every layer translates to roughly doubling the number of parameters for the model. Subsequently, model overfitting can become a concern.

In this chapter, the problem of training very deep networks using gating mechanisms as in the highway network [207] is revisited. We address the aforementioned problems of the highway network [207] by employing gate constraints and reformulating the fashion in which transformed and untransformed features are gated. As such, two highway networks, highway network with gate constraints (HWGC) and highway network with concatenated and compressed features (HWCC) are proposed. HWGC uses features summation, while HWCC relies on the concatenation of gated features and compression to achieve better results. Particularly, we note that HWCC is computationally more efficient than HWGC, and allows us to outperform or achieve competitive results with ResNet [67, 69] using similar number of model parameters, but with significantly lesser depth. The model *HWGC* was first presented in [158].

The two highway network models, HWGC and HWCC, presented in this chapter allow a natural extraction of hierarchical feature representations as training progresses. At the start of training, we learn very close to the identity function regime to aid model optimization. However, as training progresses, the model can learn further away from the identity function regime such that new transformations can be extracted in favour of generalization performance. Importantly, the aforementioned merits are achieved without sacrificing the ease of model optimization. In fact, the models can be trained using less epochs than for the original highway network [207]. Specifically, our contributions based on the proposed highway blocks are as follows:

1. More effective use of model depth, since there is a natural learning of new latent representations for data as training progresses; this is critical for the generalization of deep networks.

2. Constructing a very deep highway network that relies on a far smaller number of gates for training than in the original highway network [207]; this can alleviate the problem of model overfitting and improve computational efficiency. Specifically, one gate and three gates can be used for training models having up to 19 layers and 32 layers, respectively.

3. Using the similar (or a smaller) number of model parameters, the proposed models outperform the original highway network [207] and many state-of-the-art models [67, 118, 107, 101, 119] on five benchmarking datasets. Namely, on CIFAR-10, CIFAR-100, Fashion-MNIST and SVHN datasets, our 19-layers HWGC and HWCC outperform (or at least match) ResNet with 110 [67] and 164 layers [69] that are roughly 6 and 8 times deeper, respectively. On the imagenet dataset, our 34-layers HWGC and HWCC outperform (or at least match) ResNet with 101 layers [67] that is roughly 3 times deeper; and again our 50-layers HWGC and HWCC outperform (or at least match) ResNet with 152 layers [67]. Importantly, HWCC performs better than HWGC on the datasets.

4. For obtaining similar level of model generalization capacity, the proposed highway network models (HWGC, and especially HWCC) in comparison to the original highway network [207], ResNets [67, 69] and DenseNet [77] result in significant reduction of inference time and Graphics Processing Unit (GPU) memory requirement for supporting real-time applications, since they require less depth.

For validating the proposed highway network blocks, CIFAR-10, CIFAR-100, Fashion-MNIST, SVHN and imagenet datasets are used. The remaining sections of this chapter are organized as follows. In Section 4.2, works discussing the theoretical motivation for very deep networks, the challenges of training them, the impact of depth on model generalization, along with the different approaches for training are discussed as related works. Section 4.3 contains the background on highway networks, including the problem statement. In Section 4.4, we give the formulation and construction of HWGC and HWCC for alleviating the difficulty of training. Section 4.5 presents the settings for model training and the experimental results. The chapter is concluded in Section 4.6.

4.2 Related work

4.2.1 Theoretical motivation for deeper models

Along side empirical evidence [207, 67], there are theoretical works [40, 135, 15] that study the importance of model depth for learning highly varying functions. In [40], sum-product networks are used for analyzing the family of functions that can be learned using models of different depth. The conclusion in the work is that there are families of functions that can be efficiently learned with a deeper model, but using a much shallower model would require an exponential increase in the number of model parameters for learning the same family of functions. Another work [135] arrived at a similar conclusion as in [40], that deeper models are more efficient in approximating very complex target functions; especially those that are compositional. In addition, the same work, Mhaskar et al. [135] noted that deeper networks are capable of working at lower Vapnik-Chervonenkis (VC) dimensions compared to shallower models. The power of deeper models to tackle more difficult problems was also studied in [15]. This work derived upper and lower bounds on model complexity considering the number of hidden units and different activation functions. The observation was that using a similar number of computing units, depth increase resulted in an improved capacity of deep networks for function approximation.

4.2.2 Challenges of training deeper models

Training very deep networks is not trivial. The first challenge is in the fundamental problem of optimization; that is, fitting the training set to the model is difficult. High error rates are typically obtained on the training set. Subsequently, this nullifies any chance of good model generalization to unseen examples, since the training set cannot be learned properly in the first place.

Recently, there has been active research in trying to identify the problems that plague the training of very deep models. In [67], it is observed that learning the identity function is hard. In the work, 20-layer and 56-layer models were constructed for exemplifying the problem of learning the identity function. It was expected that the 56-layer model would be capable of

reaching the same training error as the 20-layer model, supposing that the extra 36 layers in the 56-layer model would simply learn the identity function; in this operation regime, the 56-layer model should behave similarly to the 20-layer model, at least on the training set. Instead, it was observed that the 56-layer model converged to a much higher training error than the 20-layer model; this empirically suggests that learning the identity function is hard for deeper networks. In another work [76], the *problem of feature reuse* was considered for the difficulty of training very deep networks. The work described feature reuse as a scenario where there is the dilution of feature transformations (or latent representations) learned from one layer to the following in the apparently long chain of layers in a very deep model. As a result, at the start of training, the features that reach the output layer are considerably unreflective of the input data, especially since model weights are typically initialized randomly. Subsequently, the error gradients computed in the output layer of the model are quite ineffective for driving the model parameters towards convergence. Hence, the model may fail or take 'forever' to converge.

Another well studied problem for the training of very deep networks is *exploding and vanishing gradients* [164, 185, 97]. Since for training, computed error gradients are back-propagated through several layers with multiplicative effect, there is a considerable chance that error gradients diminish or explode. In [7], it was showed that shattering gradients in which gradients resemble white noise (i.e. are uncorrelated) plague training of very deep networks. Moreover, very deep networks seem to have badly conditioned Jacobians and Hessians of loss functions that hinder optimization [252, 115, 112].

4.2.3 Model performance with depth

Different tasks have been successfully learned with interesting results using deep networks of few layers (i.e. 3-7 layers) [176, 18]. Nevertheless, addressing more challenging tasks necessitates that the learning characteristics of current models is reassessed with a view to improving their capacity for generalization. For instance, the best top-5 error rate reported on the ILSVRC2012 dataset in 2012 was 15.3%, and it was obtained using AlexNet [100] which has about 60 million parameters. Considering the current trends in deep networks, AlexNet

with 5 convolution layers, 3 max pooling layers and 3 fully connected layers can be considered as a modest model. The ILSVRC2013 object classification challenge was won using a modified AlexNet model [255]. The model was named ZFNet [255] and was constructed by increasing the number of convolution layers to 8; it achieved a top-5 error rate of 14.8%. The ILSVRC2014 object classification challenge winner employed GoogleLeNet [210] which features a signficantly increased model depth in comparison to ZFNet. GoogleLeNet has 22 layers and uses smaller convolution filters of different sizes which produce aggregated features for what was referred to as an inception module. A drastic improvement in generalization performance was observed; a top-5 error rate of 6.67% was achieved. Interestingly, the runner up to ILSVRC2014 object classification challenge relied on a 16-layer model named 'VGG16' [194] that achieved a top-5 error rate of 7.3%. An 'unconventional' and exceptionally deep model with 152 layers achieved the smallest top-5 error rate in ILSVRC2015 object classification challenge; the model was referred to as ResNet [67] achieved an impressive top-5 error rate of 3.57%.

4.2.4 Approaches for training very deep networks

Motivated by empirical results on different hard tasks and insights from theoretical works, there are works that have successfully trained very deep networks with good results. Interestingly, one feature that all the works have in common is the use of untransformed features (i.e. skip connections) in the different layers of the proposed models. This has also been shown to alleviate the problem of vanishing and exploding gradient, and badly conditioned loss function Hessians [164, 7, 112].

In [67], the ResNet with skip connections was proposed. The different layers of the ResNet are grouped into residual blocks; typically, each block is composed of 2 or 3 layers. The output of a block is usually the addition of the learned transformation and the output of the preceding residual block via a skip connection. It is straightforward to see that the introduction of skip connections addresses the problem of learning the identity function and feature reuse. Consequently, this alleviates the difficulty of model optimization. In the same work [67], models having up to 1202 layers were successfully trained. In a following work [76],

very deep networks with stochastic depth were proposed. Essentially, the learning scheme is such that a random subset of the model layers are removed during training for every iteration. The remaining layers are bridged using the identity connections. The training scheme was reported to allow a successful training and even improve model regularization as compared to ResNet. Again, it can be seen that this training setup address the problem of learning identity functions and feature reuse. In a more recent work [77], the proposed model is such that each layer is connected to every other layer; the model was referred to as DenseNet. Many state-of-the-art results on different benchmarking datasets were reported using the DenseNet.

In a different model referred to as *highway network* [207], gating mechanisms were used for addressing the problem of learning identity function and feature reuse. It is one of the pioneering works that first successfully trained deep models having up to 50 layers proposed highway network [207]. Highway network was reported to have been motivated by the construction of the Long Short Term Memory (LSTM) recurrent network [74] for addressing the problem of learning long term dependencies. The highway network relies on gating mechanisms for routing untransformed features from lower layers to higher layers. Specifically, rather than feeding features from the previous layer (or block) all together to the succeeding one as in ResNet, gating mechanisms are used for learning what part of the features in the previous layer (or block) to be routed to the succeeding layer (or block). It was posited that learning what part of the previous feature transformation to carry over to the next layer may be more optimal than 'blindly' carrying over all the features for every layer (or block); gating mechanisms may help reduce features redundancy.

4.3 Background and problem statement

The background on the model that we build on, highway network [135], is presented in this section. In addition, a high level discussion of the problem statement is given.

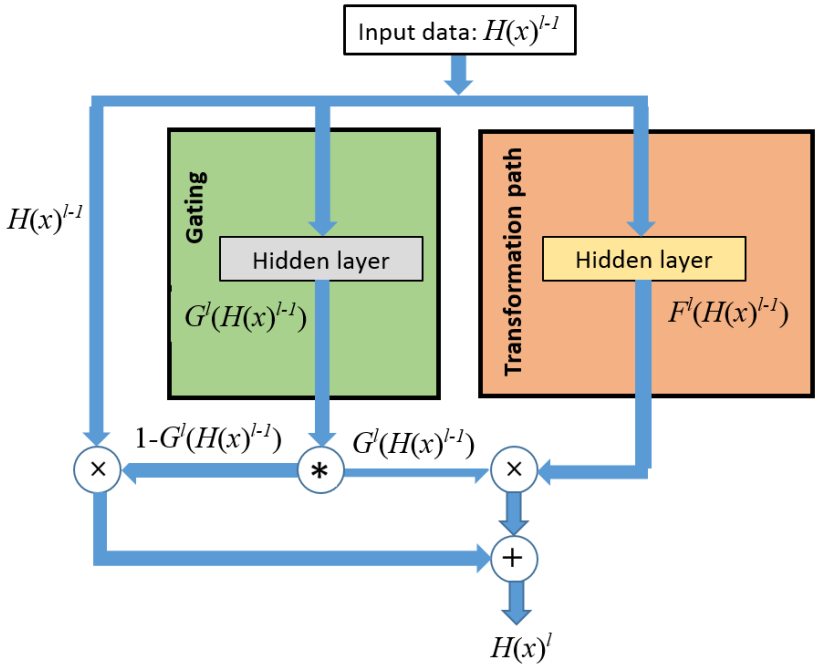


Figure 4.1: Highway network block [207]

4.3.1 Background: highway network

The highway network is usually a stack of highway blocks that are trained end-to-end. Each highway block employs a gating mechanism for controlling information flow through the model. Given that $\boldsymbol{H}(\boldsymbol{x})^{l-1}$ is the information on the highway at layer $l-1$, the gating module outputs a signal $G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})$ for controlling what information is routed to the succeeding highway network block. The final output of the highway network block at layer l, $\boldsymbol{H}(\boldsymbol{x})^l$, can be written as

$$\boldsymbol{H}(\boldsymbol{x})^l = F^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1}) + \boldsymbol{H}(\boldsymbol{x})^{l-1}(1 - G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})), \tag{4.1}$$

where $F^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})$ is the transformation learned at hidden layer l for input $\boldsymbol{H}(\boldsymbol{x})^{l-1}$. The form of feature learning in (1) was shown to allow the successful training of very deep networks [207]. Based on the given formulation, the gate G^l at layer l can either permit or hinder the flow of incoming signals $\boldsymbol{H}(\boldsymbol{x})^{l-1}$ and $F^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})$, depending on the current state of

the gating units; the gates can be either *open* or *closed*. The highway block is shown in Figure 4.1. For alleviating the difficultly of model optimization due to signal 'degradation' with depth increase (as earlier discussed in Section 4.1), it follows that the gating units are initialized in manner that most of $\boldsymbol{H}(\boldsymbol{x})^{l-1}$ (untransformed incoming features) are routed via the block early at the start of training to favour early model optimization. The training scheme for achieving this is described as follows:

i The units in the transformation path use the rectified linear function or similar activation function. However, the gating units use the Log-Sigmoid function as the activation function so that the range of units' outputs is from 0 to 1; where, states 0 and 1 denote closed and open gates, respectively.

ii The biases of the gating units, g^l, in G^l, at layer l are initialized to negative values such as -1 or -3 [135] to ensure that most of the features learned at layer l, $\boldsymbol{H}(\boldsymbol{x})^l$, are untransformed features, $\boldsymbol{H}(\boldsymbol{x})^{l-1}$. This is due to the gating units' activations, $G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})$, being close to 0; see (4.1).

4.3.2 Problem statement

To address the problem of training very deep networks and fast convergence, [67] proposed the ResNet for which the transformation of the form given below is learned

$$\boldsymbol{H}(\boldsymbol{x})^l = F^l(\boldsymbol{H}(\boldsymbol{x})^{l-1}) + \boldsymbol{H}(\boldsymbol{x})^{l-1}, \tag{4.2}$$

where notations are the same as in (1).

We observe that [67] posited that ResNet blocks *always* allow the routing of untransformed features from one ResNet block to the succeeding one, and therefore good convergence is achieved. However, the output of a highway network block is the combination of some transformed and untransformed features. The main operation of the highway network block is to learn using a gating mechanism that controls what part of the transformed features and untransformed features to route to the next layer (or block). Generally, the highway

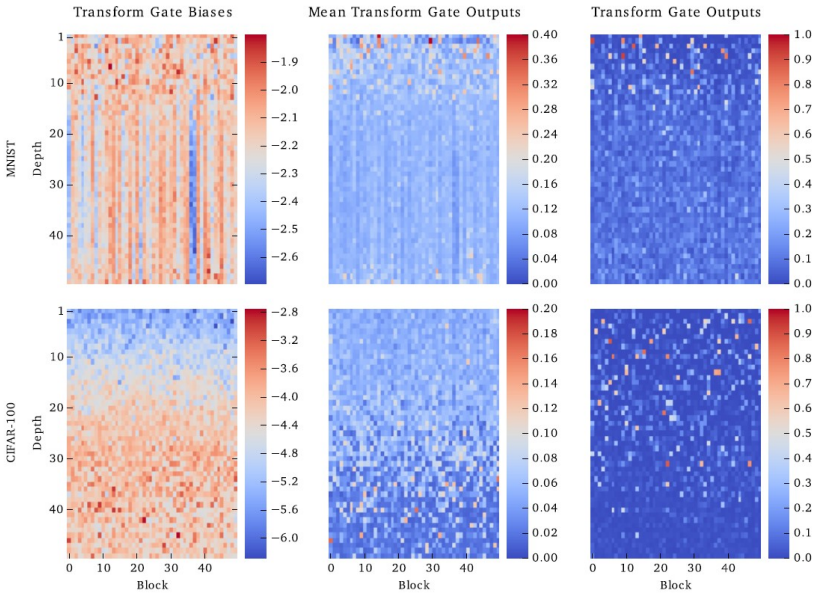


Figure 4.2: Original highway block learning attributes. Top row: 50-layer model trained on MNIST dataset. Bottom row: 50-layer model trained on CIFAR-100 dataset. First column: transform gate biases. Second column: mean of the transform gate outputs over the whole training set. Third column: transform gate output using a random training sample [207]

block routes mostly untransformed features at the start of training to alleviate the difficulty of training. As the training progresses, the hope is that the highway block starts to route more transformed features. The smooth transitioning from routing untransformed features to routing transformed features is a nice attribute of the highway network. It is noted in [207] that the gating units of the highway blocks respond selectively to different incoming features. This means that the gates perform some sort of feature filtering. In contrast, ResNet does not possess this interesting learning characteristic. Nevertheless, we discuss in the remaining part of this section some concerns on the learning scheme of the original highway network block that may negatively impact model generalization.

Mainly, we argue that the form of transformation learning for the highway block proposed in [207] and given in (4.1) does not facilitate the natural extraction of new latent representations. At the start of training, the biases of the gating units for the highway block are initialized to negative values, and therefore positions the gating units close to the lower end of the saturation regime of the sigmoid activation function; that is, close to zero. While this training

setup suffices for routing untransformed features from the lower layers to higher layers, this may not be the optimal way to use the gating units since it is well known that Log-Sigmoid units saturate to very small values during training for deep networks [237]. This was also noted in [207]. A direct result of this is that learning new latent representations (or transformations) is hard, since the gating units go further into the lower end of the saturation regime (i.e. become smaller than the initialized values) and therefore places emphasis on routing untransformed features at the expense of routing transformed features which are critical for model generalization; see (4.1). Specifically, $G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})$ converges to zero as training progresses and therefore we can rewrite (4.1) as

$$\boldsymbol{H}(\boldsymbol{x})^l \approx \boldsymbol{H}(\boldsymbol{x})^{l-1} \quad \text{for} \quad i \gg 1 : 1 \leq i \leq t, \tag{4.3}$$

where i is the epoch index and t is the number of maximum epochs. It can be seen that the result given in (4.3) does not readily favour model generalization, since the highway network block routes mostly untransformed features to the succeeding block, and new features are not being effectively learned. The fact that gating units go into the lower end of the saturation spectrum (i.e. very close to zero) during training was also acknowledged in the original highway network block [207]. At the start of training, the biases of the gating units for 50-layers models trained on MNIST and CIFAR-100 datasets were initialized to -2 and -4, respectively. However, it was noted that the biases of the gating units became smaller after training. Correspondingly, the mean transform gating units' outputs over the whole MNIST and CIFAR-100 datasets are close to zero; the same is observed for a single random sample as shown in Figure 4.2.

In addition, another concern is that for tackling the problem of optimization, the original highway network [207] uses gates for every layer of features transformation. This leads to a significant increase in the number of model parameters. A gate can double the overall number of parameters of any layer where it is used. Consequently, there is increased concern of model over-fitting the training data. Moreover, relying on gates for every layer could result in the transformation path becoming somewhat redundant (i.e. not constrained to capture

important features), since the gate parameters are enormously large and sufficient to fit the training data [158]. Again, it is observed in our experiments that this may impact model generalization performance. In the following section, we focus on the formulation and learning characteristics of the proposed highway network block for training very deep networks.

4.4 Proposed approach

The details of the proposed highway network block that addresses the aforementioned problems (i.e. discussed in Section 4.3.2) for the original highway network block are presented in this section. Namely, the two proposed models, HWGC and HWCC, are discussed.

4.4.1 Highway network with gate constraints

The proposed highway network in this chapter, HWGC, is reformulated to achieve the following important learning characteristics:

(i) Prioritize model optimization at the start of training. That is, the gates mostly route untransformed features at the start of training.

(ii) Prioritize learning new feature transformations further into training. That is, the gates route mostly transformed features.

(iii) Require significantly fewer gates for learning several layers of feature transformations, since gate transitioning is more effective for model optimization and generalization.

HWGC block is so constructed to realize the aforementioned learning characteristics by learning a transformation of the form

$$\boldsymbol{H}(\boldsymbol{x})^l = F^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})(1 - (G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1}))^m) + \boldsymbol{H}(\boldsymbol{x})^{l-1}(G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1}))^n, \tag{4.4}$$

where m and n are model hyper-parameters which are employed for remapping the gating function and other denotations remain as earlier stated. HWGC block is shown in Figure 4.3. The nice characteristic transitioning of learning regime mentioned in (i) & (ii) permits

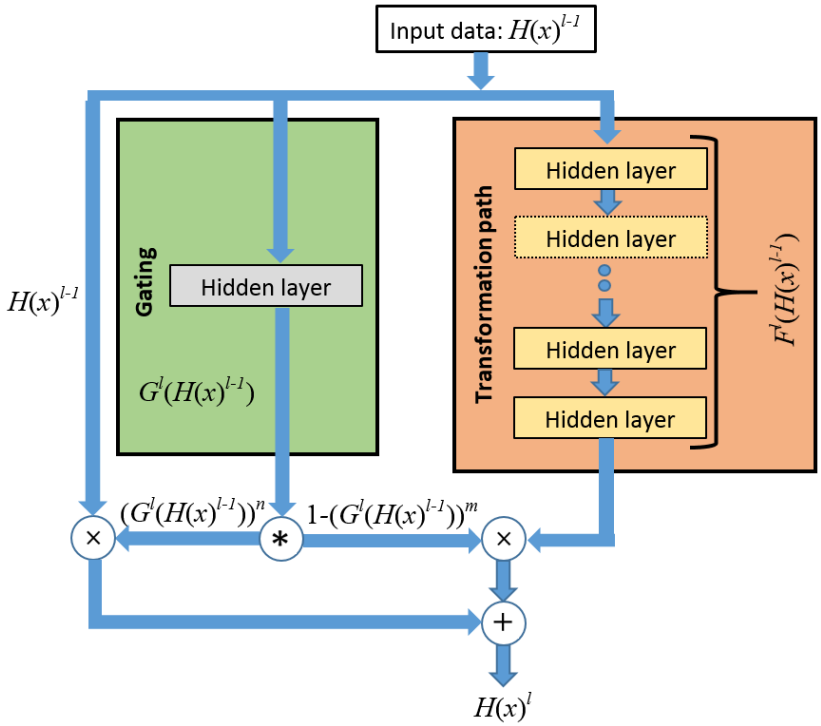


Figure 4.3: Highway network block with gate constraints (HWGC block) [158]

the efficient use of model parameters and necessitates fewer gates for successful training as mentioned in (iii). Thus, there is a lower risk of model over-fitting the training data. In the remaining parts of this section, we discuss the three components of the proposed highway network block given in (4.4) that are required for learning. Namely, the components are parameters initialization, feature remapping and gate constraints.

Parameters initialization

The initialization of gating units in highway blocks is very critical for the successful training of highway networks. Similar to the original highway network block [207], the biases of the gating units are leveraged for controlling the states of the gates. Specifically, we desire gates' states that favour the routing of mostly untransformed features at the start of training to ease model optimization. Following the formulation given in (4.4), the biases of the gating units are initialized to relatively small values such as 3 so that the gates are open at the

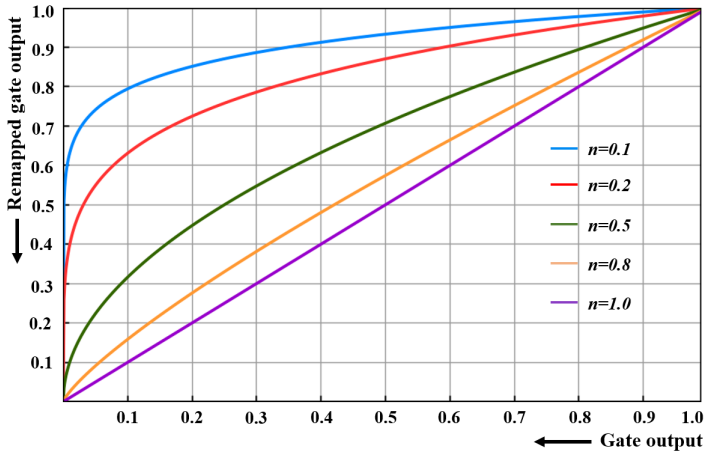


Figure 4.4: Gate remapping

start of training and therefore favour model optimization. Generally, higher bias values are preferred for deeper models [207]. However, Log-Sigmoid units in the proposed highway block can quickly saturate to small values during training, and slow down convergence. In order to tackle this problem, a simple remapping of gating units that is discussed in the following section is employed.

Feature gating and re-mapping

For extending the regime of training where the highway block gate routes mostly untransformed features (i.e. in open state and therefore prioritize optimization), model hyperparameters m and n given in (4.4) are introduced. Since the gating units are Log-Sigmoid units, their outputs are thus bounded as follows: $0 < G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1}) < 1$. A simple way to emphasize early optimization in training is to put the constraint $n < 1 \leq m$ on the gate as shown in (4.4); this impedes gate closure on the information highway (i.e. $\boldsymbol{H}(\boldsymbol{x})^{l-1}$) early in training, and subsequently reduces the impact of gate saturation on optimization. Also, m can be chosen to control the impact of the gate on the highway block transformation path, $F^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})$. However, as optimization is a major problem for very deep networks, we set $m = 1$ and thus focus on n that directly affects how fast gating units go into saturation. The remapping (or scaling) of gating outputs from $G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1})$ to $(G^l(\boldsymbol{H}(\boldsymbol{x})^{l-1}))^n$ for the

information highway path $\boldsymbol{H}(\boldsymbol{x})^{l-1}$ for different values of n is shown in Figure 4.4. It can be observed in Figure 4.4 that the output values of the gating units would be maximum (i.e. roughly 1) at the start of training; this is due to the manner in which the gates are initialized; see Section 4.4.1. The arrows show that the output of gating units would decrease as training progresses; however, the remapped output of gating units would decrease much slower. As such, n is a hyperparameter that is set during training.

Gate constraint

Highway network blocks rely on gates that are parameterized, and therefore can aggravate the problem of model over-fitting. Moreover, in large models with enormous gate parameters, the gates could become flexible enough to hold reasonable amount of latent representations, and as such perform more than alleviating the difficulty of model training; consequently, the transformation path may become somewhat redundant. In order to tackle this problem, the gate parameters are constrained with the hope that their resulting degree of freedom allows for mainly filtering (gating) operation on the information highway, $\boldsymbol{H}(\boldsymbol{x})^{l-1}$. To achieve this, the max-norm constraint is put on the weights of the gating units; that is, $\| \overrightarrow{w} \| < c$, where w is the weight vector of an arbitrary gating unit. Generally, the appropriate value of the hyperparameter, c, would be chosen based on model size. This training setup thus enforces the transformation path to learn important features (or representations).

When the different components of the new highway network block are put together, it can be seen that when the gating units go into saturation as it is typical for Log-sigmoid units in deep networks during training, we can write (4.5) as

$$\boldsymbol{H}(\boldsymbol{x})^{l} \approx F^{l}(\boldsymbol{H}(\boldsymbol{x})^{l-1}) \quad \text{for} \quad i \gg 1 : 1 \leq i \leq t, \tag{4.5}$$

since $G^{l}(\boldsymbol{H}(\boldsymbol{x})^{l-1}) \longrightarrow 0$, where notations remain the same as previously stated. From the result given in (4.5), important implications of the proposed highway block can be drawn as follows:

(i) The proposed highway block essentially learns feature transformations in (4.5) further

in training; this is important for model generalization. This contrasts with the original highway network block that learns mostly untransformed features and prioritize this learning regime much further in training at the expense of learning new latent representations; see (4.3) and Figure 4.2.

(ii) The extreme scenario of the proposed highway block suggests that the proposed model starts out as a relatively 'shallow' model as in (4.3) and mostly routing untransformed features. However, the model depth grows towards the effective depth as training progresses, since gating units saturate such that we can realize (4.5). That is, there is nice transitioning from optimization-based to generalization-based regime.

4.4.2 Highway network with gate constraints, and concatenated and compressed features

Computational efficiency is very important in deep network as it is generally desirable to achieve good performance using a model of reasonable size. As such, we take inspiration from recent works that employed fusion of latent representations [156, 77] to propose features concatenation (rather than features summation in HWGC) such that we can leverage features that are already learned at different scales in the earlier layers. The subsequent compression of the output of the proposed highway block allows the extraction of concise representations and maintaining the same number of convolution maps for the succeeding layer. As a result, HWCC generalizes better than HWGC and it can be trained in reasonable time. Thus, it is suitable for applications that require reasonably small inference time. To achieve this, we reformulate the proposed HWGC block as HWCC block to realize the following learning characteristics:

(i) Preserve features that are already learned at different scales in the earlier layers via features concatenation, as opposed to feature summation in [207, 158].

(ii) Reduce features redundancy and proliferation of convolution feature maps for computation via feature compression. This keeps required computation time comparable to HWGC that uses features summation.

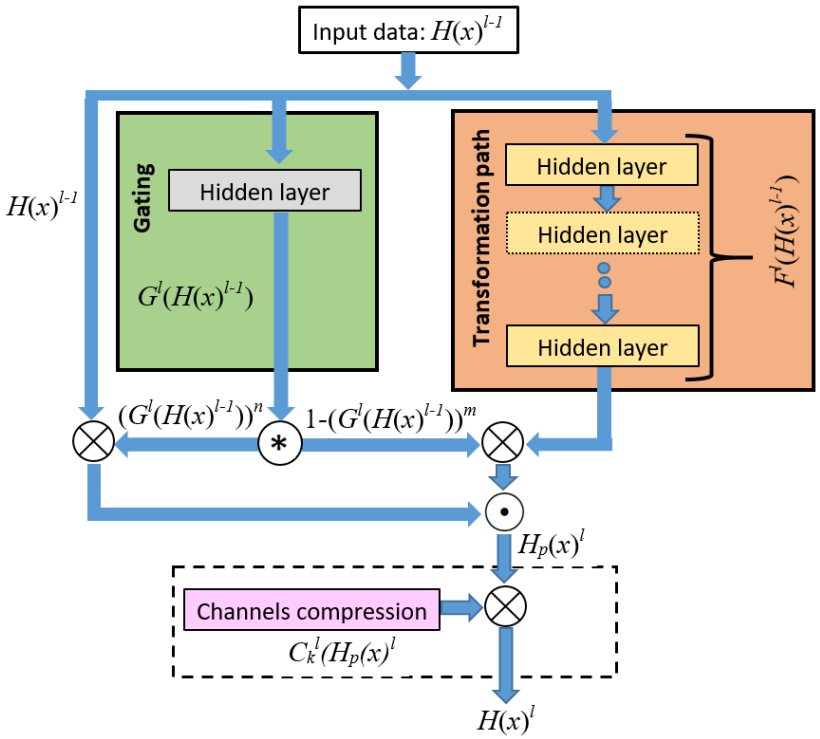


Figure 4.5: Proposed highway network block with gate constraints and concatenated features (HWCC block)

For obtaining the aforementioned learning characteristics, HWCC is so formulated that it learns a transformation of the form

$$\boldsymbol{H_p(x)}^l = [F^l(\boldsymbol{H(x)}^{l-1})(1-(G^l(\boldsymbol{H(x)}^{l-1}))^m) \odot \boldsymbol{H(x)}^{l-1}(G^l(\boldsymbol{H(x)}^{l-1}))^n], \tag{4.6}$$

where $\odot$ denotes the *concatenation* operation along convolution maps (or channel) axis. Figure 4.5 shows the proposed highway network block. The output of HWCC given in (4.6) is obtained by concatenating the outcome of the gated-transformed features, $F^l(\boldsymbol{H(x)}^{l-1})(1-(G^l(\boldsymbol{H(x)}^{l-1}))^m)$, and gated-untransformed features $\boldsymbol{H(x)}^{l-1}(G^l(\boldsymbol{H(x)}^{l-1}))^n$. However, this can result in features redundancy and doubling the number of convolution maps feeding into the succeeding layer. In order to tackle these problems, an appropriate number of convolutions filters of suitable size can be employed to achieve features compression as shown in Figure 4.5. Subsequently, using (4.6), we can

write the final block output

$$\boldsymbol{H}(\boldsymbol{x})^l = C_k^l(\boldsymbol{H_p}(\boldsymbol{x})^l), \tag{4.7}$$

where C_k^l uses r×r convolution filters at layer l, and k is the number of convolution filters. It can be observed that besides the function of reducing the number of output convolution maps for the highway block, compression (bottleneck) operation of C^l can further aid the disentangling of factors of variations in the training data by capturing more concise features representation from both new transformations and transformations that are carried over.

4.5 Experiments

In this section, we describe experiments that are used for validating the proposed highway network block. Specifically, experiments are conducted using five benchmarking datasets which include CIFAR-10[3] and CIFAR-100[4], Fashion-MNIST[4], SVHN[5] and Imagenet[6] datasets. First, experimental settings for the different datasets and models, including hyper-parameters are discussed. Subsequently, results comparisons with the original highway network [207] and state-of-the-art models [67, 101, 202, 69, 77] are given. Lastly, analysis of learned features for the proposed highway blocks gates are presented, along with a high-level discussion that further unravels its manner of operation for successful learning.

The architecture of the most computationally efficient model in this chapter on the CIFAR, Fashion-MNIST and SVHN datasets, 19-layers HWCC, is shown in Table 4.1; where for convL-M(s×s), L denotes the convolution layer, M is the number of output feature maps and s is the size of convolution filter; Highway__convL-D(s×s) is a highway layer at layer L with D output feature maps; Comp.__convL-D is a compression layer at layer L with D output feature maps. For MP__N(k×k), N denotes the index of the max pooling layer and k is the size of the pooling window. For C-way softmax, C is the number of classes. For reporting experimental results, very deep networks learned using the proposed highway blocks with r×r convolu-

[3]https://www.cs.toronto.edu/ kriz/cifar.html
[4]https://github.com/zalandoresearch/fashion-mnist
[5]http://ufldl.stanford.edu/housenumbers/
[6]http://image-net.org/challenges/LSVRC/2012/index

Table 4.1: Proposed 19-layer HWCC architecture

19-layer HWCC architecture	
Output size	Input: 32×32×3
32×32	Conv1-64(3×3)
32×32	Conv2-64(1×1)
32×32	Conv3-64(3×3)
32×32	Conv4-96(1×1)
32×32	Conv5-96(1×1)
16×16	MP_1(2×2); stride=2
16×16	Conv6-96(3×3)
16×16	Conv7-96(1×1)
16×16	Conv8-128(1×1)
16×16	Conv9-192(3×3)
7×7	MP_2(3×3); stride=2
32×32	conv10-128(1×1)
32×32	conv11-128(3×3)
32×32	Conv12-192(3×3)
7×7	Conv13-128(1×1)
7×7	Conv14-128(3×3)
7×7	Conv15-192(3×3)
7×7	Highway_conv16-192(1×1)
7×7	Comp._conv17-128(2×2)
7×7	Conv18-256(1×1)
2×2	MP_3(3×3); stride=3
1×1	2×2 global average pool
C-way softmax	

tion filters for compression is referred to as 'HWCC'. For the imagenet dataset, we train the HWGC and HWCC models with 34 layers and roughly 43M parameters. In addition, deeper HWGC and HWCC models with 50 layers and roughly 59M parameters are trained.

4.5.1 Model training settings and evaluation criteria

Model training settings

For HWGC and HWCC, 19 and 32-layers models are constructed. A similar number of model parameters is used for models with similar depth. Note that compared with the original highway network [207] that employ gates for every layer to alleviate training difficulty,

the proposed 19-layers HWGC and HWCC use one gate, while HWGC and HWCC with 32-layers use three gates for successful training. Specifically, gates are employed every 8-13 layers for routing gated untransformed features to alleviate the difficulty of optimization. It is observed from parallel experiments that using a gate for every layer significantly increases the overall number of model parameters, and without any notable improvement in generalization results. The gating units' biases of all models are initialized to a value of 3 at the start of training. The gates re-mapping hyperparameter, n, is set to a value of 0.1; this value is found to give good results on all datasets in this chapter. Ablation studies for the hyperparameter, n, in relation to training convergence rates are presented later on in Section 4.5.1. For the compression of the output of the proposed highway blocks, 2$\times$2 convolutional filters are used; the filter size is chosen solely from the perspective of computational efficiency, and thus other filter sizes can be used. A max-norm value of 3 was found suitable for constraining the parameters of gating units. For improving model generalization, units' dropout in the range 5% to 50% are applied in the different hidden layers of the constructed models; dropout is not applied to the gating units. All models are trained using mini-batch gradient decent optimization algorithm with a batch size of 128. A starting learning rate of 0.1 is used; the learning rate is annealed over training to a final value of 10^{-5}. For pre-processing all datasets used in this chapter, training and testing data are standardized using training data mean and standard deviation.

Evaluation criteria

- Test error rate: note that since our main aim is to demonstrate the effectiveness of the proposed highway blocks for learning very deep networks, we train models of moderate depth and parameters: one model of 19 layers with 1.7M parameters, and the other with 32 layers with 2.6M parameters. It is well known that the performance of deep models can be improved by significantly increasing model depth [67] or width [254]. Consequently, the results reported for the proposed models can be improved by learning much deeper or wider models. Particularly, the original highway network [207] is taken as the baseline model for performance comparison with HWGC and HWCC

proposed in this chapter.

- Training time: fast training of deep models is important for timely development and testing. Generally, it is desirable that the constructed models require reasonable training time. For comparison, we consider the proposed highway networks against the original highway network [207], ResNet [67] and DenseNet [77].

- Inference time per frame: in many real-life applications, very low latency for inference is of utmost importance to support real-time applications. Therefore, inference time for the proposed highway network in this chapter is evaluated against original highway network [207], ResNet [67] and DenseNet [77] which give competitive results on the different datasets.

- GPU memory requirement: typically, minimal GPU memory resource is favoured for training and testing deep neural models. Especially, considering that these models are often deployed for various applications on portable electronic devices with tight GPU memory budgets. As such, the required GPU memory for the proposed highway network in this chapter is evaluated against original highway network [207], ResNet [67] and DenseNet [77] which give competitive results on the different datasets.

4.5.2 CIFAR-10 and CIFAR-100 datasets

CIFAR-10 dataset contains natural RGB images of size 32$\times$32 pixels belonging to 10 different object classes. There are 50,000 and 10,000 samples for training and testing respectively. Experimental results for the dataset without and with data augmentation are given in Table 4.2 as C10 and C10+, respectively. For data augmentation, we follow the common protocol [207, 67, 254] of only random horizontal flipping, translation by 4 pixels and reflection. The proposed models, HWGC and HWCC, outperform based on achieved test error rates the baseline model, original highway network [207]; see Table 4.2.
The learning curves showing training and testing error rates without and with data augmentation for HWCC with 19-layers (i.e. HWCC-19) is shown in Figure 4.6. HWCC-19 achieves

Table 4.2: Test error rate (%) on CIFAR-10 and CIFAR-100 dataset

Models	Param.	Depth	C10	C10+	C100	C100+
Network in network [119]	1.3M	10	10.41	8.81	35.68	–
Maxout [58]	>6M	5	11.68	9.38	38.57	–
Recurrent CNN [118]	1.86M	–	8.69	7.09	31.75	–
Deeply supervised [107]	1.3M	10	9.69	7.97	–	34.57
All-CNN [202]	1.3M	8	9.08	7.25	–	33.71
ResNet [67]	1.7M	110	13.63	6.41	44.76	27.22
ResNet [69]	1.7M	164	11.26	5.46	35.58	24.33
ResNet [69]	10.2M	1202	10.56*	4.92	33.47	22.71
Wide ResNet [254]	36.5M	28	–	4.17	–	20.50
Wide ResNet [254]	2.2M	40	–	5.33	–	26.04
Weighted ResNet [189]	19.1M	1192	–	5.10	–	–
FractalNet [101]	38.6M	21	7.33	4.60	29.05	23.73
DiracNet [253]	59M	34	–	4.54	–	20.96
DiracNet [253]	3.7M	34	–	5.60	–	26.72
DenseNet (k=24) [77]	1.0M	40	7.00	5.24	27.55	24.42
DenseNet (k=24) [77]	27.2M	100	5.83	3.74	23.42	19.25
Baseline: Highway net [207]	∼2.3M	19	–	7.54	–	32.24
Proposed: HWGC-19*	1.7M	19	**8.44**	**5.30**	**29.81**	**24.31**
Proposed: HWGC-32*	2.6M	32	**8.21**	**5.26**	**29.26**	**24.18**
Proposed: HWCC-19*	1.7M	19	**8.06**	**5.22**	**29.41**	**24.26**
Proposed: HWCC-32*	2.6M	32	**7.96**	**5.14**	**28.77**	**24.02**

*Fair comparison is with models that employ similar number of parameters

error rates of 8.06% and 5.22% without and with data augmentation, respectively. In addition, 19-layers HWGC-19 achieves error rates of 8.44% and 5.30% without and with data augmentation, respectively. We outperform the baseline model, highway network, ResNet and many state-of-the-art results as given in Table 4.2. It is noted that though DenseNet [77] achieved lower error rates, without a very careful implementation [209], it has extremely high GPU (Graphics Processing Unit) memory requirements [165]. Also important is that the original highway network [207] requires 400 epochs for training, while HWGC and HWCC require a maximum of 250 and 200 epochs for training, respectively to converge to better test errors. This translates to 37.5% and 50% reduction in the number of training epochs for the original highway network [207], respectively.

CIFAR-100 dataset contains natural RGB images of size 32×32 pixels belonging to 100 different object classes. There are 50,000 and 10,000 samples for training and testing respectively. Experimental results for the dataset without and with data augmentation are given in Table 4.2 as C10 and C10+, respectively. For data augmentation, we follow the common

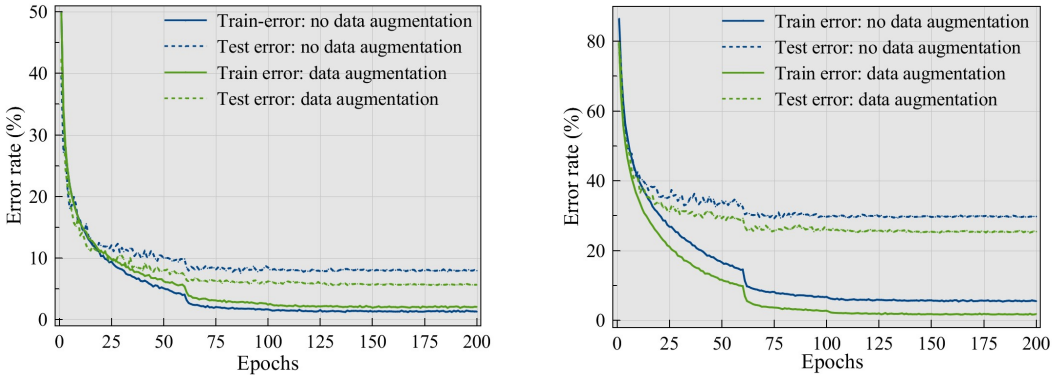


Figure 4.6: Error rate for HWCC-19. Left: CIFAR-10 dataset. Right: CIFAR-100 dataset

protocol [207, 67, 254] of only random horizontal flipping, translation by 4 pixels and reflection. Again, HWCC and HWGC outperform based on achieved test error rates the baseline model, original highway network [207]; see Table 4.2. The learning curves for HWCC-19 showing error rates on the training and testing data without and with data augmentation are shown in Figure 4.6. HWCC-19 achieves error rates of 29.41% and 24.26% without and with data augmentation, respectively. In addition, HWGC-19 achieves error rates of 29.81% and 24.31% without and with data augmentation, respectively. Furthermore, ResNet and many state-of-the-art results are outperformed as given in Table 4.2. DenseNet [77] reported lower error rates, but without a very careful implementation, the problem of enormous GPU memory requirement is again a huge concern [165]. Again, it is important to note that the original highway network [207] requires 400 epochs for training, while HWGC and HWCC require a maximum of 300 and 200 epochs for training, respectively. This translates to 25% and 50% reduction in the number of training epochs for the original highway network [207], respectively.

4.5.3 Fashion-MNIST dataset

Fashion-MNIST dataset is a recently collected dataset for benchmarking learning algorithms. The motivation for the collection of the dataset follows from the fact that the popular MNIST handwritten digits dataset is now considered trivial to learn as several works have reported test error rates in the range 0.21-0.50% in recent times. Fashion-MNIST dataset is simi-

Table 4.3: Test error rate (%) on Fashion-MNIST dataset

Models	Param.	Depth	Fashion-MNIST
2C1P2F+Drouout [209]	3.27M	–	8.40
3C1P2F+Dropout [209]	7.14M	–	7.40
GoogleNet [209]	101M	22	6.30
AlexNet [209]	60M	8	10.10
SqueezeNet-200 [209]	0.5M	18	10.00
VGG16 [209]	26M	16	6.50
EvoCNN [209]	6.68M	–	5.47
ResNet [67]	1.7M	110	7.16*
Baseline: highway network [207]	1.7M	19	6.17*
Proposed: HWGC-19	1.7M	19	**5.26**
Proposed: HWCC-19	1.7M	19	**4.87**

*Experiment ran by us

lar to MNIST dataset. The dataset contains 50,000 and 10,000 training and testing samples, respectively. However, instead of handwritten digits in MNIST dataset, Fashion-MNIST dataset contains images of fashion outfits belonging to 10 different classes that include T-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. The images are grayscale and of size 28$\times$28 pixels. Fashion-MNIST dataset is considered to be simpler to learn as compared to CIFAR-10 and CIFAR-100 datasets, therefore we train models of smaller depth; that is, only 19-layers HWGC and HWCC. Both HWGC-19 and HWCC-19 are trained for 150 epochs and without data augmentation; obtained test error rates are given in Table 4.3, along with state-of-the-art results that rely on much more model parameters or depth. It can be seen that HWCC-19 achieves an error rate of 4.87% which, to our knowledge, is the best result reported so far on Fashion-MNIST dataset without data augmentation. The learning curves for HWCC-19 showing error rates on the training and testing data without data augmentation are shown in Figure 4.7.

4.5.4 SVHN dataset

SVHN (Street View House Number) dataset contains real-life Google street view images with digits in the range 0-9; that is, the images belong to 10 different classes. There are 73,257 and 26,032 digits for training and testing, respectively. In addition, there are 531,131 somewhat easier digits that can be used as extra training data. As with standard training

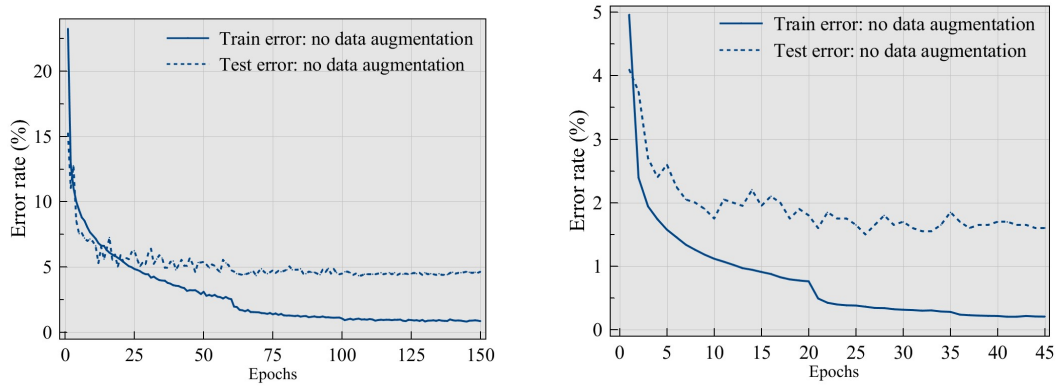


Figure 4.7: Error rate for HWCC-19. Left: Fashion-MNIST dataset. Right: SVHN dataset

Table 4.4: Test error rate (%) on SVHN dataset

Models	Param.	Depth	SVHN
Network in network [119]	1.3M	10	2.35
Maxout [58]	–	5	2.47
Recurrent CNN [118]	1.86M	–	1.80
Recurrent CNN [118]	2.67M	–	1.77
Deeply supervised [107]	1.3M	10	1.92
FractalNet [101]	38.6M	21	1.87
ResNet [67] (in [77])	1.7M	110	2.01
DenseNet (k=12) [77]	1.0M	40	1.79
DenseNet (k=24) [77]	27.2M	100	1.59
Baseline: highway network [207]	1.7M	19	2.22*
Proposed: HWGC-19	1.7M	19	**1.75**
Proposed: HWCC-19	1.7M	19	**1.71**

*Experiment ran by us

and testing protocol [58], the training and extra training data altogether are used for model training and hyper-parameter selection. For model performance evaluation, the testing data is used. Again, the dataset is considered easier to learn, therefore only 19-layers HWGC and HWCC models are trained for the classification problem. Error rates on the testing data are shown in Table 4.4. The learning curves for HWCC-19 showing the error rates on the training and testing data without data augmentation is shown in Figure 4.7. The proposed 19-layers HWGC and HWCC with an error rates of 1.75% and 1.71% respectively outperform the original highway network [207], ResNet with 110 layers [67] and other state-of-the-art models based on similar number of model parameters.

4.5.5 Imagenet-2012 (ILSVRC) dataset

The imagenet dataset contains natural images of 1.2M samples belonging to 1000 different classes. The dataset contains 50,000 natural images for testing. All images are resized to 224×224 pixels for training. The images are augmented during training using random horizontal flips. For testing, the images are first resized to 256×256, and then center crops of 224×224 pixels are fed to the trained models. We train the HWCC and HWGC models both with 34 and 50 layers on the imagenet dataset for 120 epochs. The initial learning rate, which is set to 0.1 is annealed during training to the final value of 10^{-3}. The momentum rate is set to 0.9, while the weight decay penalty is set to 10^{-4}; a training batch size of 64 is used. The training and testing errors per epoch over the training and testing data are not computed during training, since it considerably increases the training time by several hours; the models are simply tested at the end of training.

The top-1 error rates obtained from the trained models are reported in Table 4.5, where the HWGC and HWCC models outperform the baseline model, the original highway network. Considering models with similar number of parameters, the proposed models with 34 layers slightly outperforms the ResNet-101, which is roughly three times deeper. Similarly, the proposed models with 50 layers surpass the ResNet with 152 layers that is roughly three times deeper. We note that DenseNet-161 with fewer parameters slightly outperforms the proposed models with 34 layers. However, we show later on in Section 4.5.6 and Section 4.5.7 that the DenseNet requires more training/testing time and GPU memory, respectively; these are crucial limitations on the practical applicability of the DenseNet. Overall, these observations altogether strengthen the proposed reformulations of the original highway network for improved learning.

4.5.6 Training and inference time

Training time: As earlier discussed in Section 4.5.1, aside obtained error rates, other evaluation criteria that are of interest in this chapter such as training and inference time are observed. Table 4.6 shows the required time per epoch on CIFAR-10 dataset for models

Table 4.5: Test error rate (%) on imagenet-2012 (ILSVRC) dataset

Models	Param.	Depth	ILSVRC
ResNet [67]	44.5M	101	22.56
ResNet [67]	60.2M	152	21.57
DenseNet (k=48) [77]	30.6M	161	22.33
Baseline: highway network [207]	43.6M	34	23.45*
Baseline: highway network [207]	59.3M	50	22.72*
Proposed: HWGC-34	43.6M	34	**22.46**
Proposed: HWGC-50	58.8M	50	**21.31**
Proposed: HWCC-34	43.1M	34	**22.17**
Proposed: HWCC-50	58.3M	50	**20.78**

*Experiment ran by us

Table 4.6: Training time per epoch on CIFAR-10 dataset

Models	Param.	Depth	Time (s)*
ResNet [69]	1.7M	164	151
ResNet [67]	1.7M	110	106
DenseNet [77]	1.0M	40	175
Baseline: highway network [207]	1.7M	19	55
Proposed: HWGC-19	1.7M	19	**52**
Proposed: HWCC-19	1.7M	19	**48**

*Result with a batch size of 128 on a single Nvidia GTX Titan X GPU

Table 4.7: Inference time per frame for CIFAR-10 dataset

Models	Param.	Depth	Time (ms)*
ResNet [69]	1.7M	164	1.56
ResNet [67]	1.7M	110	1.21
DenseNet [77]	1.0M	40	0.69
Baseline: highway network [207]	1.7M	19	0.41
Proposed: HWGC-19	1.7M	19	**0.39**
Proposed: HWCC-19	1.7M	19	**0.35**

*Result with a single Nvidia GTX Titan X GPU

with similar number of model parameters; a training batch-size of 128 samples is used for all models. For comparison, we select the original highway network [207] (considering number of parameters and depth), ResNet [67, 69] and DenseNet [77] models that give similar results with the proposed highway networks on CIFAR-10 and CIFAR-100 datasets; see Table 4.6. DenseNet has the highest training time of 175 secs per epoch, which is expected considering its complicated architecture as every layer's output is connected to all other layers. The ResNet-164 and ResNet-110 require lesser training times of 151 secs and 106

Table 4.8: Inference time per frame for imagenet dataset

Models	Param.	Depth	Time (ms)*
ResNet [69]	60.2M	152	49.53
DenseNet (k=48) [77]	30.6M	161	55.63
Baseline: highway network [207]	59.3M	50	34.61
Proposed: HWGC-50	58.8M	50	**34.55**
Proposed: HWCC-50	58.3M	50	**34.79**

*Result with two Nvidia Titan V GPUs

secs per epoch, respectively. HWGC-19 and HWCC-19 require training times of 52 and 48 secs per epoch respectively, while the original highway network [207] requires 55 secs. The original highway network [207] is slightly slower to train than HWGC and HWCC; this may be attributed to the fact it uses highway blocks (i.e. two parallel paths) for every layer, while HWGC-19 and HWCC-19 uses only one highway block. However, HWGC-19 and HWCC-19 require 250 and 200 training epochs respectively, and as such have overall smaller training times than the original highway network [207] which requires 400 training epochs.

Inference time: For inference time evaluation, the time to perform inference on the entire CIFAR-10 dataset is reported in Table 4.7. It can be seen that HWCC-19 offers a significant reduction in inference time compared to ResNet [67, 69] and DenseNet [77], while at the same time matching or even outperforming (on achieved test error rates) much deeper ResNet models. HWGC-19 and HWCC-19 have inference times that are comparable to the original highway network [207]. However, note that the original highway network, ResNet-110 and ResNet-164 achieved higher test errors than the proposed highway networks; see Tables 4.2, 4.3, & 4.4. The interesting results of the proposed highway model can be attributed to the fact that it makes more efficient use of model depth (smaller depth results in improved results) following the analysis and construction given in Section 4.4. From Table 4.7, HWGC-19 and HWCC-19 require inference times of 0.39 and 0.35 secs respectively, while the original highway network requires 0.41 secs (i.e. comparable to the proposed highway networks), DenseNet requires 0.69 secs (i.e. 97.1% increase in inference time from HWCC-19), ResNet-164 requires 1.56 secs (i.e. 345.7% increase in inference time from HGCC-19) and ResNet-110 requires 1.21 secs (i.e. 245.7% increase in inference time).

Furthermore, the inference times per frame for the best models trained on the imagenet dataset in Table 4.5 are reported in Table 4.8, where it is seen that the proposed models, HWGC and HWCC, require considerably smaller inference time than the significantly deeper ResNet and DenseNet.

It can be observed that the much deeper ResNets have limited use for supporting applications that require fast inference. The proposed model requires an inference time that is similar to those for the original highway network [207] and highway network with gates constraints [158], however with better generalization performance. We acknowledge that DenseNets require fewer parameters but increased depth to achieve similar results for the proposed model. Unfortunately, DenseNets incur much higher inference time and without a careful implementation require enormous GPU memory resources. Moreover, a careful implementation of DenseNet to reduce GPU memory footprint increases overhead computation cost, and as such further increases training and inference time [165]. This can limit its usefulness for many practical applications that require low latency for operation.

4.5.7 GPU memory requirement

The GPU memory requirement for training deep neural networks is a very critical attribute of a *truly* successful model. The goal is to build deep neural networks that require modest GPU memory. Consequently, using the CIFAR-10 dataset, the evaluation of the proposed models against the original highway network, ResNet-110, ResNet-164 and DenseNet are given in Table 4.9. It will be seen that the proposed models, HWGC-19 and HWCC-19, have similar requirements to the original highway network, while ResNet and DenseNet clearly have much higher requirements. Particularly, observe that DenseNet with 40 layers and only 1M parameters has a higher GPU memory requirement than even ResNet with 110 layers and 1.7M parameters. Furthermore, Table 4.10 shows the GPU memory requirements for the models trained on the imagenet dataset. Keeping in mind from Table 4.5 that the proposed models (i.e. HWGC and HWCC) can match or outperform the ResNet and DenseNet, Table 4.10 shows that the proposed models require smaller GPU memory than the ResNet and DenseNet. Ultimately, Table 4.9 and Table 4.10 results motivate the computational efficiency

Table 4.9: GPU memory resource for CIFAR-10 dataset

Models	Param.	Depth	GPU (GB)*
ResNet [69]	1.7M	164	3.94
ResNet [67]	1.7M	110	1.76
DenseNet [77]	1.0M	40	2.36
Baseline: highway network [207]	1.7M	19	0.98
Proposed: HWGC-19	1.7M	19	**1.00**
Proposed: HWCC-19	1.7M	19	**1.06**

*Result with a batch size of 128

Table 4.10: GPU memory resource for Imagenet dataset

Models	Param.	Depth	GPU (GB)*
ResNet [69]	44.5M	101	14.57
ResNet [67]	60.2M	152	29.92
DenseNet (k=48) [77]	30.6M	161	16.29
Baseline: highway network [207]	43.6M	34	11.65
Baseline: highway network [207]	59.3M	50	25.23
Proposed: HWGC-34	43.6M	34	**11.33**
Proposed: HWCC-50	58.8M	50	**25.41**
Proposed: HWGC-34	43.1M	34	**11.79**
Proposed: HWCC-50	58.3M	50	**25.96**

*Result with a batch size of 64

of the proposed models in comparison to others.

4.5.8 Qualitative evaluation of gating units activations

For qualitative evaluation and analysis, the activations (response) of gating units in HWCC to the different image classes are observed. Specifically, gating units in the first highway network block of HWCC-19 model are observed; similar results are obtained for HWCC with 32 layers, and HWGC with 19 and 32-layers. The major motivation for this is to verify that the gating units indeed perform some sort of feature filtering; this is a nice learning characteristic as earlier stated in Section 4.3.2. Some of the gates' convolution maps in the first highway block for randomly chosen images for the 10 different classes in CIFAR-10 dataset are shown as grayscale images in Figure 4.8. It is straightforward to observe from the convolution maps which appear as binary *discriminative masks* for important parts in the

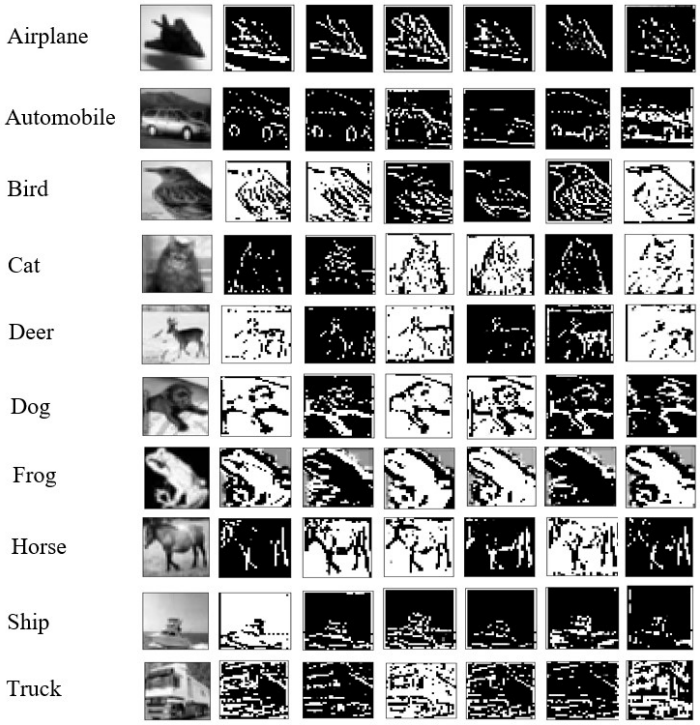


Figure 4.8: Gating units' activations (responses) for the first highway block in HWCC-19 trained on CIFAR-10 dataset. The first column shows the input images for the 10 different classes; other columns show units' activations for different convolution maps

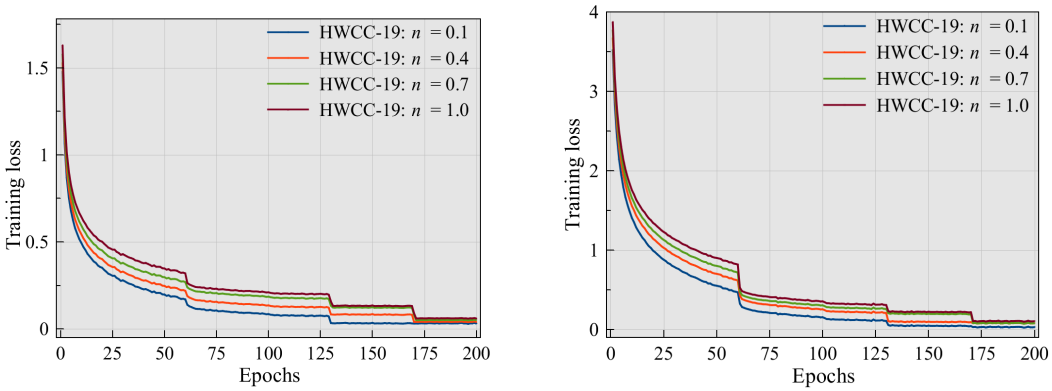


Figure 4.9: Convergence rate for HWCC-19. Left: CIFAR-10 dataset. Right: CIFAR-100 dataset

different images that the gating units actually perform some sort of feature filtering that may make learning more interesting than for ResNet which lacks this attribute.

4.5.9 Ablation studies

For further validation of the reformulated gating blocks proposed in this chapter, we perform ablation studies to study how the gate remapping hyperparameter, n, presented in Section 4.4.1 impacts the rate of training convergence for the proposed models. Figure 4.9 shows the training loss curve for different values of n using HWCC-19 on CIFAR-10 and CIFAR-100 datasets, respectively; similar observations are made for the HWGC models. It is seen for both datasets that training converges faster for smaller values of n as expected and discussed in Section 4.4.1.

4.6 Conclusion

Very deep neural networks have been shown empirically and theoretically to have better function approximating capacity as compared to shallow networks. However, training very deep networks is hard especially from an optimization perspective. This chapter proposes two highway networks with gate constraints, HWGC and HWCC, for improving the training of very deep networks. The capability to control learning nearly exact identity functions by a clever construction and formulation of the proposed highway blocks is shown analytically and experimentally to allow a more effective usage of model depth than the original highway network. As such, the proposed models generalize better than original highway network and other state-of-the-art models.

Furthermore, experimental results using CIFAR-10, CIFAR-100, Fashion-MNIST, SVHN and imagenet datasets show that using the same number of model parameters, the proposed HWGC and HWCC models with 19-layers mostly outperform ResNets with 110 and 164 layers on all the benchmarking datasets used in this chapter; many state-of-the-art models are also outperformed using the proposed model. In addition, experimental results using the imagenet dataset show that the proposed HWGC and HWCC, with 50 layers can match or outperform the ResNet with 101 and 152 layers. From this observation, we conclude that HWGC and HWCC models are more computationally and memory efficient than the original highway network and ResNet, since similar or better test errors can be achieved with a

significantly smaller depth that requires much smaller training time, and especially inference time and GPU memory for supporting real-time applications.

Chapter 5

Training Very Deep Neural Network without Skip Connections

The different approaches in Chapter 3 and Chapter 4 for improving the optimization of very DNNs all rely on some form of skip connections that complicate their operation. In this Chapter, we propose the training of very deep networks without shortcut connections, considering that their operations are easy to understand; these networks are referred to as very deep PlainNets. However, training very deep PlainNets is a notoriously hard problem due to: (1) the relatively popular challenge of 'vanishing and exploding activations', and (2) the less studied 'near singularity'problem. We argue that if the aforementioned problems are tackled together, the training of deeper PlainNets becomes easier. Subsequently, we propose the training of very deep PlainNets by leveraging Leaky Rectified Linear Units (LReLUs), parameter constraint and strategic parameter initialization. Our approach allows to successfully train very deep PlainNets having up to 100 layers without employing shortcut connections. To validate this approach, we first perform a preliminary experimental investigation using COIL-20, USPS and MNIST datasets. Second, we validate on four challenging datasets; namely, CIFAR-10, CIFAR-100, SVHN and ImageNet datasets. We report the best results known on the ImageNet dataset using a PlainNet with top-1 and top-5 error rates of 24% and 7.3%, respectively.

5.1 Introduction

Deep neural networks are increasingly finding usefulness in diverse scientific fields. One can attribute their success to their capacity for approximating highly complex target functions. The importance of depth for learning some classes of functions is studied in [15, 40]. However, training deep plain networks, i.e., PlainNets, with over 15 layers is still hard [207, 67]. It has been posited that the poor test error rates obtained for very deep PlainNets arise from poor model optimization, and not overfitting [67]; at only 24-layers, it is shown in [7] that error gradients are already quite erratic and unfavourable for training. For a successful training, very deep networks, with over 15 hidden layers, typically rely on shortcut or skip connections from lower to higher layers in the model, such as residual [67] and highway [207] networks, along with their variants [207, 67]. Such networks have produced remarkable results for learning different hard tasks. However, a satisfactory account of their operation is still lacking [221, 7]. For example, [221] shows that residual networks (ResNets) simply rely on the shortcut connections as shorter paths through the network for propagating error gradients and therefore do not directly address the problem of vanishing or exploding gradients. Moreover, the classical-hierarchical feature representation concept does not strictly apply to ResNet as it has been shown that the effective depth is much smaller than the topological depth, e.g., 17 layers for a 110 layer ResNet [221]. This, for instance, can obfuscate feature transfer as feature representation level may become unclear.

We note the proliferation of works [101, 216, 211, 259] on deep networks with various forms of shortcut connections. Unfortunately, a concrete explanation of their learning characteristics remains elusive to the deep learning community. As such, the unconventional objective that we pursue in this chapter is an approach for the successful training of very deep PlainNets with competitive results, given that we understand their operation. A major motivation for this is that for several decades, these classical models with strictly hierarchical representations have been studied, and there are abundant theoretical and empirical works for understanding their operation. Moreover, being able to train very deep PlainNets would allow to study their representational capacity and performance gain. Figure 5.1 shows

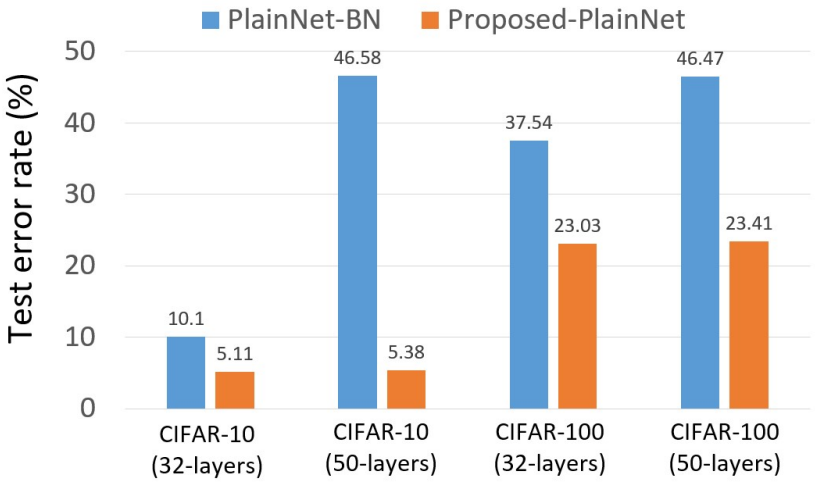


Figure 5.1: Error rates for very deep PlainNets. PlainNet-BN: PlainNet trained with only batch normalization; Proposed-PlainNet: PlainNet proposed in this chapter. PlainNet-BN is hard to optimize; batch normalization is insufficient for successful training

that the performance of very deep PlainNet trained with only batch normalization, PlainNet-BN, is unsatisfactory, especially as the model becomes deeper. Interestingly, our proposal, Proposed-PlainNet, maintains similar test error rates with an increase in model depth. Our major contributions are as follows:

1. Deeper insights for two fundamental problems that make the training of very deep PlainNets difficult: (i) *vanishing and exploding gradients* which result from units activations that decrease or increase exponentially from the output layer to the input layer (ii) *near singularity*, where model units collapse into one another and become unidentifiable leading to degenerate solutions for which gradient descent-based optimization is impotent.

2. A solution that is simple, comprehensible and implementable in most deep learning libraries without hassles. The solution is demonstrated extensively on three simple datasets (i.e. COIL20, USPS & MNIST) and four benchmarking datasets (i.e. CIFAR-10, CIFAR-100, SHVN & ImageNet). We report the best results in literature for PlainNets on all benchmarking datasets.

The rest of this chapter is organized as follows. In Section 5.2, we discuss related works.

Section 5.3 gives detailed investigation as background on the problem of training very deep PlainNets. In Section 5.4, we provide the details of the proposed approach, along with preliminary experimental results. Main experimental results and discussions are covered in Section 5.5. The conclusion is given in Section 5.6.

5.2 Related work

Vanishing and exploding gradients were identified as major problems for training recurrent neural networks [13]. Recently, they have been investigated for very deep PlainNets [164]. Very good results were reported[207, 67], addressing the problem of training very deep networks. ResNet [67] employs shortcut connections for alleviating this problem, and [207] proposes gating mechanisms for routing mostly untransformed features through the model highway at the start of training. It has been suggested that inspecting the forward pass during the training could reveal a lot about their optimization condition [7, 164]. For example, [143] showed that the hierarchical structure of deep networks results in a number of critical points; these points grow with depth and may increase the difficulty of training. In [229], permutation symmetry in multilayer networks was investigated and revealed that it could result into learning near singularities [164, 229]. It is posited that in this training regime, the Fisher information matrix degenerates, leading to strange learning characteristics [229]. Recent works attempt to unravel the problem of training very deep PlainNets by taking inspiration from how the ResNet alleviates the problem[7, 164]. The problem of shattered gradients for very deep PlainNets where error gradients become uncorrelated and resemble white noise was studied in [164]; the authors showed that ResNet largely circumnavigates this problem using shortcut connections. The work in [220] relied on information-theoretic analysis for studying the problem of volume conservation that results in vanishing gradients. The DiracNet [253] with layer weights reparameterization was proposed as a very deep PlainNet. The reported results are promising; however, a careful inspection shows that the DiracNet does not qualify as a PlainNet. Indeed, it requires feeding untransformed features into higher layers just as the ResNet, and explicitly uses skip connections, with the difference in the

positioning of activation functions. Moreover, the same work [253] states that 'Dirac parameterization and ResNet differ only by the order of nonlinearities'. The work in [235] reported the training of PlainNets having up to 10,000 layers by using Delta-Orthogonal initialization. Although, all the models performed well on the training set, they generalize quite poorly. e.g. their 32-layer model on CIFAR-10 dataset with 17.8M parameters achieved a test error rate of 18%.

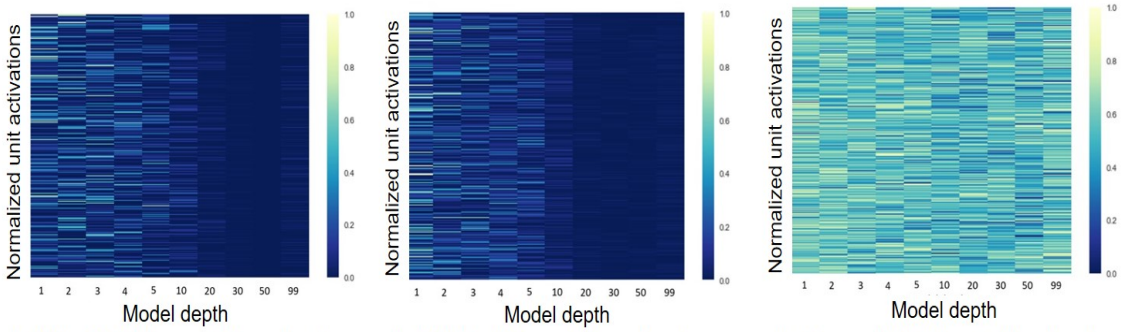
5.3 Background: the problem of training very deep PlainNets

In this section, the problems that plague the training of very deep PlainNets are investigated both analytically and experimentally; namely, learning in the regime of (i) *vanishing/exploding gradients* (ii) *near singularities*. In order to aid the discussion, we herein run preliminary investigation results on two simple datasets[1], i.e., COIL-20 [137] and USPS [81]. We consider two different training scenarios: (1) PlainNet without batch normalization, referred to as 'naive PlainNet'; and (2) PlainNet with batch normalization, referred to as 'PlainNet-BN'. The work in [162] showed that different initialization and batch normalization do not resolve the problem of training very deep PlainNets beyond a certain depth. The motivation for using simple and small datasets as an extreme scenario in the work was to emphasize that very deep PlainNets with millions of parameters could not even fit the training sets. Similarly, we train fully connected PlainNets on the COIL-20 and USPS datasets; the models are 100-layer PlainNets with 300 units in all the hidden layers. In the following sub-sections, we describe the learning characteristic of only the naive PlainNet and PlainNet-BN.

5.3.1 Vanishing and exploding gradients

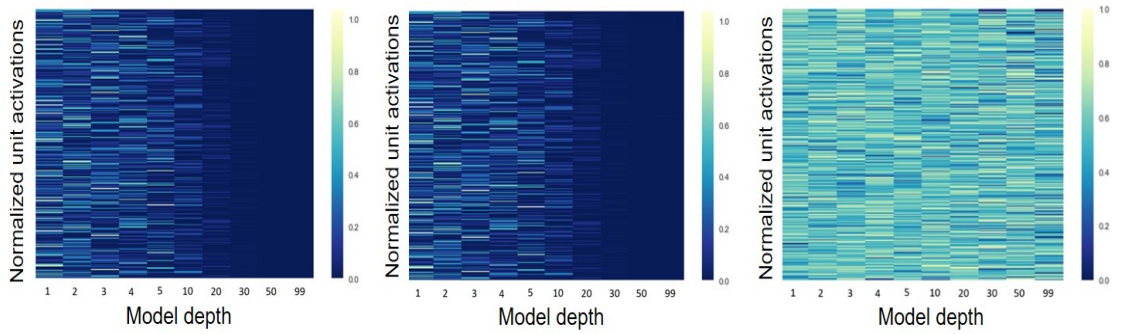
Vanishing gradients is essentially a problem with the units' activations that saturate in the forward-pass [13]. Model units that have very small or zero activations receive little or no error gradients for parameters update during the back-pass phase. Consider the transfor-

[1] Final results on more complex datasets are given in Section 5.5.



(a) Naive PlainNet: vanishing activations (b) PlainNet-BN: vanishing activations (c) Proposed PlainNet: stable activations

Figure 5.2: Normalized mean layer activations for a 100 layer PlainNet over COIL-20 dataset



(a) Naive PlainNet: vanishing activations (b) PlainNet-BN: vanishing activations (c) Proposed PlainNet: stable activations

Figure 5.3: Normalized mean layer activations for a 100 layer PlainNet over USPS dataset

mation learned at hidden layer l of a model for input $\boldsymbol{H}(\boldsymbol{x})^{l-1}$ as

$$\boldsymbol{H}(\boldsymbol{x})^{l} = \varphi(\boldsymbol{W}^{l}\boldsymbol{H}(\boldsymbol{x})^{l-1} + \boldsymbol{b}^{l}) = \varphi(\boldsymbol{Z}^{l}), \tag{5.1}$$

where φ is the activation function, $\boldsymbol{b}^l$ and $\boldsymbol{Z}^l$ are the bias and pre-activation for layer l, respectively. If φ saturates, then we may have a dead unit j with gradient $\nabla h_j(x)^l \approx 0$, that receives zero (or negligible) error gradients during the back-pass phase, and therefore neither learn nor allow the update of other units earlier in the model that are connected to it. Also, since ReLUs are unbounded in the upper end spectrum, moderate-valued activations can accumulate to result in extremely high activations for which resulting error gradients are very high [13, 164]; this can lead to *wild* weights update that make training unstable. Experimental investigation results discussed below support these observations.

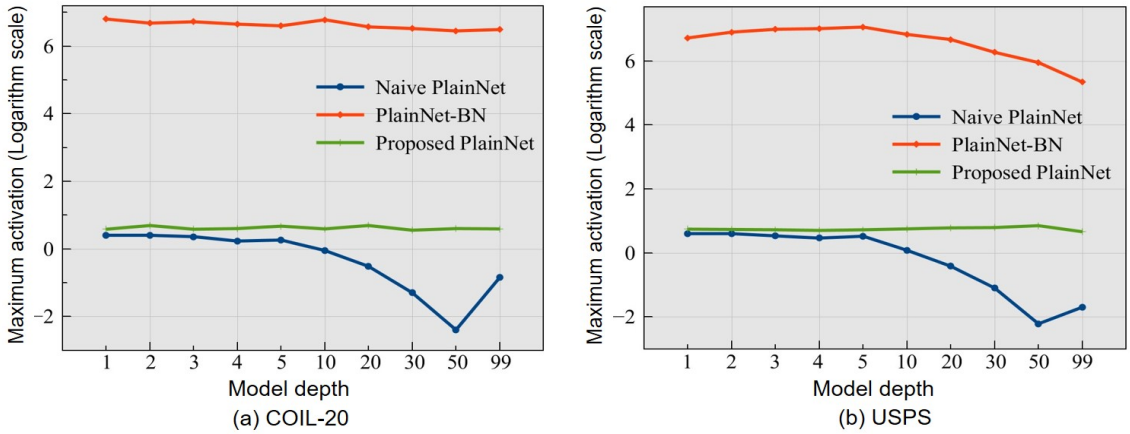


Figure 5.4: Maximum absolute units' activations with depth

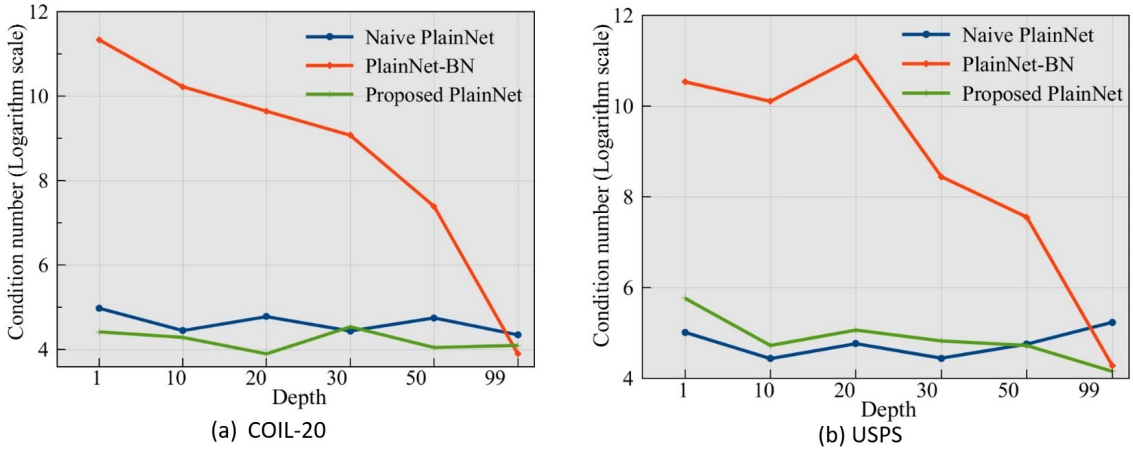
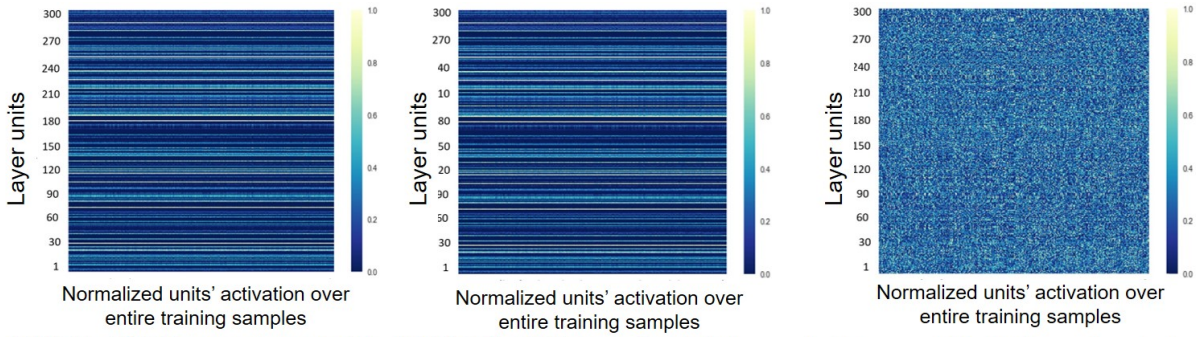


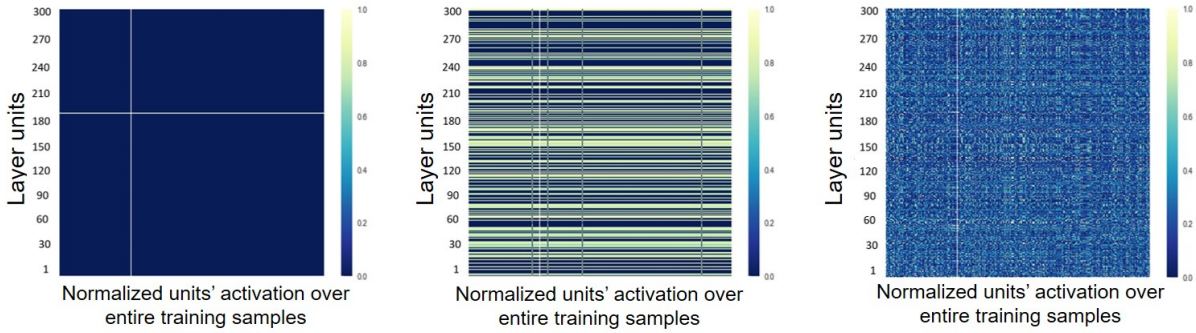
Figure 5.5: Infinity-norm based condition number for model weights

Figures 5.2 (a) & (b) show the normalized mean layer activations for the naive PlainNet and PlainNet-BN over the entire COIL-20 dataset. Figures 5.3 (a) & (b) show the normalized mean layer activations for the naive PlainNet and PlainNet-BN over the entire USPS dataset. Note that in order to readily capture the decay of units' activations with depth, global normalization is used as opposed to layer-wise normalization. It will be seen that for both naive PlainNet and PlainNet-BN, units' activations decrease with depth for both datasets. It can be observed that at about 20 layers, units' activations have considerably decayed. Importantly, it is emphasized that the PlainNet-BNs have extremely large activations, but global



(a) Naive PlainNet: near singularity (b) PlainNet-BN: near singularity (c) Proposed PlainNet: no singularity

Figure 5.6: 50th layer activations in a 100 layer PlainNet for the entire COIL-20 training set



(a) Naive PlainNet: near singularity (b) PlainNet-BN: near singularity (c) Proposed PlainNet: no singularity

Figure 5.7: 99th layer activations in a 100 layer PlainNet for the entire USPS training set

units' activations normalization obfuscate this fact. Figure 5.4 shows the maximum absolute units' activations for both the naive PlainNet and PlainNet-BN; the activations are plotted to Logarithmic scale since the PlainNet-BN units have very high activations. Also, note that the absolute activations are plotted, since the units in the proposed PlainNet that is discussed in Section 5.4 can attain negative values. It is observed that the naive PlainNet operates with reasonably small activations (in the range 0–10); however, units' activations decrease rapidly with depth. In contrast, the PlainNet-BN operate with extremely high activations (in the range 0–10^6). Ultimately, both scenarios result in difficulty of training. This reveals that though batch normalization speeds up the training of moderately deep PlainNets by preserving units' activations, using it naively for very deep PlainNets can plague training. It is shown in Section 5.4 that the proposed PlainNet in this chapter has interesting learning

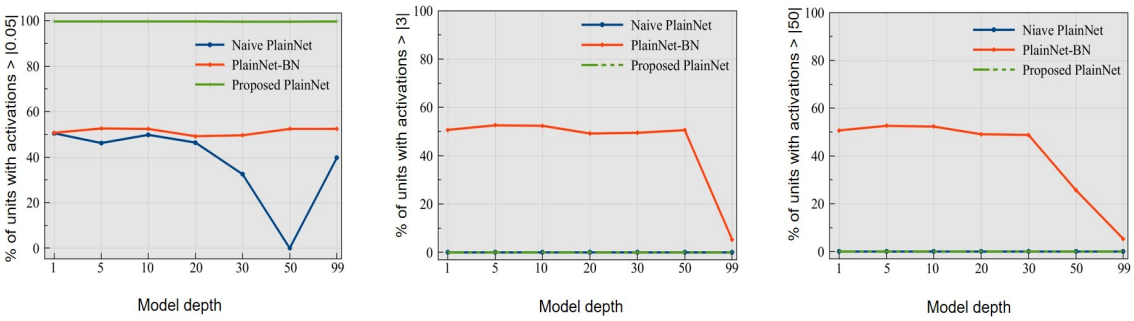


Figure 5.8: % of units' unnormalized absolute activations above threshold for the USPS dataset

characteristics such as moderate valued units' activations. It is a middle ground between the two extremities, naive PlainNet and Plain-BN.

The analytical investigation on units' activations is further substantiated by examining the weights of the trained very deep PlainNets. Specifically, we expect that the naive PlainNets would have moderate condition numbers, since gradients typically vanish, weights are mostly not updated and remain close to the initialized values. For the PlainNet-BN, we expect that the extremely high units' activations would cause large error gradients that would subsequently result in wild weights update and therefore very high condition numbers. Figures 5.5(a) & (b) show the condition numbers for the naive PlainNet and PlainNet-BN; the plots confirm earlier analysis and expectations.

5.3.2 Near singularities

Near singularities can result from excessive dead units (or units with very small activations) in a deep model [164]. This problem grows with increased model depth, since the specialty (or uniqueness) of the units vanishes and symmetry dominates. Symmetry is used to refer to a scenario where swapping two units does not change the output of the model [229]. Singularity implies that some of the units in a layer are linearly dependent or collinear; this can be quite severe for PlainNets. This means that the units respond in the same manner irrespective of the input received. Assume that $\boldsymbol{H}(\boldsymbol{x})^{l-1}$ is singular and $\boldsymbol{b}^l$ is initialized as zeros as it is typical, then the result of the transformation $\boldsymbol{H}(\boldsymbol{x})^l$ given in (1) is rank deficient

and collapses the space. Furthermore, it is easy to verify formally that learned transformation at a hypothetical layer L, $\boldsymbol{H}(\boldsymbol{x})^L : L \geq l$ is singular, since the product of a non-singular matrix and a singular matrix yields a singular transformation. Hence, subsequent layers in the model suffer singularity problems. Our experimental results show that this is indeed the case. Of course, perfect collinearity is not required for optimization to become difficult. Near singularity (i.e. considerable collinearity) is sufficient to render gradient descent an impotent solver, since the problem becomes ill-conditioned and lie in a degenerate solution space. Experimental investigation results discussed below support this position.

Figures 5.6 (a) & (b), and Figures 5.7(a) & (b) show units' activations over the entire COIL-20 and USPS datasets at 50th layer and 99th layer. For the COIL-20 dataset, it is observed that the units of the naive PlainNet have very small and essentially similar response over the entire dataset (i.e. training examples), showing the near singularity problem; see Figures 5.6 (a). The PlainNet-BN also exhibits near singularity, albeit with very high activations; see Figures 5.6 (b). For the USPS dataset, it is seen that the vanishing and exploding gradients, along with near singularity problems are more severe; see Figures 5.7 (a) & 7 (b). Figure 5.8 shows the percentage of units with activations above different thresholds after training on the USPS dataset. We note that the number of units with zero activations may be misleading in observing training difficulties, as the percentage of units with exactly zero activations is roughly 50% across all layers in our experiments. The work in [164] corroborates our argument that having many units with very small activations in very deep PlainNets encourages pseudo-linearity and therefore singularity. Our experiments reveal that training difficulty arises when the activations of many units are smaller than $|0.05|$. Figure 5.8 shows that the percentage of units' activations above $|0.05|$ begins to reduce rapidly from the 20th layer in the naive PlainNet, while many units in the PlainNet-BN maintain very high activations much deeper into the model, making training unstable.

5.4 Proposed approach: training very deep PlainNets

In order to solve the three challenges described in Section 5.3, we propose a new approach for training very deep PlainNets based on a modified PlainNet-BN. Our solution employs three components that are known in the literature but never been used together. In addition to batch normalization, the proposed PlainNet employs (1) LReLU, (2) max-norm constraint, and (3) strategic parameter initialization for successful training. Preliminary experimental results are given for validation, along with a high-level comparison of the operation of the proposed PlainNet with the ResNet.

5.4.1 Proposed PlainNet

We first start by presenting the individual key components of the proposed PlainNet approach, then give the formulation of the proposed solution, showing how the combination of its component enables a successful alleviation of the three targeted challenges.
(1) The LReLU at layer l outputs can be described as

$$\boldsymbol{H}(\boldsymbol{x})^l = 1(\boldsymbol{Z}^l \leq 0)(\beta \boldsymbol{Z}^l) + 1(\boldsymbol{Z}^l > 0)(\boldsymbol{Z}^l), \tag{5.2}$$

where $\boldsymbol{Z}^l$ is the pre-activation and β is a small specified scaling factor. The LReLU with $\beta = 0.01$ was proposed in [129] for alleviating dead units. We use LReLUs to tackle the near singularity problem that results from lots of dead units or units with very small activations. Also, LReLUs can alleviate the problem of exploding activations, since some units can attain negative values.
(2) The *max-norm* penalty imposes a constraint of the form

$$\| \overrightarrow{\boldsymbol{w}_j} \| \leq c, \tag{5.3}$$

where w_j is the weight vector of an arbitrary unit j and c is the specified maximum norm. The max-norm constraint is aimed at tackling activations explosion introduced by batch normalization; see Figures 5.4 & 5.5. The max-norm constraint is a natural companion for dropout regularization [206]; this serves as an additional motivation for its use.

(3) The weights of the proposed PlainNet are initialized from a uniform distribution

$$U[\sqrt{6/n_{in}^l}, -\sqrt{6/n_{in}^l}], \tag{5.4}$$

where n_{in}^l is the number of units feeding into layer l [68]. The layer biases, b^l, are initialized to small positive values. This initialization strategy is aimed at improving the diversity of model parameters at the start of training and therefore discourage units' symmetry.

Considering a PlainNet block, l, composed of two hidden layers indexed 1 and 2 that learns an overall transformation of the form

$$\boldsymbol{H}(\boldsymbol{x})^l = \varphi(\boldsymbol{W}_2^l(\varphi(\boldsymbol{W}_1^l \boldsymbol{H}(\boldsymbol{x})^{l-1} + \boldsymbol{b}_1^l)) + \boldsymbol{b}_2^l), \tag{5.5}$$

and let

$$\boldsymbol{Z}_1^l = \boldsymbol{W}_1^l \boldsymbol{H}(\boldsymbol{x})^{l-1} + \boldsymbol{b}_1^l; \tag{5.6}$$

so that we can write our solution in a more compact fashion as

$$\boldsymbol{H}(\boldsymbol{x})^l = \varphi(\boldsymbol{W}_2^l(\varphi(\boldsymbol{Z}_1^l)) + \boldsymbol{b}_2^l). \tag{5.7}$$

The condition of $\boldsymbol{Z}_1^l$ is critical for the avoidance of units' symmetry as training progresses. Let a hypothetical unit j in layer 1 have a pre-activation z_{1j}^l. In an extreme case, where a training batch produces many units $z_{1j}^l \ll 0$ in $\boldsymbol{Z}_1^l$, the proposed PlainNet transformation becomes

$$\boldsymbol{H}(\boldsymbol{x})^l \approx \varphi(\boldsymbol{W}_2^l(\beta(\boldsymbol{Z}_1^l)) + \boldsymbol{b}_2^l), \tag{5.8}$$

where β is the scaling factor for the LReLU. The fashion in which W_1^l and b_1^l are initialized mitigates symmetry, while the units' pre-activation in $\boldsymbol{Z}_1^l$ are scaled to ensure that units have non-zero outputs. In addition, $\boldsymbol{W}_1^l$, $\boldsymbol{W}_2^l$, $\boldsymbol{b}_1^l$ and $\boldsymbol{b}_2^l$ are constrained to have a max-norm of c to ensure that activations do not blow up. It will be seen that in the proposed PlainNet, units z_{1j}^l in $\boldsymbol{Z}_1^l$ receive error gradients and get reasonably updated.

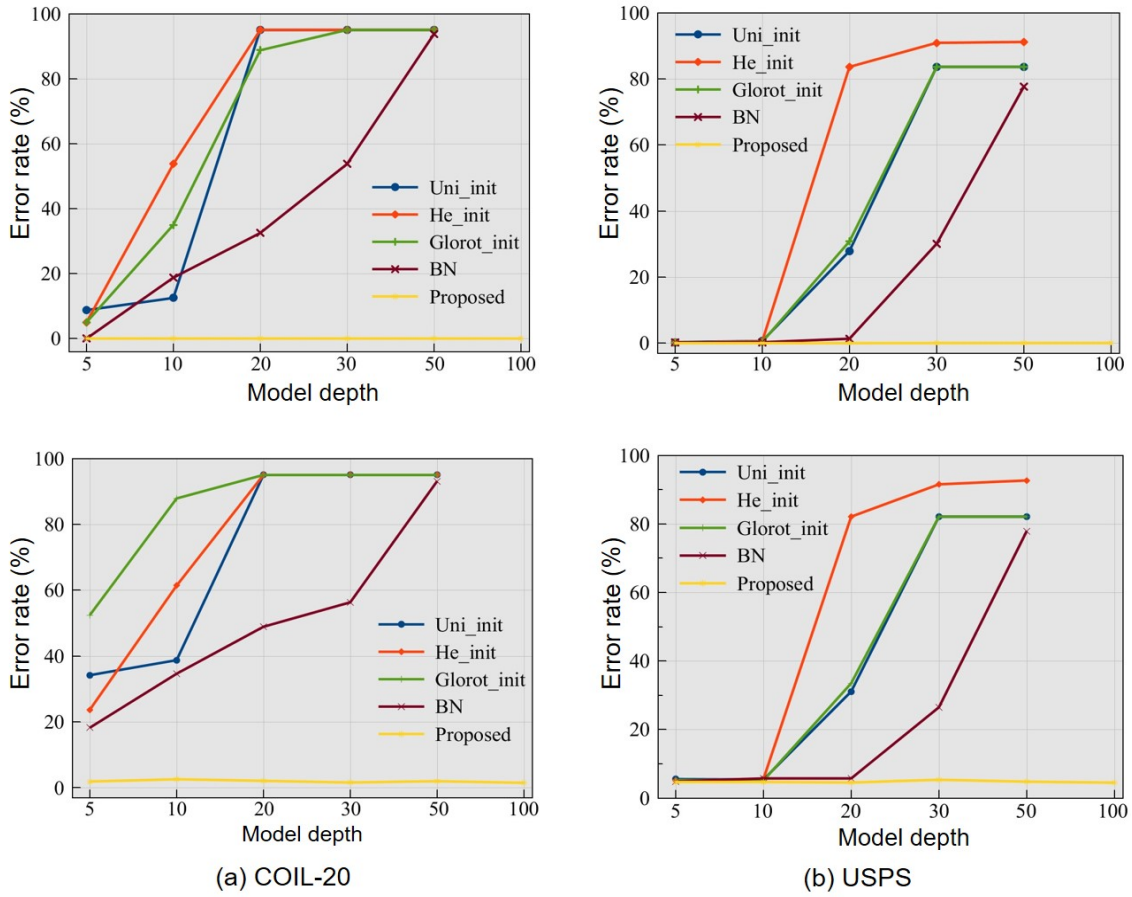


Figure 5.9: Top: PlainNet error rates on training set for COIL-20 (left) and USPS (right) datasets. Bottom: Error rates on test set for COIL-20 (left) and USPS (right) datasets [162]

5.4.2 Preliminary experiments

Experimental results show that setting $0.3 \leq \beta \leq 0.5$, $0.1 \leq \boldsymbol{b}^l \leq 0.8$ and $1 \leq c \leq 5$ for fully connected 100-layer PlainNets alleviates training difficulty without generalization loss. In Section 5.5, we show that similar training settings work for the proposed PlainNets with convolutions. It is emphasized that smaller values for c would necessitate lesser model regularization such as dropout or weight decay. Note that setting $\beta = 0$ recovers the ReLU. Importantly, it is observed that for the proposed PlainNet with $\beta \leq 0.5$, similar test errors can be obtained for ReLU-based models (PlainNet-BN) with 5 to 15 layers; beyond 15 layers, PlainNet-BNs show optimization problems. This suggests that the degree of non-linearity of

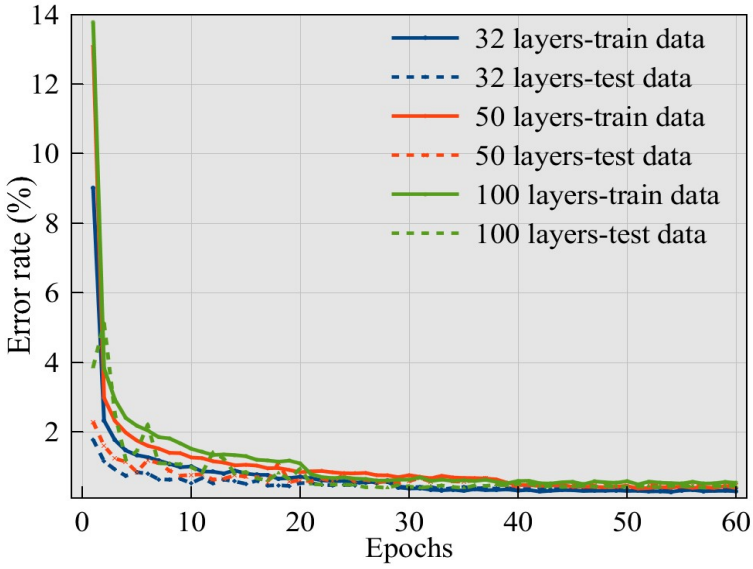


Figure 5.10: Error rates with proposed PlainNet using MNIST dataset

the ReLU in relation to expected generalization capacity may be currently over-estimated. Lastly, parallel experiments show that gradient clipping for tackling the wild updates from exploding activations did not ease training of the PlainNets.

As preliminary experiments, the proposed PlainNet is validated using the COIL-20 and USPS datasets as in Section 5.3. Figures 5.2(c) and 5.3(c) show that units' activations neither diminish nor explode, as against the naive PlainNet and PlainNet-BN. Figures 5.4 (a) & (b) show that the maximum absolute activations are within reasonable range, a compromise between the naive PlainNet and PlainNet-BN. Figures 5.5 (a) & (b) show that the weights of the proposed PlainNet indeed have moderate condition numbers, and therefore optimization is easier. Importantly, an examination of the weights of the proposed PlainNet after training shows that they have been reasonably updated, as against the naive PlainNet where weights were mostly not updated. Figures 5.6 (c) & 5.7 (c) show that the proposed PlainNet do not suffer near singularity problems; the units respond in different manners to the different training examples. Furthermore, it is observed for the proposed PlainNet on both COIL-20 and USPS datasets that every units' activation value is above $|0.05|$ and less than $|50|$. This constitute a reasonable middle ground between the naive PlainNet and PlainNet-BN as shown in

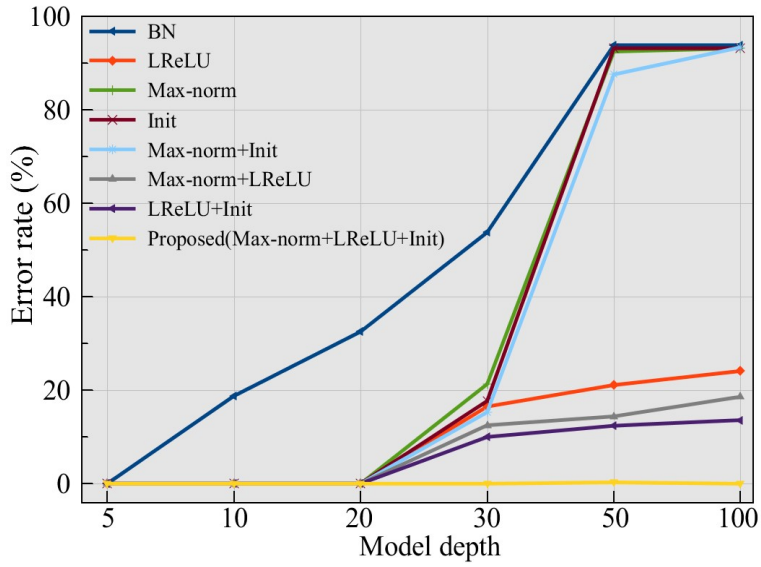


Figure 5.11: Test-error based ablation study of the different components for improving PlainNet training using COIL-20 dataset

Figure 5.8. Figures 5.9 (a) & (b) show that initialization schemes such as Uniform, He et al. (He_init) [68], Glorot et al. (Glorot_init) [56], and batch normalization (BN) [84] do not resolve training difficulty. However, the proposed PlainNet (i.e shown as proposed) is trainable even with 100 layers; both training and test errors remain small with depth increase. Lastly, only for sanity check, 50 and 100 layers convolutional PlainNets with 550k and 959K parameters respectively, were trained using our proposal for 60 epochs on MNIST dataset [104]; without data augmentation and much tweaking of model hyper-parameters, test errors of 0.33% and 0.36% were achieved respectively. Experimental details for training and testing errors for different model depths are given in Figure 5.10.

We perform ablation studies to investigate the contribution of each component (LReLUs, max-norm constraint, and strategic paramters initialization) for alleviating the difficulty of training. Specifically, one or two of the three components is (are) removed at a time in the proposed PlainNet; see Figure 5.11 for obtained training errors on COIL-20 dataset; all experiments use batch normalization, and plot denotations remain as mentioned in the manuscript. It is observed that using the three components as proposed gives the best results for alleviating the difficulty of training.

Table 5.1: Test error rate (%) on the CIFAR-10, CIFAR-100 and SVHN datasets

Models	Param.	Depth	C10	C10+	C100	C100+	SVHN
ResNet [67] (reported by [76]*)	1.7M	110	13.63*	6.41	44.76*	27.76	1.80*
ResNet [69] (reported by [77]*)	10.2M	1202	10.56*	4.92	33.47*	22.71	–
Wide ResNet [254]	36.5M	28	–	4.17	–	20.50	–
Wide ResNet [254]	2.7M	16	–	–	–	–	1.64
FractalNet [101]	38.6M	20	7.27	4.60	28.20	23.30	1.87
DiracNet (28-5) [253]	9.1M	28	–	5.16	–	23.44	–
DiracNet (28-10) [253]	36.5M	28	–	4.75	–	21.54	–
DenseNet (k=24) [77]	27.2M	24	5.83	3.74	23.42	19.25	–
Highway net. [207]	$\sim$ 2.3M	19	–	7.72	–	32.39	–
VGG [253]	20M	16	–	6.09	–	25.92	–
ELU [34]	>17M	18	–	6.55	–	24.28	–
Maxout [58]	>6M	5	11.68	9.38	38.57	–	2.47
DSN [107]	1.2M	10	9.69	7.97	–	34.57	1.92
All-CNN [202]	1.3M	8	9.08	7.25	–	33.71	–
NiN [119]	1.3M	10	10.41	8.81	35.68	–	2.35
NiN	11.1M	17	9.50	6.00	32.30	25.05	–
Baseline: Delta-Orthogonal init [235]	17.8M	32	18.54	18.00	55.65	36.96	–
Baseline: PlainNet-BN	7.3M	32	12.21	10.10	50.07	37.54	–
Baseline: PlainNet-BN	10.1M	50	18.39	46.58	66.30	46.47	–
Ours: Proposed PlainNet	1.7M	32	**8.23**	**6.36**	**30.00**	**25.91**	**1.75**
Ours: Proposed PlainNet	7.3M	32	**7.18**	**5.11**	**27.76**	**23.03**	**–**
Ours: Proposed PlainNet	10.1M	50	**7.89**	**5.38**	**28.56**	**23.41**	**–**
Ours: Proposed PlainNet †	36.3M	26	**–**	**4.92**	**–**	**21.60**	**–**

†Using Wide-ResNet model where skip connections are first eliminated before training

Table 5.2: Error rate (%) on ImageNet dataset using a single model with single crop

Models	Param.	Depth	Top-1	Top-5
ResNet (MatConvNet)[a] [67]	45M	101	23.4	7.0
ResNet (MatConvNet)[a] [67]	60M	152	23.0	6.7
AlexNet with batch norm (MatConvNet)[a] [100]	62M	8	41.8	19.2
VGG-16 with batch norm (MatConvNet)[a] [194]	138M	16	28.3	9.5
VGG-19 with batch norm [193]	144M	19	26.9	8.8
Ours: Proposed PlainNet	93M	22	**26.6**	**8.4**
Ours: Proposed PlainNet	41M	30	**28.6**	**9.9**
Ours: Proposed PlainNet	147M	30	**25.4**	**8.2**
Ours: Proposed PlainNet with dropout[b]	147M	30	**24.0**	**7.3**

[a] Retrived on 15 April 2020 from http://www.vlfeat.org/matconvnet/pretrained/

5.4.3 Comparison with ResNet

Consider a ResNet block [67], l, composed of two hidden layers indexed 1 and 2 that learns an overall transformation of the form

$$\boldsymbol{H}(\boldsymbol{x})^l = \varphi(\boldsymbol{W}_2^l(\varphi(\boldsymbol{W}_1^l\boldsymbol{H}(\boldsymbol{x})^{l-1})) + \boldsymbol{H}(\boldsymbol{x})^{l-1}), \tag{5.9}$$

where the biases are omitted for the sake of simplicity. Let

$$\boldsymbol{Z}_1^l = \boldsymbol{W}_1^l\boldsymbol{H}(\boldsymbol{x})^{l-1}, \tag{5.10}$$

then we can write

$$\boldsymbol{H}(\boldsymbol{x})^l = \varphi(\boldsymbol{W}_2^l(\varphi(\boldsymbol{Z}_1^l)) + \boldsymbol{H}(\boldsymbol{x})^{l-1}). \tag{5.11}$$

For ResNets, the units are ReLUs, while for the proposed PlainNet, the units are LReLUs. Again, in an extreme case, where a training batch produces many units $z_{1j}^l \ll 0$ in $\boldsymbol{Z}_1^l$, then the ResNet transformation becomes

$$\boldsymbol{H}(\boldsymbol{x})^l \approx \varphi(\boldsymbol{H}(\boldsymbol{x})^{l-1}). \tag{5.12}$$

For the same condition, the proposed PlainNet transformation becomes

$$\boldsymbol{H}(\boldsymbol{x})^l \approx \varphi(\boldsymbol{W}_2^l(\beta(\boldsymbol{Z}_1^l))) = \varphi(\boldsymbol{W}_2^l(\beta(\boldsymbol{W}_1^l\boldsymbol{H}(\boldsymbol{x})^{l-1}))). \tag{5.13}$$

It is observed that for both ResNet and the proposed PlainNet, most units h_j^l in $\boldsymbol{H}(\boldsymbol{x})^l$ are non-zero as a result of the bad condition of $\boldsymbol{Z}_1^l$. Consequently, most units h_j^l in $\boldsymbol{H}(\boldsymbol{x})^l$ receive error gradients and get updated during the training. It is easy to see that when training (i.e optimization) is factored over $\boldsymbol{W}_1^l$ and $\boldsymbol{W}_2^l$, the transformation learned by the proposed PlainNet is more non-linear than that learned by the ResNet. However, the increase of non-linearity comes at the price of harder optimization as against the ResNet. Interestingly, the resulting transformation given above for the ResNet with badly conditioned $\boldsymbol{Z}_1^l$ (which is prevalent in very deep networks [143]) could explain why its effective depth is much smaller than the topological depth [221] and easier to optimize [67]; $\boldsymbol{H}(\boldsymbol{x})^l$ is *essentially* equivalent to passing $\boldsymbol{H}(\boldsymbol{x})^{l-1}$ through the rectified linear function.

5.5 Experiments

In this section, we report the experimental results on four benchmarking datasets, CIFAR-10 (C10), CIFAR-100 (C100) [98], SVHN [138] and ImageNet (ILSVRC-2012) [145]. C10 and C100 experiments that employ data augmentation (+) use random horizontal flipping, translation by 4 pixels and reflection. ImageNet experiment use random horizontal flipping and cropping of size 224$\times$224 pixels for the 1.2M training samples. ImageNet model evaluation uses a single centre crop of size 224$\times$224 pixels from resized 50,000 validation images of size 256$\times$256 pixels.

We acknowledge the work [254] which suggests that going arbitrarily deep may not significantly improve model performance. Moreover, wide and moderately deep networks are computationally more efficient [254]. Therefore, 32-layer and 50-layer PlainNets are trained on C10 and C100 datasets; while 22-layer and 30-layer PlainNets are trained on the ImageNet dataset. The architectures of the PlainNets for CIFAR datasets are similar to the network-in-network (NiN) [119], with 3 $\times$ 3 and 1 $\times$ 1 convolution filters, three max-pooling layers and a Global Average Pooling (GAP) layer that is followed by a softmax layer. All our experiments use $\beta = 0.3$ for LReLUs and a max-norm of 1 to 5 for all weights and biases. All models are trained using batch normalization and mini-batch gradient descent with an initial learning rate of 0.1 which is annealed over training; the Nesterov momentum rate is set at 0.9. The different layers use units' dropout in the range 2.5% to 35%; dropout is not used for ImageNet experiments. The proposed PlainNets are trained for 250 and 350 epochs on C10 and C100 datasets, respectively; 90 epochs are used for ImageNet. CIFAR and SVHN dataset results are given in Table 5.1. It is seen that the proposed 32-layer PlainNet is very competitive with the ResNet [67] and some of its variants [207, 69, 76, 254, 101, 253]. It is noted that the DenseNet [77] yields better results; however, it is a more complicated architecture than the ResNet [67]. The proposed PlainNet outperforms all earlier PlainNets [253, 58, 119, 118, 107, 202]. The 50-layer PlainNet shows slight degraded performance compared to the 32-layer PlainNet; however, models with shortcut connections also exhibit similar behavior beyond certain depth [67, 254]; this could be due to overfitting [67], as this chapter

focuses on optimizing the proposed PlainNet. Nevertheless, the results are promising given that PlainNets of these depths have never been trained to achieve such error rates as reported in this work; 15% error rate on C10+ was reported in [67] for a 56-layer PlainNet. Also, experiments show that a 56-layer PlainNet trained conventionally using C100+ gives a noticeably poor error rate of 45.3%. Moreover, this is the first work on very deep PlainNets that directly compares with models that employ shortcut connections.

For ImageNet experiment, the proposed 30-layer PlainNet with dropout gives the best results reported in literature using the same training and testing protocols as employed in this work; top-1 and top-5 error rates of 24% and 7.3% are obtained respectively as given in Table 5.2. This is an improvement of 2.9% on top-1 error rate over VGG-19 result, which has not been surpassed since it was proposed in 2015 [194]. Particularly, observe that the VGG models were not trained from scratch due to optimization problems [194]. Rather, a shallower (11-layer) model that can be properly optimized is first trained, then its parameters used for initializing a deeper model. In contrast, the proposed PlainNets which are even deeper than the VGG models are trained from scratch. To the best of our knowledge, this work presents the deepest PlainNets that are successfully trained from scratch on challenging datasets. We note that our 30-layer model with 41M parameters outperforms AlexNet with 61M parameters by 13.2% on error rate. Furthermore, our 22-layer model with 91M parameters slightly outperforms VGG-16 with 138M parameters. Considering improved results on the ImageNet dataset, the proposed PlainNets are expected to yeild improved results on transfer learning tasks, as against AlexNet and VGG models. It is observed that models with shortcut connections [67, 31] outperform the proposed PlainNets on ImageNet. We posit that their ensembling characteristic [221] maybe a major contributor for success on such a hard classification task. Nonetheless, the results of this first work on successful very deep PlainNets is a large step in closing the gap with ResNet.

5.6 Conclusion

Many theoretical works suggest the benefit of depth for learning deep networks. However, training very deep PlainNets is hard. In fact, it has become almost stereotypical to think that very deep networks without shortcut connections cannot be successfully trained. We investigate analytically and experimentally the problems that plague training. This chapter proposes directly implementable very deep PlainNets that rely on LReLUs, max-norm constraint and strategic parameters initialization. In contrast to earlier works that are mostly theoretical, the proposed PlainNet is thoroughly validated on benchmarking datasets. Experimental results show that it is competitive with the performance of the ResNet while outperforming all earlier PlainNets. We hope that this work will inspire further investigation and results on PlainNets.

Chapter 6

Explaining Deep Neural Networks with Skip Connections and Summed Hidden Representations

State-of-the-art deep neural networks (DNNs) typically consist of several layers of features representations, and especially rely on skip connections to avoid the difficulty of model optimization as seen in Chapter 3 and Chapter 4. Despite the proliferation of different DNN models that employ various forms of skip connections to achieve remarkable results on benchmarking datasets, a concrete explanation for the successful operation and improved generalization capability of these DNNs is surprisingly still lacking. In this chapter, we focus on investigating the role of skip connections for training very deep DNNs. Our exposition directly provides interesting insights and new interpretations to the following important questions (i) why model optimization is easier (ii) why model generalization is better. Theoretical results reveal that skip connections allow DNNs to circumnavigate the singularity of latent representations that translate to optimization and generalization problems, which plague models without skip connections referred to as PlainNets. For substantiating our analysis, our investigation puts into context some of the most successful skip-connection based DNNs, which include residual networks (ResNets) and residual network with aggre-

gated features (ResNeXt) in relation to PlainNets. Experimental evaluations of these models support the theoretical analysis.

6.1 Introduction

Deep neural networks (DNNs) have become an indispensable tool for numerous learning tasks. Unprecedented applications and results on different difficult computer vision tasks have been reported using DNNs with several layers of features representations [82, 168, 239]. Lately, the computer vision community has started to theoretically investigate many techniques employed for various tasks; examples include DNN capacity [28], batch normalization [117] and variational auto encoders [174]. It is not enough to just construct 'model A' that outperforms 'model B', without a concrete explanation of why it is so. A fascinating empirical observation in recent times is the positive correlation between the depth of DNNs and their generalization performance [158, 64]. In the past, DNN models mostly employed 2 to 4 hidden layers [153, 33], and did not use skip connections, hence referred to as *PlainNets*. However, there has been consistent increase in the number of model layers over time, e.g., Network-in-Network (NiN) [119] with 5 layers, AlexNet [100] with 8 layers, VGG-13 [194], and VGG-16 [194] and VGG-19 [194] with 13, 16 and 19 layers, respectively. Concurrently, there are theoretical works that study the benefit of model depth for learning complex target functions [15, 40, 234].

It is known that extending PlainNets' depths beyond some certain number of layers results in optimization problems; that is, the training set cannot be successfully learned [67, 207]. Subsequently, the difficulties of training such DNNs are alleviated by using skip connections of identity mappings [67, 69]; the works [67, 69] referred to the proposed DNN with skip connections as a residual network (*ResNet*).

Following the success of ResNets, the use of very complicated forms of skip connections can be seen in DNNs such as FractalNet [101], ResNeXt [236], PolyNet [260], DenseNet [77] and Inception-ResNet [211] where the summation or concatenation (or both) of previous layer representations with the current one is employed. In this chapter, we theoretically and

experimentally investigate the role of skip connections for training very deep DNNs. Specifically, we provide new interpretations to the role of skip connections in: 1)simplifying model optimizations, and 2)in improving model generalization. However, for the sake of theoretical analysis, we focus in this chapter on DNNs that use skip connections of identity mappings and the summation of previous layers' representations with the current layer. Particularly, we consider two popular and very successful models, ResNet [67] and residual network with aggregated features (ResNeXt) [236]. Our contributions in this chapter are as follows:

1. Theoretical analyses of the optimization characteristics of very deep DNNs with skip connections is provided where:
    - Training problems beyond saturating activation functions are identified using the power iteration method [17].
    - Generalization capabilities are explained based on the stability of learned parameters (i.e. solutions) for small changes in input or latent data.
2. Extensive experiments are performed to validate the theoretical analysis results using benchmarking datasets which include, MNIST, CIFAR-10, CIFAR-100 and ImageNet.

The remainder of this chapter is organized as follows. In Section 6.2, we give an overview of the related work. The preliminaries on DNN with skip connections are provided in Section 6.3. In Section 6.4, the proposed theoretical analysis of the role of skip connections for different DNN models is presented. We further discuss the impact of skip connections on generalization in Section 6.5. Afterwards, the experiments and discussions are given in Section 6.6. Section 6.7 summarizes the chapter as conclusion.

6.2 Related work

Several works have reported improved results on various tasks using DNNs that are deeper than previous state-of-the-art DNNs [210, 136, 108]. Consequently, emerging works naturally gravitate towards learning deeper DNNs, especially on hard tasks [52]. However, optimizing PlainNets beyond a certain depth is usually not successful [67, 207]. The work [162]

noted that initialization schemes [68, 56] and batch normalization [84] do not resolve optimization problems beyond some certain model depth.

It has been shown that the optimization problems encountered in PlainNets can be circumvented by using skip connections [67, 207]. Some popular DNNs that have been successfully trained using this approach include ResNet [67], ResNeXt [236], highway network [207], Densely Connected Network (DenseNet) [77], Resnet of Resnet (RoR) [256], Dual Path network (DPN) [27], PolyNet [260] and Inception-ResNet [211]. Although, the aforementioned models are very successful for learning different tasks, their operation has remained considerably elusive. Namely, *theoretical explanations* for how DNNs with skip connections manage to avoid optimization problems, and on top of that, generalize even better than PlainNets are still unclear.

In [7], optimization problems associated with gradients shattering are studied. Using carefully crafted experiments and auto correlation function values, it is observed that the gradients of shallow PlainNets are assimilated to brown noise. For deep PlainNets, the gradients are noted to be disorderly and resemble white noise. Interestingly, the gradients of ResNets are found to be well-behaved; Resnet gradients are considerably resistant; their structure is between white and brown noise. They posited that optimizing DNNs with gradients that resemble white noise is difficult; the closer gradients are to white noise, the more difficult optimization becomes.

In [221], it is observed that ResNets act like ensemble of shallow neural networks models; that is, their true depths are lesser than their architectural depth. In order to reveal this fascinating attribute of ResNets, the work in [221] provided and explored an unrolled view of ResNets. The work [221] further posited that paths through ResNets vary in length, and have small dependence on one another. It is concluded that ResNets avoid optimization problems by leveraging shorter paths in the models. In [60], an unrolled iterative estimation concept of features at different stages was proposed. The work argues that a collection of layers iteratively refines the approximations of similar features rather than computing wholely new features. New features are computed at other stages. Our work differs from the aforementioned related works [7, 221, 60] in the approach of investigation and interpretations of

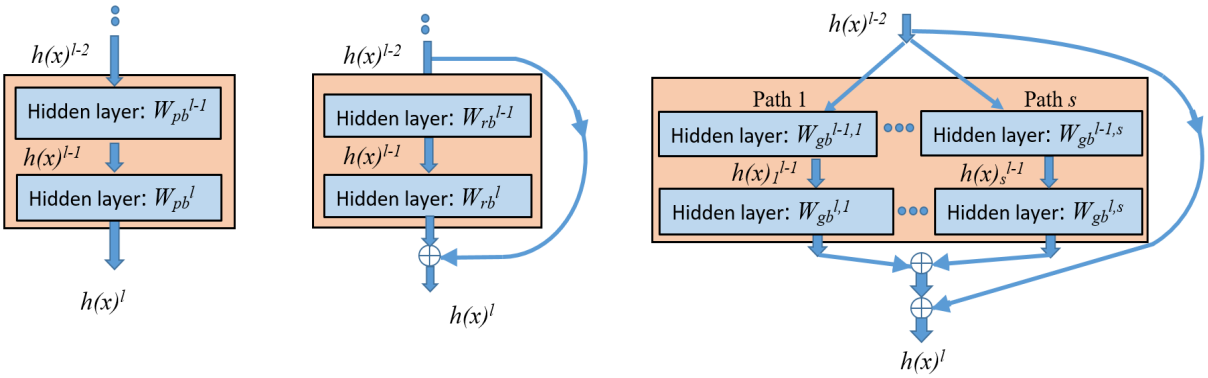


Figure 6.1: Models with skip connections considered in this chapter. $\oplus$ denotes addition operation. Left: PlainNet block. Middle: ResNet block. Right: ResNeXt block. Full DNNs are constructed by stacking several blocks

model operation. In addition, [7] provides no explanation for improved generalization observed in the ResNet.

6.3 Background and preliminaries

6.3.1 Background: importance of studying linear models

For the ease of formalization, the linear activation function is assumed in this chapter, as it is typical in the literature [270, 200, 185, 91, 141]. This is a very common assumption amongst other simplifications that make DNNs amenable to theoretical study; the convergence analysis of gradient descent on linear PlainNets is carried out in [2]. Furthermore, [91] discusses the relevance of studying linear DNNs, and posits that results are often relatable to DNNs with non-linear activation functions given that the loss function of a DNN (i.e., having two or more layers) with linear activation function is still non-convex with respect to the parameters space, as in DNNs with non-linear activation functions. Nevertheless, all of our experimental results use the non-linear activation function to show clear agreement with our theoretical analyses that assume linear activation function.

6.3.2 Preliminaries: PlainNet, ResNet and ResNeXt

In this section, the transformation learned by the DNN without skip connections, PlainNet, is presented. Furthermore, the different transformations learned by the DNNs with skip connections in this chapter, ResNet and ResNeXt, are given. The formalization of the distinct transformations learned by the different DNNs would be useful for the main analyses in the following sections.

(a) Plain network (PlainNet)

This class of DNNs are the strictly hierarchical models, where only a single path connects a current layer to the succeeding one, and there is no skip connection of any sort; see Figure 6.1. Considering the input, $\boldsymbol{h}(\boldsymbol{x})_p^{l-2} \in \mathbb{R}^n$ (where $\boldsymbol{x} \in \mathbb{R}^n$ is the input to the DNN), for the two consecutive layer weights in Figure 6.1, $\boldsymbol{W}_{pb}^l$ and $\boldsymbol{W}_{pb}^{l-1}$, the block's output can be expressed as

$$\boldsymbol{h}(\boldsymbol{x})_p^l = \boldsymbol{W}_{pb}^l \boldsymbol{W}_{pb}^{l-1} \boldsymbol{h}(\boldsymbol{x})_p^{l-2}, \tag{6.1}$$

where $\boldsymbol{W}_{pb}^l, \boldsymbol{W}_{pb}^{l-1} \in \mathbb{R}^{n \times n}$, and pb indicates the weights in a PlainNet block. For simplicity, the result of transformation $\boldsymbol{W}_{pb}^l \boldsymbol{W}_{pb}^{l-1}$ in the PlainNet block in (6.1) is lumped as $\boldsymbol{W}_p^l = \boldsymbol{W}_{pb}^l \boldsymbol{W}_{pb}^{l-1}$, where $\boldsymbol{W}_p^l \in \mathbb{R}^{n \times n}$, so that we now obtain

$$\boldsymbol{h}(\boldsymbol{x})_p^l = \boldsymbol{W}_p^l \boldsymbol{h}(\boldsymbol{x})_p^{l-2}. \tag{6.2}$$

(b) Residual network (ResNet)

The Residual network (ResNet) block [67] is shown in Figure 6.1, and mainly relies on skip connections of identity mappings, which connect every residual block (having 2 or 3 weight layers) to the preceding one. Given the ResNet block, we can write

$$\boldsymbol{h}(\boldsymbol{x})_r^l = \boldsymbol{W}_{rb}^l \boldsymbol{W}_{rb}^{l-1} \boldsymbol{h}(\boldsymbol{x})_r^{l-2} + \boldsymbol{h}(\boldsymbol{x})_r^{l-2}, \tag{6.3}$$

where $\boldsymbol{W}_{rb}^{l}, \boldsymbol{W}_{rb}^{l-1} \in \mathbb{R}^{n\times n}$, and rb indicates the weights in the ResNet block. In addition, considering the identity matrix $\mathbf{I} \in \mathbb{R}^{n\times n}$, factorizing (6.3) yields

$$\boldsymbol{h}(\boldsymbol{x})_r^l = \left(\boldsymbol{W}_{rb}^{l}\boldsymbol{W}_{rb}^{l-1} + \mathbf{I}\right)\boldsymbol{h}(\boldsymbol{x})_r^{l-2}. \tag{6.4}$$

Again, the result of the transformation $\boldsymbol{W}_{rb}^{l}\boldsymbol{W}_{rb}^{l-1}$ in the residual block in (6.4) is lumped as $\boldsymbol{W}_r^l = \boldsymbol{W}_{rb}^{l}\boldsymbol{W}_{rb}^{l-1}$, where $\boldsymbol{W}_r^l \in \mathbb{R}^{n\times n}$, such that (6.4) becomes

$$\boldsymbol{h}(\boldsymbol{x})_r^l = \left(\boldsymbol{W}_r^{l} + \mathbf{I}\right)\boldsymbol{h}(\boldsymbol{x})_r^{l-2}. \tag{6.5}$$

(c) Residual network with features aggregation (ResNeXt)

Residual network with features aggregation (ResNeXt) block [236] aggregates as output the summation of features learned via the different s paths of the ResNeXt block as in Figure 6.1, where $\boldsymbol{W}_{gb}^{l,k}, \boldsymbol{W}_{gb}^{l-1,k} \in \mathbb{R}^{q\times q}$ with $k = 1, \cdots, s$, and gb indicates the weights in the ResNeXt block; s is the referred to as the cardinality of the ResNeXt. Importantly, we observe from the construction of the PlainNet, ResNet [67] and ResNeXt [236] blocks in Figure 6.1 that $q < n$. That is, the PlainNet and ResNet blocks use one large matrix for every transformation, as in $\boldsymbol{W}_p^l \in \mathbb{R}^{n\times n}$ and $\boldsymbol{W}_r^l \in \mathbb{R}^{n\times n}$, respectively; see (6.2) and (6.5). In contrast, the ResNeXt block uses many smaller s number of matrices for every transformation, as in $\boldsymbol{W}_g^{l,k} \in \mathbb{R}^{q\times q}$ given in Figure 6.1. Note that the condition $q < n$ will be important for analysing the subtle difference between the ResNet and ResNeXt in relation to generalization in Section 6.5. For an input $\boldsymbol{h}(\boldsymbol{x})^{l-2} \in \mathbb{R}^q$, the output of the ResNeXt block, $\boldsymbol{h}(\boldsymbol{x})_c^l \in \mathbb{R}^q$, is

$$\begin{aligned}\boldsymbol{h}(\boldsymbol{x})_c^l =& \boldsymbol{W}_{gb}^{l,s}\boldsymbol{W}_{gb}^{l-1,s}\boldsymbol{h}(\boldsymbol{x})_c^{l-2} + \cdots + \boldsymbol{W}_{gb}^{l,k}\boldsymbol{W}_{gb}^{l-1,k}\boldsymbol{h}(\boldsymbol{x})_c^{l-2} \\ &+ \cdots + \boldsymbol{W}_{gb}^{l,1}\boldsymbol{W}_{gb}^{l-1,1}\boldsymbol{h}(\boldsymbol{x})^{l-2} + \boldsymbol{h}(\boldsymbol{x})_c^{l-2}.\end{aligned} \tag{6.6}$$

Similarly, for compactness, we let $\boldsymbol{W}_g^{l,k} = \boldsymbol{W}_{gb}^{l,k}\boldsymbol{W}_{gb}^{l-1,k}$ in (6.6), where $\boldsymbol{W}_g^{l,k} \in \mathbb{R}^{q\times q}$. Finally, using $\mathbf{I} \in \mathbb{R}^{q\times q}$ and factorizing (6.6) gives

$$\boldsymbol{h}(\boldsymbol{x})_c^l = (\boldsymbol{W}_g^{l,s} + \cdots + \boldsymbol{W}_g^{l,k} + \cdots + \boldsymbol{W}_g^{l,1} + \mathbf{I})\boldsymbol{h}(\boldsymbol{x})_c^{l-2}, \tag{6.7}$$

which in a compact form is

$$\boldsymbol{h}(\boldsymbol{x})_c^l = (\sum_{k=1}^{s} \boldsymbol{W}_g^{l,k} + \mathbf{I})\boldsymbol{h}(\boldsymbol{x})_c^{l-2}. \tag{6.8}$$

6.4 Theoretical analysis of the role of skip connections for optimization

The central idea of this section is to investigate and analyse the role of skip connections presented in Section 6.3.2 in avoiding information loss while training a very deep DNN. Our analysis builds on the following assumption, propositions, corollary, definition and theorems.

Assumption 6.1. *For an m-layer DNN, all layer weight matrices, $\boldsymbol{W}^l \in \mathbb{R}^{n\times n}$, for $1 \leq l \leq m$, are assumed to be jointly diagonizable; implying the same eigenvectors $\{\boldsymbol{v}_i\}_{i=1}^n$, forming a basis in $\mathbb{R}^n$, and their corresponding eigenvalues $\{\lambda_i^l\}_{i=1}^n$.*

Although Assumption 6.1 is somewhat strict, it can be found in the literature [153], where it allows for the tractability of theoretical analysis.
Our analysis starts first on the forward-pass and then on the backpropagation in relation to optimization conditions.

Proposition 6.1. *For a matrix $\boldsymbol{W} \in \mathbb{R}^{n\times n}$ whose column vectors, $\boldsymbol{w}_i$, are drawn from a uniform distribution $U[-r, r]$ or a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, the probability that $\boldsymbol{W}$ is non-singular, $P(\boldsymbol{w_i} \notin \boldsymbol{W}_{-i}^{span})$, is*

$$P(\boldsymbol{w}_i \notin \boldsymbol{W}_{-i}^{span}) = 1 \ : 1 \leq i \leq n \tag{6.9}$$

Proof. See Section A1 in the appendix. □

Corollary 6.1. *Considering that DNN parameters, $\boldsymbol{W}^l : 1 \leq l \leq L$, are randomly initialized [68, 56], W^l is therefore guaranteed to be a non-singular matrix at the start of training.*

Proof. This follows directly from Proposition 6.1. □

Definition 6.1. *Given $\{\boldsymbol{v_i}\}_{i=1}^n$ is a set of eigenvectors (i.e. orthogonal vectors) as in Assumption 6.1, any $\boldsymbol{x} \in \mathbb{R}^n$ can be expressed as*

$$\boldsymbol{x} = \sum_{i=1}^{n} \alpha_i \boldsymbol{v_i}, \quad \alpha_i \in \mathbb{R}. \tag{6.10}$$

Definition 6.2. *A batch of N hidden representations, $\boldsymbol{H}(\boldsymbol{x}) \in \mathbb{R}^{n \times N}$, is composed of N individual data points, $\boldsymbol{h}(\boldsymbol{x})_i \in \mathbb{R}^n$; that is, $\boldsymbol{H}(\boldsymbol{x}) = [\boldsymbol{h}(\boldsymbol{x})_1, \cdots, \boldsymbol{h}(\boldsymbol{x})_i, \cdots, \boldsymbol{h}(\boldsymbol{x})_N]$.*

Proposition 6.2. *Given a non-invertible matrix $\boldsymbol{M} \in \mathbb{R}^{n \times n}$, and vector $\boldsymbol{v} \in \mathbb{R}^n$, then the transformation $\boldsymbol{w} = \boldsymbol{M}\boldsymbol{v}$ incurs information loss.*

Proof. First note that a non-invertible matrix is singular, has a determinant of zero, and its transformation collapses space. Consequently, $\boldsymbol{v}$ cannot be recovered with certainty by $\boldsymbol{M}^\dagger \boldsymbol{w}$, where † denotes the pseudo-inverse of a matrix, since the mapping $\boldsymbol{v} \mapsto \boldsymbol{w}$ is non-injective. □

It is critical to note that *singularity*, where at least one singular value of $\boldsymbol{M}$ becomes zero is not necessary for optimization problems; *near-singularity*, where at least one singular value is extremely small is sufficient. This is directly reflected in the condition number κ of $\boldsymbol{M}$ defined as

$$\kappa(\boldsymbol{M}) = \sigma_{max}(\boldsymbol{M}) / \sigma_{min}(\boldsymbol{M}), \tag{6.11}$$

where $\sigma_{max}(\boldsymbol{M})$ and $\sigma_{min}(\boldsymbol{M})$ are the maximum and minimum singular values of $\boldsymbol{M}$, respectively. Problems that have $\kappa(\boldsymbol{M}) \gg 1$ are commonly referred to as ill-conditioned, and the solutions obtained are typically quite unstable.

6.4.1 Forward-pass: loss of basis in hidden representations

(a) Plain network (PlainNet)

For the PlainNet in Section 6.3.2 (a), we state the following theorem.

Theorem 6.1. *Given an input $\boldsymbol{x} \in \mathbb{R}^n$ to a linear m-layer PlainNet parameterized as $\{\boldsymbol{W^l}\}_{l=1}^m \in \mathbb{R}^{n\times n}$, the hidden layer output, $\boldsymbol{h}(\boldsymbol{x})_p^m : m \gg 1$, is*

$$\begin{cases} \boldsymbol{h}(\boldsymbol{x})_p^m \approx \prod_{l=1}^{m} \lambda_1^l \left(\alpha_1 \boldsymbol{v}_1\right), \\ \text{s.t.} \quad \left|\lambda_1^l\right| > \left|\lambda_i^l\right| \quad : \ 2 \leq i \leq n, \end{cases} \tag{6.12}$$

where λ_i^l are the first components of the eigenvalues. The basis $\{\boldsymbol{v}_i\}$ is the eigenvectors of $\boldsymbol{W}^l$, with $l = 1, \cdots, m$, and the scalar α_1 is the first coordinate in this space.

Proof. The power iteration method [17] is modified (since different weights are applied at the different layers of the PlainNet) for analysing the impact of successive layer weights transformations on $\boldsymbol{h}(\boldsymbol{x})_p^m$ as follows.
Using Assumption 6.1 for layer l, we can write

$$\boldsymbol{W}_p^l \boldsymbol{v}_i = \lambda_i^l \boldsymbol{v}_i. \tag{6.13}$$

For the final PlainNet output at layer m, we can write

$$\boldsymbol{h}(\boldsymbol{x})_p^m = \prod_{l=1}^{m} \boldsymbol{W}_p^l \boldsymbol{x}. \tag{6.14}$$

Putting (6.10) and (6.13) into (6.14) and considering Assumption 6.1, we have (see Section A2 in the appendix)

$$\boldsymbol{h}(\boldsymbol{x})_p^m = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i \prod_{l=1}^{m} \lambda_i^l, \tag{6.15}$$

which can be rewritten as

$$\boldsymbol{h}(\boldsymbol{x})_p^m = \left(\alpha_1 \boldsymbol{v}_1 \prod_{l=1}^{m} \lambda_1^l + \sum_{i=2}^{n} \alpha_i \boldsymbol{v}_i \prod_{l=1}^{m} \lambda_i^l\right). \tag{6.16}$$

Furthermore, (6.16) can be factorized so that

$$\boldsymbol{h}(\boldsymbol{x})_p^m = \prod_{l=1}^{m} \lambda_1^l \left(\alpha_1 \boldsymbol{v}_1 + \sum_{i=2}^{n} \alpha_i \boldsymbol{v}_i \prod_{l=1}^{m} (\lambda_i/\lambda_1)^l \right). \tag{6.17}$$

Finally, (6.12) is obtained from (6.17) given that $\prod_{l=1}^{m} (\lambda_i/\lambda_1)^l \longrightarrow 0$ for $m \gg 1$ and $\left|\lambda_1^l\right| > \left|\lambda_i^l\right| \ : \ 2 \leq i \leq n$, as expected for eigenvalues. □

Ultimately, note that a similar observation $\prod_{l=1}^{m} (\lambda_i/\lambda_1)^l \longrightarrow 0$ for $m \gg 1$ is used for concluding the following theorems.

Remark 6.1. *For an input $\boldsymbol{x}$, only one basis vector, $\boldsymbol{v_1}$, contributes to the computation of $\boldsymbol{h}(\boldsymbol{x})^m$ for $m \gg 1$, as a result of repeated multiplication by $\{\boldsymbol{W}^l\}_{l=1}^m$. As such, $\boldsymbol{h}(\boldsymbol{x})_p^m$ for the PlainNet exhibits significant information loss. Considering Definition 6.2 and Theorem 6.1, the columns of $\boldsymbol{H}(\boldsymbol{x})_p^m$ are colinear; hence, $\boldsymbol{H}(\boldsymbol{x})_p^m$ is singular.*

Going by Remark 6.1, we observe that [45] used deep Gaussian process to arrive at a similar result that the representational capacity of DNNs collapses to one degree of freedom as $m \longrightarrow \infty$. Furthermore, it can be concluded from Theorem 6.1 that the information loss of hidden representations is not associated with the popular unit saturation problem [68], as Theorem 6.1 uses linear units.

(b) Residual network (ResNet)

For analyzing features conditioning of ResNet given in Section 6.3.2 (b), we rely on the following lemma and axiom.

Lemma 6.1. *For a matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ with eigenvector $\boldsymbol{v} \in \mathbb{R}^n$ and eigenvalue λ such that $\boldsymbol{W}\boldsymbol{v} = \lambda \boldsymbol{v}$, it can be shown that $(\boldsymbol{W} + c\mathbf{I})\boldsymbol{v} = (\lambda + c)\boldsymbol{v}$ for a scalar c.*

Proof. $(\boldsymbol{W} + c\mathbf{I})\boldsymbol{v} = \boldsymbol{W}\boldsymbol{v} + c\boldsymbol{v} = \lambda\boldsymbol{v} + c\boldsymbol{v} = (\lambda + c)\boldsymbol{v}$. □

Axiom 6.1. *For sequential products of a sum of variable and constant, we can write $\prod_{l=1}^{m}(\lambda^l + 1) = 1 + \sum_{l=1}^{m} \lambda^l + ... + \prod_{l=1}^{m} \lambda^l$, where the middle terms are inconsequential for stating the next theorem.*

Theorem 6.2. *Given an input $\boldsymbol{x} \in \mathbb{R}^n$ to a linear m-layer ResNet parameterized as $\{\boldsymbol{W}^l\}_{l=1}^m \in \mathbb{R}^{n\times n}$, the hidden layer output, $\boldsymbol{h}(\boldsymbol{x})_r^m : m \gg 1$, is*

$$\begin{cases} \boldsymbol{h}(\boldsymbol{x})_r^m \approx \sum_{i=1}^n \alpha_i \boldsymbol{v}_i + \alpha_1 \boldsymbol{v}_1 \left(\sum_{l=1}^m \lambda_1^l + \cdots + \prod_{l=1}^m \lambda_1^l \right) \\ \qquad\qquad \textit{s.t.} \ \left|\lambda_1^l\right| > \left|\lambda_i^l\right| \quad : \ 2 \le i \le n. \end{cases} \tag{6.18}$$

Proof. Again, the modified power iteration method is employed. Using (6.5), we can write for the ResNet

$$\boldsymbol{h}(\boldsymbol{x})_r^m = \prod_{l=1}^m (\boldsymbol{W}_r^l + \mathbf{I})\boldsymbol{x}. \tag{6.19}$$

Applying Lemma 6.1 with $c = 1$ gives $(\boldsymbol{W}_r^l + \mathbf{I})\boldsymbol{v}_i = (\lambda_i^l + 1)\boldsymbol{v}_i$; putting (6.10) into (6.19) and considering Assumption 6.1 gives

$$\boldsymbol{h}(\boldsymbol{x})_r^m = \sum_{i=1}^n \alpha_i \boldsymbol{v}_i \prod_{l=1}^m (\lambda_i^l + 1). \tag{6.20}$$

Using Axiom 6.1 for (6.20), we can write

$$\boldsymbol{h}(\boldsymbol{x})_r^m = \sum_{i=1}^n \alpha_i \boldsymbol{v}_i \left(1 + \sum_{l=1}^m \lambda_i^l + \cdots + \prod_{l=1}^m \lambda_i^l \right). \tag{6.21}$$

Furthermore, (6.21) can be expanded as

$$\boldsymbol{h}(\boldsymbol{x})_r^m = \sum_{i=1}^n \alpha_i \boldsymbol{v}_i + \alpha_1 \boldsymbol{v}_1 \left(\sum_{l=1}^m \lambda_1^l + ... + \prod_{l=1}^m \lambda_1^l \right) + \sum_{i=2}^n \alpha_i \boldsymbol{v}_i \left(\sum_{l=1}^m \lambda_i^l + ... + \prod_{l=1}^m \lambda_i^l \right). \tag{6.22}$$

The factorization of (6.22) yields

$$\boldsymbol{h}(\boldsymbol{x})_r^m = \sum_{i=1}^n \alpha_i \boldsymbol{v}_i + \left(\sum_{l=1}^m \lambda_1^l + ... + \prod_{l=1}^m \lambda_1^l \right) \left\{ \alpha_1 \boldsymbol{v}_1 + \sum_{i=2}^n \alpha_i \boldsymbol{v}_i \left(\frac{\sum_{l=1}^m \lambda_i^l + ... + \prod_{l=1}^m \lambda_i^l}{\sum_{l=1}^m \lambda_1^l + ... + \prod_{l=1}^m \lambda_1^l} \right) \right\}. \tag{6.23}$$

Lastly, applying $m \gg 1$ and the condition $\left|\lambda_1^l\right| > \left|\lambda_i^l\right| \ : \ 2 \le i \le n$, as expected for eigenvalues to (6.23) completes the proof of Theorem 6.2. □

Remark 6.2. *Given the input $\boldsymbol{x}$, all basis vectors $\{\boldsymbol{v}_i\}_{i=1}^n$, contribute to the computation of $\boldsymbol{h}(\boldsymbol{x})_r^m$ for $m \gg 1$. As such, $\boldsymbol{h}(\boldsymbol{x})_r^m$ for the ResNet retain full information. From Definition 6.2 and Theorem 6.2, the columns of $\boldsymbol{H}(\boldsymbol{x})_r^m$ are distinct, and therefore $\boldsymbol{H}(\boldsymbol{x})_r^m$ is non-singular.*

(c) ResNeXt

The following ResNeXt analysis is based on Section 6.3.2 (c)

Theorem 6.3. *Given an input $\boldsymbol{x} \in \mathbb{R}^n$ to a linear m-layer ResNeXt parameterized as $\{\boldsymbol{W}_c^{l,k}\}_{l=1}^m$, the hidden layer output, $\boldsymbol{h}(\boldsymbol{x})^m : m \gg 1$, is*

$$\begin{cases} \boldsymbol{h}(\boldsymbol{x})_c^m \approx \sum_{i=1}^n \alpha_i \boldsymbol{v}_i + \alpha_1 \boldsymbol{v}_1 \left(\sum_{l=1}^m \sum_{j=1}^s \lambda_1^{l,s} + ... + \prod_{l=1}^m \sum_{j=1}^s \lambda_1^{l,s} \right) \\ \quad \textit{s.t.} \ \left|\lambda_1^l\right| > \left|\lambda_i^l\right| \ : \ 2 \leq i \leq n. \end{cases} \tag{6.24}$$

Proof. See Section A3 in the appendix. □

Remark 6.3. *Given the input $\boldsymbol{x}$, all basis vectors $\{\boldsymbol{v}_i\}_{i=1}^n$, contribute to the computation of $\boldsymbol{h}(\boldsymbol{x})_c^m$ for $m \gg 1$. Similar to ResNet, $\boldsymbol{h}(\boldsymbol{x})_c^m$ for the ResNeXt retain full information. Again, employing Definition 6.2 and Theorem 6.3 shows that the columns of $\boldsymbol{H}(\boldsymbol{x})_c^m$ are distinct, and thus $\boldsymbol{H}(\boldsymbol{x})_c^m$ is non-singular.*

6.4.2 Foward-pass: Without the Layer Weights Joint Diagonalization Assumption

In this section, we propose an improved theoretical analysis of the forward-pass of the PlainNet, ResNet and ResNeXt studied in Section 6.4.1 by eliminating Assumption 6.1 to obtain more practical formal results. The theoretical study relies on the following definition, lemma, proposition and corollary.

Definition 6.3. *If the vector set, $\{\boldsymbol{w}_i \in \mathbb{R}^n\}_{i=1}^n$, are the columns of a non-singular matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$, then the vectors $\{\boldsymbol{w}_i\}_{i=1}^n$ are linearly independent and span $\mathbb{R}^n$.*

For remaining parts of this chapter, we clarify the notations relating to DNN weight matrices, $\boldsymbol{W}^m$ and $(\boldsymbol{W})^m$. Note that $\boldsymbol{W}^m$ is used to refer to the weight matrix at layer m, while

$(\boldsymbol{W})^m$ is used to refer to a weight matrix $\boldsymbol{W}$ raised to power m. That is, $(\boldsymbol{W})^m$ means the weight matrix $\boldsymbol{W}$ multiplied m times.

Lemma 6.2. *(Alexeev-Götze-Tikhomirov [**52**]) Given the entries of the matrix $\boldsymbol{W}^l \in \mathbb{R}^{n\times n}$: $1 \leq l \leq m$ are randomly drawn from a Gaussian distribution and $\boldsymbol{W} \in \{\boldsymbol{W}^1, \cdots, \boldsymbol{W}^l, \cdots, \boldsymbol{W}^m\}$ so that we have the products $\boldsymbol{Y} = \prod_{l=1}^{m} \boldsymbol{W}^l$ and $\boldsymbol{Z} = (\boldsymbol{W})^m$, the asymptotic behaviour of the singular spectrum of $\boldsymbol{Y}$ is the same as that of the singular spectrum of $\boldsymbol{Z}$.*

Proof. See [**52**] for the proof that the limit distributions of the singular spectrum of both $\boldsymbol{Y}$ and $\boldsymbol{Z}$ is the Fuss-Catalan distribution.

Proposition 6.3. *Given a matrix $\boldsymbol{W} \in \mathbb{R}^{n\times n}$ whose column vectors, $\boldsymbol{w}_i$, are drawn from a uniform distribution or a Gaussian distribution, the probability that $\boldsymbol{W}$ is non-singular, $P(\boldsymbol{w_i} \notin \boldsymbol{W}_{-i}^{span})$, is*

$$P(\boldsymbol{w}_i \notin \boldsymbol{W}_{-i}^{span}) = 1 \ : 1 \leq i \leq n \tag{6.25}$$

Proof. See Section A1 in the appendix.

Corollary 6.2. *For an m-layer DNN, the initialization [68, 56] of all the layer weight matrices, $\{\boldsymbol{W}^l \in \mathbb{R}^{n\times n}\}_{l=1}^{m}$, follows Proposition 6.1, and thus are all non-singular; using Definition 6.3 shows that every $\boldsymbol{W}^l$ is of rank n. Consequently, any $\boldsymbol{W} \in \{\boldsymbol{W}^l\}_{l=1}^{m}$ admits a singular value decomposition (SVD) of the form*

$$\boldsymbol{W} = \sum_{i=1}^{n} \sigma_i \boldsymbol{u_i} \boldsymbol{v_i}^T, \ \ \sigma_i \in \mathbb{R}, \ \sigma_i > 0, \tag{6.26}$$

where σ_i are the singular values of $\boldsymbol{W}$; $\boldsymbol{u_i} \in \mathbb{R}^n$ and $\boldsymbol{v_i} \in \mathbb{R}^n$ are the left and right singular vectors of $\boldsymbol{W}$, respectively; and T denotes vector transpose.

Lemma 6.3. *For a matrix $\boldsymbol{W} \in \mathbb{R}^{n\times n}$, using SVD as in* (6.26)*, $(\boldsymbol{W})^m$ (i.e. $\boldsymbol{W}$ raised to the power of m) can be written as*

$$(\boldsymbol{W})^m = \sum_{i=1}^{n} \sigma_i^{2m} \boldsymbol{u_i} \boldsymbol{u_i}^T. \tag{6.27}$$

Proof. First, let $\boldsymbol{W} = BB^T$, and $B = \sum_{i=1}^{n} \sigma_i \boldsymbol{u_i} \boldsymbol{v_i}^T$ so that

$$\boldsymbol{W} = \left(\sum_{i=1}^{n} \sigma_i \boldsymbol{u_i} \boldsymbol{v_i}^T \right) \left(\sum_{j=1}^{n} \sigma_j \boldsymbol{v_j} \boldsymbol{u_j}^T \right), \tag{6.28}$$

where an expansion gives

$$\boldsymbol{W} = \sum_{i,j=1}^{n} \sigma_i \sigma_j \boldsymbol{u_i} (\boldsymbol{v_i}^T \boldsymbol{v_j}) \boldsymbol{u_j}^T. \tag{6.29}$$

Given, $\boldsymbol{v_i^T} \boldsymbol{v_j} = 0$ for $i \neq j$, and $\boldsymbol{v_i^T} \boldsymbol{v_i} = 1$, we have

$$\boldsymbol{W} = \sum_{i}^{n} \sigma_i^2 \boldsymbol{u_i} \boldsymbol{u_i}^T. \tag{6.30}$$

From repeated multiplication, we can arrive at the general and compact expression

$$(\boldsymbol{W})^m = \sum_{i=1}^{n} \sigma_i^{2m} \boldsymbol{u_i} \boldsymbol{u_i}^T. \tag{6.31}$$

This concludes the proof. □

Furthermore, given $\{\boldsymbol{u_i}\}_{i=1}^{n}$ is a set of orthonormal vectors as in (6.26) so that they form a basis in $\mathbb{R}^n$, we can express any $\boldsymbol{x} \in \mathbb{R}^n$ as

$$\boldsymbol{x} = \sum_{i=1}^{n} \alpha_i \boldsymbol{u_i}, \quad \alpha_i \in \mathbb{R}. \tag{6.32}$$

Our analysis for the different models are as follows.

(b) Plain network (PlainNet)

Given the PlainNet in Section 6.3.2 (a), we state the following theorem.

Theorem 6.4. *For an input $\boldsymbol{x} \in \mathbb{R}^n$ to a linear m-layer PlainNet parameterized by $\theta_p = \{\boldsymbol{W}_p^m \in \mathbb{R}^{n \times n}, \cdots, \boldsymbol{W}_p^l \in \mathbb{R}^{n \times n}, \cdots, \boldsymbol{W}_p^2 \in \mathbb{R}^{n \times n}, \boldsymbol{W}_p^1 \in \mathbb{R}^{n \times n}\}$, the hidden layer output, $\boldsymbol{h}(\boldsymbol{x})_p^m : m \longrightarrow \infty$, is*

$$\boldsymbol{h}(\boldsymbol{x})_p^m \approx \alpha_1 \sigma_{p_1}^{2m} \boldsymbol{u_1}, \tag{6.33}$$

where σ_{p_1} and $\boldsymbol{u}_1$ are the first singular value component and first singular vector of $\boldsymbol{W}_p^l$, respectively; the scalar α_1 is the first component of $\{\boldsymbol{\alpha}_i\}_{i=1}^n$ as in (6.32).

Proof. The output of the PlainNet in the last layer m can be written as

$$\boldsymbol{h}(\boldsymbol{x})_p^m = \prod_{l=1}^{m} \boldsymbol{W}_p^l \boldsymbol{x}. \tag{6.34}$$

Since we are ultimately interested in studying the singular values of $\boldsymbol{h}(\boldsymbol{x})_p^m : m \longrightarrow \infty$, Lemma 6.2 simplifies (6.34) to

$$\boldsymbol{h}(\boldsymbol{x})_p^m = (\boldsymbol{W}_p)^m \boldsymbol{x}. \tag{6.35}$$

From Lemma 6.3, putting $(\boldsymbol{W}_p)^m$ in (6.35) yields

$$\boldsymbol{h}(\boldsymbol{x})_p^m = \sum_{i=1}^{n} \sigma_{p_i}^{2m} \boldsymbol{u_i} \boldsymbol{u_i}^T . \boldsymbol{x}, \tag{6.36}$$

where writing out (6.36) gives

$$\boldsymbol{h}(\boldsymbol{x})_p^m = \left(\sigma_{p_1}^{2m} \boldsymbol{u_1} \boldsymbol{u_1}^T + \sum_{i=2}^{n} \sigma_{p_i}^{2m} \boldsymbol{u_i} \boldsymbol{u_i}^T \right) \boldsymbol{x}. \tag{6.37}$$

Subsequently, (6.37) is factorized so that we obtain

$$\boldsymbol{h}(\boldsymbol{x})_p^m = \sigma_{p_1}^{2m} \boldsymbol{u_1} \boldsymbol{u_1}^T \left(\mathbf{I} + \sum_{i=2}^{n} \frac{\sigma_{p_i}^{2m}}{\sigma_{p_1}^{2m}} (\boldsymbol{u_1} \boldsymbol{u_1}^T)^{-1} \boldsymbol{u_i} \boldsymbol{u_i}^T \right) \boldsymbol{x}. \tag{6.38}$$

Given that $m \longrightarrow \infty$ and $\sigma_{p_1} > \sigma_{p_i} : \ 2 \leq i \leq n$ as expected for singular values, (6.38) becomes

$$\boldsymbol{h}(\boldsymbol{x})_p^m \approx \sigma_{p_1}^{2m} \boldsymbol{u_1} \boldsymbol{u_1}^T \ \boldsymbol{x}. \tag{6.39}$$

Putting (6.32) in (6.39) gives

$$\boldsymbol{h}(\boldsymbol{x})_p^m \approx \sigma_{p_1}^{2m} \boldsymbol{u_1} \boldsymbol{u_1}^T \sum_{i=1}^{n} \alpha_i \boldsymbol{u_i}. \tag{6.40}$$

Finally, applying $\boldsymbol{u_i}^T \boldsymbol{u_j} = 0$ for $i \neq j$ and $\boldsymbol{u_i}^T \boldsymbol{u_i} = 1$ to (6.40) concludes the proof of Theorem 6.4. □

Remark 6.4. *Considering the input, $\boldsymbol{x}$, only the first basis vector, $\boldsymbol{u_1}$, contributes to the computation of $\boldsymbol{h}(\boldsymbol{x})_p^m$ for $m \longrightarrow \infty$ due to repeated multiplication by $\{\boldsymbol{W}_p^l\}_{i=1}^m$. Hence, $\boldsymbol{h}(\boldsymbol{x})_p^m \in \mathbb{R}^n$ for the PlainNet incurs considerable information loss. From Definition 6.2 and Theorem 6.2, the columns of a data batch for the PlainNet, $\boldsymbol{H}(\boldsymbol{x})_p^m \in \mathbb{R}^{n \times N}$, are colinear and thus $\boldsymbol{H}(\boldsymbol{x})_p^m$ exhibits singularity.*

(b) Residual network (ResNet)

The next theorem relates the features conditioning of the ResNet as in Section 6.3.2 (b).

Theorem 6.5. *For an input vector $\boldsymbol{x} \in \mathbb{R}^n$ to a linear m-layer ResNet parameterized by $\theta_r = \{\boldsymbol{W}_r^m \in \mathbb{R}^{n \times n}, \cdots, \boldsymbol{W}_r^l \in \mathbb{R}^{n \times n}, \cdots, \boldsymbol{W}_r^2 \in \mathbb{R}^{n \times n}, \boldsymbol{W}_r^1 \in \mathbb{R}^{n \times n}\}$, the hidden layer output, $\boldsymbol{h}(\boldsymbol{x})_r^m : m \longrightarrow \infty$, is*

$$\boldsymbol{h}(\boldsymbol{x})_r^m \approx \alpha_1 \boldsymbol{u_1} [(\sigma_{r_1}^2 + 1)^m - 1] + \sum_{i=1}^{n} \alpha_i \boldsymbol{u_i}, \tag{6.41}$$

where σ_{r_1} and $\boldsymbol{u}_1$ are the first singular value component and first singular vector of $\boldsymbol{W}_r^l$, respectively.

Proof. Using (6.5), the final output of the ResNet can be written as

$$\boldsymbol{h}(\boldsymbol{x})_r^m = \prod_{l=1}^{m} (\boldsymbol{W}_r^l + \mathbf{I}) \boldsymbol{x}. \tag{6.42}$$

Again, applying Lemma 6.2 to (6.42) simplifies it to

$$\boldsymbol{h}(\boldsymbol{x})_r^m = (\boldsymbol{W}_r + \mathbf{I})^m \boldsymbol{x}. \tag{6.43}$$

Expanding $(\boldsymbol{W}_r + \mathbf{I})^m$ in (6.43) using binomial theorem, we have

$$\boldsymbol{h}(\boldsymbol{x})_r^m = ((\boldsymbol{W}_r)^m + m(\boldsymbol{W}_r)^{(m-1)} + m(m-1)(\boldsymbol{W}_r)^{(m-2)}/2 + \cdots + \mathbf{I})\boldsymbol{x}. \tag{6.44}$$

Furthermore, using Lemma 6.3 for (6.44) gives

$$\boldsymbol{h}(\boldsymbol{x})_r^m = \left(\sum_{i=1}^{n} \sigma_{r_i}^{2m} \boldsymbol{u_i} \boldsymbol{u_i}^T + m \sum_{i=1}^{n} \sigma_{r_i}^{2(m-1)} \boldsymbol{u_i} \boldsymbol{u_i}^T + m(m-1) \sum_{i=1}^{n} \sigma_{r_i}^{2(m-2)} \boldsymbol{u_i} \boldsymbol{u_i}^T/2 + \cdots + \mathbf{I} \right) \boldsymbol{x}. \tag{6.45}$$

Given $m \longrightarrow \infty$ and $\sigma_{r_1} > \sigma_{r_i} : 2 \leq i \leq n$, and using (6.32), (6.45) becomes

$$\boldsymbol{h}(\boldsymbol{x})_r^m \approx \left(\sigma_{r_1}^{2m} \boldsymbol{u_1} \boldsymbol{u_1}^T + m\sigma_{r_1}^{2(m-1)} \boldsymbol{u_1} \boldsymbol{u_1}^T + m(m-1)\sigma_{r_1}^{2(m-2)} \boldsymbol{u_1} \boldsymbol{u_1}^T/2 + \cdots + \mathbf{I} \right) \sum_{i=1}^{n} \alpha_i \boldsymbol{u_i}. \tag{6.46}$$

Applying $\boldsymbol{u_i}^T \boldsymbol{u_j} = 0$ for $i \neq j$ and $\boldsymbol{u_i}^T \boldsymbol{u_i} = 1$ to (6.46) yields

$$\boldsymbol{h}(\boldsymbol{x})_r^m \approx \alpha_1 \sigma_{r_1}^{2m} \boldsymbol{u_1} + m\alpha_1 \sigma_{r_1}^{2(m-1)} \boldsymbol{u_1} + m(m-1)\alpha_1 \sigma_{r_1}^{2(m-2)} \boldsymbol{u_1}/2 + \cdots + \sum_{i=1}^{n} \alpha_i \boldsymbol{u_i}. \tag{6.47}$$

Factorizing out $\alpha_1 \boldsymbol{u_1}$ gives

$$\boldsymbol{h}(\boldsymbol{x})_r^m \approx \alpha_1 \boldsymbol{u_1} \left(\sigma_{r_1}^{2m} + m\sigma_{r_1}^{2(m-1)} + m(m-1)\sigma_{r_1}^{2(m-2)}/2 + \cdots + 1 \right) - \alpha_1 \boldsymbol{u_1} + \sum_{i=1}^{n} \alpha_i \boldsymbol{u_i}. \tag{6.48}$$

Finally, applying binomial theorem again so that (6.48) can be compactly written completes the proof of Theorem 6.5. □

Remark 6.5. *Considering the input, $\boldsymbol{x}$, all the basis, $\{\boldsymbol{u_i}\}_{i=1}^n$ contribute to the computation of $\boldsymbol{h}(\boldsymbol{x})_r^m \in \mathbb{R}^n$ for $m \longrightarrow \infty$; repeated multiplication by $\{\boldsymbol{W}_r^l\}_{i=1}^m$ does not cause any basis to vanish. Subsequently, $\boldsymbol{h}(\boldsymbol{x})_r^m$ for the ResNet retains full information. From Definition 6.2 and Theorem 6.5, the columns of a data batch from the ResNet $\boldsymbol{H}(\boldsymbol{x})_r^m \in \mathbb{R}^{n \times N}$ are distinct, and thus $\boldsymbol{H}(\boldsymbol{x})_r^m$ is non-singular.*

Furthermore, the invulnerability of ResNet performance to layer deletion as noted in [**35**] can be explained by Theorem 6.5. In the worst case, where the deletion of all the m layer

weights sets $\sigma_{r_1} = 0$ in Theorem 6.5, we obtain $\boldsymbol{h}(\boldsymbol{x})_r^m = \sum_{i=1}^n \alpha_i \boldsymbol{u_i}$. Hence, $\boldsymbol{h}(\boldsymbol{x})_r^m = \boldsymbol{x}$, and there is no catastrophic performance.

(b) ResNeXt

The ResNeXt analysis is based on the discussion in Section 6.3.2 (c), and will employ the following lemmas.

Lemma 6.4. *Let the set of s independent Gaussian random matrices be $\{\boldsymbol{X}_1 \sim \mathcal{N}(\mu_1,\, \beta_1^2), \boldsymbol{X}_2 \sim \mathcal{N}(\mu_2,\, \beta_2^2), \cdots, \boldsymbol{X}_k \sim \mathcal{N}(\mu_k,\, \beta_k^2), \cdots, \boldsymbol{X}_s \sim \mathcal{N}(\mu_s,\, \beta_s^2)\}$, and $\boldsymbol{X}_s = \sum_{k=1}^s \boldsymbol{X}_k$; where μ_k and β_k^2 are the mean and variance of X_k, respectively. Then, it can be shown that*

$$\boldsymbol{X}_s \sim \mathcal{N}(\sum_{k=1}^s \mu_k,\, \sum_{k=1}^s \beta_k^2), \tag{6.49}$$

Proof. We rely on the method of characteristic functions [39] that sufficiently defines the probability distributions of real-valued random variables for this proof. Particularly, we leverage the fact that different random variables have distinct characteristic functions as supported by the *Inversion Formulae* [190, 55]. The characteristic function of an independent random variable $\boldsymbol{X}$, $\varphi(u)$, is given as

$$\varphi_{\boldsymbol{X}}(u) = \mathbb{E}(\exp^{iu\boldsymbol{X}}), \tag{6.50}$$

where $i = \sqrt{-1}$ as in the imaginary unit, $u \in \mathbb{R}$ is a parameter of the function, and $\mathbb{E}$ denotes expectation. Furthermore, it can be shown that the characteristic function of a normally distributed random variable $\boldsymbol{X} \sim \mathcal{N}(\mu,\, \beta^2)$ is

$$\varphi_{\boldsymbol{X}}(u) = \exp\left(iu\mu_{\boldsymbol{X}} - \beta_{\boldsymbol{X}}^2 u^2/2\right). \tag{6.51}$$

Subsequently, the sum of the sequence of s independent random Gaussian variables $\{\boldsymbol{X}_1 \sim \mathcal{N}(\mu_1,\, \beta_1^2), \boldsymbol{X}_2 \sim \mathcal{N}(\mu_2,\, \beta_2^2), \cdots, \boldsymbol{X}_k \sim \mathcal{N}(\mu_k,\, \beta_k^2), \cdots, \boldsymbol{X}_s \sim \mathcal{N}(\mu_s,\, \beta_s^2)\}$, $\boldsymbol{Y}$, can be ex-

pressed as the product of their respective characteristic functions as in

$$\varphi_{\boldsymbol{Y}}(u) = \prod_{k=1}^{s} \exp\left(iu\mu_k - \beta_k^2 u^2/2\right), \tag{6.52}$$

$$\varphi_{\boldsymbol{Y}}(u) = \exp\left(iu\sum_{k=1}^{s}\mu_k - \sum_{k=1}^{s}\beta_k^2 u^2/2\right). \tag{6.53}$$

The resulting characteristic function of $\boldsymbol{Y}$ shows that it is also a normally distributed variable as with any of its constituents $\boldsymbol{X}_k \sim \mathcal{N}(\mu_k,\, \beta_k^2)$, but with mean $\sum_{k=1}^{s}\mu_k$ and variance $\sum_{k=1}^{s}\beta_k^2$. □

Namely, Lemma 6.4 allows us to replace $\sum_{k=1}^{s}\boldsymbol{W}_g^{l,k}$ in (6.8) with a single matrix $\boldsymbol{W}_c^l$. As such, we can expresses (6.8) compactly as

$$\boldsymbol{h}(\boldsymbol{x})_c^l = (\boldsymbol{W}_c^l + \mathbf{I})\boldsymbol{h}(\boldsymbol{x})^{l-2}, \tag{6.54}$$

where $\boldsymbol{W}_c^l \sim \mathcal{N}(\sum_{k=1}^{s}\mu_k^l,\, \sum_{k=1}^{s}\beta_k^2)$, and μ_k^l and β_k^2 are the mean and variance of the distribution of $\boldsymbol{W}_g^{l,k}$, respectively . Note that $\boldsymbol{W}_c^l \in \mathbb{R}^{q\times q}$ similar to $\boldsymbol{W}_g^{l,k}$. Equipped with (6.54) for the ResNeXt, the next theorem is stated as follows.

Theorem 6.6. *For the input $\boldsymbol{x} \in \mathbb{R}^n$ to a linear m-layer ResNeXt parameterized by $\theta_c = \{\boldsymbol{W}_c^m \in \mathbb{R}^{q\times q}, \cdots, \boldsymbol{W}_c^l \in \mathbb{R}^{q\times q}, \cdots, \boldsymbol{W}_c^2 \in \mathbb{R}^{q\times q}, \boldsymbol{W}_c^1 \in \mathbb{R}^{q\times n}\}$, the hidden layer output, $\boldsymbol{h}(\boldsymbol{x})_c^m : m \longrightarrow \infty$, is*

$$\boldsymbol{h}(\boldsymbol{x})_c^m = \alpha_1 \boldsymbol{u_1}[(\sigma_{c_1}^2 + 1)^m - 1] + \sum_{i=1}^{q}\alpha_i \boldsymbol{u_i}, \tag{6.55}$$

where σ_{c_1} and $\boldsymbol{u}_1$ are the first singular value component and first singular vector of $\boldsymbol{W}_c^l$, respectively.

Proof. From (6.54), following a similar proof for the ResNet given in Theorem 6.5, we arrive at an analogous expression for the ResNeXt. □

Remark 6.6. *Similar to the ResNet, it is seen that $\{\boldsymbol{u_i}\}_{i=1}^n$ contributes to the computation of $\boldsymbol{h}(\boldsymbol{x})_c^m \in \mathbb{R}^q$ for $m \longrightarrow \infty$, as repeated multiplication by $\{\boldsymbol{W}_c^l\}_{i=1}^m$ does not cause any*

basis to vanish. Hence, the ResNeXt retains full information. Again, using Definition 6.2 and Theorem 6.6, the columns of a data batch for the ResNeXt $\boldsymbol{H}(\boldsymbol{x})_c^m \in \mathbb{R}^{q \times N}$ are different, and therefore there is no singularity.

Although the vulnerability of ResNeXt's performance to layer deletion has not been empirically studied in any work to the best of our knowledge, Theorem 6.6 that is analogous to Theorem 6.5, theoretically shows that the ResNeXt behaves in a similar way to the ResNet. Therefore, the ResNeXt circumnavigates catastrophic performance resulting from deleting m layer weights. That is, $\sigma_{c_1} = 0$ in Theorem 6.6, so that $\boldsymbol{h}(\boldsymbol{x})^m = \sum_{i=1}^{q} \alpha_i \boldsymbol{u_i}$, and thus $\boldsymbol{h}(\boldsymbol{x})^m = \boldsymbol{x}$.

6.4.3 Backpropagation: error gradients singularity

It is well known that the trainability of DNNs is subject to *well-conditioned* error gradients; otherwise, optimization is unlikely to converge. For the sake of simplicity, we use the following lemmas in the subsequent parts.

Lemma 6.5. *Let $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times p}$ be matrices. If $\boldsymbol{V}$ is singular, then their product, $\boldsymbol{Y} = \boldsymbol{XV}$, is also singular.*

Proof. Some row(s) or column(s) of $\boldsymbol{V}$ are linearly correlated. As such, $\exists$ a vector $\boldsymbol{u} \in \mathbb{R}^n$ with $\| \boldsymbol{u} \|_2 \neq 0$: $\boldsymbol{Vu} = 0$. Hence, $\boldsymbol{XVu} = 0$, and thus $\boldsymbol{XV}$ is singular. □

Lemma 6.6. *Let $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times p}$ be two matrices. If $\boldsymbol{X}$ and $\boldsymbol{V}$ are non-singular, then their product, $\boldsymbol{Y} = \boldsymbol{XV}$, is also non-singular.*

Proof. All row(s) or column(s) of both $\boldsymbol{X}$ and $\boldsymbol{V}$ are linearly uncorrelated. As such, $\nexists$ a vector $\boldsymbol{u} \in \mathbb{R}^n$ with $\| \boldsymbol{u} \|_2 \neq 0$ such that $\boldsymbol{Xu} = 0$ or $\boldsymbol{Vu} = 0$. Consequently, $\boldsymbol{XVu} \neq 0$, and thus $\boldsymbol{XV}$ is non-singular. □

(a) Plain network (PlainNet)

Definition 6.4. *Given the outputs of units in a hypothetical layer l is $\boldsymbol{H}(\boldsymbol{x})^l$, the local error*

gradient of units, $\boldsymbol{\Delta}^l$, that is connected to units in layer $(l+1)$ is of the form[1]

$$\boldsymbol{\Delta}^l = \boldsymbol{H}(\boldsymbol{x})^l \left(\boldsymbol{W}^{l+1}\boldsymbol{\Delta}^{l+1}\right). \tag{6.56}$$

For training cost, E, the weight update for $\boldsymbol{W}^l$ at iteration t using the learning rate η is

$$\boldsymbol{\Delta W}^l(t) = -\eta\frac{\partial E}{\partial \boldsymbol{W}^l} = \eta\Delta^l\boldsymbol{H}(\boldsymbol{x})^{l-1}, \tag{6.57}$$

$$\text{with} \quad \boldsymbol{W}^l(t) = \boldsymbol{W}^l(t-1) + \boldsymbol{\Delta W}^l(t). \tag{6.58}$$

Given that DNN parameters by convention are randomly initialized and are generally quite far from the solution, large weight updates are made at the start of training. As such, if the total number of iterations is T, we can write

$$\boldsymbol{W}^l(t) \approx \boldsymbol{\Delta W}^l(t) \quad : 1 \leq t \ll T. \tag{6.59}$$

Remark 6.7. *Using Remark 6.1 or Remark 6.4 that $\boldsymbol{H}(\boldsymbol{x})^l$ exhibits singularity and Lemma 6.5, it can be concluded from* (6.56) *that the error gradient estimate, $\boldsymbol{\Delta}^l$, is singular. Subsequently, via a similar argument, $\Delta\boldsymbol{W}^l$ in* (6.57) *is singular, and so is $\boldsymbol{W}^l$ in* (6.59)*. Thus, optimization exhibits numerical instability [49, 26].*

From Remark 6.7, DNN computational results lose precision, and errors can accrue to cause significant erratic weights updates, and therefore non-convergence. This observation is interesting, as it directly corroborates the work in [7] that studied error gradients as white noise in very deep PlainNets. Our experimental results indeed confirm this.

(b) ResNet and ResNeXt

Remark 6.8. *Using Remark 6.2 or Remark 6.5 that $\boldsymbol{H}(\boldsymbol{x})^l$ is non-singular and Corrollary 6.1 that $\boldsymbol{W}^{l+1}$ is non-singular, it follows from Lemma 6.6 and* (6.56) *that the error gradient, $\boldsymbol{\Delta}^l$, is non-singular. Likewise, from* (6.57) *and* (6.59)*, weights updates and weights are both non-singular and decent estimates for successful optimization.*

[1] Given that the linear activation function is employed

Remark 6.9. *Given that $\boldsymbol{H}(\boldsymbol{x})^l$ for the ResNeXt is non-singular from Remark 6.3 or Remark 6.6 and $\boldsymbol{W}^{l+1}$ is non-singular from Corrollary 6.1, it follows from Lemma 6.6 and* (6.56) *that the error gradient, $\boldsymbol{\Delta}^l$, is also non-singular. Similar to ResNet, note the non-singularity of weight updates and weights in* (6.57) *and* (6.59)*, respectively, which contribute to a successful optimization.*

6.5 Theoretical Analysis of Skip connections for Model Generalization

In this section, we build on the findings in Section 4 for discussing how the learning characteristics of the PlainNet and models with skip connections, ResNet and ResNeXt, relate to model generalization. It is seen that the conditions of hidden layer representations not only contribute to the successful optimization of the models (as seen in Section 4), but also impact their generalization capacities. Namely, the relationship between the condition of the hidden representations and stability of solutions learned by the DNNs is given in the following proposition.

Proposition 6.4. *Assuming that the already learned optimal solution for a DNN is θ. A relative change of the hidden representation at layer l, $\Delta\boldsymbol{H}(\boldsymbol{x})^l$, translates to a relative solution change, $\Delta\theta$, as follows*

$$\frac{\parallel \Delta\theta \parallel}{\parallel \theta \parallel} \leq \kappa(\boldsymbol{H}(\boldsymbol{x})^l)\frac{\parallel \Delta\boldsymbol{H}(\boldsymbol{x})^l \parallel}{\parallel \boldsymbol{H}(\boldsymbol{x})^l \parallel} \ : 0 \leq l \leq m, \tag{6.60}$$

where $\theta = \{\boldsymbol{W}^l\}_{l=1}^m$ and $\boldsymbol{H}(\boldsymbol{x})^0 = [\boldsymbol{x_1}, ..., \boldsymbol{x_N}]$ for $l = 0$ is the input to the DNN.

Proof. See Section A4 in the appendix for proof sketch.

Proposition 6.4 shows that the robustness of the already learned solution, θ, depends on $\kappa(\boldsymbol{H}(\boldsymbol{x})^l)$. That is, for good generalization, $\boldsymbol{H}(\boldsymbol{x})^l$ with small condition numbers are favoured. We note that $\Delta\boldsymbol{H}(\boldsymbol{x})^l$ is analogous to the change, which manifests when a trained DNN is tested with novel data. Our expectation is that the solution already learned by the DNN gives good performance on the novel data, given that the novel data comes from the

same distribution as the data used for training the DNN. That is, the novel data for testing can be seen as only small changes in the individual data samples that constitute the training data.

6.5.1 PlainNet

The following remark summarizes the generalization behaviour of very deep PlainNets.

Remark 6.10. *From Remark 6.4, the singularity of $\boldsymbol{H}(\boldsymbol{x})^m$ for $m \longrightarrow \infty$ for the PlainNet, translates to a theoretically infinite $\kappa(\boldsymbol{H}(\boldsymbol{x})^m)$ in* (6.60)*. Hence, very small changes in the input data translates to extremely large changes in the solution. Importantly, the convergence of the PlainNet even on the training data, where data samples of the same class are slightly different is not guaranteed. Finally, model optimization almost certainly fails so that generalization is subsequently impossible.*

The optimization failures for the PlainNet on different datasets, which corroborates Remark 6.10 are reported in the experiment section.

6.5.2 ResNet and ResNeXt

Remark 6.11. *For the ResNet and ResNet, the non-singularity of the hidden representation $\boldsymbol{H}(\boldsymbol{x})^m$ for $m \longrightarrow \infty$ from Remark 6.5 and Remark 6.6 means a theoretically finite $\kappa(\boldsymbol{H}(\boldsymbol{x})^m)$ in* (6.60)*. Therefore, for moderate values of $\kappa(\boldsymbol{H}(\boldsymbol{x})^m)$, we expect that small changes in the input data result in negligible changes in the solution.*

Interestingly, experimental results show that both ResNet and ResNeXt operate with moderate condition numbers.
Furthermore, we study why the ResNeXt [236] mostly generalizes better than the ResNet [67]. For characterizing the subtle attribute of the ResNeXt that contributes to its improved generalization, we state the following theorem that relates the condition of a matrix to its dimension.

Theorem 6.7. *(Rudelson-Vershynin [179]) Given a standard Gaussian random matrix, $\boldsymbol{B} \in \mathbb{R}^{s\times z}$, with independent entries, the bounds on the distribution of its singular values is*

$$\sqrt{z} - \sqrt{s} \leq \mathbb{E}\,\sigma_{min}(\boldsymbol{B}) \leq \mathbb{E}\,\sigma_{max}(\boldsymbol{B}) \leq \sqrt{z} + \sqrt{s}, \tag{6.61}$$

where $\sigma_{min}(\boldsymbol{B})$ and $\sigma_{max}(\boldsymbol{B})$ are the minimum and maximum singular values of $\boldsymbol{B}$, respectively; $\mathbb{E}$ denotes expectation, and $s < z$.

Proof. See Theorem 2.6 in [179] for proof.

When $s = z$ so that the bound on $\sigma_{min}(\boldsymbol{B})$ in Theorem 6.7 becomes problematic, [179] further showed that the relation $\sigma_{min}(\boldsymbol{B}) \sim \sqrt{s} - \sqrt{s-1}$ suffices. Importantly, considering that both ResNet and ResNeXt do not suffer optimization problems as seen in Remark 6.8 and Remark 6.9, the following corollary is particularly useful for characterizing the improved generalization capacity of the ResNeXt over the ResNet as observed in the literature [**21**].

Corollary 6.3. *For an m-layer DNN, the condition numbers of the hidden representations at layer l in the ResNet and ResNeXt, $\boldsymbol{H}(\boldsymbol{x})_r^l$ and $\boldsymbol{H}(\boldsymbol{x})_c^l$, respectively, have the relation*

$$\kappa(\boldsymbol{H}(\boldsymbol{x})_r^l) > \kappa(\boldsymbol{H}(\boldsymbol{x})_c^l) : 1 \leq l \leq m. \tag{6.62}$$

Proof. We can deduce $\boldsymbol{H}(\boldsymbol{x})_r^l \in \mathbb{R}^{n\times N}$ and $\boldsymbol{H}(\boldsymbol{x})_c^l \in \mathbb{R}^{q\times N}$ from Remark 6.2 and Remark 6.6 for the ResNet and ResNeXt, respectively. Furthermore, the discussion in Section 6.3.2 (c) shows that by construction $q < n$. Consequently, applying Theorem 6.7 to $\boldsymbol{H}(\boldsymbol{x})_r^l$ and $\boldsymbol{H}(\boldsymbol{x})_c^l$ with $q < n < N$ shows the following relations (i) $\mathbb{E}\,\sigma_{max}(\boldsymbol{H}(\boldsymbol{x})_r^l) > \mathbb{E}\,\sigma_{max}(\boldsymbol{H}(\boldsymbol{x})_c^l)$, and (ii) $\mathbb{E}\,\sigma_{min}(\boldsymbol{H}(\boldsymbol{x})_r^l) < \mathbb{E}\,\sigma_{min}(\boldsymbol{H}(\boldsymbol{x})_c^l)$. Finally, the proof concludes by using (6.11) for the condition numbers of $\boldsymbol{H}(\boldsymbol{x})_r^l$ and $\boldsymbol{H}(\boldsymbol{x})_c^l$.

Remark 6.12. *Considering the ResNet and ResNeXt with $\boldsymbol{H}(\boldsymbol{x})_r^l \in \mathbb{R}^{n\times N}$ and $\boldsymbol{H}(\boldsymbol{x})_c^l \in \mathbb{R}^{q\times N}$, respectively, where $q < n$. Applying* (6.62) *to* (6.60) *shows that the ResNeXt exhibits a more stable solution than the ResNet.*

In the section of experiments, it is seen that the the ResNeXt generalizes better than the

Table 6.1: Results after training on the MNIST dataset

Model	Train accuracy	Test accuracy
PlainNet-110	11.20%	11.35%
ResNet-110	99.99%	99.57%
ResNeXt-110	99.99%	99.71%

ResNet, and the hidden representations of the ResNeXt have smaller condition numbers than the hidden representations of the ResNet.

6.6 Experiments

6.6.1 Datasets, settings and evaluations

For the experiments, MNIST [106], CIFAR-10 [99], CIFAR-100 [99] and ImageNet-2012 [145] datasets are used. All experiments except on MNIST dataset use standard data augmentation as in [254]. Very deep PlainNet, ResNet and ResNeXt with 110 layers are trained on MNIST, CIFAR-10 and CIFAR-100 datasets. For ImageNet-2012 dataset, only ResNet [67] and ResNeXt [236] models with 101-layers are trained. Mini-batch gradient descent, learning rate in the range [0.0001-0.1], momentum rate in the range [0.1-0.9], weight decay of 10^{-4}, batch size of 128 and a maximum of 200 epochs are used for training; all model parameters are initialized as random Gaussian tensors using [68]. All our experiments use the rectified linear units (ReLUs) to demonstrate the agreement between theoretical analysis and practical DNNs. Namely, we aim to observe for PlainNets, ResNet and ResNeXt, the following (i) evolution of units' activations with depth (ii) weights updates during training (iii) error gradients during training (iv) progressive near-singularity of model layer weights with depth, which is measured via condition number computed from singular values as in (8), and (v) information loss of hidden layer representations with depth via near-singularity (i.e. condition number). The singular values for model tensors are obtained using Higher Order Singular value decomposition (HOSVD) [14], which generalizes the well-known matrix SVD to n-dimensional arrays (i.e tensors).

Table 6.2: Results after training on the CIFAR-10 dataset

Model	Train accuracy	Test accuracy
PlainNet-110	10.26%	10.05%
ResNet-110	99.98%	93.29%
ResNeXt-110	99.98%	93.65%

Table 6.3: Results after training on CIFAR-100 dataset

Model	Train accuracy	Test accuracy
PlainNet-101	3.72%	3.68%
ResNet-101	99.84%	72.27%
ResNeXt-101	99.99%	72.43%

Table 6.4: Results (i.e. top-1 accuracy) after training on the ImageNet dataset

Model	Train accuracy	Test accuracy
PlainNet-101	15.37%	14.94%
ResNet-101	88.56%	77.32%
ResNeXt-101	89.74%	78.48%

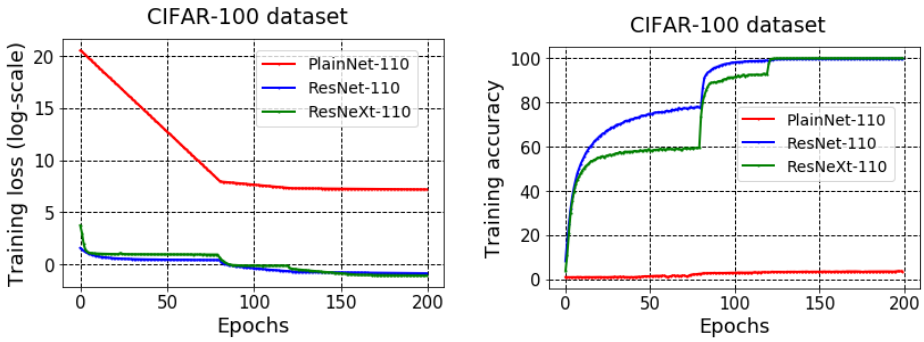


Figure 6.2: Training curves for models. Left: Training loss. Right: Training accuracy

6.6.2 Results and discussion

The experimental results of our investigation are discussed as follows. Tables 6.1, 6.2, 6.3 & 6.4 show the training and testing accuracies on the other three datasets. The unsuccessful training of the PlainNet is clearly reflected in the obtained training and testing accuracies. We note that hyperparameters search did not yield trainable PlainNets. Figure 6.2 and Figure 6.3 show the training and testing curves, respectively, for the different models trained on CIFAR-100 dataset. Note that the training and testing losses are plotted to log-scale due

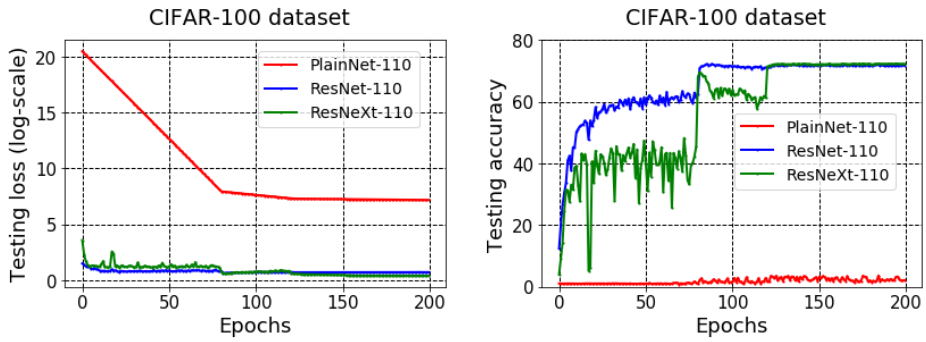


Figure 6.3: Testing curves for models. Left: Testing loss. Right: Testing accuracy

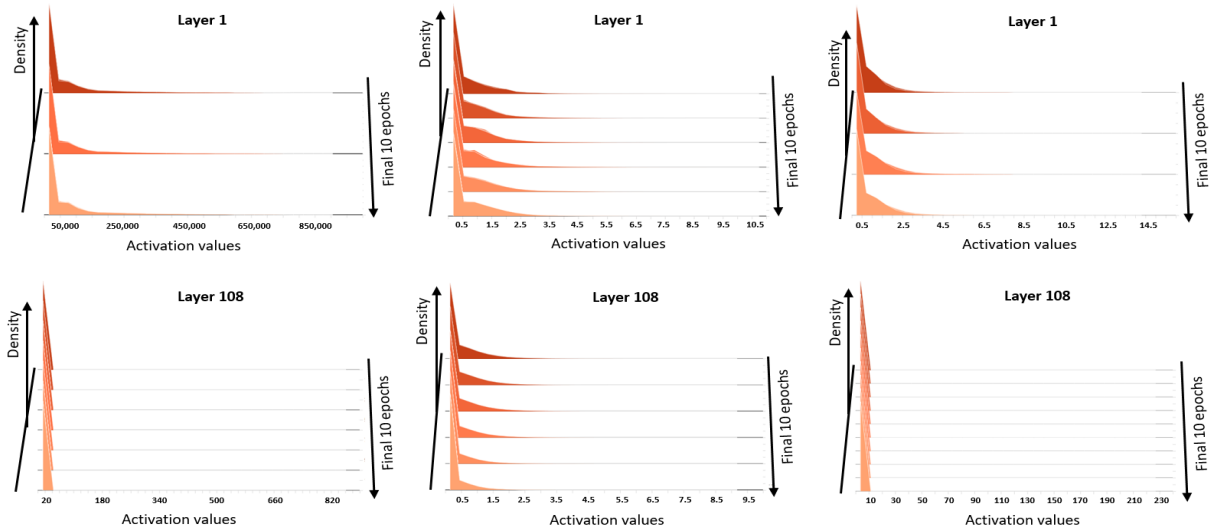


Figure 6.4: Units' activations for 110 layer models trained on the CIFAR-100 dataset. Left column: PlainNet. Middle column: ResNet. Right column: ResNeXt

to extremely high values for the PlainNet. The difficult of optimizing the PlainNet is evident in the extremely large training loss and poor training accuracy observed. Figure 6.4 shows the units' activations for PlainNet, ResNet and ResNeXt with 110 layers trained CIFAR-100 dataset. For observing the training conditions of the aforementioned DNNs, early and later layers are studied. Consequently, layers 1 and 108 are taken for inspection. It is seen in the layer 1 of the PlainNet that most units have very large activations (i.e. up to $\approx 10^6$) that are problematic. For ResNet and ResNeXt models, activations in layer 1 are less than 6.5. At layer 108, all models operate with reasonable units' activitions values.

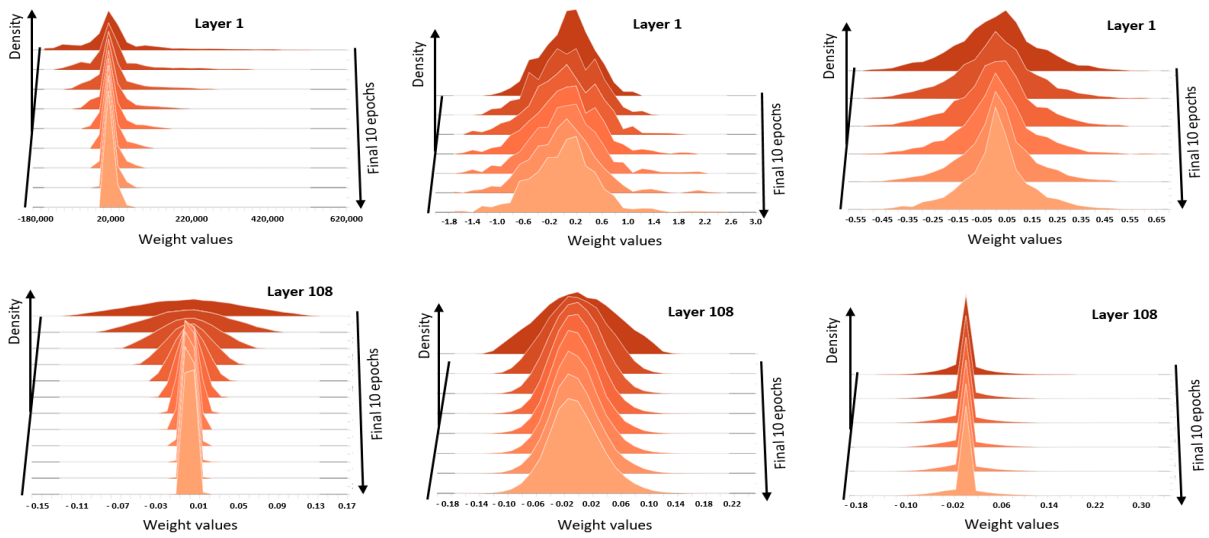


Figure 6.5: Units' weights for 110 layer models trained on the CIFAR-100 dataset. Left column: PlainNet. Middle column: ResNet. Right column: ResNeXt

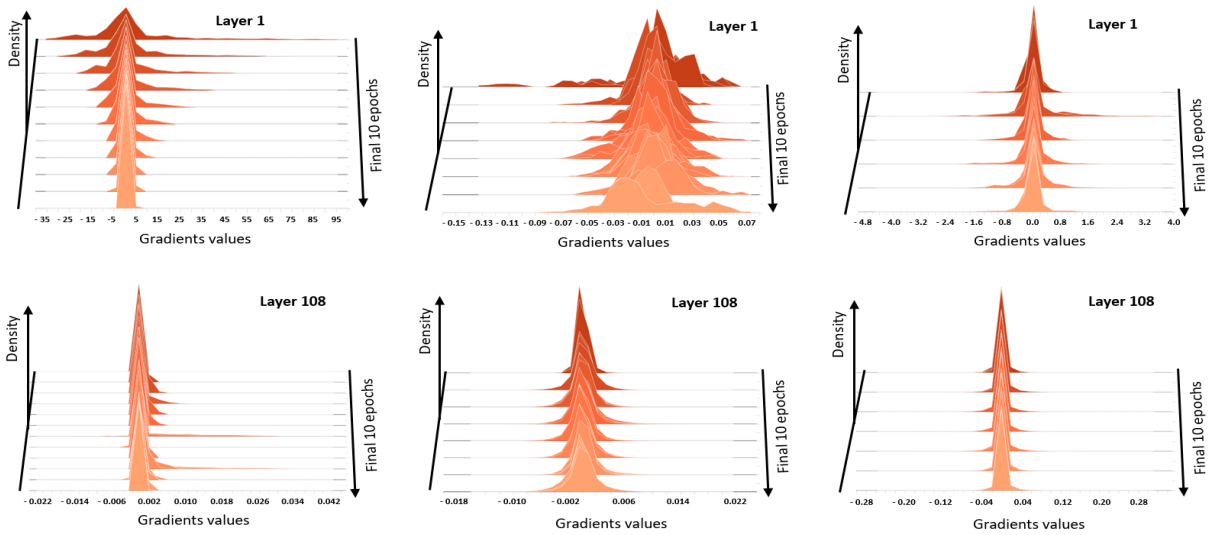


Figure 6.6: Units' gradients for 110 layer models trained on CIFAR-100 dataset. Left column: PlainNet. Middle column: ResNet. Right column: ResNeXt

Figure 6.5 shows the progress of weights update for the PlainNet, ResNet and ResNeXt given in Figure 6.4. For layer 1, it is seen that the PlainNets have weight values in the problematic range $-180,000$ to $620,000$. Conversely, ResNet and ResNeXt weights are in the reasonable ranges of -1.8 to 2.6 and -0.55 to 0.65, respectively. At layer 108, the PlainNet's

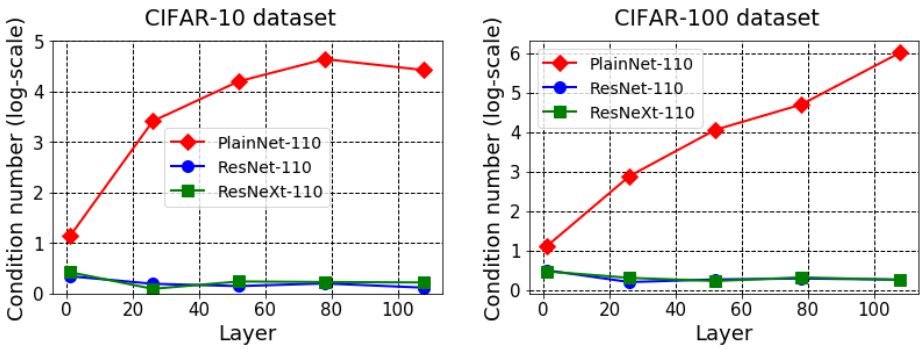


Figure 6.7: Model weights condition number. Left: CIFAR-10 dataset. Right: CIFAR-100 dataset

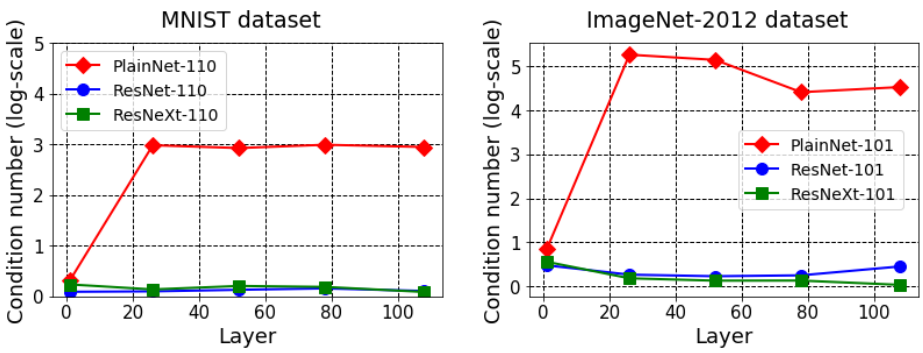


Figure 6.8: Model weights condition number. Left: MNIST dataset. Right: ImageNet dataset

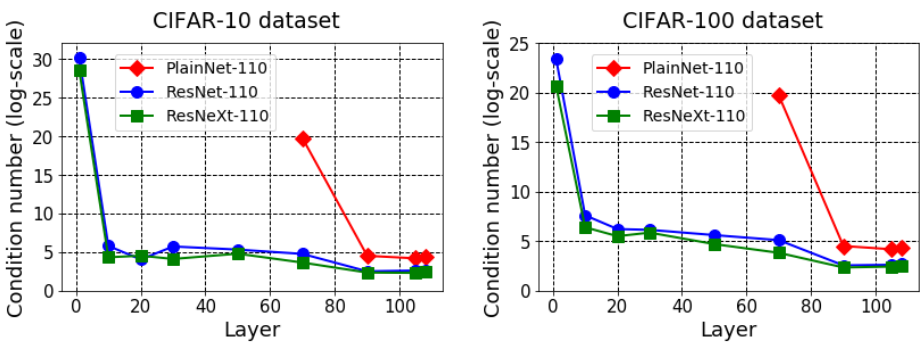


Figure 6.9: Model hidden layer condition number; layers with missing condition numbers have infinite values. Left: CIFAR-10 dataset. Right: CIFAR-100 dataset

weight values progressively collaps to zero, while the ResNet and ResNeXt weights both have fairly consistent values in the reasonable range -0.18 to 0.30.

Figure 6.6 shows model gradients. Most gradients in layer 1 of the PlainNet are considerably large in the range -5 to 5; large gradients depict optimization problems. However, ResNet and ResNeXt have most gradients in the reasonable range -0.07 to 0.07 and -0.8 to 0.8, respectively. The gradients for the PlainNet, ResNet and ResNeXt at layer 108 are at reasonable values of -10^{-2} to 10^{-2}. Note that Figure 6.4, Figure 6.5 and Figure 6.6 are unnormalized to directly convey the severity of the the training problems.

Figure 6.7 and Figure 6.8 show the condition number of model weights. It can be seen that the condition number of the PlainNet weights grows with depth, indicating that the learned model is progressively ill-posed with depth increase. Interestingly, the condition numbers of ResNet and ResNeXt weights remain small with depth increase; this indicates good stability of learned parameters in line with proposition 3. Figure 6.9 shows the condition number of hidden layer activations for the trained PlainNet, ResNet and ResNeXt. As such, the PlainNets have infinite condition numbers from layer 1 to 70. Note that infinite condition number indicates perfect singularity, and thus the worst scenario of optimization difficulty. The early layers in the ResNet and ResNeXt have high condition numbers (but no perfect singularity) that rapidly reduces to allow successful optimization.

6.6.3 Results for linear models

In Section 6.6.2, the reported experimental results use the rectified linear activation function (i.e. rectified linear units: ReLUs) for practicality reasons. That is, to show that our theoretical results in Section 6.4 and Section 6.5, which assumed the linear activation function are valid for the non-linear activation function. Herein, we perform additional experiments using the linear activation function for the different models trained on the MNIST dataset [106] to show that similar conclusions can be made. Specifically, we train the PlainNet, ResNet and ResNeXt with 164 layers using the linear activation function (i.e. with linear units). The condition numbers for the different layer weights in the trained models are shown in Figure 6.10. Similar to models with ReLUs, it is seen that the PlainNet layer weights have high

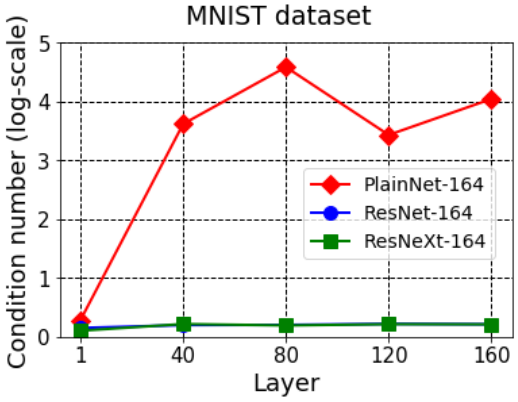


Figure 6.10: Condition number for layer weights in the models trained with the linear activation function

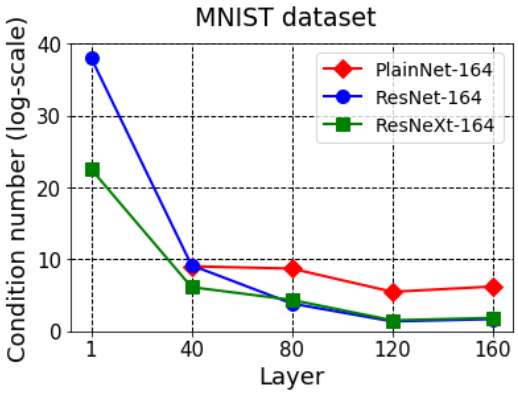


Figure 6.11: Condition number for hidden layer representations in the models trained with the linear activation function; layers with missing condition numbers have infinite values

condition numbers as compared to the ResNet and ResNeXt; compare Figure 6.10 with Figure 6.7 and Figure 6.8. Furthermore, Figure 6.11 shows the condition numbers for the different hidden layer representations in the trained models. Again, the PlainNet operates with high condition number as compared to the ResNet and ResNeXt. In fact, the hidden representation in the first layer of the PlainNet has an infinite condition number; this is the worst scenario for optimization problems to ensue. However, the ResNet and ResNeXt have finite condition numbers for the hidden representations in the different layers. Importantly, the condition of the hidden representations in the models with linear units is similar to the models with ReLUs; compare Figure 6.11 with Figure 6.9.

6.6.4 Skip connections in hindsight

(a) Training problem is beyond activation function: Although ReLUs are used for the reported experiments, similar training characteristics for units' activations, weights and gradients are observed when linear units are used. As such, the training problems of PlainNets are beyond the type of activation function employed.

(b) Successful training of PlainNet: We posit that working solutions for the successful training of very deep PlainNets would be related to weights regularization so that the largest eigenvalues are only slightly larger than others; this would alleviate information loss in the hidden representations that results from the domination of small eigenvalues by extremely large eigenvalues.

(c) Problem with the DiracNet [253]: The DiracNet claims to reslove the training problem of very deep PlainNet via a special initialization scheme. The weight initialization of an L-layer DiracNet, parameterized by $\widetilde{\boldsymbol{W}}^l \in \mathbb{R}^{n\times n} : 0 \leq l \leq L$, is

$$\widetilde{\boldsymbol{W}}^l = (\boldsymbol{W}^l + \text{diag}(\boldsymbol{a})) : 0 \leq l \leq L \tag{6.63}$$

where $\boldsymbol{W}^l$ is a randomly initialized matrix. e.g. as in [68]; and diag$(\boldsymbol{a}) \in \mathbb{R}^{n\times n}$ is the diagnoal matrix with diagonal entries $\boldsymbol{a} \in \mathbb{R}^n$. Subsequently, the output in the final layer of the DiracNet, $\widetilde{\boldsymbol{h}(\boldsymbol{x})}^L$, can be expressed as

$$\widetilde{\boldsymbol{h}(\boldsymbol{x})}^L = \prod_{l=1}^{L} \widetilde{\boldsymbol{W}}^l \boldsymbol{x}. \tag{6.64}$$

The DiracNet, which initializes $\boldsymbol{a} = \mathbf{1}$ mimics the ResNet at the start of training considering that (6.63) temporarily holds, since diag$(\boldsymbol{a})$ is the identity matrix. That is, the DiracNet behaves like it uses skip connections of identity mapping at the start of training. *Unfortunately, as training progresses,* $\widetilde{\boldsymbol{W}}^l$ *in* (6.63) *can deviate from the desired initialized setting* $\boldsymbol{a} = \mathbf{1}$*, since all the entries in* $\widetilde{\boldsymbol{W}}^l$ *are free parameters; there is no constraint on diag*$(\boldsymbol{a})$ *in* (6.63) *so that it remains the identity matrix during training.* As such, it is not surprising that the DiracNet's performance in [253] is worse than the ResNet, and the ResNet is the preferred

model for many tasks based on performance. In contrast, the DiracNet is rarely used in the literature. Interestingly, keeping the hidden layer representations close to the input has been posited as crucial for the successful training of very deep models [60]. We note that the DiracNet invariably violates diag$(\boldsymbol{a}) = \mathbf{I}$ in (6.63) after several training iterations so that $\widetilde{\boldsymbol{h}(\boldsymbol{x})}^L$ is far from $\boldsymbol{x}$ in (6.64), and thus optimization becomes difficult.

(d) Skip connections are immutable constraints on the geometric structure of layer weights: The physical skip connections seen in the ResNet [67] and ResNeXt [236] impose an immutable constraint on the geometric structure of layer weights, $(\boldsymbol{W}^l + \mathbf{I})$, *where* $\mathbf{I}$ *is always an identity matrix given that the entries of* $\mathbf{I}$ *are not free parameters by model construction, and hence does not change during training*. Consequently, $\boldsymbol{h}(\boldsymbol{x})^L$ stays close to the input $\boldsymbol{x}$ in (6.19), and hence optimization successfully converges as posited in [60].

6.7 Conclusion

Remarkable results on various computer vision tasks have been obtained using DNNs with skip connections. However, their mode of operation as compared to DNNs without skip connections (PlainNets) remains largely elusive. This chapter studies the unique properties of models with skip connections that eases optimization and improves generalization. Namely, important properties such as the information fidelity (i.e. usefulness) of computed hidden representation basis and error gradient, and stability of solutions based on the conditioning of learned model parameters are investigated. Our results reveal that the training problem of very deep neural networks is beyond the saturation of hidden units that is caused by nonlinear activation function. Without skip connections, very deep models exhibit singularity problems that plague optimization. However, with skip connections, the singularity problem is circumnavigated so that optimization successfully converges.

Chapter 7

Understanding Deep Neural Networks with Skip Connections and Concatenated Hidden Representations

Although there are ongoing theoretical works to understand very DNNs that employ the summation of layer outputs with the outputs of previous layers (e.g. residual network) similar to Chapter 6, there is none to the best of our knowledge on very DNNs that employ the concatenation of layer outputs with outputs of previous layers (e.g. Inception, FractalNet and DenseNet). As such, this chapter builds on Chapter 6 in that it presents the first theoretical analysis of very DNNs with concatenated hidden representations based on a general framework that can be extended to specific cases. Our results reveal the unique properties of DNNs with concatenated hidden representations that make them amenable to optimization, and on top of this generalize well. For substantiating the theoretical results, extensive experiments are reported on two standard datasets.

7.1 Introduction

Classical deep neural networks (DNNs) have only one path from the input layer to the output layer for information flow. This type of DNNs are commonly referred to as 'PlainNets', and many works [107, 194] have reported improved results on various tasks by simply extending the depth of previous state-of-the-art PlainNets. Following this observation, theoretical studies that characterize the role of model depth for the compact representation of complicated functions can be found in [48, 167, 181, 40]. Interestingly, training PlainNets with many layers of feature representations (i.e. very deep PlainNets) is difficult, since optimization typically becomes problematic when the number of model layers exceeds fifteen [162, 207]. However, it is known that the optimization problem of very dep PlainNets can be mitigated by employing skip connections of various form. A popular form of skip connection in the literature is based on the summation of the output of any given layer with the outputs of earlier layers. DNNs that adopt this type of construction include the ResNet [67], ResNeXt [236] and RoR [256]. While very impressive results for different tasks have been reported in several works using skip connections, the literature is still lacking of a concrete theoretical account of their operation. Considering that using skip connections with the summation of hidden representations results in complicated models, analytical studies have been limited to the ResNet. Notwithstanding, controversies such as the useful number of layers [221], how optimization problem is circumnavigated [221, 7, 60], and the source of improved generalization capacity [221] continue to trail the ResNet. As such, it is not surprising that the theoretical analysis of similar but more complicated models such as the ResNeXt [236] and RoR [256] are outrightly missing in the literature.

The other popular and successful form of skip connection in the literature entails the concatenation of the output of any given layer with earlier layers. DNNs that employ this type of construction are the Inception [210], Inception V3 [212], FractalNet [101] and DenseNet [77]. Despite the success of these DNN models, to the best of our knowledge, there are no theoretical works to understand their operation. This is not surprising, given the complicated architectural construction of this type of DNNs in comparison to the DNNs that employ skip

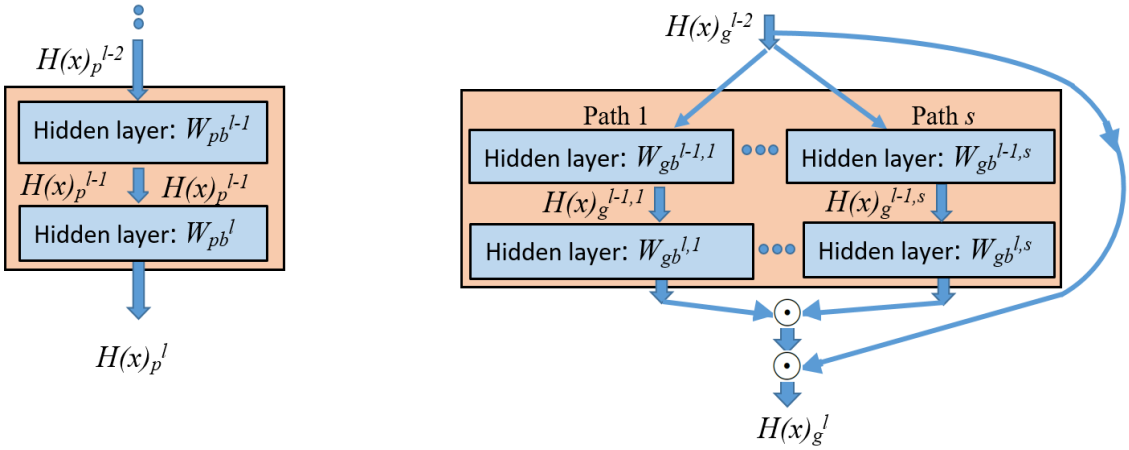


Figure 7.1: Generic DNN block used for analysis, where $\odot$ denotes the vertical concatenation operation. Left: PlainNet block. Right: ConcNeXt block.

connections and the summation of hidden representations.

In this chapter, we provide a theoretical framework for understanding the complicated operation of DNN models that employ skip connections with concatenated hidden representations. We acknowledge the proliferation of DNN architectures that employ skip connections and concatenated hidden representations in the literature [210, 212, 101, 77]. Hence, a generic DNN architecture that can be adapted to specific cases is employed for analysis in this chapter. Specifically, our generic architecture is similar to the ResNeXt, where the summation operation is replaced by the concatenation operation. We borrow the nomenclature of the ResNeXt so that the DNN architecture used in this chapter is referred to as 'ConcNeXt'. We note that the theoretical results for the generic DNN architecture used in this work are relatable to similar DNNs such as the Inception [210] and DenseNet [77]. Namely, our contributions are summarized as follows.

1. Present the first theoretical study of DNNs with concatenated hidden representations using elements of linear algbera and random matrix theory. Our results unravel unique model characteristics that translate to a good optimization condition for models with several hidden layers.

2. Report extensive experimental results to support the analytical results using bench-

marking datasets such as the MNIST and CIFAR-10.

The chapter is organized as follows. Section 7.2 discusses related works. Section 7.3 present the preliminaries on DNNs with skip connection and concatenated features using a generic framework. Theoretical analysis is given in Section 7.4. Experimental results with discussions are in Section 7.5. Section 7.6 concludes the chapter.

7.2 Related Work

Deep neural networks with skip connections: In recent times, there is a growing concern of the interpretability of DNNs, considering the proliferation of different model architectures. Particularly, we are interested in how the architectures of the DNNs impact their training characteristics and performace. Interestingly, the unconventional construction of the state-of-the-art DNNs, which employ various forms of skip connections obfuscate their operation. Nevertheless, we note that considerable progress has been made in understanding the operation of DNNs that employ the summation of the outputs of hidden layers with the outputs of the previous layers; particularly, the ResNet [67]. For instance, the work [221] argued that the effective depth of the ResNet is significantly smaller than the architectural depth. Namely, the ResNet with 110 layers was shown to have an effective depth of 17 layers [221]. The gradients shattering problem was studied in [7], where skip connections were found to be helpful for fostering well-structured gradients that make the training of very DNNs successful. The unrolled iterative estimation concept was proposed in [60, 69], where it was argued that groups of ResNet blocks iteratively refine the representations computed in a particular stage, and new representations are computed in other stages [60]. Other works from the dynamical systems perspective are in [23, 24]. Importantly, a unanimous account of the operation of the ResNet remains a challenge in the deep learning community. Subsequently, the literature is lacking of the theoretical study of DNNs that employ skip connections and the concatenation of the outputs of hidden layers with the outputs of previous layers. Some DNNs that use this type of construction include the Inception [210, 212], FractalNet [101] and DenseNet [77]. We presume that the main reason for the lack of analytical study for this

type of DNNs is their very complicated constructions and operations.

Studying deep linear neural networks: The strict theoretical study of DNNs with all its 'bells and whistles' often results in mathematical intractability. Among other simplifications that make DNN amenable to theoretical analysis, many works [256, 200, 141] assume the linear activation function. In fact, stricter assumptions such as the number of hidden units, data points and convexity are in the literature [200, 102]. Interestingly, it is known that the theoretical results obtained using linear DNNs are relatable to non-linear DNNs [91, 185]. This observation is not confounding, since a linear DNN with two or more layers is still a non-convex optimization problem in the parameter space.

7.3 Preliminaries

This section discusses as preliminaries the simplified and generic DNN architecture used for theoretical analysis in this work. Specifically, the building blocks for the PlainNet and ConcNeXt that the analyses are based on are shown in Figure 7.1, where we use $\odot$ to denote the vertical concatenation operation. Note that we assume the linear activation function for analysis. However, we show via experiments (in Section 7.5) that theoretical results are valid for practical models that use the non-linear activation function such as the rectified linear function. First, we give the following axiom that is important for expressing the hidden representations of the ConcNeXt in interesting forms.

Axiom 7.1. *Let a matrix $\boldsymbol{C} \in \mathbb{R}^{n \times N}$ be the vertical concatenation of matrices $\boldsymbol{A} \in \mathbb{R}^{n_1 \times N}$ and $\boldsymbol{B} \in \mathbb{R}^{n_2 \times N}$ as in $\boldsymbol{C} = (A \odot B)$; where $n = n_1 + n_2$. Assuming $\boldsymbol{A} = \boldsymbol{D}\boldsymbol{B}$ with $\boldsymbol{D} \in \mathbb{R}^{n_1 \times n_2}$, so that $\boldsymbol{C} = (\boldsymbol{D}\boldsymbol{B} \odot \boldsymbol{B})$. Then, we can write $\boldsymbol{C} = (\boldsymbol{D} \odot \mathbf{I})\boldsymbol{B}$, where $\mathbf{I} \in \mathbb{R}^{n_2 \times n_2}$ is the identity matrix.*

7.3.1 Plain network (PlainNet)

The output of the PlainNet block is strictly hierarchical as in Figure 7.1 (left). Let the input data to the PlainNet model be $\boldsymbol{X} \in \mathbb{R}^{n \times N}$. Subsequently, let the input to a hypothetical

PlainNet block composed of two hidden layers be $\boldsymbol{H}(\boldsymbol{X})_p^{l-2} \in \mathbb{R}^{n \times N}$; where $\boldsymbol{W}_{pb}^l \in \mathbb{R}^{n \times n}$ and $\boldsymbol{W}_{pb}^{l-1} \in \mathbb{R}^{n \times n}$ are the layers' parameters initialized as Gaussian random matrices, and pb indicates the weights in a PlainNet block. Thus, the output of the PlainNet block is

$$\boldsymbol{H}(\boldsymbol{X})_p^l = W_{pb}^l W_{pb}^{l-1} \boldsymbol{H}(\boldsymbol{X})_p^{l-2}. \tag{7.1}$$

For the sake of simplicity, the transformation $\boldsymbol{W}_{pb}^l \boldsymbol{W}_{pb}^{l-1}$ given in Eqn. (12.18) is lumped as $\boldsymbol{W}_p^l = \boldsymbol{W}_{pb}^l \boldsymbol{W}_{pb}^{l-1}$, where $\boldsymbol{W}_p^l \in \mathbb{R}^{n \times n}$. Hence, the output of the PlainNet block can be expressed as

$$\boldsymbol{H}(\boldsymbol{X})_p^l = \boldsymbol{W}_p^l \boldsymbol{H}(\boldsymbol{X})_p^{l-2}. \tag{7.2}$$

7.3.2 Concatened Network of hidden representations (ConcNeXt)

The ConcNeXt employs skip connections that concatenate the output of the previous block with the outputs of the s parallel paths in the current block, as in Figure 7.1 (right); $s \in \mathbb{N}^+$ is referred to as the cardinality of the ConcNeXt, where $1 \leq k \leq s$. As such, let the the weight matrix at layer l and path k be $\boldsymbol{W}_{gb}^{l,k} \in \mathbb{R}^{q \times q}$, where gb indicates the weights in the ConcNeXt block. Note that by construction, the parameterization of the ConcNeXt is such that $q \leftarrow q/s$ [236]. That is, the dimensions of the weight matrix is reduced with an increase in cardinality. For compactness, we use $\Upsilon_{k=1}^s \boldsymbol{\Gamma}^k$ to denote successive vertical concatenations of the variable $\boldsymbol{\Gamma}^k$, where the dimension of $\boldsymbol{\Gamma}^k$ permits the concatenation operations. That is, $\Upsilon_{k=1}^s \boldsymbol{\Gamma}^k = (\boldsymbol{\Gamma}^1 \odot \cdots \odot \boldsymbol{\Gamma}^k \odot \cdots \odot \boldsymbol{\Gamma}^s)$. Given the block's input, $\boldsymbol{H}(\boldsymbol{X})_g^{l-2} \in \mathbb{R}^{q \times N}$, the output of the ConcNeXt block, $\boldsymbol{H}(\boldsymbol{X})_{g_s}^l$, with the parameters initialized as Gaussian random matrices $\boldsymbol{W}_{gb}^{l,k}, \boldsymbol{W}_{gb}^{l-1,k} \in \mathbb{R}^{q \times q}$ is

$$\boldsymbol{H}(\boldsymbol{X})_g^l = (\Upsilon_{k=1}^s \boldsymbol{W}_{gb}^{l,k} \boldsymbol{W}_{gb}^{l-1,k} \boldsymbol{H}(\boldsymbol{X})_g^{l-2} \odot \boldsymbol{H}(\boldsymbol{X})_g^{l-2}), \tag{7.3}$$

Again, for simplicity, the transformation $\boldsymbol{W}_{gb}^{l,k} \boldsymbol{W}_{gb}^{l-1,k}$ in Eqn. (12.23) is lumped as $\boldsymbol{W}_g^{l,k} = \boldsymbol{W}_{gb}^{l,k} \boldsymbol{W}_{gb}^{l-1,k}$, where $\boldsymbol{W}_g^{l,k} \in \mathbb{R}^{q \times q}$. Hence, factorizing Eqn. (12.23) using Axiom 7.1 gives

$$\boldsymbol{H}(\boldsymbol{X})_g^l = (\Upsilon_{k=1}^s \boldsymbol{W}_g^{l,k} \odot \mathbf{I}) \boldsymbol{H}(\boldsymbol{X})_g^{l-2}. \tag{7.4}$$

7.4 Theoretical Study of DNNs with Skip Connections and Concatenated Hidden Represenations

This section theoretically investigates why DNNs with skip connections and concatenated hidden representations alleviate the training problems of very DNNs. The analysis of the ConcNeXt is positioned relative to the optimization problem of very deep PlainNets based on the singularity of hidden representations. First, the definitions, propositions and lemmas that are crucial for the theoretical study are given as follows.

Definition 7.1. *The condition number of a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times N}$ denoted $\kappa(\boldsymbol{A})$ is given as*

$$\kappa(\boldsymbol{A}) = \sigma_{max}(\boldsymbol{A}) / \sigma_{min}(\boldsymbol{A}), \tag{7.5}$$

where $\sigma_{max}(\boldsymbol{A})$ and $\sigma_{min}(\boldsymbol{A})$ are the maximum and minimum singular values of $\boldsymbol{A}$, respectively. From Definition 7.1, singularity implies that $\sigma_{min}(\boldsymbol{A}) = 0$, so that $\kappa(\boldsymbol{A}) = \infty$. Generally, the optimization of ill-posed problems are very difficult [139, 140].

Proposition 7.1. *Given a matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ whose column vectors, $\boldsymbol{w}_i$, are drawn from a Gaussian or uniform distribution, the probability that $\boldsymbol{W}$ is non-singular, $P(\boldsymbol{w_i} \notin W_{-i}^{span})$, is*

$$P(\boldsymbol{w}_i \notin W_{-i}^{span}) = 1 \ : 1 \leq i \leq n \tag{7.6}$$

Proof. See the proof for Proposition 6.1.

Corollary 7.1. *The initialization [68, 56] of an m-layer DNN weight matrices, $\{\boldsymbol{W}^l \in \mathbb{R}^{n \times n}\}_{l=1}^{m}$, follows Proposition 7.1, and hence are all non-singular. That is, $\sigma_{min}(\boldsymbol{W}^l) \neq 0 : 1 \leq l \leq m$.*

Proof. Proof follows from Proposition 7.1.

Corollary 7.2. *For an m-layer DNN, we have $0 < \sigma_{min}(\boldsymbol{W}^l) < \infty : 1 \leq l \leq m$. Hence, popular weights initialization schemes [68, 56] leads to $P(\sigma_{min}(\boldsymbol{W}^l) < 1) = 1 : 1 \leq l \leq m$.*

Proof. Formal proof follows from Corrollary 7.1. For further validation, empirical proof is given in Appendix A4.

Assumption 7.1. *The input to the m-layer DNN, $\boldsymbol{X} \in \mathbb{R}^{n \times N}$, is non-singular. This is an important assumption required for the validity of many machine learning algorithms.*

Lemma 7.1. *Let $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times p}$ be matrices. If $\boldsymbol{V}$ is singular, then their product, $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{V}$, is also singular.*

Lemma 7.2. *Let $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times p}$ be two matrices. If $\boldsymbol{X}$ and $\boldsymbol{V}$ are non-singular, then their product, $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{V}$, is also non-singular.*

Lemma 7.3. *(Solymosi [199]) Let $Z \in \mathbb{R}^{n \times n}$ be the set of non-singular matrices. Thus, for many pairs $X_1,\ X_2 \in Z$, the sum $Y = X_1 + X_2$ is non-singular.*

Proposition 7.2. *Assuming that the optimal solution for a DNN is $\boldsymbol{\theta}$. A relative change of the hidden representation at layer l, $\Delta \boldsymbol{H}(\boldsymbol{X})^l$, translates to a relative solution change, $\Delta\theta$, as follows*

$$\frac{\parallel \Delta \boldsymbol{\theta} \parallel}{\parallel \boldsymbol{\theta} \parallel} \leq \kappa(\boldsymbol{H}(\boldsymbol{X})^l) \frac{\parallel \Delta \boldsymbol{H}(\boldsymbol{X})^l \parallel}{\parallel \boldsymbol{H}(\boldsymbol{X})^l \parallel} \ : 0 \leq l \leq m, \tag{7.7}$$

where $\boldsymbol{\theta} = \{\boldsymbol{W}^l\}_{l=1}^m$ and $\boldsymbol{H}(\boldsymbol{X})^0 = \boldsymbol{X}$ for $l = 0$ is the DNN's input. We note that changes in the hidden representation $\parallel \Delta \boldsymbol{H}(\boldsymbol{X})^l \parallel$ can result from small variations intentionally (or inevitable noise) captured in the input data samples.

Proof. See the proof for Proposition 6.4.

Definition 7.2. *For an m-layer DNN, let the input to layer l be $\boldsymbol{H}(\boldsymbol{X})^{l-1}$, and the weight and error gradient at iteration t be $\boldsymbol{W}^l(t)$ and $\boldsymbol{\Delta}^l(t)$, respectively. The weight update for $\boldsymbol{W}^l$ at iteration $t+1$, $\boldsymbol{W}^l(t+1)$ is*

$$\boldsymbol{W}^l(t+1) = \boldsymbol{W}^l(t) + \eta \boldsymbol{\Delta}^l(t) \boldsymbol{H}(\boldsymbol{X})^{l-1} : 1 \leq l \leq m, \tag{7.8}$$

where η is the learning rate.

7.4.1 PlainNet (PlainNet)

Let the input of a linear m-layer PlainNet be $\boldsymbol{X} \in \mathbb{R}^{n \times N}$, and the hidden layers parameterized by $\boldsymbol{\theta}_p = \{\boldsymbol{W}_p^l \in \mathbb{R}^{n \times n}\}_{l=1}^m$. The output of the PlainNet in the last layer m, $\boldsymbol{H}(\boldsymbol{X})_p^m$, can be

written as

$$\boldsymbol{H}(\boldsymbol{X})_p^m = \prod_{l=1}^{m} \boldsymbol{W}_p^l \boldsymbol{W}_p^{l-1} \cdots \boldsymbol{W}_p^2 \boldsymbol{W}_p^1 \boldsymbol{X}. \tag{7.9}$$

Let $\gamma_{p_{min}}^m = \prod_{l=1}^m \sigma_{min}(\boldsymbol{W}_p^l)$ characterize the cummulative outcome of the product of the different layer weight minimum singular values, $\sigma_{min}(\boldsymbol{W}_p^l)$. Particularly, $\gamma_{p_{min}}^m$ allows the characterization of the singularity or near-singularity of the hidden representation $\boldsymbol{H}(\boldsymbol{X})_p^m$ in a DNN using Eqn. (12.25). Considering very deep PlainNets where $m \gg 1$, we have $\gamma_{p_{min}}^m \ll 1$ using Corrollary 7.2. Furthemore, depeding on m, we note that $\gamma_{p_{min}}^m$ can become extremely small such that insufficient machine floating point precision results in a rounding-off to zero. i.e. $\gamma_{p_{min}}^m = 0$. As such, the transformation caused by $\prod_{l=1}^m \boldsymbol{W}_p^l$ collapses space, and is thus singular. Subsequently, the result of the transformation of $\boldsymbol{X}$ based on $\prod_{l=1}^m \boldsymbol{W}_p^l$ in Eqn. (7.9) is singular using Lemma 7.1; that is, $\boldsymbol{H}(\boldsymbol{X})_p^m$ is singular, and $\kappa(\boldsymbol{H}(\boldsymbol{X})_p^m) = \infty$.

Remark 7.1. *From Proposition 7.2, $\| \Delta\boldsymbol{\theta}_p \| / \| \boldsymbol{\theta}_p \|$ is unbounded given $\kappa(\boldsymbol{H}(\boldsymbol{X})_p^m) = \infty$ for the PlainNet. Consequently, small changes in the hidden representation, $\Delta\boldsymbol{H}(\boldsymbol{X})^l$, in Proposition 7.2 are so magnified that optimization is haphazard in the solution space, $\boldsymbol{\theta}_p$. The constant and extreme fluctuation of θ means that optimization cannot progress, and convergence to any decent local minima is impossible. Since $\boldsymbol{H}(\boldsymbol{X})_p^m$ is singular, the term $\eta\boldsymbol{\Delta}^l(t)\boldsymbol{H}(\boldsymbol{X})^{l-1}$ in Definition 7.2 is singular using Lemma 7.1 so that weight update, $\boldsymbol{W}^l(t+1)$ can become badly conditioned; this observation is confirmed via experiments in Section 7.5. In constrast, $\gamma_{p_{min}}^m$ does not become extremely small for shallow PlainNets where typically $m < 20$, so that $\kappa(\boldsymbol{H}(\boldsymbol{X})_p^m) < a : a \in \mathbb{R}$. Hence, $\| \Delta\boldsymbol{\theta}_p \| / \| \boldsymbol{\theta}_p \|$ in Proposition 7.2 for the shallow PlainNet is bounded by a reasonable value, and the fluctuation of $\boldsymbol{\theta}_p$ due to $\Delta\boldsymbol{H}(\boldsymbol{X})^l$ is moderate so that shallow PlainNets can be successfully optimized. i.e. model converges.*

7.4.2 Concatened Network of aggregated hidden representations (ConcNeXt)

The theoretical analysis of the hidden representations of the ConcNeXt show that they are never singular. However, we first give the following important lemma and axiom that allows the characterization of the minimum singular value of the concatenation of any matrix and

the identity matrix.

Lemma 7.4. *Let a matrix $\boldsymbol{B} \in \mathbb{R}^{n \times n_2}$ be the vertical concatenation of any matrix $\boldsymbol{A} \in \mathbb{R}^{n_1 \times n_2}$ with $n_1 \geq n_2$ and the identity $\mathbf{I} \in \mathbb{R}^{n_2 \times n_2}$ as in $\boldsymbol{B} = (\boldsymbol{A} \odot \mathbf{I})$, so that $n = n_1 + n_2$. Let the singular values of $\boldsymbol{B}$ and $\boldsymbol{A}$ be $\{\sigma(\boldsymbol{B})_i\}_{i=1}^{n_2}$ and $\{\sigma(\boldsymbol{A})_i\}_{i=1}^{n_2}$, respectively. Then, it can be shown that $\sigma(\boldsymbol{B})_i = \sqrt{(\sigma(\boldsymbol{A})_i)^2 + 1} : 1 \leq i \leq n_2$.*

Proof. Let matrix $\boldsymbol{A} \in \mathbb{R}^{n_1 \times n_2}$ with $n_1 \geq n_2$ have the singular values $\{\sigma(\boldsymbol{A})_1, \cdots, \sigma(\boldsymbol{A})_i, \cdots, \sigma(\boldsymbol{A})_{n_2}\}$, so that the positive semidefinite matrix $\boldsymbol{A}^T\boldsymbol{A}$ have the eigenvalues $\{\lambda(\boldsymbol{A})_1, \cdots, \lambda(\boldsymbol{A})_i, \cdots, \lambda(\boldsymbol{A})_{n_2}\}$. Subsequently, $\boldsymbol{A}$ has singular values $\sigma(\boldsymbol{A})_i = \sqrt{\lambda(\boldsymbol{A}^T\boldsymbol{A})_i}$. Furthermore, given that $\boldsymbol{B} = [\boldsymbol{A} \oplus \mathbf{I}] = \begin{bmatrix} \boldsymbol{A} \\ \mathbf{I} \end{bmatrix}$, we have $\boldsymbol{B}^T\boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}^T & \mathbf{I}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{A} \\ \mathbf{I} \end{bmatrix} = \boldsymbol{A}^T\boldsymbol{A} + \mathbf{I}$. Noting that $\lambda(\boldsymbol{A}^T\boldsymbol{A})_i = (\sigma(\boldsymbol{A})_i)^2$, the singular values for $\boldsymbol{B}$, $\sigma(\boldsymbol{B})_i = \sqrt{(\sigma(\boldsymbol{A})_i)^2 + 1}$. The implication of Lemma 7.4 is that $\sigma_{min}(B) \geq 1 \because \nexists\, \sigma_{min}(A) < 0$.

Axiom 7.2. *Let a matrix $\boldsymbol{B} \in \mathbb{R}^{q \times z}$ be the vertical concatenation of s random matrices $\boldsymbol{A}_k \in \mathbb{R}^{q_k \times z} : 1 \leq k \leq s$, as in $\boldsymbol{B} = \Upsilon_{k=1}^{s} \boldsymbol{A}_k = (\boldsymbol{A}_1 \odot \cdots \odot \boldsymbol{A}_k \odot \cdots \odot \boldsymbol{A}_s)$; where $q = \sum_{k=1}^{s} q_k$. Then, we conclude that $\boldsymbol{B}$ is also a random matrix.*

Let the input of a linear m-layer ConcNeXt be $\boldsymbol{X} \in \mathbb{R}^{n \times N}$, and the hidden layers parameterized by $\boldsymbol{W}_g^1 \in \mathbb{R}^{q \times n}$ and $\{\boldsymbol{W}_g^l \in \mathbb{R}^{q \times q}\}_{l=2}^{m}$, so that $\boldsymbol{\theta}_g = \{\boldsymbol{W}_g^l\}_{l=1}^{m}$. Following Eqn. (12.24), the output of the ConcNeXt in the last layer m, $\boldsymbol{H}(\boldsymbol{X})_g^m$, can be written as

$$\boldsymbol{H}(\boldsymbol{X})_g^m = \prod_{l=1}^{m} (\Upsilon_{k=1}^{s} \boldsymbol{W}_g^{l,k} \odot \mathbf{I}) \boldsymbol{X}. \tag{7.10}$$

Using Axiom 7.2, $\Upsilon_{k=1}^{s} \boldsymbol{W}_g^{l,k}$ in Eqn. (7.10) can be aggregated in a compact form so that we have $\boldsymbol{W}_a^l = \Upsilon_{k=1}^{s} \boldsymbol{W}_g^{l,k}$. Therefore, Eqn. (7.10) can now becomes

$$\boldsymbol{H}(\boldsymbol{X})_g^m = \prod_{l=1}^{m} (\boldsymbol{W}_a^l \odot \mathbf{I}) \boldsymbol{X}. \tag{7.11}$$

Let $\gamma_{g_{min}}^m = \prod_{l=1}^{m} \sigma_{min}((W_a^l \odot \mathbf{I}))$. From Lemma 7.4, $\sigma_{min}((W_a^l \odot \mathbf{I})) \geq 1 : 1 \leq l \leq m$ in Eqn. (7.11). As such, $\gamma_{g_{min}}^m \geq 1$, and thus the overall transformation effect of $\prod_{l=1}^{m} (W_a^l \odot \mathbf{I})$

in Eqn. (7.11) does not collapse space, and thus is non-singular. Since $\boldsymbol{X}$ is non-singular from Assumption 7.1, we can conclude using Lemma 7.2 that $\boldsymbol{H}(\boldsymbol{x})_g^l$ in Eqn. (7.11) is non-singular too. i.e $\kappa(\boldsymbol{H}(\boldsymbol{x})_g^m) < a : a \in \mathbb{R}$.

Remark 7.2. *Considering $\kappa(\boldsymbol{H}(\boldsymbol{X})_g^m) < a : a \in \mathbb{R}$, $\| \Delta\boldsymbol{\theta}_g \| / \| \boldsymbol{\theta}_g \|$ in Proposition 7.2 for the ConcNeXt is bounded. Thus, small changes in the hidden representations, $\Delta\boldsymbol{H}(\boldsymbol{X})^l$, in Proposition 7.2 results in moderate changes in the current solution, $\boldsymbol{\theta}_g$. The stability of $\boldsymbol{\theta}_g$ means that optimization is successful, and decent local minima can be reached. Again, noting that $\boldsymbol{H}(\boldsymbol{X})_g^m$ is non-singular and $\boldsymbol{W}_g^l$ is non-singular using Proposition 7.1, the term $\eta\boldsymbol{\Delta}^l(t)\boldsymbol{H}(\boldsymbol{X})^{l-1}$ in Definition 7.2 is non-singular by applying Lemma 7.2. Finally, the weight update $\boldsymbol{W}^l(t+1)$ in Eqn.* (7.8) *is non-singular with a high probablity using Lemma 7.3. Our experiments indeed confirm these observations.*

7.5 Experiments

7.5.1 Experimental settings

This section reports the experimental results that validate the theoretical analysis given in Section 7.4 using 110 layers PlainNet and ConcNeXt models, which are multilayer perceptrons (MLPs) trained on the MNIST [106] dataset; see Table 1 for model architecture and parameters size.. We perform additional experimental results for convolutional neural network (CNN) models using CIFAR-10 [99] dataset. All model parameters are initialized from Gaussian distributions using the He method [167]. In addition, all models are trained using gradient descent, learning rate = [0.0001-0.1], momentum rate = 0.9, batch normalization (BN), weight decay of 10^{-4}, batch size of 128, and for 60 epochs. For showing the agreement between our theoretical analysis and practical models, the experiments in Section 7.5.2 use the rectified linear activation (i.e. nonlinear activation function). In Section 7.5.3, experiments using MLPs with the linear activation are reported.

7.5.2 Results for models with non-linear linear activation

(a) Multilayer Perceptron (MLP)

Training results are given in Table 7.1, where the PlainNet clearly reflects poor optimization. In contrast, the ConcNeXt-110 ($s = 1$) and ConcNeXt-110 ($s = 2$) are both well optimized. For the testing, the ConcNeXt-110 ($s = 2$) slightly outperforms ConcNeXt-110 ($s = 1$). The hidden representations for PlainNet-110 are shown in Figure 7.2 (top), where singularity (i.e. units respond in similar fashion for different input data) is seen. In contrast, ConcNeXt-110 ($s = 1$) in Figure 7.2 (bottom) shows no singularity problems (i.e. units respond in different manners for different input data); the hidden representations for ConcNeXt-110 ($s = 2$) is similar to that of ConcNeXt-110 ($s = 1$) so are not shown. Figure 7.3 and Figure 7.4 show the weight values and hidden units' ouputs in the different layers for the MLPs, respectively. It is seen that the PlainNet-110 have bizzarely high weight and units' output values, while the ConcNeXt-110 ($s = 1$) have reasonable weight and units' output values; the ConcNeXt-110 (s=2) have similar values to the ConcNeXt-110 ($s = 1$), and thus not shown.

Figure 7.5 and Figure 7.6 show the condition of layer weights and hidden representations of the MLPs, respectively. From Figure 7.5 (left), the layer weights of the PlainNet-110 have extremely high conditions number, while the hidden layer weights of the ConcNeXt-110 ($s = 1$) and ConcNeXt-110 ($s = 2$) both have small condition numbers. From Figure 7.5 (right), the hidden layer weights of the PlainNet-110 have zero singular values starting from the 20th layer, while the hidden layer weights of the ConcNeXt-110 ($s = 1$) and ConcNeXt-110 ($s = 2$) both have a minimum singular value of one for all the layers. Figure 7.6 (left) shows that the hidden representations of the PlainNet-110 have extremely high and even infinite condition numbers, while the ConcNeXt-110 ($s = 1$) and ConcNeXt-110 ($s = 2$) both have small condition numbers. For clarity, Figure 7.6 (right) shows that the hidden representations of the ConcNeXt-110 ($s = 2$) operates with smaller condition numbers as compared to the ConcNeXt-110 ($s = 1$). In the Section 7.6, we show empirically that the condition number of a random matrix generally grows with the matrix size.

(b) Convolutional neural Network (CNN)

In order to further validate the analytical results with empirical evidence, convolutional neu-

Table 7.1: 110 layers MLP details and results on the MNIST dataset.

MLP model	PlainNet-110	ConcNeXt-110 ($s = 1$)	ConcNeXt-110 ($s = 2$)
Hidden units per layer	180	50	25
Parameters	3.66M	3.92M	3.77M
Training accuracy	13.22%	100%	100%
Testing accuracy	12.45%	98.42%	98.83%

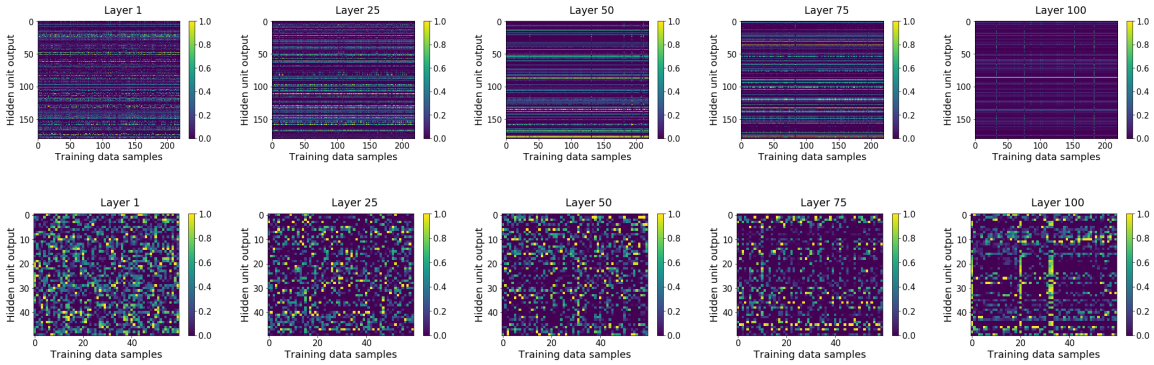


Figure 7.2: Hidden representations for the 110 layers MLPs using randomly chosen batch of input data from the MNIST dataset. Top row: PlainNet-110. Bottom row: ConcNeXt-110 ($s = 1$).

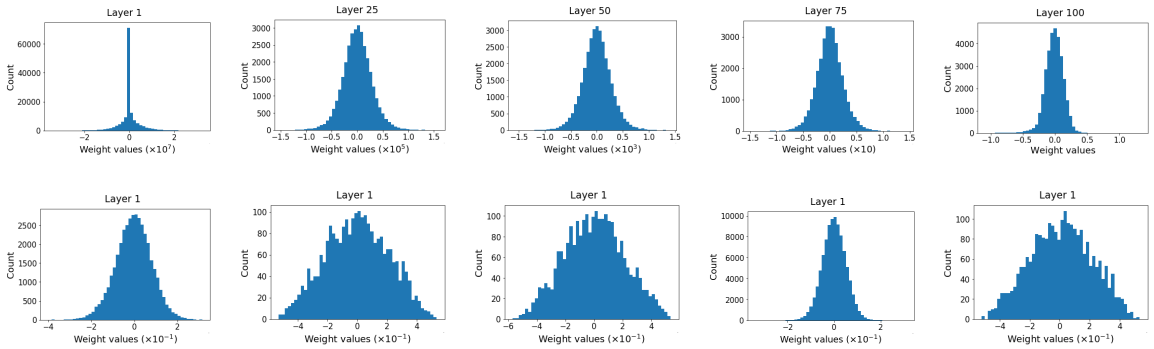


Figure 7.3: Weights distribution for the 110 layers MLPs using randomly chosen batch of input data from the MNIST dataset. Top row: PlainNet-110. Bottom row: ConcNeXt-110 ($s = 1$).

ral Network (CNN) with 110 layers are constructed for the PlainNet and ConcNeXt. The PlainNet is based on the ResNeXt [236], where the skip connections are eliminated. The ConcNeXt is also based on the ResNeXt, where the summation operation is replaced with the concatenation operation. For these models, experiments are performed using the harder

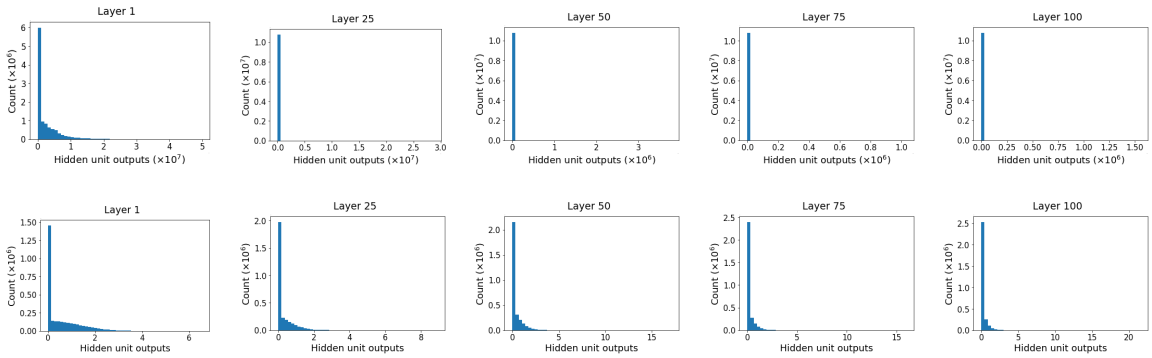


Figure 7.4: Units' outputs distribution for the 110 layers MLPs using randomly chosen batch of input data from the MNIST dataset. Top row: PlainNet-110. Bottom row: ConcNeXt-110 ($s = 1$).

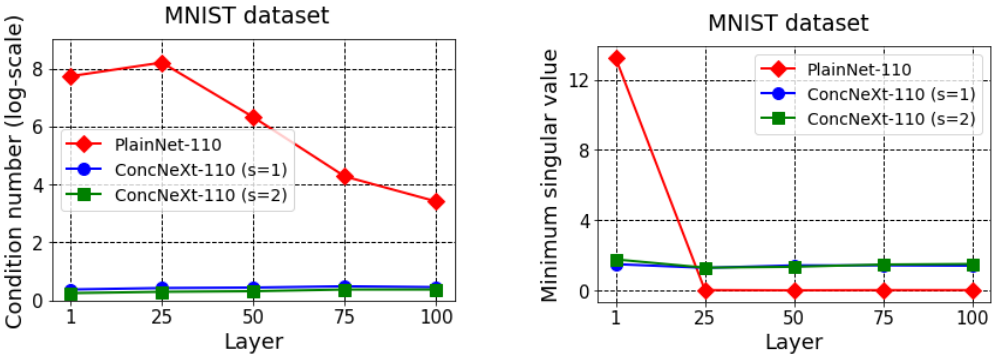


Figure 7.5: Condition of the layer weights in the MLPs. Left: condition number of the weights in the different layers *plotted to log-scale*. Right: smallest singular value of the weights in the different layers.

CIFAR-10 dataset. This allows us to again show that the PlainNet is untrainable, but the ConcNeXt models can be successfully trained. Furthermore, we again observe the improved generalization of the ConcNeXt ($s = 2$) over ConcNeXt ($s = 1$). All the CNN experiments employ rectified linear units (ReLUs), batch normalization (BN), weight decay of 1×10^{-4} and trained for 300 epochs. All layer weights are initialized from Gaussian distributions using the He initialization technique [68].

The hidden representations and weights in CNNs are tensors. As such, to obtain the singular values of the hidden representations and weights, we use Higher Order SVD (HOSVD) [14] that generalizes the conventional SVD for matrices to tensors. Finally, the condition numbers

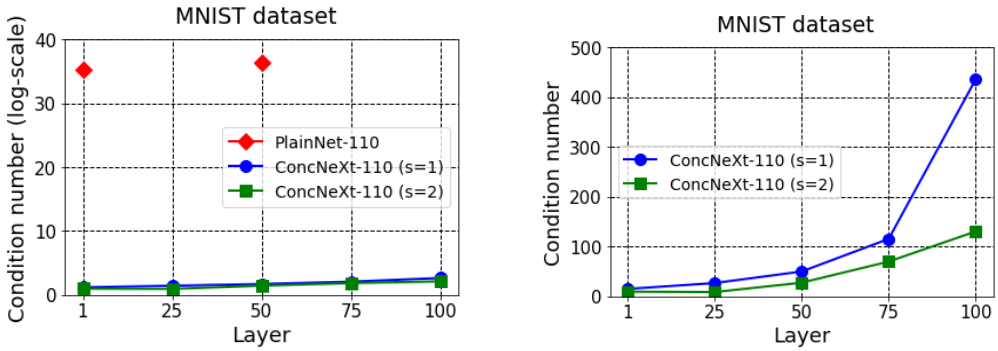


Figure 7.6: Condition number of the hidden layer representations in the MLPs. The layers for which values are not shown have infinite condition numbers. Left: condition number of the hidden representations in the different layers *plotted to log-scale*. Right: condition number of the hidden representations in the different layers of the ConcNeXt ($s = 1$) and ConcNeXt ($s = 2$).

Table 7.2: Model details and results of the CNN models with 110 layers trained on CIFAR-10 dataset.

CNN model	PlainNet-110	ConcNeXt-110 ($s = 1$)	ConcNeXt-110 ($s = 2$)
Parameters	1.74M	1.73M	1.51M
Training accuracy	10.26%	99.94%	99.95%
Testing accuracy	10.05%	94.51%	94.86%

for the hidden representations and weights of the models are computed using the ratio of the maximum singular values to the minimum singular values.

Table 7.2 shows the training results for the different CNN models on the CIFAR-10 dataset. It is observed that the PlainNet has poor training and testing accuracies, showing poor optimization. In contrast, the ConcNeXt-110 ($s = 1$) and ConcNeXt-110 ($s = 2$) models both have good training and testing accuracies; this shows successful optimization. Importantly, we note that ConcNeXt-110 ($s = 2$) again outperforms ConcNeXt-110 ($s = 1$) on the test accuracy. This observation is similar to that seen in the ResNeXt [236], where cardinality increase improves model generalization. Figure 7.7 (left) shows the condition numbers of the layer weights in the different CNN models, where it is seen that the PlainNet operates with very high condition numbers. In contrast, the ConcNeXt models operate with small condition numbers. Importantly, Figure 7.7 (right) shows the minimum singular values for the

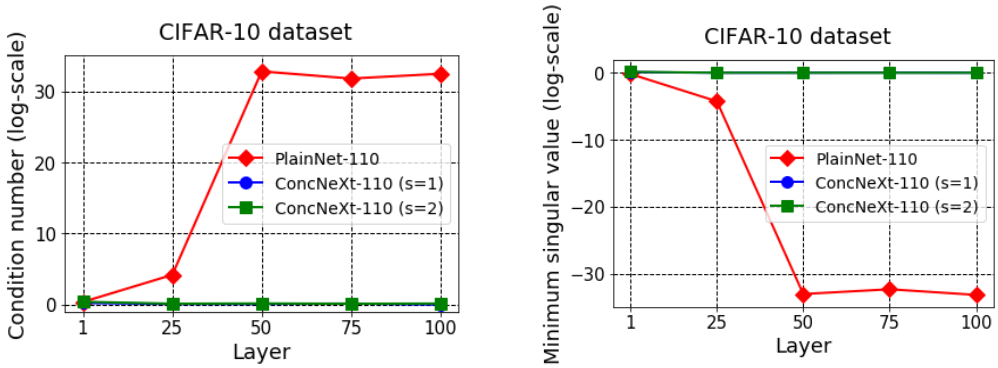


Figure 7.7: Condition of the layer weights in the CNNs. Left: condition number of the weights in the different layers. Right: smallest singular value of the weights in the different layers.

layer weights in the models; note that the figure is drawn to log-scale. Therefore, it is seen that the PlainNet weights have roughly zero minimum singular values starting from the 25th layer. However, the layer weights in ConcNeXt models have minimum singular values that are roughly one. In Figure 7.8, the condition numbers of the hidden representation of the different layers in the CNN models are shown. For the PlainNet, it is seen that the hidden representations for the 1st, 25th and 50th layers have infinite condition numbers, which is the worst scenario for optimization problem; the 75th has a moderate condition number, while the 100th layer has a very high condition number. In contrast, all the hidden representations in the different layers of the ConcNeXt models (i.e. for $s = 1$ and $s = 2$) have finite condition numbers. Furthermore, it is observed that the hidden representations in the ConcNeXt-110 ($s = 2$) have smaller condition numbers than ConcNeXt-110 ($s = 1$); this can again be related to the improved test accuracy of ConcNeXt-110 ($s = 2$) as compared to ConcNeXt-110 ($s = 1$). Further experimental results on the effect of matrix size on the condition number are discussed in Section A9.

7.5.3 Results for models with the linear activation function

Herein, the experimental results are obtained using the linear activation function for the different MLP models to show that similar results are obtained in comparision with the models that employed the rectified linear activation function. The same model architectures and

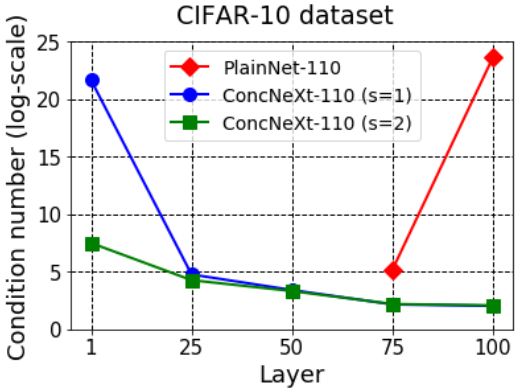


Figure 7.8: Condition number of the hidden layer representations in the 110 layers CNN models. The layers for which values are not shown have infinite condition numbers.

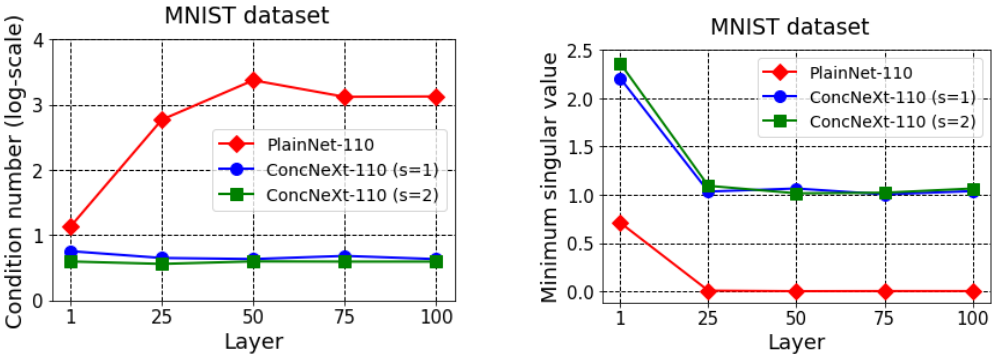


Figure 7.9: Condition of the layer weights in the 110 layers MLPs with the *linear activation function*. Left: condition number of the weights in the different layers *plotted to log-scale*. Right: smallest singular value of the weights in the different layers.

training settings as in the models with ReLUs are used. Figure 7.9 shows the condition of the weights in the different hidden layers for the MPL models. Similar to the models with ReLUs, the condition numbers of PlainNet layer weights are significantly higher than those for the ConcNeXt models; see Figure 7.9 (left). In addition, from Figure 7.9 (right), the minimum singular values for the layer weights in the ConcNeXt models are one. Importantly, these observations validate the bad conditioning of layer weights in very deep PlainNets. In contrast, the ConcNeXt circumnavigate this problem by being capable of operating with considerably smaller condition numbers.

7.6 Cardinality increase in ConcNeXt and model performance

We provide empirical evidence relating the distributions of the condition numbers of matrices with their sizes in Figure 7.10, where it is seen that the condition number of a matrix generally increases with its size. From a practical perspective, we make the following observations for neural networks layer weights in relation to performance.

On one hand, increasing the size of layer weights increases the number of model parameters, so that there is an increase in the degree of freedom for capturing the statistical patterns in the training data. On the other hand, our experimental results show that increasing the sizes of layer weights also increases the condition number of the weights, which may be undesirable from an optimization perspective. As such, a good technique for increasing the number of model parameters is increasing the ConcNeXt's cardinality s so that the number of parameterized paths per block is increased, but the size of the layer weights (matrices or tensors) in each path is unaltered. In contrast, keeping $s = 1$, and increasing the size of layer weights invariably results in condition number increase, which above some limit may hurt model performance.

7.6.1 Main insights into models concatenated hidden representations

The main insights from the theoretical and empirical results are summarized as follows.

1. The minimum singular values for PlainNet layer weights are typically less than one. i.e. $\sigma_{min}(\boldsymbol{W}_p^l) < 1 : 1 \leq l \leq m$. As such, the repeated multiplication of the input data $\boldsymbol{X}$ with the layer weights $\{\boldsymbol{W}_p^1, \cdots, \boldsymbol{W}_p^m\}$ that gives $\boldsymbol{H}(\boldsymbol{X})_p^m$ as seen in very DNNs (i.e. $m \gg 1$) causes some components of $\boldsymbol{X}$ to vanish so that collinearity and thus singularity occurs. Finally, singularity plagues model optimization.

2. Skip connections that concatenate hidden representations operates such that the minimum singular values for the layer weights is $\sigma_{min}(\boldsymbol{W}_g^l) \geq 1 : 1 \leq l \leq m$. Thus, the repeated multiplication of the input data $\boldsymbol{X}$ with the layer weights $\{\boldsymbol{W}_g^1, \cdots, \boldsymbol{W}_g^m\}$ that gives $\boldsymbol{H}(\boldsymbol{X})_g^m$ as seen in very DNNs (i.e. $m \gg 1$) does not cause any component of $\boldsymbol{X}$ to vanish so that collinearity and thus singularity are mitigated. Consequently, such models

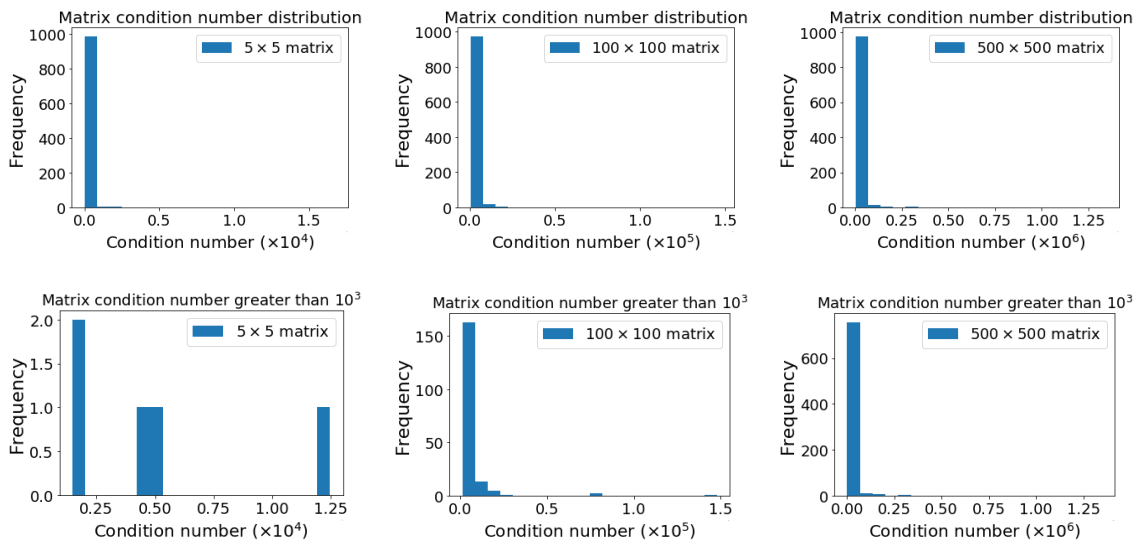


Figure 7.10: The distribution of the condition number of random matrices of different sizes with entries sampled from Gaussian distributions obtained using He method [68]; the experiment is repeated 1000 times. Top row: distribution of all condition numbers. Bottom row: distribution of condition numbers greater than 10^3, which improves the clarity of condition number increase with matrix size.

with several layers can be successfully optimized.

7.7 Conclusion

The successful training of very DNNs entails using skip connections of various forms. However, understanding why DNN architectures with skip connections mitigates optimization problems and generalize well is challenging. Specifically, the operation of DNNs that employ the concatenation of hidden representations is so complicated that their theoretical analysis is outrightly lacking in the literature. This chapter presents the theoretical analysis of DNNs with skip connections and concatenated hidden representations, which to the best of our knowledge is the first in the literature. Our theoretical results confirmed via extensive experiments show that the concatenation of hidden representations drastically improve the conditioning of the hidden representations by mitigating singularity problems of hidden representations and weights.

Chapter 8

Structured Compression of Deep Neural Networks with Debiased Elastic Group Lasso

In the preceding Chapters 3, 4 & 5, different approaches for improving the training of very DNNs are studied. However, most state-of-the-art Deep Neural Networks (DNNs) are typically too cumbersome to be practically useful in portable electronic devices. As such, model compression that seeks to drastically reduce computational memory footprints, FLOPS and memory for storage. Many of these works achieve unstructured compression, where the compressed models are not directly useful since dedicated hardware and specialized algorithms are required for storage of sparse weights and fast sparse matrix-vector multiplication respectively. In this chapter, we propose structured compression of large DNNs using debiased elastic group LASSO (DEGL), which is motivated by different interesting characteristics of the individual components. That is, where group LASSO penalty enforces structured sparsity, $L2$-norm penalty promotes features grouping, and debiasing disentangles sparsity and shrinkage effects of group LASSO. We perform extensive experiments by applying DEGL to different DNN architectures including LeNet, VGG, AlexNet and ResNet on several datasets. Furthermore, we validate the effectiveness of our proposal on domain adaptation using two

other small datasets. Results show that DEGL can compress DNNs by several folds with small or no loss of performance. Particularly, DEGL outperforms conventional group LASSO and several other state-of-the-art methods that perform structured compression.

8.1 Introduction

Computer vision tasks benefit from the success of DNNs, as several interesting results have been reported for different tasks [87, 171, 156]. Importantly, exceptional results on challenging datasets have been obtained using large DNN models [194, 100]. On one hand, there seems to be a positive correlation between model size and performance for many tasks [254, 158]; many works posit that large models allow the expoloration of extensive solution configurations, and thus reduces the possibly of getting stuck in poor local minima. On the other hand, these high-performance models are usually too cumbersome[1] for deployment in real-life applications. A solution to the aforementioned problems of large DNNs is to train from scratch smaller models with acceptable computational memory footprint, FLOPS and memory for storage. However, such small models typically incur noticeable generalization loss in comparison to large models. Consequently, model compression [196, 63] seeks to address the problems associated with cumbersome models, but incur minimal or even no performance loss in comparison to the reference large models.

Model compression results can be classified as unstructured or structured. In unstructured compression [204], the outcome is implicitly compressed, but the overall architecture of the compressed model is the same as the original (or reference) large model. Consequently, the aforementioned benefits of compression are not directly obtainable; dedicated hardware to store model weights with unstructured sparsity are required to realize a reduction in memory size, specialized algorithms are required for fast matrix-vector multiplications to realize a reduction in computational footprint and faster inference in comparison to the original large model [120]. In contrast, structured compression [272, 1] leads to an explicitly smaller model that requires no ad-hoc algorithms or specialized hardware for operation.

[1]The term 'cumbersome'is used to refer to the computer memory size and computational footprint

Model compression works [204, 120] have directly taken inspiration from *sparse signal processing* [275, 218]. The convergence and transferability of key concepts in signal processing to neural networks (NNs) is not surprising, since one can consider NNs as directed graphs with information flowing through them. Several works [120, 186] on unstructured and structured model compression have relied on enforcing some form of sparsity in model parameters by incorporating various penalty terms in the model's cost function. Depending on the specific characteristics of the penalty term used, different forms of sparsity can be achieved for model parameters. It is noteworthy that these sparsity-enforcing penalty terms that are now used for model compression are essentially not new; they have existed in classical signal processing for several years [218, 166]. Very good results have been reported using such penalty terms for compressing DNNs, and thus their characteristics deserve to be investigated even further for improved results. For enforcing unstructured and structured sparsity, LASSO [218] and group LASSO [249] penalties, respectively, have been reported to yield interesting results. Motivated by the aformentioned drawbacks of unstructured model compression, we focus in this chapter on structured model compression based on group LASSO.

We start by examining the drawbacks of group LASSO (which simply reduces to LASSO in the case where each group is considered as one feature [78]) in the context of model compression. First is the problem of *LASSO saturation* [275, 46] that occurs when the number of model parameters is considerably larger than the number of training data points; here, the number of features that LASSO selects is at most the number of training data points. This poses a huge limitation for application in DNNs where it is customary that the number of model parameters are exponentially higher than available training data points. Second is that group LASSO does not work well with correlated features [275]; the work [250] discusses the problems of ordinary least squares (OLS), LASSO and group LASSO with correlated variables. If, for instance, sets of features form groups, LASSO 'randomly' selects one set from each group, and discards the rest [224, 238, 264]. Interestingly, features correlation (especially among hidden units or convolution filters) is highly prevalent in DNNs [89, 35]. Third is that group LASSO simultaneously enforces both feature selection (which is

desirable for model compression) and feature shrinkage (which is not so desirable) [275]; as such, model compression (via features selection) is entangled with model regularization (via feature shrinkage). Obviosuly, this is a concern since our actual interest lies in feature selection which characterizes truly important features (identified by non-zero valued weights), as other features (with zero valued weights) can be discarded without hurting model performance. In fact, the shrinkage effect of group LASSO can be so impactful such that it causes models to underfit training data [264].

Our proposal in this work is aimed at tackling the aforementioned problems encountered with group LASSO for compressing DNNs. The overall framework of the proposed compression method is shown in Figure 8.1. Namely, the contributions in this chapter can be summarized as follows:

1. Compression of DNNs using $L2$-norm penalty and group LASSO, where group LASSO enforces group sparsity of parameters, and $L2$-norm promotes grouping and selection stability of correlated features.

2. Model debiasing after pruning by eliminating group LASSO prior to the retraining phase. This simple step disentangles the interwoven impact of feature selection and shrinkage, and thus ameliorate interpretability.

3. Experimental validation on various DNN architectures using six benchmarking datasets. Improved results over conventional group LASSO and several state-of-the-art approaches are reported.

The rest of this chapter is organized as follows. In Section 8.2, related works are discussed. Section 8.3 presents as background conventional group LASSO and subsequntly the problem statement. Section 8.4 describes the proposed approach that addresses the problems highlighted in Section 8.3. Section 8.5 contains experimental results and discussions. The chapter is concluded in Section 8.6.

8.2 Related work

High performance DNNs are typically cumbersome for real-life applications. Consequently, several works [109, 72] have proposed different methods of compression.

In [246], neuron importance score was defined for assessing the impact of hidden neurons in trained models; the score takes into account the reconstruction of important responses in the layer before the classification layer. It is argued that global pruning methods yield better results as opposed to pruning methods that take into account only adjoining layers. Soft filter pruning was proposed in [70], where pruning is performed dynamically. During training, previously pruned filters can be included again and participate in learning if it becomes necessary.

Sparse convolutional neural network was proposed that in [120]. This work employed sparsity-inducing contraints to impose sparsity in model parameters during training. The original model is trimmed after training using some defined thresholds to determine inconsequential weights. Unfortunately, the compression is unstructured, and thus the reported results, despite being good, are not directly usable as discussed earlier. In [230], group LASSO is used for learning structured sparsity in DNNs; interesting results are reported given that they directly achieve model speedup during inference without recourse to custom algorithms. In a similar work [186], sparse group sparsity was proposed. The idea is similar in spirit with [120]; that is, imposing unstructured sparsity among model parameters. Again, the overall result leads to unstructured sparsity and thus suffers the aforementioned drawbacks. In [65], $L2$-norm penalty was used to enforce model parameters to have small values. Although good results were achieved, the sparsity of resulting models were unstructured. In another work [71], LASSO penalty is used for channel selection and then subsequent pruning. The work reported compact models with reduced model size and accelerated inference.

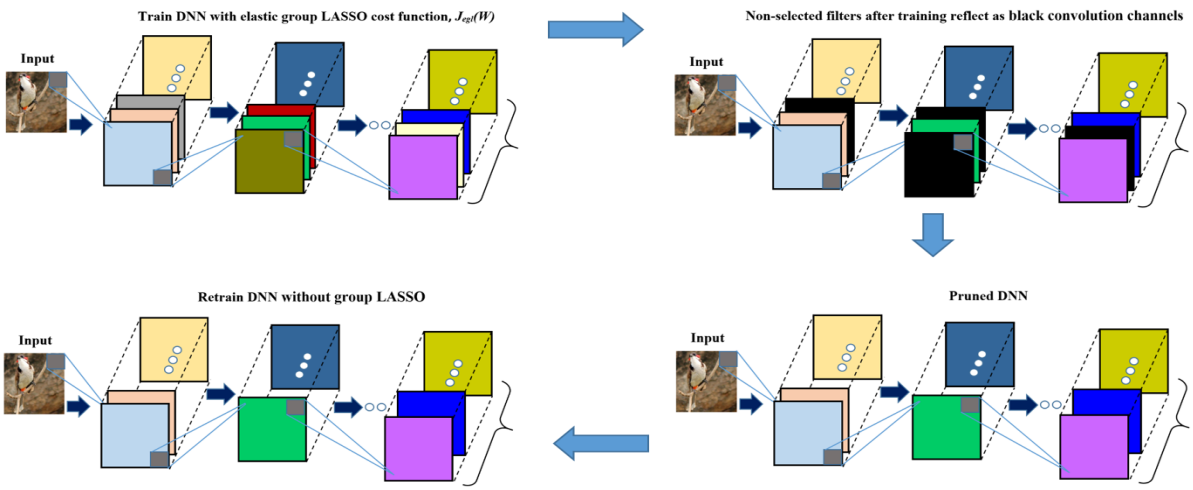


Figure 8.1: Overall framework for the proposed approach in Section 8.4. Top left: DNN is trained using the proposed elastic group LASSO cost function, $J_{egl}(W)$, in Section 8.4.1 (a). Top right: the non-selected filters after training translate to the black convolution channels. Bottom right: Non-selected filters are pruned, as in Section 8.4.1 (b). Bottom left: Pruned model is retrained without group LASSO, as in Section 8.4.1 (c)

8.3 Background and problem statement

This section discusses the simple premise that the application of conventional group LASSO can be suboptimal for the purpose of compressing DNNs.

8.3.1 Background

Consider the cost function, $J(\boldsymbol{W})$, for a DNN parameterized by W; where $W = \{\boldsymbol{W}_1, \boldsymbol{W}_2, ..., \boldsymbol{W}_l, .., \boldsymbol{W}_L\}$, and $\boldsymbol{W}_l$ is the weight matrix at layer l; and L is the total number of weight layers in the DNN. For instance, say $J(\boldsymbol{W})$ is the Mean Squared Error (MSE) given as

$$J(\boldsymbol{W}) = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{2N} \| \mathbf{y_d} - \mathbf{y_o} \|^2, \tag{8.1}$$

such that

$$\mathbf{y_o} = f(x; \boldsymbol{W}), \tag{8.2}$$

where x is the input data; f is the mapping function defined by the DNN; $\mathbf{y_d}$ and $\mathbf{y_o}$ are the desired and computed output vectors, respectively; and N is the number of training data points. Given that the group of features at layer l is denoted $\boldsymbol{W}_l^g$ and there are G groups, the new cost for the DNN with group LASSO, $J_{gl}(\boldsymbol{W})$, can then be written as

$$\left.\begin{aligned} J_{gl}(\boldsymbol{W}) = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W}), \quad \text{s.t.} \\ \| \boldsymbol{W}_l \|_2 = \sum_{g=1}^{G} \| \boldsymbol{W}_l^g \|_2 \leq c : \forall\, 1 \leq l \leq L. \end{aligned}\right\} \tag{8.3}$$

Note that each convolution filter or all the weights of a hidden unit are taken as a group; that is, g. In Langragian form, (3) can be written explicitly as

$$J_{gl}(\boldsymbol{W}) = \underset{\boldsymbol{W}}{\operatorname{argmin}} \left\{ J(\boldsymbol{W}) + \gamma \sum_{g=1}^{G} \sum_{l=1}^{L} \| \boldsymbol{W}_l^g \|_2 \right\}, \tag{8.4}$$

where the relationship between γ and c is such that $\gamma \propto {}^1\!/_c$.

8.3.2 Problem statement

We herein discuss the problems that can impact the application of group LASSO to the task of DNN compression.

(a) Saturation

For problems where the total number of model parameters, θ_t, is considerably larger than the number of training samples, N, LASSO does not select more than N features prior to saturation [275, 46]. This can severely impact the expressiveness of the resulting compact model. A formal treatment and proof of this scenario can be found in [177]. Interestingly, the case $\theta_t \gg N$ is extremely prevalent in DNNs, and thus deserves to be addressed when LASSO is employed for enforcing parameters sparsity.

(b) Correlated features and grouping

It has been shown that for correlated features, LASSO simply *randomly* selects a feature from the group and discards the rest [275]. This ultimately results in instability of feature se-

lection over different training runs. Coincedentally, DNNs are popular for learning highly correlated features [89, 35]. As such, for model compression techniques that employ LASSO, features selection instability is a concern in view of model interpretability.

(c) Entangled feature selection and shrinkage

An examination of LASSO penalty in [275, 218] reveals that LASSO performs both feature selection and shrinkage. This combined impact could be interesting for specific applications where alleviating model overfitting is the main objective; thus, being unable to identify their individual impact is of no concern. However, in model compression, our main goal is feature selection to facilitate interpretability and therefore compactness. As such, it becomes important to disentangle feature selection and shrinkage when LASSO is employed for model compression. Additionaly, LASSO shrinkage effect can over-penalize model parameters so that model underfits [264].

8.4 Proposed approach

We present in Figure 8.1 the proposed approach that addressesthe challenges discussed in Section 8.3.2. The objective is to obtain compact DNNs by employing a more interesting penalized cost function than the conventional group LASSO.

8.4.1 Debiased Elastic Group LASSO (DEGL)

We propose Debiased Elastic Group LASSO (DEGL) that aims to separate model selection via elastic group LASSO (EGL) from model estimation via ridge regression.

(a) Elastic group LASSO

Motivated by the success of elastic net [275], the problems of LASSO saturation and erratic selection for correlated features discussed in Sections 8.3.2 (a) and 8.3.2 (b) are addressed in this work by incorporating $L2$-norm penalty into the group LASSO cost given in (5). The Elastic group LASSO (EGL) cost function that is proposed to enforce group sparsity, promote

features grouping and alleviate selection instability is

$$\left.\begin{aligned} J_{egl}(\boldsymbol{W}) = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W}), \quad \text{s.t.} \\ \| \boldsymbol{W}_l \|_2 = \sum_{g=1}^{G} \| \boldsymbol{W}_l^g \|_2 \leq c_1 : \forall\, 1 \leq l \leq L, \\ \| \boldsymbol{W}_l \|_2^2 \leq c_2 : \forall\, 1 \leq l \leq L. \end{aligned}\right\} \tag{8.5}$$

Again, in Langragian form, (5) can be combined and written explicitly as

$$J_{egl}(\boldsymbol{W}) = \underset{\boldsymbol{W}}{\operatorname{argmin}} \left\{ J(\boldsymbol{W}) + \gamma_g \sum_{g=1}^{G} \sum_{l=1}^{L} \| \boldsymbol{W}_l^g \|_2 + \lambda \sum_{l=1}^{L} \| \boldsymbol{W}_l \|_2^2 \right\}, \tag{8.6}$$

where γ_g addresses over-penalization of small feature groups by taking into account the length of each group, ρ_g, for any given γ as in

$$\gamma_g = \gamma \sqrt{\rho_g}. \tag{8.7}$$

(b) Pruning unimportant filters

Given a specific filter m out of M filters in layer l denoted $\boldsymbol{W}_l^m$, we assess its importance by computing the maximum value of its individual absolute weight values, $w_l^{m_k}$ as in

$$max(\boldsymbol{W}_l^m) = max\{| \, w_l^{m_k} \, |\}_{k=1}^{K}, \tag{8.8}$$

where k indexes the individual weights in filter $\boldsymbol{W}_l^m$. The set of unimportant filters in layer l that least contribute to model performance is denoted $\boldsymbol{W}_l^p$. Given the pruning threshold t_{th}, W_l^p is initialized as an empty set, and then populated from $\boldsymbol{W}_l$ using the condition

$$max(\boldsymbol{W}_l^m) < t_{th} \ : \forall\, 1 \leq m \leq M. \tag{8.9}$$

Hence, $\boldsymbol{W}_l^p \subseteq \boldsymbol{W}_l$, and the set of remaining filters after pruning layer l is denoted $\boldsymbol{W}_l^r = \boldsymbol{W} \backslash \boldsymbol{W}_l^p$. Similar procedure can be repeated for all weight layers so that the overall resulting trimmed model is now parameterized by $\boldsymbol{W}^r$ as in $\boldsymbol{W}^r = \{\boldsymbol{W}_1^r, ..., \boldsymbol{W}_l^r, ..., \boldsymbol{W}_L^r\}$.

(c) Debiasing elastic group LASSO

To tackle the problem of entangled feature selection and shrinkage discussed in Section 8.3.2 (c), we completely eliminate group LASSO from the cost function given in (10) by setting $\gamma = 0$, and retraining the trimmed model now using the cost function, $J_{degl}(\boldsymbol{W}_r)$, given as

$$J_{degl}(\boldsymbol{W}_r) = \underset{\boldsymbol{W}_r}{\operatorname{argmin}} \left\{ J(\boldsymbol{W}_r) + \lambda_{rt} \sum_{l=1}^{L} \| \boldsymbol{W}_l^r \|_2^2 \right\}, \tag{8.10}$$

where λ_{rt} denotes the new $L2$-norm penalty weight for retraining; and $\lambda_{rt} = s\lambda : 0 < s \leq 1$, since $\boldsymbol{W}_r \subseteq \boldsymbol{W}$; s is chosen considering the size of $\boldsymbol{W}^r$. The new cost in (10) is similar in spirit to LARS-OLS [46] and relaxed LASSO [134].

8.4.2 Evaluation metrics

The different metrics used for evaluating the performance of the proposed approach are discussed as follows.

Model parameters

We consider a DNN with L weight layers, out of which L_{cnv} are convolution layers. Given a convolution weight layer $\boldsymbol{W}_l \in \mathbb{R}^{f_l^w \times f_l^h \times c_{l-1}^n \times c_l^n}$ that receives an input $\boldsymbol{H}(\boldsymbol{X})_{l-1} \in \mathbb{R}^{b_s \times c_{l-1}^w \times c_{l-1}^h \times c_{l-1}^n}$ and output $\boldsymbol{H}(\boldsymbol{X})_l \in \mathbb{R}^{b_s \times c_l^w \times c_l^h \times c_l^n}$, where w_l^f, h_l^f, c_{l-1}^n, c_l^n are the filter width, filter height, number of input feature channels and output feature channels, respectively; b_s, c_{l-1}^w, c_{l-1}^h are the batch size, incoming feature channel width and height, respectively. The number of parameters in a convolution layer l, θ_l^{cnv}, can be obtained using

$$\theta_l^{cnv} = f_l^w f_l^h c_{l-1}^n c_l^n. \tag{8.11}$$

For a fully connected layer l, the hyperparameters f_l^w, f_l^h, c_{l-1}^w, c_{l-1}^h, c_{l-1}^n, c_l^w, c_l^h, c_l^n can all be set to 1, so that the number of parameters, θ_l^{fc}, is given as

$$\theta_l^{fc} = c_{l-1}^n c_l^n. \tag{8.12}$$

Hence, the total number of model parameters, θ_t, is

$$\theta_t = \sum_{l=1}^{L_{cnv}} \theta_l^{cnv} + \sum_{l=1}^{L-L_{cnv}} \theta_l^{fc}. \tag{8.13}$$

Float point operations per second (FLOPS)

Required Float point operations per second (FLOPS) is a very useful criterion for assessing how fast a model runs. The number of FLOPS for a convolution layer l, $FLOPS_{cnv}^{l}$, can be obtained using

$$FLOPS_l^{cnv} = b_s f_l^w f_l^h c_{l-1}^n c_l^w c_l^h c_l^n. \tag{8.14}$$

Similarly, the number of FLOPS for a fully connected layer l, referred to as $FLOPS_l^{fc}$, can be computed as

$$FLOPS_l^{fc} = b_s c_{l-1}^n c_l^n. \tag{8.15}$$

Thus, the total number of FLOPS for the model, $FLOPS_t$, is given as

$$FLOPS_t = \sum_{l=1}^{L_{cnv}} FLOPS_l^{cnv} + \sum_{l=1}^{L-L_{cnv}} FLOPS_l^{fc}. \tag{8.16}$$

Computational memory footprint (CMF)

Computational memory footprints (CMFs) of compression results are quite critical, as it describes the memory required to hold model parameters and units' activations; small CMFs are generally desirable. At a given layer l, the CMF for a convolution layer and a fully connected layer can be computed as CMF_l^{cnv} and CMF_l^{fc}, respectively,

$$CMF_l^{cnv} = (Q/8)(\theta_l^{cnv} + b_s c_l^w c_l^h c_l^n), \tag{8.17}$$

and

$$CMF_l^{fc} = (Q/8)(\theta_l^{fc} + b_s c_l^n), \tag{8.18}$$

where Q is the number of bits of precision as in 'float$Q$ '. The total CMF for the model, CMF^t, is

$$CMF_t = \sum_{l=1}^{L_{cnv}} CMF_l^{cnv} + \sum_{l=1}^{L-L_{cnv}} CMF_l^{fc}. \tag{8.19}$$

Model memory size (MMS)

The compression results are also evaluated in view of the required memory storage, since one of the critical objectives of model compression is to obtain models that require considerably less memory for storage. MMS is reported in megabytes (MB) or kilobytes (KB) in this work.

8.5 Experiments

8.5.1 Main experiments

We perform extensive experiments using six benchmarking datasets. The results of our proposed DEGL are compared to conventional group LASSO and several state-of-the-art methods. Note that no comparison is made with $L2$-norm penalty and LASSO since they do not on their own yield structured sparsity for compression.

We train the different models in this work using mini-batch gradient descent; batch size for all experiments is chosen in the range 256-512. The initial learning rate, l_r for all experiments are chosen in the range 0.001 to 0.1; l_r is reduced by a factor of 0.1 during training whenever the training loss fails to improve for 5 epochs. Different suitable values for γ and λ in (10) are used to enforce the desired sparsity levels and grouping effects respectively in the models. For pruning, different values of t_{th} (as in (9)) are used to obtain different levels of compression; larger values for t_{th} translate to more compact models. Note that DEGL and group LASSO are both retrained after pruning. For all results, 'Error $\uparrow$ ', 'FLOPS $\downarrow$ ' and 'Param. $\downarrow$ ' denote error rate increase, FLOP percentage decrease and Parameters

percentage decrease, respectively. Error $\uparrow$ is calculated as in

$$Error \uparrow \ = Error_{ac} - Error_{bc}, \tag{8.20}$$

where $Error_{ac}$ and $Error_{bc}$ are the error rates after and before compression, respectively. *Thus, a negative value for 'Error $\uparrow$ ' actually shows a decrease in error rate after compression*. The evaluated metrics for the reference (i.e. uncompressed) models are appended at the end of each table.

8.5.2 Results

LeNet-5 on MNIST

The MNIST[2] dataset contains 60K training and 10K testing samples, respectively. LeNet-5 [65] is a DNN with two convolution layers and three fully connected weight layers including the softmax layer; it has 431K parameters. LeNet-5 is trained for 100 epochs with $\gamma = 5 \times 10^{-3}$ and $\lambda = 10^{-4}$. Table 8.1 shows the compression results of LetNet-5, and comparison with state-the-of-art methods. DEGL1, DEGL2, DEGL3, DEGL4, DEGL5 are obtained by setting the pruning threshold values, t_{th} : 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2} and 2.5×10^{-2}, respectively. Figure 8.2 shows the CMF and MMS of compressed models using DEGL. For example, DEGL5 with 98.4% parameters pruned reduces CMF from 57MB to 27.58MB and MMS from 2180KB to 52KB, while achieving an impressive error of 0.73%. It is seen that EGL performs better than group-LASSO, but worse than DEGL.

VGG-16 and ResNet-56 on CIFAR-10

CIFAR-10[3] dataset contains 50K training and 10K testing images, respectively; there are 10 different classes. VGG-16 [43] model has 15M parametres that consist of 13 convolution layers and 2 fully connected weight layers. We apply DEGL to VGG-16 for 300 epochs using $\gamma = 1 \times 10^{-5}$ and $\lambda = 10^{-6}$. Compression results are given in Table 8.2, where DEGL1,

[2]http://yann.lecun.com/exdb/mnist/
[3]https://www.cs.toronto.edu/ kriz/cifar.html

Table 8.1: LeNet-5 compression results on MNIST dataset

Models	Error ↑	FLOPS ↓	Param. ↓
LTPWC [65]	-0.03%	83.7%	91.7%
AFP [43]	-0.06%	93.3%	93.0%
TSN [204]	0.01%	–	95.8%
LiM [268]	0.08%	–	98.4%
Group-LASSO [43]	0.00%	87.0%	86.4%
Group-LASSO [43]	0.20%	93.7%	93.4%
EGL	0.05%	96.1%	94.1%
EGL	0.14%	98.5%	98.4%
Ours: DEGL1	**-0.10%**	**96.1%**	**94.1%**
Ours: DEGL2	**-0.03%**	**96.9%**	**95.3%**
Ours: DEGL3	**-0.15%**	**97.9%**	**96.9%**
Ours: DEGL4	**-0.07%**	**98.3%**	**97.5%**
Ours: DEGL5	**0.01%**	**98.5%**	**98.4%**

Reference LeNet-5: Error = 0.80%, Param. = 431K, FLOPS = 4.6M

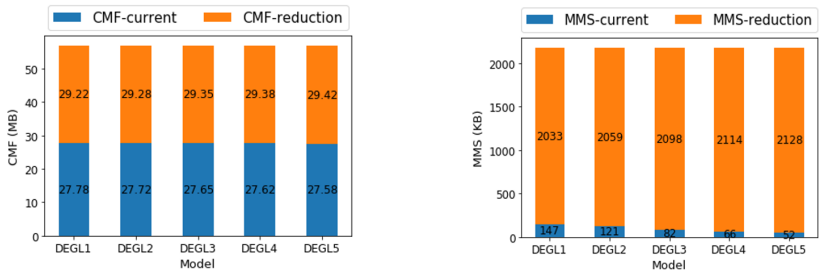


Figure 8.2: LeNet-5 CMF and MMS results on MNIST

Table 8.2: VGG-16 compression results on CIFAR-10 dataset

Models	Error ↑	FLOPS ↓	Param. ↓
AFP [43]	0.38%	81.4%	–
AFP [43]	0.31%	79.6%	–
Pruning [111]	-0.15%	34.2%	64.0%
L2PF [80]	1.90%	64.5%	86.5%
Group-LASSO	0.08%	86.9%	87.8%
Group-LASSO	0.23%	88.1%	90.0%
EGL	0.03%	87.0%	88.0%
EGL	0.31%	88.1%	90.0%
Ours: DEGL1	**-0.30%**	**85.6%**	**86.7%**
Ours: DEGL2	**-0.30%**	**87.0%**	**88.0%**
Ours: DEGL3	**-0.18%**	**88.1%**	**90.0%**

Reference VGG-16: Error = 6.75%, Param. = 15M, FLOPS = 313M

DEGL2 and DEGL3 are obtained by setting t_{th} : 10^{-4}, 10^{-3}, and 10^{-2}, respectively. Figure 8.3 shows the CMFs and MMMs for DEGL1, DEGL2 and DEGL3 in Table 8.2. For example, DEGL3 with 90.0% parameters pruned reduces CMF from 1.12GB to 0.81GB, and MMS

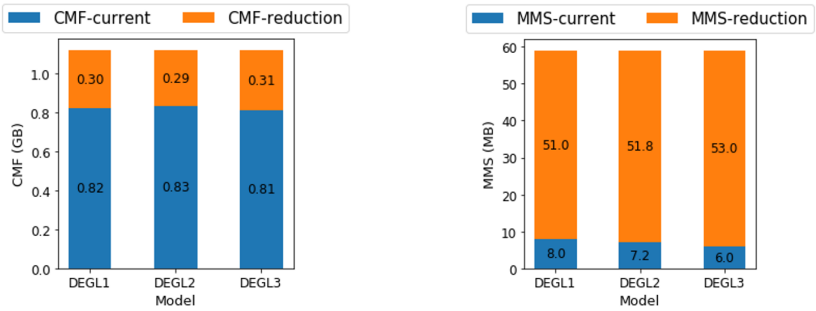


Figure 8.3: VGG-16 CMF and MMS results on CIFAR-10

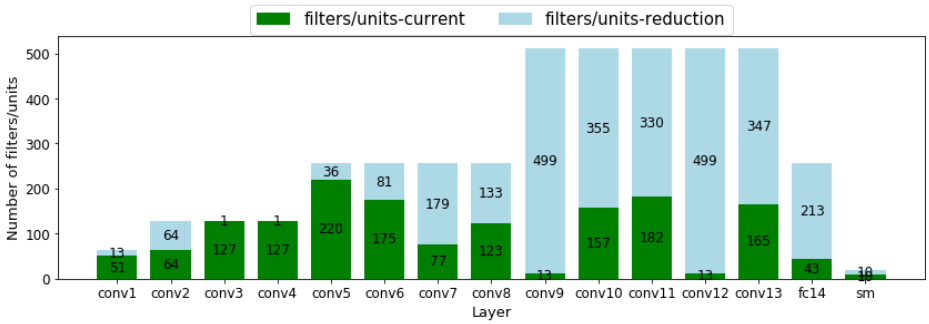


Figure 8.4: DEGL2 compression results for VGG-16 on CIFAR-10

Table 8.3: ResNet-56 compression results on CIFAR-10 dataset

Models	Error ↑	FLOPS ↓	Param. ↓
Pruning [111]	-0.02%	27.6%	13.7%
NISP [246]	0.03%	43.6%	42.6%
CNN-FCF [116]	-0.24%	42.78%	43.1%
KSE [158]	0.15%	60%	57.6%
Group-LASSO	0.03%	37.9%	44.3%
Group-LASSO	0.24%	47.1%	59.4%
EGL	0.05%	47.8%	44.7%
EGL	0.48%	53.4%	60.0%
Ours: DEGL1	**-0.37%**	**38.2%**	**31.8%**
Ours: DEGL2	**-0.31%**	**47.8%**	**44.7%**
Ours: DEGL3	**0.09%**	**53.4%**	**60.0%**

Reference ResNet-56: Error = 6.96%, Param. = 0.85M, FLOPS = 125M

from 59MB to 6MB, while even improving generalization. Figure 8.4 shows the number of current and pruned filters (or units) for DEGL2.

Results of DEGL using ResNet-56 on CIFAR-10 is given in Table 8.3, where DEGL1, DEGL2 and DEGL3 are obtained using the same t_{th} values as in VGG-16. We particularly observe

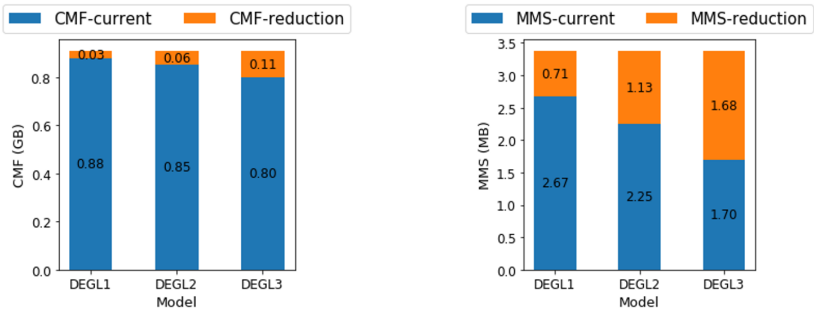


Figure 8.5: ResNet-56 CMF and MMS results on CIFAR-10

that DEGL outperforms conventional group LASSO for ResNet architectures, where skip connections can increase features correlations among different layers, and therefore conventional group LASSO is struggle with consistent feature selection; see Section 8.3.2 (b) for details. Figure 8.5 shows how pruning impacts CMF and MMS for models reported in Table 8.3. Importantly, for both VGG-16 and ResNet-56, DEGL outperforms both group-LASSO and EGL.

VGG-16 and ResNet-56 on CIFAR-100

CIFAR-100[3] dataset contains 50K training and 10K testing samples, respectively; the dataset composes 100 different classes. The same VGG-16 architecture used for CIFAR-10 is used for CIFAR-100; the model is trained for 350 epochs with $\gamma = 1 \times 10^{-5}$ and $\lambda = 10^{-4}$. Table 8.4 shows pruning results, where DEGL1, DEGL2 and DEGL3 are obtained by setting $t_{th}: \ 10^{-4}, \ 5 \times 10^{-3}, \ \text{and} \ 10^{-2}$, respectively. As CIFAR-100 is a more challenging dataset, and the effectiveness of DEGL compared to group LASSO and EGL clearly reflects. The impact of compression on CMF and MMS is reported in Figure 8.6. Figure 8.7 shows the current and pruned filters (or units) for DEGL3.

AlexNet and ResNet-50 on ImageNet

The ImageNet dataset[4] contains about 1.2M training and 50K testing samples, respectively. The dataset has 1000 different categories. AlexNet has 61M parameters that consist of

[4]http://image-net.org/challenges/LSVRC/2012/index

Table 8.4: VGG-16 compression results on CIFAR-100 dataset

Models	Error ↑	FLOPS ↓	Param. ↓
Group-LASSO	0.07%	84.0%	83.3%
Group-LASSO	0.11%	84.2%	85.6%
EGL	0.06%	84.5%	84.0%
EGL	0.14%	84.7%	86.0%
Ours: DEGL1	**-0.01%**	**81.7%**	**82.0%**
Ours: DEGL2	**0.00%**	**84.5%**	**84.0%**
Ours: DEGL3	**0.03%**	**84.7%**	**86.0%**

Reference VGG-16: Error = 27.41%, Param. = 15M, FLOPS = 313M

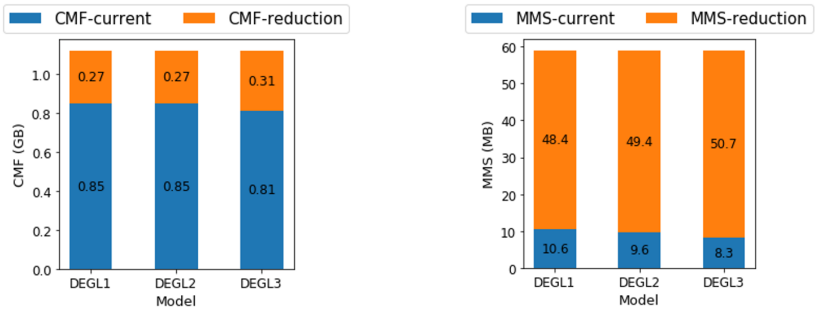


Figure 8.6: VGG-16 CMF and MMS results on CIFAR-100

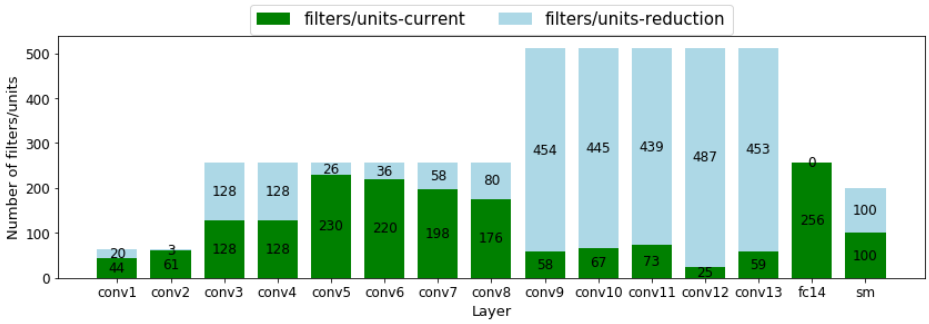


Figure 8.7: DEGL3 compression results for VGG-16 on CIFAR-100

5 convolution layers and 3 fully connected weight layers including the softmax layer. The model is trained for 70 epochs with $\gamma = 5 \times 10^{-6}$ and $\lambda = 10^{-4}$. ResNet-50 has 25.6M parameters, and the same values of γ and λ used for AlexNet are employed for training. ResNet50 is trained for 90 epochs, since it is much deeper than AlexNet.

The results of using the proposed DEGL for compressing AlexNet are given in Table 8.5, where DEGL1, DEGL2 and DEGL3 are obtained by setting t_{th} : 10^{-4}, 5×10^{-3}, and 5×10^{-2},

Table 8.5: AlexNet compression results on ImageNet dataset

Models	Error ↑	FLOPS ↓	Param. ↓
LiM [268]	0.57%	–	76.8%
NISP [246]	0.00%	40.1%	47.1%
DFP [203]	4.08%	–	45.8%
Group-LASSO	0.81%	31.4%	53.1%
Group-LASSO	1.37%	44.2%	66.7%
Group-LASSO	2.41%	51.6%	78.5%
EGL	1.10%	40.2%	57.4%
EGL	1.68%	44.4%	67.6%
EGL	2.84%	52.3%	79.3%
Ours: DEGL1	**-0.02%**	**40.2%**	**57.4%**
Ours: DEGL2	**0.25%**	**42.1%**	**62.0%**
Ours: DEGL3	**0.31%**	**44.4%**	**67.6%**
Ours: DEGL4	**0.43%**	**52.3%**	**79.3%**

Reference AlexNet: Error = 41.81%, Param. = 61M, FLOPS = 727M

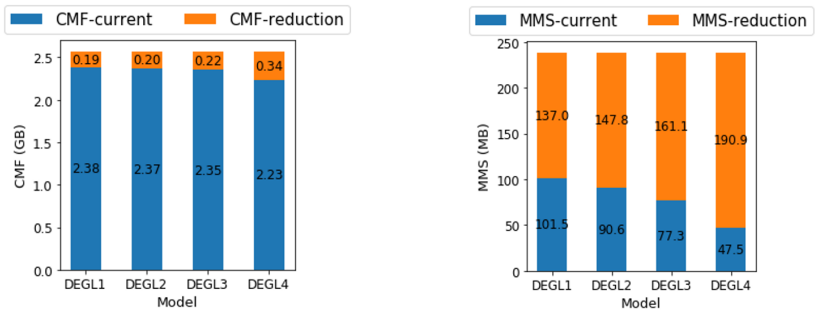


Figure 8.8: AlexNet CMF and MMS results on ImageNet

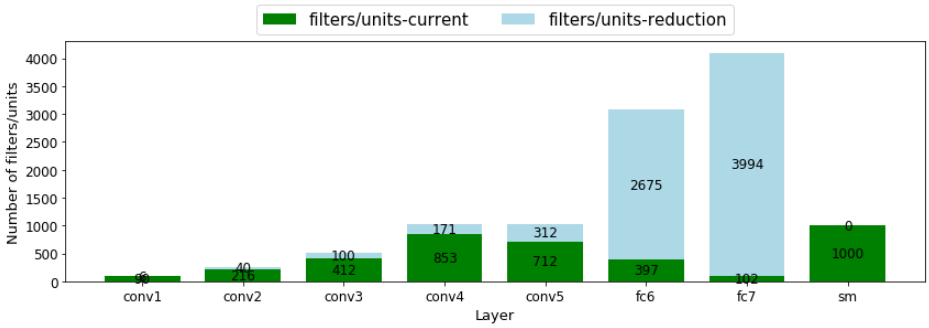


Figure 8.9: DEGL4 compression results for AlexNet on ImageNet

respectively; similar values are used for ResNet experiments. It is seen that based on achieved error rates, DEGL4 with 67.6% of model parameters pruned incurs no performance loss, while with group LASSO with 66.7% of model parameters pruned and EGL with 67.6%

Table 8.6: ResNet-50 compression results on ImageNet dataset

Models	Error ↑	FLOPS ↓	Param. ↓
ThiNet-50 [127]	1.15%	36.8%	33.8%
ThiNet-30 [127]	6.12%	71.5%	66.1%
NIPS [246]	0.89%	44.0%	43.8%
DCP [272]	1.06%	55.6%	51.5%
CNN-FCF [116]	0.47%	46.1%	42.4%
CNN-FCF [116]	2.62%	66.2%	61.0%
KSE [158]	0.84%	78.5%	65.8%
Group-LASSO	1.18%	44.3%	43.5%
Group-LASSO	1.55%	59.2%	54.1%
EGL	1.06%	47.6%	44.4%
EGL	1.79%	59.3%	54.4%
Ours: DEGL1	**0.11%**	**47.6%**	**44.4%**
Ours: DEGL2	**0.23%**	**59.3%**	**54.4%**
Ours: DEGL3	**0.52%**	**79.2%**	**67.0%**

Reference ResNet-50: Error = 24.02%, Param. = 25.06M, FLOPS = 4.1B

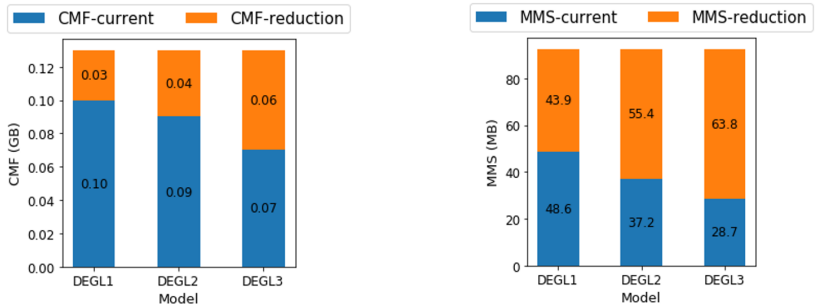


Figure 8.10: ResNet50 CMF and MMS results on ImageNet

of model parameters pruned incur performance loss of 0.17% and 0.28%, respectively. It is observed that further compression using group LASSO or EGL leads to larger performance loss. The current CMF and MMS along with the reductions after compression for the DEGL models (given in Table 8.5) are shown in Figure 8.8. For instance, DEGL4 reduces FLOPS from 727M to 347M, CMF from 2.57GB to 2.23GB and MMS 238.4MB to 47.5MB, with a test error increase of only 0.07%. In addition, the current number of filters or units in the different layers of DEGL2 are reported in Figure 8.9.

Compression results for ResNet-50 are given in Table 8.6. DEGL2 with 54.4% of model parameters pruned clearly outperforms group-LASSO with 39.6% of model parameters pruned, EGL with 54.4% of model parameters pruned and several state-of-the-art methods. Further compression leads to DEGL3 with 67.0% of model parameter pruned; DEGL3 is competi-

Table 8.7: ImageNet pre-trained ResNet-50 compression results

Dataset	Models	Error ↑	FLOPS ↓	Param. ↓
Oxford-102	Group-LASSO	0.93%	36.9%	34.7%
	Group-LASSO	1.26%	59.5%	55.2%
	EGL	0.89%	44.4%	43.9%
	EGL	1.52%	68.0%	62.9%
	Ours: DEGL1	**0.01%**	**44.4%**	**43.9%**
	Ours: DEGL2	**0.22%**	**68.0%**	**62.9%**
Food-5K	Group-LASSO	1.18%	39.9%	39.4%
	Group-LASSO	2.29%	69.5%	64.6%
	EGL	1.35%	42.1%	40.2%
	EGL	2.62%	69.7%	65.8%
	Ours: DEGL1	**0.53%**	**42.1%**	**40.2%**
	Ours: DEGL2	**1.31%**	**69.7%**	**65.8%**

Oxford-102 ResNet-50: Error = 3.43%, Param. = 25.06M, FLOPS = 4.1B
Food-5K ResNet-50: Error = 0.50%, Param. = 25.06M, FLOPS = 4.1B

tive with group-LASSO with 39.6% of model parameters pruned and EGL with 38.6%. The current CMF and MMS along with the reductions after compression for the DEGL models (given in Table 8.6) are presented in Figure 8.10.

Model compression for domain adaptation

We consider the effectiveness of our compression method for domain adaptation using Oxford-102 [142] flower species datasets and Food-5k [197] datasets, which contain 102 different flower species and food/non-food images, respectively. Oxford-102 flower species dataset has 6149, 1020 and 1020 training, validation and testing samples, respectively; for this dataset, we follow the training and testing protocols in [233]. Food-5K dataset contains 2500 food images and 2500 non-food images; the training and testing protocols in [197] are employed for this dataset. Namely, we evaluate the original pre-trained and compressed models for performance loss. Our compression results using pre-trained ResNet-50 are given in Table 8.7, where DEGL1 and DEGL2 are obtained by setting $t_{th} : 5 \times 10^{-3}$ and 5×10^{-4}, respectively. All models are trained using the same hyperparameters to support fair comparison. It is seen that the proposed DEGL models significantly outpeform Group-LASSO and EGL models. As such, DEGL is well-suited for domain adaptation tasks.

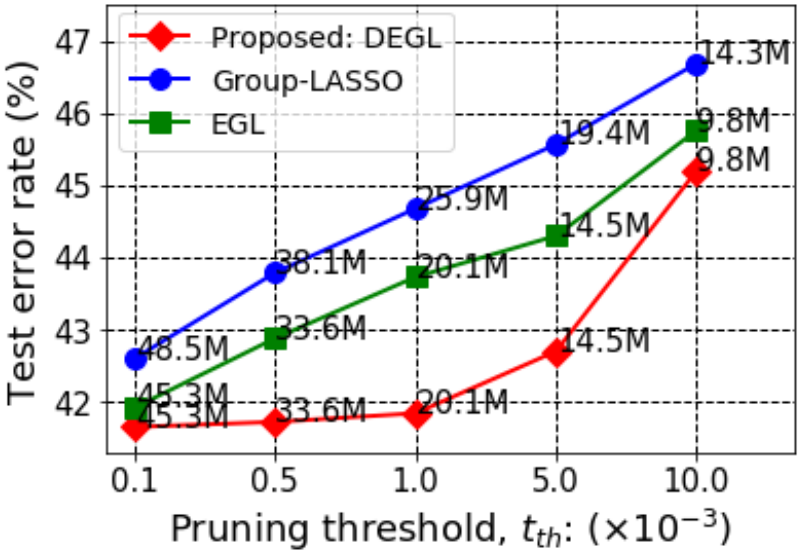


Figure 8.11: Pruning threshold values and AlexNet performance loss on ImageNet dataset

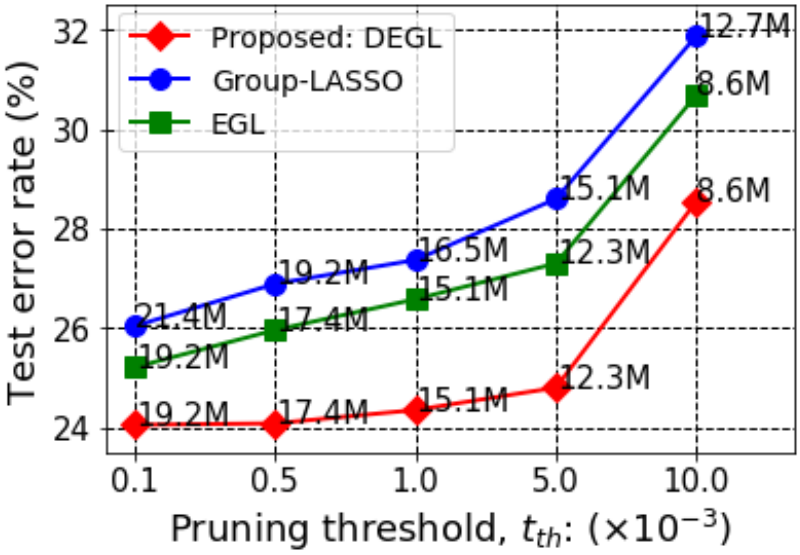


Figure 8.12: Pruning threshold values and ResNet-50 performance loss on ImageNet dataset

8.5.3 Ablation studies

As ablation studies, we observe the performance loss of elastic group LASSO without model debaising. That is, after pruning, we employ the same cost objective in (8.6) for model re-

training. Results of these experiments from Elastic group LASSO without the debiasing step are denoted (EGL), and are given along side DEGL and group LASSO for all experiments. In addition, the impact of pruning threshold values and group LASSO penalty weight hyper-parameter, γ in (6) & (7), on model performance loss are studied; see supplemtary material for results, including the training times for DEGL, group LASSO and EGL.

Pruning threshold values and performance

Herein, we perform additional experiments to observe how pruning threshold values (t_{th}) impacts performance loss for compressed models, which include the proposed DEGL, group LASSO and EGL. For this investigation, experiments are carried out on ImageNet using AlexNet and ResNet-50 models. Figure 8.11 and Figure 8.12 show obtained results including the current number of model parameters on AlexNet and ResNet-50, respectively; results given are recorded after pruning and retraining all models. It is seen that the proposed model compression approach, DEGL, consistently incurs smaller performance loss as compared to group-LASSO and EGL. Furthermore, it is noted that EGL outperforms group LASSO. Also, given a specified pruning threshold value, t_{th}, DEGL results in smaller number of model parameters than group-LASSO. Overall, it is seen that the performance losses of all the compression methods increase with an increase in pruning threshold values, since the resulting models become progressively smaller.

Feature selection regularization hyperparameter and performance

We also observe the impact of group feature selection hyperparameter, γ, on compression results based on performance loss. Experiments are performed on imageNet dataset using AlexNet and ResNet-50 models, and results are shown in Figure 8.13 and Figure 8.14, respectively. It is observed that for small values of γ, EGL outperforms group-LASSO. However, for both AlexNet and ResNet-50, the progress increase of γ leads to worse EGL performance than group-LASSO models. This interesting scenario is directly attributed to high model bias for EGL when γ exceeds a certain limit. This follows from the fact, both group

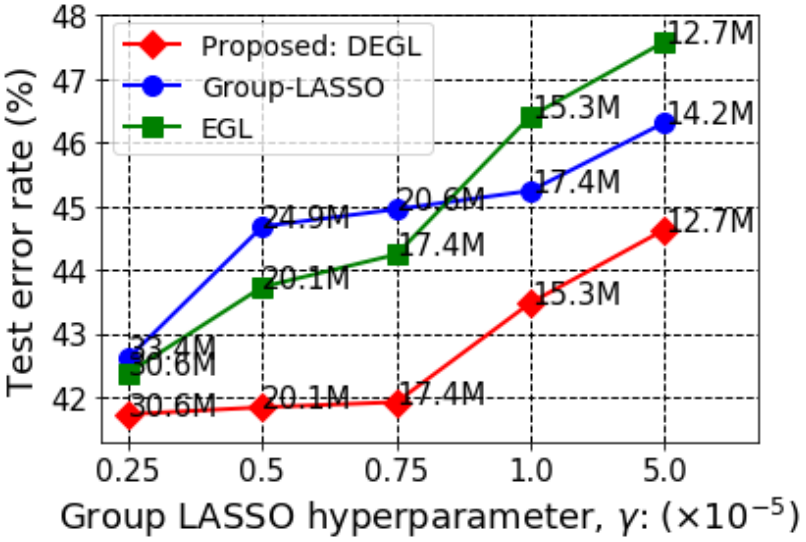


Figure 8.13: Group LASSO penalty weight and AlexNet performance loss

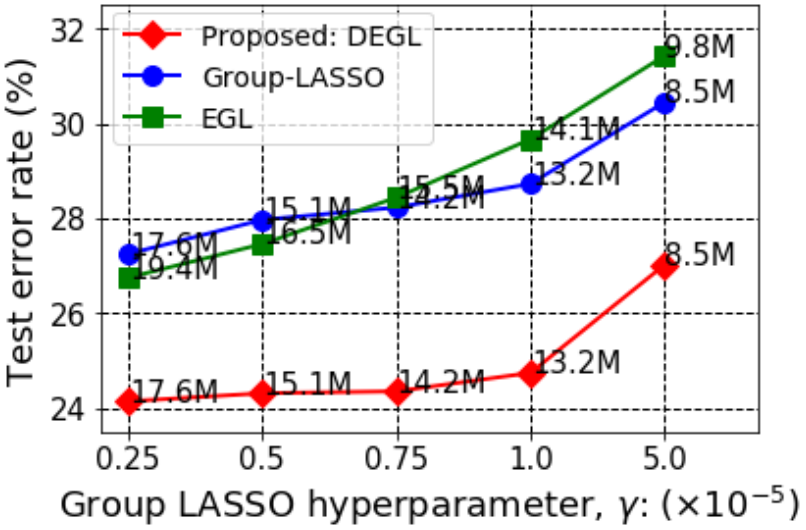


Figure 8.14: Group LASSO penalty weight and ResNet-50 performance loss

LASSO and $L2$-norm penalties are used for retraining after model pruning. Conversely, group-LASSO models use only the group LASSO penalties for retraining after model pruning. Importantly, it is seen that for all values of γ for the compared models on AlexNet and ResNet-50, the proposed DEGL models incur the smallest performance loss. The good

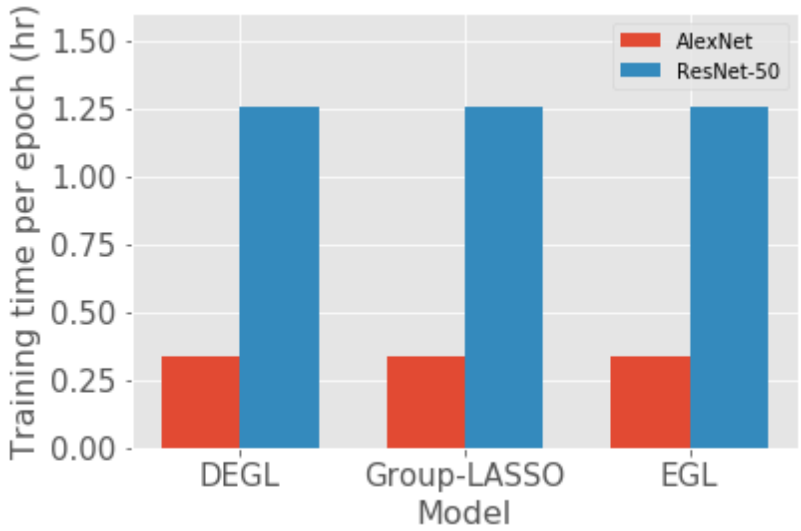


Figure 8.15: Training time for compression methods on imageNet

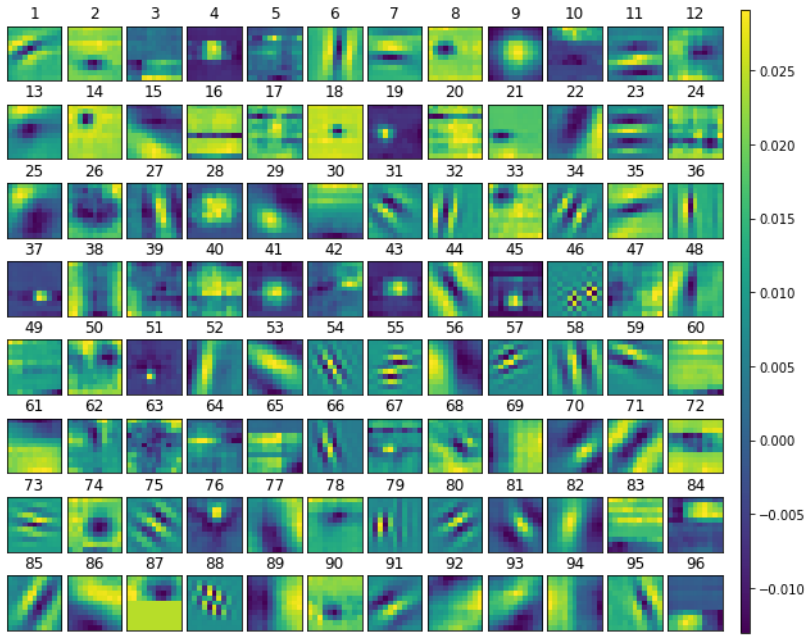


Figure 8.16: First convolution layer filters in AlexNet trained with DEGL. Filters '3', '20', '40', '49', '60'and '64' are selected for pruning

performance of DEGL is attributed to the debiasing step after model pruning; the retrained model uses only the $L2$-norm penalty.

Training time

Figure 8.15 shows the training times for compression methods DEGL, group LASSO and EGL on AlexNet and ResNet-50. Specifically, the times for the completion of one epcoh for the different models are given; results for each model are averaged over 3 different runs using a training batch size of 256; four V100 GPUS running on a workstation with 128GB of RAM are used. As such, it is seen that the proposed DEGL for compression does not increase training time; all the compression approaches compared require approximately the same training time. The same observation is made on all the other datasets used in this work.

Visualization of filters selected for pruning

The 96 convolution filters of the first layer of AlexNet using DEGL are shown in Figure 8.16. For compression, the six filters reported are selected for pruning; selected filters are determined as in Section 8.4.1 (b) in the main manuscript.

8.6 Conclusion

In this chapter, we address structured compression of deep neural networks by taking inspiration from sparse signal processing. Directly motivated by the drawbacks of applying conventional group LASSO for compressing deep neural networks (DNNs), we propose 'debiased elastic group lasso (DEGL)' that is applied to several state-of-the-art DNNs using six benchmarking datasets. Compression results are evaluated based on the percentage of pruned parameters, FLOPS, computational memory footprint and memory for storage. Results of extensive compression experiments show that DEGL performs better than conventional group LASSO and several state-of-the-art methods. Overall, the results reveal that considerable model compression can be achieved using DEGL with little performance loss or even improved generalization.

Chapter 9

Deep Neural Network Compression with Teacher Latent Subspace Learning and LASSO

Deep neural networks excel in understanding multimedia by using latent representations to learn complex and useful abstractions. However, they remain unpractical for embedded devices due to memory constraints, their high latency and considerable power consumption at runtime. In this chapter, we propose the compression of deep models based on learning lower dimensional subspaces from their latent representations while maintaining a minimal loss of performance. We leverage on the premise that deep convolutional neural networks extract many redundant features to learn new subspaces for feature representation. We construct a compressed model by reconstruction from representations captured by an already trained large model. In comparison to other methods, the proposed requires no labeled data. Moreover, it allows the use of sparsity inducing LASSO parameter penalty to achieve better compression results than when used to train models from scratch. We perform extensive experiments using different models on benchmarking datasets to show that small generalization loss is incurred using the proposed compression method, while realizing more compact models that require smaller storage memories and inference times.

9.1 Introduction

Many computer vision tasks work well with features that are learned using deep neural networks (DNNs) of few (i.e. 3-10) layers of latent representations [240, 153]. However, in recent times, analysing more complex multimedia with reasonable accuracies has necessitated that we rely on features learned from deep networks of several layers of latent representations [67, 95, 207, 254] (i.e. over 10 layers), as many works [135, 15, 254] posit the benefit of depth and width for approximating complex target functions. The success of very deep networks on learning hard multimedia tasks has motivated their deployment in various electronic devices. However, memory consumption is a major concern when deploying applications on devices such as smart phones and other portable devices. Furthermore, running large very deep models on portable devices may result in unsatisfactory long inference time for operation. Lastly, such large very deep models would inevitably increase the power consumption of portable electronic devices and subsequently shorten battery life. The different aforementioned problems can work together to limit the deployment of very deep models in portable electronic devices. For example, the popular AlexNet [100] has 61M parameters and is about 250MB in memory size [169]. Model compression aims to reduce the number of model parameters and memory size of large deep models, while incurring minimal loss of model performance. Essentially, given a large model, referred to as *teacher*, that has been trained to an acceptable performance; model compression attempts to obtain from the teacher a small model, referred to as *student*, with minimal performance loss [73].

Knowledge Distillation (KD) has been proposed as a solution to achieve model compression [125, 25, 271] by training a student model to mimic an ensemble [73] or via intermediate-level teacher hidden layers to guide the training process of a single large model known as *FitNets* [175]. Both KD and FitNet employ objective functions that aim to constrain the student model to learn global mapping functions[1] that are similar to that of the teacher model. However, the FitNet is an improvement over KD, as in addition to directly employing the KD training objective, the hint from the middle layer of the Teacher model is used to guide (i.e.

[1] Mapping from input data space to output (softmax) space

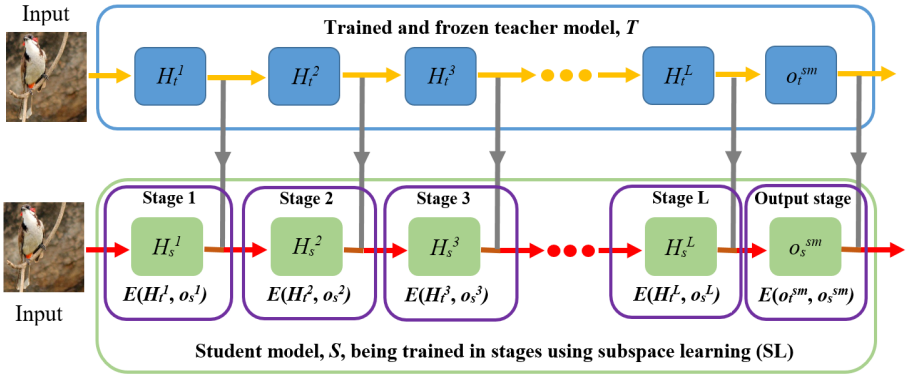


Figure 9.1: Overall framework for the proposed subspace learning. It is assumed that the teacher model, T, is already trained to convergence and then its parameters are frozen. The Student model, S, is trained in a layer-wise fashion. In each stage, the hidden layer of the student model, H_s^l, is used to learn a subspace of the corresponding hidden layer of teacher model, H_t^l; a cost function $E(H_t^l, o_s^l)$ that minimizes the reconstruction error of the hidden repersentation of the teacher model H_t^l and the student model's output o_s^l is defined to achieve this. The output of each layer is the variable given in the layer's block. o_t^{sm} and o_s^{sm} represent the softmax outputs of the teacher model and student model, respectively.

constrain) the latent representation in the middle layer of the student model to closely mimic the Teacher model. Nevertheless, we argue that the constraints used for learning both KD and FitNet are *considerably weak* and do not provide a tight constraint on the family of local mapping functions[2] that can be learned for composing the global mapping function of the teacher model. Indeed, one can observe that the different hidden layers of both KD and FitNet still have a large number of family of functions that can be used to approximate the teacher's global mapping function. In addition, the objective functions for both KD and FitNet rely on data true labels for training. In real-life scenarios, availability of labeled data is usually limited; consequently, both KD and FitNet become inapplicable.

In this chapter, we address the aforementioned problems of KD and FitNet by viewing model compression as a subspace learning problem. On one hand, the proposed subspace learning allows knowledge transfer from a large model to a small model for the purpose of model compression. On the other hand, the same proposal permits small models to mimic the hidden representations of trained large models without using the KD training objective.

[2]Mapping from a hypothetical hidden layer to the adjoining one

Our approach imposes a tighter constraint (than KD and FitNet) on the latent representations that the different hidden layers of the student model can learn, given the teacher model. Furthermore, the proposed approach suffices for training a student model with unlabeled data where both KD and FitNet cannot be used. In addition, contrary to unpromising results reported in [65] using sparsity inducing Least Absolute Shrinkage and Selection Operator (LASSO) [218] as a penalty for model pruning, even after model retraining, the proposed latent subspace learning allows LASSO to be used with improved results without the need for retraining. We posit that the end-to-end LASSO optimization of DNNs with complex loss landscapes can impact convergence and solution stability [94].

Particularly, the model compression approach proposed herein is motivated by two well-known important premises about deep networks: (i) they learn many redundant features [203, 29], (ii) they are often over-parameterized [3]. Based on (i) and (ii), one can assume that, with a high probability, there exist feature subspaces that can considerably characterize the target function without much loss of generalization capacity. As such, our contributions in this work are as follows:

1. A new student model is constructed by learning latent subspaces for the representations captured by an already trained large model. This provides a constraint on the family of local functions that is learnable by all layers of the student in comparison with KD [73] and FitNet [175].

2. A teacher model is compressible even when available data is unlabeled; where, KD and FitNet are not applicable. In fact, the proposed student model results can be further improved by using additional unlabeled data.

3. LASSO penalty can be incorporated in the proposed student model to improve compression results without retraining as in [65]; this approach outperforms conventional training with the LASSO penalty.

The overall framework of the proposed subspace learning is shown in Figure 9.1. This concept is similar in spirit to model pruning where a large model is first trained, and then pruned using some scheme or criteria. Essentially, the idea is that large models have enough

parameters such that we have reasonably large search spaces to explore different configurations of solutions for fitting complicated target functions. However, such large models are generally cumbersome so that we are interested in trimming them to obtain compact versions.

This work is different from KD [73] and FitNet [175] as follows: (i) compression is performed in a layer-wise fashion; KD and FitNet compress all layers jointly, (ii) the teacher and student models are of the same depth; the student is deeper than the teacher in FitNet, and as such can undesirably increase inference time (iii) the student model does not use the KD objective as in FitNet.

The proposed model compression approach has been applied to VGG-16 and wide ResNet (W-ResNet) models using CIFAR-10 [99], CIFAR-100 [99], MNIST [106] and SVHN [251] datasets. We achieve a considerable reduction in the number of model parameters and therefore memory size with minimal loss of performance; its application results in reduced power consumption and inference time. Our extensive experimental results show that the proposed approach outperforms similar compression techniques, namely, KD and FitNets. Furthermore, the proposed approach outperforms several state-of-the-art results [119, 58, 135, 107] for similar models trained from scratch. The rest of this chapter is organized as follows.

In Section 9.2, related works are discussed. Section 9.3 gives the background on the proposed model compression approach along with the problem statement. Section 9.4 presents the proposed approach and Section 9.5 contains experimental results. The chapter is concluded in Section 9.6.

9.2 Related work

DNNs have found applications in various learning problems such as document representation [258] and traffic flow prediction [263]. However, large DNNs are typically cumbersome so that we can have problems that include large memory for storage, enormous computing resource for operation and high latency during inference. Consequently, several works have

addressed model compression. They include parameter pruning [65], quantization [37, 110], low-rank factorization [42, 213, 85], sparse convolution filters [120], and KD [20, 73, 175]. Pruning performs well in removing model parameters that are redundant or not critical to the performance of the model [65]. Generally, all model parameters that fall below a certain threshold are removed from the model, leading to a notable reduction in their number. We note that pruninng methods that are based on $L1$-norm or $L2$-norm sparsity inducing penalties require significantly higher number of training iterations for successful convergence [30]. The advantage of parameter quantization is the reduction of the required number of bits for representing the individual model parameters [37, 110]. However, the drawbacks of the quantization approach are significant performance loss [30] and specialized algorithms and hardware [226, 265] for running the compressed model.

Since most of the computational bottleneck in deep models usually lies in the convolution filters, low-rank factorization has been proposed in [42, 213] for tensor decomposition that allows compressed models to perform faster inference. For example, [85] used low rank factorization to achieve a 4.5$\times$ speedup while incurring only 1% generalization loss on the ILSVRC2012 dataset [145]. Unfortunately, low-rank factorization methods are computationally expensive, and require extensive model retraining after compression [30].

Sparsity constraints have also been employed for model compression. By learning considerably sparse solutions, many model parameters can be removed from the model without a significant loss of generalization capacity. The approach proposed in [120] yielded over 90% of model parameters that were zeros, while incurring a generalization loss of under 1% on the ILSVRC2012 dataset [145]. Furthermore, [204] used regularization penalty to enforce model parameters to have small values, have been explored for building compact and explainable models. In [88], LASSO penalty was used for model selection for random forest models given high-dimensional inputs; good results are reported for the resulting models. However, this approach suffers from inconsistent results; for example, [65] reported unpromising results, and thus employed $l2$-norm penalty instead for enforcing model parameters to have small values. Other works that employed LASSO penalty for enforcing parameter sparsity include feature selection [269], classification of Byopse images [79], temporal

effects in a data mining approach [195], regularization of hybrid models that work for falls detection [201], time series forecasting [144] and regularizing fuzzy neural network [21].
One of the closely related works to this work is KD. The idea is essentially either of the following: (i) compress the knowledge from several models (i.e. ensemble) into a single model [20, 73], or (ii) compress the knowledge from an already trained large model (i.e. teacher) to a small one (i.e. student) [175]. In the ensemble case, the aim is to obtain a single model that mimics the average prediction characteristics of the ensemble. For instance, using automatic speech recognition, the KD proposed approach [73] successfully compressed the knowledge in an ensemble of 10 models into a single model with 0.49% loss in test frame accuracy. In [271], online native ensemble is proposed for learning a single multi-branch network. The work reported improved model generalization over the original models. Furthermore, knowledge distillation has been combined with generative adversarial networks (GAN) for compression [227], where it is argued that direct knowledge distillation results in suboptimal learning of student models. Although interesting results were reported in the work [227], a fair comparison with other compression methods is difficult due to the data augmentation influence of the GAN employed. The proposed FitNets in [175] took inspiration from [73] for model compression. FitNet is essentially KD with intermediate layer guidance for the student model such that the similarity of the representations learned by the large and small model is improved.
Herein, we propose a new paradigm for deep learning model compression where the goal is to learn a student model from an already learned teacher model as opposed to training a student model from scratch.

9.3 Background and problem statement

9.3.1 Background: KD and FitNet

For distilling knowledge [73, 175] from a single teacher model (T) into a student one (S), the softened softmax outputs of T are incorporated into the target outputs for training S. The goal is to constrain S to mimic the global mapping function of T. Let T be the teacher model

with an output softmax probability P_T, S be the student model with parameters W_S and output softmax probability P_S, τ the hyperparameter for softening the logits of the teacher and student models, P_S^τ the softened version of P_S and P_T^τ the softened softmax output of the teacher model. The student model in [73] is trained by optimizing a KD loss function L_{KD} such that:

$$L_{KD}(W_S) = \lambda H(y, P_S) + H(P_T^\tau, P_S^\tau), \tag{9.1}$$

where H is the cross-entropy loss function, λ is a hyperparameter for controlling the impact of the hard target cross-entropy and y is the ground truth label for the training data.

In FitNet [175], S is *much deeper* than T, and training is achieved using hint, which is the latent representation of T at the middle of the model for guiding a single hidden layer choosen in the student model, S. Note that by construction, S is deeper and thinner than T for the FitNet. The parameters of S from the input (layer 1) up to layer l in the network $W_S^{1:l}$ are first pre-trained by reconstructing the outputs of T at layer l, after which all the parameters of S denoted W_S, are trained using KD objective as in (9.1). A problem with the FitNet is that choosing the appropriate hidden layer in S to be guided by the hints from T can be tricky in relation to compression results. This can significantly increase the range of experiments required for obtaining decent compression results.

9.3.2 Problem statement

Let the global mapping functions for T and S be denoted by F_t and F_s, respectively. Also, let the local mapping functions for the different layers be f. Deep networks learn compositional functions, thus using input data x. We can write the global mapping functions for T and S with L hidden layers and softmax layer denoted by sm as follows

$$F_t = f_t^{sm}(f_t^L(...f_t^1(x; \boldsymbol{W}_t^1)...); \boldsymbol{W}_t^L); \boldsymbol{W}_t^{sm}), \tag{9.2}$$

and

$$F_s = f_s^{sm}(f_s^L(...f_s^1(x; \boldsymbol{W}_s^1)...); \boldsymbol{W}_s^L); \boldsymbol{W}_s^{sm}). \tag{9.3}$$

First, from (9.2) and (9.3), it can be seen that KD in (9.1) constrains only f_s^{sm} to mimic f_t^{sm}. However, layers in S have a large family of functions to learn local mapping functions from f_s^1 to f_s^L. FitNet attempts to address this problem by constraining f_s^{sm} to mimic f_t^{sm} as in KD and also f_s^j to mimic f_t^j; where j denotes a layer in the middle of S and T. Nevertheless, the other layers in S still have a large family of functions to learn local mapping functions from. As such, it seems that the task of joint knowledge transfer from all the (or several) hidden layers of the teacher model to the student model is somewhat complicated. Subsequently, this may impact generalization capacity of student models. Furthermore, the success of LASSO for DNN compression based on sparsity requires the models to be well optimized so that many parameters can attain *very small values*, and can therefore be removed without hurting model performance. Unfortunately, training DNNs with many layers is challenging due to units' outputs saturation and explosion [56, 68], internal covariate shift [84], and gradients instability [56, 11]. Subsequently, incorporating an additional constraint such as the LASSO penalty can make optimization even more difficult. This problematic scenario using LASSO is reported in [65], where the work settled for the $L2$-norm penalty instead. As such, decomposing the difficult joint optimization of all the hidden layers into a layer-wise optimization scheme would help improve model optimization even with LASSO penalty, since the non-convexity of the optimization problem is greatly reduced. Interestingly, we note that layer-wise training typically improves model optimization and generalization [10, 86]. Second, both KD and FitNet rely on the true data label y for training the student model and therefore limit their applicability.

In this work, the goal is to address the aforementioned compression problems: (i) learn a student model in which local mappings are constrained to follow the local mappings of the teacher model and (ii) a more general scenario in which an already trained teacher model is available, but labeled data is unavailable for training the student model.

9.4 Proposed latent subspace learning for compression

This section describes the compression of DNNs using the proposed latent subspace learning. The overall compression framework is given in Figure 9.2. The main aspects of the framework include training the teacher model to convergence, extracting its latent representations, training the Student model in a layer-wise fashion to reconstruct the latent representations of the Teacher model, and finally fine-tuning the Student model in an end-to-end fashion to align estimated parameters.

9.4.1 Latent subspace reparameterization

Deep networks typically learn many redundant features [203, 29, 42, 120] and are overparameterized [3, 120]; therefore there exists, with high likelihood, a latent subspace that approximately captures the same latent representation. As such, we propose to learn subspace features which construct a lower dimensional manifold that has learned latent representations of a Teacher model. This has the benefit of a streamlined solution space that the individual layers have to explore. The proposed approach consists in learning the Student model in a layer-wise fashion which decomposes and eases the task of approximating the function learned by the teacher model. Hence, the student model can focus on the solution that approximates the function that is already captured by the teacher model via latent subspace learning. Consequently, this approach provides a tight constraint on the latent representations that S can learn, given T, as supported by our experimental results in Section 9.5. Furthermore, the proposed layer-wise latent subspace learning (i.e. layer pre-training) eases DNN training so that the incoporation of the LASSO penalty does not plague optimization, since layers are trained one at a time, as opposed to the conventional approach of joint training of all layers [50]. That is, the original task of learning the Student model in an end-to-end fashion is decomposed into a simpler optimization problem by learning the student model in a layer-wise fashion using the latent representations of the Teacher model as intermediate targets. On top of this, layer-wise training is useful for improving model regularization [50, 51], as seen even in recent works [54, 187, 126].

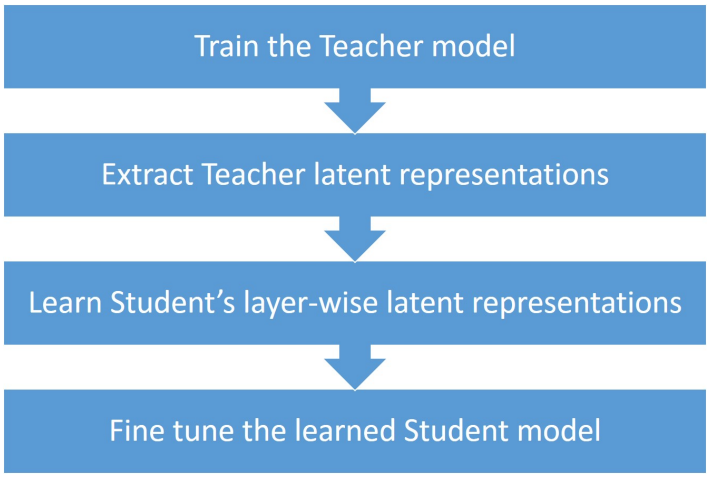


Figure 9.2: Overall framework for compressing a Teacher deep model to obtain a Student deep model via subspace learning

We start by assuming that all the different layers of a deep model are over-parameterized such that we obtain an overall over-parameterized teacher model T. We consider a deep teacher model indexed, t, with latent representation H_t^l at layer l and the number of free parameters $\boldsymbol{\theta}_t^l = |\boldsymbol{W}_t^l|$, where $\boldsymbol{W}$ is the weight matrix. Generally, T has $\boldsymbol{\theta}_t^l \gg 10^p$, where p can range from 2 to 6 such that a large solution space (or solution configurations) can be explored during optimization [29]. This is obvious for deep models which are typically over-parameterized relative to the number of training examples and therefore result in under-determined systems [172]. The objective herein is to learn a deep student model S indexed s from T such that the number of student's free parameters $\boldsymbol{\theta}_s^l$, where $\boldsymbol{\theta}_s^l = |\boldsymbol{W}_s^l|$, is less than their corresponding teacher's free parameters $\boldsymbol{\theta}_t^l$ without loss of model generalization performance. The problem of finding a compressed model S is formulated as learning a feature subspace (i.e. lower dimensional manifold) for the latent representations already captured by the teacher model T. The task of compressing via feature subspace learning allows us to learn a deep student model S such that $\boldsymbol{\theta}_s^l < \boldsymbol{\theta}_t^l$ via reconstruction as summarized in Figure 9.2 and detailed below.

The teacher model T is first trained to good convergence using a training dataset $\{x^n, y^n\}_{n=1}^N$. Thereafter, $\{x^n\}_{n=1}^N$ or another unlabeled dataset $\{u^n\}_{n=1}^K$ is used for obtaining the teacher latent representations $\boldsymbol{H}(\boldsymbol{X})_t^l$ at every layer, where $1 \leq l \leq L$ and L is the depth of the

model excluding the output (i.e. softmax) layer.

The student model S is then constructed of the same number of hidden layers as T, but with reduced number of convolution filters or hidden units for every layer; that is $\boldsymbol{\theta}_s^l < \boldsymbol{\theta}_t^l$. Use $\boldsymbol{H}(\boldsymbol{X})_s^{l-1}$ as the input and task the hidden layer $\boldsymbol{H}(\boldsymbol{X})_s^l$ to encode the reconstruction of the teacher's latent representation $\boldsymbol{H}(\boldsymbol{X})_t^l$ in the student model output layer as o_s^l . Note that $\boldsymbol{H}(\boldsymbol{X})_s^1$ is obtained by using $\{x^n\}_{n=1}^N$ as the input and reconstructing $\boldsymbol{H}(\boldsymbol{X})_t^1$ as the target output. Henceforth, the weights between the input and hidden layer is referred to as the 'encoder weights'. Similarly, the weights between the hidden and output layer of any model is referred to as the 'decoder weights'. The teacher model latent representation reconstruction at layer l using the student model layer l with encoder weights $\boldsymbol{W}_s^{l^{enc}}$, decoder weights $\boldsymbol{W}_s^{l^{dec}}$ encapsulated as $\boldsymbol{W}_s^l$ and activation function φ, can be performed by optimizing the cost function:

$$J(\boldsymbol{W}_s^l) = \underset{\boldsymbol{W}_s^l}{\operatorname{argmin}} \sum_{n=1}^{N} (\boldsymbol{H}(\boldsymbol{X})_t^{l^n} - o_s^{l\,n})^2, \tag{9.4}$$

where,

$$o_s^l = \boldsymbol{W}_s^{l^{dec}} \boldsymbol{H}(\boldsymbol{X})_s^l = \boldsymbol{W}_s^{l^{dec}} \varphi(\boldsymbol{W}_s^{l^{enc}} \boldsymbol{H}(\boldsymbol{X})_s^{l-1}). \tag{9.5}$$

Model parameters $\boldsymbol{W}_s^{l^{dec}}$ and $\boldsymbol{W}_s^{l^{enc}}$ can be updated with block gradient descent algorithm using the following equations.

$$\Delta \boldsymbol{W}_s^{l^{dec}} = -\eta \frac{\partial J(\boldsymbol{W}_s^l)}{\partial \boldsymbol{W}_s^{l^{dec}}}, \tag{9.6}$$

$$\Delta \boldsymbol{W}_s^{l^{enc}} = -\eta \frac{\partial J(\boldsymbol{W}_s^l)}{\partial \boldsymbol{W}_s^{l^{enc}}}. \tag{9.7}$$

For the student model softmax layer output o_s^{sm}, we use $\boldsymbol{H}(\boldsymbol{X})_s^L$ as input and task o_s^{sm} to reconstruct the softmax layer output of the teacher model o_t^{sm} as its output; the incoming and outgoing weights of output layer altogether are denoted $\boldsymbol{W}_s^o$. Note that the output logits are not softened for both the teacher and student models. The true labels $\{y^n\}_{n=1}^N$ are not used for the input data $\{x^n\}_{n=1}^N$ to learn o_s^{sm}. Rather, the softmax layer outputs o_t^{sm} of the teacher model are used. The encoder weights of $\boldsymbol{W}_s^l$ and $\boldsymbol{W}_s^o$, $\boldsymbol{W}_s^{l^{enc}}$ and $\boldsymbol{W}_s^{o^{enc}}$, respectively, are saved for the alignment stage.

Furthermore, model compression via sparsity inducing LASSO[3] constraint is employed to improve the level of compression while incurring minimal (or no loss) of model test performance. This is achieved by adding $L1$-norm of model parameters [218] as a penalty term to (9.4); thus we can write a new training cost function as

$$J(\boldsymbol{W}_s^l) = \underset{\boldsymbol{W}_s^l}{\operatorname{argmin}} \left\{ \sum_{n=1}^{N} (\boldsymbol{H}(\boldsymbol{X})_t^{l\,n} - o_s^{l\,n})^2 + \lambda \parallel \boldsymbol{W}_s^{l^{enc}} \parallel_1 \right\}, \tag{9.8}$$

where λ controls the impact of $L1$-norm penalty on $\boldsymbol{W}_s^{l^{enc}}$. Note that $L1$-norm penalty is not applied to the weights connections from the hidden layers to the output layer.

9.4.2 Parameters assembly, fine-tuning and pruning

Note that during subspace learning, the different layers are trained without the explicit knowledge of the other layers. Hence, there is a need to perform parameters alignment by assembling all the encoder weights of all the different layers of the student model, and training the whole student model weights altogether again using gradient descent algorithm. The procedure of parameters alignment is referred to as 'fine-tuning' and is described as follows. The student model is assembled for fine-tuning to improve the alignment of the parameters of the different layers towards the global cost function. All the parameters of the student model, $\boldsymbol{W}_s$, are trained in an end-to-end fashion; that is, using input x^n, target teacher output ${o_t}^n$ and cost function $J(\boldsymbol{W}_s)$, we can minimize the negative conditional loglikehood of the training data defined as follows:

$$J(\boldsymbol{W}_s) = -\underset{\boldsymbol{W}_s}{\operatorname{argmin}} \sum_{n=1}^{N} \log P({o_t}^n | x^n). \tag{9.9}$$

Models that employ LASSO penalty for subspace learning also employ the same penalty term for parameters fine-tuning. Similar to the subspace learning stage, an $L1$-norm penalty term is added to the conventional cost function in (9.10); thus the fine-tuning cost function

[3]LASSO and $L1$-norm are used interchangably

Algorithm 1: Batch-wise subspace learning and fine-tuning

1. Given an already trained teacher model, T, do (2).
2. Sample H_t^l, the activations of T at layer l, using training data.
3. Initialize the layer weights of the student model S.
4. Using H_s^{l-1} as input and student model layer H_s^l, reconstruct H_t^l in the student's model output layer. At convergence, do (5)
5. Save $W_s^{l^{enc}}$, the encoder parameters of H_s^l.
6. Update $l \longrightarrow l+1$, and do (2) to (5) while $l \leq L$
7. Using H_s^L as input and student model output layer, o_s^{sm}, o_t^{sm} as output, and save encoder weights, $W_s^{o^{enc}}$.
8. Restore $W_s^{l^{enc}}$ and $W_s^{o^{enc}}$, all the individual layer weights of S obtained via subspace learning.
9. Align all student layer weights via fine-tuning.

for models that use LASSO penalty can be written as

$$J(\boldsymbol{W}_s) = \underset{\boldsymbol{W}_s}{\operatorname{argmin}} \left\{ -\sum_{n=1}^{N} \log P({o_t}^n|x^n) + \lambda \parallel \boldsymbol{W}_s \parallel_1 \right\}, \tag{9.10}$$

where notations remain as earlier described. For the sake of conciseness, the proposed compression approach is summarized in Algorithm 1 using a suitable training batch size; LASSO penalty can also be employed.

For student models that are trained using sparsity inducing LASSO penalty, parameters pruning is performed after model fine-tuning to further increase compression level at minimal performance loss. This follows from the fact that depending on the value of λ for $L1$-norm penalty, there are many parameters that are zeros. Also, many other parameters have very small values. These parameters can be pruned (i.e. removed) from the model without significantly impacting model performance. In order to achieve this, a pruning threshold, T_h, is set such that any model parameter, w_j, below the threshold is pruned: $w_j \longrightarrow 0, \because w_j < T_h$. However, in contrast to a similar approach [65], the pruning method in this work is not iterative; that is, the student model is not retrained after pruning. Later on, we show that the initial subspace learning indeed conditions the parameters of the student model such that good compression results without the need for model retraining after pruning.

For compressing W-ResNet models with skip connections, we consider each ResNet block as a layer. Specifically, each ResNet block in the student model is tasked with reconstructing the output of the corresponding ResNet block in the teacher model.

9.4.3 Subspace learning, mutual information and convergence

In this section, we show the relationship between the proposed subspace learning and mutual information I maximization similar to auto-encoding. Considering (9.4) and (9.5), where $\boldsymbol{H}(\boldsymbol{X})_s^l$ is used to learn the reconstruction of H_t^l; we can write I between $\boldsymbol{H}(\boldsymbol{X})_s^l$ and $\boldsymbol{H}(\boldsymbol{X})_t^l$ as

$$I(\boldsymbol{H}(\boldsymbol{X})_s^l; \boldsymbol{H}(\boldsymbol{X})_t^l) = \mathbb{E}\left[\log \frac{P(\boldsymbol{H}(\boldsymbol{X})_s^l, \boldsymbol{H}(\boldsymbol{X})_t^l)}{P(\boldsymbol{H}(\boldsymbol{X})_s^l)P(\boldsymbol{H}(\boldsymbol{X})_t^l)}\right], \tag{9.11}$$

where (9.11) can be written in terms of entropy $\mathbb{H}$ as

$$I(\boldsymbol{H}(\boldsymbol{X})_s^l; \boldsymbol{H}(\boldsymbol{X})_t^l) = \mathbb{H}(\boldsymbol{H}(\boldsymbol{X})_t^l) - \mathbb{H}(\boldsymbol{H}(\boldsymbol{X})_t^l | \boldsymbol{H}(\boldsymbol{X})_s^l), \tag{9.12}$$

and maximizing I in (9.11) amounts to maximizing

$$\underset{W_s^l}{\operatorname{argmax}}\, I(\boldsymbol{H}(\boldsymbol{X})_s^l; \boldsymbol{H}(\boldsymbol{X})_t^l) = -\underset{W_s^l}{\operatorname{argmax}}\, \mathbb{H}(\boldsymbol{H}(\boldsymbol{X})_t^l | \boldsymbol{H}(\boldsymbol{X})_s^l). \tag{9.13}$$

Particularly, observe from (9.11) that I is zero when $\boldsymbol{H}(\boldsymbol{X})_s^l$ and $\boldsymbol{H}(\boldsymbol{X})_t^l$ are independent; this is the case before training in which the weights of the student model are randomly initialized. However, subspace learning aims to maximize I as in (9.13); after training, $\boldsymbol{H}(\boldsymbol{X})_s^l$ and $\boldsymbol{H}(\boldsymbol{X})_t^l$ are highly dependent [222]. In Section 9.5, experiments that validate this are reported.

Furthermore, DNNs can be hard to train due to numerous local minima that arise owing to their compositional structure. Namely, propagating input data over several layers in the forward pass and backpropagating error gradients via several layers for weights update can make optimization challenging. Interestingly, the proposed subspace learning decomposes learning the student model into stages, where each layer is trained singly and thus eliminates the aforementioned problems associated with joint training of all layers. Again, in Section 9.5, experiments that support faster convergence as compared to training from scratch are given.

9.5 Experiments

9.5.1 General settings

For validating our proposal, we perform extensive experiments using images from benchmarking datasets. In order to demonstrate the effectiveness of the proposed compression approach, we consider two scenarios: (1) Labeled data is available (CIFAR-10, CIFAR-100 and MNIST datasets) for training the teacher and student models as in [73, 175] and (2) An already trained teacher model is available (SVHN dataset), but labeled data is unavailable for training the student model.

For the teacher model, plain network VGG-16, and W-ResNet-16-8 with skip connections models are used. The architectures of the student models are akin to the teacher model except for the reduction in the number of convolution filters or hidden units. The teacher and student models are trained with mini-batch stochastic gradient descent using the ADAptive Moment estimation (ADAM) optimization technique [96], dropout [206] and batch normalization [84]. For the teacher and student models, their respective initial learning rates are annealed from 10^{-3} to a final value of 10^{-6}; the stagnation of the training loss within a value of 10^{-3} after 5 epochs is used to achieve the annealing. For all the proposed student models reported in this work, a maximum of 30 epochs are used for the reconstruction of teacher model latent representations. i.e. subspace learning. All the reported teacher and student models have the same depth. The design of the architectures of student models for KD approaches is usually achieved via experimentation [125, 175, 73]. However, we simplify the architectural design of the student models by only reducing the layer width of the teacher models by a constant factor. We experiment with two different compressed student models referred to as stud-s ("s" denotes small) and stud-m ("m" denotes medium) that are obtained by reducing the number of filters (i.e. for convolution layers) and units (i.e. for fully connected layers) in every layer of the teacher model by one-half and one-quarter, respectively.

In the first scenario, the teacher model is trained, after which the proposed approach is employed for compressing the knowledge already acquired into the student models. To further improve compression results based on number of model parameters and memory size

Table 9.1: Compression results on CIFAR-10 dataset

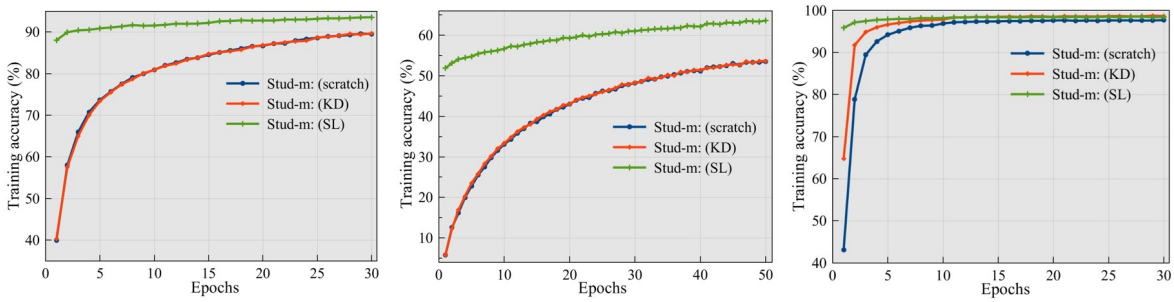
Models	Param.	P%	Sz(MB)	Acc.(%)
VGG-16				
Stud-s: (scratch)	0.56M	93.75	2.20	88.72
Stud-m: (scratch)	1.7M	81.03	6.72	91.90
Stud-s: (KD)	0.56M	93.75	2.20	88.93
Stud-m: (KD)	1.7M	81.03	6.72	92.06
Stud-s: (SL)	**0.56M**	**93.75**	**2.20**	**91.72**
Stud-m: (SL)	**1.7M**	**81.03**	**6.72**	**93.55**
Pruning [111] in [80]	2.6M	82.7	9.90	91.25
Pruning [111] in [80]	1.08M	92.8	4.30	86.85
L2PF [80]	2.6M	82.7	9.90	92.15
L2PF [80]	1.08M	92.8	4.30	89.85
Slimming [267]	2.26M	88.7	9.22	92.27
LSTM pruning [267]	2.26M	88.7	9.22	93.30
Fitnet [175]	2.50M	72.20	9.90	91.61
W-ResNet-16-8				
Stud-s: (scratch)	0.7M	93.64	2.83	93.37
Stud-m: (scratch)	2.7M	75.45	10.87	94.89
Stud-s: (KD)	0.7M	93.64	2.83	93.76
Stud-m: (KD)	2.7M	75.45	10.87	94.74
Stud-s: (SL)	**0.7M**	**93.64**	**2.83**	**94.35**
Stud-m: (SL)	**2.7M**	**75.45**	**10.87**	**95.21**
Pruning [111] in [80]	3.53M	67.9	13.86	88.73
Pruning [111] in [80]	2.38M	78.4	9.37	74.23
L2PF [80]	3.53M	67.9	13.86	93.53
L2PF [80]	2.38M	78.4	9.37	92.33

(in Megabytes: MB), experiments on student models trained with $L1$-norm are performed as discussed in Sections 9.4.1 and 9.4.2. Different pruning threshold values of T_h in the range 10^{-5} to 5×10^{-3} are used for training. Particularly, detailed result metrics after pruning such as test accuracy, % (percentage) loss of test accuracy, new number of model parameters, memory size and % (percentage) reduction in memory size are reported. All the results for comparison are obtained after fine-tuning. For control experiments and results comparison, the students are trained from scratch on the same dataset with and without $L1$-norm penalty. In addition, knowledge distillation is employed for training the student models.

9.5.2 Results and discussion

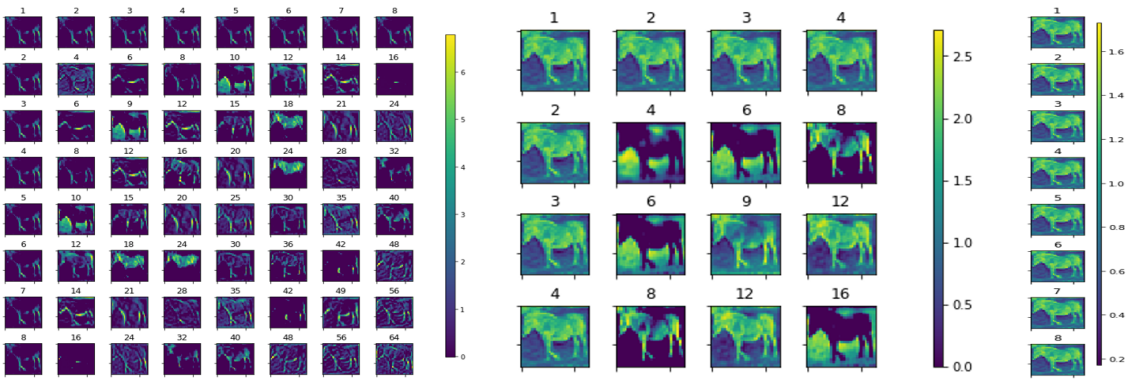
(a) CIFAR-10 dataset

The CIFAR-10 dataset contains natural color images that belong to 10 different classes. There are 50,000 and 10,000 training and testing images, respectively. Also, we perform



(a) CIFAR-10 dataset (b) CIFAR-100 dataset (c) MNIST dataset

Figure 9.3: Compressed VGG models convergence rate for the first few epochs



Teacher model Stud-m Stud-s

Figure 9.4: Convolution maps activations for the first layer of VGG-16 models trained on CIFAR-10 dataset; models have been simulated using an image from the 'horse' class. Compression enforces increased units activities in stud-m and stud-s relative to the teacher model

standard data augmentation as in horizontal flipping, translation by 4 pixels and border reflection. The teacher and student models are trained for a maximum of 250 epochs. The student models are fine-tuned for 250 epochs after initial subspace learning. The VGG-16 teacher model has 8.96M parameters (param.) of size 35MB, and achieves a test accuracy of 93.79%. W-ResNet-16-8 has 11M param. of size 43MB, and achieves a test accuracy of 95.23%. The results of comparing the compressed and trained student models are given in Table 9.1, where Param., P%, Sz(MB) and Acc.(%) denote the number of model parameters after compression, percentage of pruned parameters, model size and test accuracy. Also, scratch, SL and N.A. denote models trained from scratch, using the proposed Subspace

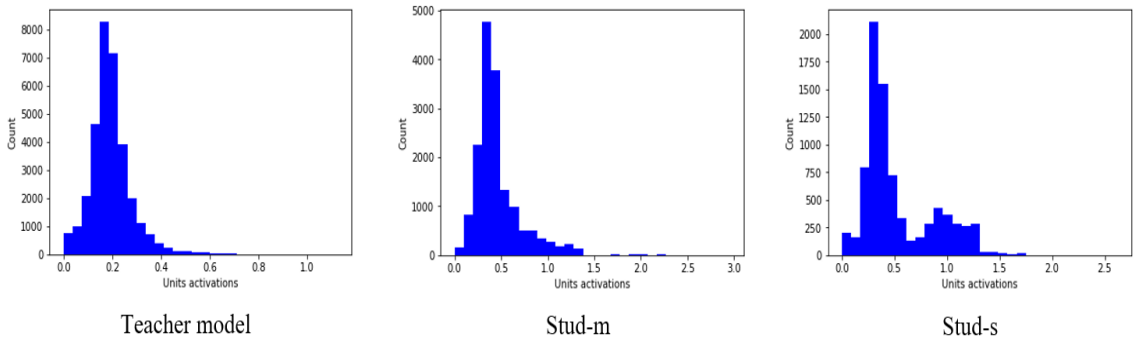


Figure 9.5: Average units' activations over CIFAR-10 dataset; the third convolution layer of trained VGG-16 models are shown. Most units in the teacher models operate with small activations; most units' activations in stud-m and stud-s have been redistributed to have larger values to compensate for compression

Learning and Not Applicable, respectively. Figure 9.3 (a) shows that the proposed VGG stud-m model trained with subspace learning (i.e. stud-m (SL)) is close to the final solution than when the same models are trained from scratch or with KD, since layerwise subspace learning can explore the reduction of model non-convexity to estimate initial model parameters that are close to one of the optima solutions for the end-to-end fine-tuning phase. During fine-tuning, stud-m with SL starts with a training accuracy of 88%. Using the VGG model, it is observed that before fine-tuning, stud-s and stud-m already yield test accuracies of 87.94% and 91.81%, respectively; this validates that local mapping functions are indeed learned during subspace learning. Figure 9.4 shows the convolution maps for the first layer of the teacher, stud-m and stud-s models. We observe that compression enforces the hidden units of stud-m and stud-s to be more active (i.e. attain higher values) as feature detectors than the units in the teacher model. To further validate this, we report in Figure 9.5 the distribution of average activations of the hidden units over the entire dataset, where increased units activitions allow the model to compensate for the pruned units (i.e. compression), and thus ameliorate performance loss; the work [65] made a similar observation on the weights of pruned models. From Table 9.1, it is seen for both VGG-16 and W-ResNet that the student models obtained using the proposed approach outperforms the same models trained from scratch and those obtained using knowledge distillation and FitNet. In addition, the test errors for deep models from earlier works with comparable depth and parameters as the

Table 9.2: Compression results for proposed stud-m model trained with LASSO and then pruned using different T_h (pruning threshold value) values on CIFAR-10 dataset; P% is in reference to the VGG teacher model used for Table 9.1

T_h ($\times 10^{-3}$)	0.01	0.05	0.1	0.5	1	5
Proposed: Acc.(%)	92.82	92.82	92.83	**92.75**	92.79	92.32
Proposed: Param.	0.78M	0.74M	0.7M	**0.56M**	0.48M	0.28M
Proposed: P%	91.29	91.74	92.19	**93.75**	94.64	96.88
Proposed: Sz(MB)	3.04	–	–	**2.20**	–	1.17
Stud-m (scratch): Acc.(%)	90.95	–	–	**89.85**	–	89.68

*Baseline (proposed stud.-m without LASSO): Param. = 1.7M, P%=81.03, 6.72MB and Acc.(%)=92.98

student models trained in this work are given.

Table 9.2 reports results on compression results for the proposed stud-m, using VGG-16 as the teacher model, trained based on latent subspace learning with a $L1$-norm penalty (λ) of 5×10^{-5} and then pruned using different T_h values. It is seen that further compression after pruning is achieved with minimal loss of model generalization performance, especially compared to student models with the same number of model parameters that are trained from scratch. For example, with 0.56M parameters, the proposed student model achieves a test accuracy of 92.75%, while a similar student model with $L1$-norm trained from scratch achieves only 89.85% test accuracy.

(b) CIFAR-100 dataset

The CIFAR-100 dataset contains natural colour images belong to 100 different classes. There are 50,000 and 10,000 training and testing images, respectively. Also, we perform data augmentation similar to the CIFAR-10 dataset. The VGG-16 teacher model has 8.96M parameters (param.) of size 35MB, and achieves a test accuracy of 72.32%. W-ResNet-16-8 has 11M param. of size 43MB, and achieves a test accuracy of 78.53%. Figure 9.3 (b) shows that stud-m trained using subspace learning (i.e. stud-m (SL)) is again closer to the final solution than when training from scratch or when using KD; during fine-tuning, stud-m with SL starts with a training accuracy of 52%. The results of comparing the compressed and trained student models are given in Table 9.3 as the test classification accuracy.

Figure 9.6 shows the convolution maps for the first layer of the teacher, stud-m and stud-s models. We observe that compression enforces the hidden units of stud-m and stud-s to

Table 9.3: Compression results on CIFAR-100 dataset

Models	Param.	P%	Sz(MB)	Acc.(%)
VGG-16				
Stud-s: (scratch)	0.56M	93.75	2.20	63.55
Stud-m: (scratch)	1.70M	81.03	6.72	67.19
Stud-s: (KD)	0.56M	93.75	2.20	64.50
Stud-m: (KD)	1.70M	81.03	6.72	67.64
Stud-s: (SL)	**0.56M**	**93.75**	**2.20**	**68.83**
Stud-m: (SL)	**1.70M**	**81.03**	**6.72**	**70.07**
Slimming [267]	3.4M	83.00	13.47	69.41
LSTM pruning [267]	5.98M	70.10	23.41	73.26
Fitnet [175]	2.50M	72.20	9.90	64.96
W-ResNet-16-8				
Stud-s: (scratch)	0.70M	93.64	2.83	71.73
Stud-m: (scratch)	2.70M	75.45	10.87	75.88
Stud-s: (KD)	0.7M	93.64	2.83	71.76
Stud-m: (KD)	2.7M	75.45	10.87	75.97
Stud-s: (SL)	**0.70M**	**93.64**	**2.83**	**73.04**
Stud-m: (SL)	**2.70M**	**75.45**	**10.87**	**77.12**

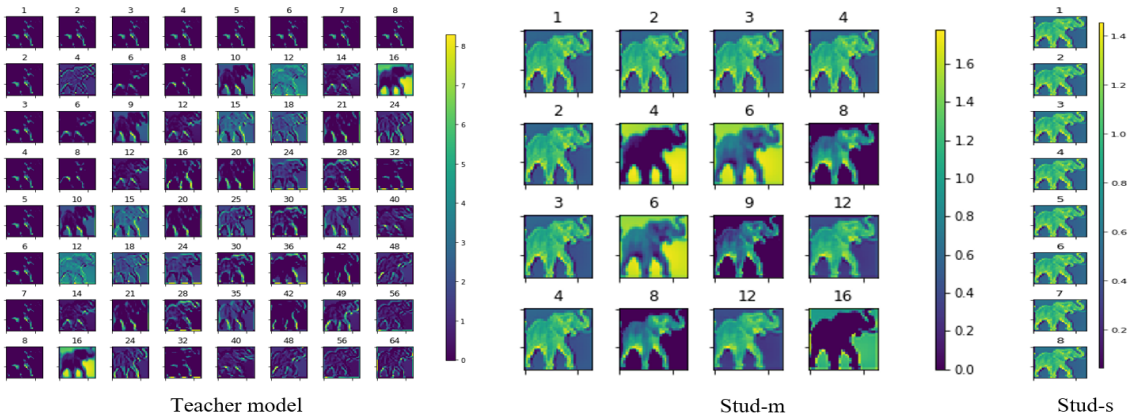


Figure 9.6: Convolution maps activations for the first layer of VGG-16 models trained on CIFAR-100 dataset; models have been simulated using an image from the 'elephant' class. It is observed that compression imposes increased units activities in stud-m and stud-s relative to the teacher model

be more active (i.e. attain higher values) as feature detectors than the units in the teacher model. To further validate this, we report in Figure 9.7 the distribution of average activations of the hidden units overall the entire dataset. From Table 9.3, it is seen that the student models obtained using the proposed approach outperforms the same models trained from scratch and those obtained using knowledge distillation. Again, it is seen that for VGG-16 and W-ResNet architectures, the proposed student models yield better results as compared

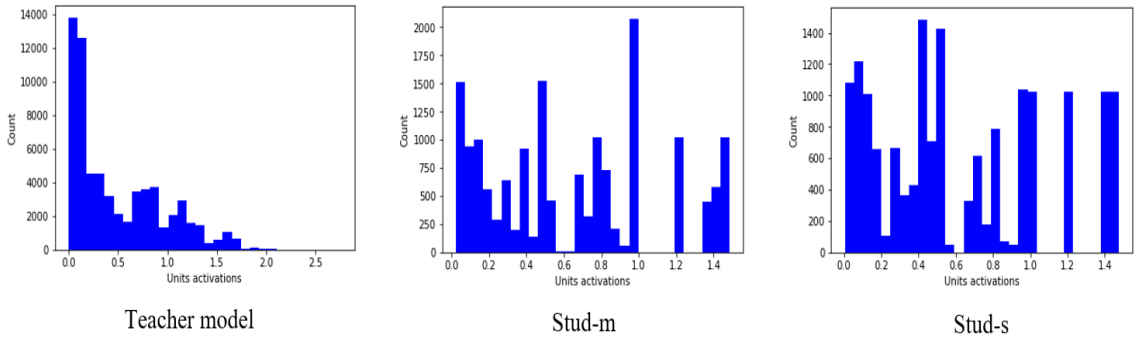


Figure 9.7: Average units' activations over CIFAR-100 dataset; the first convolution layer of trained VGG-16 models are shown. Most units in the teacher models operate with small activations; most units' activations in stud-m and stud-s have been redistributed to have larger values to compensate for compression

with student models trained from scratch, KD and FitNet. In addition, the test errors for deep models from earlier works with comparable depth as the student models trained in this work are given.

Again, using VGG-16 as the teacher model, compression results for the proposed stud-m trained based on latent subspace learning with a $L1$-norm penalty (λ) of 5×10^{-5} and then pruned using different T_h values are given in Table 9.4. It is seen that further compression after pruning is achieved with *improved* model generalization performance, especially compared to student models with the same number of model parameters that are trained from scratch. For example, with 0.86M parameters, the proposed student model achieves a test accuracy of 71.59%, while a similar student model with $L1$-norm trained from scratch achieves only 68.22% test accuracy.

We observe that compression also allows the model to perform faster inference; this is an extremely important attribute for processing multimedia such as photos in real (or near real) time. Figure 9.8 shows the inference time for compressed VGG and W-ResNet models on CIFAR-10 and CIFAR-100 test sets. The direct implication is that with minimal loss of generalization performance (i.e. Tables 9.1 & 9.3), model latency for operation can be reduced. The results in Figure 9.8 are obtained using a workstation with 32GB of RAM, i7 processor and Nvidia GTX 1080Ti GPU.

(c) MNIST dataset

Table 9.4: Compression results for proposed stud-m model trained with LASSO and then pruned using different T_h (pruning threshold value) values on CIFAR-100 dataset; P% is in reference to the VGG teacher model used for Table 9.3

T_h $(\times 10^{-3})$	0.01	0.05	0.1	0.5	1	5
Proposed: Acc.(%)	71.52	71.54	71.52	71.59	71.58	70.03
Proposed: Param.	0.95M	0.94M	0.93M	**0.86M**	0.8M	0.54M
Proposed: P%	89.40	89.51	89.62	**90.40**	91.07	93.97
Proposed: Sz(MB)	3.80	–	–	**3.50**	–	2.20
Stud-m (scratch): Acc.(%)	68.92	–	–	**68.22**	–	64.34

*Baseline (proposed stud.-m without LASSO): Param. = 1.7M, P%=81.03, 6.77MB and Acc.(%)=71.31

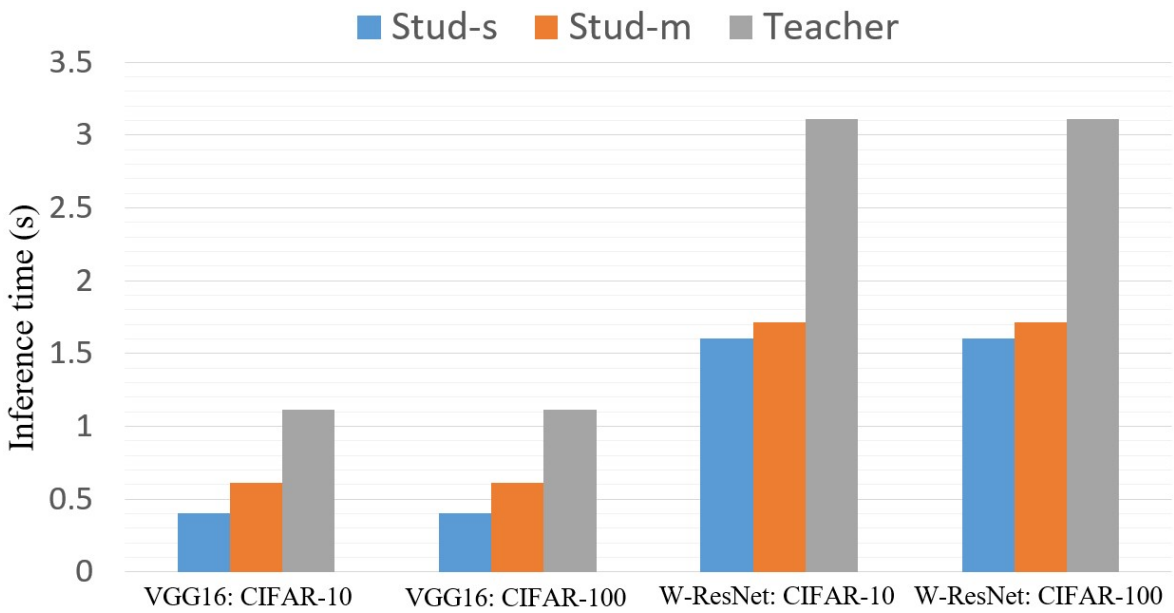


Figure 9.8: Model inference time for test set of dataset

The MNIST dataset contains binary images of handwritten digits 0-9. There are 50,000 and 10,000 test samples in the dataset. For this dataset, we train a VGG-like teacher model with 560K parameters on the MNIST dataset with no data augmentation. W-ResNet is not trained on MNIST dataset which is considered simpler to learn than CIFAR and SHVN datasets. The VGG-16 teacher model has 560K param. of size 2.32MB, and achieves a test accuracy of 99.61%. Subsequently, two student VGG-like models with 8.4K (stud-s) and 29K (stud-m) parameters are constructed and trained using the proposed compression approach. Again, Figure 9.3 (c) shows that stud-m trained using subspace learning (i.e. stud-m (SL)) converges to a solution that is in the proximity of the final solution than when training

Table 9.5: Compression results on MNIST dataset

Models	Param.	P%	Sz(MB)	Acc.(%)
Stud-s: (scratch)	8.4K	98.50	0.12	99.11
Stud-m: (scratch)	29K	94.82	0.20	99.34
Stud-s: (KD)	8.4K	98.50	0.12	99.20
Stud-m: (KD)	29K	94.82	0.20	99.49
Stud-s: (SL)	**8.4K**	**98.50**	**0.12**	**99.30**
Stud-m: (SL)	**29K**	**94.82**	**0.20**	**99.54**
Fitnet [175]	30K	91.69	0.26	99.49
Highway nets. [207]	39K	N.A.	0.30	99.43

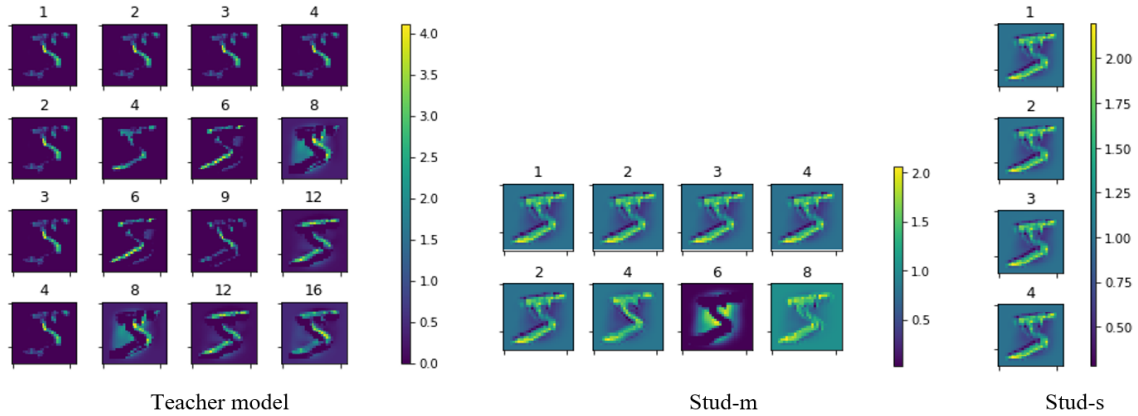
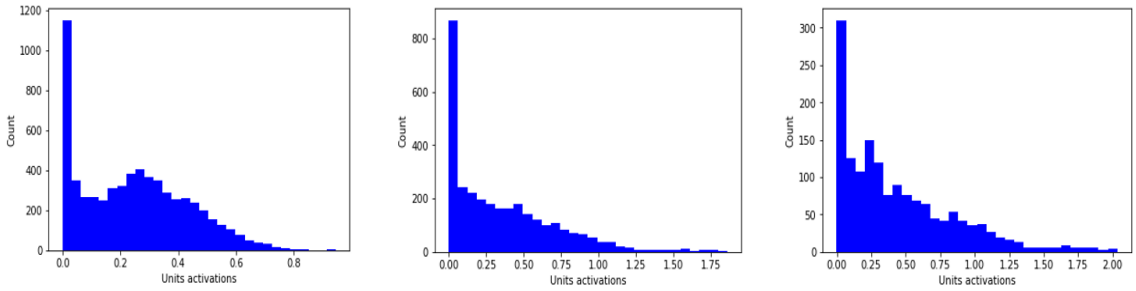


Figure 9.9: Convolution maps activations for the first layer of models trained on MNIST dataset; models have been simulated using an image from the 'five' class. Compression enforces increased units activities in stud-m and stud-s relative to the teacher model

from scratch or using KD; during fine-tuning, stud-m with SL starts with a training accuracy of 96%. It is observed that before fine-tuning, stud-s and stud-m have test accuracies of 98.36% and 99.30%, respectively; this once again shows that local mapping functions are indeed learned by the student models.

Table 9.5 shows the experimental results, along with other results from models of similar depth reported earlier works. Again, it will be seen that the student models obtained using the proposed approach outperforms the same models trained from scratch and those obtained using knowledge distillation. For example, at 94.82% and 91.38% reduction in parameters and memory size respectively, the proposed stud-m incurs a generalization loss of 0.07%, while stud-m trained from scratch incurs a generalization loss of 0.27% and the same model obtained using knowledge distillation has generalization loss of 0.12%. Also,



Teacher model Stud-m Stud-s

Figure 9.10: Average units' activations over MNIST dataset; the first convolution layer of trained models are shown. It is observed that most units in the teacher models operate with small activations, while most units' activations in stud-m and stud-s have been redistributed to have larger values to compensate for compression

Table 9.6: Compression results for proposed stud-m model trained with LASSO and then pruned using different T_h (pruning threshold value) values on MNIST dataset; P% is in reference to the teacher model used for Table 9.5

T_h $(\times 10^{-3})$	0.01	0.05	0.1	0.5	1	2
Proposed: Acc.(%)	99.44	99.44	99.44	99.40	99.43	**99.38**
Proposed: Param.	20.07K	17.46K	15.21K	9.29K	7.84K	**6.53K**
Proposed: P%	96.42	96.88	97.28	98.34	98.60	**98.83**
Proposed: Sz(MB)	0.19	–	–	0.15	–	**0.14**
Stud-m (scratch): Acc.(%)	99.44	–	–	99.32	–	**99.27**

*Baseline (proposed stud.-m without LASSO): Param. = 29K, P%=94.82, 0.2MB and Acc.(%)=99.54

stud-m with 29K parameters that is obtained using the proposed approach outperforms the Fitnet with 30K parameters. In addition, the test errors for deep models from earlier works with comparable depth as the student models trained in this work are given. Figure 9.9 shows the convolution maps for the first layer of the teacher, stud-m and stud-s models. We observe that compression enforces the hidden units of stud-m and stud-s to be more active (i.e. attain higher values) as feature detectors than the units in the teacher model. To further validate this, we report in Figure 9.10 the distribution of average activations of the hidden units overall the entire dataset.

Compression results for the proposed stud-m trained based on latent subspace learning with a $L1$-norm penalty (λ) of 10^{-3} and then pruned using different T_h values are given in Table 9.6. It will be seen that further compression after pruning is achieved with minimal

Table 9.7: Compression results on SVHN dataset

Models	Param.	P%	Acc.(%)
Stud-m: SL	**1.7M**	**81.03**	**96.74**
Stud-m: (KD) [73] with $\lambda = 0$	1.7M	81.03	96.04

loss of model generalization performance, especially compared to student models with the same number of model parameters that are trained from scratch. For example, with 6.53K parameters, the proposed student model achieves a test accuracy of 99.38%, while a similar student model with $L1$-norm trained from scratch achieves only 99.27% test accuracy.

(d) SVHN dataset

For this dataset, which has over 600,000 samples, the aim is to demonstrate that one can easily construct a model compression scenario in which neither knowledge distillation nor Fitnet is applicable. Particularly, one can assume that only a trained teacher model and unlabeled data that comes from the same distribution as the original data used for training teacher model are available. This scenario is not uncommon in practice where the collection of labeled data is quite expensive. For this situation where knowledge distillation and Fitnet cannot be employed since they rely on labeled data, we show that the proposed compression approach yields very promising results using the SVHN dataset. Furthermore, it demonstrates that the proposed approach can be employed for large datasets such as SVHN.

The SVHN (Street View House Number) dataset contains real-life Google street view images with digits in the range 0-9. There are 73,257 and 26,032 digits for training and testing, respectively. In addition, there are 531,131 *somewhat easier* digits that are used as extra training data. For this dataset, the standard training is not followed as in [58]. Instead, an interesting scenario that allows the validation of applicability of the proposed approach is crafted. A VGG teacher model of 8.96M parameters is trained on the training data (i.e. 73,257 samples) for 50 epochs. We assume that the extra training data provided is unlabeled. For model compression, *only* the extra training data are used for obtaining a trained student model of 1.7M (stud-m) parameters based on the proposed compression approach; the student model is trained for 50 epochs. The teacher model achieves a test accuracy of

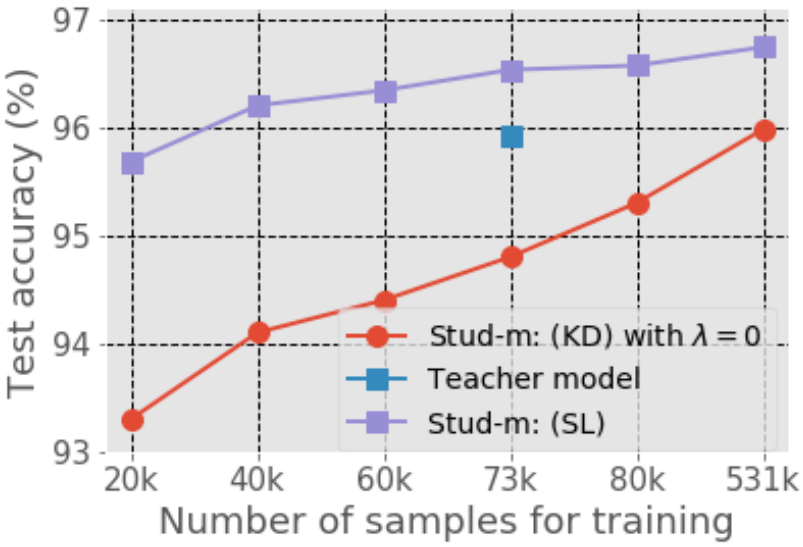


Figure 9.11: Student model test classification accuracy with number of training samples for the SVHN dataset

95.92%.

Table 9.7 shows the classification accuracy of the trained models on the SVHN test data (i.e. 26,032 samples) that were not used for training the teacher and student models; a test accuracy of 96.74% is achieved. It is observed that the student model outperforms the teacher model. This observation is interesting, since we would expect the student model to slightly lag the teacher model in test performance. The student model was never trained on any of the data samples found in the training set for the teacher model. Our explanation is that in contrast to the teacher model that acquires knowledge using only the original training set, the student model in this case acquires knowledge in two ways (i) by learning the latent representations of the teacher model, it indirectly learns from the original training set (ii) learns from the newly provided unlabeled data. KD requires true data labels y for training (i.e. as in (1)); [73] particularly mentioned that this is the formulation for which KD works well, and no results without true data labels were reported. However, one can set $\lambda = 0$ in (9.1) so that y is not required; the first term is ignored and we arrive at the scenario similar to [20]. This approach is thus employed for testing the performance of KD using the same training settings as in subspace learning; test accuracy of 96.04% is obtained for KD as

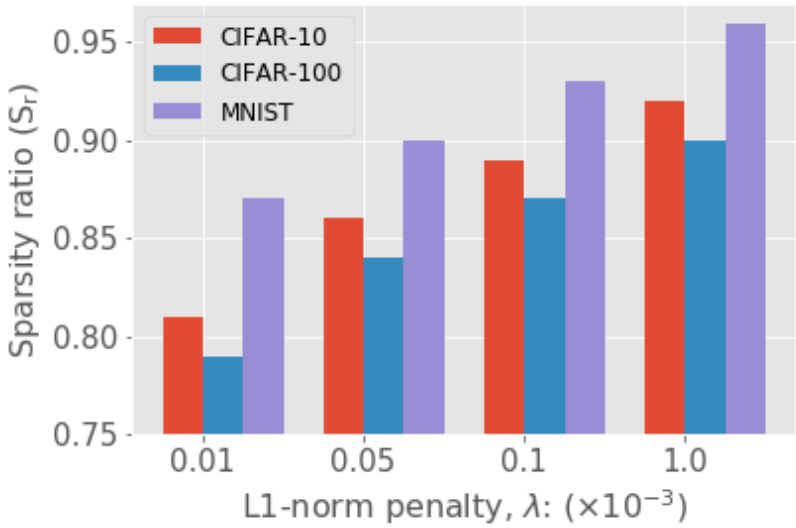


Figure 9.12: Sparsity ratios for different values of $L1$-norm penalty term, λ

given in Table 9.7.

Figure 9.11 shows how the size of the unlabeled dataset impacts the performance of the student model; the teacher model is used as a reference. For this experiment, the number of unlabeled data samples used for learning the student model is varied from 20,000 to 531,131. It is shown that the student model outperforms the teacher model starting from when 40,000 samples and beyond are used for learning. Also, it is shown that the performance of the proposed approach does not degrade drastically as the size of unlabeled dataset available for model compression decreases. For example, with 20,000 samples, the student model already achieves a test accuracy of 95.68%; this corresponds to a generalization loss of 0.25%. In addition, Figure 9.11 suggests that increasing the size of unlabeled data used for model compression should result in further improvement of the student model performance.

9.5.3 Ablation studies

In this section, ablation studies for experiments that employed LASSO regularization for further model compression are presented. First, the sparsity ratio (i.e. amount of sparsity) that

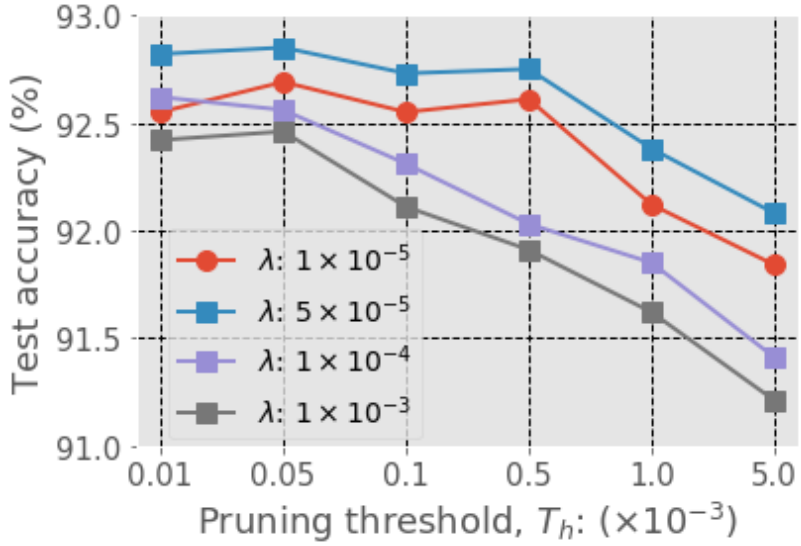


Figure 9.13: CIFAR-10 test accuracy for different pruning threshold, T_h, and L1-norm penalty, λ

can be induced for different values of the $L1$-norm penalty, λ, is studied; the sparisty ratio, S_r, is the ratio of the number of zero-valued parameters to the total number of model parameters. The sparsity ratio results for the models trained with $L1$-norm penalty on CIFAR-10, CIFAR-100 and MNIST datasets discussed in Sections 9.5.2 (a), 9.5.2 (b) and 9.5.2 (c) are given in Figure 9.12. It is seen that sparsity ratios increase with an increase in the value of $L1$-norm penalty. For example, using $L1$-norm penalty of 5×10^{-4} for CIFAR-10, CIFAR-100 and MNIST datasets, sparsity ratios of 0.87, 0.84 and 0.9 can be induced, respectively.

In addition, we observe how the $L1$-norm penalty, λ, and pruning threshold values, T_h, hyperparameters impact compression results based on achieved test accuracies. For experiments, the same models and settings on CIFAR-10, CIFAR-100 and MNIST datasets as in Sections 9.5.2 (a), 9.5.2 (b) and 9.5.2 (c), respectively, are used. Figures 9.13, 9.14 & 9.15 show results on CIFAR-10, CIFAR-100 and MNIST datasets, respectively. It is seen that $\lambda = 5 \times 10^{-5}$ yields best accuracy given compression results for the models trained on CIFAR-10 and CIFAR-100 datasets, while $\lambda = 1 \times 10^{-3}$ yields best accuracy given compression results for the model trained on the MNIST dataset. Finally, note that the sparsity ratios for different values of λ in Figure 9.12 corresponds with the test accuracies reported in

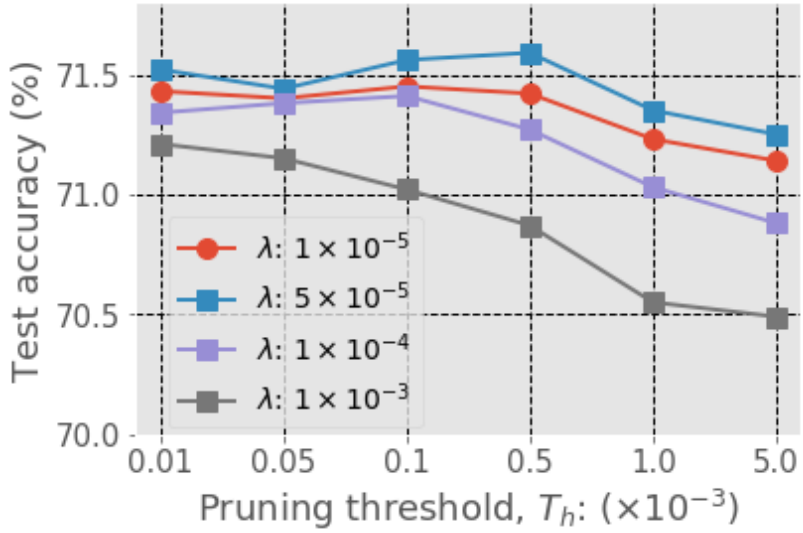


Figure 9.14: CIFAR-100 test accuracy for different pruning threshold, T_h, and $L1$-norm penalty, λ

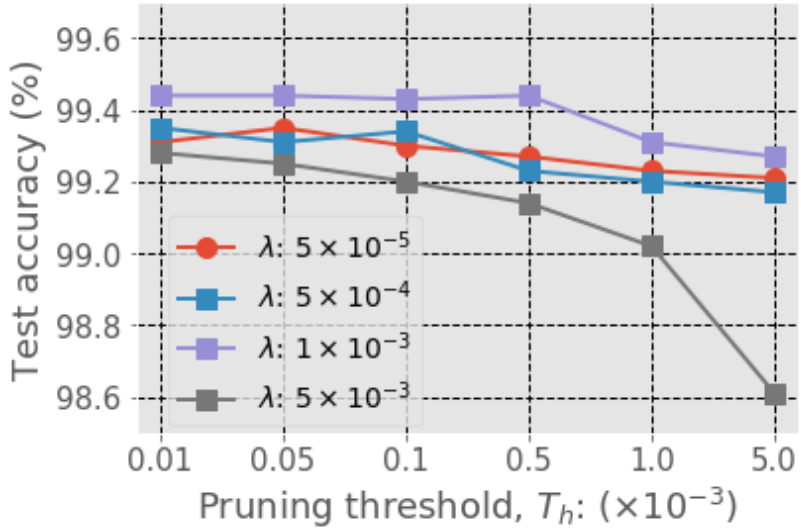


Figure 9.15: MNIST test accuracy for different pruning threshold, T_h, and $L1$-norm penalty, λ

Figure 9.13, 9.14 & 9.15 for similar datasets. For instance, on MNIST dataset, $\lambda = 1 \times 10^{-3}$ corresponds to a sparsity ratio of 0.96 from Figure 9.12, and a test accuracy of 99.5% using $T_h = 5 \times 10^{-4}$ from Figure 9.15.

9.6 Conclusion

Deep neural networks have proven powerful for understanding multimedia, which include photos, videos and texts. However, the large memory size of these networks and long inference time can limit their applications on electronic devices with small computational resources. As such, we propose in this work, a generic method to compress large (teacher) models into smaller ones (student models). As deep networks typically learn lots of redundant features and are over–parameterized, therefore there exists with high likelihood a feature subspace that approximately captures the same latent representation. The essence of the approach is to learn this feature subspace in a layer-wise fashion to construct a deep student model. Moreover, we show that LASSO penalty results in improved compression results when employed for the proposed student model compared to a student model trained from scratch. Using the proposed compression method, extensive experiments on four standard datasets show that a drastic reduction of model parameters can be achieved without incurring significant loss of generalization performance. It is interesting that the proposed approach suffices in the cases where knowledge distillation and FitNets cannot be employed; that is, unavailability of labeled data. The experimental results of the proposed student models outperform those trained from scratch, using knowledge distillation, FitNet and other models with comparable number of parameters or depth. In addition, results obtained using the proposed approach suggests that abundant unlabeled data (which is usually available in real-life) can be leveraged to improve compression results such that the student model even outperforms the teacher model.

For future work, it will be interesting to explore the automatic estimation of the suitable number of convolution filters and hidden units in the Student model that results in the optimal subspace learning; this is as opposed to hand engineering of the Student model as seen in this work and similar works [73, 125, 175]. The estimation scheme should allow the best reconstruction of the latent representations of the Teacher model using an optimal number of hidden units. A promising direction for this is the gradual increase of the Student model's convolution filters and hidden units based on the maximization of the mutual information

between the Student and Teacher models. The goal is to stop adding more convolution filters and hidden units to the Student model when the mutual information objective ceases to improve.

Chapter 10

Facial Expression Recognition with Multimodal Learning of RGB-depth Data

The previous chapters studied the training and compression of very deep neural networks. In this chapter, we focus on the application of very deep neural networks to multimodal learning. Namely, the problem of facial expression recognition is considered. Humans use facial expressions successfully for conveying their emotional states. However, replicating such success in the human-computer interaction domain is an active research problem. In this chapter, we propose deep convolutional neural network (DCNN) for joint learning of robust facial expression features from fused RGB and depth map latent representations. We posit that learning jointly from both modalities result in a more robust classifier for facial expression recognition (FER) as opposed to learning from either of the modalities independently. Particularly, we construct a learning pipeline that allows us to learn several hierarchical levels of feature representations and then perform the fusion of RGB and depth map latent representations for joint learning of facial expressions. Our experimental results on the BU-3DFE dataset validate the proposed fusion approach, as a model learned from the joint modalities outperforms models learned from either of the modalities.

10.1 Introduction

Facial expressions have been used successfully and effectively by humans for communicating their emotional states. The six basic facial expressions include *anger*, *disgust*, *fear*, *happiness*, *sadness* and *surprise*. Our excellent ability to correctly read-off facial expressions from people usually lead to a host of corresponding responses. For example, when a teacher read-offs an expression of surprise on the faces of students during a lecture, he may decide to restate his point in a more comprehensible way for the students. In general, other areas where facial expression recognition play critical roles for improving communication and responses include counselling, interrogation, reward games, etc.

Interestingly, tasking computers to perform facial expression recognition (FER) is an active research problem in the human-computer interaction domain [180]. A major motivation for this is to allow a computer adapt its response based on the facial expression of the user; this consequently fosters better human-computer interaction. However, recognizing facial expressions of subjects with reasonably high accuracy is not an easily achievable task for computers. Several works have studied FER and proposed different approaches for improving the performance of FER systems. Many of these works rely on learning facial expression features via appearance, fiducial landmarks or detecting action units first, and then subsequent classification [257, 121]. For feature based FER systems, learning highly discriminative and robust facial expression is very critical. This approach is challenging as the features that discriminate the different facial expression are quite subtle [217]; the usually more abundant features pertaining to the identity of the subject could easily dominate the discriminative features for the different facial expressions.

In recent times, deep learning has become very useful for many computer vision applications [62, 90]. One of the main motivation behind deep learning is that learning several hierarchical levels of feature abstractions fosters the disentanglement of the different aspects in the training dataset, as the different levels in the model can represent different aspects

in the training dataset [105, 12]. Moreover, deep neural networks are powerful function approximators, with the capacity for learning complex and highly varying functions [5].

In this chapter, we approach FER via appearance based representation using deep learning and the fusion of RGB and depth map latent representations. Our motivation for the proposed approached lies in learning jointly from depth map and RGB modalities. Since, we propose an appearance based modelling of FER, we employ deep convolutional neural network (DCNN) for learning the *subtle* features that discriminate the six basic different facial expressions. Particularly, we posit that learning jointly from depth map and RGB modalities improves the robustness of discriminative features learned for FER as against learning singly from either of the two modalities. We take inspiration from the demerits of employing either of the two modalities separately for learning facial expression features that are highly discriminative. RGB data can effectively capture very rich information about facial expressions; however, RGB data are quite susceptible to severe corruption due to illumination variations. In fact, RGB data may be rendered considerably ineffective for learning very discriminative features. Conversely, depth map data are modestly tolerant of illumination changes and therefore features learned from such data are more stable and robust. In addition, depth map data carry important information describing geometric relations in an image. However, in well setup and stable environmental settings, depth map data may be less rich in information as compared to RGB data. The contributions of this chapter are as follows:

1. We posit that learning jointly from depth map and RGB modalities result in a more robust classifier for FER as against learning singly from either of the two modalities. Our extensive experiments and results on the BU-3DFE dataset validates our position.

2. We construct and describe a pipeline based on transfer learning and deep convolutional neural network (DCNN) that are later used to learn and fuse several hierarchical levels of feature representations from RGB data and depth map.

3. We provide a comparison between our method and state-of-the-art approaches on BU-3DFE dataset. Particularly, the result of our fusion approach is better than many state-of-the-art results.

The rest of this chapter is organized as follows. Section 10.2 discusses related works along with their merits and demerits for FER. Section 10.3 describes the necessary background and the problem formulation followed by the proposed approach, in Section 10.4. In Section 10.5, we give the details of experiments and results along with discussions. In Section 10.6, we conclude the chapter by highlighting our key findings.

10.2 Literature review

Several studies have been carried out on facial expression recognition (FER). Many of these works essentially rely on two stages of information processing: (1) features extraction [9] (2) classification of extracted features [128, 147]. Occasionally, there is a feature selection stage as an intermediate stage between the feature extraction stage and classification [183]. Additionally, FER systems can be developed based on geometric [184] or appearance [247] features modelling. For geometric based FER modelling, [131] relied on Haar-based features and the Viola-Jones' algorithm for detecting the eye regions; the work emphasized the challenges of accurately detecting the eyes and other facial regions from which features for FER are extracted; for classification a total of 26 geometric features where extracted and fed into a self-organizing map for classification. In [244], estimation of principal curvature was employed for labelling the vertices of 3D facial surface individual models. The work [244] then segmented facial surfaces into expressive regions and applied histogram statistics on the segmented regions. Finally, the features obtained from the histogram statistics were used to train a linear discriminant classifier for FER. In [214], a FER system was proposed based on the combination of Bayesian Belief Net (BBN) and statistical facial features model. The work described an approach where manual landmarking for facial features extraction is eliminated. Another interesting work [215] proposed using 2D and 3D features for FER. The work employed incremental Parallel Cascade of Linear Regression (iParCLR) for simultaneously localizing fiducial landmarks from 2D and 3D scans. For extracting features from the 2D scans, a novel Histogram of Second Order Gradients (HSOG) and first-order gradient based SIFT descriptor were used. For extracting features from the 3D scans, Histogram of

mesh Gradients (meshHOG) and Histogram of mesh Shape index were used. Subsequently, the features obtained from both 2D and 3D scans were used to train a support vector machine (SVM); further improvement in result was reported by fusing 2D and 3D modalities. In [66], an elaborate FER system was proposed composing low pass filtering, eye, nose, lip corner and eyebrow corner detection; features were extracted from detected regions of interests using local binary pattern (LBP) encoding. Later, support vector machine (SVM) based classifier is trained on a feature space estimated with Principal Component Analysis (PCA) from LBP encoding. One obvious demerit of the geometric based modelling is the elaborate arrangement for the detection of regions of interest in face images.

Alternatively, other works pursue appearance based modelling for FER. Here, arrangement for elaborate detection of regions of interest in face images is eliminated; instead, through highly robust learning schemes, facial expression features are learned (or extracted) from face images with no or minimal processing. For example, [122] proposed a boosted deep belief network for performing feature learning, selection and classification all together as a unified framework. In another work [93], a deep neural network was employed for FER, and they report state-of-the-art results on extended Cohn-Kanade (CK+) dataset and the Toronto Face Dataset (TFD). More interestingly, they show that the features learned by the deep network are akin to facial action units (FAUs) for facial expressions.

Another approach for FER which have been studied in several works relies on the detection of specific FAUs based on the facial action coding system (FACS) [262]. In [44], FAUs were explored for composing new expressions which they referred to as compound expressions, while [231] proposed a FER system based on the detection of some specific FAUS in face images.

10.3 Problem formulation

Learning robust discriminative features is critical to the performance of appearance based approaches for FER. However, learning robust discriminative features directly from raw data is not at all trivial. Deep neural networks have large representation capacity allowing the

learning of complex target functions, and therefore seems a natural choice for modelling such a daunting task. For example, [248] employed an ensemble of deep neural networks based on transfer learning; firstly, they trained their models on a much larger dataset (i.e. 2013 facial Expression Recognition (FER) Challenge dataset), then fine-tuned the models on the target dataset-Static Facial Expressions in the Wild (SFEW) dataset.

However, training DCNNs require massive datasets as deep models typically compose lots of parameters; with small datasets, deep models have the tendency to quickly over-fit [245]. One solution which has been explored in many works that employed deep models for small datasets [245, 146] is transfer learning, where the features learned by a deep model on a generic and very large dataset are transferred to another task. Here, the hope is that the features learned by the pre-trained model are considerably general enough for other tasks; this is especially the case when the dataset on which the model has been pre-trained considerably shares similarity with the target dataset. Otherwise, one cannot guarantee the performance of feature transfer when the target dataset is quite different from the one on which the model has been pre-trained; in this case, the suitability of feature transfer can only be determined via experimentation [245]. Since, the dataset used in this work is relatively small, we experiment with transfer learning to determine the scenario (i.e. data modality) where it is useful for improving feature learning.

Considering the aforementioned strengths of depth map and RGB data, we formulate an approach for extracting more interesting features for facial expression recognition. Namely, the proposed approach is aimed at solving the following problems:

- Feature learning from a small dataset in cases where transfer learning is not effective (the depth map case).
- More effective learning of facial expression features from both depth map and RBG modalities via the fusion of their latent representations.

10.4 Proposed facial expression framework

In this section, we relate how the proposed approach addresses the problems that we propose to solve as mentioned in Section 10.3. We give the details of the dataset used in this work, data pre-processing, proposed learning pipeline, model architectures, and considerations for the different pipeline constructions. The highlights of the proposed facial expression recognition framework and the problems that are tackled are as follows:

- We can leverage both depth map and RGB modalities for extracting more robust discriminative features for FER. The aim is that the individual strength of the modalities can be used compliment each other via the fusion of extracted latent representations.
- We consider the usefulness of hierarchical feature representations to construct a pipeline that allows us to learn several levels of latent representations and perform fusion of latent representations of RGB and depth map modalities; such a pipeline should allow the different levels learn different and interesting aspects of the training data.
- We rely on the state-of-the-art pre-trained models (i.e. ResNet50 [67] and VGG19 [194]) where they are helpful for learning several levels of latent representation; otherwise, we train our model from scratch and rely on other training schemes such as batch normalization [84] to improve optimization and generalization.

10.4.1 Dataset

For validating the proposed approach in this work, we use the BU-3DFE dataset [244] that is composed of 100 subjects (56 female and 44 males) with a total of 2500 textured 3D scans along with corresponding 2D (RGB) data. Furthermore, each subject perform the six basic facial expressions at 4 different intensity levels; see Figure 10.1 [4].

10.4.2 Data pre-processing

The BU-3DFE dataset offers both 3D and 2D (RGB) data for all expressions contained therein. Since, the conventional DCNN can be employed directly only for structured data

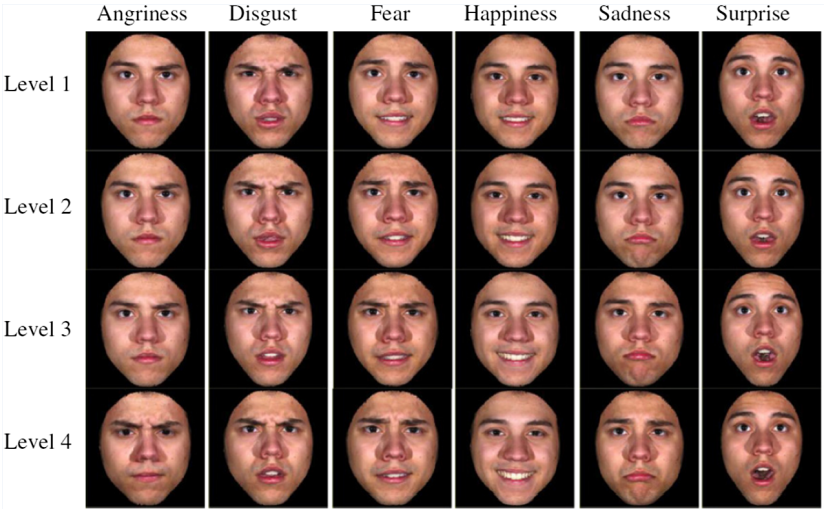


Figure 10.1: BU-3DFE dataset facial expressions [4]

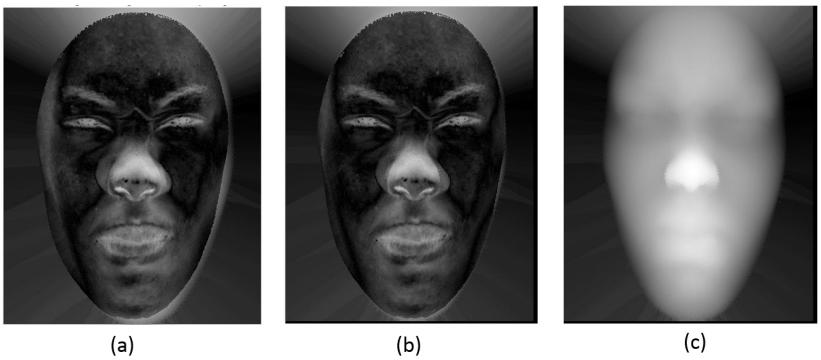


Figure 10.2: BU-3DFE depth map pre-processing

such as images, we obtain depth map images from the 3D data and then perform registration using the corresponding RGB images via maximization of mutual information. The registration of depth map images is meant to improve the joint learning of facial features from both depth map and RGB data; this is straightforward since we want the DCNN to strictly capture the same aspect of the facial images during training. For example, Figure 10.2(a) shows the superposition of the obtained and unregistered depth map over the grayscale transform of the RGB data for an expression; Figure 10.2 (b) shows the superposition of the registered depth map over the grayscale transform of the RGB data for the same expression; Figure 10.2(c) shows the registered depth map. Furthermore, we employ an algorithm that

captures the face images in bounding boxes, eliminating the redundant background data; this should make learning facial expression features more concise.

10.4.3 Learning pipeline

Since, we propose to show that learning jointly from depth map and RGB data allows us to learn more robust features with higher discriminative power than learning singly from either of the two modalities, we construct three different pipelines to validate the argument presented in this work. We refer to the three different pipelines as PL-depth map, PL-RGB and PL-fusion; where, PL-depth map, PL-RGB and PL-fusion denote the constructed pipelines for depth map data, RGB data and their fusion, respectively. For the depth map and RGB data, we experiment with transfer learning using ResNet50 [67] and VGG19 [194] models that were pre-trained on the imagenet dataset. The three pipelines are discussed below.

PL-depth map (a)

We experiment with transfer learning using ResNet50 and VGG19 for the pre-processed depth map images. However, we observe that employing the pre-trained models for the depth map data resulted into poor performance. Our explanation is that the features learned by the pre-trained models are considerably unuseful for the depth map images. This is not surprising, as the target data is quite dissimilar to the one that the pre-trained models were trained on; the target data, depth maps, are range images while the data (i.e. imagenet dataset) used for pre-training are natural colour images.

Therefore, we opt for training a DCNN from scratch on the depth map images. Firstly, we resize the depth map images from the original size of 512×512 pixels to 64×64 pixels to reduce training time and computational requirement. We then construct a deep model with 5 convolution layers and 3 max pooling layers. The architecture of the DCNN is as follows: {input:depth map}-{conv layer:10 FM}-{conv layer:20 FM}-{max pool:2×2}-{conv layer:30 FM}-{conv layer:40 FM}-{max pool:2×2}-{conv layer:50 FM}–{max pool:4×4}-{FC:300 units}-{FC:300 units}-{softmax:6 units}; where, FM and FC denotes feature maps and fully connected layers, respectively; all convolution operations use filters of size 6×6.

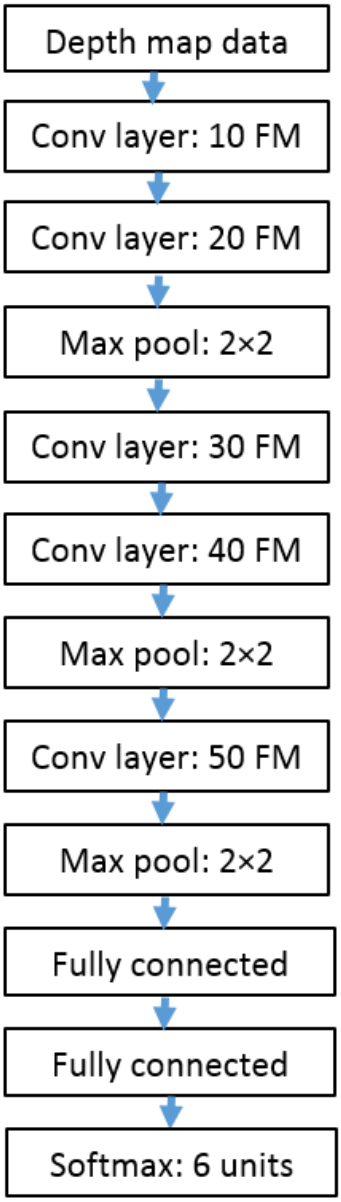


Figure 10.3: Learning pipeline for depth map modality: PL-depth map

The constructed pipeline is shown in Figure 10.3.

PL-RGB (b)

For the RGB data, we find that the pre-trained networks ResNet50 and VGG19 work well, hence we do not construct models that are trained from scratch as carried out for

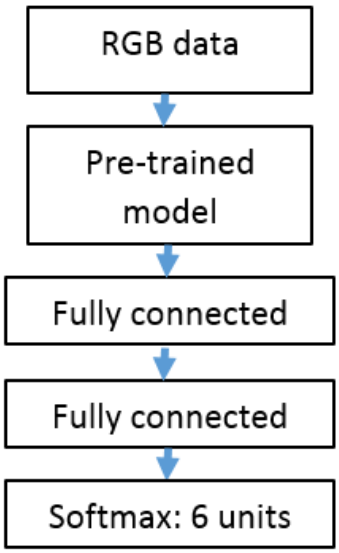


Figure 10.4: Learning pipeline for RGB modality: PL-RGB

the depth map data. We take the pre-trained models, remove the fully connected layers including the softmax and stack two newly initialized fully connected layers and a 6-way softmax layer for training to classify the six basic facial expressions. The full architecture of the PL-RGB pipeline is as follows: {input:RGB data}-{pre-trained model}-{FC1}-{FC2}-{softmax:6 units}; where, 'pre-trained model' is either ResNet50 or VGG19; for ResNet50, FC1 and FC2 have 1000 and 700 units respectively; while for VGG19, FC1 and FC2 have 500 units each. The constructed pipeline is shown in Figure 10.4. Furthermore, for the ease of referral, we tag the pipeline with pre-trained ResNet50 as *PL-RGB-ResNet50*, and the other pipeline with pre-trained VGG19 as *PL-RGB-VGG19*.

PL-fusion (c)

Here, we construct a pipeline for the joint learning of latent representations obtained from the depth map and RGB modalities. We take the following considerations in constructing the pipeline to achieve two important goals: (1) learning several hierarchical representations for both data modalities (2) fusing the separately learned high level representations and then learning jointly from the new and much richer representations. The constructed pipeline is shown in Figure 10.5. The depth map and RGB pipelines before the fusion of latent

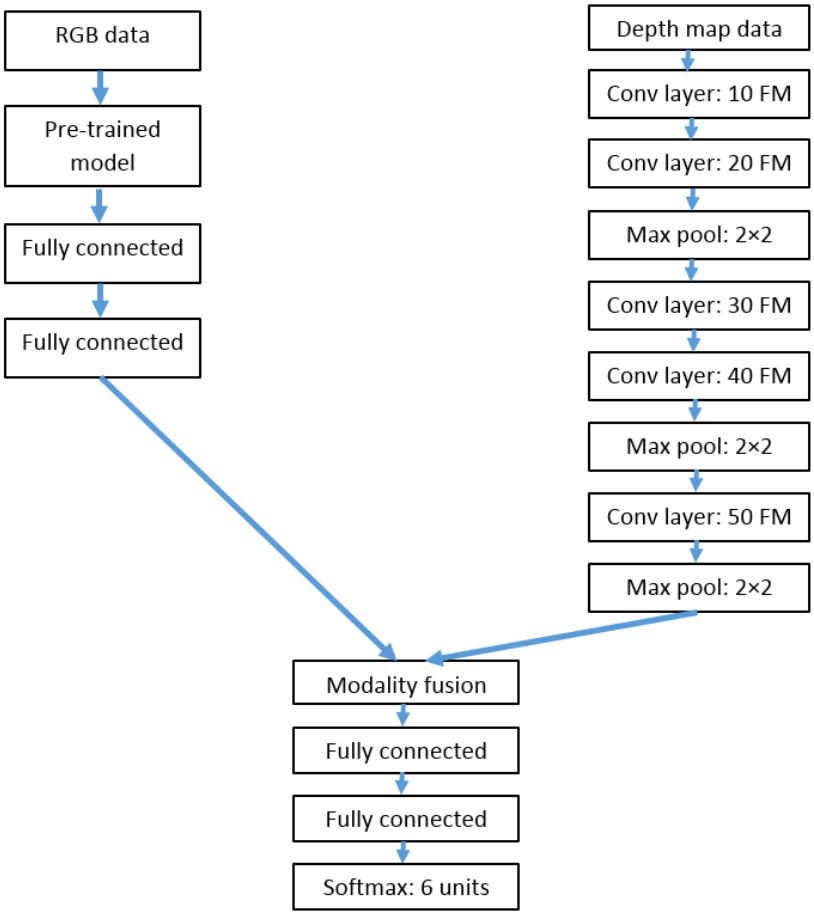


Figure 10.5: Depth map and RGB latent representation fusion pipeline: PL-fusion

representations have the same construction as though they operate singly; this scenario can be seen as a form of late fusion. Note that for the fusion of depth map and RGB modalities, we concatenate the latent representations obtained from both depth map and RGB data, and then learn two new levels of latent representations before feeding into a 6-way softmax classifier layer.

10.5 Experiments

10.5.1 Pipeline training

For all the pre-training scenarios reported in this work, we perform global average pooling at the last layer to reduce spatial dimensionality of data before feeding into the fully connected layers. Since we employ batch normalization [84] for improving generalization and optimization, we initialize all the newly added layers via He initialization [68] to improve convergence rates of models during training. Our observation aligns with [84] on the use of dropout and batch normalization together for training; in our experiments, using dropout along with batch normalization did not bring any improvement in learning. Hence, we did not employ dropout for all newly added layers reported in this work. Since, our pipelines are based on several stacks of feature representations, all the pipelines in this work use hidden units with the rectified linear function for activation; again, this is aimed at tackling units saturation and improving gradients condition for fast convergence during training.

Furthermore, we employ mini-batch gradient descent optimization for training all models. Particularly, we rely on the adaptive moment estimation (adam) technique as in [214] for mini-batch gradient descent optimization so that we can be less careful about the set learning rate for model training. Nevertheless, we set the learning rate for all models in this work to a considerably small value of 0.003, and training all the different pipelines for a maximum of 1000 epochs.

10.5.2 Experimental setup

As mentioned earlier, we follow the common protocol as in [244, 225, 214, 266] and many other works for choosing training and testing data. We randomly select 60 subjects from the BU-3DFE dataset; from these selected subjects, we randomly select 54 subjects for building the training data and the remaining 6 subjects for building the test data; this is akin to a 10-fold cross validation training scheme. In addition, we use only the third and fourth intensity levels of facial expressions. Again, in conformity with the training and model evaluation protocol found in [244, 225, 214, 266] and many other works, we fix the randomly

Table 10.1: Experimental results for the different pipelines

Approach	Test acc. (%)
Depth map+pre-trained ResNet50*	61.11
Depth map+pre-trained VGG19*	28.06
Depth map: PL-depth map	84.72
RGB+ pre-trained ResNet50: PL-RGB-ResNet50	82.92
RGB+ pre-trained VGG19: PL-RGB-VGG19	81.25
Depth map+RGB: PL-fusion-ResNet50	87.08
Depth map+RGB: PL-fusion-VGG19	**89.31**

selected 60 subjects, and perform for a total of 200 experiments for the random selection of 54 subjects as training data and remaining 6 subjects as test data similar to a 10-fold cross-validation scheme; this is meant to reduce sampling bias on training and testing data for model training and evaluation. All the experiments and results given in this work are based on this training and testing protocol.

However, we note that some other works [225] among others follow a different training and evaluation protocol where the selected 60 subjects are not fixed, but are randomly selected for each experiment. In this work, we do not consider such a training and evaluation experimental setup.

10.5.3 Results and discussion

We report in Table 10.1 the results of the experiments performed for the different learning pipelines discussed in Section 10.4. We observe that the depth map and RGB modalities achieve promising results when they are used separately for FER. When the depth maps only are used for training a DCNN from scratch (i.e. PL-depth map) as described in Section 10.4.3 (a), we obtain a test accuracy of 84.72%. Also, when the RGB data are used for obtaining features from pre-trained models, we obtain a test accuracy of 82.92% for ResNet50 and 81.25% for VGG19. Interestingly, combining both depth map and RGB modalities, we are able to reach a test accuracy of 87.08% using pre-trained ResNet50 for the RGB data (i.e. PL-fusion-ResNet50 in Table 10.2) and 89.31% using pre-trained VGG19 for the RGB data (i.e. PL-fusion-VGG19 in Table 10.2); for both fusion experiments, DCNNs are trained

from scratch on the depth map data.

We include in Table 10.1 the results of parallel experiments obtained using pre-trained models ResNet50 and VGG19 on depth maps; these are the first two results given in Table 10.1 with asterisk. Particularly, we observe poor performance in this scenario as features do not seem transferable from the imagenet pre-trained models to depth maps. When we rely on features obtained from the pre-trained ResNet50 for the depth map data, we obtain a poor test accuracy of 61.11%; using the pre-trained VGG19 in this setting, a much worse test accuracy of 28.06% is obtained. Here, we note that the results are not at all surprising as there is a large dissimilarity between the imagenet data on which the ResNet50 and VGG19 were pre-trained and the target data in this work which are depth maps; features obtained from the pre-trained models in this setting are far from the optimal representations of the target data. In fact, we note that [245] also acknowledged that the transferability of features decreases with the increase in disparity between the base task (i.e. pre-training data) and target task (i.e. target data). Hence, we did not consider these models useful enough for constructing any of the pipelines described in this work; rather, we chose to train a DCNN from scratch on the depth map data.

Table 10.2 shows the results (i.e. test accuracy in %) for the different pipelines constructed in this work, along with state-of-the-art results reported in other works that used a similar training and testing protocol to ours on the BU-3DFE dataset. Firstly, we observe that our result based on modality fusion, outperforms many state-of-the-art results on the BU-3DFE dataset as reported in Table 10.2. For example, [114] employed deep representation, explored geometric and photometric attributes for obtaining 6 different types of 2D facial maps, after which it relied on three pre-trained VGG-Nets for obtaining feature representations from the facial maps; the outputs of the VGG-Nets were then used for training several SVMs. Finally, [114] performed score fusion based on the trained SVMs for FER. We note that despite the elaborate framework proposed in [114], our less complex pipeline based on modality fusion at latent representation level significantly outperforms their reported result; see Table 10.2.

Furthermore, we observe that [83] reported a test accuracy of 92% based on the combi-

Table 10.2: Our best experimental result via depth map and RGB fusion along with state-of-the-art results for comparison

Approach	Test acc. (%)
3D geometric shape model+LDA [244]	83.60
Bayesian Belief net+statistical facial features [266]	82.30
Distance+slopes+SVM [215]	87.10
2D+3D features fusion+SVM [113]	86.32
Geometric scattering representation+SVM [242]	84.80
Geometric+photometric attributes+VGG19 [114]	84.87
Depth map+RGB: PL-fusion-ResNet50 (ours)	**87.08**
Depth map+RGB: PL-fusion-VGG19 (ours)	**89.31**

nation of 2D and 3D features trained on DCNNs from scratch. However, [83] did not follow any of the well known protocols for training and evaluation on the BU-3DFE dataset; that is, using a set of randomly selected 60 subjects that are either fixed or not in a 10-fold cross validation scheme for training and evaluation. Instead, they consider the whole 100 subjects for training and evaluation, using 90 subjects for training and the remaining 10 subjects for testing. In addition, [83] failed to report that they carried out cross validation in their experimental setup; this naturally raises two concerns: (1) a larger dataset has been used for training the proposed model (2) sampling bias for the training and testing data in view of the reported result. However, using the fusion approach that we propose in this work and the well established training and evaluation protocol for the BU-3DFE dataset (i.e. a smaller training data), we reach a competitive test accuracy of 89.31%.

10.6 Conclusion

In this chapter, we apply multimodal learning to the problem of facial expression recognition. Facial expression recognition is an important domain in human-computer interaction. Endowing computers with the ability to analyse the facial expressions of users should improvement their utility; as such, a computer can adapt its response given the facial expression of the user. Motivated by the individual strengths of depth map and RGB data for representation information, we propose a facial expression recognition pipeline that is based on multimodal learning, which allows the extraction of robust features. We posit that learning

jointly from both depth map and RGB modalities would result in learning more discriminative features as against singly learning from either modalities. We construct three different learning pipelines for learning the BU-3DFE dataset; experimental results validate the effectiveness of the fusion approach over learning facial expression features separately from depth map or RGB data.

Chapter 11

Learning to Fuse Latent Representations for Multimodal Data

Multimodal learning leverages data from different modalities to improve the performance of a trained model. As seen in the previous chapter, latent representations extracted from multimodal data are typically provided via direct feature fusion for end-to-end training of a deep neural network towards a specific task. However, the information quality of the different data modalities can easily vary across a collected dataset. As such, directly or *naively* fusing the latent representations obtained for one modality and the other, as is commonly done in state-of-the-art works, may burden the model in finding concise representations that are indeed useful for learning. In this chapter, we propose to instead learn the fusion of latent representations for multimodal data by using a modality gating mechanism that allows the dynamic weighting of extracted latent representations based on their informativness. Extensive experiments using the BU-3DFE dataset for facial expression recognition and the Washington object classification multimodal RGB-D dataset show that learning the fusion of the latent representations for different data modalities leads to improved model generalization than the conventional naive fusion method.

11.1 Introduction

Several computer vision tasks benefit from the powerful representation capacity of deep neural networks (DNNs). The latent representations obtained from trained DNNs typically capture the different factors of variations in the training data [173, 153]. DNN features obtained in the earlier layers of the model are usually low-level features characterizing input data attributes such as edges at different orientations and corners. The features extracted in the later layers are generally more abstract (or high-level) for performing different tasks such as object classification, segmentation, scene understanding, etc. Generally, DNNs are trained using inputs of a single modality. For example, RGB images [100], grayscale images [32], binary images [153], audio signal waveforms [170], etc. However, different works [47, 90, 198, 205] have proposed learning DNNs with multimodal data; such works show that multimodal data can be leveraged to improve the robustness of the latent representations learned towards better model generalization. In particular, the aforementioned works show that DNNs trained using multimodal data often generalize better as compared to similar DNNs trained on only one data modality. In [47], a dataset of RGB and images were collected for object classification. A DNN architecture based on pre-trained AlexNet was constructed for concurrent processing of RGB and depth images. Subsequently, the extracted latent representations obtained from the pre-trained AlexNet were directly fused via direct concatenation for end-to-end training of the whole model. Reported results show that learning from both RGB and depth data modalities yielded better results as opposed to learning from either of the two modalities singly. In another work [90], image and audio data modalities were leveraged for improving the performance of DNN on emotion recognition. Convolutional neural network (CNN) was used for extracting latent representations for image data, while a Deep Belief Network (DBN) was used for learning latent representations for processed audio data based on Mel-frequency cepstral coefficients (MFCCs). Although multimodal learning with naive fusion[1] (NF) works, one problem is that the level of informativeness of the different data modalities are typically ignored for learning. Existing works

[1] Naive fusion and direct fusion are used interchangeably

basically concatenate extracted latent representations of different data modalities and feed them into a stack of DNN layers irrespetive of the fact that one data modality may be far less informative (or corrupted) than the other; this can for instance distort learning. The assumption is that layers following the fusion stage in the DNN can somehow manage to extract descent latent representations for the classification layer. However, we argue that instead of burdening the DNN to cope with the aforementioned problem, employing a DNN that is specifically designed to address this scenario would lead to a more concise learning and model generalization. In extreme cases, given a pair of multimodal inputs for training, one of the modalities may be so noisy as to even *corrupt* learning given the input from the other data modality that truely characterizes the underlying factor of variation in the problem being modeled.

In this work, we propose to learn how to fuse the different features extracted from multi-modal data as opposed to direct fusion. We extend our recent work [158] to multimodal learning. In [158], we proposed a highway block for gating transformed and untransformed features in very deep networks, and as such alleviates the problem of model optimization. First, we clarify that the purpose of using a similar gating mechanism in this work is not for addressing the problem of training very deep networks as in [158, 208]. In contrast, such a gating mechanism with appropriate modifications can be used to learn the fusion of latent representations for improved model generalization as presented in this work. We use RGB and depth modalities to experimentally validate the proposed approach. Specifically, the proposed fusion approach offers the following contributions:

1. Learn appropriate fusion (weighting) of RGB and depth latent representations; this alleviates learning problems that can result from inconsistent informativeness of RBG and depth data over collected datasets.

2. Realize improved model generalization as opposed to naive fusion of latent representations using BU-3D facial expression recognition and Washington object classification RGB-D datasets.

The remainder of this chapter is organized as follows. In Section 11.2, background and

problem statement are presented. Section 11.3 gives the details of the proposed approach for learning the fusion of RBG and depth latent representations. Section 11.4 reports our experimental results. We conclude the chapter in Section 11.5.

11.2 Background and Problem Statement

11.2.1 Background on learning with multimodal data

Unimodal data can often have many training samples which lack fine details or that are even unreliable for capturing reasonable latent representations, due to noise, occlusion, illumination changes, etc. This can negatively impact learning of consistent latent representations, and consequently limit the generalization capacity of a trained model. Hence, learning from multimodal data can improve the consistency and reliability of learned latent representations; that is, where one data modality fails to be sufficiently informative, the other data modality can be leveraged to still render decent latent representations. In view of this, many works extract latent representations (e.g using pre-trained models) from multimodal data and thereafter perform direct fusion via concatenation to carry out end-to-end training [47, 90, 198, 205]. Assume a multimodal training dataset of the form

$$D = \{((\boldsymbol{x}_n^{M_1}, \boldsymbol{x}_n^{M_2}), \boldsymbol{y}_n)\}_{n=1}^{N}, \tag{11.1}$$

where $\boldsymbol{x}_n^{M_1}$ and $\boldsymbol{x}_n^{M_2}$ are the nth input from the data modality M_1 and the data modality M_2, respectively; $\boldsymbol{y}_n$ is the desired output for the nth input data. The main assumption in multomodal learning is that $\boldsymbol{x}_n^{M_1}$ and $\boldsymbol{x}_n^{M_2}$ mostly provide complementary information for capturing the important factors of variations in a specific task such as classification, detection, regression, etc. Otherwise, multimodal learning may not yield any benefit in the case where both data modalities provide information for explaining the same factors of variations. For the rest of this work, factors of variations are denoted using a set F. We use factors of variations to refer to the actual features that explain a specific task. For example, in a facial recognition task, features such as raised eyebrows, unelevated eyebrows, dropped jaw, closed mouth,

slightly opened mouth, widely open mouth, etc. are different factors of variations that can explain facial expressions.

Let us denote the total factors of variations for a specific task S_T by F^T, and consider that they are *altogether* captured in a hypothetical dataset, T. Also, assume that we have a dataset A that partially captures F^T as F^A such that we can write[2] $F^A \subset F^T$; this is the case with unimodal data for learning. However, we can have multimodal data A and data B subject to $A \subset T$ and $B \subset T$ such that we can capture factors of variations F^A and F^B, respectively. Furthermore, if we encapsulate F^A and F^B using a multimodal dataset D as in (1), then we have $F^D = \{F^A, F^B\}$ and thus can write a general relation

$$F^A, F^B \subseteq F^D \subset F^T. \tag{11.2}$$

Note from (2) that mulitmodal learning becomes unuseful when $F^A \cup F^B = F^A$, since we have the relation

$$F^A = F^D \subset F^T. \tag{11.3}$$

Particularly, observe that this can happen even when data A and B are different in the input data space, since F^A and F^B actually depend on extracted latent representations. Therefore latent representation and factors of variations are used interchangeably.

Multimodal learning becomes useful when $F^A \cup F^B \neq F^A$; this is the goal of multimodal learning. Subsequently, we can write (2) as

$$F^A \subset F^D \subset F^T. \tag{11.4}$$

We posit that the level of contribution of modality data B for learning a specific task based on modality data A can be related to $F^B \backslash F^A = \{f \epsilon F^A \mid f \notin F^B\}$; specifically, considering its cardinality as in $\mid F^B \backslash F^A \mid$. Furthermore, we can write a general expression for F^T as

$$F^T = \{F^A \backslash F^B, F^B \backslash F^A, F^R\}, \tag{11.5}$$

[2]Since A is a snaphot of the task domain defined by T

where F^R contains the remaining factors of variations that can be captured using other datasets.

11.2.2 Problem statement

In many real-life scenarios, we will have that $F^A \backslash F^T \neq \emptyset$ and $F^B \backslash F^T \neq \emptyset$, since data $A \subseteq T$ and $B \subseteq T$ subject to $A \cup T \neq A$ and $B \cup T \neq B$, respectively. Therefore, we have that the factors of variations that are actually useful for task T denoted F_u^A and F_u^B are such that $\exists$ $F_u^A \subset F^A$ and $F_u^B \subset F^B$; u is used to index a set of useful $f \in F$. As such, we can have the following cases:

- Where $\mid F_u^A \mid \approx \mid F_u^B \mid$, and naive fusion looks interesting, since subsequent layers in the model can *almost equally* rely on F_u^A and F_u^B for model generalization.
- Where $\mid F_u^A \mid \gg \mid F_u^B \mid$ or $\mid F_u^B \mid \gg \mid F_u^A \mid$, and performing a naive fusion may *burden* subsequently layers with focusing on what is more important, since the succeeding layers in the model are trying to learn F_u^A and F_u^B with *roughly equal importance* for generalization.

The second scenario is not uncommon in practice, and as such is addressed in this work. The following section discusses the proposed approach taken to tackle the highlighted problem.

11.3 Learning to Fuse RGB-Depth Latent representations

For alleviating the problem mentioned in Section 11.2.2, we propose to realize a fusion scheme that is *data driven (i.e. dynamic)* and allows the model to learn by itself the importance (or weighting) of the different extracted latent representations, F^A and F^B. The goal is that the proposed *modality gate* filters F^A and F^B, and thereby passes mostly F_u^A and F_u^B to the features concatenation stage that completes the fusion phase as shown in Figure 11.1; fully connected layers and a softmax layer can be trained on top of the fusion outcome. Namely, our approach in this work for constructing the modality gate is motivated by [158,

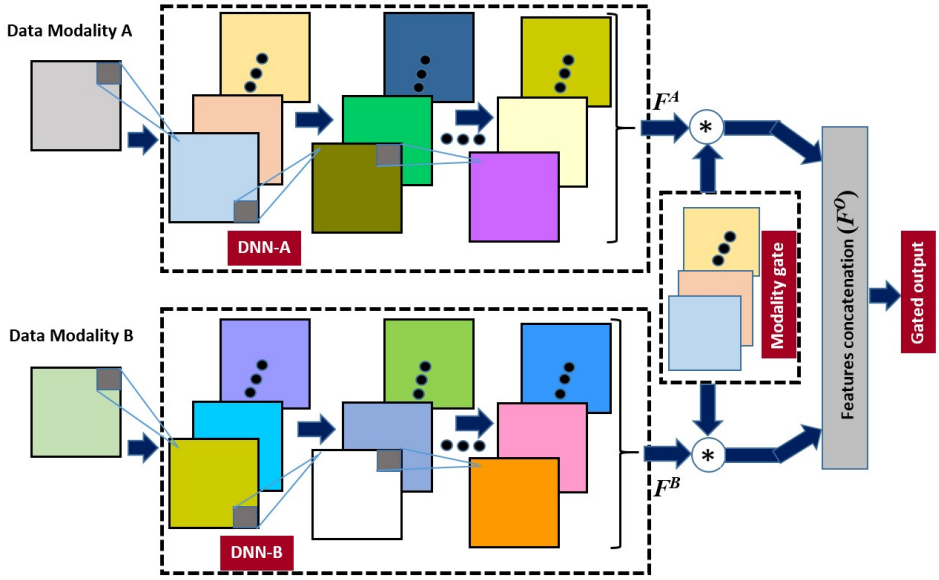


Figure 11.1: Proposed data dependent fusion of latent representations, F^A and F^B extracted using DNN-A and DNN-B respectively. A modality gate is used for filtering F^A and F^B to realize improved fusion

208]; that is, using a mechanism for learning what part of the latent representations are routed to subsequent layers in the model. Again, it is emphasized that the purpose of using a gating mechanism in this work is not to alleviate training problems of very deep network as in [158, 208]. Specifically, this work uses a similar form of gating mechanism proposed in [158] which has improved learning characteristics over that in [208] for learning fusion of latent representations for multimodal data.

In Figure 11.1, we associate with F^A, F^B, F^O the hidden representations $\boldsymbol{H}(\boldsymbol{x})^A$, $\boldsymbol{H}(\boldsymbol{x})^B$ and $\boldsymbol{H}(\boldsymbol{x})^O$, respectively. Subsequently, the modality gate used in this work and shown in Figure 11.1 is of the form

$$\boldsymbol{G}(\boldsymbol{x}) = \varphi(\boldsymbol{W}\boldsymbol{H}(\boldsymbol{x})^B + \boldsymbol{\theta}), \tag{11.6}$$

where $\boldsymbol{G}(\boldsymbol{x})$ denotes gate transformation on $\boldsymbol{H}(\boldsymbol{x})^B$ using weight $\boldsymbol{W}$, bias $\boldsymbol{\theta}$ and Log-sigmoid activation function φ. $\boldsymbol{G}(\boldsymbol{x})$ learns the part of $\boldsymbol{H}(\boldsymbol{x})^B$ that is captured for fusion. Subsequently, $1 - \boldsymbol{G}(\boldsymbol{x})$ is used for selecting the part of $\boldsymbol{H}(\boldsymbol{x})^B$ that is captured for fusion. The overall formulation of the modality gate given in Figure 11.1 can thus lead to the fusion

outcome $\boldsymbol{H}(\boldsymbol{x})^O$ as follows

$$\boldsymbol{H}(\boldsymbol{x})^O = \{\boldsymbol{H}(\boldsymbol{x})^A(1 - \boldsymbol{G}(\boldsymbol{x})) \odot \boldsymbol{H}(\boldsymbol{x})^B\boldsymbol{G}(\boldsymbol{x})\}, \tag{11.7}$$

where $\odot$ is the concatenation operation. Other notations remain as earlier stated. Following the fusion stage, additional weight layers can be stacked in the model and trained end-to-end using $\boldsymbol{H}(\boldsymbol{x})^O$ as input.

11.4 Experiments

In this section, experimental results to validate the usefulness of learning to fuse latent representations as opposed to naive fusion (NF) are reported. Namely, the BU-3DFE facial expression [244] and the Washingtion object RGB-D classification dataset [228] are used. For the constructed *RGB-ResNet50+DepthMap-scratch* model, F^A is extracted from RGB images using pre-trained ResNet50 and F^B is extracted from depth maps using a model trained from scratch[3]. For the proposed *RGB-VGG19+DepthMap-scratch* model, F^A is extracted from RGB images using pre-trained VGG19 and F^B is extracted from depth images using a model trained from scratch. Fusion via modality gating is performed using G as in (7), after which two fully connected and softmax layers are stacked on top of it as discussed in Section 11.3. The whole model is then trained end-to-end using Adaptive Moment (ADAM) mini-batch gradient descent. An initial learning rate of 10^{-3} is used; the learning is annealed to a final value of 10^{-5}. All models are trained for a maximum of 400 epochs.

11.4.1 BU-3DFE facial expression RGB-D dataset

BU-3DFE dataset contains both 3D and 2D data modalities for facial expressions of 100 different subjects. The data processing in [156] is followed for preparing the training data. Also, a similar experimental setting is used for extracting latent expresentations from both data modalities; that is, using pre-trained models (ResNet-50 and VGG19) on ImageNet

[3]Pre-trained model give poor results with depth maps as reported in [156]

Table 11.1: BU-3DFE RGB-D dataset results with 10-fold CV

Approach	Test acc. (%)
DepthMap-ResNet50 [156]	61.11
DepthMap-VGG19 [156]	28.06
DepthMap-scratch [156]	84.72
RGB-ResNet50 [156]	82.92
RGB-VGG19 [156]	81.25
NF: RGB-ResNet50+DepthMap-scratch	87.08
NF: RGB-VGG19+DepthMap-scratch	89.31
Ours: RGB-ResNet50+DepthMap-scratch	**89.86**
Ours: RGB-VGG19+DepthMap-scratch	**90.69**

Table 11.2: Results comparison on BU-3DFE dataset

Approach	Test acc. (%)
3D geometric shape model+LDA [244]	83.60
Bayesian Belief net+statistical facial features [266]	82.30
Distance+slopes+SVM [215]	87.10
2D+3D features fusion+SVM [113]	86.32
Geometric scattering representation+SVM [242]	84.80
Geometric+photometric attributes+VGG19 [114]	84.87
NF:RGB-ResNet50+DepthMap-scratch [156]	87.08
NF: RGB-VGG19+DepthMap-scratch [156]	89.31
Ours: RGB-ResNet50+DepthMap-scratch	**89.86**
Ours: RGB-VGG19+DepthMap-scratch	**90.69**

dataset for RGB data and training a DNN from scratch on depth data. Furthermore, 10-fold cross-validation (CV) is employed for evaluating the models as in [156, 244, 266, 114]. The results are given in Table 11.1, along with models trained on RGB only, depth map only and via naive fusion as in [156]. It will be seen that the proposed fusion approach gives improved results over naive fusion. Results comparison with state-of-the-art models that further validates our proposal is presented in Table 11.2.

11.4.2 Washington object RGB-D dataset

For Washington object classification dataset, we collect a total of 4500 samples for both 3D and 2D data modalities to *clearly* demonstrate that naive fusion can sometimes even hurt model performance as discussed in Section 11.2.2; the samples span the different 51 categories in the dataset. We perform 10-fold and 5-fold cross-validation (CV) using RGB

Table 11.3: Washington RGB-D dataset results with 10-fold CV

Approach	Test acc. (%)
Depth map-ResNet50	2.39
Depth map-VGG19	2.38
DepthMap-scratch	77.07
RGB-ResNet50	50.08
RGB-VGG19	98.25
NF: RGB-ResNet50+DepthMap-scratch	76.65
NF: RGB-VGG19+DepthMap-scratch	90.56
Ours: RGB-ResNet50+DepthMap-scratch	**99.32**
Ours: RGB-VGG19+DepthMap-scratch	**99.54**

Table 11.4: Washington RGB-D dataset results with 5-fold CV

Approach	Test acc. (%)
Depth map-ResNet50	1.67
Depth map-VGG19	2.38
DepthMap-scratch	73.87
RGB-ResNet50	29.76
RGB-VGG19	96.73
NF: RGB-ResNet50+DepthMap-scratch	61.23
NF: RGB-VGG19+DepthMap-scratch	83.16
Ours: RGB-ResNet50+DepthMap-scratch	**98.93**
Ours: RGB-VGG19+DepthMap-scratch	**99.51**

only, depth only, naive fusion (NF) and the proposed fusion approach. Table 11.3 and Table 11.4 show obtained experimental results for 10-fold and 5-fold CV, respectively. Again, it will be observed that using depth maps with pre-trained models gives poor results; see Tables 11.3 & 11.4. Importantly, it is observed that naive fusion using RGB and depth modalities results in degraded model performance as compared to when either of the two data modalities are used. From Tables 11.3 & 11.4, *DepthMap-scratch* outperforms *NF:RGB-ResNet50+DepthMap-scratch*; and *RGB-VGG19* outperforms *NF:RGB-VGG19+DepthMap-scratch*. Interestingly, it is observed that learning the fusion stage gives improved results as compared to naive fusion. This shows that the proposed fusion approach can learn appropriate weightings for the different latent representations such that not only is performance degradation mitigated, but notably improved.

11.5 Conclusion

Multimodal learning is useful for improving model performance. The latent representations extracted from the different modalities are typically fused via direction concatenation. We argue and show that such fusion is naive and can even hurt model performance. In contrast, we propose in this work to allow the model to learn a data driven fusion stage using a gate mechanism that filters latent representations from multimodal data. Experimental results on two different datasets validate the proposed fusion approach. Improved results over naive fusion and several state-of-the-art models are obtained.

Chapter 12

Conclusions

This chapter discusses the main findings in this thesis, along with future directions. First, the contributions of the thesis in the different chapters are summarized. Subsequently, the various interesting directions identified for additional study are related as future work.

12.1 Summary

This thesis studies the improved training and compression of very deep neural networks (DNNs). In the first part, we focus on improving the training of deep models with several hidden layers. DNNs are typically overparameterized so that models have high propensity to overfit many datasets. Furthermore, DNNs with rectified linear units (ReLUs) can die during training so that model fitting capacity is reduced. We address this problem by taking inspiration from the earlier work, deep network stochastic depth [76], to propose an improved residual network with stochastic input skip connections that randomly applies the input data to the different model layers. This helps to further break the co-adaptation of the hidden units to the continuous activation due to the availability of the clean input data. It is shown to improve the generalization performance of the model in comparison to the ResNet [67]. Subsequently, we tackle the dying ReLUs and the assocaited increased model parameterization by employing maxout units and elastic net penalty that learn the activation function and regularize the model, respectively. Furthermore, we tackle the feature transformation

problem of the highway network [207] by reformulating the gating mechanism in such a way that learning new transformations for improved model generalization is facilitated.
In the second part, we investigate theoretically the training problems of very DNNs without skip connections (i.e. PlainNets) to show that the singularity of hidden representations can plague model optimization. Going forward, it is shown that skip connections help to circumnavigate the singularity problem .
In the third part, we address the compression of very deep models by improving the effective sparsity enforcing penalty terms used for training. The debiased elastic group LASSO (DEGL) is proposed to tackle the feature saturation, random selection due to feature correlation, and entangled feature selection and shrinkage. In a following work, the problem of joint knowledge transfer in knowledge distillation (KD) is addressed using the proposed subspace learning (SL). In addition, we show that subspace learning can be combined with LASSO to increase effectiveness of conventional LASSO.
Finally, the application of multimodal learning based on deep neural network is considered for the problems of facial expression and object recognition. We show how joint the learning and static fusion of latent representations of RGB and depth images yields better results for the classification of facial expressions, as compared to singly using either of RGB or depth images. Subsequently, as opposed to the static fusion of latent representations of multimodal data, an interesting approach that allows learning the fusion of the latent representations of RGB and depth images is proposed. It is shown that learning the fusion of latent repersentations give better results than the static fusion approach.

12.2 Future directions

Following the three main research areas in this thesis, the aspects where the contributions given in the different chapters can be extended are discussed as follows.

12.2.1 Neural architecture search for deep neural networks

In Chapter 3, Chapter 4 and Chapter 5, different successful handcrafted model architectures have been proposed for improving the performance of very DNNs. However, the construction of the models can be somewhat tedious, and require some practical experience. In recent times, the automatic determination of suitable and high performance model architectures via neural architecture search (NAS) [274] is an interesting research direction for automatically engineering model architectures. NAS relies on first defining a search space of possible inter-layer connections and operations, after which a search strategy based on reinforcement learning and performance evaluation are applied in a cyclic manner for updating the current hypothetical method.

12.2.2 Analysing non-linear deep models with skip connections

As with many works [270, 200, 185, 91, 141], the theoretical analysis for the problems of training very deep neural networks in the thesis assumed the linear activation function. Although the obtained theoretical results are shown to agree with experimental observations from non-linear models, it would be interesting as a future work to incorporate non-linear activation function for the theoretical analysis.

12.2.3 Optimal structure for model compression

The architecture of the student model for the subspace learning proposed in the thesis is heuristically determined. This practice follows from earlier works [125, 175, 73]. However, for future work, the estimation of the importance of individual weights or convolution filters can improve the automatic engineering of the student model. For instance, a useful criterion for achieving this is the mutual information, where hidden units and convolution filters are added to the student model until the mutal information between the hidden layers of the student and teacher models ceases to improve.

12.2.4 Fusing Several Multimodal data for Improved Learning

Multimodal data with two sources has been considered in this thesis. However, it is not uncommon to find instances, where there are more than two sources of multimodal data. For example, multimodal data including image data, audio data and natural language text data; or RGB image data, depth map data and thermal image data. In the aforementioned cases where there are several data modalities, it will be interesting to explore novel methods for learning the fusion of the latent representations extracted from the different data sources. This is in constrast to the static fusion approach, where the latent representations of the mulitimodal data are simply concatenated (i.e. directly fused) without considering the information quality of the different data.

Appendices

A1 Proof of Proposition 6.1

Let $\boldsymbol{W} = \{\boldsymbol{w}_i\}_{i=1}^n$, where $\boldsymbol{w}_i$ is the i-th vector in $\boldsymbol{W}$, and its entries are randomly sampled from a continuous uniform or Gaussian distribution. Given $\boldsymbol{w}_i$, the matrix $\boldsymbol{W}$ excluding $\boldsymbol{w}_i$ is denoted $\boldsymbol{W}_{-i}$; thus, the span of $\boldsymbol{W}_i$ is $\boldsymbol{W}_{-i}^{span} = span(\boldsymbol{w}_1, .., \boldsymbol{w}_{n-1})$. As such, validating *Proposition 1* means showing that any given $\boldsymbol{w}_i$ does not lie in the span of $\boldsymbol{W}_{-i}^{span}$. i.e. $\boldsymbol{w}_i$ and $\boldsymbol{W}_{-i}^{span}$ are linearly independent.

To start with, it is trivial to observe that $P(\boldsymbol{w}_1 \neq 0) = 1$, as the Lebesgue measure of any singleton set is zero [19]; hence, $P(w_1 \notin \boldsymbol{W}_{-i}^{span}) = 1$. In addition, for $p \in \{2, ..., n-1\}$, let $\boldsymbol{W}_{-1,p} = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_p\}$ be the set of first p vectors, excluding vector i, such that $\boldsymbol{W}_{-1,p}$ is linearly independent with a probability of 1. Thus, we can conclude with a probablity of 1 that $\boldsymbol{W}_{-1,p}$ spans the p-dimensional subspace of $\mathbb{R}^n$, and therefore also has a Lebesgue measure of zero. Importantly, $\boldsymbol{w_{p+1}}$ resides on $\mathbb{R}^n$, and hence is on the exterior of the subspace with a probability of 1. □

A2 Expanded derivation of (6.15)

Herein, we provide the expanded derivation of (6.15) in the main paper, which is based on Assumption 6.1.

Let the vector basis formed by the eigenvectors of $\boldsymbol{W}^l$ (for $1 \leq l \leq L$) in $\mathbb{R}^n$ be $\boldsymbol{V} = [\boldsymbol{v}_1, ..., \boldsymbol{v}_n]$, where $\boldsymbol{V}$ is a unitary matrix. Furthermore, given matrices $\boldsymbol{W}^a$ and $\boldsymbol{W}^b$ that are

diagonizable, we have

$$\boldsymbol{V}^{-1}\boldsymbol{W}^{a}\boldsymbol{V} = \boldsymbol{D}^{a} = \begin{bmatrix} \lambda_1^a & & \\ & \ddots & \\ & & \lambda_n^a \end{bmatrix}, \text{ and} \tag{12.1}$$

$$\boldsymbol{V}^{-1}\boldsymbol{W}^{b}\boldsymbol{V} = \boldsymbol{D}^{b} = \begin{bmatrix} \lambda_1^b & & \\ & \ddots & \\ & & \lambda_n^b \end{bmatrix}. \tag{12.2}$$

Since $\boldsymbol{V}$ is unitary, we can write the following

$$\boldsymbol{W}^{a} = \boldsymbol{V}\boldsymbol{D}^{a}\boldsymbol{V}^{-1}, \text{ and} \tag{12.3}$$

$$\boldsymbol{W}^{b} = \boldsymbol{V}\boldsymbol{D}^{b}\boldsymbol{V}^{-1}. \tag{12.4}$$

Multiplying (12.3) by (12.4) gives

$$\boldsymbol{W}^{a}\boldsymbol{W}^{b} = \boldsymbol{V}\boldsymbol{D}^{a}\boldsymbol{V}^{-1}\boldsymbol{V}\boldsymbol{D}^{b}\boldsymbol{V}^{-1} = \boldsymbol{V}\boldsymbol{D}^{a}\boldsymbol{D}^{b}\boldsymbol{V}^{-1}. \tag{12.5}$$

For $\boldsymbol{x} = \sum_{i=1}^{n} \alpha_i \boldsymbol{v_i},\ \alpha_i \in \mathbb{R}$; where $\boldsymbol{\alpha} = [\alpha_1, ..., \alpha_n]^T$, we can write

$$\boldsymbol{x} = \boldsymbol{V}\boldsymbol{\alpha}, \tag{12.6}$$

The output of a linear L-layer DNN with weight matrices $\{\boldsymbol{W}^L, \boldsymbol{W}^{L-1} \cdots \boldsymbol{W}^l \cdots \boldsymbol{W}^1\}$ can be written as

$$\boldsymbol{h}(\boldsymbol{x})^L = \prod_{l=1}^{L} \boldsymbol{W}^l \boldsymbol{x} = \boldsymbol{W}^L \boldsymbol{W}^{L-1} \cdots \boldsymbol{W}^l \cdots \boldsymbol{W}^1 \boldsymbol{x}. \tag{12.7}$$

Putting (12.6) in (12.7), and applying (12.5) yields

$$\boldsymbol{h}(\boldsymbol{x})^L = [\boldsymbol{V}\boldsymbol{D}^L\boldsymbol{V}^{-1}\boldsymbol{V}\boldsymbol{D}^{L-1}\boldsymbol{V}^{-1} ... \boldsymbol{V}^1\boldsymbol{D}^1\boldsymbol{V}^{-1}]\boldsymbol{V}\boldsymbol{\alpha}. \tag{12.8}$$

Tidying up (12.8) results in

$$\boldsymbol{h}(\boldsymbol{x})^L = \boldsymbol{V}[\prod_{l=1}^{L} \boldsymbol{D}^l]\boldsymbol{\alpha}, \tag{12.9}$$

which can be expressed as

$$\boldsymbol{h}(\boldsymbol{x})^L = [\boldsymbol{v}_1, ..., \boldsymbol{v}_n] \begin{bmatrix} \prod_{l=1}^{L} \lambda_1^l & & \\ & \ddots & \\ & & \prod_{l=1}^{L} \lambda_n^l \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}, \text{ and} \tag{12.10}$$

$$\boldsymbol{h}(\boldsymbol{x})^L = [(\prod_{l=1}^{L} \lambda_1^l)\boldsymbol{v}_1, ..., (\prod_{l=1}^{L} \lambda_n^l)\boldsymbol{v}_n] \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}. \tag{12.11}$$

The derivation is completed by writing (12.11) as

$$\boldsymbol{h}(\boldsymbol{x})^L = \prod_{l=1}^{L} \lambda_1^l \alpha_1 \boldsymbol{v}_1 + ... + \prod_{l=1}^{L} \lambda_n^l \alpha_n \boldsymbol{v}_n, \tag{12.12}$$

and

$$\boldsymbol{h}(\boldsymbol{x})^L = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i \prod_{l=1}^{L} \lambda_i^l. \ \square \tag{12.13}$$

A3 Proof of Theorem 6.3

First, we can write the output for the ResNeXt as

$$\boldsymbol{h}(\boldsymbol{x})_c^m = \prod_{l=1}^{m} \left(\sum_{k=1}^{s} \boldsymbol{W}_c^{l,k} + \mathbf{I} \right) \boldsymbol{x}. \tag{12.14}$$

Using Lemma 6.1 with $c = 1$ in the main paper, we have $(\boldsymbol{W}^{l,k} + \mathbf{I})\boldsymbol{v}_i = (\lambda_i^{l,k} + 1)\boldsymbol{v}_i$. Using (6.10) and Assumption 6.1 from the main paper, we get

$$\boldsymbol{h}(\boldsymbol{x})_c^m = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i \prod_{l=1}^{m} (\sum_{j=1}^{s} \lambda_i^{l,k} + 1). \tag{12.15}$$

Furthermore, let $p_i^{l,k} = \sum_{j=1}^{s} \lambda_i^{l,k}$, such that (12.15) can now be written as

$$\boldsymbol{h}(\boldsymbol{x})_c^m = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i \prod_{l=1}^{m} (p_i^{l,k} + 1). \tag{12.16}$$

Again, using Axiom 6.1 in Chapter 6 gives

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x})_c^m = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i + \left(\sum_{l=1}^{m} p_1^{l,k} + \cdots + \prod_{l=1}^{m} p_1^{l,k} \right) \Bigg\{ \alpha_1 \boldsymbol{v}_1 + \\ \sum_{i=2}^{n} \alpha_i \boldsymbol{v}_i \left(\frac{\sum_{l=1}^{m} p_i^{l,k} + \cdots + \prod_{l=1}^{m} p_i^{l,k}}{\sum_{l=1}^{m} p_1^{l,k} + \cdots + \prod_{l=1}^{m} p_1^{l,k}} \right) \Bigg\}. \end{aligned} \tag{12.17}$$

In conclusion, when $m \gg 1$, and $\left| p_1^{l,k} \right| > \left| p_i^{l,k} \right| : 2 \leq i \leq n$, as expected for eigenvalues to (12.17), and substituting $p_i^{l,k} = \sum_{j=1}^{m} \lambda_i^{l,k}$ finalizes the proof . □

A4 Empirical Proof of Corollary 7.2

We rely on extensive empirical observations for the proof of Corollary 2. The objective is to show that given the fashion in which the DNN weight at an arbitrary layer l, $\boldsymbol{W}^l$, is initialized, the smallest singular value is less than one; that is, $\exists\ \sigma_{min}(\boldsymbol{W}^l) < 1$ with a probability of one. In fact, we show that there are often several singular values that are less than one. In order to achieve this, several experiments are performed using popular initialization methods such in [68] and random initializations. For exhaustiveness, weight values drawn from uniform and Gaussian distributions using both methods are considered. Furthermore, for completeness, matrices of different sizes are considered for the experiments. Lastly, to improve results reliability, each experiment is repeated 1000 times. The different initialization schemes employed, along with their implementation details are summarized as follows.

1. **Gaussion distribution using He method** : weight values are drawn from a Gaussian distribution using the He method [68]. The weight values (i.e. entries) $w_{ij}^l \in \boldsymbol{W}^l$ are sampled as follows

$$w_{ij}^l \sim \mathcal{N}(\mu = 0,\, std = \sqrt{6/n_{in}^l}), \tag{12.18}$$

where n_{in}^l is the number of units feeding into layer l. See Figure 1 for results.

2. **Uniform distribution using He method**: weight values are drawn from a uniform distribution using the He method [68]. The weight values $w_{ij}^l \in W^l$ are sampled as follows

$$w_{ij}^l \sim U[\sqrt{6/n_{in}^l}, -\sqrt{6/n_{in}^l}], \tag{12.19}$$

where n_{in}^l is the number of units feeding into layer l. See Figure 2 for results.

3. **Uniform distribution**: weight values are drawn from a uniform distribution in the range $-r$ to r, where weight values $w_{ij}^l \in W^l$ are sampled as follows

$$w_{ij}^l \sim U[-r, r]. \tag{12.20}$$

The value $r = 0.05$ is used for the reported experiments; see Figure 3. However, it is observed that similar results are obtained using other reasonable values for r.

4. **Gaussion distribution**: weight values are drawn from a Gaussian distribution, where weight values $w_{ij}^l \in W^l$ are sampled as follows

$$w_{ij}^l \sim \mathcal{N}(\mu = 0,\, std = \beta), \tag{12.21}$$

The reported experiments use standard deviation values, $\beta = 0.01$ and $\beta = 0.05$ with $\mu = 0$ that are typical for neural networks; see Figure 4 and Figure 5. Again, it is noted that similar results are obtained using other reasonable values for β.

In proofing Corollary 2, Figure A1 to Figure A5 show that irrespective of initialization method or matrix dimension, there are several singular values with values, which are less than one. It is crucial to note that for all the experiments, none of the singular values is zero. This observation further strengthens the claim in *Proposition 1* about the non-singularity of random matrices. Furthermore, we report the median of the singular values, which are less than one for each experiment.

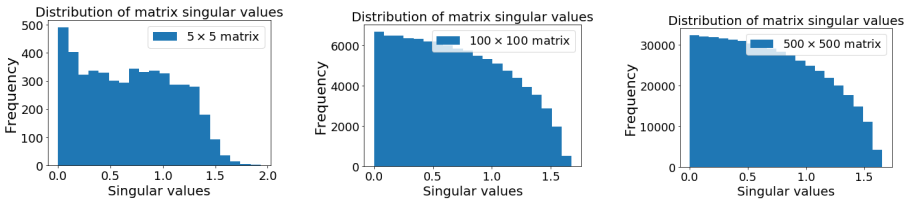


Figure A1: The distribution of singular values of random matrices with entries sampled from uniform distribution obtained using the He method [68]; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.4711. Middle: 0.4728. Right: 0.4734.

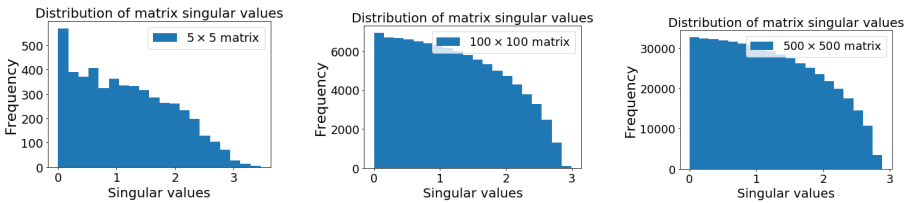


Figure A2: The distribution of singular values of random matrices with entries sampled from Gaussian distribution obtained using the He method [68]; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.4450. Middle: 0.4887. Right: 0.4914.

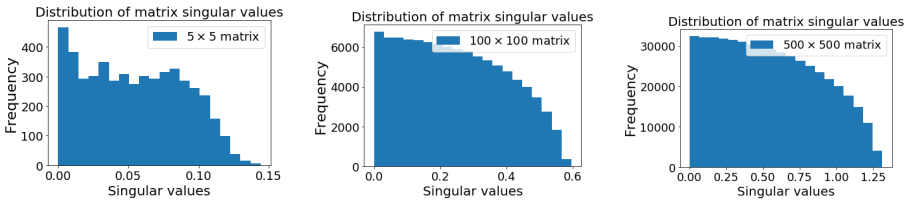


Figure A3: The distribution of singular values of random matrices with entries sampled from uniform distribution in the range -0.05 to 0.05; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.0531. Middle: 0.2336. Right: 0.4537.

A5 Proof of Lemma 1

Some row(s) or column(s) of $\boldsymbol{V}$ are linearly correlated. As such, $\exists$ a vector $\boldsymbol{u} \in \mathbb{R}^n$ with $\| \boldsymbol{u} \|_2 \neq 0$: $\boldsymbol{V}\boldsymbol{u} = 0$. Hence, $\boldsymbol{X}\boldsymbol{V}\boldsymbol{u} = 0$, and thus $\boldsymbol{X}\boldsymbol{V}$ is singular. $\square$

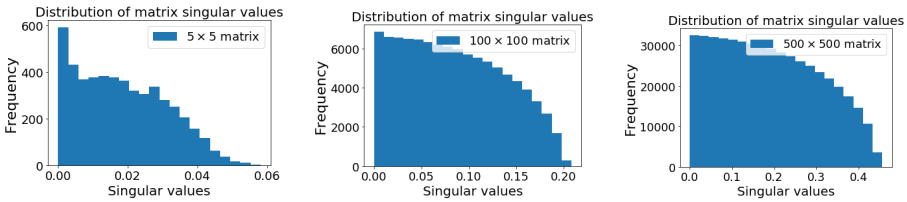


Figure A4: The distribution of singular values of random matrices with entries sampled from Gaussian distribution with $\mu = 0$ and $\beta = 0.01$; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.0173. Middle: 0.0806. Right: 0.1806.

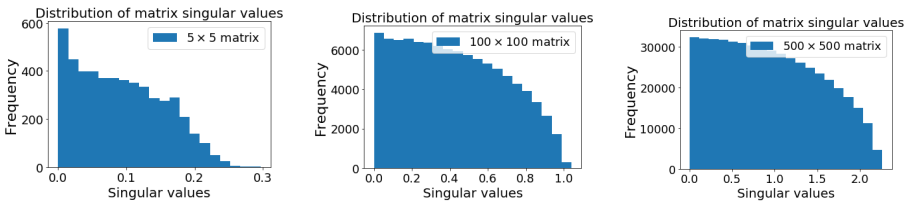


Figure A5: The distribution of singular values of random matrices with entries sampled from Gaussian distribution with $\mu = 0$ and $\beta = 0.05$; each experiment is repeated 1000 times. The median of the singular values that are less than one are as follows. Left: 0.0856. Middle: 0.4027. Right: 0.4863.

A6 Proof of Lemma 2

All row(s) or column(s) of both $\boldsymbol{X}$ and $\boldsymbol{V}$ are linearly uncorrelated. As such, $\nexists$ a vector $u \in \mathbb{R}^n$ with $\| \boldsymbol{u} \|_2 \neq 0$ such that $\boldsymbol{X}\boldsymbol{u} = 0$ or $\boldsymbol{V}\boldsymbol{u} = 0$. Consequently, $\boldsymbol{X}\boldsymbol{V}\boldsymbol{u} \neq 0$, and thus $\boldsymbol{X}\boldsymbol{V}$ is non-singular. $\square$

A7 Proof of Proposition 2

First, let $\boldsymbol{\theta} \in \mathbb{R}^{c \times n}$, $\boldsymbol{X} \in \mathbb{R}^{n \times r}$ and $\boldsymbol{Y} \in \mathbb{R}^{c \times r}$. Now, let us consider the simple problem, $\boldsymbol{Y} = \theta \boldsymbol{X}$: $\boldsymbol{X}^{\dagger}$ is the pseudoinvrese of $\boldsymbol{X}$. The objective is to estimate the solution, $\boldsymbol{\theta}$, given $\boldsymbol{X}$ and $\boldsymbol{Y}$. Furthermore, let a *small* perturbation of $\Delta \boldsymbol{X}$ result in a *small* solution perturbation, $\Delta \boldsymbol{\theta}$, so that

$$\boldsymbol{Y} = (\boldsymbol{\theta} + \Delta \boldsymbol{\theta})(\boldsymbol{X} + \Delta \boldsymbol{X}). \tag{12.22}$$

Observing that $\Delta\boldsymbol{\theta}\Delta\boldsymbol{X} \approx 0$ and $\boldsymbol{Y} = \boldsymbol{\theta}\boldsymbol{X}$, (12.22) becomes

$$\frac{\Delta\boldsymbol{\theta}}{\boldsymbol{\theta}} = -\boldsymbol{X}^{\dagger}\Delta\boldsymbol{X}. \tag{12.23}$$

Employing Cauchy-Schwarz inequality in (12.23) yields

$$\frac{\| \Delta\boldsymbol{\theta} \|}{\| \boldsymbol{\theta} \|} \leq \| \boldsymbol{X}^{\dagger} \| \| \Delta\boldsymbol{X} \| \leq \| \boldsymbol{X}^{\dagger} \| \| \Delta\boldsymbol{X} \| \frac{\| \boldsymbol{X} \|}{\| \boldsymbol{X} \|}. \tag{12.24}$$

In conclusion, taking $\kappa(\boldsymbol{X}) \approx \| \boldsymbol{X}^{\dagger} \| \| \boldsymbol{X} \|$, we obtain

$$\frac{\| \Delta\boldsymbol{\theta} \|}{\| \boldsymbol{\theta} \|} \leq \kappa(\boldsymbol{X}) \frac{\| \Delta\boldsymbol{X} \|}{\| \boldsymbol{X} \|}. \square \tag{12.25}$$

References

[1] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. “Structured pruning of deep convolutional neural networks”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13.3 (2017), p. 32.

[2] Sanjeev Arora et al. “A convergence analysis of gradient descent for deep linear neural networks”. In: *International Conference on Learning Representations*. 2019.

[3] Devansh Arpit et al. “A closer look at memorization in deep networks”. In: *arXiv preprint arXiv:1706.05394* (2017).

[4] Fazael Ayatollahi, Abolghasem A Raie, and Farshid Hajati. “Expression-invariant face recognition using depth and intensity dual-tree complex wavelet transform features”. In: *Journal of Electronic Imaging* 24.2 (2015), pp. 023031–023031.

[5] Jimmy Ba and Rich Caruana. “Do deep nets really need to be deep?” In: *Advances in neural information processing systems*. 2014, pp. 2654–2662.

[6] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. “Segnet: A deep convolutional encoder-decoder architecture for image segmentation”. In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[7] David Balduzzi et al. “The Shattered Gradients Problem: If resnets are the answer, then what is the question?” In: *International Conference on Machine Learning*. 2017, pp. 342–350.

[8] Herbert Bay et al. “Speeded-up robust features (SURF)”. In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[9] Olivia Beaudry et al. “Featural processing in recognition of emotional facial expressions”. In: *Cognition & emotion* 28.3 (2014), pp. 416–432.

[10] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. “Greedy Layerwise Learning Can Scale To ImageNet”. In: *International Conference on Machine Learning*. 2019, pp. 583–593.

[11] Yoshua Bengio. “Practical recommendations for gradient-based training of deep architectures”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[12] Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. “Deep learning”. In: *Nature* 521 (2015), pp. 436–444.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[14] Göran Bergqvist and Erik G Larsson. “The higher-order singular value decomposition: Theory and an application [lecture notes]”. In: *IEEE Signal Processing Magazine* 27.3 (2010), pp. 151–154.

[15] Monica Bianchini and Franco Scarselli. “On the complexity of neural network classifiers: A comparison between shallow and deep architectures”. In: *IEEE transactions on neural networks and learning systems* 25.8 (2014), pp. 1553–1565.

[16] Danushka Bollegala. “Dynamic feature scaling for online learning of binary classifiers”. In: *Knowledge-Based Systems* 129 (2017), pp. 97–105.

[17] Thomas E Booth. “Power iteration method for the several largest eigenvalues and eigenfunctions”. In: *Nuclear science and engineering* 154.1 (2006), pp. 48–62.

[18] Alexander de Brebisson and Giovanni Montana. “Deep neural networks for anatomical brain segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2015, pp. 20–28.

[19] James M Briggs and Thomas Schaffter. “Measure and cardinality”. In: *The American Mathematical Monthly* 86.10 (1979), pp. 852–855.

[20] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. “Model compression”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 535–541.

[21] Paulo Vitor de Campos Souza et al. “Data density-based clustering for regularized fuzzy neural networks based on nullneurons and robust activation function”. In: *Soft Computing* 23.23 (2019), pp. 12475–12489.

[22] Tsung-Han Chan et al. “PCANet: A simple deep learning baseline for image classification?” In: *IEEE transactions on image processing* 24.12 (2015), pp. 5017–5032.

[23] Bo Chang et al. “Multi-level residual networks from dynamical systems view”. In: *International Conference on Learning Representations*. 2018.

[24] Bo Chang et al. “Reversible architectures for arbitrarily deep residual neural networks”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[25] Guobin Chen et al. “Learning efficient object detection models with knowledge distillation”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 742–751.

[26] Sheng Chen. “Local regularization assisted orthogonal least squares regression”. In: *Neurocomputing* 69.4-6 (2006), pp. 559–585.

[27] Yunpeng Chen et al. “Dual path networks”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4467–4475.

[28] Hao Cheng et al. “Evaluating capability of deep neural networks for image classification via information plane”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 168–182.

[29] Yu Cheng et al. “An exploration of parameter redundancy in deep networks with circulant projections”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2857–2865.

[30] Yu Cheng et al. “Model compression and acceleration for deep neural networks: The principles, progress, and challenges”. In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 126–136.

[31] Francois Chollet. “Xception: Deep Learning With Depthwise Separable Convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1251–1258.

[32] Dan Ciresan et al. “Deep neural networks segment neuronal membranes in electron microscopy images”. In: *Advances in neural information processing systems*. 2012, pp. 2843–2851.

[33] Dan CireşAn et al. “Multi-column deep neural network for traffic sign classification”. In: *Neural networks* 32 (2012), pp. 333–338.

[34] Djork-Arne Clevert, Thomas Unterthiner, and Sepp Hochreiter. “Fast and accurate deep network learning by exponential linear units (elus)”. In: *International Conference on Learning Representations*. 2016.

[35] Michael Cogswell et al. “Reducing overfitting in deep networks by decorrelating representations”. In: *International Conference on Learning Representations(ICLR)*. 2016.

[36] Marcella Cornia et al. “Meshed-Memory Transformer for Image Captioning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10578–10587.

[37] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. “Binaryconnect: Training deep neural networks with binary weights during propagations”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 3123–3131.

[38] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)*. Vol. 1. IEEE. 2005, pp. 886–893.

[39] Robert B Davies. “Numerical inversion of a characteristic function”. In: *Biometrika* 60.2 (1973), pp. 415–417.

[40] Olivier Delalleau and Yoshua Bengio. “Shallow vs. deep sum-product networks”. In: *Advances in neural information processing systems*. 2011, pp. 666–674.

[41] Girum G Demisse et al. “Pose Encoding for Robust Skeleton-Based Action Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 188–194.

[42] Misha Denil et al. “Predicting parameters in deep learning”. In: *Advances in Neural Information Processing Systems*. 2013, pp. 2148–2156.

[43] Xiaohan Ding et al. “Auto-balanced filter pruning for efficient convolutional neural networks”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[44] Shichuan Du, Yong Tao, and Aleix M Martinez. “Compound facial expressions of emotion”. In: *Proceedings of the National Academy of Sciences* 111.15 (2014), E1454–E1462.

[45] David Duvenaud et al. “Avoiding pathologies in very deep networks”. In: *Artificial Intelligence and Statistics*. 2014, pp. 202–210.

[46] Bradley Efron et al. “Least angle regression”. In: *The Annals of statistics* 32.2 (2004), pp. 407–499.

[47] Andreas Eitel et al. “Multimodal deep learning for robust rgb-d object recognition”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, pp. 681–687.

[48] Ronen Eldan and Ohad Shamir. “The Power of Depth for Feedforward Neural Networks”. In: *Conference on Learning Theory*. 2016, pp. 907–940.

[49] Lars Elden. “Algorithms for the regularization of ill-conditioned least squares problems”. In: *BIT Numerical Mathematics* 17.2 (1977), pp. 134–145.

[50] Dumitru Erhan et al. “The difficulty of training deep architectures and the effect of unsupervised pre-training”. In: *Artificial Intelligence and Statistics*. 2009, pp. 153–160.

[51] Dumitru Erhan et al. “Why does unsupervised pre-training help deep learning?” In: *Journal of Machine Learning Research* 11.Feb (2010), pp. 625–660.

[52] Tharindu Fernando et al. “Going deeper: Autonomous steering with neural memory networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 214–221.

[53] Ken-Ichi Funahashi. “On the approximate realization of continuous mappings by neural networks”. In: *Neural networks* 2.3 (1989), pp. 183–192.

[54] Deepti Ghadiyaram, Du Tran, and Dhruv Mahajan. “Large-scale weakly-supervised pre-training for video action recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12046–12055.

[55] J Gil-Pelaez. “Note on the inversion theorem”. In: *Biometrika* 38.3-4 (1951), pp. 481–482.

[56] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[57] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).

[58] Ian J Goodfellow et al. “Maxout networks”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning-Volume 28*. JMLR. org. 2013, pp. III–1319.

[59] Benjamin Graham. “Fractional max-pooling”. In: *arXiv preprint arXiv:1412.6071* (2014).

[60] Klaus Greff, Rupesh K Srivastava, and Jürgen Schmidhuber. “Highway and residual networks learn unrolled iterative estimation”. In: *International Conference on Learning Representations*. 2017.

[61] Zaiwang Gu et al. “Ce-net: Context encoder network for 2d medical image segmentation”. In: *IEEE transactions on medical imaging* 38.10 (2019), pp. 2281–2292.

[62] Yanming Guo et al. “Deep learning for visual understanding: A review”. In: *Neurocomputing* 187 (2016), pp. 27–48.

[63] Yiwen Guo, Anbang Yao, and Yurong Chen. “Dynamic network surgery for efficient dnns”. In: *Advances In Neural Information Processing Systems*. 2016, pp. 1379–1387.

[64] Dongyoon Han, Jiwhan Kim, and Junmo Kim. “Deep pyramidal residual networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5927–5935.

[65] Song Han et al. “Learning both weights and connections for efficient neural network”. In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[66] SL Happy and Aurobinda Routray. “Automatic facial expression recognition using features of salient facial patches”. In: *IEEE transactions on Affective Computing* 6.1 (2015), pp. 1–12.

[67] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[68] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[69] Kaiming He et al. “Identity mappings in deep residual networks”. In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[70] Yang He et al. “Soft filter pruning for accelerating deep convolutional neural networks”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. AAAI Press. 2018, pp. 2234–2240.

[71] Yihui He, Xiangyu Zhang, and Jian Sun. “Channel pruning for accelerating very deep neural networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1389–1397.

[72] Yihui He et al. “Amc: Automl for model compression and acceleration on mobile devices”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 784–800.

[73] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *Advances in neural information processing systems Workshop*. 2015, pp. 1–9.

[74] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[75] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2 (1991), pp. 251–257.

[76] Gao Huang et al. “Deep networks with stochastic depth”. In: *European conference on computer vision*. Springer. 2016, pp. 646–661.

[77] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017, pp. 2261–2269.

[78] Jian Huang, Patrick Breheny, and Shuangge Ma. “A selective review of group selection in high-dimensional models”. In: *Statistical science: a review journal of the Institute of Mathematical Statistics* 27.4 (2012).

[79] Pan Huang et al. “Classification of Cervical Biopsy Images Based on LASSO and EL-SVM”. In: *IEEE Access* 8 (2020), pp. 24219–24228.

[80] Qiangui Huang et al. “Learning to prune filters in convolutional neural networks”. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 709–718.

[81] J. J. Hull. “A database for handwritten text recognition research”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16.5 (1994), pp. 550–554. ISSN: 0162-8828. DOI: 10.1109/34.291440.

[82] Noureldien Hussein, Efstratios Gavves, and Arnold WM Smeulders. “Timeception for Complex Action Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 254–263.

[83] Xuan-Phung Huynh, Tien-Duc Tran, and Yong-Guk Kim. “Convolutional Neural Network Models for Facial Expression Recognition Using BU-3DFE Database”. In: *Information Science and Applications (ICISA) 2016*. Springer Singapore, 2016, pp. 441–450.

[84] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International Conference on Machine Learning*. 2015, pp. 448–456.

[85] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. “Speeding up convolutional neural networks with low rank expansions”. In: *arXiv preprint arXiv:1405.3866* (2014).

[86] Mahesh Jangid and Sumit Srivastava. “Handwritten devanagari character recognition using layer-wise training of deep convolutional neural networks and adaptive gradient methods”. In: *Journal of Imaging* 4.2 (2018), p. 41.

[87] Vinoj Jayasundara et al. “TextCaps: Handwritten Character Recognition With Very Small Datasets”. In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 254–262.

[88] Arnaud Joly et al. “L1-based compression of random forest models”. In: *20th European Symposium on Artificial Neural Networks*. 2012.

[89] Md Monirul Kabir, Md Monirul Islam, and Kazuyuki Murase. “A new wrapper feature selection approach using neural network”. In: *Neurocomputing* 73.16-18 (2010), pp. 3273–3283.

[90] Samira Ebrahimi Kahou et al. “Emonets: Multimodal deep learning approaches for emotion recognition in video”. In: *Journal on Multimodal User Interfaces* 10.2 (2016), pp. 99–111.

[91] Kenji Kawaguchi. “Deep learning without poor local minima”. In: *Advances in neural information processing systems*. 2016, pp. 586–594.

[92] Daniel Keysers et al. “Experiments with an extended tangent distance”. In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*. Vol. 2. IEEE. 2000, pp. 38–42.

[93] Pooya Khorrami, Thomas Paine, and Thomas Huang. “Do deep neural networks learn facial action units when doing expression recognition?” In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2015, pp. 19–27.

[94] Jinseog Kim, Yuwon Kim, and Yongdai Kim. “A gradient-based optimization algorithm for lasso”. In: *Journal of Computational and Graphical Statistics* 17.4 (2008), pp. 994–1009.

[95] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. “Accurate image super-resolution using very deep convolutional networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1646–1654.

[96] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[97] Günter Klambauer et al. “Self-normalizing neural networks”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 971–980.

[98] Alex Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: *Technical Report*. 2009.

[99] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR-10, CIFAR-100 (Canadian Institute for Advanced Research)*. `http://www.cs.toronto.edu/~kriz/cifar.html`. Last accessed, October. 2019.

[100] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[101] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. “Fractalnet: Ultra-deep neural networks without residuals”. In: *International Conference on Learning Representations (ICLR)*. 2017.

[102] Thomas Laurent and James Brecht. “Deep Linear Networks with Arbitrary Loss: All Local Minima Are Global”. In: *International Conference on Machine Learning*. 2018, pp. 2902–2907.

[103] Quoc V Le et al. “On optimization methods for deep learning”. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. 2011, pp. 265–272.

[104] Y. LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE, 86(11):2278-2324*. 1998.

[105] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[106] Yann LeCun and Corinna Cortes. *MNIST handwritten digit database*. `http://yann.lecun.com/exdb/mnist/`. Last accessed, October. 2019.

[107] Chen-Yu Lee et al. “Deeply-supervised nets”. In: *Artificial intelligence and statistics*. 2015, pp. 562–570.

[108] Hyungtae Lee and Heesung Kwon. “Going deeper with contextual CNN for hyperspectral image classification”. In: *IEEE Transactions on Image Processing* 26.10 (2017), pp. 4843–4855.

[109] Dawei Li, Xiaolong Wang, and Deguang Kong. “Deeprebirth: Accelerating deep neural network execution on mobile devices”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[110] Fengfu Li, Bo Zhang, and Bin Liu. “Ternary weight networks”. In: *arXiv preprint arXiv:1605.04711* (2016).

[111] Hao Li et al. “Pruning filters for efficient convnets”. In: *International Conference on Learning Representations(ICLR)*. 2017.

[112] Hao Li et al. “Visualizing the loss landscape of neural nets”. In: *arXiv preprint arXiv:1712.09913* (2017).

[113] Huibin Li et al. “An efficient multimodal 2D+ 3D feature-based approach to automatic facial expression recognition”. In: *Computer Vision and Image Understanding* 140 (2015), pp. 83–92.

[114] Huibin Li et al. “Deep representation of facial geometric and photometric attributes for automatic 3d facial expression recognition”. In: *arXiv preprint arXiv:1511.03015* (2015).

[115] Sihan Li et al. “Demystifying resnet”. In: *arXiv preprint arXiv:1611.01186* (2016).

[116] Tuanhui Li et al. “Compressing Convolutional Neural Networks via Factorized Convolutional Filters”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3977–3986.

[117] Xiang Li et al. “Understanding the disharmony between dropout and batch normalization by variance shift”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2682–2690.

[118] Ming Liang and Xiaolin Hu. “Recurrent convolutional neural network for object recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3367–3375.

[119] Min Lin, Qiang Chen, and Shuicheng Yan. “Network in network”. In: *arXiv preprint arXiv:1312.4400* (2013).

[120] Baoyuan Liu et al. “Sparse convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 806–814.

[121] Mengyi Liu et al. “Au-inspired deep networks for facial expression feature learning”. In: *Neurocomputing* 159 (2015), pp. 126–136.

[122] Ping Liu et al. “Facial expression recognition via a boosted deep belief network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1805–1812.

[123] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: vol. 60. 2. Springer, 2004, pp. 91–110.

[124] David G Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[125] Liang Lu, Michelle Guo, and Steve Renals. “Knowledge distillation for small-footprint highway networks”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 4820–4824.

[126] Loren Lugosch et al. “Speech Model Pre-Training for End-to-End Spoken Language Understanding”. In: *Proc. Interspeech 2019* (2019), pp. 814–818.

[127] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. “Thinet: A filter level pruning method for deep neural network compression”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5058–5066.

[128] Yuan Luo, Cai-ming Wu, and Yi Zhang. “Facial expression recognition based on fusion feature of PCA and LBP with SVM”. In: *Optik-International Journal for Light and Electron Optics* 124.17 (2013), pp. 2767–2770.

[129] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. Citeseer. 2013.

[130] Julien Mairal et al. “Convolutional kernel networks”. In: *Advances in neural information processing systems*. 2014, pp. 2627–2635.

[131] Anima Majumder, Laxmidhar Behera, and Venkatesh K Subramanian. “Emotion recognition from geometric facial features using self-organizing map”. In: *Pattern Recognition* 47.3 (2014), pp. 1282–1293.

[132] Brais Martinez et al. “Lipreading Using Temporal Convolutional Networks”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 6319–6323.

[133] Jonathan Masci et al. “Stacked convolutional auto-encoders for hierarchical feature extraction”. In: *International conference on artificial neural networks*. Springer. 2011, pp. 52–59.

[134] Nicolai Meinshausen. “Relaxed lasso”. In: *Computational Statistics & Data Analysis* 52.1 (2007), pp. 374–393.

[135] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. “Learning functions: when is deep better than shallow”. In: *arXiv preprint arXiv:1603.00988* (2016).

[136] Ali Mollahosseini, David Chan, and Mohammad H Mahoor. “Going deeper in facial expression recognition using deep neural networks”. In: *2016 IEEE Winter conference on applications of computer vision (WACV)*. IEEE. 2016, pp. 1–10.

[137] S. A. Nene, S. K. Nayar, and H. Murase. “Columbia Object Image Library (COIL-20)”. In: *Technical Report CUCS-005-96*. 1996.

[138] Yuval Netzer et al. “Reading Digits in Natural Images with Unsupervised Feature Learning”. In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. 2011.

[139] Andreas Neubauer. “A new gradient method for ill-posed problems”. In: *Numerical Functional Analysis and Optimization* 39.6 (2018), pp. 737–762.

[140] Andreas Neubauer and Otmar Scherzer. “A convergence rate result for a steepest descent method and a minimal error method for the solution of nonlinear ill-posed problems”. In: *Zeitschrift für Analysis und ihre Anwendungen* 14.2 (1995), pp. 369–377.

[141] Quynh Nguyen and Matthias Hein. “Optimization Landscape and Expressivity of Deep CNNs”. In: *International Conference on Machine Learning*. 2018, pp. 3727–3736.

[142] Maria-Elena Nilsback and Andrew Zisserman. “Automated flower classification over a large number of classes”. In: *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE. 2008, pp. 722–729.

[143] Tohru Nitta. “On the Singularity in Deep Neural Networks”. In: *International Conference on Neural Information Processing*. Springer. 2016, pp. 389–396.

[144] Tong Niu et al. “Developing a deep learning framework with two-stage feature selection for multivariate financial time series forecasting”. In: *Expert Systems with Applications* 148 (2020), p. 113237.

[145] Hao Su Jonathan Krause Sanjeev Satheesh Sean Ma Zhiheng Huang Andrej Karpathy Aditya Khosla Michael Bernstein Alexander C. Berg Olga Russakovsky Jia Deng and Li Fei-Fei. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* (December 2015), pp. 211–252.

[146] Maxime Oquab et al. “Learning and transferring mid-level image representations using convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1717–1724.

[147] Ebenezer Owusu, Yongzhao Zhan, and Qi Rong Mao. “A neural-AdaBoost based facial expression recognition system”. In: *Expert Systems with Applications* 41.7 (2014), pp. 3383–3390.

[148] Oyebade Oyedotun, Djamila Aouada, and Bjorn Ottersten. “Structured compression of deep neural networks with debiased elastic group lasso”. In: *The IEEE Winter Conference on Applications of Computer Vision*. 2020, pp. 2277–2286.

[149] Oyebade K Oyedotun and Djamila Aouada. “Why do Deep Neural Networks with Skip Connections and Concatenated Hidden Representations Work?” In: *International Conference on Neural Information Processing*. 2020.

[150] Oyebade K Oyedotun, Al Ismaeil Kassem, and Djamila Aouada. “Training Very Deep Neural Networks: Rethinking the Role of Skip Connections”. In: *Neurocomputing (Under review)* (2020).

[151] Oyebade K Oyedotun, Al Ismaeil Kassem, and Djamila Aouada. “Why is Everyone Training Very Deep Neural Network with Skip Connections?” In: *IEEE Transactions on Neural Networks and Learning System (Under review)* (2020).

[152] Oyebade K Oyedotun and Adnan Khashman. “Banknote recognition: investigating processing and cognition framework using competitive neural network”. In: *Cognitive neurodynamics* 11.1 (2017), pp. 67–79.

[153] Oyebade K Oyedotun and Adnan Khashman. “Deep learning in vision-based static hand gesture recognition”. In: *Neural Computing and Applications* 28.12 (2017), pp. 3941–3951.

[154] Oyebade K Oyedotun and Adnan Khashman. “Prototype-incorporated emotional neural network”. In: *IEEE transactions on neural networks and learning systems* 29.8 (2018), pp. 3560–3572.

[155] Oyebade K Oyedotun et al. “Deep Network Compression with Teacher Latent Subspace Learning and LASSO”. In: *Applied Intelligence* (2020).

[156] Oyebade K Oyedotun et al. “Facial Expression Recognition via Joint Deep Learning of RGB-Depth Map Latent Representations.” In: *ICCV Workshops*. 2017, pp. 3161–3168.

[157] Oyebade K Oyedotun et al. “Going Deeper with Neural Networks without Skip Connections”. In: *Proceedings of the IEEE Conference on Image Processing*. 2020.

[158] Oyebade K Oyedotun et al. “Highway Network Block With Gates Constraints for Training Very Deep Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1658–1667.

[159] Oyebade K Oyedotun et al. “Improved Highway Block for Training Very Deep Neural Network”. In: *IEEE Access (Under revie)* (2020).

[160] Oyebade K Oyedotun et al. “Improving the Capacity of Very Deep Networks with Maxout Units”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2018.

[161] Oyebade K Oyedotun et al. “Revisiting the Training of Very Deep Neural Networks without Skip Connections”. In: *Proceedings of the IEEE Conference on Pattern Recognition*. 2020.

[162] Oyebade K Oyedotun et al. “Training very deep networks via residual learning with stochastic input shortcut connections”. In: *International Conference on Neural Information Processing*. Springer. 2017, pp. 23–33.

[163] Konstantinos Papadopoulos et al. “Enhanced trajectory-based action recognition using human pose”. In: *Image Processing (ICIP), 2017 IEEE International Conference on*. IEEE. 2017, pp. 1807–1811.

[164] George Philipp, Dawn Song, and Jaime G Carbonell. “Gradients explode-Deep Networks are shallow-ResNet explained”. In: *arXiv preprint arXiv:1712.05577* (2017).

[165] Geoff Pleiss et al. “Memory-efficient implementation of densenets”. In: *arXiv preprint arXiv:1707.06990* (2017).

[166] Tomaso Poggio and Federico Girosi. “Regularization algorithms for learning that are equivalent to multilayer networks”. In: *Science* 247.4945 (1990), pp. 978–982.

[167] Ben Poole et al. “Exponential expressivity in deep neural networks through transient chaos”. In: *Advances in neural information processing systems*. 2016, pp. 3360–3368.

[168] Yongming Rao, Jiwen Lu, and Jie Zhou. “Spherical Fractal Convolutional Neural Networks for Point Cloud Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 452–460.

[169] Mohammad Rastegari et al. “Xnor-net: Imagenet classification using binary convolutional neural networks”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[170] Mirco Ravanelli and Yoshua Bengio. “Speaker Recognition from raw waveform with SincNet”. In: *arXiv preprint arXiv:1808.00158* (2018).

[171] Wenqi Ren et al. “Gated fusion network for single image dehazing”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3253–3261.

[172] JH Rick Chang et al. “One Network to Solve Them All–Solving Linear Inverse Problems Using Deep Projection Models”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5888–5897.

[173] Salah Rifai et al. “Disentangling factors of variation for facial expression recognition”. In: *Computer Vision–ECCV 2012*. Springer, 2012, pp. 808–822.

[174] Michal Rolinek, Dominik Zietlow, and Georg Martius. “Variational autoencoders pursue pca directions (by accident)”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12406–12415.

[175] Adriana Romero et al. “Fitnets: Hints for thin deep nets”. In: *International Conference on Learning Representations (ICLR)*. 2015, pp. 1–13.

[176] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241.

[177] Saharon Rosset, Ji Zhu, and Trevor Hastie. “Boosting as a regularized path to a maximum margin classifier”. In: *Journal of Machine Learning Research* 5.Aug (2004), pp. 941–973.

[178] Edward Rosten and Tom Drummond. “Machine learning for high-speed corner detection”. In: *European conference on computer vision*. Springer. 2006, pp. 430–443.

[179] Mark Rudelson and Roman Vershynin. “Non-asymptotic theory of random matrices: extreme singular values”. In: *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*. World Scientific. 2010, pp. 1576–1602.

[180] Anwar Saeed et al. “Frame-based facial expression recognition using geometrical features”. In: *Advances in Human-Computer Interaction* 2014 (2014), p. 4.

[181] Itay Safran and Ohad Shamir. “Depth-width tradeoffs in approximating natural functions with neural networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2979–2987.

[182] Jorge Sanchez and Florent Perronnin. “High-dimensional signature compression for large-scale image classification”. In: *CVPR 2011*. IEEE. 2011, pp. 1665–1672.

[183] Evangelos Sariyanidi, Hatice Gunes, and Andrea Cavallaro. “Automatic analysis of facial affect: A survey of registration, representation, and recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.6 (2015), pp. 1113–1133.

[184] Neeta Sarode and Shalini Bhatia. “Facial expression recognition”. In: *International Journal on computer science and Engineering* 2.5 (2010), pp. 1552–1557.

[185] Andrew M Saxe, James L McClelland, and Surya Ganguli. “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks”. In: *arXiv preprint arXiv:1312.6120* (2013).

[186] Simone Scardapane et al. “Group sparse regularization for deep neural networks”. In: *Neurocomputing* 241 (2017), pp. 81–89.

[187] Steffen Schneider et al. “wav2vec: Unsupervised Pre-Training for Speech Recognition”. In: *Proc. Interspeech 2019* (2019), pp. 3465–3469.

[188] Bernhard Scholkopf et al. “Prior knowledge in support vector kernels”. In: *Advances in neural information processing systems*. 1998, pp. 640–646.

[189] Falong Shen, Rui Gan, and Gang Zeng. “Weighted residuals for very deep networks”. In: *Systems and Informatics (ICSAI), 2016 3rd International Conference on*. IEEE. 2016, pp. 936–941.

[190] Neil G Shephard. “From characteristic function to distribution function: a simple framework for the theory”. In: *Econometric theory* 7.4 (1991), pp. 519–529.

[191] Patrice Simard, Yann LeCun, and John S Denker. “Efficient pattern recognition using a new transformation distance”. In: *Advances in neural information processing systems*. 1993, pp. 50–58.

[192] Patrice Y Simard et al. “Transformation invariance in pattern recognitiontangent distance and tangent propagation”. In: *Neural networks: tricks of the trade*. Springer, 1998, pp. 239–274.

[193] Marcel Simon, Erik Rodner, and Joachim Denzler. “Imagenet pre-trained models with batch normalization”. In: *arXiv preprint arXiv:1612.01452* (2016).

[194] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *International Conference on Learning Representations*. 2015.

[195] Serhat Simsek et al. “A hybrid data mining approach for identifying the temporal effects of variables associated with breast cancer survival”. In: *Expert Systems with Applications* 139 (2020), p. 112863.

[196] Pravendra Singh et al. “Stability based filter pruning for accelerating deep cnns”. In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 1166–1174.

[197] Ashutosh Singla, Lin Yuan, and Touradj Ebrahimi. “Food/non-food image classification and food categorization using pre-trained googlenet model”. In: *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management*. ACM. 2016, pp. 3–11.

[198] Kihyuk Sohn, Wenling Shang, and Honglak Lee. “Improved multimodal deep learning with variation of information”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 2141–2149.

[199] Jozsef Solymosi. “The sum of nonsingular matrices is often nonsingular”. In: *Linear Algebra and its Applications* 552 (2018), pp. 159–165.

[200] Sho Sonoda and Noboru Murata. “Transport analysis of infinitely deep neural network”. In: *JMLR* 20.1 (2019), pp. 31–82.

[201] Paulo Vitor C Souza et al. “An interpretable machine learning model for human fall detection systems using hybrid intelligent models”. In: *Challenges and Trends in Multimodal Fall Detection for Healthcare*. Springer, 2020, pp. 181–205.

[202] Jost Tobias Springenberg et al. “Striving for simplicity: The all convolutional net”. In: *arXiv preprint arXiv:1412.6806* (2014).

[203] Suraj Srinivas and R Venkatesh Babu. “Data-free Parameter Pruning for Deep Neural Networks”. In: *British Machine Vision Conference (BMVC)*. 2015.

[204] Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. “Training sparse neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 138–145.

[205] Nitish Srivastava and Ruslan R Salakhutdinov. “Multimodal learning with deep boltzmann machines”. In: *Advances in neural information processing systems*. 2012, pp. 2222–2230.

[206] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[207] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. “Training very deep networks”. In: *Advances in neural information processing systems*. 2015, pp. 2377–2385.

[208] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. “Training very deep networks”. In: *Advances in neural information processing systems*. 2015, pp. 2377–2385.

[209] Yanan Sun, Bing Xue, and Mengjie Zhang. “Evolving Deep Convolutional Neural Networks for Image Classification”. In: *arXiv preprint arXiv:1710.10741* (2017).

[210] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[211] Christian Szegedy et al. “Inception-v4, inception-ResNet and the impact of residual connections on learning”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI Press. 2017, pp. 4278–4284.

[212] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[213] Cheng Tai et al. “Convolutional neural networks with low-rank regularization”. In: *arXiv preprint arXiv:1511.06067* (2015).

[214] Hao Tang and Thomas S Huang. “3D facial expression recognition based on automatically selected features”. In: *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW’08. IEEE Computer Society Conference on*. IEEE. 2008, pp. 1–8.

[215] Hao Tang and Thomas S Huang. “3D facial expression recognition based on properties of line segments connecting facial feature points”. In: *Automatic Face & Gesture Recognition, 2008. FG’08. 8th IEEE International Conference on*. IEEE. 2008, pp. 1–6.

[216] Sasha Targ, Diogo Almeida, and Kevin Lyman. “Resnet in resnet: Generalizing residual architectures”. In: *International Conference on Learning Representations(ICLR) Workshop*. 2016.

[217] Yingli Tian, Takeo Kanade, and Jeffrey F Cohn. “Facial expression recognition”. In: *Handbook of face recognition*. Springer, 2011, pp. 487–519.

[218] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[219] Ludovic Trottier, Brahim Chaib-draa, and Philippe Giguere. “Incrementally built dictionary learning for sparse representation”. In: *International Conference on Neural Information Processing*. Springer. 2015, pp. 117–126.

[220] Thomas Unterthiner and Sepp Hochreiter. “Understanding Very Deep Networks via Volume Conservation”. In: *International Conference on Learning Representations Workshop*. 2016.

[221] Andreas Veit, Michael J Wilber, and Serge Belongie. “Residual networks behave like ensembles of relatively shallow networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 550–558.

[222] Pascal Vincent et al. “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion”. In: *Journal of machine learning research* 11.Dec (2010), pp. 3371–3408.

[223] Li Wan et al. “Regularization of neural networks using dropconnect”. In: *International conference on machine learning*. 2013, pp. 1058–1066.

[224] Haohan Wang et al. “Precision lasso: accounting for correlations and linear dependencies in high-dimensional genomic data”. In: *Bioinformatics* (2018).

[225] Jun Wang et al. “3D facial expression recognition based on primitive surface feature distribution”. In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 2. IEEE. 2006, pp. 1399–1406.

[226] Kuan Wang et al. “Haq: Hardware-aware automated quantization with mixed precision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 8612–8620.

[227] Xiaojie Wang et al. “Kdgan: Knowledge distillation with generative adversarial networks”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 775–786.

[228] *Washington Object Classification Dataset*. July 2018. URL: `https://rgbd-dataset.cs.washington.edu`.

[229] Haikun Wei et al. “Dynamics of learning near singularities in layered networks”. In: *Neural computation* 20.3 (2008), pp. 813–843.

[230] Wei Wen et al. “Learning structured sparsity in deep neural networks”. In: *Advances in neural information processing systems*. 2016, pp. 2074–2082.

[231] Jacob Whitehill et al. “The faces of engagement: Automatic recognition of student engagementfrom facial expressions”. In: *IEEE Transactions on Affective Computing* 5.1 (2014), pp. 86–98.

[232] Mingrui Wu, Bernhard Scholkopf, and Gokhan Bakir. “Building sparse large margin classifiers”. In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 996–1003.

[233] Yong Wu et al. “Convolution Neural Network based Transfer Learning for Classification of Flowers”. In: *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*. IEEE. 2018, pp. 562–566.

[234] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. “Wider or deeper: Revisiting the resnet model for visual recognition”. In: *Pattern Recognition* 90 (2019), pp. 119–133.

[235] Lechao Xiao et al. “Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks”. In: *International Conference on Machine Learning*. 2018, pp. 5389–5398.

[236] Saining Xie et al. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[237] Bing Xu, Ruitong Huang, and Mu Li. “Revise Saturated Activation Functions”. In: *International Conference on Learning Representations (ICLR) Workshop*. 2016.

[238] Huan Xu, Constantine Caramanis, and Shie Mannor. “Sparse algorithms are not stable: A no-free-lunch theorem”. In: *IEEE transactions on pattern analysis and machine intelligence* 34.1 (2011), pp. 187–193.

[239] Xiangyu Xu, Yongrui Ma, and Wenxiu Sun. “Towards Real Scene Super-Resolution with Raw Images”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1723–1731.

[240] Jianbo Yang et al. “Deep Convolutional Neural Networks on Multichannel Time Series for Human Activity Recognition.” In: *IJCAI*. 2015, pp. 3995–4001.

[241] Jianchao Yang, Kai Yu, and Thomas Huang. “Supervised translation-invariant sparse coding”. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE. 2010, pp. 3517–3524.

[242] Xudong Yang et al. “Automatic 3d facial expression recognition using geometric scattering representation”. In: *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on*. Vol. 1. IEEE. 2015, pp. 1–6.

[243] Zichao Yang et al. “Deep fried convnets”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1476–1483.

[244] Lijun Yin et al. “A 3D facial expression database for facial behavior research”. In: *Automatic face and gesture recognition, 2006. FGR 2006. 7th international conference on*. IEEE. 2006, pp. 211–216.

[245] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[246] Ruichi Yu et al. “Nisp: Pruning networks using neuron importance score propagation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9194–9203.

[247] Zhiding Yu and Cha Zhang. “Image based static facial expression recognition with multiple deep network learning”. In: *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. ACM. 2015, pp. 435–442.

[248] Zhiding Yu and Cha Zhang. “Image based static facial expression recognition with multiple deep network learning”. In: *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. ACM. 2015, pp. 435–442.

[249] Ming Yuan and Yi Lin. “Model selection and estimation in regression with grouped variables”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.

[250] M Yunus, Asep Saefuddin, and Agus M Soleh. “Characteristics of Group LASSO in Handling High Correlated Data”. In: *Applied Mathematical Sciences* 11.20 (2017), pp. 953–961.

[251] Adam Coates Alessandro Bissacco Bo Wu Andrew Ng Yuval Netzer Tao Wang. *The Street View House Numbers (SVHN) Dataset*. `http://ufldl.stanford.edu/housenumbers/`. Last accessed, Jan. 2019.

[252] Alireza Zaeemzadeh, Nazanin Rahnavard, and Mubarak Shah. "Norm-Preservation: Why Residual Networks Can Become Extremely Deep?" In: *arXiv preprint arXiv:1805.07477* (2018).

[253] Sergey Zagoruyko and Nikos Komodakis. "DiracNets: training very deep neural networks without skip-connections". In: *arXiv preprint arXiv:1706.00388* (2017).

[254] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks". In: *Proceedings of the British Machine Vision Conference (BMVC)*. Ed. by Edwin R. Hancock Richard C. Wilson and William A. P. Smith. BMVA Press, 2016, pp. 87.1–87.12.

[255] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[256] Ke Zhang et al. "Residual networks of residual networks: Multilevel residual networks". In: *IEEE Transactions on Circuits and Systems for Video Technology* 28.6 (2017), pp. 1303–1314.

[257] Wei Zhang et al. "Multimodal learning for facial expression recognition". In: *Pattern Recognition* 48.10 (2015), pp. 3191–3202.

[258] Wenyue Zhang, Yang Li, and Suge Wang. "Learning document representation via topic-enhanced LSTM model". In: *Knowledge-Based Systems* 174 (2019), pp. 194–204.

[259] Xiang Zhang et al. "CrescendoNet: A New Deep Convolutional Neural Network with Ensemble Behavior". In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2018, pp. 311–317.

[260] Xingcheng Zhang et al. "Polynet: A pursuit of structural diversity in very deep networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 718–726.

[261] Huizhen Zhao et al. “A novel softplus linear unit for deep convolutional neural networks”. In: *Applied Intelligence* 48.7 (2018), pp. 1707–1720.

[262] Kaili Zhao et al. “Joint patch and multi-label learning for facial action unit detection”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 2207–2216.

[263] Lu Zhao et al. “Parallel computing method of deep belief networks and its application to traffic flow prediction”. In: *Knowledge-Based Systems* 163 (2019), pp. 972–987.

[264] Peng Zhao and Bin Yu. “On model selection consistency of Lasso”. In: *Journal of Machine learning research* 7.Nov (2006), pp. 2541–2563.

[265] Ritchie Zhao et al. “Improving neural network quantization without retraining using outlier channel splitting”. In: *International Conference on Machine Learning*. 2019, pp. 7543–7552.

[266] Xi Zhao et al. “Automatic 3D facial expression recognition based on a Bayesian Belief Net and a Statistical Facial Feature Model”. In: *Pattern Recognition (ICPR), 2010 20th International Conference on*. IEEE. 2010, pp. 3724–3727.

[267] Jing Zhong et al. “Where to Prune: Using LSTM to Guide End-to-end Pruning.” In: *IJCAI*. 2018, pp. 3205–3211.

[268] Hao Zhou, Jose M Alvarez, and Fatih Porikli. “Less is more: Towards compact cnns”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 662–677.

[269] Yang Zhou, Rong Jin, and Steven Chu-Hong Hoi. “Exclusive lasso for multi-task feature selection”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 988–995.

[270] Yi Zhou and Yingbin Liang. “Critical points of linear neural networks: Analytical forms and landscape properties”. In: *International Conference on Learning Representations*. 2019.

[271] Xiatian Zhu, Shaogang Gong, et al. “Knowledge distillation by on-the-fly native ensemble”. In: *Advances in neural information processing systems*. 2018, pp. 7517–7527.

[272] Zhuangwei Zhuang et al. “Discrimination-aware channel pruning for deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 875–886.

[273] Fatema Tuz Zohra, Md Wasiur Rahman, and Marina Gavrilova. “Occlusion detection and localization from Kinect depth images”. In: *2016 International Conference on Cyberworlds (CW)*. IEEE. 2016, pp. 189–196.

[274] Barret Zoph and Quoc V Le. “Neural architecture search with reinforcement learning”. In: *International Conference on Learning Representations*. 2017.

[275] Hui Zou and Trevor Hastie. “Regularization and variable selection via the elastic net”. In: *Journal of the royal statistical society: series B (statistical methodology)* 67.2 (2005), pp. 301–320.