Software Engineering for Dataset Augmentation using Generative Adversarial Networks

Benjamin Jahić, Nicolas Guelfi and Benoît Ries
Laboratory for Advanced Software Systems
University of Luxembourg
Esch-Belval, Esch-sur-Alzette 4365, Luxembourg
{benjamin.jahic, nicolas.guelfi & benoit.ries}@uni.lu

Abstract—**Software engineers require a large amount of data for building neural network-based software systems. The engineering of these data is often neglected, though, it is a critical and time-consuming activity. In this work, we present a novel software engineering approach for dataset augmentation using neural networks. We propose a rigorous process for generating synthetic data to improve the training of neural networks. Also, we demonstrate our approach to successfully improve the recognition of handwritten digits using conditional generative adversarial networks (cGAN). Finally, we shortly discuss selected important issues of our process, presenting related work and proposing some improvements.**

Keywords-software engineering; development process; dataset engineering; automated data generation; neural network training

I. INTRODUCTION

Software Engineering (SE) [1] [2] is a field in computer science that focuses on all aspects of the development phases of the production of computer programs. Engineers develop these digital systems within business-critical conditions (e.g. financial, time constraints,…) using appropriate theories, methodologies and tools. This becomes even more critical for engineering datasets for neural network based software systems.

There is a rising demand for data to engineer neural network-based software systems. The necessary amount of training data is increasing too. Software engineers will need to engineer these datasets. However, engineering datasets is often not supported by a rigorous process. It is complicated, expensive and time-consuming to build new datasets. Concerning dataset engineering, there are very few methodologies supporting dataset requirement specification, design or production.

Neural networks are trained on data to acquire knowledge in performing certain tasks (e.g. classification,…). Dataset augmentation can be used to engineer improved neural network-based systems [3] and to improve the knowledge after a training phase. It may help to produce better results during the test phase [4]. Moreover, it can also be used to strengthen some knowledge for some task. Engineers can better influence the knowledge of the neural network based on the customer's requirements.

After a deep state of the art research, we have found very few related works. It is not surprising since the majority of research focuses on neural network architecture improvement. We present the main studies we found related to our problem domain in the discussion section.

In order to solve the lack of methodology and to reduce the cost and complexity to produce/obtain datasets, we propose a development process that exploits techniques for the synthesis of data. This approach is useful to support the efficient creation of a dataset based on the requirements coming from a customer. It is also dedicated to the validation and verification of the neural network-based system according to the customer's needs. Finally, it eases the interpretation and analysis of outputs of such a system with the customer.

In Section II, we describe our proposed approach and its constituting activities. Section III presents an illustration of our approach through the description of an experimentation that we conducted on the MNIST [5] case study. Then, in Section IV we discuss three points of particular interest that we wish to put emphasis on.

II. APPROACH

Illustration To have a concrete understanding of our approach, let us consider that we want to engineer a neural network(NN) to recognize handwritten digits. The default process would be to select a neural network architecture, instantiate this architecture, train the neural network with a given dataset, check the results, change the NN parameters and continue until a satisfactory result is found. This is the traditional craftsmanship approach. Our approach does not only formalize the engineering process but also introduces a mean to improve the recognition qualities of neural networks. In our example, the neural network to engineer is specialized in recognition of images representing a handwritten "seven" in a numerical representation (i.e. '7'). A software engineer will build a NN architecture and start collecting images for training and testing purposes. He then evaluates the NN's recognition capacity by processing and grouping the images into

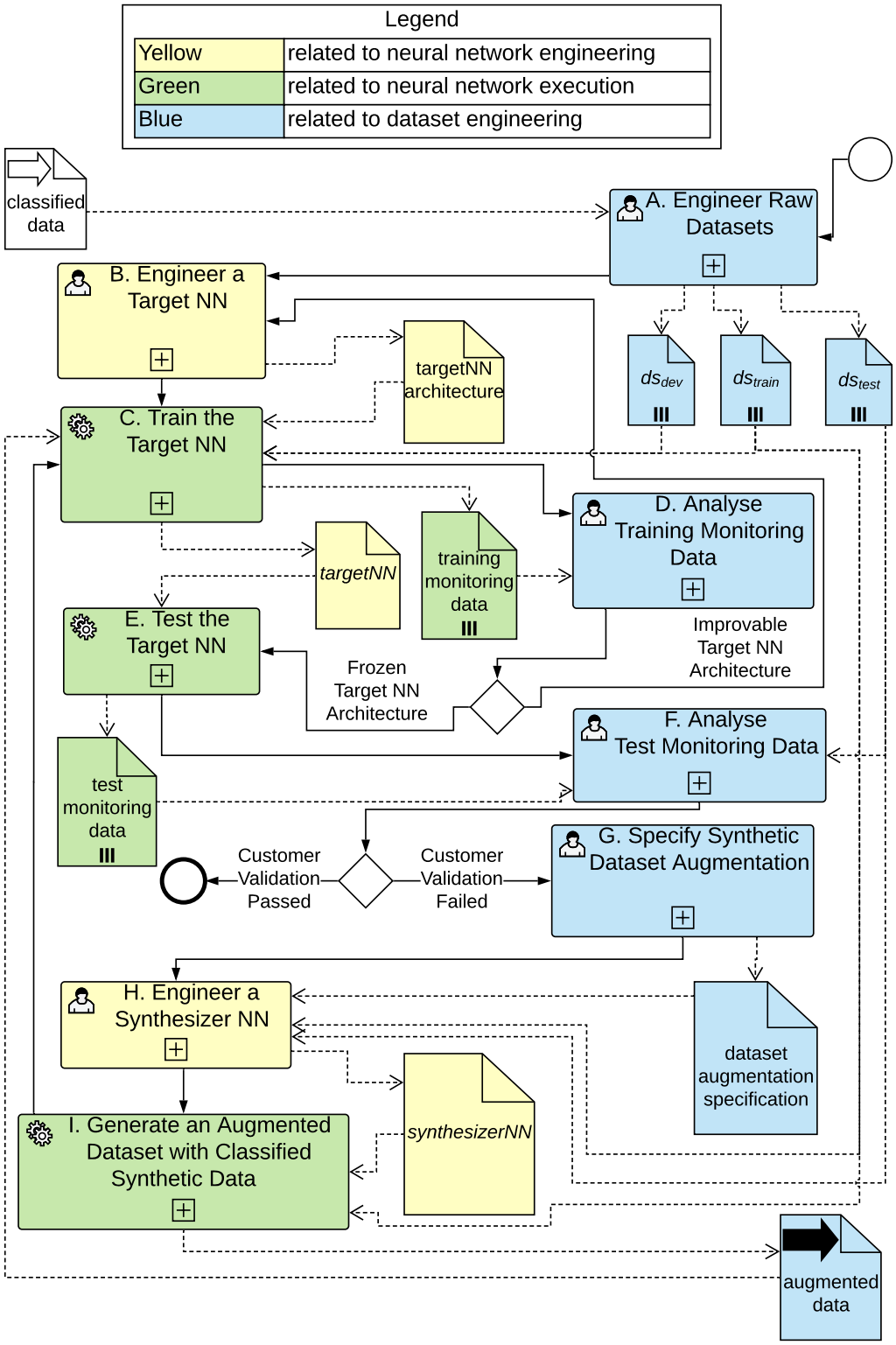

Figure 1. Our Business Process for Dataset Augmentation using a conditional generative adversarial network

different recognition categories (i.e. errors in recognizing 7 written with or without a middle-bar, errors in labels from the input datasets in which some images representing a 'one' are labelled 'seven', etc.); The engineer introduces a synthesizer for automated generation of synthetic training images to reduce the errors for the determined cases (i.e. synthetic images of handwritten 7 without a middle-bar). The engineer then uses the new "hybrid" augmented datasets combining initial and synthetic images to improve the NN's recognition capacity.

In this section, we describe our novel Software Engineering approach for augmenting datasets using neural networks. The proposed approach is formalized using a business process (compliant with the BPMN 2.0 [6] modeling language). Thus, it allows engineers to better apply our process for their deep learning projects. Our process is composed of nine activities presented in the following subsections. Fig. 1 shows an overview of the business process introduced in this paper.

Our process deals with three types of activities, represented by rounded rectangles and different colors, listed below; each of these types of activities requires different expertise:

- *Dataset engineering activities* (in blue). These activities require skills in conceptual modelling, statistical modelling and domain expertise. The stakeholder is typically an analyst who works in close collaboration with the customer for the constitution of datasets needed for IT system engineering (it includes architecture, implementation, ...).
- *Neural network software engineering activities*, (in yellow). These activities aim at designing and implementing the best NN for the targeted recognition problem and are expected to be performed by senior software engineers with a strong specialization in NN development.
- *Neural network execution activities* (in green). These activities aim at using the computing infrastructure to train and test NNs, compute the synthetic dataset. They require knowledge in NN development environments and are typically performed by technicians.

Within the process execution three different types of Data Objects are handled, represented in Fig. 1 by folded corner vertical rectangles of different colors:

- *Datasets* (in blue). Datasets are inputs of NNs using data of the application domain to be used for learning or verifying recognition capacities (e.g. images, text, scenarios, etc.). They are usually designed as a tensor (i.e. a matrix of matrices).
- *Neural Networks* (in yellow). Two NNs are used: on the one hand, the target neural network (i.e. targetNN) is the NN which we aim to improve its quality by augmenting its initial input datasets; on the other hand, a synthesizer neural network (i.e. synthesizerNN) used for the generation of data to augment the initial datasets of the targetNN.
- *Neural Networks Data* (in green). These data encompass several kinds of monitoring data resulting from the execution of neural networks: e.g. accuracy, loss, evolution of learning accuracy over time,...

Two data objects are coming in and out of our process, they are represented by folded corner vertical rectangles with an arrow in Fig. 1:

- *The Classified Data* (arrow in white), in the BPMN modeling language, is the external input of the process. In our approach, the Data Input is a collection of classified data.
- *The Augmented Data* (arrow in black), in the BPMN modeling language, is the external output of the process. In our approach, the Data Output is the augmented dataset resulting from the overall process execution.

The activities are not all described with the same level of detail. We focus on those directly related to dataset augmentation engineering.

A. Engineering Raw Datasets

The first activity (A in Fig. 1) takes as input the process' Data Input, i.e. a collection of classified data. This activity is a subprocess composed of the 4 following sequential tasks :

- *Create the set of equivalence classes*. The first task to engineer raw datasets is to define a number of equivalence classes based on how the Data Input is classified. We suggest to create an equivalence class for each class of the classified data. Indeed, depending on the project application of the business process, the mapping may be defined differently. In our approach, we denote the set of equivalence classes ec_{data}.
- *Engineer the training dataset*. This dataset is composed of a selection of input data and is used to feed the neural network with data to be learned from. We denote this dataset ds_{train}.
- *Engineer the development dataset.* This dataset is composed of an optimal number of data used to observe the evolution of the learning accuracy of the neural network during its training process, see activity C in Fig. 1. We denote this dataset ds_{dev}.
- *Engineer the test dataset.* This dataset is composed of data used to validate the NN after it has been trained. This dataset should ideally be specified together with the customer, as it is used as a final condition to exit our process and deliver the augmented dataset for the targetNN. We denote this dataset ds_{test}.

B. Engineer a targetNN

Based on the datasets defined in Activity A. This second activity consists in building the architecture of the targetNN and constructing an implementation respecting the built architecture. The neural network engineer performs the architecture design and implementation based on properties of the previously engineered datasets. It includes, among others: format of the datasets, size of the different datasets, number of equivalence classes, etc.

TABLE I. OUR FOUR CATEGORIES OF TEST DATA ANALYSIS

		Recognition	
		Correct	***Incorrect***
Classification	***Correct***	Class ∧ Reco	Class ∧ !Reco
	Incorrect	!Class ∧ Reco	!Class ∧ !Reco

C. Train the targetNN

After having engineered the targetNN, the neural network shall be trained. This training activity consists of executing the targetNN architecture implementation with the two input datasets: ds_{train} and ds_{dev}. During the training phase, the neurons' weights are adjusted, resulting in the learning of the targetNN neural network. This is an iterative activity, each iteration is called an epoch. When all the epochs have been executed, the targetNN is considered to be trained on ds_{train}.

D. Analyse Training Monitoring Data

During this activity, the software engineer analyses the various computations resulting from the previous training activity. We propose the three following kinds of data to be analyzed:

- The *training data recognition accuracy values for each equivalence class*, computed by the software engineer. A threshold for the overall accuracy value shall be defined under which it is considered to be improvable.
- The *evolution of the accuracy and loss at the end of each epoch*, computed by the neural network on the training and development dataset. The performance of the neural network (maximal accuracy, minimal loss) shall be satisfied on both datasets. Decreasing performance on the development dataset permits to detect signs of overfitting.
- The *overall loss value*, computed by the neural network on its outputs. A threshold shall be defined under which the loss value is considered to be sufficient, i.e. the targetNN architecture can be frozen.

When all these conditions are satisfied the targetNN's architecture is frozen, otherwise, its architecture should be improved by redoing Activity B.

E. Test the targetNN

Testing the targetNN involves executing the targetNN with ds_{test} as input, in order to validate (or not) the recognition capability of the current targetNN.

F. Analyse Test Monitoring Data

During this activity, the engineer analyses the test data to extract relevant information on the recognition capability of the targetNN. We defined four categories that we propose for analyzing the test data, as summarized in Table I: Class ∧ Reco for the correctly classified and recognized data; Class ∧ !Reco for the correctly classified but incorrectly recognized data; !Class ∧ Reco for the incorrectly classified but correctly recognized data; !Class ∧ !Reco for the incorrectly classified and recognized data.

G. Specify Synthetic Dataset Augmentation

In this activity, based on the analysis of the test data and its categorization in the four defined categories, the dataset engineer is in charge of specifying a dataset augmentation. This specification will comprise the objectives to be met by the synthesizerNN in Activity H.

The augmentation of a dataset involves, among other tasks, the generation of new data. In our approach, we introduce the notion of data similarity. Similarity is a function, strongly-dependent on the business context, that evaluates the distance between 2 data; it is measured in percentage. For instance, we may express "data d_1 is 80% similar to another data d_2".

In order to create the dataset augmentation specification, we propose that the dataset (DS) engineer reviews the test data following the tasks below:

- *!Class ∧ Reco* and *!Class ∧ !Reco*: The engineer should parse all these data an keep them in the test dataset for being recognized. The DS engineer shall move these data from their incorrect to their correct equivalence-class. The unrecognizable equivalence-class of data not to be recognized is removed.
- *Class ∧ !Reco*: All these data should be evaluated, if there is a need to refine some of the existing equivalence classes into sub-classes. For instance, in a digit-recognition business context, the {7} equivalence class, may be refined into two sub-classes, one with a bar (european-style), another one without a bar (american-style). A Decision of which equivalence classes are critical must be taken in order to generate additional similar data to strengthen recognition capacity of the targetNN.
- *Class ∧ Reco*: Generate additional synthetic data with synthesizerNN to strengthen the capacity of targetNN for recognizing data in that category.

Finally, the dataset augmentation specification is created, consisting of a set of objectives resulting from the tasks above. Examples of such specification could contain phrases like "Generate 1000 additional data of minimum similarity of 80% for equivalence class ec_1", "Move $d_1,\ldots d_{10}$ to the unrecognizable equivalence class", "Move data $d_{11},\ldots d_{20}$ from ec_2 to ec_3", etc.

H. Engineer a SynthesizerNN

Based on the dataset augmentation specification, a neural network is engineered that shall be able to generate the specified data synthetically.

I. Augment the Training Dataset with Generated Synthetic Data

This task consists of a monitored execution of the synthesizerNN engineered in Activity H, composed of these sub-tasks:

- The synthetic data is generated by synthesizerNN complying with the dataset augmentation specification and are collected in a new dataset ds_{syn}.

- Each data in ds_{syn} is evaluated with the similarity function against a certain threshold, determined based on the context, such that we do not introduce new synthetic data which are too distant from the initial dataset.
- Finally, a new augmented training dataset ds'_{train} is created as the union from the original ds_{train} and ds_{syn}.

J. Engineer an improved targetNN Instance with the Augmented Datasets (Activities B' to F')

Now that the augmented dataset ds'_{train} has been generated, the software engineer performs the Activities B'-F' a second time using ds'_{train} to create a new instance of an improved targetNN. This process being iterative, it may be repeated as many times as necessary until reaching customer validation.

III. Experimenting the Approach

In this section, we present an experimentation that we conducted on our approach described in the previous Section II. This experimentation consists in instantiating our process on the MNIST [5] case study.

A. Engineering Raw Datasets

For this experimentation, the process's Data Input is a collection of classified grayscale images of handwritten digits, called MNIST having the following characteristics:

- Each image is a matrix of size 28×28, where each element representing an image pixel is a value from 0 to 255 representing the grayscale intensity
- Each monochrome image is associated to a label, being an integer value in [0..9] reflecting the handwritten depiction of the digit, visible on the image.
- The MNIST image collection, consists of two sets of classified images, the MNIST training dataset containing 60.000 classified images and MNIST testing dataset containing 10.000 classified images.

Given the Data Input, we performed the four tasks defined in our approach, creating:

- The set of equivalence classes, named ec_{MNIST} hereafter, defined as $ec_{MNIST} = \{'0',...,'9'\}$.
- The training dataset, $ds_{train} \in P([0,255]^{28\times28}) \times P(ec_{data})$, consists of a random selection of 90% of the initial MNIST training dataset of 60.000 images. ds_{train} contains 54.000 images;
- The development dataset, $ds_{dev} \in P([0,255]^{28\times28}) \times P(ec_{data})$, consists of a random selection of 10% of the MNIST training dataset, 6.000 images;
- The testing dataset, $ds_{test} \in P([0,255]^{28\times28}) \times P(ec_{data})$, is the MNIST testing dataset of 10.000 images.

We created the datasets satisfying these properties :

$$ds_{dev} \cap ds_{train} \cap ds_{test} = \emptyset \quad (1)$$

$$ds_{dev} \cap ds_{train} = \emptyset \quad (2)$$

$$ds_{train} \cap ds_{test} = \emptyset \quad (3)$$

$$ds_{dev} \cap ds_{test} = \emptyset \quad (4)$$

B. Engineering a targetNN

TargetNN has been designed as a Convolutional Neural Network (CNN) architecture inspired from Kizhevsyky et al. [4] and implemented in Python [7] using the Keras Library [8]. We selected a CNN architecture, because of the high ranking of CNNs for the MNIST recognition proposed by [9].

Our CNN has 7 layers with randomly initialized weights; 4 convolutional and 2 fully connected layers. The first 2 layers are convolutional layers. The output kernel maps are max pooled with a (2 × 2) pool and a random dropout of 50%. The next 2 layers are additional convolutional layers. Again, the output kernel maps are max pooled with a (2 × 2) pool, a random dropout of 50% and the resulting kernel maps are flattened. The last 2 layers are fully connected layers. 50% of the outputs of the first fully connected layer are randomly dropped out. The last fully connected layer is the output layer of 10 neurons and outputs a probability distribution over the 10 equivalence classes. The loss is calculated with the categorical cross-entropy function.

Krizhevsky et al. [4] proposed a CNN for the classification of 224×224×3 ImageNet images [10] with many more layers and neurons. Our targetNN is much smaller and designed for recognizing MNIST images of size 28×28. We took inspiration from their architecture to design our targetNN. We adjusted the parameters of the architecture (e.g. layers, neurons, input/ output dimension) without changing the activation functions, such that the targetNN recognizes the MNIST images. We used dropouts to prevent overfitting. According to Srivastava et al. [11], dropout prevents overfitting and make a NN more robust.

Owing to our loss function and the development dataset, we were able to determine during the training whether the NN was overfitting. We optimized the architecture's parameters after several training processes.

C. Training the targetNN

We trained our model on 2 GPUs (Nvidia GTX1080Ti). The training execution of our experimentation lasted for ±10 minutes and 50 epochs. During the epochs, we observed the accuracy and the loss to optimize the targetNN's parameters to improve the training results. The following parameters of the architecture have been optimized: the kernel-size on the convolutional layers; the dropout probability for the different layers ; the number of nodes on the first fully connected layer; the total number of fully connected layers after the last convolutional layer and before the last fully connected layer.

By adjusting these parameters, we reduced signs of overfitting by observing the accuracy and loss on the training and development dataset as shown in Fig. 2.

TABLE II. OVERALL ACCURACY AND LOSS OF TARGETNN

ds_{train}		ds_{dev}		ds_{test}	
acc.(%)	***loss***	***acc.(%)***	***loss***	***acc.(%)***	***loss***
99.91	0.0033	99.45	0.0371	99.47	0.0251

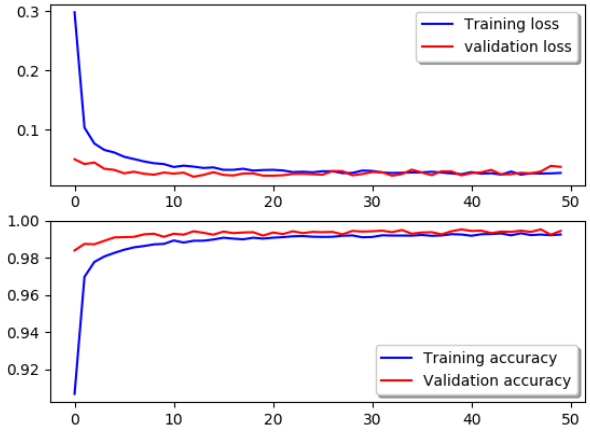

Figure 2. targetNN's accuracy and loss for ds_{train} and ds_{dev}

D. *Analysing the Training Monitoritng Data*

Table II illustrates the targetNN's accuracy and loss on the training, development and testing datasets that represent the training monitoring data. These values are computed with a built-in Keras [8] function that outputs the evaluation of the model. For the three datasets, training, development and testing dataset, we observed a very high accuracy and a low loss.

Thus, it is unlikely that the targetNN is overfitting. Moreover, thanks to the adjustments of the parameters, the targetNN didn't tend to overfit during the training.

Based on these observations, we stopped the targetNN's training and accepted its architecture, by freezing it.

E. *Testing the targetNN*

In this activity, we tested our targetNN with images from the testing dataset. The targetNN takes as input an image and returns probability distribution over the 10 equivalence classes that represent the handwritten digit of the image. Each probability describes the likelihood that the image belongs to an equivalence class. We decided to choose the equivalence class with the maximal probability as selection criteria.

Our selection criteria allows us to compare the recognized class with the expected class of every test image. We collected the incorrectly recognized test images and sorted them by its equivalence class. Fig. 3 shows examples of these images and Table III shows the summary of correctly and incorrectly recognized images for each equivalence class.

F. *Analysing the Test Monitoring Data*

The tested targetNN has an accuracy of 99.47% and a loss of 0.0251 on the testing dataset. Thus, due to the low loss and high accuracy, the targetNN did not tend to overfit and recognized the test images very well .

TABLE III. TESTING DATASET ACCURACY FOR EACH EQUIVALENCE CLASS

Equivalence class	0	1	2	3	4
Accuracy (%)	99.80	99.92	99.52	99.71	99.39
Equivalence class	5	6	7	8	9
Accuracy (%)	99.44	99.17	99.13	99.59	99.01

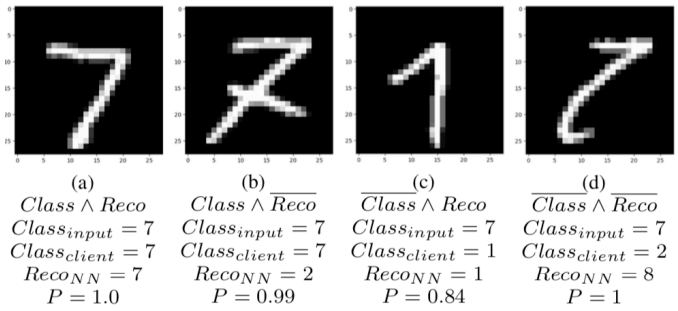

Figure 3. Sample Categorisations for handwritten digit 7

Table III represents the accuracy for each equivalence class. We observe that images for the equivalence classes 7 and 9 are worst recognized. We focus on analyzing the test monitoring data on images of the equivalence class 7.

We categorized the test data according to the four categories defined in our approach, see Section II-F, we categorized the test data. Table IV represents the number of images per category for the equivalence class for digit 7.

Fig. 3 shows some sample image categorizations of the handwritten digit 7. We describe the equivalence class as the input classification $Class_{input}$; the corrected classification as the client's classification $Class_{client}$ and the recognition class of the neural network as the neural network's recognition $Reco_{NN}$. The classification category (*Class* or *!Class*) is selected by comparing the input to client's classification. The recognition category (*Reco* or *!Reco*) is selected by comparing the client's classification and neural network's recognition.

G. *Specifying a Synthetic Dataset Augmentation*

We designed our synthetic dataset augmentation based on the results of the test analysis and the categorization. The testing images have been grouped in 4 categories. We observed that the NN incorrectly recognized 9 images of the equivalence class 7. This group of images contains 1019 correctly classified and recognized images; 4 *Class* $\wedge$ *Reco* images ; 1 *Class* $\wedge$ *!Reco; 1 !Class* $\wedge$ *Reco* image and 4 *!Class* $\wedge$ *!Reco* images

Firstly, we analyzed the *Class* $\wedge$ *Reco* and *Class* $\wedge$ *!Reco* images. We would like to strengthen and improve the recognition of these images. The *Class* $\wedge$ *!Reco* images can be grouped into two equally-sized groups of seven's with a bar and without a bar in the middle. There is no need for refining the equivalence class 7 because we have an equal number of incorrectly recognized images. Thus, we specified the generation of 5 different sets of 1.000, 2.000, 3.000, 5.000 and 10.000 randomly generated synthetic images of the equivalence class 7. We defined the input parameters of the synthetiserNN as our equivalence classes ec_{data}. Thus, the synthesizerNN should be trained on the training dataset ds_{train}.

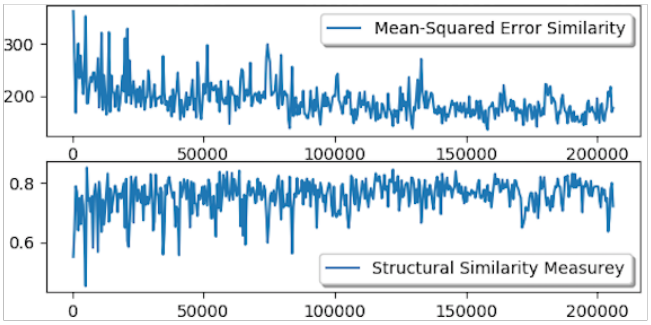

Figure 4. synthetiserNN's similarity evolution in between generated synthetic and training images

We specified a similarity threshold for generating synthetic images at ±80% average similarity to the training images.

Secondly, we analyzed the *!Class* ∧ *Reco* and *!Class* ∧ *!Reco* images. We decided to remove these images from the testing dataset ds_{test}, because we do not want them to be recognized by the targetNN. Thus, we reduced the number of incorrectly recognized images to 4.

H. Engineering a SynthesizerNN

We engineered a conditional Generative Adversarial Network (cGAN) [12] to train a synthetic image generator. Our cGAN is composed of two CNNs: one CNN, called generator, which generates synthetic MNIST images, based on a set of input data; and the other CNN, called the discriminator, which computes the probability of an image being part of the set of input data.

The cGAN generator has been chosen as our synthetiserNN for automating the generation of synthetic images that corresponds to the synthetic dataset augmentation specification resulting from the previous Activity G. We trained the synthetiserNN on 5.000 epochs on our initial training dataset and 10 equivalence classes. Its accuracy is described by the average of the maximal similarities between a generated synthetic image and the training images of the same equivalence class. The similarity is computed with 2 different functions; the mean-squared error (MSE) and the structural similarity measure [13] (SSM). We trained the synthetiserNN by maximizing the SSM and minimizing the MSE error over the number of training iterations.

In Fig. 4, we illustrate the synthetiserNN's accuracy evolution. The synthetiserNN reached an accuracy of ±81% (SSM similarity function) and ±160 (MSE function). We accepted these results to generate the synthetic images.

I. Generating an Augmented Training Dataset with Classified Synthetic Data

In this activity, we ran the synthetiserNN to generate the synthetic images based on the previously defined specification.

In Fig. 5, we show some examples of generated synthetic images. The generated synthetic images are automatically classified and collected in the synthetic dataset ds_{syn}. The new augmented training dataset contains the data of the synthetic dataset ds_{syn} and the raw training dataset ds_{train}.

TABLE IV. ACCURACY AND LOSS FOR TARGETNN

$\lvert ds_{syn} \rvert$	0	1000	2000	3000	5000	10000
accuracy (%)	99.47	99.45	99.42	99.45	99.6	99.47
loss	0.025	0.021	0.024	0.022	0.021	0.025

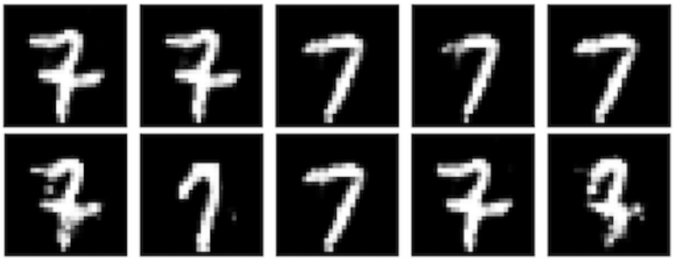

Figure 5. Synthetic Images for Digit 7

J. Engineering of an Improved targetNN Instance with the Augmented Dataset (Activities B' to F')

After re-designing an augmented dataset, we unfroze the targetNN and re-train it on the augmented dataset.

Table V illustrates the overall accuracy and loss of the targetNN after being trained on the augmented datasets. We observe that the overall accuracy is stable. Thus, the targetNN correctly recognizes the same total number of images on average. However, we observe a lower loss for the dataset augmented with 1.000, 2.000, 3.000, 5.000 images, compared to the loss of the targetNN trained on a non-augmented dataset. It seems that a targetNN trained on augmented data generalizes better. In the case of a dataset with 10.000 synthetic images, we achieved equal accuracy and loss for the targetNN compared to the targetNN trained on a non- augmented dataset. Due to the decrease in performance on the testing dataset, it is probable that this targetNN showed first signs of overfitting in that case.

Thus, the retrained targetNN recognizes images of the equivalence class 7 at a higher accuracy of 99.6% compared to the accuracy 99.13% (from Table III). For each dataset augmentation of 1.000, 2.000, 3.000, 5.000 and 10.000 images, we were capable of reducing the unrecognized images from 9 to 5, 8, 5, 4 and 4. We compared the results to the results of the targetNN trained on non-augmented data. We were able to reduce the incorrectly classified images of equivalence class 7 from 9 to 4 (2 *Class* ∧ *!Reco* and 2 *!Class* ∧ *!Reco*).

IV. DISCUSSION

In this section, we have identified two main points of discussion. We present the issue and current limitations as well as related work and possible improvements.

A first point to discuss is that in our process, the dataset augmentation specification is performed informally. It involves manual steps translating from the specification to a synthesizerNN architecture engineering and usage. Related work on that matter is mainly twofold. Firstly, most papers talking about augmentation techniques [14]–[17] do not formally specify the dataset augmentation strategy. In their work, they generate data transformations without a rigorous process and it mostly consists of describing how many more synthetic data are generated. Secondly, other works tackle the usage of AI to provide a dataset augmentation strategy as

Cubuk et al. [18] with their AutoAugment approach computing the probability for the best data augmenta- tion strategy based on a set of input data transformations func- tions. From a software engineering perspective, we would suggest as a possible improvement of our approach, to use model-driven engineering MDE [19]. In particular domain-specific languages (DSL) to rigorously specify the synthetic dataset augmentation and provide a transformation program that generates an implementation of a synthesizerNN architecture.

A second point to discuss is that in our process we introduce a loose notion of a similarity function (e.g. in our experiment we have mean-squared errors, structural similarity, …) to compare synthetic images with the training images, for computing the accuracy of our synthesizerNN. We found some related work on similarity functions for image comparison. One based on the morphological similarity of images, Vizilter and Zhetlov [20] present four techniques for determining similarity and dissimilarity on the morphology of images. They transform images into a set of mosaic shapes and compare different techniques for comparing the shapes. A second one using AI techniques for computing the similarity in between images. Appalaraju and Chaoji [21] present a convolutional neural network for computing the similarity between two images. They present a CNN architecture that takes as input two images and output a distance measure. We would suggest as a possible improvement of our approach to propose multiple similarity functions for computing the accuracy of the synthesizerNN based on the domain problem. Selecting the appropriate similarity function can improve the accuracy of the synthesizerNN. Thus, we could specify more precisely the similarity of the synthetic data for dataset augmentation.

V. CONCLUSION

In this paper, we have presented our approach for improving the quality of neural networks by defining an semi-formal engineering process. The approach has been specified in compliance with the business process modeling notation BPMN 2.0. We have described a concrete experimentation of our approach with the synthetic generation of augmented dataset using conditional Generative Adversarial Network on the MNIST case study for image recognition. The experimentation has shown that our approach is promising, as we managed to improve the accuracy of the initial network by augmenting the MNIST dataset with automatically generated synthetic data.

As a future work, we will work on a formal definition of a DSL for the specification of datasets and dataset augmentation strategies. The DSL grammar should be designed for dataset engineers. Following a model-engineering approach, the specifications written with the DSL would then be used as input to generate automatically an architecture of the synthesizerNN.

Another future work is to perform experimentations with other types of similarity functions for image recognition, e.g. morphological similarity. These functions can be used to improve the accuracy computation of a synthesizerNN and better capture the similarities between synthetic and training images.

REFERENCES

[1] Software Engineering – Guide to the Software Engineering Body of Knowledge (SWEBOK). *International Organization for Standardization*, 2014.ISO-IEC TR 19759-2014, 2014.

[2] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2016.

[3] P.Y. Simard, D. Steinkraus,and J.C. Platt,"Best practices for convolutional neural networks applied to visual document analysis, "In *Proceedings of the 7th International Conference on Document Analysis and Recognition*, ICDAR 2003, Edinburgh, UK, 2003, pp. 958–963.

[4] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet classification with deep convolutional neural networks, "In *Advances in neural information processing systems*, May 2012. pp. 84–90.

[5] L. Deng, "The MNIST database of handwritten digit images for machine learning research. "IEEE Signal Processing Magazine, vol. 29, 2012, p.141-142.

[6] Object Management Group, "*Business Process Modeling Notation (BPMN) v2.0*, "Object Management Group, Full Specification formal/2011-01-03, 2011.

[7] M. Summerfield. Programming in Python 3: A Complete Introduction to the Python Language. Addison-Wesley. 2013.

[8] A. Gulli. and P. Sujit. "Deep Learning with Keras. "Packt Publishing Ltd. 2017.

[9] Y. Lecun, C. Cortes, and C. J.C. Burges, "The MNIST database of handwritten digits. [Online]. Available: http://yann.lecun.com/. [Accessed May. 1,2018].

[10] A. Berg, J. Deng, and L. Fei-Fei. Large scale visual recognition challenge. 2010. [Online]. Available: http://www.image-net.org/. [Accessed May. 1,2018].

[11] N.Srivastava,G.Hinton,A.Krizhevsky,I.Sutskever,andR.Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *The Journal of Machine Learning Research*. 2014. p. 30.

[12] M. Mehdi et S. Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784, 2014.

[13] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity, "*IEEE Transactions on Image Processing*, Apr. 2004. pp. 600–612.

[14] A. Mikolajczyk and M. Grochowski,"Data augmentation for improving deep learning in image classification problem, "In *International Interdisciplinary PhD Workshop,* IIPhDW, Swinoujscie, 2018, pp.117–122.

[15] A. Antoniou, A. Storkey, and H. Edwards, "Data Augmentation Generative Adversarial Networks, "arXiv preprint arXiv:1711.04340, 2017. in press.

[16] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?, "In *International conference on digital image computing: techniques and applications*, DICTA, Sep. 2016. pp. 1-6.

[17] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique, "In *Journal of Artificial Intelligence Research*, vol. 16, Jun. 2002. pp. 321–357.

[18] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan and Q. V. Le, "AutoAugment: Learning Augmentation Policies from Data, arXiv preprint arXiv:1805.09501, 2018. in press.

[19] S. Kent, "Model driven engineering., "In *International Conference on Integrated Formal Methods*, 2002.

[20] Yu. V. Vizilter and S. Yu. Zheltov, "Similarity measures and comparison metrics for image shapes, "In *Journal of Computer and Systems Sciences International*, vol. 53, no. 4, Jul. 2014. pp. 542–555.

[21] S. Appalaraju and V. Chaoji, "Image similarity using Deep CNN and Curriculum Learning, arXiv preprint arXiv:1709.08761, 2017. in press.