Hardware Aware Evolutionary Neural Architecture Search using Representation Similarity Metric

Nilotpal Sinha, Abd El Rahman Shabayek, Anis Kacem, Peyman Rostami,
Carl Shneider, Djamila Aouada
{nilotpal.sinha, abdelrahman.shabayek, anis.kacem,peyman.rostami,
carl.shneider, djamila.aouada}@uni.lu
SnT, University of Luxembourg

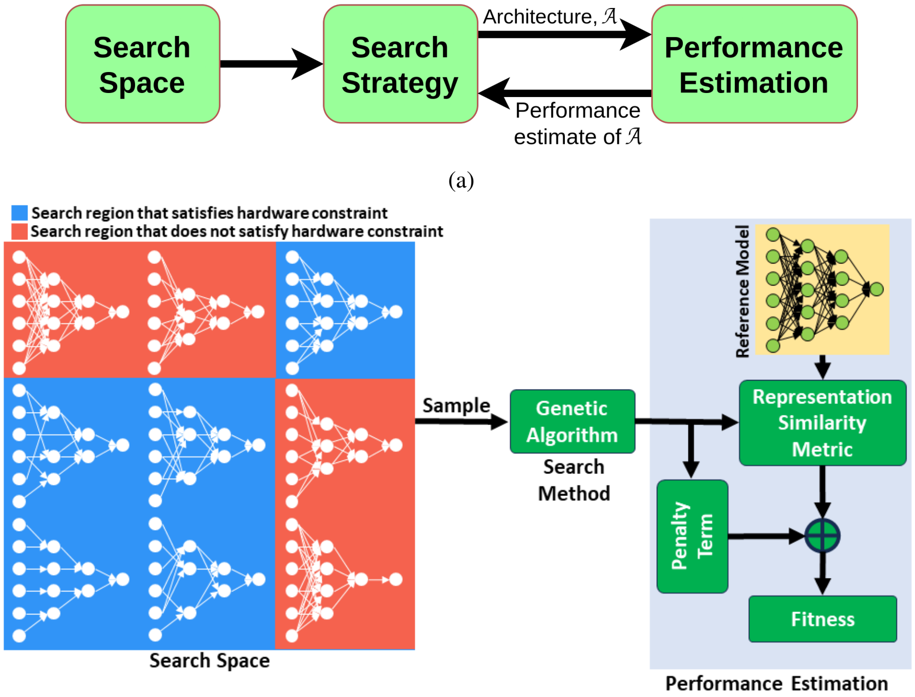
Abstract

Hardware-aware Neural Architecture Search (HW-NAS) is a technique used to automatically design the architecture of a neural network for a specific task and target hardware. However, evaluating the performance of candidate architectures is a key challenge in HW-NAS, as it requires significant computational resources. To address this challenge, we propose an efficient hardware-aware evolution-based NAS approach called HW-EvRSNAS. Our approach re-frames the neural architecture search problem as finding an architecture with performance similar to that of a reference model for a target hardware, while adhering to a cost constraint for that hardware. This is achieved through a representation similarity metric known as Representation Mutual Information (RMI) employed as a proxy performance evaluator. It measures the mutual information between the hidden layer representations of a reference model and those of sampled architectures using a single training batch. We also use a penalty term that penalizes the search process in proportion to how far an architecture's hardware cost is from the desired hardware cost threshold. This resulted in a significantly reduced search time compared to the literature that reached up to $8000\times$ *speedups resulting in lower* CO_2 *emissions. The proposed approach is evaluated on two different search spaces while using lower computational resources. Furthermore, our approach is thoroughly examined on six different edge devices under various hardware cost constraints.*

1. Introduction

In recent years, deep learning systems have revolutionized the technology around us across multiple domains such as computer vision [31, 30, 43, 26, 15, 23], natural language processing [8, 40, 11], etc. These advancements were made possible by the abundance of big data, huge growth in computational power, algorithmic improvements, and enhancements in hardware acceleration. The proliferation of low-energy Internet of Things (IoT) and edge devices has accelerated technological progress by generating massive amounts of data. This has led to the growing need to design deep learning systems that can process such huge amounts of data while consuming limited energy [3, 24, 25]. However, manually designing highly performant neural networks is very challenging as different tasks require different architectural designs and optimizations. Also, the availability of a variety of hardware platforms makes it difficult to design an efficient architecture that performs equally well on all hardware. For example, [19] showed that for a classification task, the same architecture has different latency values from different edge devices. This led to the need of transitioning from conventional *Neural Architecture Search* (*NAS*) algorithms to more specialized types of algorithms, called *HardWare-aware Neural Architecture Search* (*HW-NAS*) [2, 3]. While NAS aims to find the architecture with the best performance for a specific task in a given search space [13], HW-NAS [2, 3] aims to find the architectures with the least trade-off between performance and target hardware usage efficiency.

In any NAS algorithm (Figure 1 (a)), the estimation of architecture performance is typically the main bottleneck. This estimation is crucial in guiding the search process towards architectures that perform well [13, 49, 50]. As a result, there is a growing demand for HW-NAS algorithms with low search time. This is important for reducing the environmental impact, as measured by CO_2 emissions. Additionally, [19] highlights that as the accuracy of an architecture increases, so too does its latency. Therefore, our objective is to develop an efficient HW-NAS algorithm that: (1) performs the architecture search with a low search time, and (2) takes a desired hardware cost measure (e.g., latency) as input, to identify the best architecture for the target device with a hardware cost lower than the hardware cost metric


Search Space
Search Strategy
Architecture, $\mathcal{A}$
Performance Estimation
Performance estimate of $\mathcal{A}$
🟦 Search region that satisfies hardware constraint
🟥 Search region that does not satisfy hardware constraint
Search Space
Sample
Genetic Algorithm
Search Method
Reference Model
Representation Similarity Metric
Penalty Term
Fitness
Performance Estimation

(a)

(b)

Figure 1: (a) Abstract illustration of Neural Architecture Search methods. (b) Abstract illustration of the proposed method. The blue region in the search space denotes the architectures that satisfy the hardware cost constraint (no penalty) while the red region denotes the architectures that do not satisfy the hardware cost constraint (penalty) for the target hardware.

constraint. Recent works [38, 5, 4] perform the HW-NAS by trying to find the best architecture for a target hardware. However, their performance evaluation of the sampled architectures requires a lot of computational resources. To address this issue, we propose an efficient evolution-based HW-NAS method called, **HW-EvRSNAS**. It reformulates the architecture search problem to find the architecture with the closest performance to a reference model while satisfying a specific target hardware constraint. Central to this assumption is that similar performing architectures will have similar Deep Neural Network (DNN) layer representations. Recently, [17] discussed the desired properties of representation similarity metrics for DNNs. A suitable metric should enable finding the closest architecture to a reference model in terms of its representation.

In this work, we sample an architecture from the search space using a genetic algorithm as the evolution-based search method due to its strong performance in NAS problems [32, 4]. A DNN layer representation similarity metric is used to compute the score between an architecture and a reference model, see Figure 1 (b). The *penalty term* block in Figure 1 (b) is used to guide the architecture search to the architectures satisfying the hardware constraint. The *fitness* (a proxy architecture performance metric) of the sampled architecture is calculated by adding the similarity score and the penalty term. The objective of the search method is to converge to the architecture with a high fitness value (more details in Section 3). To summarize, our main contributions are as follows:

- The HW-NAS is reformulated to find an architecture with similar performance to a reference model for a target hardware while satisfying a specific hardware cost constraint. This is achieved by employing a DNN layer representation similarity metric and a penalty term.

- The penalty term is designed to guide the search to a sub-space of the whole architecture search space that satisfies a specific hardware constraint. In particular, the term penalizes the search process in proportion to how far the hardware cost is from the given hardware cost constraint. We show the effectiveness of the penalty term over the rejection sampling used in previous methods [4, 48].

- The robustness of the proposed method is demonstrated in two different search spaces for classification tasks and on six different edge devices.

The rest of the paper is organized as follows: Section 2 discusses relevant works. The proposed method is explained in Section 3. Section 4 presents the experiments performed to validate the method, and Section 5 concludes the work.

2. Related Works

Any NAS method, as described in [14], consists of three key components (depicted in Figure 1 (a)): *search space*, *search strategy*, and *performance estimation*. The search space typically defines the types of architectures that can be represented in principle. The search strategy defines how the exploration of this search space is conducted. This process often involves techniques such as reinforcement learning (RL)-based methods [50, 27], evolutionary algorithm (EA)-based methods [28, 32, 33, 34, 35], and gradient-based methods [21, 44].

The performance estimation refers to estimating the performance of a neural architecture for a given task. This aspect is often the bottleneck of any NAS algorithm. In general, it can be categorized in the following ways:

1. *Training from scratch*: In this approach, each architecture is trained from scratch for a certain number of epochs before evaluation. However, this method demands significant computational resources [13, 49, 50].

2. *Supernet methods*: This approach entails training a single, overly-parameterized network (called a supernet) [21, 5] that encompasses all architectures in the search space as its constituent sub-networks. Training the supernet only once yields the training of all the architectures within the search space simultaneously. While this approach reduces the computational cost, it results in degraded architecture search performance as a result of the inaccurate performance estimation by the supernet [1].

3. *Accuracy predictor based methods*: These approaches employ a smaller model, such as an RNN model [9, 20] to predict the accuracy based on the architectural specifications. However, training such a model is not straightforward as it requires collecting the dataset of architecture specifications and their corresponding accuracies on the given task, consequently increasing the computational requirements.

Emerging as a novel direction is the evaluation of architecture performance by assessing the similarity of its learned representation to that of a pre-trained, high performing reference model. In this context, Zheng et al. [48] have showcased appealing results for classification tasks by adopting one of the similarity metrics proposed in [17]. Inspired by these promising findings, we employ the DNN layer representation similarity metric utilized in [48] as a fundamental component of our proposed hardware-aware NAS algorithm.

Hardware-aware NAS algorithms determine the optimum architecture for a target edge device by modifying the performance estimation component in Figure 1 (a). The assessment of an architecture's performance is made according to two aspects: (1) accuracy on the given task and (2) its projected computational cost when deployed on the target hardware. The objective is to have architectures that exhibit proficient performance (e.g. high classification accuracy) with minimal latency during inference. Thus, achieving a favorable hardware cost stands as a pivotal element in any HW-NAS algorithm. In this context, a plethora of hardware cost metrics have been employed [3]. These metrics include: (1) *FLOPs and Model Size*: In this case, the premise is that the number of parameters and FLOPs correlate positively with model execution time quantified in terms of latency [36, 4]. However, recent research [19, 45] has indicated that models with the same FLOPs may indeed exhibit different latencies on different devices. The results obtained in our ablation studies (Section 4.4.3) corroborate these findings. (2) *Latency*: Incorporating actual latency measurements from practical deployment on hardware can enhance the performance of HW-NAS algorithms [38]. However, it is important to note that this enhancement comes at the expense of increased search costs. Consequently, many works in the literature resort to employing prediction models [5, 46], pre-collected look up tables [39], and analytical estimation[46] based techniques. In this work, we have designed a HW-NAS algorithm that is flexible enough to accommodate either FLOPs or latency within the performance estimation block and is agnostic towards the type of the hardware cost metric adopted.

Since there are multiple objectives in HW-NAS, existing methods generally tackle this challenge by pursuing two distinct strategies: [3]: (1) *Multi-Objective Optimization*: Striking a balance between the objective of obtaining the finest accuracy architecture and the goal of minimizing hardware costs involves trade-offs as the two are inherently conflicting. The core challenge lies in identifying pareto-optimal solutions [41, 7]. Notably, Pareto optimal solutions refer to solutions that cannot be enhanced in one objective without sacrificing at least one other objective. For example, enhancing the accuracy of an architecture might involve increasing network parameters, consequently raising the hardware cost. Finally, it is worth noting that this strategy does not grant us control over the desired latency of an architecture for a target device. (2) *Single-Objective Optimization*: In this strategy, hardware cost is regarded as a constraint presented in the form of thresholds to be respected during the search process. A method like [4] employs *rejection sampling* to rule out any architecture that does not satisfy the constraint during the search process. However, rejection sampling suffers from the risk of the halting problem when it rejects all architectures for not satisfying an excessively low hardware cost constraint (discussed in Section 4.4.2). In contrast, we use a penalty term that reduces the performance metric of the architecture with respect to its proximity to the constraint threshold. In a similar work, [38] also employs a penalty term for the hardware aware part, but their approach introduces two extra hyper-parameters in the penalty term which requires additional efforts for finding the optimal values for those hyper-parameters, whereas in our method, no extra hyper-parameters are introduced in the penalty term.

3. Proposed Method

As explained in Figure 1 (b), the architecture search problem is stated as a search method whose objective is the following: (1) Find an architecture in the search space with similar performance with respect to a baseline model called *reference model*, measured in terms of a performance metric score for a given task. (2) The searched architecture needs to satisfy the hardware cost constraints for the target hardware. We use a representation similarity metric called Representation Mutual Information (RMI) [48, 17] as it has shown promising results as an efficient performance estimation method. Formally, given a pre-trained *reference model* α^* with desired performance metric (e.g. classification accuracy for classification task), an architecture search space $\mathcal{A}$, a device with a hardware cost constraint Ω, HW-EvRSNAS performs the architecture search in the given search space such that the discovered architecture has high performance metric score while satisfying the hardware constraint as follows,

$$\max_{\alpha \in \mathcal{A}} \ \phi(\alpha^*, \alpha), \quad s.t. \ \Psi(\alpha) < \Omega \ , \tag{1}$$

where α is an architecture in the search space $\mathcal{A}$, $\Psi(.)$ is the function that measures the hardware cost (e.g. FLOPs, latency etc.), and $\phi(\alpha^*, \alpha)$ is the deep neural network representation similarity metric that measures the representation similarity between α^* and α, which is used as a proxy architecture performance estimator.

Algorithm 1: HW-EvRSNAS

```
Input: Reference model $\alpha*$, Search space $\mathcal{A}$,
       Hardware constraint $\Omega$, Total generations
       $N_{gen}$, Population size $N_{pop}$, training epochs
       $N_{train}$
Output: Searched architecture, $\alpha^{(N_{gen})}$
$g \leftarrow 0$ (Initialize the generation counter);
Initialize the population $\mathcal{P}$ with random
  architectures;
while $g \leq N_{gen}$ do
    for each individual architecture ($\alpha$) in $\mathcal{P}$ do
        Train $\alpha$ using Eq.(3) for $N_{train}$ epochs;
        Evaluate its fitness score using Eq.(4) ;
    end
    $\alpha^{(g)} \leftarrow$ Best architecture in $\mathcal{P}$ ;
    $\mathcal{P} \leftarrow$ Create next generation population using
      crossover and mutation;
    $g \leftarrow g + 1$;
end
```

3.1. Performance Estimator

In order to find the closest architecture with similar representation with respect to the reference model, [48] used one of the proposed representation similarity metrics in [17] called RMI. It measures the mutual information between hidden layer representations of an architecture and the hidden layer representations of the reference model. RMI score is calculated using a single training batch which makes it a faster and efficient alternative to the naive computationally intensive train from scratch strategy. In particular, given a pre-trained reference architecture, α^*, with a specific performance metric for a task (i.e. *reference model*), the RMI score of any architecture, α, sampled from the search space is defined as,

$$\phi(\alpha^*, \alpha) = \sum_{i=1}^{L} \frac{\left\| X^{i*^T} X^{i} \right\|_F^2}{\left\| X^{i*^T} X^{i*} \right\|_F \left\| X^{i^T} X^{i} \right\|_F} \ , \tag{2}$$

where $X^{1*}, X^{2*}, .., X^{L*}$ and $X^1, X^2, .., X^L$ represent the random variables of feature maps in each layer of α^* and α, respectively, and $\|.\|_F$ is the Frobenius norm. In order to use the RMI score as performance estimator for the proposed HW-EvRSNAS, each sampled architecture α is first trained on a single selected batch using the following loss function,

$$\mathcal{L}_{loss} = \beta \, \phi(\alpha^*, \alpha) + (1 - \beta) \, \mathcal{L}_{task} \ , \tag{3}$$

where $\mathcal{L}_{task}$ is the task specific loss term (e.g. classification loss for classification task), $\phi(\alpha^*, \alpha)$ is the RMI loss term, and β is a parameter weighting the contribution of the two loss terms. Once the sampled architecture α is trained, the RMI score defined in Eq.(2) is used as its performance metric.

3.2. Hardware Aware Architecture Search

As shown in Figure 1 (b), HW-EvRSNAS uses Genetic Algorithm (*GA*) [32] for performing the architecture search in the given search space. GA begins with a population of architectures and each architecture, α, in the population is given a fitness value

$$fitness(\alpha^*, \alpha, \Omega) = \phi(\alpha^*, \alpha) + \psi(\alpha, \Omega) \ , \tag{4}$$

where $\psi(\alpha, \Omega)$ is the penalty term which is used for guiding the search method towards regions in the search space that satisfy the hardware constraint. The fitness of an architecture is a combination of performance metric for the given task and a penalty term. The goal of GA is to generate/evolve the next generation population such that the fitness values of the architectures in the population increases (i.e. *maximizing the fitness*). For a given hardware cost constraint, Ω, the penalty term, ψ, for an architecture, α, is defined as follows,

$$\psi(\alpha, \Omega) = \begin{cases} 0, & \Psi(\alpha) \leq \Omega \\ \Omega - \Psi(\alpha), & \Psi(\alpha) > \Omega \end{cases} \tag{5}$$

where $\psi(\alpha, \Omega)$ penalizes the search method by reducing the fitness value of α if it does not satisfy the hardware cost constraint Ω. Note that the penalty function penalizes the search method in proportion the proximity of hardware cost to the given hardware constraint. In other words, the closer a penalized architecture is to the hardware constraint, the lower is the penalty value. For example in Figure 1 (b), the penalty term for sampled architectures that satisfy the hardware constraint (blue region) is 0. Whereas the penalty term value for those that do not satisfy the hardware constraint (red region) is negative. This in turn reduces the fitness value defined in Eq.(4).

The pseudo-code of HW-EvRSNAS is given in Algorithm 1. The method begins with a population of architecture, $\mathcal{P}$, with N_{pop} (population size) number of random architectures sampled from $\mathcal{A}$. HW-EvRSNAS runs for N_{gen} generations. In each generation, each architecture α in the current generation population is evaluated by training α using the loss function in Eq.(3) for N_{train} epochs. Note that Eq.(3) requires only single training batch and hence will be significantly faster as compared to normal training using only L_{task} on all the training batches. After training, the fitness of the architecture α is evaluated. Once the fitness value is computed, HW-EvRSNAS creates the next generation population using crossover and mutation [32] with the goal of maximizing the fitness value. The best architecture, $\alpha^{(N_{gen})}$, after N_{gen} generations is returned as the discovered architecture while satisfying the hardware constraint, Ω. Note that the best architecture refers to the architecture in the population with the highest fitness value for a given generation.

4. Experiments

Details about the experiments such as search space are provided in Section 4.1, implementation details in Section 4.2 and datasets in supplementary. The re-phrasing of the HW-NAS problem allows us to perform the architecture search with less computational resources. To illustrate this, the architecture search is demonstrated on the classification task in order to compare our method to the literature in Section 4.3. The architecture search performance is reported for six different edge devices under different hardware cost settings for those devices in Section 4.3. Finaly, ablation studies to shed light on the effects of various design choices are reported in Section 4.4. The code is available at https://gitlab.uni.lu/cvi2/elite/hw-evrsnas

4.1. Search Space

The effectiveness of the proposed method is demonstrated on two search spaces: (1) **OFA-search-space** [4]: Here, the type of operation (e.g. convolution, max pooling, etc) are fixed. The HW-NAS algorithm aims to find the optimal setting of the model width, depth, and its kernels sizes. FLOPs are used as a hardware cost metric (constraint) to compare HW-EvRSNAS with the existing literature on HW-NAS, see Table 1. (2) **NAS-Bench-201** [12] provides a unified benchmark for almost any up-to-date NAS algorithm by providing the results on CIFAR-10, CIFAR-100 and ImageNet16-120. The objective is to search for the type of the operation (i.e. convolution 3x3, convolution 1x1, max pooling 3x3, skip connect and none) present between two nodes. Note that *none* operation is used for denoting that there is no connection between two nodes. However, the benchmark does not provide the hardware-cost of the architectures in the search space. For this, we use **HW-NAS-Bench** [19] which augments NAS-Bench-201 by providing various hardware-cost of all the architectures in the search space for six edge devices: *NVIDIA Edge GPU Jetson TX2, Raspberry Pi 4, Edge TPU, Pixel 3, ASIC-Eyeriss, and FPGA*.

4.2. Implementation Details

Since both search spaces involve searching over different aspects of the neural architecture, different architecture representations are used for them. For the OFA-search-space, the architecture representation used in [4] is employed and for NAS-Bench-201, we use the architecture representation used in [32]. Architecture search for both search spaces are done using a single NVIDIA RTX A4000 GPU with a population size (N_{pop}) of 20. For the OFA-search-space, the architecture with highest values for model width, depth and kernel sizes is used as a reference model. For NAS-Bench-201, ResNet-20 is used as the reference model following [48]. The RMI score for both search spaces are calculated after training for 100 epochs (N_{train}) using a value of 0.8 for β in Eq.(3). The architecture search is performed for 100 generations (N_{gen}) for both search spaces.

4.3. Results

The results of HW-EvRSNAS on OFA-search-space for different hardware constraints are shown in Table 1 where FLOPs are used as the hardware cost constraint. The table shows that the proposed method is able to find architectures in the Top1% classification accuracy with a low search cost. This cost is the number of GPU hours required to perform the architecture search. The lower the cost, the more efficient the NAS method is. It is considered more environ-

Table 1: Comparison of the proposed method with other NAS methods in OFA-search-space for ImageNet dataset and FLOPs (MACs) as a hardware cost constraint. "*Manual*" and "*Auto*" in "*Method Type*" refer to hand-crafted and NAS methods respectively. "CO_2e" denotes the CO_2 emission which is calculated based on [37]. † indicates fine-tuning to make a fair comparison with our settings.

Model	Method Type	ImageNet Top1(%)	MACs	Search Cost (GPU Hours)	CO_2e (lbs)
Mobilenetv2 [29]	Manual	72.0	300M	0	-
ShuffleNet [47]	Manual	71.5	292M	0	-
ShuffleNetV2 [22]	Manual	72.6	299M	0	-
NASNet-A [50]	Auto	74.0	564M	48,000	13.6k
DARTS [21]	Auto	73.1	595M	96	27.4
MnasNet [38]	Auto	74.0	317M	40,000	113.4k
FBNet-C [39]	Auto	74.9	375M	216	61
ProxylessNAS [5]	Auto	74.6	320M	200	57
SinglePathNAS [16]	Auto	74.7	328M	312	88.1
AutoSlim [42]	Auto	74.2	305M	180	51
OFA† [4]	Auto	74.7	270M	40	11.3
HW-EvRSNAS (Ours)	Auto	74.6	240M	5	1.05
HW-EvRSNAS (Ours)	Auto	75	300M	5	1.05
HW-EvRSNAS (Ours)	Auto	76.8	374M	5	1.05

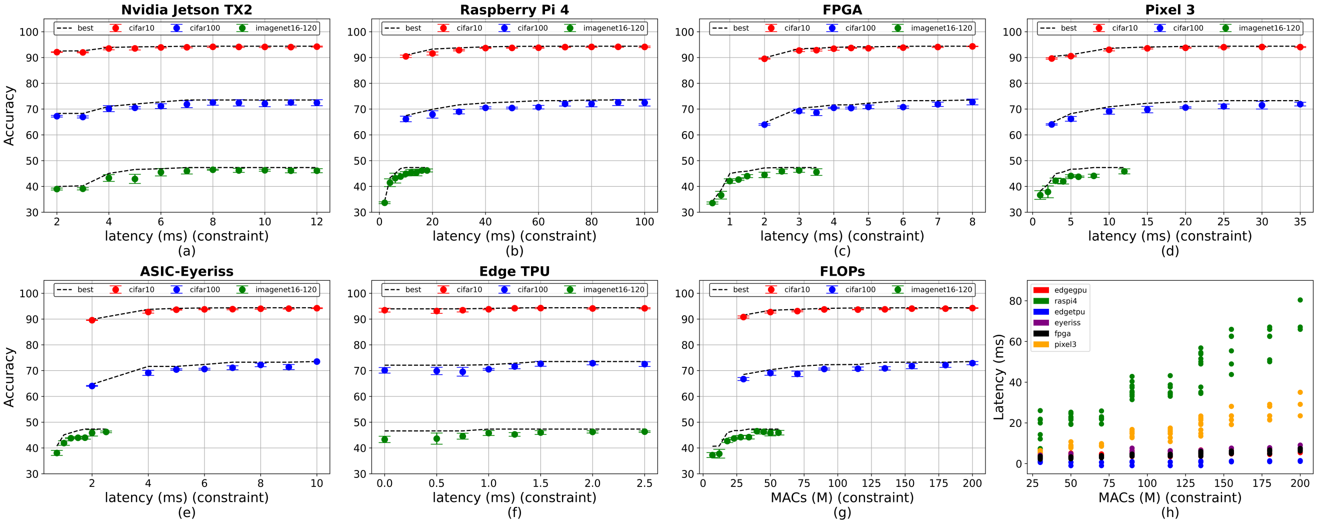


Figure 2: (a)-(g) shows the performance (test accuracy in y-axis) of HW-EvRSNAS with NVIDIA Jetson TX2, Raspberry Pi 4, FPGA, Pixel 3, Edge TPU, ASIC-Eyeriss, TPU latency constraint, flops value constraint respectively in the x-axis. The *dashed* line shows the best architecture under the specific constraint. The different colored dots represents the mean±std accuracy of 10 experiments with different random seeds for the specific constraint. (h) shows the latency of the searched architecture (y-axis) for different edge devices when MACs/FLOPs is used as hardware constraint (x-axis).

mental friendly if it has a lower carbon footprint (measured as CO_2e (lbs). Table 1 shows that HW-EvRSNAS discovers better performing architectures than FBNet [39], ProxylessNAS [5], SinglePathNAS [16], and AutoSlim [42] while having lower FLOPs and using lower search cost (i.e. computational resources) which in turn has a lower carbon footprint. In particular, HW-EvRSNAS performs the HW-NAS in a significantly lower search time: **8000** × **faster** than MnasNet [38], **40** × **faster** than ProxylessNAS [5] and **8** × **faster** than OFA [4]. Note that the reported GPU hours in Table 1 are taken from the respective papers as most of them use different GPU for their experiments.

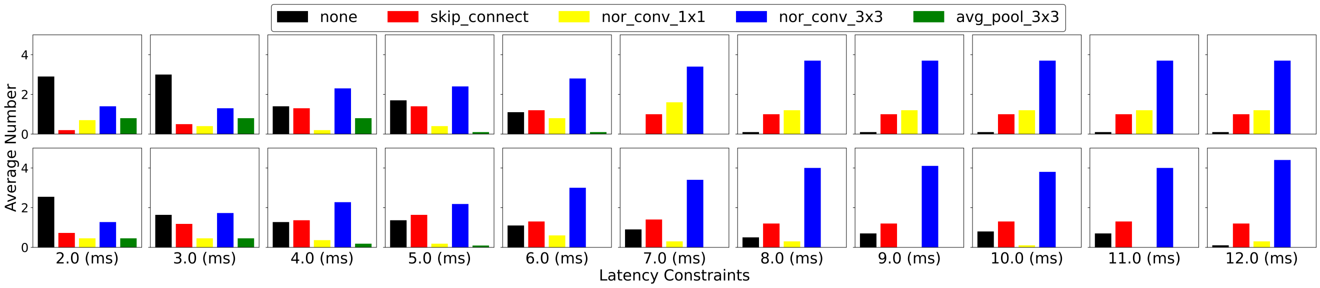


Figure 3: y-axis represents the average number of 5 different types of operations from NAS-Bench-201 search space. Each column of subplot presents the result for different latency constraint. *Top row*: Average number of different types of operations present in the top-10 architectures in terms of test accuracy on CIFAR10 dataset on NVIDIA Jetson TX2. *Bottom row*: Average number of different types of operations present in the 10 architectures discovered from 10 different search runs for a given latency constraint.

The proposed method is agnostic to the search space, utilized device and hardware constraint. Results of HW-EvRSNAS applied on NAS-Bench-201 are reported in Figure 2 (a)-(f) for six different edge devices. Hardware latency is taken as the hardware cost constraint. The architecture search results using FLOPs as hardware constraint are presented in Figure 2 (g). Note that Figure 2 (a)-(g) shows the mean and standard deviation of 10 experiments performed with different random seeds for each hardware constraint. The x-axis represents the different hardware cost constraint values used for architecture search for a specific hardware device. For example in Figure 2 (a), entries for 4 ms (x-axis) represent the architecture search results for the Nvidia Jetson TX2 with 4 ms latency constraint for three different datasets. The figure shows that HW-EvRSNAS is able to find the closest architecture to the optimal one (shown as dashed line) under different latency constraints for the edge devices. It is important to note that the proposed method performance will depend on how accurate the hardware cost metric is, as will be discussed in Section 4.4.

The proposed method is also investigated on its ability to capture the distribution of operations of top-10 performing architectures for a target hardware at different latency constraints. Note that this can be viewed as the sub-space with highest quality architectures under the given latency constraint. Figure S1 (Top) shows that the average number of *none* operation is more for the latency constraint of 2.0s. As the latency increases, the number of *nor_conv_3x3* operation (convolution operation with 3x3 kernel size) increases in the top-10 architectures for the classification task on CIFAR10 on NVIDIA Jetson TX2. This behaviour reflects the harmony between the imposed latency constraint and the retrieved architectures. Figure S1 (Bottom) plots the average number of different operations present in the architectures discovered in 10 independent runs of HW-EvRSNAS for each latency constraint. It shows that HW-EvRSNAS is able to capture architectures with similar operations distribution. In other words, HW-EvRSNAS is able to converge to the sub-space with highest performing architectures given certain constraint. This similarity pattern is observed for other edge devices; Raspberry Pi 4, Edge TPU, Pixel 3, ASIC-Eyeriss, FPGA on all the datasets, see the supplementary material.

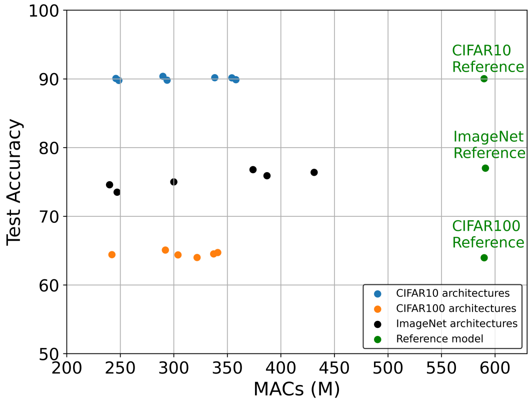


Figure 4: Effect of the use of DNN layer representation similarity metric in HW-NAS (y-axis represents test accuracy) with FLOPs (MACs) as the hardware cost constraint (x-axis).

4.4. Ablation Study

4.4.1 Effect of Rephrasing HW-NAS Problem

The formulation of RMI score, Eq.(2), measures the mutual information between layers of a reference model and architectures in the search space. Hence, a search looking for an architecture with maximum RMI score under a certain constraint finds one whose layers have the maximum mutual information with that reference model. Figure 4 shows the

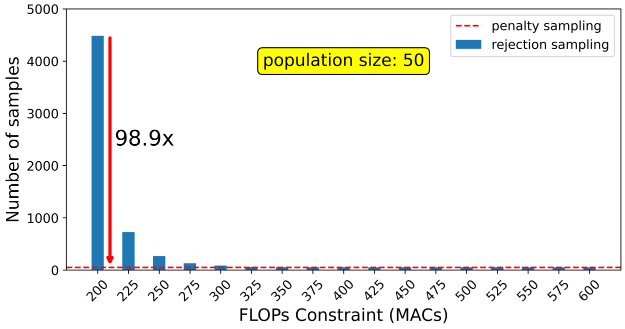


Figure 5: Number of samples (x-axis) required to create a population of size 50 for various hardware constraints (y-axis). The reported number of samples required is the average of 10 runs.

test accuracy of the discovered architectures and the reference model in OFA-search-space for three different datasets (CIFAR10, CIFAR100 and ImageNet). It finds architectures with similar accuracy to the reference model while satisfying the FLOPs constraint. Hence, the rephrased formulation of the HW-NAS problem has successfully retrieved highly constrained and efficient architectures for a given a reference model.

4.4.2 Penalty Term vs Rejection Sampling

In *rejection sampling* [4, 48], the search method rejects the sampled architecture from the search space if it does not satisfy the hardware constraint. This leads to situations where the smaller the sub-space that satisfies that constraint, the more costly the sampling process becomes, see Figure 5. This is due to the high probability of sampling architectures from a bigger sub-space that do not satisfy the hardware constraint. Figure 5 plots the average number of samples required to create a population of architectures with size 50 under different FLOPs constraints in 10 runs. For rejection sampling, Figure 5 shows that as we reduce the hardware cost constraint (in MACs), the number of samples increases exponentially reaching 4500 samples required for the 200 MACs constraint. In contrast, our approach requires a constant number of samples (i.e. 50) for all hardware constraints (a 98.9% reduction in the number of samples for 200 MACs constraint). It does not reject any sampled architecture that does not satisfy the hardware constraint. However, it uses the penalty term, Eq.(5), to reduce the fitness, Eq.(4), of the sampled architecture that does not satisfy the hardware constraint.

4.4.3 Is FLOPs Constraint a Good Indicator for Hardware Cost?

Figure 2 (h) shows architectures found using FLOPs (MACs) as a hardware cost proxy constraint against their respective hardware cost (i.e. latency) for different edge devices on CIFAR10. It is obvious that FLOPs constraint is not a good indicator of hardware cost as the same architecture will have different latencies in different edge devices. Furthermore, architectures with higher FLOPs values have lower latencies on some devices as compared to the latencies on other devices with lower FLOPs. For example, architectures with 200 MACs as FLOPs constraint have lower latency on pixel3 as compared to the latency on raspberry pi 4 (raspi4) of the discovered architectures with 80 MACs as the FLOPs constraint. Note that the reduction in the search cost of HW-EvRSNAS in Table 1 is mainly attributed to the reduction of the performance estimation cost. The search cost of all methods in Table 1 will increase if hardware latency is used as hardware cost. In our method, we primarily focus on the reduction of the performance estimation cost as it is the bottleneck part of any HW-NAS method.

5. Conclusion and Discussion

In this work, we propose an efficient HW-NAS search algorithm that rephrases the NAS problem as a process of finding similar performing architecture with respect to a reference model for a target hardware. This is achieved by utilizing a DNN layer representation similarity metric, RMI score, as a proxy to evaluate architectures performance. A penalty term to penalize the search process is used. It controls the proportion of how far a hardware cost of an architecture is from the given hardware constraint on a target device. The proposed method is evaluated on two different search spaces. It showed a significantly lower search time that resulted into speedups of up to 8000×. This resulted directly into lower usage of computational resources and lower CO_2 emissions consequently. Furthermore, the robustness of the proposed method is demonstrated on finding high performing architectures for the classification task on six different edge devices using two different types of hardware cost metrics (FLOPs and hardware latency). The use of RMI score for estimating the performance of an architecture makes HW-EvRSNAS dependent on the choice of the reference model used for calculating the RMI score (Section 4.4.1). An interesting future direction will be to investigate the effect of choosing different reference models. The effectiveness of HW-EvRSNAS for classification task encourages to extend current work to more downstream tasks such as object detection, image segmentation etc.

6. Acknowledgement

This work is supported by the Luxembourg National Research Fund (FNR), under the project reference C21/IS/15965298/ELITE.

References

[1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR, 2018.

[2] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. Hardware-aware neural architecture search: Survey and taxonomy. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4322–4329. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Survey Track.

[3] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336*, 2021.

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[5] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[6] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[7] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Multi-objective reinforced evolution in mobile neural architecture search. In *European Conference on Computer Vision*, pages 99–113. Springer, 2020.

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

[9] Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017.

[10] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Fei-Fei Li. Imagenet: a large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[13] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[14] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

[15] Albert Garcia, Mohamed Adel Musallam, Vincent Gaudilliere, Enjie Ghorbel, Kassem Al Ismaeil, Marcos Perez, and Djamila Aouada. Lspnet: A 2d localization-oriented spacecraft pose estimation neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2048–2056, 2021.

[16] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 544–560. Springer, 2020.

[17] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMLR, 2019.

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[19] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan (Celine) Lin. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021.

[20] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Fei-Fei Li, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[22] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[23] Mohamed Adel Musallam, Vincent Gaudilliere, Enjie Ghorbel, Kassem Al Ismaeil, Marcos Damian Perez, Michel Poucet, and Djamila Aouada. Spacecraft recognition leveraging knowledge of space environment: Simulator, dataset, competition design and analysis. In *2021 IEEE International Conference on Image Processing Challenges (ICIPC)*, pages 11–15, 2021.

[24] Oyebade Oyedotun, Djamila Aouada, and Bjorn Ottersten. Structured compression of deep neural networks with debiased elastic group lasso. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[25] Oyebade K Oyedotun, Abd El Rahman Shabayek, Djamila Aouada, and Björn Ottersten. Deep network compression with teacher latent subspace learning and lasso. *Applied Intelligence*, 51:834–853, 2021.

[26] Marcos Perez, Mohamed Adel Mohamed Ali, Albert Garcia Sanchez, Enjie Ghorbel, Kassem Al Ismaeil, Paul Le Henaff, and Djamila Aouada. Detection & identification of on-orbit objects using machine learning. In *European Conference on Space Debris*, volume 8, 2021.

[27] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine*

Learning, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[28] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[29] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[30] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[32] Nilotpal Sinha and Kuan-Wen Chen. Evolving neural architecture using one shot model. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 910–918, 2021.

[33] Nilotpal Sinha and Kuan-Wen Chen. Neural architecture search using progressive evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1093–1101, 2022.

[34] Nilotpal Sinha and Kuan-Wen Chen. Novelty driven evolutionary neural architecture search. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 671–674, 2022.

[35] Nilotpal Sinha and Kuan-Wen Chen. Neural Architecture Search Using Covariance Matrix Adaptation Evolution Strategy. *Evolutionary Computation*, pages 1–28, 08 2023.

[36] Sean C Smithson, Guang Yang, Warren J Gross, and Brett H Meyer. Neural networks designing neural networks: multi-objective hyper-parameter optimization. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2016.

[37] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics.

[38] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828, 2019.

[39] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[40] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[41] Weiqin Ying, Kaijie Zheng, Yu Wu, Junhui Li, and Xin Xu. Neural architecture search using multi-objective evolutionary algorithm based on decomposition. In *Artificial Intelligence Algorithms and Applications: 11th International Symposium, ISICA 2019, Guangzhou, China, November 16–17, 2019, Revised Selected Papers 11*, pages 143–154. Springer, 2020.

[42] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.

[43] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[44] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020.

[45] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 692–693, 2020.

[46] Xinyi Zhang, Weiwen Jiang, Yiyu Shi, and Jingtong Hu. When neural architecture search meets hardware implementation: from hardware awareness to co-design. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 25–30. IEEE, 2019.

[47] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[48] Xiawu Zheng, Xiang Fei, Lei Zhang, Chenglin Wu, Fei Chao, Jianzhuang Liu, Wei Zeng, Yonghong Tian, and Rongrong Ji. Neural architecture search with representation mutual information. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11912–11921, 2022.

[49] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

[50] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.

Supplementary

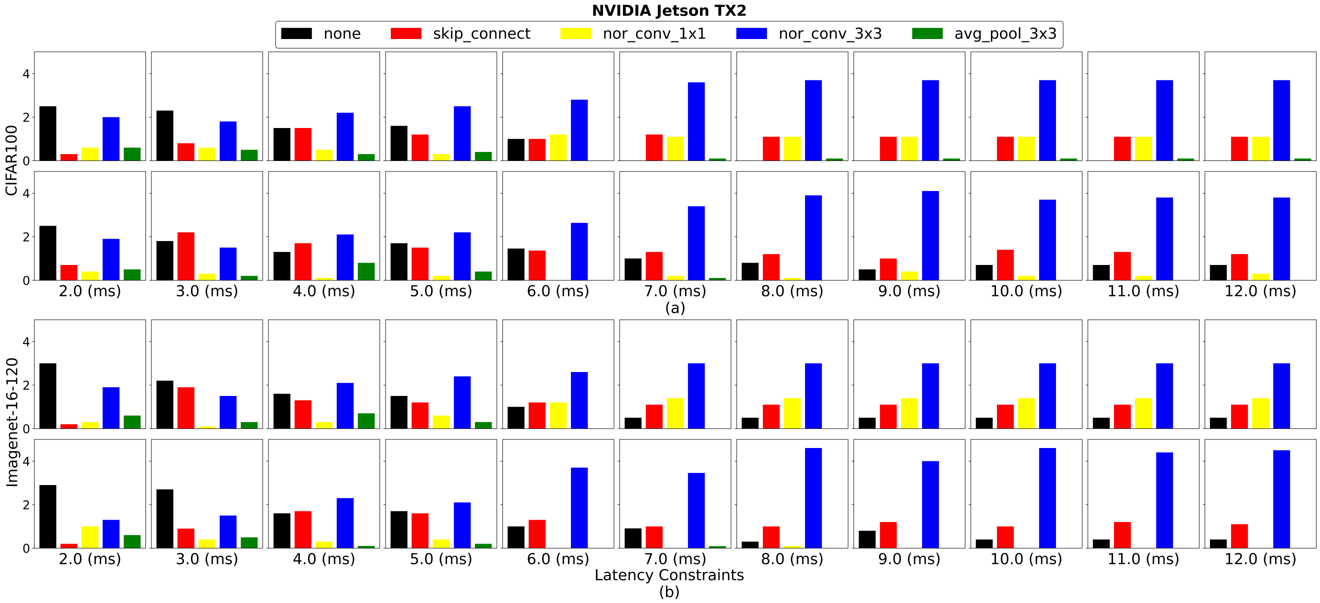


Figure S1: Results of HW-NAS with different latency constraints for (a) CIFAR100 (b) ImageNet16-120 experiments. The top row of each experiment shows the average number of different types of operations (i.e. convolution 3x3, convolution 1x1, max pooling 3x3, skip connect and none) present in the top-10 architectures of NAS-Bench-201 in terms of test accuracy. The bottom row of each experiment shows the same quantities present in the architectures discovered from ten independent architecture search runs. Note that the architecture search is performed in the NAS-Bench-201 search space for NVIDIA Jetson TX2 as the target hardware in this case. The similarity of the distribution of operations at different latency constraints may suggests that the proposed method converges to the high performing architecture sub-space.

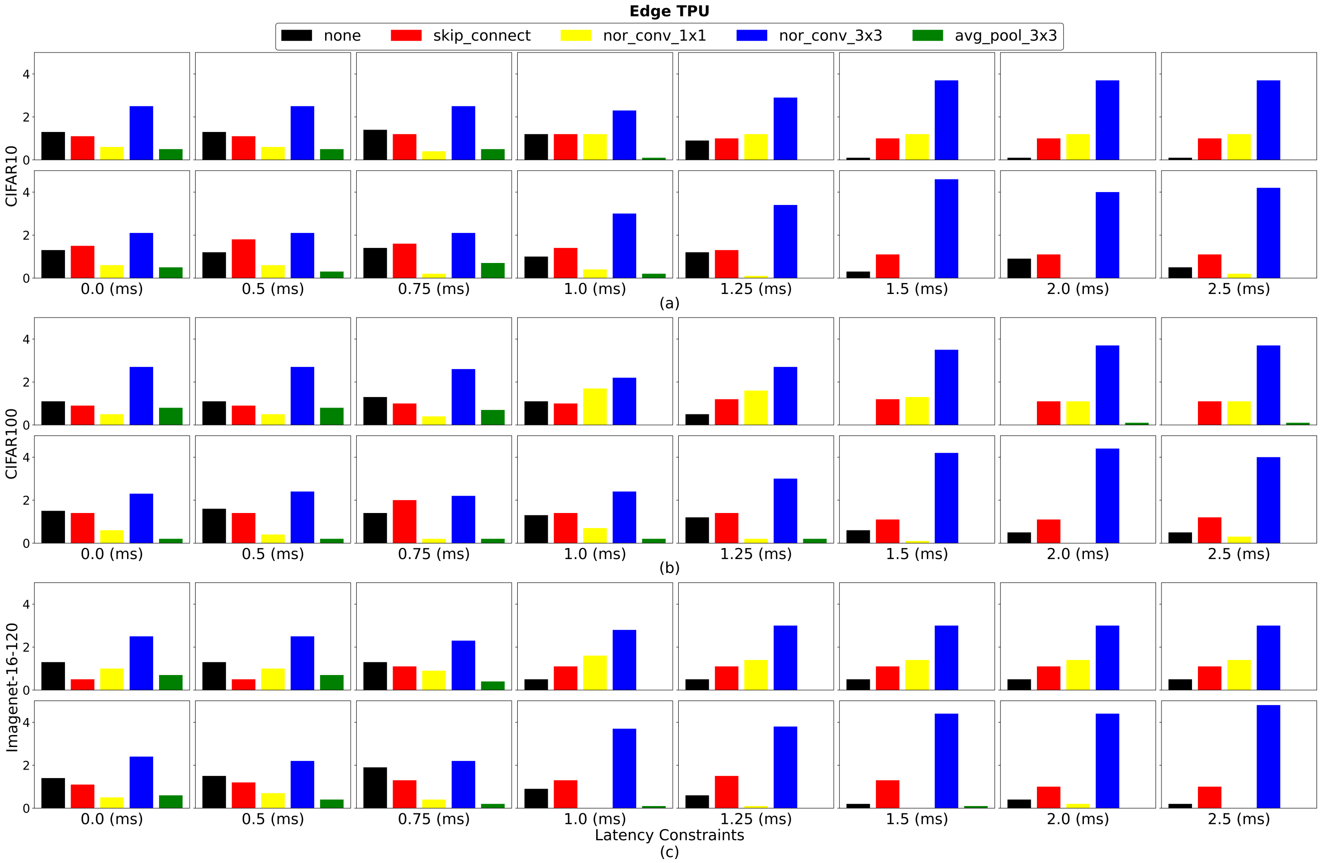


Figure S2: Same experiments as in Figure S1 with different latency constraints for Edge TPU on (a) CIFAR10 (b) CIFAR100 (c) ImageNet16-120.

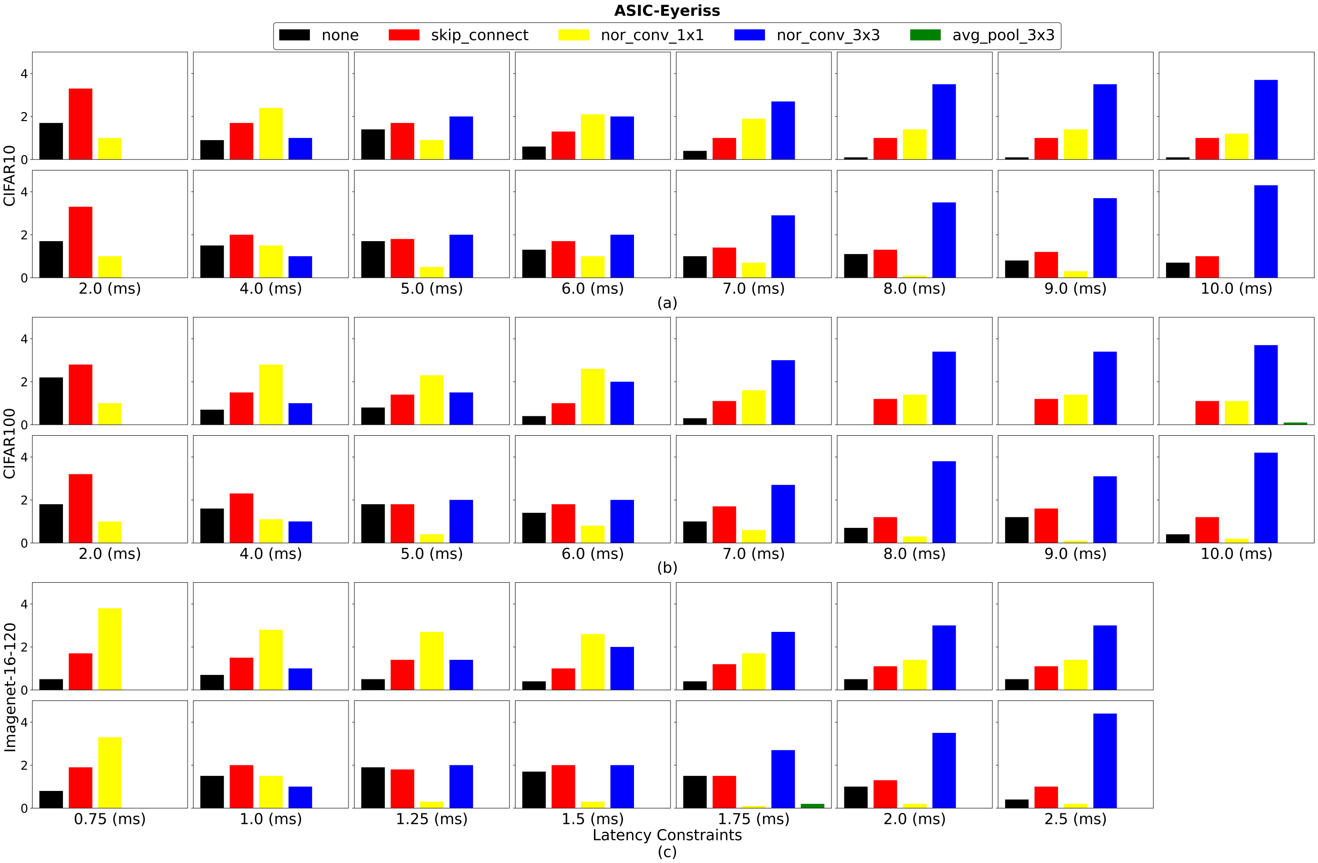


Figure S3: Same experiments as in Figure S1 with different latency constraints for ASIC-Eyeriss on (a) CIFAR10 (b) CIFAR100 (c) ImageNet16-120.

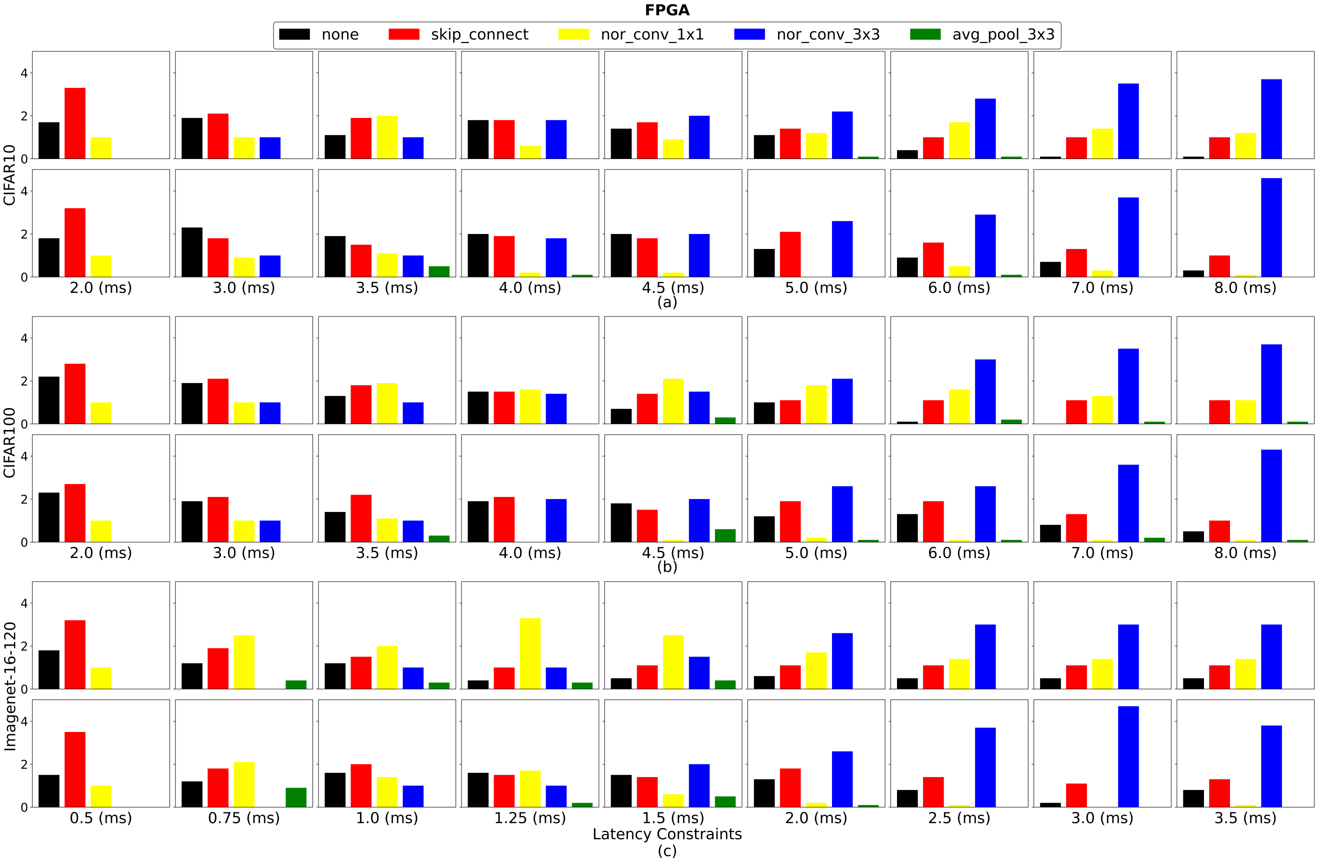


Figure S4: Same experiments as in Figure S1 with different latency constraints for FPGA on (a) CIFAR10 (b) CIFAR100 (c) ImageNet16-120.

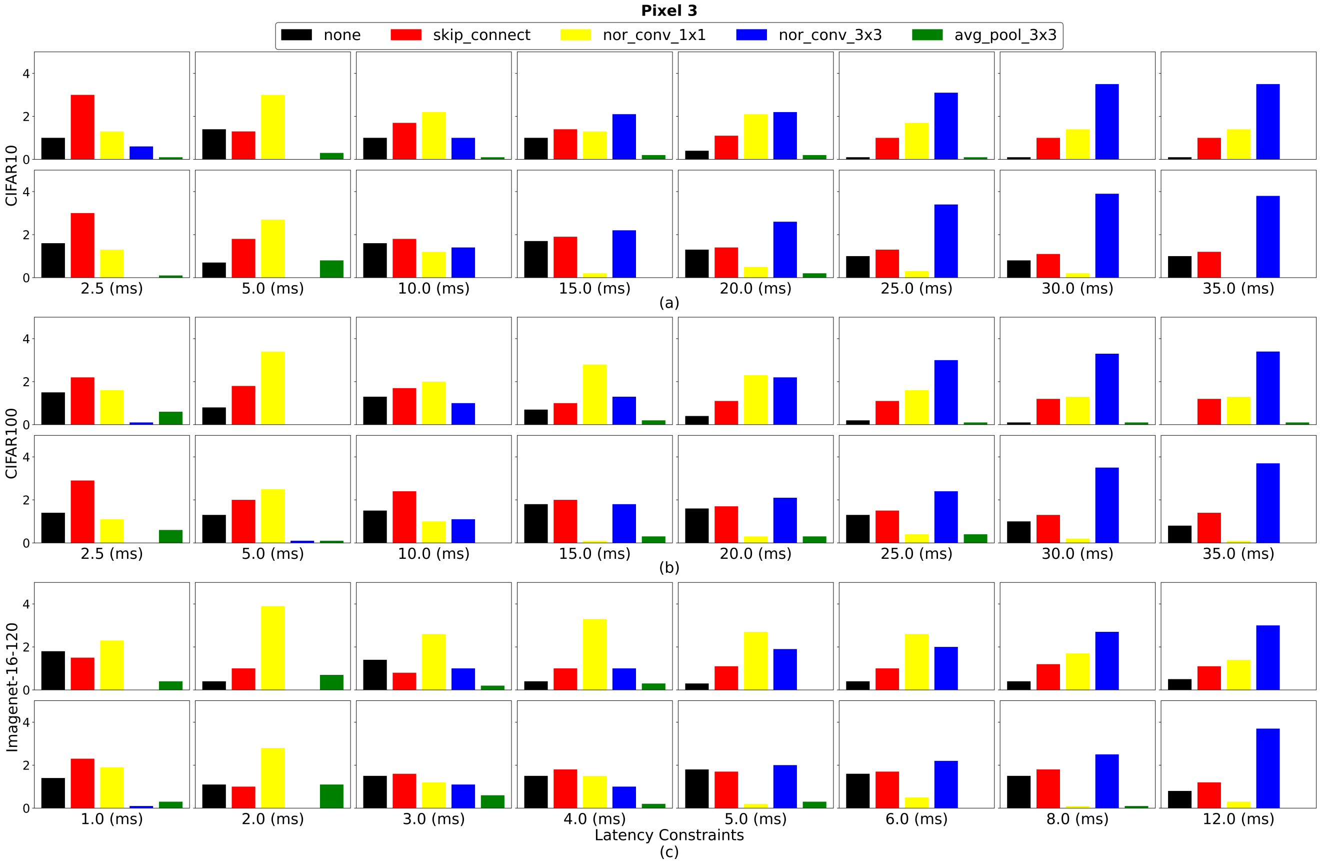


Figure S5: Same experiments as in Figure S1 with different latency constraints for Pixel 3 on (a) CIFAR10 (b) CIFAR100 (c) ImageNet16-120.

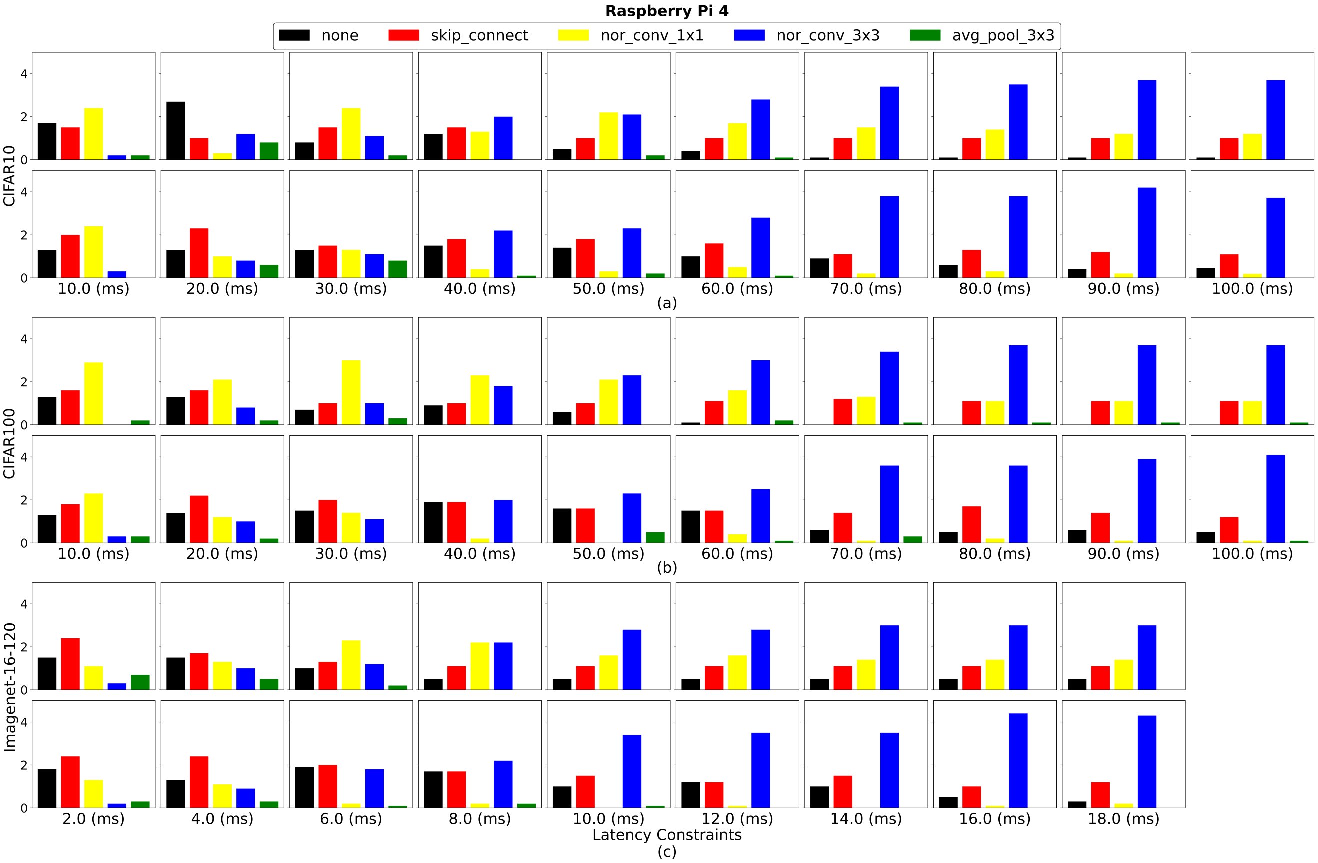


Figure S6: Same experiments as in Figure S1 with different latency constraints for Raspberry Pi 4 on (a) CIFAR10 (b) CIFAR100 (c) ImageNet16-120.

S1. Datasets

CIFAR-10 and **CIFAR-100** [18] have 50K training images and 10K testing images with images classified into 10 and 100 classes respectively. **ImageNet**[10] is a well known benchmark for image classification containing 1K classes with 1.28 million training images and 50K test images. **ImageNet-16-120** [6] is a variant of ImageNet which is downsampled to 16x16 pixels with labels $\in [0, 120]$ to construct ImageNet-16-120 dataset.