Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours

Dimitrios Stamoulis[1]✉, Ruizhou Ding[1], Di Wang[2], Dimitrios Lymberopoulos[2], Bodhi Priyantha[2], Jie Liu[3], and Diana Marculescu[1]

[1] Department of ECE, Carnegie Mellon University, Pittsburgh, PA, USA
[2] Microsoft, Redmond, WA, USA
[3] Harbin Institute of Technology, Harbin, China
dstamoul@andrew.cmu.edu

Abstract. Can we automatically design a Convolutional Network (ConvNet) with the highest image classification accuracy under the latency constraint of a mobile device? Neural architecture search (NAS) has revolutionized the design of hardware-efficient ConvNets by automating this process. However, the NAS problem remains challenging due to the combinatorially large design space, causing a significant searching time (at least 200 GPU-hours). To alleviate this complexity, we propose *Single-Path NAS*, a novel differentiable NAS method for designing hardware-efficient ConvNets in **less than 4 hours**. Our contributions are as follows: 1. **Single-path search space**: Compared to previous differentiable NAS methods, *Single-Path NAS* uses one single-path over-parameterized ConvNet to encode all architectural decisions with shared convolutional kernel parameters, hence drastically decreasing the number of trainable parameters and the search cost down to few epochs. 2. **Hardware-efficient ImageNet classification**: *Single-Path NAS* achieves 74.96% top-1 accuracy on ImageNet with 79ms latency on a Pixel 1 phone, which is state-of-the-art accuracy compared to NAS methods with similar inference latency constraints ($\leq 80ms$). 3. **NAS efficiency**: *Single-Path NAS* search cost is only **8 epochs** (30 TPU-hours), which is up to **5,000× faster** compared to prior work. 4. **Reproducibility**: Unlike all recent mobile-efficient NAS methods which only release pretrained models, we open-source our entire codebase at: `https://github.com/dstamoulis/single-path-nas`.

Keywords: Neural Architecture Search · Hardware-aware ConvNets.

1 Introduction

"*Is it possible to reduce the considerable search cost of Neural Architecture Search (NAS) down to only few hours?*" NAS has revolutionized the design of Convolutional Networks (ConvNets) [25], yielding state-of-the-art results in several deep learning applications [14]. NAS methods already have a profound impact on the design of hardware-efficient ConvNets for computer vision tasks under the constraints (*e.g.*, inference latency) imposed by mobile devices [18].

Despite the recent breakthroughs, NAS remains an intrinsically costly optimization problem. Searching for which convolution operation to use per ConvNet layer, gives rise to a combinatorially large search space: *e.g.*, for a mobile-efficient ConvNet with 22 layers, choosing among five candidate operations yields $5^{22} \approx 10^{15}$ possible ConvNet architectures. To traverse this design space, earlier NAS methods guide the exploration via reinforcement learning (RL) [18]. Nonetheless, training the RL controller poses prohibitive computational challenges, and thousands of candidate ConvNets need to be trained [19].

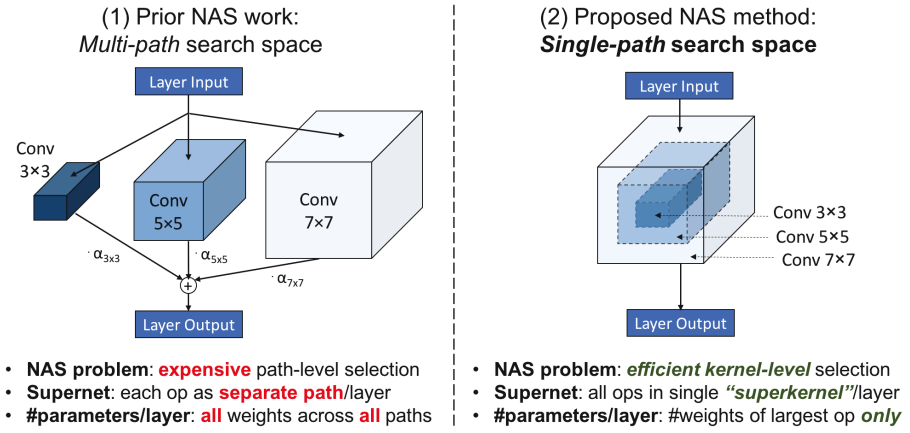

Fig. 1. *Single-Path NAS* directly optimizes for the subset of convolution kernel weights and searches over an over-parameterized "**superkernel**" in each ConvNet layer (right). This **novel view** of the design space eliminates the need for maintaining separate paths for each candidate operation, as in previous *multi-path* approaches (left). Our **key insight** drastically reduces the NAS search cost by up to **5,000**× with state-of-the-art accuracy on ImageNet for the same mobile latency setting, compared to prior work.

Inefficiencies of *multi-path* NAS: Recent NAS literature has seen a shift towards one-shot differentiable formulations [12, 13, 20] which search over a supernet that encompasses all candidate architectures. Specifically, current NAS methods relax the combinatorial optimization problem of finding the optimal ConvNet architecture to an operation/path selection problem: first, an over-parameterized, *multi-path* supernet is constructed, where, for each layer, every candidate operation is added as a *separate* trainable path, as illustrated in Figure 1 (left). Next, NAS formulations solve for the (distributions of) paths of the *multi-path* supernet that yield the optimal architecture.

As expected, naively branching out all paths is inefficient due to an intrinsic limitation: the number of trainable parameters that need to be maintained and updated during the search grows linearly with respect to the number of candidate operations per layer [1]. To tame the memory explosion introduced by the *multi-path* supernet, current methods employ creative "workaround" solutions:

e.g., searching on a proxy dataset (subset of ImageNet [19]), or employing a memory-wise scheme with only a subset of paths being updated during the search [3]. Nevertheless, these techniques remain considerably costly, with an overall computational demand of at least 200 GPU-hours.

In this paper, we propose *Single-Path NAS*, a novel NAS method for designing hardware-efficient ConvNets in **less than 4 hours**. Our **key insight** is illustrated in Figure 1 (right). We build upon the observation that different candidate convolutional operations in NAS can be viewed as subsets of a single **"superkernel"**. Without having to choose among different paths/operations as in *multi-path* methods, we instead solve the NAS problem as *finding which subset of kernel weights to use in each ConvNet layer*. By sharing the convolutional kernel weights, we encode all candidate NAS operations into a single superkernel, *i.e.*, with a single path, for each layer of the one-shot NAS supernet. This novel encoding of the design space yields a drastic reduction to the number of trainable parameters/gradients, allowing our NAS method to use batch sizes of 1024, a four-fold increase compared to prior art's search efficiency.

Our contributions are as follows:

1. **Single-path NAS**: We propose a novel view of the one-shot, supernet-based design space, hence drastically decreasing the number of trainable parameters. To the best of our knowledge, this is the *first* work to formulate the NAS problem as finding the subset of kernel weights in each ConvNet layer.
2. **State-of-the-art results**: *Single-Path NAS* achieves 74.96% top-1 accuracy on ImageNet with 79ms latency on a Pixel 1, *i.e.*, a +0.31% improvement over the current best hardware-aware NAS [18] under $80ms$.
3. **NAS efficiency**: The overall search cost is only **8 epochs**, *i.e.*, **3.75 hours** on TPUs (30 TPU-hours), up to **5,000× faster** compared to prior work.
4. **Reproducibility**: Unlike recent hardware-efficient NAS methods which release pretrained models only, we open-source and fully document our method at: `https://github.com/dstamoulis/single-path-nas`.

2 Related Work

Hardware-efficient ConvNets: While complex ConvNet designs have unlocked unprecedented performance levels in computer vision tasks, the accuracy improvement has come at the cost of higher computational complexity, making the deployment of state-of-the-art ConvNets to mobile devices challenging [17]. To this end, a significant body of prior work aims to co-optimize for the inference latency of ConvNets. Earlier approaches focus on human expertise to introduce hardware-efficient operations [9, 15, 22]. Pruning [4] and quantization [7] methods share the same goal to improve the efficiency of ConvNets.

Neural Architecture Search (NAS): NAS aims at automating the process of designing ConvNets, giving rise to methods based on reinforcement learning (RL), evolutionary algorithms, or gradient-based methods [12–14, 24, 25]. Earlier approaches train an agent (*e.g.*, RNN controller) by sampling candidate architectures over a cell-based design space, where the same cell is repeated in all layers

and the focus is on searching the cell architecture [25]. Nonetheless, training the controller over different architectures makes the search costly.

Hardware-aware NAS: Earlier NAS methods focused on maximizing accuracy under FLOPs constraints [20, 23], but low FLOP count does not necessarily translate to hardware efficiency [8, 16]. More recent methods incorporate hardware terms (*e.g.*, runtime, power) into cell-based NAS formulations [8, 10], but cell-based implementations are not hardware friendly [19]. Breaking away from cell-based assumptions in the search space encoding, recent work employs NAS over a generalized MobileNetV2-based design space introduced in [18].

Hardware-aware Differentiable NAS: Recent NAS literature has seen a shift towards one-shot NAS formulations [13, 20]. Gradient-based NAS in particular has gained increased popularity and has achieved state-of-the-art results [12]. One-shot-based methods use an over-parameterized super-model network, where, for each layer, every candidate operation is added as a separate trainable path. Nonetheless, *multi-path* search spaces have an intrinsic limitation: the number of trainable parameters that need to be maintained and updated with gradients during the search grows linearly with respect to the number of different convolutional operations per layer, resulting in memory explosion [1, 3].

To this end, state-of-the-art approaches employ different novel "workaround" solutions. FBNet [19] searches on a "proxy" dataset (*i.e.*, subset of the ImageNet dataset). Despite the decreased search cost thanks to the reduced number of training images, these approaches do not address the fact that the entire super-model needs to be maintained in memory during search, hence the efficiency is limited due to inevitable use of smaller batch sizes. ProxylessNAS [3] has employed a memory-wise one-shot model scheme, where only a set of paths is updated during the search. However, such implementation-wise improvements do not address a second key suboptimality of one-shot approaches, *i.e.*, the fact that separate gradient steps are needed to update the weights and the architectural decisions interchangeably [12]. Although the number of trainable parameters, with respect to the memory cost, is kept to the same level at any step, the way that *multi-path*-based methods traverse the design space remains inefficient.

3 Proposed Method: *Single-Path* NAS

In this Section, we present our proposed method. First, we discuss our novel *single-path* view (Subsection 3.1) of the search space. Next, we encode the NAS problem as finding the subset of convolution weights over the over-parameterized superkernel (Subsection 3.2), and we discuss how it compares to existing *multi-path*-based NAS (Subsection 3.3). Last, we formulate the hardware-aware NAS objective function, where we incorporate an accurate inference latency model of ConvNets executing on the Pixel 1 smartphone (Subsection 3.4).

3.1 Mobile ConvNets Search Space: A Novel View

Background - Mobile ConvNets: State-of-the-art NAS builds upon a fixed "backbone" ConvNet [3] inspired by the MobileNetV2 design [15], illustrated in

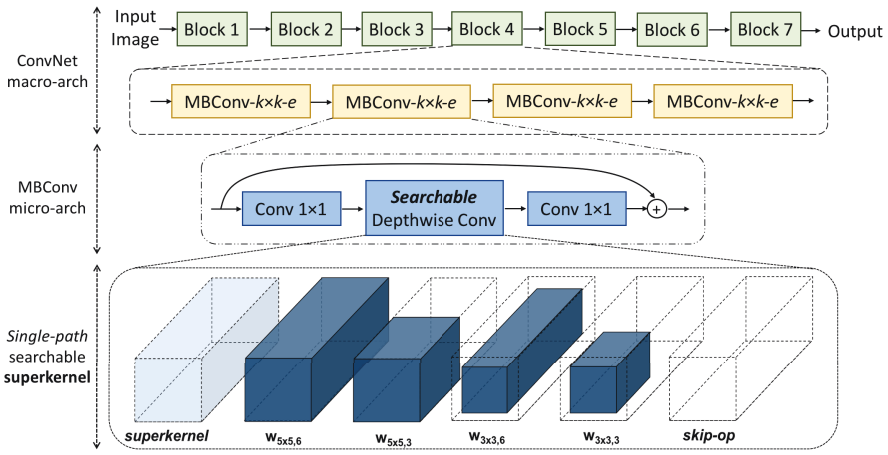

Fig. 2. ***Single-path*** **search space**: Our method builds upon *hierarchical* MobileNetV2-like search spaces [15, 18], where the goal is to identify the type of mobile inverted bottleneck convolution (MBConv) [15] per layer. Our *one-shot supernet* encapsulates all possible NAS architectures in the search space, without the need for appending each candidate operation as a separate path. *Single-Path* NAS directly searches over the weights of a **searchable** superkernel that encodes all MBConv types.

Figure 2 (top). Specifically, in this fixed macro-architecture, except for the head and stem layers, all ConvNet layers are grouped into blocks based on their filter sizes. The filter numbers per block follow the values in [19], *i.e.*, we use seven blocks with up to four layers each. Each layer of these blocks follows a mobile inverted bottleneck convolution MBConv [15] micro-architecture, which consists of a point-wise (1×1) convolution, a $k \times k$ depthwise convolution, and a linear 1×1 convolution (Figure 2, middle). Unless the layer has a stride value of two, a skip path is introduced to provide a residual connection from input to output.

Each MBConv layer is parameterized by k, *i.e.*, the kernel size of the depthwise convolution, and by expansion ratio e, *i.e.*, the ratio between the output and input of the first 1×1 convolution. Based on this parameterization, we denote each MBConv as MBConv-$k \times k$-e. Mobile-efficient NAS aims to choose each MBConv-$k \times k$-e layer, by selecting among different k and e values [3, 19]. In particular, we consider MBConv layers with kernel sizes $\{3, 5\}$ and expansion ratios $\{3, 6\}$. NAS also considers a special *skip-op* "layer", which "zeroes-out" the kernel and feeds the input directly to the output, *i.e.*, the entire layer is dropped.

Novel view of design space: Our *key insight* is illustrated in Figure 2. We build upon the observation that different candidate convolutional operations in NAS can be viewed as subsets of the weights of an over-parameterized single superkernel (Figure 2, bottom). This observation allows us to view the NAS combinatorial problem as *finding which subset of kernel weights to use in each MBConv layer*. This observation is important since it allows sharing the kernel parameters across different MBConv architectural options. As shown in Figure 2,

we encode all candidate NAS operations to this single **superkernel**, *i.e.*, with a **single path**, for each layer of the one-shot NAS supernet.

3.2 Proposed Methodology: Single-Path NAS formulation

Key idea - Relaxing NAS decisions over an over-parameterized kernel: To simplify notation and to illustrate the key idea, without loss of generality, we show the case of choosing between a 3×3 or a 5×5 kernel for an MBConv layer. Let us denote the weights of the two candidate kernels as $\mathbf{w}_{3\times3}$ and $\mathbf{w}_{5\times5}$, respectively. As shown in Figure 3 (left), we observe that the weights of the 3×3 kernel can be viewed as the *inner* core of the weights of the 5×5 kernel, while "zeroing" out the weights of the "*outer*" shell. We denote this (*outer*) subset of weights (that does not contribute to output of the 3×3 kernel but only to the 5×5 kernel), as $\mathbf{w}_{5\times5\setminus3\times3}$. Hence, the NAS architectural choice of using the 5×5 convolution corresponds to using both the *inner* $\mathbf{w}_{3\times3}$ weights and the *outer* shell, *i.e.*, $\mathbf{w}_{5\times5} = \mathbf{w}_{3\times3} + \mathbf{w}_{5\times5\setminus3\times3}$ (Figure 3, left).

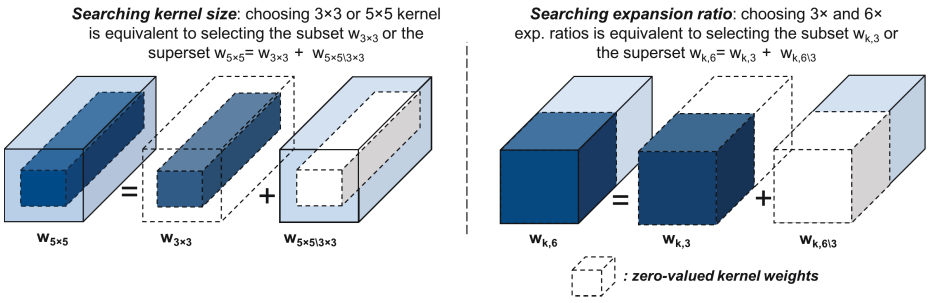

Fig. 3. Encoding NAS decisions into the **superkernel**: We formulate all candidate convolution operations (*i.e.*, different kernel size (left) and expansion ratio (right) values) directly into the searchable superkernel.

We can therefore encode the NAS decision directly into the superkernel of an MBConv layer as a function of kernel weights as follows:

$$\mathbf{w}_k = \mathbf{w}_{3\times3} + \mathbb{1}(\text{use } 5 \times 5) \cdot \mathbf{w}_{5\times5\setminus3\times3} \tag{1}$$

where $\mathbb{1}(\cdot)$ is the indicator function that encodes the architectural NAS choice, *i.e.*, if $\mathbb{1}(\cdot) = 1$ then $\mathbf{w}_k = \mathbf{w}_{3\times3} + \mathbf{w}_{5\times5\setminus3\times3} = \mathbf{w}_{5\times5}$, else $\mathbb{1}(\cdot) = 0$ then $\mathbf{w}_k = \mathbf{w}_{3\times3}$.

Trainable indicator/condition function: While the indicator function encodes the NAS decision, a critical choice is how to formulate the condition over which the $\mathbb{1}(\cdot)$ is evaluated. Our intuition is that, for an indicator function that represents whether to use the subset of weights, its condition should be *directly a function of the subset's weights*. Thus, our goal is to define an "importance" signal of the subset weights that intrinsically captures their contribution to the

overall ConvNet loss. We draw inspiration from weight-based conditions that have been successfully used for quantization-related decisions [6] and we use the *group Lasso term*. Specifically, for the indicator related to the $\mathbf{w}_{5\times5\backslash3\times3}$ "outer shell" decision, we write the following condition:

$$\mathbf{w}_k = \mathbf{w}_{3\times3} + \mathbb{1}(\left\|\mathbf{w}_{5\times5\backslash3\times3}\right\|^2 > t_{k=5}) \cdot \mathbf{w}_{5\times5\backslash3\times3} \tag{2}$$

where $t_{k=5}$ is a latent variable that controls the decision (*e.g.*, a threshold value) of selecting kernel 5×5. The threshold will be compared to the Lasso term to determine if the *outer* $\mathbf{w}_{5\times5\backslash3\times3}$ weights are used to the overall convolution. It is important to notice that, instead of picking the thresholds (*e.g.*, $t_{k=5}$) by hand, we seamlessly treat them as trainable parameters to learn via gradient descent. To compute the gradients for thresholds, we relax the indicator function $g(x,t) = \mathbb{1}(x > t)$ to a sigmoid function, $\sigma(\cdot)$, when computing gradients, *i.e.*, $\hat{g}(x,t) = \sigma(x > t)$.

Searching for expansion ratio and skip-op: Since the result of the kernel-based NAS decision $\mathbf{w}_k$ (Equation 2) is a convolution kernel itself, we can in turn apply our formulation to also encode NAS decisions for the expansion ratio of the $\mathbf{w}_k$ kernel. As illustrated in Figure 3 (right), the channels of the depthwise convolution in an MBConv-$k \times k$-3 layer with expansion ratio $e = 3$ can be viewed as using one half of the channels of an MBConv-$k \times k$-6 layer with expansion ratio $e = 6$, while "zeroing" out the second half of channels $\{\mathbf{w}_{k,6\backslash3}\}$. Finally, by "zeroing" out the first half of the output filters as well, the entire superkernel contributes nothing if added to the residual connection of the MBConv layer: *i.e.*, by deciding if $e = 3$, we can encode the NAS decision of using, or not, only the "skip-op" path. For both decisions over $\mathbf{w}_k$ kernel, we write:

$$\mathbf{w} = \mathbb{1}(\|\mathbf{w}_{k,3}\|^2 > t_{e=3}) \cdot (\mathbf{w}_{k,3} + \mathbb{1}(\left\|\mathbf{w}_{k,6\backslash3}\right\|^2 > t_{e=6}) \cdot \mathbf{w}_{k,6\backslash3}) \tag{3}$$

Hence, for input $\mathbf{x}$, the output of the i-th MBConv layer of the network is:

$$o^i(\mathbf{x}) = \text{conv}(\mathbf{x}, \mathbf{w}^i | t^i_{k=5}, t^i_{e=6}, t^i_{e=3}) \tag{4}$$

Searchable MBConv kernels: Each MBConv uses 1×1 convolutions for the point-wise (first) and linear stages, while the kernel-size decisions affect only the (middle) $k\times k$ depthwise convolution (Figure 2). To this end, we use our searchable $k \times k$ depthwise kernel at this middle stage. In terms of number of channels, the depthwise kernel depends on the point-wise 1×1 output, which allows us to directly encode the expansion ratio e at the middle stage as well: by setting the point-wise 1×1 output to the maximum candidate expansion ratio, we can instead solve for which of them not to "zero" out at the depthwise (middle) state. In other words, we directly use our searchable depthwise convolution superkernel to effectively encode the NAS decision for the expansion ratio. Hence, our *single-path*, convolution-based formulation can sufficiently capture any MBConv type (*e.g.*, MBConv-3×3-6, MBConv-5×5-3, *etc.*) in the MobileNetV2-based design space (Figure 2).

3.3 Single-Path vs. Existing Multi-Path Assumptions

Comparison with multi-path over-parameterized networks: We briefly illustrate how our *single-path* formulation compares to multi-path NAS approaches. In existing methods [3, 12, 19], the output of each layer i is a (weighted) sum defined over the output of N different paths, where each path j corresponds to a different candidate kernel $\mathbf{w}^{i,j}_{k\times k,e}$. The weight of each path $\alpha^{i,j}$ corresponds to the probability that this path is selected over the parallel paths:

$$o^i_{multi-path}(\mathbf{x}) = \sum_{j=1}^{N} \alpha^{i,j} \cdot o^{i,j}(\mathbf{x}) = \alpha^{i,0} \cdot \text{conv}(\mathbf{x}, \mathbf{w}^{i,0}_{3\times 3}) + \cdots + \alpha^{i,N} \cdot \text{conv}(\mathbf{x}, \mathbf{w}^{i,N}_{5\times 5}) \tag{5}$$

It is easy to see how our novel *single-path* view is advantageous, since the output of the convolution at layer i of our search space is *directly a function of the weights of our single over-parameterized kernel* (Equation 4):

$$o^i_{single-path}(\mathbf{x}) = o^i(\mathbf{x}) = \text{conv}(\mathbf{x}, \mathbf{w}^i | t^i_{k=5}, t^i_{e=6}, t^i_{e=3}) \tag{6}$$

Comparison with multi-path NAS optimization: Multi-path NAS methods solve for the optimal architecture parameters α (path weights), such that the weights w_α of the corresponding α-architecture have minimal loss $\mathcal{L}(\alpha, w_\alpha)$:

$$\min_{\alpha} \min_{w_\alpha} \mathcal{L}(\alpha, w_\alpha) \tag{7}$$

However, solving Equation 7 gives rise to a challenging *bi-level* optimization problem [12]. Existing methods interchangeably update the α's while freezing the w's and vice versa, leading to more gradient steps.

In contrast, with our *single-path* formulation, the overall network loss is directly a function of the superkernel weights, where the learnable kernel- and expansion ratio-related threshold variables, $\mathbf{t}_k$ and $\mathbf{t}_e$, are directly derived as a function (norm) of the kernel weights $\mathbf{w}$. Consequently, *Single-Path NAS* formulates the NAS problem as solving *directly over the weight kernels* $\boldsymbol{w}$ *of a single-path, compact neural network*. Formally, the NAS problem becomes:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w} | \mathbf{t}_k, \mathbf{t}_e) \tag{8}$$

Efficiency of *Single-Path NAS*: Unlike the bi-level optimization problem in prior work, solving our NAS formulation in Equation 8 is as expensive as training the weights of a single-path, **branchless**, compact neural network with vanilla gradient descent. Therefore, our formulation eliminates the need for separate gradient steps between the ConvNet weights and the NAS parameters. Moreover, the reduction of the trainable parameters $\mathbf{w}$ per se, further leads to a drastic reduction of the search cost down to **just a few epochs**, as our experimental results show later in Section 4. Our NAS problem formulation allows us to efficiently solve Equation 8 with batch sizes of 1024, a four-fold increase compared to prior art's search efficiency.

3.4 Hardware-Aware NAS with Differentiable Runtime Loss

To design hardware-efficient ConvNets, the differentiable objective in Equation 8 should reflect both the accuracy of the searched ConvNet and its inference latency on the target hardware. Hence, we use a latency-aware formulation [3, 19]:

$$\mathcal{L}(\mathbf{w}|\mathbf{t}_k, \mathbf{t}_e) = CE(\mathbf{w}|\mathbf{t}_k, \mathbf{t}_e) + \lambda \cdot \log(R(\mathbf{w}|\mathbf{t}_k, \mathbf{t}_e)) \tag{9}$$

The first term CE corresponds to the cross-entropy loss of the single-path model. The hardware-related term R is the runtime in milliseconds (ms) of the searched NAS model on the target mobile platform. Finally, the coefficient λ modulates the trade-off between cross-entropy and runtime.

Runtime model over the single-path design space: To preserve the differentiability of the objective, another critical choice is the formulation of the latency term R. Prior art has showed that the total network latency of a mobile ConvNet can be modeled as the sum of each i-th layer's runtime R^i, since the runtime of each operator is independent of other operators [2, 3, 19]:

$$R(\mathbf{w}|\mathbf{t}_k, \mathbf{t}_e) = \sum_i R^i(\mathbf{w}^i|\mathbf{t}_k^i, \mathbf{t}_e^i) \tag{10}$$

For our approach, we adapt the per-layer runtime model as a function of the NAS-related decisions $\mathbf{t}$. We profile the target mobile platform (Pixel 1) and we record the runtime for each candidate kernel operation per layer i, *i.e.*, $R^i_{3\times3,3}$, $R^i_{3\times3,6}$, $R^i_{5\times5,3}$, and $R^i_{5\times5,6}$. We denote the runtime of layer i by following the notation in Equation 3. Specifically, the runtime of layer i is defined first as a function of the expansion ratio decision:

$$R^i_e = \mathbb{1}(\|\mathbf{w}_{k,3}\|^2 > \mathbf{t}_{e=3}) \cdot (R^i_{5\times5,3} + \mathbb{1}(\left\|\mathbf{w}_{k,6\setminus3}\right\|^2 > \mathbf{t}_{e=6}) \cdot (R^i_{5\times5,6} - R^i_{5\times5,3})) \tag{11}$$

Next, by incorporating the kernel size decision, the total runtime is:

$$R^i = \frac{R^i_{3\times3,6}}{R^i_{5\times5,6}} \cdot R^i_e + R^i_e \cdot (1 - \frac{R^i_{3\times3,6}}{R^i_{5\times5,6}}) \cdot \mathbb{1}(\left\|\mathbf{w}_{5\times5\setminus3\times3}\right\|^2 > \mathbf{t}_{k=5}) \tag{12}$$

As in Equation 2, we relax the indicator function to a sigmoid function $\sigma(\cdot)$ when computing gradients. By using this model, the runtime term in the loss function remains differentiable with respect to layer-wise NAS choices. As we show in our results, the model is accurate, with an average prediction error of 1.76%.

4 Experiments

4.1 Experimental Setup

Dataset and target application: We use *Single-Path NAS* to design ConvNets for image classification on ImageNet. We use Pixel 1 as the target mobile platform. The choice of this experimental setup is important, since it allows for

a representative comparison with prior hardware-efficient NAS methods that optimize for the same Pixel 1 device around a target latency of $80ms$ [3, 18].

Implementation and deployment: We implement our NAS framework in TensorFlow (`TF` version 1.12). During both search and training stages, we use TPUs (version 2) [11]. To this end, we build on top of the `TPUEstimator` classes following the TPU-related documentation of the MnasNet repository[4]. Last, all models (ours and prior work) are deployed with TensorFlow TFLite to the mobile device. On the device, we profile runtime using the Facebook AI Performance Evaluation Platform (`FAI-PEP`)[5] that supports profiling for `tflite` models with detailed per-layer runtime breakdown.

Implementing the custom superkernels: We use `Keras` to implement our trainable superkernels. Specifically, we define a custom `Keras`-based depthwise convolution kernel where the output is a function of both the weights and the threshold-based decisions (Equations 2-3). Our custom layer also returns the effective runtime of the layer (Equations 11-12). We document our implementation in our project GitHub repository: `https://github.com/dstamoulis/single-path-nas`, with detailed steps on how to reproduce the results.

4.2 State-of-the-art Runtime-Constrained ImageNet Classification

We apply our method to design ConvNets for the Pixel 1 phone with an overall target latency of $80ms$. We train the derived *Single-Path* NAS model for 350 epochs, following the MnasNet training schedule [18]. We compare our method with mobile ConvNets designed by human experts and state-of-the-art NAS methods in Table 1, in terms of classification accuracy and search cost. In terms of hardware efficiency, prior work has shown that low FLOP count does not necessarily translate to high hardware efficiency [8], we therefore evaluate the various NAS methods with respect to the inference runtime on Pixel 1 ($\leq 80ms$).

Enabling a representative comparison: While we provide the original values from the respective papers, our goal is to ensure a fair comparison. To this end, we retrain the baseline models following the same schedule (in fact, we find that the MnasNet-based training schedule improves the top1 accuracy compared to what is reported in several previous methods). Similarly, we profile the models on the same Pixel 1 device. For prior work that does not optimize for Pixel 1, we retrain and profile their model closest to the MnasNet baseline (*e.g.*, the FBNet-B and ChamNet-B networks [5, 19], since the authors use these ConvNets to compare against the MnasNet model). Finally, to enable a representative comparison of the search cost per method, we directly report the number of epochs reported per method, hence canceling out the effect of different hardware systems (GPU vs TPU hours).

ImageNet classification: Table 1 shows that our *Single-Path* NAS achieves top-1 accuracy of **74.96%**, which is the new state-of-the-art ImageNet accuracy among hardware-efficient NAS methods. More specifically, **our method**

[4] `https://github.com/tensorflow/tpu/tree/master/models/official/mnasnet`
[5] `https://github.com/facebook/FAI-PEP`

Table 1. *Single-Path* NAS achieves state-of-the-art accuracy (%) on ImageNet for similar mobile latency setting compared to previous NAS methods ($\leq 80ms$ on Pixel 1), with up to $5,000\times$ reduced search cost in terms of number of epochs. *The search cost in epochs is estimated based on the claim [3] that ProxylessNAS is $200\times$ faster than MnasNet. ‡ChamNet does not detail the model derived under runtime constraints [5] so we cannot retrain or measure the latency.

Method	Top-1 Acc (%)	Top-5 Acc (%)	Mobile Runtime (ms)	Search Cost (epochs)
MobileNetV1 [9]	70.60	89.50	113	-
MobileNetV2 1.0x [15]	72.00	91.00	75.00	
MobileNetV2 1.0x (our impl.)	73.59	91.41	73.57	
Random search	73.78 ± 0.85	91.42 ± 0.56	77.31 ± 0.9 ms	-
MnasNet 1.0x [18]	74.00	91.80	76.00	40,000
MnasNet 1.0x (our impl.)	74.61	91.95	74.65	
ChamNet-B [5]	73.80	–	–	240‡
ProxylessNAS-R [3]	74.60	92.20	78.00	200*
ProxylessNAS-R (our impl.)	74.65	92.18	77.48	
FBNet-B [19]	74.1	-	-	90
FBNet-B (our impl.)	73.70	91.51	78.33	
Single-Path NAS (**proposed**)	**74.96**	**92.21**	79.48	**8** (**3.75 hours**)

achieves better top-1 accuracy than ProxylessNAS by +0.31%, while maintaining on par target latency of $\leq 80ms$ on the same target mobile phone. *Single-Path* NAS outperforms methods in this mobile latency range, *i.e.*, better than MnasNet (+0.35%), FBNet-B (+0.86%), and MobileNetV2 (+1.37%).

NAS search cost: *Single-Path* NAS has **orders of magnitude reduced search cost** compared to all previous hardware-efficient NAS methods. Specifically, MnasNet reports that the controller uses 8k sampled models, each trained for 5 epochs, for a total of 40k train epochs. In turn, ChamNet trains an accuracy predictor on 240 samples, which assuming an aggressively fast training schedule of five epochs per sample (same as in MnasNet), corresponds to a total search cost of 1.2k epochs. ProxylessNAS reports $200\times$ search cost improvement over MnasNet, hence the overall cost is the TPU-equivalent of 200 epochs. Finally, FBNet reports 90 epochs of training on a proxy dataset (10% of ImageNet). While the number of images per epoch is reduced, we found that a TPU can accommodate a FBNet-like supermodel with maximum batch size of 128, hence the number of steps per FBNet epoch are still $8\times$ more compared to the steps per epoch in our method.

In comparison, *Single-Path NAS* has a total cost of eight epochs, which is **5,000**$\times$ faster than MnasNet, **25**$\times$ faster than ProxylessNAS, and **11**$\times$ faster than FBNet. In particular, we use an aggressive training schedule similar to the few-epochs schedule used in MnasNet to train the individual ConvNet samples [18]. Due to space limitations, we provide implementation details (*e.g.*, label smoothing, learning rates, λ value, *etc.*) in our project repository. Overall, we visualize the search efficiency of our method in Figure 4, where we show the progress of both

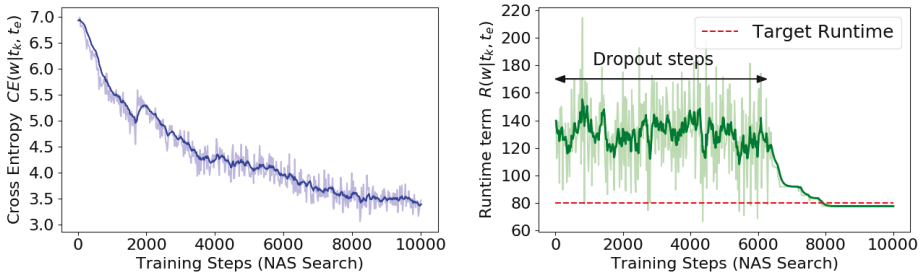

Fig. 4. *Single-Path NAS* search progress: Progress of both objective terms, *i.e.*, cross entropy CE (left) and runtime R (right) during NAS search.

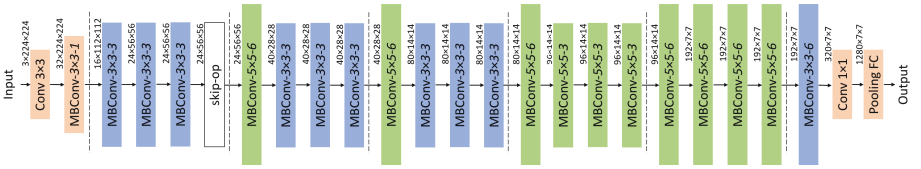

Fig. 5. Hardware-efficient ConvNet found by *Single-Path* NAS, with top-1 accuracy of **74.96%** on ImageNet and inference time of 79.48ms on Pixel 1 phone.

CE and R terms of Equation 8. Earlier during our search (first six epochs), we employ *dropout* across the different subsets of the kernel weights (Figure 4, right). Dropout is a common technique in NAS methods to prevent the supernet from learning as an ensemble. Unlike prior art that employs this technique over the separate paths of the *multi-path* supernet, we directly drop randomly the subsets of the superkernel in our *single-path* search space. We search for $\sim 10k$ **steps** (8 epochs with a batch size of 1024), which corresponds to total wall-clock time of **3.75 hours** on a TPUv2. In particular, given than a TPUv2 has 2 chips with 4 cores each, this corresponds to a total of 30 TPU-hours.

Visualization of *Single-Path NAS* ConvNet: Our derived ConvNet architecture is shown in Figure 5. Moreover, to illustrate how the searchable superkernels effectively capture NAS decisions across subsets of kernel weights, we plot the standard deviation of weight values in Figure 6 (shown in log-scale, with lighter colors indicating smaller values). Specifically, we compute the standard deviation of weights across the channel-dimension for all superkernels. For various layers shown in Figure 6 (per i-th ConvNet's layer from Figure 5), we observe that the *outer* $\mathbf{w}_{5\times5\setminus3\times3}$ "shells" reflect the NAS architectural choices: for layers where the entire $\mathbf{w}_{5\times5}$ is selected, the $\mathbf{w}_{5\times5\setminus3\times3}$ values drastically vary across the channels. On the contrary, for all layers where 3×3 convolution is selected, the *outer* shell values do not vary significantly.

Comparison with random search: We find surprising that mobile-efficient NAS methods lack a comparison against random search. To this end, we randomly sample ten ConvNets based on our design space; we employ sampling by rejection,

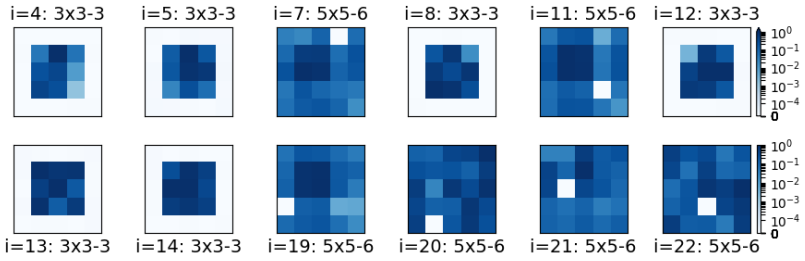

Fig. 6. Visualization of kernel-based architectural contributions. The *standard deviation* of **superkernel** values across the kernel channels is shown in log-scale, with lighter colors indicating smaller values.

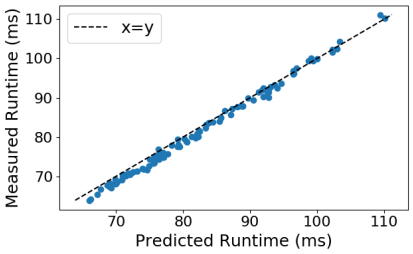

Fig. 7. The runtime model (Equation 10) is accurate, with an average prediction error of 1.76%.

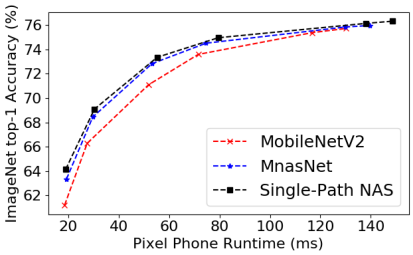

Fig. 8. *Single-Path* NAS outperforms MobileNetV2 and MnasNet across various channel size scales.

where we keep samples with predicted runtime from $75ms$ to $80ms$. The average accuracy and runtime of the random samples are reported in Table 1. We observe that, while random search does not outperform NAS methods, the overall accuracy is comparable to MobileNetV2. This highlights that the effectiveness of NAS methods heavily relies upon the properties of the MobileNetV2-based design space. Nonetheless, the search cost of random search is not representative: to avoid training all ten samples, we would follow a selection process similar to MnasNet, by training each sample for few epochs and picking the one with highest accuracy. Hence, the actual search cost for random search is not negligible, and for ≥ 10 samples it is in fact comparable to automated NAS methods.

Different channel size scaling: Next, we follow a typical analysis [3, 19], by rescaling the networks using a width multiplier [15]. As shown in Figure 8, we observe that our model consistently outperforms prior methods under varying runtime settings. For instance, Single-Path NAS with $79.48ms$ is $1.56\times$ faster than the MobileNetV2 scaled model of similar accuracy.

Runtime model: To train the runtime model, we record the runtime per layer (MBConv operations breakdown) by profiling ConvNets with different MBConv types, *i.e.*, we obtain the $R^i_{3\times3,3}$, $R^i_{3\times3,6}$, $R^i_{5\times5,3}$, and $R^i_{5\times5,6}$ runtime

Table 2. Searching across subsets of kernel weights: ConvNets with weight values trained over subsets of the kernels (3×3 as subset of 5×5) achieve performance (top-1 accuracy) similar to ConvNets with individually trained kernels.

Method	Top-1 Acc (%)	Top-5 Acc (%)
Baseline ConvNet - $\mathbf{w}_{3\times3}$ kernels	73.59	91.41
Baseline ConvNet - $\mathbf{w}_{5\times5}$ kernels	74.10	91.67
Single-Path ConvNet - inference w/ $\mathbf{w}_{3\times3}$ kernels	73.43	91.42
Single-Path ConvNet - inference w/ $\mathbf{w}_{3\times3} + \mathbf{w}_{5\times5\setminus3\times3}$ kernels	73.86	91.72

values per MBConv layer i (Equations 11-12). To evaluate the runtime-prediction accuracy of the model, we generate 100 randomly designed ConvNets and we measure their runtime on the device. As illustrated in Figure 7, our model can accurately predict the actual runtimes: the Root Mean Squared Error (RMSE) is $1.32ms$, which corresponds to an average 1.76% prediction error.

4.3 Ablation Study: Kernel-based Accuracy-Efficiency Trade-off

Single-Path NAS searches over subsets of the convolutional kernel weights. Hence, we conduct experiments to highlight how kernel-weight subsets can capture accuracy-efficiency trade-off effectively. To this end, we use the MobileNetV2 macro-architecture as a backbone (we maintain the location of stride-2 layers as default). As two baseline networks, we consider the default MobileNetV2 with MBConv-3×3-6 blocks (*i.e.*, $\mathbf{w}_{3\times3}$ kernels for all depthwise convolutions), and a network with MBConv-5×5-6 blocks (*i.e.*, $\mathbf{w}_{5\times5}$ kernels).

Next, to capture the subset-based training of weights during a *Single-Path* NAS search, we consider a *ConvNet* with MBConv-5×5-6 blocks, where we compute the loss of the model over two subsets, (i) the inner $\mathbf{w}_{3\times3}$ weights, and (ii) by also using the remaining $\mathbf{w}_{5\times5\setminus3\times3}$ weights. For each loss computed over these subsets, we accumulate back-propagated gradients and update the respective weights, *i.e.*, gradients are being applied separately to the inner and to the entire kernel per layer. We follow training steps similar to the "switchable" training across channels as in [21] (for the remaining training hyper-parameters we use the same setup as the default MnasNet). As shown in Table 2, we observe the final accuracy across the kernel granularity, *i.e.*, with the inner $\mathbf{w}_{3\times3}$ and the entire $\mathbf{w}_{5\times5} = \mathbf{w}_{3\times3} + \mathbf{w}_{5\times5\setminus3\times3}$ kernels, follows an accuracy change relative to ConvNets with individually trained kernels.

Such finding is significant in the context of NAS, since choosing over subsets of kernels can effectively capture the accuracy-runtime trade-offs similar to their individually trained counterparts. We therefore conjecture that our efficient superkernel-based design search can be flexibly adapted and benefit the guided search space exploration in other RL-based NAS methods. Beyond the NAS literature, our finding is closely related to Slimmable networks [21]. SlimmableNets limit however their analysis across the channel dimension, and our work is the first to study trade-offs across the NAS kernel dimension.

5 Conclusion

In this paper, we proposed *Single-Path NAS*, a NAS method that reduces the search cost for designing hardware-efficient ConvNets to **less than 4 hours**. The key idea is to revisit the one-shot **supernet** design space with a novel *single-path* view, by formulating the NAS problem as *finding which subset of kernel weights to use* in each ConvNet layer. *Single-Path NAS* achieved 74.96% top-1 accuracy on ImageNet with 79ms latency on a Pixel 1 phone, which is state-of-the-art accuracy with latency on-par with previous NAS methods ($\leq 80ms$). More importantly, we reduced the search cost of hardware-efficient NAS down to only **8 epochs** (30 TPU-hours), which is up to **5,000× faster** compared to prior work. **Impact beyond differentiable NAS**: While we used a differentiable NAS formulation, our novel design space encoding can be flexibly incorporated into other NAS methodologies. Hence, *Single-Path NAS* could enable future work that builds upon the efficiency of our *single-path*, one-shot design space for RL- or evolutionary-based NAS methods.

Acknowledgements

This research was supported in part by National Science Foundation CSR Grant No. 1815780 and National Science Foundation CCF Grant No. 1815899. Dimitrios Stamoulis also acknowledges support from the Qualcomm Innovation Fellowship (QIF) 2018 and the TensorFlow Research Cloud programs.

References

1. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: International Conference on Machine Learning. pp. 549–558 (2018)
2. Cai, E., Juan, D.C., Stamoulis, D., Marculescu, D.: Neuralpower: Predict and deploy energy-efficient convolutional neural networks. In: Asian Conference on Machine Learning. pp. 622–637 (2017)
3. Cai, H., Zhu, L., Han, S.: ProxylessNAS: Direct neural architecture search on target task and hardware. In: International Conference on Learning Representations (2019)
4. Chin, T.W., Zhang, C., Marculescu, D.: Layer-compensated pruning for resource-constrained convolutional neural networks. arXiv preprint arXiv:1810.00518 (2018)
5. Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., et al.: Chamnet: Towards efficient network design through platform-aware model adaptation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 11398–11407 (2019)
6. Ding, R., Liu, Z., Chin, T.W., Marculescu, D., Blanton, R.: Flightnns: Lightweight quantized deep neural networks for fast and accurate inference. In: 2019 Design Automation Conference (DAC) (2019)
7. Ding, R., Liu, Z., Shi, R., Marculescu, D., Blanton, R.: Lightnn: Filling the gap between conventional deep neural networks and binarized networks. In: Proceedings of the on Great Lakes Symposium on VLSI 2017. pp. 35–40. ACM (2017)

8. Dong, J.D., Cheng, A.C., Juan, D.C., Wei, W., Sun, M.: Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 517–531 (2018)
9. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
10. Hsu, C.H., Chang, S.H., Juan, D.C., Pan, J.Y., Chen, Y.T., Wei, W., Chang, S.C.: Monas: Multi-objective neural architecture search using reinforcement learning. arXiv preprint arXiv:1806.10332 (2018)
11. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). pp. 1–12. IEEE (2017)
12. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. In: International Conference on Learning Representations (2018)
13. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameter sharing. In: International Conference on Machine Learning. pp. 4092–4101 (2018)
14. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. arXiv preprint arXiv:1802.01548 (2018)
15. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018)
16. Stamoulis, D., Cai, E., Juan, D.C., Marculescu, D.: Hyperpower: Power-and memory-constrained hyper-parameter optimization for neural networks. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE (2018)
17. Stamoulis, D., Chin, T.W.R., Prakash, A.K., Fang, H., Sajja, S., Bognar, M., Marculescu, D.: Designing adaptive neural networks for energy-constrained image classification. In: Proceedings of the International Conference on Computer-Aided Design. ACM (2018)
18. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2019)
19. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
20. Xie, S., Zheng, H., Liu, C., Lin, L.: Snas: stochastic neural architecture search. In: International Conference on Learning Representations (2019)
21. Yu, J., Yang, L., Xu, N., Yang, J., Huang, T.: Slimmable neural networks. In: International Conference on Learning Representations (2019)
22. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6848–6856 (2018)
23. Zhou, Y., Ebrahimi, S., Arık, S.Ö., Yu, H., Liu, H., Diamos, G.: Resource-efficient neural architect. arXiv preprint arXiv:1806.07912 (2018)
24. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: International Conference on Machine Learning (2017)
25. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)