Single-Path Mobile AutoML: Efficient ConvNet Design and NAS Hyperparameter Optimization

Dimitrios Stamoulis, *Student Member, IEEE*, Ruizhou Ding, *Student Member, IEEE*, Di Wang, *Member, IEEE*, Dimitrios Lymberopoulos, *Member, IEEE*, Bodhi Priyantha, *Member, IEEE*, Jie Liu, *Fellow, IEEE*, and Diana Marculescu, *Fellow, IEEE*

***Abstract*—Can we reduce the search cost of Neural Architecture Search (NAS) from days down to only a few hours? NAS methods automate the design of Convolutional Networks (ConvNets) under hardware constraints and they have emerged as key components of AutoML frameworks. However, the NAS problem remains challenging due to the combinatorially large design space and the significant search time (at least 200 GPU-hours). In this article, we alleviate the NAS search cost down to *less than 3 hours*, while achieving state-of-the-art image classification results under mobile latency constraints. We propose a novel differentiable NAS formulation, namely *Single-Path NAS*, that uses *one* single-path over-parameterized ConvNet to encode all architectural decisions based on shared convolutional kernel parameters, hence drastically decreasing the search overhead. *Single-Path NAS* achieves state-of-the-art top-1 ImageNet accuracy (75.62%), hence outperforming existing mobile NAS methods in similar latency settings (∼80 ms). In particular, we enhance the accuracy-runtime trade-off in differentiable NAS by treating the Squeeze-and-Excitation path as a fully searchable operation with our novel *single-path* encoding. Our method has an overall cost of only *8 epochs* (24 TPU-hours), which is up to *5,000× faster* compared to prior work. Moreover, we study how different NAS formulation choices affect the performance of the designed ConvNets. Furthermore, we exploit the efficiency of our method to answer an interesting question: instead of empirically tuning the hyperparameters of the NAS solver (as in prior work), can we automatically find the hyperparameter values that yield the desired accuracy-runtime trade-off (e.g., target runtime for different platforms)? We view our extensive experimental results as a valuable exploration for NAS-based cloud AutoML services, and we open-source our entire codebase at: https://github.com/dstamoulis/single-path-nas.**

***Index Terms*—Neural architecture search (NAS), hardware-aware convnets, ConvNets, AutoML.**

Manuscript received July 1, 2019; revised January 24, 2020; accepted January 27, 2020. Date of publication February 3, 2020; date of current version August 10, 2020. This work was supported in part by National Science Foundation CSR Grant 1815780 and in part by the National Science Foundation CCF Grant 1815899. The associate editor coordinating the review of this manuscript and approving it for publication was Mr. Werner Bailer. *(Corresponding author: Dimitrios Stamoulis.)*

D. Stamoulis and R. Ding are with the Department of ECE, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: dstamoul@andrew.cmu.edu; rding@andrew.cmu.edu).

D. Marculescu is with the Department of ECE, University of Texas at Austin, Austin, TX 78712 USA and holds a courtesy adjunct position with the Department of ECE, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: dianam@cmu.edu).

D. Wang, D. Lymberopoulos, and B. Priyantha are with Microsoft, Redmond, WA 98052 USA (e-mail: wangdi@microsoft.com; dlymper@microsoft.com; bodhip@microsoft.com).

J. Liu is with the Harbin Institute of Technology, Harbin 150001, China (e-mail: jieliu@hit.edu.cn).

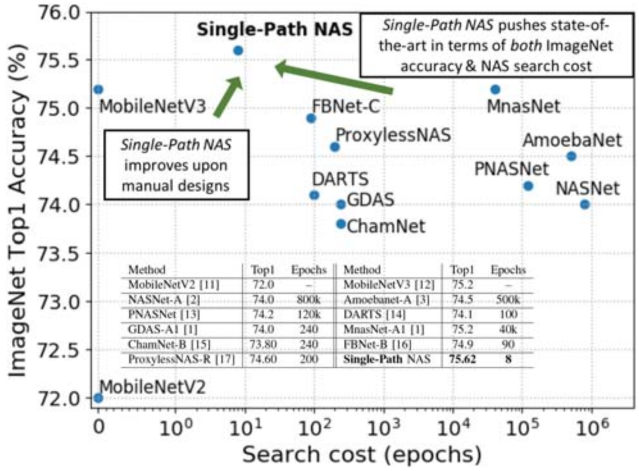
Digital Object Identifier 10.1109/JSTSP.2020.2971421

I. Introduction

Is it possible to automatically design the Convolutional Network (ConvNet) with highest classification accuracy that satisfies the inference latency constraints of a mobile phone? Can we have a push-button solution that automatically finds such design within only a few hours?" ConvNets have been traditionally designed by human experts in a painstaking and expensive process. AutoML approaches, and Neural Architecture Search (NAS) methods in particular, present a promising path for alleviating the engineering costs that are intrinsic to the manual ConvNet design, by automating the tuning of DNN hyperparameters (e.g., the number of layers, the type of operations per layer, etc.).

NAS approaches formulate the design of hardware-efficient ConvNets as a *multi-objective hyperparameter optimization* problem [2]. In fact, we are witnessing a proliferation of novel AutoML approaches, with NAS formulations spanning many different optimization methodologies, such as reinforcement learning [3], evolutionary algorithms [4], and Bayesian optimization [5]. More importantly, NAS-based AutoML has drawn significant interest from industry, as demonstrated by the immense amount of computational resources used in NAS research [3], [4], [6] and by the plethora of commercial cloud-based AutoML services and frameworks [7]–[11]. Overall, AutoML is a research topic of paramount importance, since "push-button" solutions such as NAS frameworks are expected to significantly advance numerous deep learning (DL) applications, especially when designing ConvNets for computer vision tasks under the constraints of mobile devices [2].

Despite the recent breakthroughs, NAS remains an intrinsically costly optimization problem due to the combinatorially large search space: e.g., for a mobile-efficient ConvNet with 22 layers, choosing among five candidate operations yields $5^{22} \approx 2.3 \times 10^{15}$ possible ConvNet architectures. NAS literature has seen a shift towards one-shot differentiable formulations [12]–[14] 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 Fig. 2 (left). Next, NAS formulations solve for the (distributions of) paths of the *multi-path* supernet that yield the optimal architecture.



Method	Top1	Epochs	Method	Top1	Epochs
MobileNetV2 [11]	72.0	–	MobileNetV3 [12]	75.2	–
NASNet-A [2]	74.0	800k	Amoebanet-A [3]	74.5	500k
PNASNet [13]	74.2	120k	DARTS [14]	74.1	100
GDAS-A1 [1]	74.0	240	MnasNet-A1 [1]	75.2	40k
ChamNet-B [15]	73.80	240	FBNet-B [16]	74.9	90
ProxylessNAS-R [17]	74.60	200	**Single-Path** NAS	**75.62**	**8**

Fig. 1. **Search Cost** *vs.* **ImageNet Accuracy:** Our *Single-Path NAS* outperforms Mobile NAS methods in both search cost and ImageNet accuracy, while also improving upon manually-designed MobileNets [1]. In particular, Single-Path NAS achieves new state-of-the-art 75.62% top-1 accuracy compared to methods designing for similar latency setting (∼80 ms). We report results from Mobile NAS and the "Mobile setting" of NAS literature (x-axis is shown in `symlog`-scale). Detailed discussion follows in Table I.

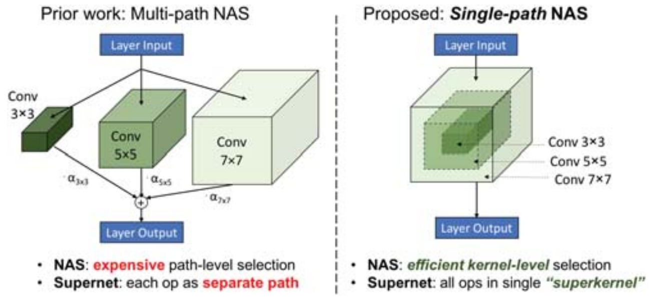

Prior work: Multi-path NAS
Layer Input
Conv 3×3
Conv 5×5
Conv 7×7
α_{3x3}
α_{5x5}
α_{7x7}
Layer Output
• NAS: expensive path-level selection
• Supernet: each op as separate path
Proposed: Single-path NAS
Layer Input
Conv 3×3
Conv 5×5
Conv 7×7
Layer Output
• NAS: efficient kernel-level selection
• Supernet: all ops in single "superkernel"

Fig. 2. *Single-Path NAS* directly optimizes for the subset of convolution weights of an over-parameterized superkernel in each ConvNet layer (right). Our **novel view** of the design space eliminates the need for maintaining separate paths for each candidate operation, as in previous *multi-path* approaches (left).

As expected, naively branching out all paths is intrinsically inefficient, since the number of trainable parameters that need to be maintained and updated during the search grow linearly with respect to the number of candidate operations per layer [6]. To tame the memory explosion introduced by the *multi-path* supernet, current methods employ creative "workaround" solutions: e.g., searching on a proxy dataset [15], or employing a memory-wise scheme with only a subset of paths being updated during the search [16]. However, these techniques remain considerably costly, with an overall computational demand of hundreds of GPU-hours.

In this paper, we propose *Single-Path NAS*, a novel NAS method for designing hardware-efficient ConvNets in **less than 3 hours**. Our **key insight** is illustrated in Fig. 2 (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 searchable superkernels (i.e., a single path) for each layer of the one-shot NAS supernet. *Single-Path NAS* achieves 75.62% top-1 accuracy on ImageNet with ∼80 ms latency on a Pixel 1, i.e., a +0.4% improvement over the current best hardware-aware NAS [2] and manually-designed [1] ConvNets in similar latency settings. The overall search cost is only 8 epochs, i.e., 2.45 hours on TPU-v3 (24 TPU-hours), up to **5,000× faster** compared to prior work. Our contributions are as follows:

1) Single-path differentiable NAS: We propose a novel *single-path* encoding of the one-shot differentiable NAS problem. Moreover, while recent work investigates the use of Squeeze-and-Excitation [17] (SE) as a binary NAS decision, we are first to treat the SE path as a fully searchable operation. To the best of our knowledge, this is the first single-path, differentiable NAS approach with SE paths, and our fully searchable treatment improves the accuracy-runtime trade-off compared to manually-tuned SE paths [1].

2) NAS hyperparameter optimization: To our knowledge, our work is the first to formulate the hyperparameter tuning of a differentiable NAS solver as a hyperparameter optimization problem itself, aiming to answer the question "*instead of empirically tuning, can we automatically find the trade-off hyperparameter in differentiable NAS given a target runtime?*"

II. Related Work

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 [19]. 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 [20], [21]. Pruning [22]–[24] and quantization [25]–[27] methods share the same goal to improve the ConvNet efficiency.

NAS methods aim to automate the design of ConvNets based on reinforcement learning (RL), evolutionary algorithms, or gradient-based formulations [3], [4], [12], [13], [28]. 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 [3]. Nonetheless, training the controller over different architectures makes the search costly. An increasing number of recent methods motivate the need for alleviating the NAS search cost [29].

Hardware-aware NAS: Earlier NAS methods focused on maximizing accuracy under FLOPs constraints [14], [30], but low FLOP count does not necessarily translate to hardware efficiency [31], [32]. More recent methods incorporate hardware terms (e.g., runtime, power) into cell-based NAS formulations [33], [34], but cell-based implementations are not hardware friendly [15]. Breaking away from cell-based assumptions in the search space encoding, Mnasnet searches over a generalized MobileNetV2-based design space [2].

Recent NAS literature has seen a shift towards one-shot NAS formulations [12]–[14]. Differentiable NAS in particular has gained increased popularity and has achieved state-of-the-art results [35]. 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 grow linearly with respect to the number of different convolutional operations per layer, resulting in memory explosion [6], [16].

To this end, state-of-the-art approaches employ different "workaround" solutions. FBNet [15] 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 supermodel needs to be maintained in memory during search, hence the efficiency is limited due to inevitable use of smaller batch sizes. ProxylessNAS [16] employs a memory-wise scheme, where only a set of paths is updated during search. However, such implementation 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 in terms of memory cost is kept to the same level at any step, the way that *multi-path*-based methods traverse the design space remains inefficient.

While concurrent methods consider relaxed convolution formulations with insight similar to our work [18], [36]–[38], they either use design spaces and objectives that have been shown to be hardware inefficient (e.g., cell-based space, FLOP count), or they optimize over a subset of our design space. In our work, we optimize over multiple searchable kernels per layer and we simultaneously search across several NAS decisions, i.e., kernel sizes, channels dimensions, expansion ratio, or Squeeze-and-Excitation [17] ratio dimensions.

Searching for Squeeze-and-Excitation [17]: Recently, MobileNetV3 explored various design choices on top of the MobileNetV2 backbone, showing that augmenting the mobile inverted bottleneck convolution (MBConv) layers with a Squeeze-and-Excitation [17] (SE) path can improve the overall accuracy [1]. Recent RL-based mobile NAS has adapted this finding by adding the SE path into their search space [2], but by limiting however their exploration to a binary decision of using SE or not. Instead, in our work we are the *first* to treat the SE path as fully searchable (i.e., searching over various SE ratios), with a novel outcome. As discussed in our results section, larger SE ratios further improve the overall performance by yielding a better DNN accuracy-trade-off. Our AutoML-designed DNN achieves a new state-of-the-art ImageNet accuracy compared to methods designing for similar latency settings ($\sim$80 ms).

Hyperparameter optimization beyond DL: Last, we note that the various aforementioned NAS approaches, including our proposed methodology, share inspiration and complexity challenges similar to various hyperparameter optimization methods in other CS domains, e.g., evolutionary and simulated annealing algorithms for compiler optimization [39], [40] and *one-shot* decision theory in operation research [41].

III. Proposed Method: *Single-Path* NAS

In this Section, we present our proposed method. First, we discuss our novel *single-path* view (Subsection III-A) of the search space. Next, we encode the NAS problem as finding the subset of convolution weights over the *over-parameterized* superkernel (Subsection III-B), and we discuss how it compares to existing *multi-path*-based NAS (Subsection III-C). 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 III-D).

A. Mobile ConvNets Search Space: A Novel View

Search Space: As illustrated in Fig. 3 (left), our method builds upon a fixed "backbone" [16] which follows the MobileNetV2 design [21] and which has been successfully considered by other differentiable NAS approaches [18]. Specifically, in this 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 [15], i.e., we use seven blocks with up to four layers each. Each layer of these blocks is a mobile inverted bottleneck convolution MBConv [21] micro-architecture. In particular, an MBConv layer consists of a point-wise (1×1) convolution, a $k \times k$ depthwise convolution, a Squeeze-and-Excitation (SE) block [17], and a linear 1×1 convolution. Unless the layer has a stride value of two, a skip path is introduced to provide a residual connection from input to output. The goal of NAS is to automatically identify the type of each MBConv layer in the ConvNet design.

Our search space consists of 13 candidate layer types, with the layer-wise choices listed in Fig. 3. In particular, each MBConv layer is parameterized by the following choices: (i) the kernel size of the depthwise convolution $k \times k$, (ii) the expansion ratio e, i.e., the ratio between the output and input of the first 1×1 convolution, and (iii) the Squeeze-and-Excitation [17] (SE) ratio se, i.e., the ratio between the number of channels in the intermediate convolution and the input of the SE path. It is worth observing that, unlike prior NAS work, in our search space we treat the SE-path as fully searchable (i.e., searching over various SE ratios). Furthermore, NAS 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. This NAS choice effectively corresponds to reducing the depth of the network. Based on this parameterization, we denote each MBConv as MBConv-$k \times k$-e-se.

Novel view of design space: We build upon the *key observation* that different candidate convolutional operations in NAS can be viewed as subsets of the weights of over-parameterized *superkernels*. This observation allows us to view the NAS combinatorial problem as *finding which subset of kernel weights to use in each MBConv layer*, while sharing the kernel parameters across different MBConv architectural options. As shown in

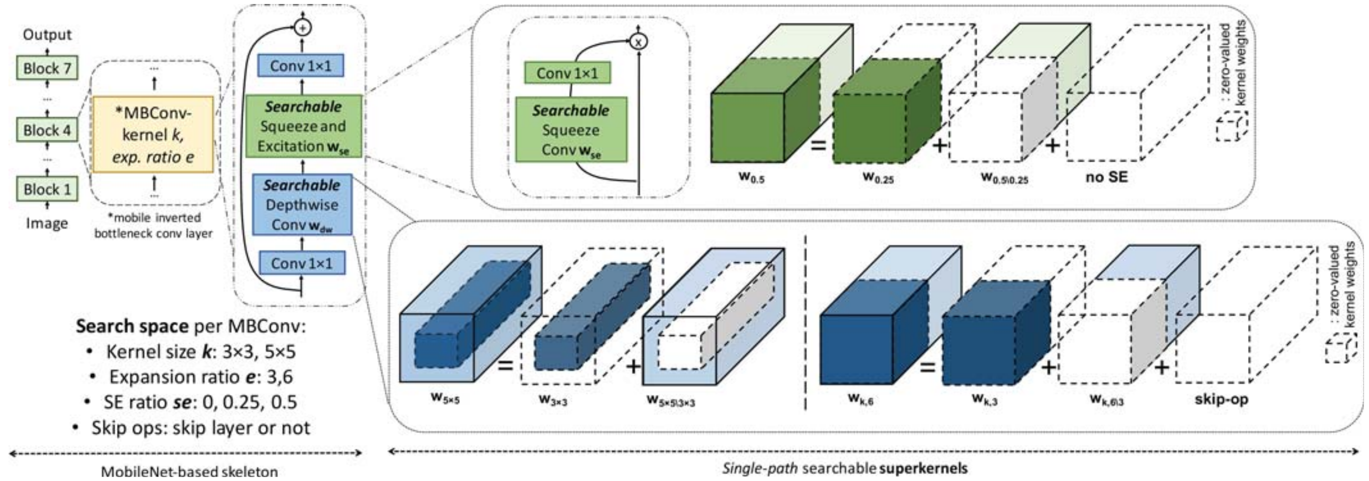

Output
Block 7
Block 4
Block 1
Image
*MBConv-kernel k, exp. ratio e
*mobile inverted bottleneck conv layer
Conv 1×1
Searchable Squeeze and Excitation w_se
Searchable Depthwise Conv w_dw
Conv 1×1
Searchable Squeeze Conv w_se
w_0.5
w_0.25
w_0.5\0.25
no SE
: zero-valued kernel weights
w_5×5
w_3×3
w_5×5\3×3
w_k,6
w_k,3
w_k,6\3
skip-op
Search space per MBConv:
• Kernel size k: 3×3, 5×5
• Expansion ratio e: 3,6
• SE ratio se: 0, 0.25, 0.5
• Skip ops: skip layer or not
MobileNet-based skeleton
Single-path searchable superkernels

Fig. 3. *Single-Path* NAS builds upon *hierarchical* MobileNetV2-like search spaces [2], [18] to identify the mobile inverted bottleneck convolution (MBConv) per layer (left). 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 two per-layer **searchable superkernels** that encode all MBConv types, i.e., the different kernel size (bottom, middle) and expansion ratio (bottom, right) values on the searchable *depthwise* superkernel, and the different Squeeze-and-Excitation [17] (SE) ratios over the searchable *squeeze* superkernel (top, right). That is, instead of treating the SE-path as a binary NAS decision (use it with fixed SE-ratio or not, as in [2]), we treat the SE path as a fully searchable operation with our *single-path* encoding. We show that this search space enhancement further improves the accuracy-runtime trade-off.

Fig. 3, we encode all candidate NAS operations to two searchable superkernels (i.e., a *single path*), for each layer of the one-shot NAS supernet.

B. Proposed Methodology: Single-Path NAS Formulation

Kernel size: To simplify notation and 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 Fig. 3 (bottom), 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 $\mathbf{w}_{5\times5\backslash3\times3}$. 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., $\mathbf{w}_{5\times5} = \mathbf{w}_{3\times3} + \mathbf{w}_{5\times5\backslash3\times3}$.

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

$$\mathbf{w}_k = \mathbf{w}_{3\times3} + \mathbb{1}(\text{use } 5\times5) \cdot \mathbf{w}_{5\times5\backslash3\times3} \tag{1}$$

where $\mathbb{1}(\cdot)$ is the indicator function that encodes the architectural NAS choice, i.e., if $\mathbb{1}(\cdot) = 1$ then $\mathbf{w}_k = \mathbf{w}_{3\times3} + \mathbf{w}_{5\times5\backslash3\times3} = \mathbf{w}_{5\times5}$, else $\mathbb{1}(\cdot) = 0$ then $\mathbf{w}_k = \mathbf{w}_{3\times3}$.

Trainable encoding: 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 [42], [43] 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 condition:

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

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., $\hat{g}(x, t) = \sigma(x > t)$.

Expansion ratio and skip-op: Since the result of the kernel-based NAS decision $\mathbf{w}_k$ (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 Fig. 3 (bottom, 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 the searchable kernel of the depthwise convolution, we write:

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

SE ratio: Next, we extend the superkernel-based definition to encode the Squeeze-and-Excitation [17] (SE) ratio se decision. In particular, we observe that choosing SE ratio (3) effectively means to choose the number of channels of the *squeeze* convolution stage in the SE path. Hence, as shown in Fig. 3 (top, right), we replace the convolution kernel of the *squeeze* convolution with a searchable superkernel, where the largest number of channels corresponds to the largest candidate se value, i.e., $se = 0.5$. By following an intuition similar to (3), we observe that "zero-ing out" the second half of the *squeeze* convolution corresponds to using $se = 0.25$, while "zero-ing out" the entire kernel corresponds to not using a SE path ($se = 0$). We therefore write:

$$\begin{aligned}\mathbf{w}_{se} &= \mathbb{1}(\|\mathbf{w}_{0.25}\|^2 > t_{se=0.25}) \\ &\quad \cdot (\mathbf{w}_{0.25} + \mathbb{1}(\|\mathbf{w}_{0.5\backslash 0.25}\|^2 > t_{se=0.5}) \cdot \mathbf{w}_{0.5\backslash 0.25})\end{aligned} \tag{4}$$

Searchable MBConvs: Each MBConv uses 1×1 convolutions for the point-wise and linear stages, while the kernel-size decisions affect only the $k \times k$ depthwise convolution (Fig. 3). Thus, we use our *searchable* depthwise kernel $\mathbf{w}_{dw}$ 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 encode the expansion ratio e into $\mathbf{w}_{dw}$ as well. That is, we set the point-wise 1×1 output to the maximum candidate expansion ratio, and we instead solve for which of them not to "zero" out at the depthwise stage. In other words, we also encode the NAS decision for the expansion ratio at $\mathbf{w}_{dw}$. Similarly, we can encode the SE ratio se by deciding which the channels of the 1×1 *squeeze* convolution to "zero" out. To this end, we can simply replace the *squeeze* kernel with the *searchable* kernel $\mathbf{w}_{se}$ to directly search for the SE-ratio across the SE path (Fig. 3, top right).

Overall, our *single-path* formulation can sufficiently capture any MBConv type (e.g., MBConv-3 × 3-6-0.25, MBConv-5 × 5-3-0.5, etc.) in the design space (Fig. 3). For input $\mathbf{x}$, the output of the i-th MBConv layer of the network is:

$$o^i(\mathbf{x}) = \text{conv}(\mathbf{x}, \mathbf{w}^i | t^i_{k=5}, t^i_{e=6}, t^i_{e=3}, t^i_{se=0.5}, t^i_{se=0.25}) \tag{5}$$

C. Single-Path vs. Existing Multi-Path Assumptions

We briefly illustrate how our *single-path* formulation compares to multi-path NAS approaches. In existing methods [12], [15], [16], 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:

$$\begin{aligned}o^i_{multi-path}(\mathbf{x}) &= \sum_{j=1}^{N} \alpha^{i,j} \cdot o^{i,j}(\mathbf{x}) \\ &= \alpha^{i,0} \cdot \text{conv}(\mathbf{x}, \mathbf{w}^{i,0}_{3\times 3}) + \cdots + \alpha^{i,N} \\ &\quad \cdot \text{conv}(\mathbf{x}, \mathbf{w}^{i,N}_{5\times 5})\end{aligned} \tag{6}$$

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* (5):

$$\begin{aligned}o^i{}_{single-path}(\mathbf{x}) &= o^i(\mathbf{x}) \\ &= \text{conv}(\mathbf{x}, \mathbf{w}^i | t^i_{k=5}, t^i_{e=6}, t^i_{e=3}, t^i_{se=0.5}, t^i_{se=0.25})\end{aligned} \tag{7}$$

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 $\mathcal{L}(\alpha, w_\alpha)$:

$$\min_{\alpha} \min_{w_\alpha} \mathcal{L}(\alpha, w_\alpha) \tag{8}$$

However, solving (8) 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* $\mathbf{w}$ *of a single-path, compact neural network*. Formally, the NAS problem becomes:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w} | \mathbf{t}_k, \mathbf{t}_e, \mathbf{t}_{se}) \tag{9}$$

Efficiency of Single-Path NAS: Unlike the bi-level optimization problem in prior work, solving our NAS formulation in (9) 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 V.

D. Hardware-Aware NAS With Differentiable Runtime Loss

To design hardware-efficient ConvNets, the differentiable objective in (9) should reflect both the accuracy of the searched architecture and its inference latency on the target hardware. Hence, we use a latency-aware formulation [15]:

$$\begin{aligned}\mathcal{L}(\mathbf{w} | \mathbf{t}_k, \mathbf{t}_e, \mathbf{t}_{se}) &= CE(\mathbf{w} | \mathbf{t}_k, \mathbf{t}_e, \mathbf{t}_{se}) \\ &\quad + \lambda \cdot \log(R(\mathbf{w} | \mathbf{t}_k, \mathbf{t}_e, \mathbf{t}_{se}))\end{aligned} \tag{10}$$

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.

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 [15], [16], [44]:

$$R(\mathbf{w}|\mathbf{t}_k, \mathbf{t}_e) = \sum_i R^i(\mathbf{w}^i|\mathbf{t}^i_k, \mathbf{t}^i_e, \mathbf{t}^i_{se}) \qquad (11)$$

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 (3). First, we express the runtime of each layer i as a function of the expansion ratio decision:

$$\begin{aligned} R^i_e = \mathbb{1}(\|\mathbf{w}_{k,3}\|^2 > t_{e=3}) \cdot (R^i_{5\times5,3} \\ + \mathbb{1}(\|\mathbf{w}_{k,6\backslash3}\|^2 > t_{e=6}) \cdot (R^i_{5\times5,6} - R^i_{5\times5,3})) \end{aligned} \qquad (12)$$

By incorporating the kernel size decision, the runtime based on the kernel k and expansion ratio decision e is:

$$\begin{aligned} R^i_{k,e} = \frac{R^i_{3\times3,6}}{R^i_{5\times5,6}} \cdot R^i_e + R^i_e \cdot \left(1 - \frac{R^i_{3\times3,6}}{R^i_{5\times5,6}}\right) \\ \cdot \mathbb{1}\left(\|\mathbf{w}_{5\times5\backslash3\times3}\|^2 > t_{k=5}\right) \end{aligned} \qquad (13)$$

Next, we capture the effect that the SE path has on the runtime. We denote the total runtime of the i-th MBConv layer with kernel size k, expansion ratio e, and SE ratios 0.25 or 0.5 as $R^i_{k\times k,e,se=0.25}$ and $R^i_{k\times k,e,se=0.5}$, respectively. Similarly, we denote the runtime of the MBConv layer without a SE path as $R^i_{k\times k,e,se=0}$. For notation clarity, let us define the relative increase in runtime due to the addition of the SE path, compared to the runtime without the SE path, as scaling factor:

$$s^i_{k,e,se} = R^i_{k\times k,e,se}/R^i_{k\times k,e,se=0} \qquad (14)$$

Based on our runtime profiling (Section IV), we make two observations: (i) due to the relatively smaller size of the *squeeze* convolution compared to the $k \times k$ convolution of the main path, the difference in the relative runtime increase from using either SE ratios is negligible, i.e., $s^i_{k,e,0.25} \approx s^i_{k,e,0.5}$. Next, (ii) the relative ratio of the runtimes with and without the SE path differs based on the type of the main MBConv path. Thus, we express the overall runtime scaling as function of the kernel and the expansion ratio choices:

$$\begin{aligned} s^i_{k,e=6,0.25} = \mathbb{1}(\|\mathbf{w}_{5\times5\backslash3\times3}\|^2 > t_{k=5}) \cdot s^i_{k=5,e=6,0.25} \\ + (1 - \mathbb{1}(\|\mathbf{w}_{5\times5\backslash3\times3}\|^2 > t_{k=5}) \cdot s^i_{k=3,e=6,0.25} \end{aligned} \qquad (15)$$

$$\begin{aligned} s^i_{k,e=3,0.25} = \mathbb{1}(\|\mathbf{w}_{5\times5\backslash3\times3}\|^2 > t_{k=5}) \cdot s^i_{k=5,e=3,0.25} \\ + (1 - \mathbb{1}(\|\mathbf{w}_{5\times5\backslash3\times3}\|^2 > t_{k=5}) \cdot s^i_{k=3,e=3,0.25} \end{aligned} \qquad (16)$$

Hence, overall we have:

$$\begin{aligned} R^i = & \left(1 - \mathbb{1}(\|\mathbf{w}_{0.5\backslash0.25}\|^2 > t_{se=0.25})\right) \cdot R^i_{k,e} \\ & + \mathbb{1}(\|\mathbf{w}_{0.5\backslash0.25}\|^2 > t_{se=0.25}) \cdot \\ & \Big\{\mathbb{1}(\|\mathbf{w}_{k,6\backslash3}\|^2 > t_{e=6}) \cdot s^i_{k,e=6,0.25} \\ & + \left(1 - \mathbb{1}(\|\mathbf{w}_{k,6\backslash3}\|^2 > t_{e=6})\right) \cdot s^i_{k,e=3,0.25}\Big\} \cdot R^i_{k,e} \end{aligned} \qquad (17)$$

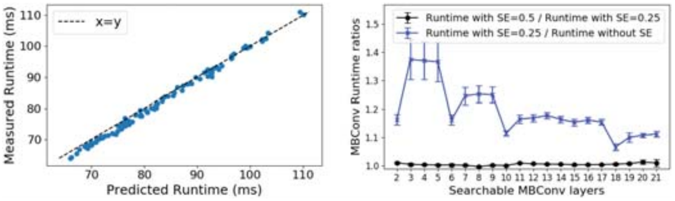


Fig. 4. Runtime profiling: (Left) The runtime model (11) is accurate, with an average prediction error of 1.76%. (Right) Runtime results with SE ratios 0, 0.25, and 0.5 show that allowing for SE ratios larger than 0.25 (i.e., 0.5 SE ratio) provides a better accuracy-runtime trade-off, since the *squeeze* step is enhanced with more channels with negligible runtime overhead ($s^i_{k,e,0.25} \approx s^i_{k,e,0.5}$), especially for the deeper layers (MBConv 18-21).

As in (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.

IV. Experimental Setup

We use *Single-Path NAS* to design ConvNets for image classification on ImageNet [46]. 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 80 ms [2], [16].

Implementation and deployment: We implement our NAS framework in TensorFlow (`TF` version 1.12). During both search and training stages, we use TPUs (version 3) [47]. To this end, we build on top of the `TPUEstimator` classes following the TPU-related documentation of the MnasNet repository [11]. 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`) [48] that supports profiling for `tflite` models with detailed per-layer runtime breakdown.

Runtime model: To train the inference runtime model, we record the runtime per layer (MBConv operations breakdown) by profiling ConvNets with all different MBConv types (12)–(17). To evaluate the runtime-prediction accuracy of the model, we generate 100 randomly designed ConvNets (with $se = 0$) and we measure their runtime on the device. As illustrated in Fig. 4 (left), our predictive model is accurate: the Root Mean Squared Error (RMSE) is 1.32 ms, which corresponds to an average 1.76% prediction error.

Superkernels implementation: 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 (2)–(3). Our custom layer also returns the effective runtime of the layer (12)–(17). We document our

TABLE I
Single-Path NAS Achieves State-of-the-art Image Classification Accuracy (%) on ImageNet for Similar on-Device Latency Setting Compared to Previous NAS Methods (~80 ms on Pixel 1), With up to 5,000× Reduced Search Cost in Terms of Number of Epochs

Method[1]	Top-1 Acc (%)	Top-5 Acc (%)	Runtime (ms)	Search Cost (epochs)
MobileNetV2 [11]	72.00	91.00	75.00	-
MobileNetV2 (our impl.)	73.59	91.41	73.57	
MobileNetV3 [12]	75.20	–	78.00†	
Random search	73.78 ± 0.85	91.42 ± 0.56	77.31 ± 0.9 ms	-
MnasNet-B1 [1]	74.00	91.80	76.00	40,000
MnasNet-B1 (our impl.)	74.61	91.95	74.65	
MnasNet-A1 [1]	75.20	92.50	78.00	
MnasNet-B1 (92) [1]	74.79	92.05	92.00	
ChamNet-B [15]	73.80	–	–	240‡
ProxylessNAS-R [17]	74.60	92.20	78.00	200*
ProxylessNAS-R (our impl.)	74.65	92.18	77.48	
FBNet-B [16]	74.1	-	-	90§
FBNet-B (our impl.)	73.70	91.51	78.33	
Single-Path NAS (**proposed**)	**75.62**	**92.61**	81.84	**8 (2.45 hours)**

implementation in our project GitHub repository: https://github.com/dstamoulis/single-path-nas, with detailed steps on how to reproduce the results.

V. 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 around ~80 ms. We train the derived *Single-Path* NAS model for 350 epochs, following the MnasNet training schedule [2]. We compare our method with mobile ConvNets designed by human experts and state-of-the-art NAS methods in Table I, in terms of classification accuracy, search cost and hardware efficiency (inference latency on Pixel 1). To ensure a fair comparison, 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 [15], [45], since the authors use these ConvNets to compare against the MnasNet model). Finally, 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 I shows that our *Single-Path* NAS achieves top-1 accuracy of 75.62%, which is the new state-of-the-art ImageNet accuracy among hardware-efficient NAS methods. More specifically, our method achieves **better** top-1 accuracy than ProxylessNAS by almost 1%, while maintaining on par target latency of ~80 ms on the same target platform. Overall, we note that *Single-Path* NAS outperforms prior NAS methods in this mobile latency range [2], [15], [45], as well as manually designed models (MobileNetV2 [21]) and ConvNets that combine both AutoML and manual-design expertise (MobileNetV3 [1]), e.g., better than MnasNet-A1 (+0.42%), FBNet-B (+1.52%), and MobileNetV3 (+0.42%).

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 8 k sampled models, each trained for 5 epochs, for a total of 40 k 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 1.2 k epochs. ProxylessNAS reports 200× 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× more compared to the steps per epoch in our method.[1]

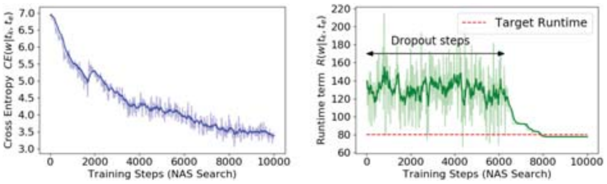


Fig. 5. *Single-Path NAS* search progress: Progress of both objective terms, i.e., cross entropy CE (left) and runtime R (right) during NAS search.

In comparison, *Single-Path NAS* has a total cost of eight epochs, which is $\mathbf{5,000}\times$ 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 [2]. Overall, we visualize the search efficiency of our method in Fig. 5, where we show the progress of both CE and R terms of (9). Earlier during our search (first six epochs), we employ *dropout* across the different subsets of the kernel weights

[1] Table I: *The search cost in epochs is estimated based on the claim that ProxylessNAS is 200× faster than MnasNet, following the *one-shot* solver setup reported in the paper [16]. ‡ChamNet does not detail the model derived under runtime constraints [45] so we cannot retrain or measure the latency. † For MobileNetV3, we report the version that matches the MnasNet space backbone, since some additional manual enhancements in the network head are directly applicable to all other ConvNets considered. Overall, the reported epochs correspond to the best search cost following the search setup reported per method.§ For FBNet, besides the discussion about the batch size to fit the TPU, we consider the solver optimizer setup as reported in the paper [15].

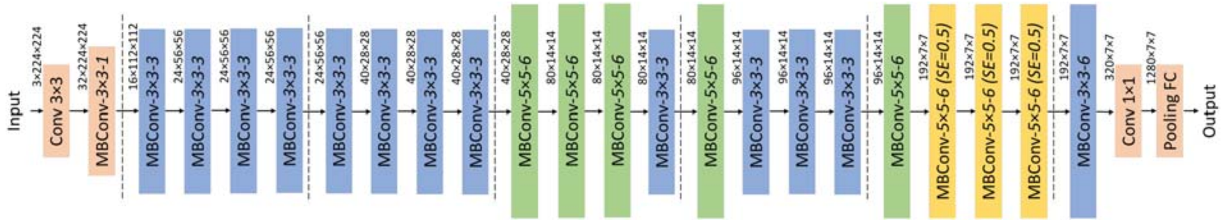


Fig. 6. Hardware-efficient ConvNet found by *Single-Path* NAS, with top-1 accuracy of **75.62%** on ImageNet and inference time of 81.84 ms on Pixel 1 phone. Compared to our previous NAS result without SE [18], some of the earlier 5×5 MBConvs have been replaced with smaller $3 \times 3 - 3$ MBConvs, and instead *Single-Path NAS* selects SE paths with SE ratio of $se = 0.5$ in the last layers. Overall, our NAS enhancement with fully searchable SE improves the accuracy-runtime trade-off of mobile ConvNets.

(Fig. 5, 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 $\sim 10\ k$ steps (8 epochs with a batch size of 1024), which corresponds to total wall-clock time of **2.45 hours** on a TPUv3 (i.e., 24 TPU-hours).

Enhancing accuracy-runtime trade-off: Our derived ConvNet architecture is shown in Fig. 6. Our goal is to understand the better accuracy-runtime trade-off achieved by the searchable SE. To this end, a comparison against the earlier version of our work [18], [49] without SE can give insightful observations. In particular, we observe that, compared to the ConvNet previously derived in [18], some of the earlier MBConv types with either 5×5 kernels or expansion ration 6, have been replaced with smaller $3 \times 3 - 3$ MBConvs, and instead the *Single-Path NAS* flow selects SE paths with SE ratio of $se = 0.5$ in the last few layers. Compared to the previous result without SE (74.96% [18]), we confirm that the use of SE improves the accuracy-runtime trade-off of mobile ConvNets, as attested by the top1 accuracy improvement while remaining around the same latency setting $\sim$80 ms.

In addition, to understand the NAS choices related to the SE paths in our ConvNet, we report the relative runtime increase per MBConv types for each layer in Fig. 4 (right). We can make the following observations. First, we observe that the relative increase in the MBConv's runtime (scaling factor $s_{k,e,0.25}$ in (17) is closer to 1.0 for the last 4 layers. This is to be expected, since the *squeeze* 1×1 convolution is performed on input feature maps with reduced spatial dimensions. Indeed, we observe that *Single-Path NAS* appends SE paths in these last layers. Second, we notice that the difference in the relative runtime increase from using either SE ratios (0.25 or 0.5) is negligible, i.e., $s^i_{k,e,0.25} \approx s^i_{k,e,0.5}$. This is important in the context of NAS, since prior work only searches over the binary decision of using $se = 0.25$ or not, without searching for the se value. Indeed, *Single-Path NAS* selects $se = 0.5$ for all the SE paths when included.

Comparison with random search: An increasing amount of recent methods appeal to the practicality of random search as a simple, parameter-free NAS alternative [50]. It is therefore important to have a comparison of our result against random search. Specifically, we randomly sample ten ConvNets with predicted runtime from 75 ms to 80 ms (simple sampling by rejection). The average accuracy and runtime of the random samples are reported in Table I. We observe that, while random search does not outperform NAS methods, the overall accuracy is comparable to MobileNetV2. This result highlights that the effectiveness of NAS methods heavily relies upon the properties of the MobileNetV2-based design space. We provide an extensive analysis in Section VII-B, where we comprehensively study the variance in solutions from differentiable NAS and random search methods.

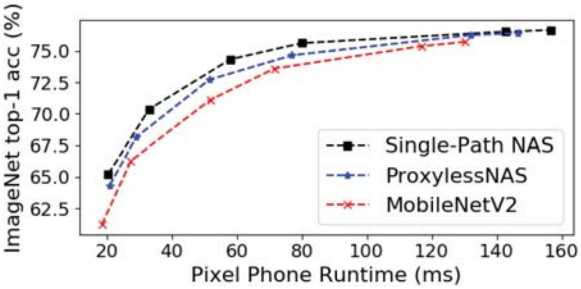


Fig. 7. *Single-Path* NAS outperforms MobileNetV2 [21] and ProxylessNAS [16] across various channel size scales.

Channel scaling: Next, we follow a typical analysis [15], [16], by rescaling the networks using a width multiplier [21]. As shown in Fig. 7, we observe that our model consistently outperforms prior methods under varying runtime settings. For instance, Single-Path NAS with 81.84 ms is 1.44× faster than the MobileNetV2 scaled model of similar accuracy.

A. 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

TABLE II
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/ $\mathbf{w}_{3\times3} + \mathbf{w}_{5\times5\backslash3\times3}$ kernels	73.86	91.72

model over two subsets, (i) the inner $\mathbf{w}_{3\times3}$ weights, and (ii) by also using the remaining $\mathbf{w}_{5\times5\backslash3\times3}$ 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 [51] (for the remaining training hyper-parameters we use the same setup as the default MnasNet). As shown in Table II, we observe the final accuracy across the kernel granularity, i.e., with the inner $\mathbf{w}_{3\times3}$ and the entire $\mathbf{w}_{5\times5} = \mathbf{w}_{3\times3} + \mathbf{w}_{5\times5\backslash3\times3}$ 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 [51] (SlimmableNets limit however their analysis across the channel dimension).

VI. COCO OBJECT DETECTION PERFORMANCE

In this Section, we assess the performance of *Single-Path NAS* as a feature extractor for object detection application. In particular, we use our network as a drop-in replacement for the backbone featurizer in the Mask-RCNN model [52], which is based on Feature Pyramid Network (FPN) [53] as head and our network as a backbone. Similarly, we train the model and we compare with other backbones networks, i.e., based on backbones from models designed from earlier mobile NAS methods. We train our model on the COCO dataset [54].

We use the open-source implementation of TPU-trained Mask-RCNN[2] for experiments. The models are trained on TPUs with batch size of 64. We train the different models on COCO `train2017` and we evaluate them on COCO `val2017`. Following typical the typical FPN flow [55], we attach the last feature extractor to the detection head. It is worth noticing that FPN is less hardware efficient compared to MobileNet-like alternatives such as SSDLite [21]. Nonetheless, the focus of this analysis is to assess the various NAS designs are feature extractor while assuming the head design (ergo, the latency) fixed. Indeed, in Table III we observe that *Single-Path NAS* outperforms other designs in terms of Average-precision (AP) and across all scales.

TABLE III
COCO OBJECT DETECTION PERFORMANCE

Method	AP	AP_S	AP_M	AP_L
MobileNet-V2 + Mask-RCNN	30.47	16.49	32.33	41.14
MnasNet-B1 + Mask-RCNN	32.47	17.74	34.45	43.88
ProxylessNAS + Mask-RCNN	32.93	17.76	34.86	44.43
Single-Path NAS + Mask-RCNN (Proposed)	**33.03**	**17.82**	**35.48**	**44.76**

VII. HYPERPARAMETER OPTIMIZATION OF DIFFERENTIABLE NAS

A. Architecture Distribution in Differentiable NAS

While NAS literature has been traditionally driven by strong empirical results, the AutoML community has motivated studies to understand the properties of NAS solvers, their limitations, how and why they yield strong performance [6]. Hence, we find important to investigate the following questions: "*How do the different NAS formulations, e.g., the encoding of NAS choices across multiple paths or a single path, affect the differentiable NAS performance?*" This is an important first step towards analyzing single-path formulations.

Moreover, prior work on mobile NAS [15], [16] lacks a detailed intra-level analysis on the statistics of differentiable methods, so a valid question to ask is: "*By how much does the quality of the ConvNet design vary across multiple runs of the same NAS search?*" For instance, Stochastic NAS [14] investigated the entropy of architecture distributions, but the analysis is limited to cell-based designs [12] and does not consider mobile AutoML.

To quantitatively answer these two questions, we consider the following differentiable NAS formulations:

1) Multi-path with sigmoid: This implementation solves the bilevel, multi-path formulation of (8). We implement a vanilla differentiable multi-path NAS solver [16]. While our implementation replicates prior work's methodology [15], we adjust the solver to the aggressive few-epoch schedule used in [2]. This allows us to assess whether existing multi-path methods can reach a high-performing ConvNet within the same number of epochs as *Single-Path NAS*.

Specifically, we set the number of total steps to eight epochs and we update the warm-up and learning rate schedules accordingly. We slim down the **multi-path supernet** by a width-multiplier factor of 0.5 (recent NAS work also employs such search on a scaled-down model [56]). Similar to [15], we generate a proxy dataset (i.e., subset of ImageNet with 100 classes) to search on. We deploy our implementation on TPUs.

Next, we investigate various **single-path**-based formulations:

2) Single-path with sigmoid: this is the default implementation detailed in Section III. That is, during search (backpropagation over the supernet) we approximate the indicator functions (e.g., $\mathbb{1}(\|\mathbf{w}_{5\times5\backslash3\times3}\|^2 > t_{k=5})$) with sigmoid functions $\sigma()$.

3) Single-path with STE: during search we approximate the indicator functions (e.g., $\mathbb{1}(\|\mathbf{w}_{5\times5\backslash3\times3}\|^2 > t_{k=5})$) with the straight-through estimator (STE) [57], [58].

[2] [Online]. Available: https://cloud.google.com/tpu/docs/tutorials/mask-rcnn

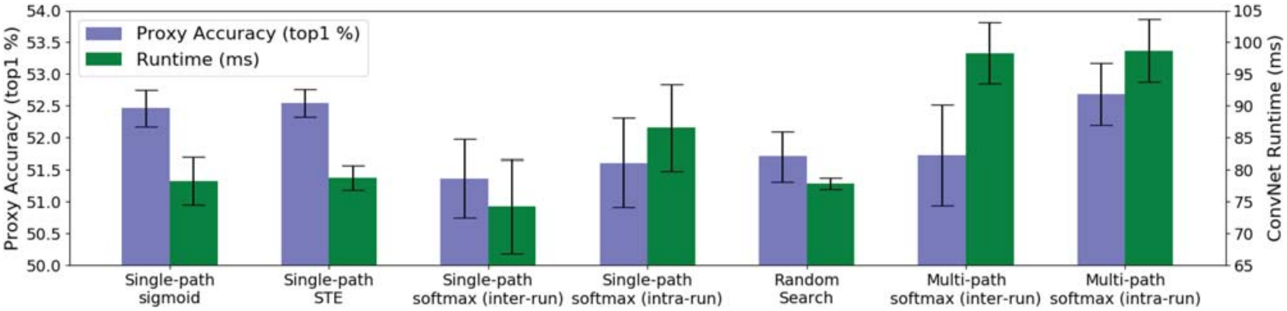


Fig. 8. *"How do the differentiable Mobile NAS formulation assumptions affect the overall performance (accuracy and runtime) of the AutoML-designed ConvNet?"* Statistics (mean and variance) for the (proxy) accuracy (top 1%) and the runtime of ConvNets designed via various formulations across 20 runs; for intra-run statistics, we pick the Pareto optimal ConvNet out of the 20 samples and we train another 20 ConvNets sampled from the softmax distribution.

4) Single-path with softmax: This implementation is a hybrid between the single-path encoding of the design space and the use of softmax, i.e., we encode the NAS choice of selecting across superkernel subsets using a softmax function parameterized by τ, i.e., softmax(τ). For instance, we represent the kernel-level decision as:

$$\mathbf{w}_k = \frac{\exp(\tau_{3\times3})}{\sum_j \exp(\tau_j)} \cdot \mathbf{w}_{3\times3} + \frac{\exp(\tau_{5\times5})}{\sum_j \exp(\tau_j)} \cdot (\mathbf{w}_{3\times3} + \mathbf{w}_{5\times5\setminus3\times3}) \tag{18}$$

To update the kernel-level softmax(τ) choices, we formulate the *Single-Path* search as a bilevel optimization problem $\min_{\tau} \min_{\mathbf{w}_\tau} \mathcal{L}(\tau, \mathbf{w}_\tau)$, where the steps for updating the NAS τ parameters and the ConvNet weights occur interchangeably.

5) Random search: Parameter-free random search via constrained sampling. That is, we employ simple sampling by rejection, i.e., we keep the samples with runtimes within the range of interest ∼80 ms.

For all the aforementioned methods, we find the λ value that achieves the desired accuracy trade-off ∼80 ms (to tune λ, we use the hyperparameter-tuning scheduler presented in the next Subsection VII-B). We repeat the same NAS search experiment 20 times and we measure the mean and (**inter-**) variance across the 20 runs for both objective terms, i.e., validation accuracy and runtime of the AutoML-designed ConvNet, denoted as *inter-run*. In addition, to capture the (**intra-**) variance within a single search in softmax-based methods, we pick the best result among the 20 runs, and we train 20 new samples from the softmax distribution (in fact, similar selection is used in [15] where 10 ConvNets are sampled and trained to pick the best). We denote the latter variant as *intra-run*. We train each ConvNet for a few epochs to obtain a representative proxy-accuracy value, following the aggressive training used in Mnasnet to study their RL method [2]. We summarize our results in Fig. 8.

Comparison vs. **random search:** This result is particularly interesting, since there has been recent discussion within the NAS community on whether simple random search could find designs with performance comparable to those of more complex methods [59]. Indeed, we observe that random search performs on par with multi-path cases, which confirms similar observations by recent work [50], [60]. Nonetheless, it is important to note that random search is still inferior compared to *Single-Path NAS* in terms of the (proxy) accuracy around the target latency range ∼80 ms.

Furthermore, the nearly-zero search cost of random search is not necessarily representative: to avoid training all random, constraint-satisfying samples, an AutoML practitioner would employ an evaluation on the proxy task, by training each sample for few epochs and by picking the one with highest accuracy. Hence, the actual search cost for random search is not negligible. In fact, the low search cost of our method (8 epochs) is comparable to the number of training epochs during the aforementioned selection process. Given that *Single-Path NAS* gives ConvNets with superior performance than random search at comparable cost, we argue that NAS remains a better AutoML option than random search methods.

Softmax intra-run variance: Next, we note the variance inherent to all the softmax-based cases. That is, we observe that sampling the softmax of the best NAS search (selected from the 20 NAS repetitions) yields high-variance in terms of both accuracy and runtime. This finding confirms a recent analysis that shows the high entropy in the architecture distribution for cell-based multi-path designs [14].

Different single-path variants: Moreover, we compare our original *Single-Path NAS* (single-path sigmoid) method against its two variants (i) with STE and (ii) with softmax (inter-run). First, once again we note that the softmax version has higher variance compared to both the sigmoid and the STE versions. For the STE version, while the variance appears smaller than sigmoid, it is important to note that we had to repeat the process multiple times to reach 20 completed searches due to encountered numerical instability issues with STE (exploding gradients). A deeper study on the STE is an interesting direction for future NAS work, similar to recent STE analysis in the context of hardware-aware quantization [57].

Single-Path NAS vs. **prior work:** Last, we highlight the advantage of using our proposed method (single-path sigmoid) *vs*. existing methods [15], [16] (multi-path softmax, inter-run). We observe that the variance across different *Single-Path NAS* runs is smaller than the variance of softmax-based methods (both inter- and intra-run).

Overall, we observe that multi-path softmax methods sample either low accuracy ConvNets (many layers skipped, which is another issue previously observed [14]) or higher accuracy

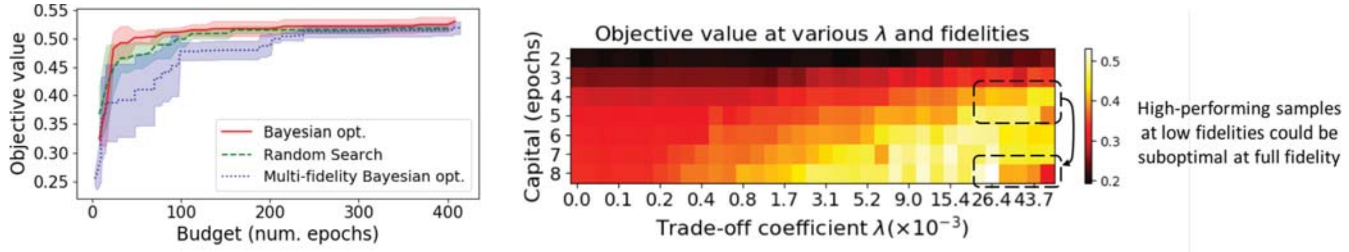


Fig. 9. **Left:** Progress of various hyperparameter optimization solvers with respect to the overall reward. **Right:** Visualizing the objective value (19) across multiple fidelities (y-axis) and hyperparameter values (x-axis) via grid search. Interestingly, low-cost function evaluations (middle, right) that reach the Pareto point around the target latency faster, tend to "overshoot" beyond this point towards over-constrained, suboptimal designs (bottom, right).

ones that violate the constraint. We hypothesize that the inferior solutions are due to the fact that the bilevel problem (8) is an intrinsically more complex optimization problem to solve, as also discussed in [12]. That is, it is difficult for the multi-path solver to reach a high quality solution within a few epochs, while our proposed *Single-Path NAS* for the same number of steps is as costly as training a compact model.

Besides the optimization complexity, one would argue that the performance of multi-path methods is decided by several hyperparameters. Indeed, we extensively experimented with numerous settings by varying the number of epochs between the interleaved steps (NAS *vs*. ConvNet weights updates), the learning rates for each update step, the batch size, the parameterization of the Gumbel-softmax [15], to name a few. Given that running each solver parameterization is expensive (hundreds of epochs), this highlights another limitation related to the tuning cost for all the hyperparameters involved, making our proposed method even more appealing to use. In fact, in the next subsection, we aim to fully erase this engineering cost for the AutoML practitioner, by automatically tuning the hyperparameters of *Single-Path NAS*.

B. Hypertuning the NAS Hyperparameterizer

NAS methods approximate Pareto solutions by a customized weighted objective parameterized by a trade-off parameter λ [2], but this value is manually picked. For instance, Mnasnet employs an empirical rule based on "prior" runtime-accuracy trade-off knowledge [2], while FBNet [15] and ProxylessNAS [16] do not provide details on the λ value used or how it was picked. Hence, we aim to answer the question: "*Instead of empirically tuning the trade-off hyperparameter, can we automatically find it for a target runtime given by the hardware engineers?*"

To this end, we formulate the tuning of λ (10) as a *hyperparameter optimization* problem itself. Specifically, we solve for the λ value that maximizes the validation accuracy around runtime target R_T. For a representative analysis, we use the weighted objective introduced in [2] that approximates Pareto optimal solutions, allowing our approach to traverse the Pareto front while solving for λ. Specifically, we write:

$$\max_{\lambda} Acc_{valid}(\lambda|\mathbf{w}, \mathbf{t}_k, \mathbf{t}_e, \mathbf{t}_{se}) \cdot \left[\frac{R(\lambda|\mathbf{w}, \mathbf{t}_k, \mathbf{t}_e, \mathbf{t}_{se})}{R_T}\right]^w$$

$$\text{with } w = \begin{cases} 0, & \text{if } R(\lambda|\mathbf{w}, \mathbf{t}_k, \mathbf{t}_e, \mathbf{t}_{se}) \leq R_T \\ -1, & \text{otherwise} \end{cases} \qquad (19)$$

We would like to stress here that each evaluation of (19) corresponds to new NAS search. Therefore, solving this hyperparameter optimization problem would be impractical with previous NAS methods where each function evaluation would cost hundreds of hours. Instead, we exploit the efficiency of *Single-Path NAS* and we investigate various *black-box* hyperparameter optimization techniques. Specifically, we consider the following methods:

1) Bayesian optimization [61]: Vanilla Bayesian optimization, as implemented in the `Dragonfly` tool [62], available online.[3] The method fits a Gaussian process (GP) [63] (probabilistic model) to the objective (19) by points sampled across the hyperparameter λ.

2) Multi-fidelity optimization [64]: Enchanced Bayesian optimization method where the GP fits both the hyperparameter space (λ values) and the *fidelity* (budget) space. The intuition is that low-fidelity evaluations could offer a good view of the function manifold at lower cost. We use discrete budget choices from two up to eight epochs (eight epochs is the default maximum in the vanilla case) as multiple fidelities. We use the multi-fidelity method from `Dragonfly` [62] which, for each new sample to evaluate, suggests the λ value and the sample budget (epochs).

3) Random search [65]: Parameter-free random search that randomly samples λ values.

We extend our AutoML framework to support hyperparameter optimization. Our implementation automates the process of launching multiple (sequential or parallel) runs on cloud TPUs and calls the *black-box optimization* solver that suggests the next sample to evaluate. Our goal is to find the trade-off λ value that yields Pareto-optimal designs around the target runtime of $R_T = 80$ ms. We run each solver for five runs with a total budget of 400 epochs and we track the current-best objective value. In Fig. 9 (left), we report the objective value per hyperparameter optimization method, where we plot the average-best and the variance across the five runs.

Vanilla vs. **multi-fidelity Bayesian optimization:** We observe that vanilla Bayesian optimization outperforms the multi-fidelity counterpart by reaching the near-optimal region faster and by converging to final solutions with higher reward. This is an interesting finding, since prior work shows that, for other hyperparameter settings (e.g., learning rate) multi-fidelity enhances the optimization process [64].

[3] [Online]. Available: https://github.com/dragonfly/dragonfly/

To fully investigate why this occurs, we employ a grid search across the budget epochs (from two to eight) and different λ values, and we plot the objective value (19) of each grid point in Fig. 9 (right). Indeed, we can observe that the main assumption that "low-cost samples give a representative view of the space" [64] does not fully hold. As highlighted in the Figure, we observe that initially promising λ values (brighter objective values obtained after four or five epochs, middle right) become suboptimal (darker, bottom right).

From a NAS design standpoint, the larger values λ penalize the runtime term more so they approach the Pareto point around the target latency faster, but they tend to "overshoot" beyond this point towards over-constrained designs. We find this result interesting, since we postulate that other *black-box* optimization techniques that rely on low-cost (early) approximation (e.g., Hyperband [66]) would encounter the same issue. Studying this hyperparameter optimization problem is an interesting research direction currently under-explored, so we aim to delve into this problem in future work.

Comparison vs. **random search:** We find that random search, while never outperforming the Bayesian optimization result, has a relatively good performance at tuning λ. Interestingly, recent work shares similar observation when tuning NAS scaling hyperparameters via grid search [56]. We hope that our analysis would foster exploration towards this direction.

VIII. 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 3 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. We enhanced the accuracy-runtime trade-off in differentiable NAS by treating the Squeeze-and-Excitation path as a fully searchable operation with our *single-path* encoding. *Single-Path NAS* achieved 75.62% top-1 accuracy on ImageNet, which is state-of-the-art accuracy compared to NAS methods around similar latency setting ($\sim$80 ms). More importantly, we reduced the NAS search cost down to only 8 epochs (24 TPU-hours), which is up to **5,000× faster** compared to prior work.

Moreover, we exploited the efficiency of our method to answer questions related to the effectiveness of differentiable NAS. In particular, we studied how different NAS formulation choices affect the performance of the designed ConvNets. Last, we explored whether we can automatically find the NAS hyperparameters that yield the desired accuracy-runtime trade-off, by formulating the tuning of the NAS solver as a hyperparameter optimization problem itself.

Acknowledgment

Dimitrios Stamoulis also acknowledges support from the Qualcomm Innovation Fellowship (QIF) 2018, GCP Research Credits program, and the TensorFlow Research Cloud (TFRC) program.

References

[1] A. Howard *et al.*, "Searching for MobileNetV3," in *Proc. IEEE Int. Conf. Comput. Vision*, 2019, pp. 1314–1324.

[2] M. Tan *et al.*, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 2820–2828.

[3] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 8697–8710.

[4] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," 2018, *arXiv:1802.01548*.

[5] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 2016–2025.

[6] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 549–558.

[7] GitHub, "Bayesian Optimization in PyTorch," 2020. [Online]. Available: https://www.botorch.org/, Accessed on: Jan. 24, 2020.

[8] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google Vizier: A service for black-box optimization," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining.*, 2017, pp. 1487–1495.

[9] Microsoft (GitHub open-source project), "NNI (Neural Network Intelligence)," 2020. [Online]. Available: https://github.com/microsoft/nni, Accessed on: Jan. 24, 2020.

[10] Google, "Google AutoML Beta," 2020. [Online]. Available: https://cloud.google.com/automl/, Accessed on: Jan. 24, 2020.

[11] M. Tan, "MnasNet: Towards Automating the Design of Mobile Machine Learning Models," 2018. [Online]. Available: https://ai.googleblog.com/2018/08/mnasnet-towards-automating-design-of.html, Accessed on: Jan. 24, 2020.

[12] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–13.

[13] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4092–4101.

[14] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–17.

[15] B. Wu *et al.*, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 10726–10734.

[16] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–13.

[17] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 7132–7141.

[18] D. Stamoulis *et al.*, "Single-Path NAS: Designing hardware-efficient convnets in less than 4 hours," in *Proc. Eur. Conf. Mach. Learn. Princ. Pract. Knowl. Discovery Databases*, 2019, pp. 1–16.

[19] D. Stamoulis *et al.*, "Designing adaptive neural networks for energy-constrained image classification," in *Proc. Int. Conf. Comput.-Aided Des.* ACM, 2018, pp. 1–8.

[20] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[21] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 4510–4520.

[22] T.-W. Chin, C. Zhang, and D. Marculescu, "Layer-compensated pruning for resource-constrained convolutional neural networks," 2018, *arXiv:1810.00518*.

[23] A. H. Ashouri, T. S. Abdelrahman, and A. D. Remedios, "Fast on-the-fly retraining-free sparsification of convolutional neural networks," 2018, *arXiv:1811.04199*.

[24] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 1–14.

[25] R. Ding, Z. Liu, R. Shi, D. Marculescu, and R. Blanton, "Lightnn: Filling the gap between conventional deep neural networks and binarized networks," in *Proc. Great Lakes Symp. VLSI ACM*, 2017, pp. 35–40.

[26] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–14.

[27] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2849–2858.

[28] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–16.

[29] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 1761–1770.

[30] Y. Zhou, S. Ebrahimi, S. Ö. Arık, H. Yu, H. Liu, and G. Diamos, "Resource-efficient neural architect," 2018, *arXiv:1806.07912*.

[31] D. Stamoulis, E. Cai, D.-C. Juan, and D. Marculescu, "Hyperpower: Power-and memory-constrained hyper-parameter optimization for neural networks," in *Proc. IEEE Des., Autom. Test Eur. Conf. Exhib.*, 2018, pp. 19–24.

[32] D. Marculescu, D. Stamoulis, and E. Cai, "Hardware-aware machine learning: modeling and optimization," in *Proc. Int. Conf. Comput.-Aided Des. ACM*, 2018, Art. no. 137.

[33] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "Dpp-net: Device-aware progressive search for pareto-optimal neural architectures," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 517–531.

[34] C.-H. Hsu *et al.*, "Monas: Multi-objective neural architecture search using reinforcement learning," 2018, *arXiv:1806.10332*.

[35] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–22.

[36] R. Shin, C. Packer, and D. Song, "Differentiable neural network architecture search," in *Proc. Int. Conf. Learn. Representations* (Workshop track), 2018, pp. 1–4.

[37] A. Hundt, V. Jain, and G. D. Hager, "sharpdarts: Faster and more accurate differentiable architecture search," 2019, *arXiv:1903.09900*.

[38] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," 2019, *arXiv:1904.00420*.

[39] T. Chen *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th {USENIX} Symp. Operating Syst. Des. Implementation ({OSDI})*, 2018, pp. 578–594.

[40] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On neural architecture search for resource-constrained hardware platforms," in *Proc. Int. Conf. Comput.-Aided Des. ACM*, 2019, pp. 1–8.

[41] P. Guo and Y. Li, "Multistage decision making based on one-shot decision theory," in *Knowledge Engineering and Management*, Y. Wang and T. Li, Eds. Berlin Heidelberg: Springer, 2011, pp. 159–164.

[42] R. Ding, Z. Liu, T.-W. Chin, D. Marculescu, and R. Blanton, "Flightnns: Lightweight quantized deep neural networks for fast and accurate inference," in *Proc. Des. Autom. Conf.*, 2019, Art. no. 200.

[43] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," 2018, *arXiv:1805.06085*.

[44] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," in *Proc. Asian Conf. Mach. Learn.*, 2017, pp. 622–637.

[45] X. Dai *et al.*, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 11 398–11 407.

[46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2009, pp. 248–255.

[47] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Architecture*, 2017, pp. 1–12.

[48] GitHub, "Facebook AI Performance Evaluation Platform (FAI-PEP)," 2020. [Online]. Available: https://github.com/facebook/FAI-PEP, Accessed on: Jan. 24, 2020.

[49] D. Stamoulis *et al.*, "Single-path nas: Device-aware efficient convnet design," 2019, *arXiv:1905.04159*.

[50] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in Proc. IEEE Int. Conf. Comput. Vision, 2019, pp. 1284–1293.

[51] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–12.

[52] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proc. IEEE Int. Conf. Comput. Vision*, 2017, pp. 2961–2969.

[53] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 2117–2125.

[54] T.-Y. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vision.*, 2014, pp. 740–755.

[55] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "NAS-FPN: Learning scalable feature pyramid architecture for object detection," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 7036–7045.

[56] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[57] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–30.

[58] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432*.

[59] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. Conf. Uncertainty in Artif. Intell. (UAI)*, 2019, pp. 1–20.

[60] M. Cho, M. Soltani, and C. Hegde, "One-shot neural architecture search via compressive sensing," 2019, *arXiv:1906.02869*.

[61] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *IEEE Proc.*, vol. 104, no. 1, pp. 148–175, Jan. 2016.

[62] K. Kandasamy *et al.*, "Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly," 2019, *arXiv:1903.06694*.

[63] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. Cambridge: MIT press, 2006, vol. 1.

[64] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos, "Multi-fidelity bayesian optimisation with continuous approximations," in *Proc. 34th Int. Conf. Mach. Learn.-Vol. 70*. JMLR. org, 2017, pp. 1799–1808.

[65] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. Feb, pp. 281–305, 2012.

[66] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6765–6816, 2017.

Dimitrios Stamoulis (Student Member, IEEE) received the B.S. degree in electrical and computer engineering (ECE) from the National Technical University of Athens, Athens, Greece, in 2013, and the M.Eng. degree in ECE from McGill University, Montreal, QC, Canada, in 2015. He is currently working toward the Ph.D. degree in ECE from Carnegie Mellon University, Pittsburgh, PA, USA. His research focuses on hyperparameter optimization of deep learning models under hardware constraints.

Ruizhou Ding (Student Member, IEEE) received the B.S. degree in electronic and information science and technology from Peking University, Beijing, China, in 2015. He is currently working toward the Ph.D. degree in electrical and computer engineering with Carnegie Mellon University, Pittsburgh, PA, USA.

Di Wang (Member, IEEE) received the B.E. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2005, the M.S. degree in computer systems engineering from the Technical University of Denmark, Lyngby, Denmark, in 2008, and the Ph.D. degree in computer science and engineering from The Pennsylvania State University, State College, PA, USA, in 2014. His research spans the areas of artificial intelligence, computer systems, computer architecture, and energy efficient system design and management.

Dimitrios Lymberopoulos (Member, IEEE) received the Ph.D. degree from the Electrical Engineering Department, Yale University, New Haven, CT, USA, in 2008, where he designed and implemented wireless sensor networks for privacy-preserving, in-home elderly care monitoring. He is the Principal Research Manager with Microsoft, Redmond, WA, USA. His current work focuses on designing custom deep learning techniques for challenging computer vision problems with an emphasis on aerial image understanding. Previously, he focused on low power sensing architectures, indoor location technologies and mobile context sensing for mobile web search services.

Bodhi Priyantha (Member, IEEE) received the Ph.D. degree in electrical engineering and computer sciences from the Massachusetts Institute of Technology, Cambridge, MA, USA. He is the Principal Researcher with Microsoft Research, Redmond, WA, USA. His research interests include low-power systems design, location technologies, and networked sensing. Among other recognitions, he was the recipient of Best Paper Awards in MobiSys 2014, MobiSys 2013, SenSys 2012, and RTAS 2010.

Jie Liu (Fellow, IEEE) is a Chair Professor with the Harbin Institute of Technology (HIT), Harbin, China and the Dean of its AI Research Institute. Before joining HIT, he spent 18 years with Xerox PARC, Microsoft Research and Microsoft product teams. He was a Partner of Microsoft. As a Principal Research Manager with MSR, he led the Sensing and Energy Research Group. In MSR-NExT and product groups, he incubated smart retail solutions, which became part of Microsoft Business AI offering. He has authored more than 120 peer-reviewed papers. His research interests root in understanding and managing the physical properties of computing. He received six Best Paper Awards from top academic conferences (h-index = 62). He has filed more than 100 patents, with 50+ awarded. He has chaired a number of top-tier conferences in sensing and pervasive computing. Currently, he is the Steering Committee Chair for Cyber-Physical Systems and Internet of Things Week (CPS-IoT Week), Steering Committee Chair for ACM/IEEE International Conference on Information Processing in Sensor Networks. He was an Associate Editor for the IEEE TRANSACTIONS ON MOBILE COMPUTING and ACM TRANSACTIONS ON SENSOR NETWORKS. He is an ACM Distinguished Scientist.

Diana Marculescu (Fellow, IEEE) received the Dipl.Ing. degree in computer science from the Polytechnic University of Bucharest, Bucharest, Romania, in 1991, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1998. She is the Department Chair, Cockrell Family Chair for Engineering Leadership #5, and a Professor, Motorola Regents Chair in Electrical and Computer Engineering #2, with the University of Texas at Austin. Prior to joining UT Austin in December 2019, she was the David Edward Schramm Professor of Electrical and Computer Engineering, the Founding Director of the College of Engineering Center for Faculty Success (2015–2019) and has served as an Associate Department Head for Academic Affairs in Electrical and Computer Engineering (2014–2018), all with Carnegie Mellon University. Her research interests include energy- and reliability-aware computing, hardware aware machine learning, and computing for sustainability and natural science applications. She was the recipient of the National Science Foundation Faculty Career Award (2000–2004), the ACM SIGDA Technical Leadership Award (2003), the Carnegie Institute of Technology George Tallman Ladd Research Award (2004), and several best paper awards. She was the IEEE Circuits and Systems Society Distinguished Lecturer (2004–2005) and the Chair of the Association for Computing Machinery (ACM) Special Interest Group on Design Automation (2005–2009). She chaired several conferences and symposia in her area and is currently an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS. She was selected as an ELATE Fellow (2013–2014), and is the recipient of an Australian Research Council Future Fellowship (2013–2017), the Marie R. Pistilli Women in EDA Achievement Award (2014), and the Barbara Lazarus Award from Carnegie Mellon University (2018). She is a Fellow of ACM.