Searching Beyond MobileNetV3

Xiangxiang Chu, Bo Zhang, Ruijun Xu

Xiaomi AI Lab
{chuxiangxiang, zhangbo11, xuruijun}@xiaomi.com

Abstract

The evolution of MobileNets has laid a solid foundation for neural network applications on mobile end. With the latest MobileNetV3, neural architecture search again claimed its supremacy in network design. Unfortunately, till today all mobile methods mainly focus on CPU latencies instead of GPU, the latter, however, is much preferred in practice for it has faster speed, lower overhead and less interference. Bearing the target hardware in mind, we propose the first Mobile GPU-Aware (MoGA) neural architecture search in order to be precisely tailored for real-world applications. Further, the ultimate objective to devise a mobile network lies in achieving better performance by maximizing the utilization of bounded resources. Urging higher capability while restraining time consumption is not reconcilable. We alleviate the tension by weighted evolution techniques. Moreover, we encourage increasing the number of parameters for higher representational power. With $200\times$ fewer GPU days than MnasNet, we obtain a series of models that outperform MobileNetV3 under the similar latency constraints, i.e., MoGA-A achieves **75.9%** top-1 accuracy on ImageNet, MoGA-B meets 75.5% which costs only 0.5 ms more on mobile GPU. MoGA-C best attests GPU-awareness by reaching 75.3% and being slower on CPU but faster on GPU. The models and test code are made publicly here [1][2].

1 Introduction

The MobileNets trilogy has opened a gate to on-device artificial intelligence for the mobile vision world (Howard et al. 2017; Sandler et al. 2018; Howard et al. 2019). In the meantime, neural architecture search becomes the new engine to empower the future architecture innovation (Zoph et al. 2018; Tan et al. 2019; Cai, Zhu, and Han 2019; Chu et al. 2019a). The guideline in designing mobile architecture is that not only should the high performance be concerned, but also we must strive for low latency in favor of rapid responsiveness and improved power efficiency to prolong battery life.

In this paper, we aim to bring forward the frontier of mobile neural architecture design by stretching out the repre-
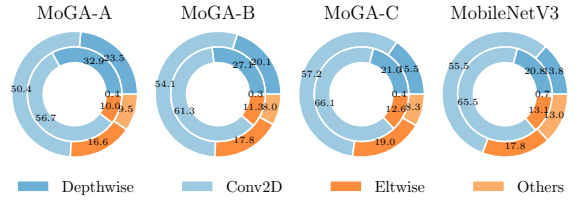

Figure 1: Latency pie chart of MoGA-A/B/C, MobileNetV3 operations when run on mobile CPUs (inner circle with TFLite) vs. on mobile GPUs (outer circle with MACE).

sentational space within the desired latency range. Our work can be summarized in following aspects.

First, we make a shift in the search trend from mobile CPUs to mobile GPUs, with which we can gauge the speed of a model more accurately and provide a production-ready solution. On this account, our overall search approach is named Mobile GPU-Aware neural architecture search (MoGA). Our results suggest that generated models show different behavior related to the targeted hardware as shown in Figure 1.

Second, we replace traditional multi-objective optimization with a weighted fitness strategy. While considering accuracy, latency and the number of parameters as our objectives, particular care is required to abate these three contending forces. One important insight is that the number of parameters should be made reasonably large instead of as few as possible, this leverages performance but doesn't necessarily increase latency. At the mobile scale, this would be the proper choice as we try to avoid underfitting instead of overfitting. On top of that, we lay more attention on accuracy and latency than the number of parameters.

Third, as per search cost, we benefit from one-shot supernet training and an accurate latency look-up table. Actually, it only requires the same expense as training a standalone model. The overall pipeline costs **12 GPU days**, about **200×** less than MnasNet (Tan et al. 2019). More importantly, to cater for various mobile devices, as we decouple search process from training, it only requires one more inexpensive search with a renewed latency table. In

---

[1] https://github.com/xiaomi-automl/MoGA

[2] This is a preview version, subject to frequent changes.

contrast, gradient descent and reinforced methods have to start all over for supernet training or incomplete training for multitudinous models (Liu, Simonyan, and Yang 2019; Tan et al. 2019).

Finally, we present our searched architectures that outperform MobileNetV3. MoGA-A that achieves 75.9% top-1 accuracy on ImageNet, MoGA-B 75.5% and MoGA-C 75.3%. MoGA-C is best comparable to MobileNetV3, with similar FLOPs and an equal number of parameters, which runs slower on mobile CPUs but faster on mobile GPUs.

2   Related Works

During the era of human craftsmanship, MobileNetV1 and V2 (Howard et al. 2017; Sandler et al. 2018) have widely disseminated depthwise separable convolutions and inverted residuals with linear bottlenecks. Moreover, Squeeze and excitation blocks are later introduced in (Hu, Shen, and Sun 2018) to enrich residual modules from ResNet (He et al. 2016).

In their aftermath, a series of automated architectures are searched based on these building blocks (Tan et al. 2019; Cai, Zhu, and Han 2019; Chu et al. 2019a; Howard et al. 2019). For instance, MnasNet frames a factorized hierarchical search space with MobileNetV2's inverted bottleneck convolution blocks (MB) of variable kernel sizes and expansion rates. Its latest variation also includes an option of squeeze and excitation module (SE) (Tan et al. 2019). ProxylessNAS and FairNAS adopt a similar design (Cai, Zhu, and Han 2019; Chu et al. 2019a) without SE modules, while MobileNetV3 achieves a new state of the art by integrating SE within MnasNet search space, along with numerous techniques like Platform-Aware NAS (Tan et al. 2019), NetAdapt (Yang et al. 2018) and improved non-linearities (Howard et al. 2019).

As for search methods, recent attention has been drawn to the one-shot approaches initiated by (Bender et al. 2018), as they tremendously reduce computing resources and also offer state of the art results (Cai, Zhu, and Han 2019; Stamoulis et al. 2019; Guo et al. 2019; Chu et al. 2019a). Briefly, a one-shot approach embodies weight-sharing across models by constructing a supernet where each step of training accounts for the final performance. Its single-path variations further cut down memory consumption by training a picked path at each step instead of the whole supernet, yielding more flexibility for architecture design (Stamoulis et al. 2019; Guo et al. 2019; Chu et al. 2019a). Among them, FairNAS proved it is critical to maintaining strict fairness for training single-path nets so to reach a steady rank, which can reasonably facilitate the search process (Chu et al. 2019a).

3   Mobile GPU-Aware NAS Based on Multi-Objective Optimization

In this section, to better formulate our design problem, we draw insights from the development of MobileNets and experiments on the mobile GPU/CPU relationship, as well as reviewing previous optimization approaches.
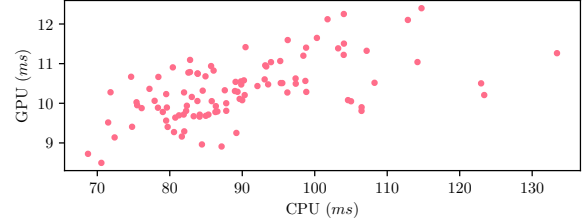
3.1   Mobile GPU Awareness

Figure 2: Latency relationship on mobile CPUs vs. on mobile GPUs.

Recent NAS approaches have an increased emphasis on target platforms, primarily on mobile CPUs (Tan et al. 2019; Dong et al. 2018; Wu et al. 2019; Cai, Zhu, and Han 2019; Stamoulis et al. 2019; Howard et al. 2019). MnasNet has developed a reward $ACC \times (LAT/TAR)^w$, which requires delicate manual tuning for a parameter w to balance between latency and accuracy (Tan et al. 2019), in MobileNetV3, w is reduced from -0.07 to -0.15 (Howard et al. 2019) to compensate for accuracy drop.

In practice, mobile neural networks are mostly deployed to run on GPUs, DSPs and recently also on specific Neural Processing Units (NPUs), while CPUs would be the last to choose. To further investigate the relationship of CPU latencies versus GPU ones, we measure 100 random models on both two platforms. The result is shown in Figure 2. We see that there is no obvious linear correspondence. Hence, To develop architectures with target hardware in mind is more than necessary. For this reason, we are driven to apply Mobile GPU awareness to the latest neural architecture search approaches.

3.2   Underfitting and Overfitting

As we try to tear apart two contradicting objectives, there isn't too much freedom left to increase accuracy with a constrained latency. Fortunately, we observe from the evolution of MobileNets as in Figure 3, the number of parameters has grown while the latencies and multiply-adds are kept low.
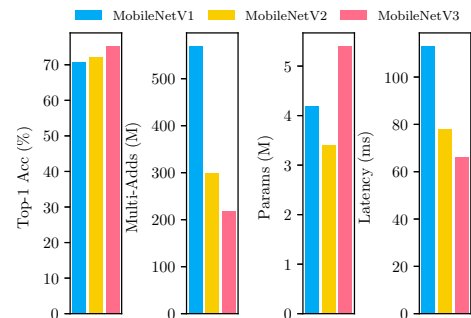

Figure 3: The evolution of MobileNets.

Moreover, for the mobile end, models tend to be underfitted instead of overfitted since they carry fewer numbers of

parameters (Zhang et al. 2018), which means we are free to encourage representational power by enlarging its range of parameters. This intuition greatly expands our design space.

3.3 Problem Formulation

Most hardware-aware methods build the classification problem as follows,

$$
\begin{aligned}
\max \quad & Accuracy(m) \\
s.t. \quad & Latency(m) < L \\
& \text{model } m \in \Omega.
\end{aligned}
\tag{1}
$$

where Ω is the whole search space and L is a given maximal acceptable latency[3]. Informally speaking, larger models have greater capacity and to achieve better accuracy. Therefore, NAS methods will prefer models which have large running time. As a result, when the requirement of L changes, the whole NAS pipeline must start over.

To address the above problem, a recent popular approach formulate it as multi-objective problem (MOP) whose solution is called Pareto Front,

$$
\begin{aligned}
max \quad & \{Accuracy(m), -Latency(m)\} \\
& m \in \Omega.
\end{aligned}
\tag{2}
$$

One popular approach for Equation 2 is converting it into a customized objective of weighted product $ACC \times (LAT/TAR)^w$, which requires delicate manual tuning for a parameter w to balance between latency and accuracy (Tan et al. 2019).

Upon these previous attempts, as inspired by Section 3.2, we maximize the number of parameters in addition to the two objectives in Equation 2. More formally, we try to solve the following problem,

$$
\begin{aligned}
max \quad & \{Accuracy(m), -Latency(m), Params(m)\} \\
& m \in \Omega.
\end{aligned}
\tag{3}
$$

As a matter of fact, these three objectives are not of equal importance in most cases. A typical case is like Equation 2. Therefore, we need to introduce some strategies to address the issue. Let $w_{acc}, w_{lat}, w_{params}$ denote customized preference for those objectives. Without loss of generality, the problem can be defined as,

$$
\begin{aligned}
min \quad & \{-Accuracy(m), Latency(m), -Params(m)\} \\
s.t. \quad & m \in \Omega \\
& w_{acc} + w_{lat} + w_{params} = 1 \\
& w_{acc}, w_{lat}, w_{params} >= 0.
\end{aligned}
\tag{4}
$$

There are two basic subproblems to be solved in the next section. One is to instantly evaluate accuracy and latency of a model, the other is to solve Equation 4. We use NSGA-II,

---

[3]The following latency means mobile GPU latency.

Index	Expansion	Kernel Size	SE
0	3	3	-
1	3	3	✓
2	3	5	-
3	3	5	✓
4	3	7	-
5	3	7	✓
6	6	3	-
7	6	3	✓
8	6	5	-
9	6	5	✓
10	6	7	-
11	6	7	✓

Table 1: Each layer in our search space has 12 choices. SE: Squeeze-and-Excitation.

which is one of the most powerful and widely used algorithms to solve such problems (Deb et al. 2002). First, it's efficient to solve MOPs, especially when the number of objectives is large, some variants can still work. Second, it's flexible to apply customized preferences for different objectives, as well as various constraints. We also benefit from its implicit objective scaling and normalization.

4 Solving it Using Weighted NSGA-II

4.1 Search Space

Our search space is built layer by layer on inverted bottleneck blocks as (Cai, Zhu, and Han 2019; Chu et al. 2019a). We keep the same number of layers and activation functions as MobilenetV3-large. For each layer, we search from three dimensions (see Table 1):

- the convolution kernel size (3, 5, 7)
- the expansion ratio for the inverted bottleneck block (3, 6)
- whether the squeezing and excitation mechanism is enabled or not.

Therefore, the total search space has a volume of 12^{14}, which needs efficient methods to differentiate better models from worse. To be simple, we search for the expansion rate instead of channels which is used by (Howard et al. 2019) based on NetAdapt (Yang et al. 2018). Besides, we utilize choice index to directly encode each model chromosome. More formally, a model chromosome m can be written as $m_1 = (x_1^1, x_2^1, ..., x_{14}^1)$.

4.2 Accuracy and Latency Prediction

The evaluation of model accuracy must be made immediate for searching efficiency. We take advantage of a variation of one-shot approaches FairNAS for fast evaluation with a stable ranking. Unlike their version, based on our previously defined search space, we construct a supernet with 12 choice blocks per layer. Then we train our supernet on the ImageNet dataset with the same fairness strategy.

As for mobile GPU latency, we don't acquire real-time latency during the pipeline from a cell phone for two reasons. One is that while the performance can be rapidly predicted

by the supernet which takes less than 1 minute, it will easily become the bottleneck when we use a mobile device to evaluate latency on the fly. The other is that latency measurement may become inaccurate as a result of overheating after long-time insistent testing.

Instead, since each choice block in our search space has a fixed input, we can efficiently approximate the latency for any sampled model. To do so, we benchmark the latency of each choice block under a given input and construct a layerwise lookup table. We can then accurately calculate the latency simply by accumulating time cost across all layers. We find that predicted GPU latency coincides with ground-truth values with a negligible RMSE, see Figure 4.
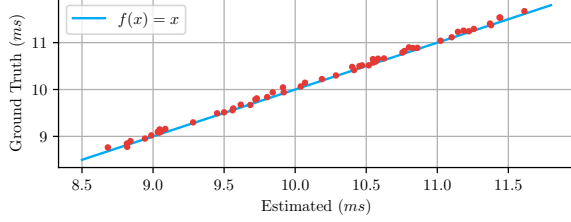

Figure 4: Mobile GPU latency measured vs. predicted ones. The latency RMSE is 0.0571ms.

4.3 Weighted NSGA-II

We comply with the standard NSGA-II procedure, and only state the difference if necessary.

Population Initialization We initialize population to introduce various choice blocks to encourage exploration.

Crossover We take a single-point crossover. Specifically, for two models $m_1 = (x_1^1, x_2^1, ..., x_{14}^1)$ and $m_2 = (x_1^2, x_2^2, ..., x_{14}^2)$, if the point position is k, the result after crossover is $(x_1^1, x_2^1, ...x_k^2..., x_{14}^1)$.

Mutation We use hierarchical mutation and the same hyperparameters as FairNAS (Chu et al. 2019a).

Non-dominated Sorting For a minimization problem with n objectives, we state that A dominates B means for any objective O^i, $O_A^i \leq O_B^i$. For a given population P, A is not dominated if and only if A is not dominated by any other individuals.

The crowding distance is a key component to achieve better trade-off among various objectives. We use the customized weights to define the crowding distance for non-boundary individuals,

$$D(m_j) = \sum_{i=1}^{n} w_i * \frac{O_{neighbor+}^i - O_{neighbor-}^i}{O_{max}^i - O_{min}^i}. \quad (5)$$

Note $O_{neighbour-}^i$ and $O_{neighbor+}^i$ are the i-th objective value of the left and the right neighbor of model m_j respectively, while O_{max}^i and O_{min}^i are the maximum and the minimum for the i-th objective in the current population. In Equation 5, customized preference can be flexibly

---

Algorithm 1 The weighted NAS pipeline.

Input: Supernet S, search space Ω, the number of generations N, population size n, validation dataset D, objective weights w
Output: A set of K individuals on the Pareto front.
Train supernet S by the FairNAS approach on Ω.
Make gpu latency table T as section 4.2.
Uniformly generate the populations P_0 and Q_0 until each has n individuals.
for $i = 0$ **to** $N - 1$ **do**
  $R_i = P_i \cup Q_i$
  $F = $ non-dominated-sorting(R_i)
  Pick n individuals to form P_{i+1} by ranks and the crowding distance **weighted** by w based on Equation 5.

  $Q_{i+1} = \emptyset$
  while $size(Q_{i+1}) < n$ **do**
    $M = $ tournament-selection(P_{i+1})
    $q_{i+1} = $ crossover$(M) \cup$ hierarchical-mutation(M)
    Obtain fitness value across all objectives
      Evaluate model q_{i+1}'s accuracy with S on D
      Regress model q_{i+1}'s latency based on T
    $Q_{i+1} = Q_{i+1} \cup \{q_{i+1}\}$
  end while
end for
Select K models at an equal distance near Pareto front from P_N

---

incorporated. If $w_{acc} = w_{lat} = w_{params} = \frac{1}{3}$, it degrades as the standard NSGA-II. In our experiment, $w_{acc} = w_{lat} = 0.4, w_{params} = 0.2$, whose requirement is from a practical application.

4.4 Our NAS Pipeline

Our search pipeline is an evolution process, drawn in Figure 5 and detailed in Algorithm 1. Specifically, we use a trained supernet as a fast evaluator, a GPU latency lookup table, and a statistic tool to compute the number of parameters. The initial random population propagates at significant speed. The pipeline evolves 120 generations with a population size 70, and it only takes about 1.5 GPU days to evaluate these 8400 models. We use the same hyperparameters as FairNAS.
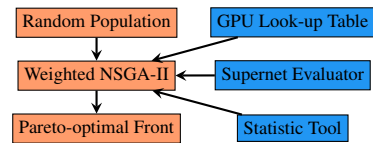

Figure 5: The overall pipeline of MoGA.

5 Experiments

5.1 Mobile GPU Latency

In practice, we employ SNPE (Qualcomm 2019) and Mobile AI Compute Engine (MACE) for mobile GPU bench-

Input	Ops	t	c	SE	NL	s
$224^2 \times 3$	conv2d, 3×3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3×3	1	16	-	RE	1
$112^2 \times 16$	bneck, 5×5	6	24	-	RE	2
$56^2 \times 24$	bneck, 7×7	6	24	-	RE	1
$56^2 \times 24$	bneck, 3×3	6	40	-	RE	2
$28^2 \times 40$	bneck, 3×3	6	40	✓	RE	1
$28^2 \times 40$	bneck, 3×3	3	40	✓	RE	1
$28^2 \times 40$	bneck, 3×3	6	80	✓	HS	2
$14^2 \times 80$	bneck, 3×3	6	80	-	HS	1
$14^2 \times 80$	bneck, 7×7	6	80	-	HS	1
$14^2 \times 80$	bneck, 7×7	3	80	✓	HS	1
$14^2 \times 80$	bneck, 7×7	6	112	-	HS	1
$14^2 \times 112$	bneck, 3×3	6	112	-	HS	1
$14^2 \times 112$	bneck, 3×3	6	160	-	HS	2
$7^2 \times 160$	bneck, 5×5	6	160	✓	HS	1
$7^2 \times 160$	bneck, 5×5	6	160	✓	HS	1
$7^2 \times 160$	conv2d, 1×1	-	960	-	HS	1
$7^2 \times 960$	avgpool, 7×7	-	-	-	HS	-
$1^2 \times 960$	conv2d, 1×1	-	1280	-	HS	1
$1^2 \times 1280$	conv2d, 1×1	-	k	-	-	-

Table 2: The architecture of MoGA-A. Note t, c, s refer to expansion rate, output channel size and stride respectively. SE for squeeze-and-excitation, NL for non-linearity. k for the number of categories.

marking (Xiaomi 2018). We randomly sample some models and report the differences between our predictions and on-device measurements, which are shown in Figure 4. For instant latency prediction, we construct a latency lookup table based on MACE measurements on all 12 choices blocks for each cell (12×14). For the final comparison with state-of-the-art models, we also report mobile GPU latencies with SNPE (Qualcomm 2019), and CPU latencies with Tensorflow Lite (Abadi et al. 2015). Considering recent updates on Tensorflow speed up the inference time, we choose a version that can reproduce the result on MobileNetV2 (Sandler et al. 2018).

Unless otherwise noted, mobile CPU latencies are measured on a Google Pixel 1 using a single large core of CPU with a batch size of 1. Mobile GPU latencies are benchmarked on a Mi MIX 3. The input size is set to 224×224.

5.2 Training

Training of our Supernet We search proxylessly on the ImageNet (Deng et al. 2009) classification dataset. We take out 50k images from the training set to form our validation set and use the official validation set as our test set to evaluate our models, which is on par with other methods. In particular, we train the supernet by SGD with momentum 0.9 for 32 epochs. The initial learning rate is 0.05 and is scheduled to arrive at zero within a single cosine cycle.

Training for Stand-Alone models To alleviate the training unfairness, we utilize the same training tricks and hyperparameters as MobileNetV3 (Howard et al. 2019).6. By doing so, we singled out various training tricks in order to

focus on the authentic model performance. In particular, We use a batch size of 4096 and RMSProp optimizer with 0.9 momentum. The initial learning rate is 0.1 and linear warm-up (Goyal et al. 2017) is applied for the first 5 epochs. We use a dropout rate of 0.2 before the last layer (Srivastava et al. 2014) and L2 weight decay $1e-5$. Besides, we make use of NVIDIA's mixed precision library Apex to enable larger batch size[4]. All our experiments are performed on two Tesla-V100 machines.

5.3 Comparisons with State-of-the-art Methods

We are mostly comparable to the latest version of MnasNet (Tan et al. 2019) and MobileNetV3 (Howard et al. 2019), as we share the similar search space. Also, we use the same training and data processing tricks as in (Tan et al. 2019) for complete training of stand-alone models. Note that with latency considered as one of the objectives, our generated models pay more attention to increase the number of parameters in order to gain higher performance, see detailed comparison results in Table 3. We list all layers of MoGA-A in Table 2, and illustrate the whole MoGA family in Figure 8.

For a fair comparison, here we only consider single path models based on inverted bottleneck blocks. MoGA-A achieves a new state-of-the-art top-1 accuracy 75.9%, surpassing Proxyless-R Mobile (+1.3%), MnasNet-A1 (+0.7%), MnasNet-A2 (+0.3%) with fewer FLOPs. MoGA-B obtains 75.5%, excelling MobileNetV3 at similar GPU speed. MoGA-C hits a higher accuracy with faster GPU speed, note it is slower on CPU, which otherwise will be treated as inferior by CPU-aware methods. Therefore, it's beneficial to fit models for specific hardware, indicating that even latency on other computing units and FLOPs are not ideal proxies.

MoGA-A makes extensive use of large kernels (4 layers with 7×7), which helps to enlarge receptive fields. Moreover, it mostly places large kernels on the stages with 14×14 input to reduce the latency cost. It also utilizes a large expansion rate after each downsampling stage to retain and to extract more useful features.

Interestingly for MoGA-B, the expansion rates across various layers mimic a sine curve. Like MoGA-A, it utilizes five 7×7 kernels to obtain a large receptive field. To cut down the latency cost, it places most of them in the 14×14 stage. Like FairNAS-A, it selects larger expansion rates right before downsampling operations.

Coincidentally, both MoGA-C and MobileNetV3-large simply contain 3×3 and 5×5 kernels only, even with same amount of such layers. While MobileNetV3-large prefers 5×5 operations in the tail of the model, MoGA-C chooses 3×3 instead. Besides, MoGA-C places 5×5 kernels in the middle and uses less squeeze-and-excitation operations. In such way, it better balances accuracy and latency cost.

5.4 Mobile GPU Awareness Analysis

We benchmark the inference cost for our three models both on mobile CPUs and GPUs. The result is shown in Figure 1. As for mobile GPUs, all models spend most of the time on

---

[4]https://github.com/NVIDIA/apex.git

Methods	Mult-Adds (M)	Params (M)	Lat_g^{SNPE} (ms)	Lat_g^{MACE} (ms)	Lat_c (ms)	Top-1 (%)	Top-5 (%)
MobileNetV2 1.0 (Sandler et al. 2018)	300	3.4	$6.9^\dagger$	$7.0^\dagger$	78	72.0	91.0
MobileNetV3 Large 1.0 (Howard et al. 2019)	219	5.4	$10.8^\star$	$9.5^\star$	$70\ (66)^\star$	$75.0\ (75.2)^\star$	92.2
MnasNet -A1 (Tan et al. 2019)	312	3.9	-	-	78	75.2	92.5
MnasNet-A2 (Tan et al. 2019)	340	4.8	-	-	84	75.6	92.7
FBNet-B (Wu et al. 2019)	295	4.5	-	-	$23^\ddagger$	74.1	-
Proxyless-R Mobile (Cai, Zhu, and Han 2019)	$320^\dagger$	4.0	$7.3^\dagger$	$7.9^\dagger$	$87\ (78)^\dagger$	74.6	92.2
Proxyless GPU (Cai, Zhu, and Han 2019)	$465^\dagger$	7.1	$9.6^\dagger$	$9.8^\dagger$	$126\ (124)^\dagger$	75.1	-
Single-Path NAS (Stamoulis et al. 2019)	365	4.3	-	-	79	75.0	92.2
Once for All (Cai, Gan, and Han 2019)	327	-	-	-	$112^\ast$	75.3	-
FairNAS-A (Chu et al. 2019a)	388	4.6	$9.8^\dagger$	$9.7^\dagger$	104	75.3	92.4
MoGA-A (Ours)	304	5.1	11.8	11.1	101	**75.9**	92.8
MoGA-B (Ours)	248	5.5	10.3	10.0	81	75.5	92.6
MoGA-C (Ours)	221	5.4	9.6	8.8	71	75.3	92.5

Table 3: Comparison of mobile models on ImageNet. $\star$: Our reimplementation. Numbers within the parentheses are reported by its authors, same for below. $\dagger$: Based on its published code. $\ddagger$: Samsung Galaxy S8. $\ast$: Samsung Note8.

2D convolutions. MoGA-A and B spend the second most of the time on depthwise convolutions because they make extensive use of large kernels and expansion rates, whereas MoGA-C pays more attention to elementwise operations instead. Note all MoGA series invest more time on depthwise convolutions, contributing for faster speed and better performance.

It is worth noticing that how models exhibit a different behavior on mobile GPUs than on CPUs. For instance, vanilla convolutions and depthwise convolutions generally share bigger percentages on CPUs than on GPUs, while elementwise operations have a smaller percentage, as seen from Figure 1. Additionally, there is a discrepancy when running the same model with the different inference frameworks as well, which could call for a framework-aware solution, see Table 3. Apart from the mobile framework we use, CPUs and GPUs differ on inherent microarchitectures, which puts hardware-specific requirements a must for the design of neural architectures.

5.5 GPU Cost Analysis with More Mobile Devices

Figure 6: Overall search cost vs. the number of target platforms.

Given a target device, our overall search cost c_{all} can be decomposed into two parts: c_{super} for supernet training (10.5 GPU days) and c_{search} for the NSGA-II pipeline. The latter estimates model accuracy by the supernet evalua-

tor. Notably, there is no need to retrain the supernet when we design neural models for different mobile platforms. In contrast, the cost for most existing NAS methods, such as RL, EA and gradient descent, increases linearly with the number of platforms (Tan et al. 2019; Wu et al. 2019; Cai, Zhu, and Han 2019; Liu, Simonyan, and Yang 2019). For N platforms, our c_{super} is amortized as $\frac{c_{super}}{N}$. When $N \geq 22$, the overall cost c_{all} reduces to less than 2 GPU days per platform. This benefit is better depicted in Figure 6.

Figure 7: Pareto Front of weighted NSGA-II with hierarchical mutator compared with that of a random mutator and of two objectives (accuracy, latency).

6 Results

6.1 Ablation Study

Model Selection For the chosen weighted NSGA-II equipped with hierarchical mutation (Chu et al. 2019b), we

(a) MoGA-A

(b) MoGA-B

(c) MoGA-C

Figure 8: The Architectures of MoGA-A, B, C. Note Ex_Ky_SE means an expansion rate of x for its expansion layer and a kernel size of y for its depthwise convolution layer, SE for squeeze-and-excitation. Grey thick lines refer to downsampling points. Dashed lines separate the stem and end layers from the backbone.

compare it with the random mutation baseline, see in Figure 7. Plenty of models from the baseline are dominated by the hierarchical version, which attests that hierarchical mutation improves searching.

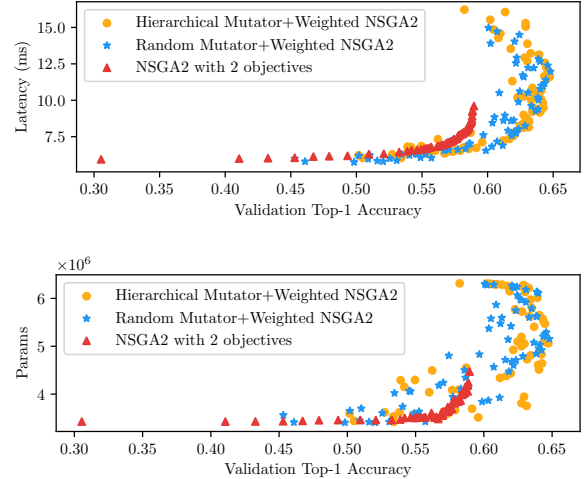We compare the best elitists for Equation 2 and 4, which is shown in the upper part of Figure 7. The Pareto front formed by the two objectives is largely surrounded by those with three.

While FairNAS states that a fair training can boost the rank relationship between the supernet predictor and stand-alone training, it also points out that it can be affected by initialization techniques and suboptimal training hyperparameters. For the latter, we empirically maximize the number of parameters as a compensation bonus.

Figure 9: Histogram on numbers of parameters of models from the last generation of weighted NSGA-II with hierarchical mutator, compared with that of two objectives (accuracy, latency).

Does it matter to use parameters as an objective? Occam's Razor doesn't fit in this case, because in such mobile setting, a neural network is prone to underfitting instead of overfitting. If we consider minimizing the number of parameters, NSGA-II is then at the risk of excluding models with more parameters generation by generation. For evidence, we show the histograms of the number of parameters for the final elitists in Figure 9.

7 Conclusion

To sum up, we have discussed several critical issues in mobile neural architecture design. **First**, we promote the first Mobile GPU-Aware (MoGA) solution, as in production, running networks on mobile GPUs are much preferred. **Second**, we adopt weighted fitness strategy to comfort more valuable objectives like accuracy and latency, other than the number of parameters. **Third**, our total search cost has been substantially reduced to 12 GPU days. Also, the trained supernet is once-for-all since the same supernet caters for all mobile contexts. It requires $o(1)$ search cost when applying to a new mobile device. **Last**, we employ an automated search approach in the search space adapted from MnasNet and MobileNetV3, which generates a new set of state-of-the-art architectures for mobile settings. In particular, MoGA-C hits 75.3% top-1 ImageNet accuracy, which outperforms MobileNetV3 with competing mobile GPU latency at similar FLOPs and an equal number of parameters.

In the future, there will still be continuous interest to squeeze out better performance within limited hardware

bounds, especially on targeted computing units. Also, balancing between architecture diversity and search space size will remain as a major topic, it also poses a challenge for searching algorithms when search space grows enormously.

References

[Abadi et al. 2015] Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. tag: v1.14.0-rc0. Software available from tensorflow.org.

[Bender et al. 2018] Bender, G.; Kindermans, P.-J.; Zoph, B.; Vasudevan, V.; and Le, Q. 2018. Understanding and Simplifying One-Shot Architecture Search. In *International Conference on Machine Learning*, 549–558.

[Cai, Gan, and Han 2019] Cai, H.; Gan, C.; and Han, S. 2019. Once for All: Train One Network and Specialize it for Efficient Deployment. *arXiv preprint. arXiv:1908.09791*.

[Cai, Zhu, and Han 2019] Cai, H.; Zhu, L.; and Han, S. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *International Conference on Learning Representations*.

[Chu et al. 2019a] Chu, X.; Zhang, B.; Xu, R.; and Li, J. 2019a. FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. *arXiv preprint. arXiv:1907.01845*.

[Chu et al. 2019b] Chu, X.; Zhang, B.; Xu, R.; and Ma, H. 2019b. Multi-Objective Reinforced Evolution in Mobile Neural Architecture Search. *arXiv preprint. arXiv:1901.01074*.

[Deb et al. 2002] Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6(2):182–197.

[Deng et al. 2009] Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. IEEE.

[Dong et al. 2018] Dong, J.-D.; Cheng, A.-C.; Juan, D.-C.; Wei, W.; and Sun, M. 2018. DPP-Net: Device-aware Progressive Search for Pareto-optimal Neural Architectures. In *Proceedings of the European Conference on Computer Vision*, 517–531.

[Goyal et al. 2017] Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint. arXiv:1706.02677*.

[Guo et al. 2019] Guo, Z.; Zhang, X.; Mu, H.; Heng, W.; Liu, Z.; Wei, Y.; and Sun, J. 2019. Single Path One-Shot Neural Architecture Search with Uniform Sampling. *arXiv preprint. arXiv:1904.00420*.

[He et al. 2016] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

[Howard et al. 2017] Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. MobileNets: Efficient. Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint. arXiv:1704.04861*.

[Howard et al. 2019] Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. 2019. Searching for MobileNetV3. *arXiv preprint. arXiv:1905.02244*.

[Hu, Shen, and Sun 2018] Hu, J.; Shen, L.; and Sun, G. 2018. Squeeze-and-Excitation Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7132–7141.

[Liu, Simonyan, and Yang 2019] Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*.

[Qualcomm 2019] Qualcomm. 2019. Snapdragon Neural Processing Engine SDK. https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk, version: 1.27.1.382.

[Sandler et al. 2018] Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

[Srivastava et al. 2014] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

[Stamoulis et al. 2019] Stamoulis, D.; Ding, R.; Wang, D.; Lymberopoulos, D.; Priyantha, B.; Liu, J.; and Marculescu, D. 2019. Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours. *arXiv preprint. arXiv:1904.02877*.

[Tan et al. 2019] Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; and Le, Q. V. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[Wu et al. 2019] Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; and Keutzer, K. 2019. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *The IEEE Conference on Computer Vision and Pattern Recognition*.

[Xiaomi 2018] Xiaomi. 2018. Mobile AI Compute Engine. https://github.com/XiaoMi/mace, commit hashtag: 03362fa0.

[Yang et al. 2018] Yang, T.-J.; Howard, A.; Chen, B.; Zhang, X.; Go, A.; Sandler, M.; Sze, V.; and Adam, H. 2018. NetAdapt: Platform-Aware Neural Network. Adaptation for

Mobile Applications. In *Proceedings of the European Conference on Computer Vision*, 285–300.

[Zhang et al. 2018] Zhang, X.; Zhou, X.; Lin, M.; and Sun, J. 2018. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *The IEEE Conference on Computer Vision and Pattern Recognition*.

[Zoph et al. 2018] Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8697–8710.