Discovering Multi-Hardware Mobile Models via Architecture Search

Grace Chu Okan Arikan Gabriel Bender Weijun Wang Achille Brighton
Pieter-Jan Kindermans Hanxiao Liu Berkin Akin Suyog Gupta Andrew Howard
Google LLC
{cxy, okana, gbender, weijunw, aib, pikinder, hanxiaol, bakin, suyoggupta, howarda}@google.com

Abstract

Hardware-aware neural architecture designs have been predominantly focusing on optimizing model performance on single hardware and model development complexity, where another important factor, model deployment complexity, has been largely ignored. In this paper, we argue that, for applications that may be deployed on multiple hardware, having different single-hardware models across the deployed hardware makes it hard to guarantee consistent outputs across hardware and duplicates engineering work for debugging and fixing. To minimize such deployment cost, we propose an alternative solution, multi-hardware models, where a single architecture is developed for multiple hardware. With thoughtful search space design and incorporating the proposed multi-hardware metrics in neural architecture search, we discover multi-hardware models that give state-of-the-art (SoTA) performance across multiple hardware in both average and worse case scenarios. For performance on individual hardware, the single multi-hardware model yields similar or better results than SoTA performance on accelerators like GPU, DSP and EdgeTPU which was achieved by different models, while having similar performance with MobilenetV3 Large Minimalistic model on mobile CPU. [1]

1. Introduction

Developing efficient on-device neural networks has become an important topic in computer vision with many real-world applications. Having models that can be fully deployed on device not only enables fast, real-time results, but also avoids exposing personal data to public servers.

Given the resource constraints of a portable device, such as latency, energy and memory footprint, on-device models need to be fast and small. While the number of multiply-and-add operations (MAdds) and the number of parameters have been widely used to optimize efficient models [11, 17, 26, 30, 33], recent research has shown that improvements on theoretical MAdds or number of parameters do not always translate into better latency on real hardware, and can actually be counterproductive in some cases [25,28]. Thus, optimizing directly on latency measurements becomes important when we want to find a fast model on device [3, 14].

Unlike MAdds or the number of parameters, latency is highly dependent on hardware (and its associated software). A neural network optimized for a specific hardware platform may perform sub-optimally on a different one in terms of inference efficiency. Therefore, different models have been developed to achieve the best performance on each individual hardware [4, 7, 8, 27].

However, for application developers who want to deploy their application on multiple hardware, using different single-hardware models for each hardware introduces deployment overhead from multiple aspects. For example, one needs to tune multiple models and dependent components in the system to guarantee consistent application outputs across hardware. In addition, duplicated debugging and fixing work are needed for any issue or update of the application and the complexity increases linearly with the number of deployed hardware.

To solve these problems, we propose multi-hardware models, where a single model is developed by optimizing on multiple hardware. It minimizes model deployment cost while still performs reasonably good on each targeted hardware. The contributions of this paper can be summarized as follows.

- It is the first work exploring the feasibility of multi-hardware models.
- It proposes a complementary search method that can be used by any existing neural architecture search (NAS) algorithms to find multi-hardware models.
- It proposes multi-hardware metrics to evaluate overall efficiency among multiple hardware.

[1] Multi-hardware models (Multi-AVG and Multi-MAX) are available at https://github.com/google-research/google-research/tree/master/tunas and https://github.com/tensorflow/models/blob/master/official/vision/beta/modeling/backbones/mobilenet.py

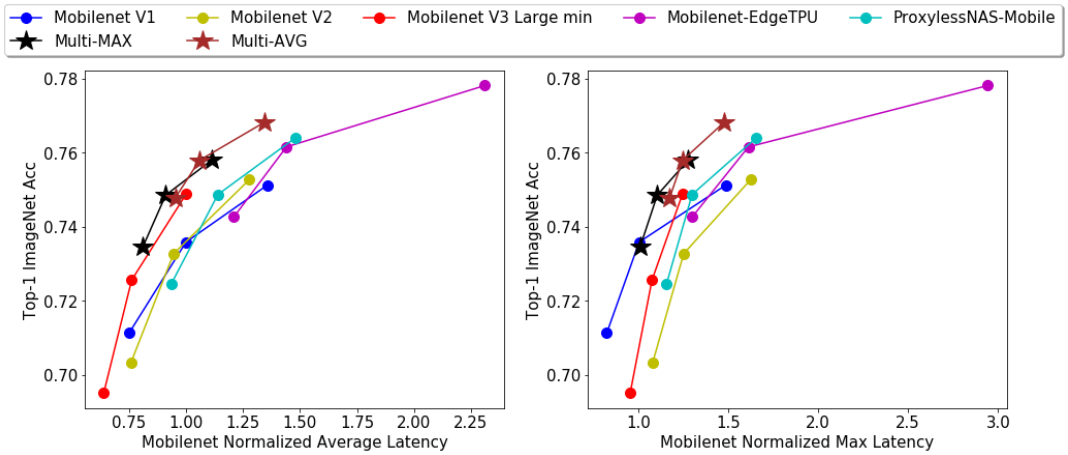


Figure 1: ImageNet test accuracy v.s. average latency (Mobilenet Normalized Average Latency in the left figure) and worst case latency (Mobilenet Normalized Max Latency in the right figure) over 5 hardware in Pixel4: CPU float, CPU uint8, GPU, DSP and EdgeTPU. See Section 6.2 for details of how the average and worst case latency on x-axis are calculated. Multi-MAX and Multi-AVG are discovered multi-hardware models which yield the SoTA accuracy-latency tradeoff w.r.t. both average and worst case performance.

- The discovered multi-hardware models via proposed method achieve SOTA performance across targeted hardware and can be generalized to other hardware as well.

2. Related Work

Hardware-aware neural architecture designs have been a popular research area in recent years. NetAdapt [28] uses empirical latency tables of the target hardware to greedily adapt a model to its highest accuracy under a target latency constraint. MnasNet [25] also uses latency tables, but applies reinforcement learning to do hardware-aware architecture search. FBNet [27] and ChamNet [8] find the best architecture for targeted hardware by incorporating latency table and resource predictive models in architecture search respectively. MoGA [7] optimizes a model for GPU. Once-for-all [4] proposes a pre-trained super-model where different sub-models can be extracted for different hardware.

Neural architecture search (NAS) has been widely used in hardware-aware architecture designs as the unpredictable hardware performance of a model makes it challenging to optimize models by hand [3,19,22,25,29,32,33]. This technique uses reinforcement learning [32], evolutionary search [23], differentiable search [20, 21] or other algorithms [10] to find the best neural architecture according to a predefined reward function which incorporates both model performance and hardware efficiency [19].

3. Why Multi-Hardware Models

Existing hardware-aware architecture designs have primarily focused on single-hardware models which aim to deliver the best model for each single hardware. With the recent advance of NAS [29] and smart training algorithms [4], the cost of architecture search and model training have been significantly reduced. However, the outcome of N models for N hardware introduces much overhead for application developers who want to deploy the application on multiple hardware.

First overhead comes from the component level tuning when deploying the model. For example, when using the classification results to suggest a user action, like scanning text in the image, or blurring background of a portrait photo, one needs to tune a score threshold to determine when to give that suggestion. Then, if N models are used for N hardware, multiple score thresholds need to be tuned to ensure consistent performance across hardware.

Moreover, having different models on different hardware for the same application makes it hard to keep consistent model performance across hardware. Besides the overhead of tuning multiple models to have the same accuracy, it is almost impossible to make sure that all models give the exactly same output for every image it received.

In addition, when unexpected performance occurs, such as the model false triggers or misses certain images, one needs to first determine whether it is a universal issue, or it only happens on some hardware as different models are used. Every debugging and fix step needs to be done N times for each different model used for this application.

Last but the not the least, having a single model for multiple hardware on the same device, like CPU and EdgeTPU on Pixel4, enables seamless transition of workload from one hardware onto another other at runtime. In addition, only one model needs to be stored in this case.

In summary, multi-hardware model is a solution to minimize deployment complexity, a factor that has been largely ignored in developing single-hardware models. We will show in this paper that, despite the challenges, with proper

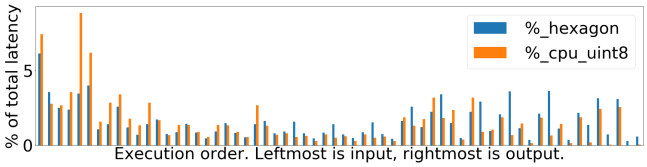


Figure 2: Per layer profiling of MobileNetV3 minimalistic on Pixel4 CPU uint8 and DSP (Hexagon). The leftmost is the input layer while the output layer is on the right.

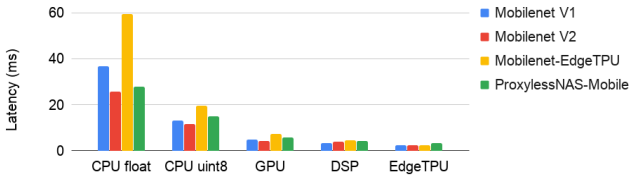


Figure 3: Latency of different models on different hardware in Pixel4 phone.

balance among hardware in search space design and metric definition, multi-hardware models can still yield decent results across a wide range of hardware.

4. Challenges of Multi-Hardware Models

Diverse design preferences: Due to the unique design of each hardware, their specialties are usually different, which may yield different directions of optimization. To demonstrate this, we take the MobilenetV3 Large minimalistic (min) model [1] and run per layer profiling on CPU (uint8) and DSP (Qualcomm 855 Hexagon) to get the latency percentage of each layer over the whole model (Figure 2). On CPU, a larger fraction of the model's total latency comes from the earlier layers of the network, while on DSP, a larger fraction of the total latency comes from the later layers. Therefore, when optimizing on CPU, one may focus mainly on the early layers, while later layers may gain more attention when optimizing for DSP.

Different supported operations: While new operations and model blocks have been proposing to improve accuracy and latency trade-offs [14–16, 31], not all of them are equally efficient on different hardware. For example, the depthwise separable convolution that was proposed in MobileNet [15] to replace the regular convolution reduces MAdds and makes model inference more efficient on CPU. However, a decrease in MAdds does not always lead to a decrease in on-device latency, especially for accelerators which have been optimized specifically to handle large number of computations as long as they follow certain pattern [6]. For example, [12] indicates that EdgeTPU favors regular convolution over depthwise separable convolution in certain layers of the model as the former can utilize the hardware resources better and gives better latency-accuracy trade-offs. This makes it hard to manually decide what operation to use at which layer if we want to have a single model that works well on both CPU and EdgeTPU.

Diverse latency relationship: It is well known that a model has different latencies when running on different hardware, but is the relationship similar for all models? That is, if a model runs 2× faster than another on one hardware, 1) will it still run faster on another hardware? 2) if faster, will it still be 2× faster? To answer this questions, we take four mobile models, MobileNetV1 [15], MobileNetV2 [24], MobileNet EdgeTPU [12], ProxylessNAS mobile [5], and benchmark them on different hardware to see whether the latency ratio among them are the same. Figure 3 shows the results, where the latency ratio among different hardware are obviously different for each model. Furthermore, while EdgeTPU runs MobileNet-EdgeTPU model faster than ProxylessNAS-Mobile, CPU executes ProxylessNAS-Mobile faster than MobileNet-EdgeTPU.

5. Problem Formulation

Due to the challenges listed in previous section, manually handcrafting a single model to accommodate the traits of multiple hardware is very difficult. Instead, we leverage neural architecture search to find multi-hardware models in this paper, where a multi-hardware search space is proposed to be compatible to all examined hardware, and two metrics are introduced to compare models under multi-hardware environment.

5.1. Multi-Hardware Search Space

Let $\mathcal{H} = \{H_1, H_2, ..., H_N\}$ be the set of hardware we want to optimize for. For $0 < i \leq N$, S_i denotes the set of neural network architectures that H_i can support, i.e., the entire network can fully run on this hardware without falling back to another slower hardware. Then, a multi-hardware search space, denoted as $S^{\mathcal{H}}$, is a set of neural network architectures that belongs to the intersection of supported architectures of the set of examined hardware. Mathematically,

$$S^{\mathcal{H}} \subseteq S_1 \cap S_2 \cap \cdots \cap S_N. \quad (1)$$

Note that, we allow the multi-hardware search space to be a subset of instead of equal to the intersection of all supported architectures, by taking into account the practical size limit for efficient architecture search.

5.2. Multi-Hardware Metrics

In order to find a single model optimized for multiple hardware, we need metrics to determine what is a better model. Without loss of generality, we examine models' accuracy and latency to compare different models, as the Pareto optimal on these two metrics has been broadly used in single-hardware architecture optimizations [3, 15, 25]. Specifically, model a is better than model b in single-hardware optimization iff

$$L_a < L_b \text{ when } A_a = A_b, \tag{2}$$

where (L_a, A_a) and (L_b, A_b) are the (latency, accuracy) measurements of model a and b on the examined single hardware, respectively.

When considering multi-hardware optimization, the biggest challenge is how to compare models given their latency measurements on various hardware. Let $\mathcal{L}_a = \{L_{a,1}, L_{a,2}, \ldots, L_{a,N}\}$ denote the latency of model a on $\mathcal{H}$, similarly for model b. We need some overall metric function $f^{\mathcal{H}}(\cdot)$ such that, if

$$f^{\mathcal{H}}(\mathcal{L}_a) < f^{\mathcal{H}}(\mathcal{L}_b) \text{ when } A_a = A_b, \tag{3}$$

we say that model a is better than model b.

As shown in Section 4, latency on different hardware may have different scales, thus $L_{a,i}$ needs to be normalized before any calculation. In this paper, we propose two intuitive metrics to measure the average and worst case performance of a model on multiple hardware. Specifically, the normalized average latency over $\mathcal{H}$ is defined as

$$f^{\mathcal{H}}_{avg}(\mathcal{L}_a) \triangleq \frac{1}{N}\sum_{i=1}^{N}\frac{L_{a,i}}{C_i}, \tag{4}$$

and the normalized max latency over $\mathcal{H}$ is defined as

$$f^{\mathcal{H}}_{max}(\mathcal{L}_a) \triangleq \max_i\left(\frac{L_{a,i}}{C_i}\right), \tag{5}$$

where $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$ are normalization factors. While there are many ways of choosing $\mathcal{C}$, we discuss two common cases as follows.

1. $\mathcal{C}$ can be chosen as the latency of a reference model on $\mathcal{H}$ to represent the latency scaling relationship among hardware. In addition, if the normalized average latency of a model is 0.5, it implies that, on average, the model runs in half of the time of the reference model.
2. On top of the natural latency scaling difference among hardware, one can further re-weight $\mathcal{C}$ with the importance of each hardware in $\mathcal{H}$. An extreme case would be to set all norm factors to be ∞ except one $C_1 = 1$. Then $f^{\mathcal{H}}(\mathcal{L}_a) = L_{a,1}$, which regresses the problem to a single hardware optimization.

Remark: This paper mainly focuses on mobile models as cross device application is the most common use case of multi-hardware models. However, the methodology introduced here can be easily generalized to discover multi-hardware server sized models when needed.

6. Case Study

With the essential concepts defined above, we use an on-device case study to demonstrate how to find multi-hardware models via architecture search. Here, we consider five hardware inside a Pixel4 phone: CPU float, CPU uint8, GPU (Qualcomm Adreno 640), DSP (Qualcomm Snapdragon 855), EdgeTPU (Google). We choose this set of hardware because

- it covers various types of hardware from different manufacturers for mobile;
- the obtained multi-hardware model is useful in application: Because it performs well on all hardware on Pixel4, it can be used as a default model for any application deployed on Pixel4 regardless of its particular inference hardware.

We use TuNAS [3] as the NAS infrastructure to use in this paper, while the proposed method can be applied on many existing NAS algorithms to get multi-hardware models.

6.1. Multi-Hardware Search Space on Device

In order to optimize for multiple hardware, multi-hardware search space needs to be both exclusive enough so that each searched operation is supported by all examined hardware, and inclusive enough so that it searches over a variety of effective (and supportive) operations for each examined hardware.

- We center the search space at MobilenetV3 Large model's architecture as it is one of the SoTA mobile models.
- We remove squeeze and excite (SE) and h-swish because they are not supported in EdgeTPU.
- Filter sizes are adjusted to be integer multiples of 32 due to a preference of DSP [2].
- Similar to TuNAS MobilenetV3 Large search space [3], we search over the number of repeated blocks per stage from {1, 2, 3, 4}, the expansion ratio from {1, 2, 3, 4, 5, 6}, and the input/output filter size ratios from {0.5, 0.625, 0.75, 1.0, 1.25, 1.5, 2.0}.
- We do not search the input/output filter sizes in the model head because in early experiments we found that, the RL controller was biased towards using large numbers of filters in the model head which blew up the model size but with marginal accuracy improvement.
- Each block can choose either regular inverted bottleneck or fused inverted bottleneck, which replaces expansion 1x1 convolution (conv) and depthwise conv with a single regular conv, as it has been shown to be effective for EdgeTPU [13].
- Convolution kernels can choose from 3x3 and 5x5 because bigger kernels than 5x5 are not widely supported by DSPs.

6.2. Mobilenet Normalized Avg and Max Metrics

Given we are optimizing for mobile models, we choose to use MobilenetV1 as the reference model to calculate the overall metrics, i.e., use its latency on the examined hardware as the normalization factors $\mathcal{C}$ in equation (4) and (5). More specific reasons to choose this reference model are:

- existed for a few years and yet still widely used;

- publicly available in multiple formats (TFLite, Caffe, etc.) for ML researchers to run benchmarks with;
- simple enough that can run on a wide variety of hardware.

We do not have a particular preference on having better performance on some of the optimized hardware, so no extra re-weights were assigned to these normalization factors.

The reward function used in architecture search needs to be adjusted with these new metrics too. In TuNAS, single hardware search maximizes the following reward function [3]:

$$r(\alpha) = A(\alpha) + \beta\left|\frac{L(\alpha)}{L_0} - 1\right|, \quad (6)$$

where α represents an architecture, $r(\cdot)$, $A(\cdot)$ and $L(\cdot)$ are reward, accuracy and latency of the architecture, respectively. $|\cdot|$ is absolute function. L_0 denotes latency target. $\beta < 0$ is an application-specific constant.

To search for multi-hardware models, the reward function becomes

$$\begin{aligned} r_{avg}(\alpha) &= A(\alpha) + \beta|f^{\mathcal{H}}_{avg}(\alpha) - 1| \\ &= A(\alpha) + \beta\left|\frac{1}{N}\sum_{i=1}^{N}\frac{L_i(\alpha)}{C_i} - 1\right|, \end{aligned} \quad (7)$$

when optimizing for average performance; and

$$\begin{aligned} r_{max}(\alpha) &= A(\alpha) + \beta|f^{\mathcal{H}}_{max}(\alpha) - 1| \\ &= A(\alpha) + \beta\left|\max_i\frac{L_i(\alpha)}{C_i} - 1\right|, \end{aligned} \quad (8)$$

when optimizing for worst case performance. Note that when optimizing for average performance, the reward function implies a prior that the searched architecture should, on average, have latency close to that of the reference model MobilenetV1.

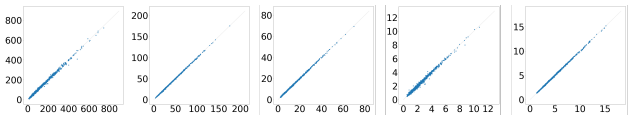
7. Experiments

7.1. Experimental Setup

Latency benchmarks: In this paper, three phones with totally ten different hardware are used in either searching for the multi-hardware model or evaluating the model on unsearched hardware. The driver's versions for these phones are: Pixel4 uses QQ1B.200205.003; Pixel3 uses QQ1A.200205.002; MediaTek Dimensity 1000 5G uses QP1A.190711.020.

The delegates used for accelerators are: GPU's latency is obtained from Jet delegate using OpenCL; DSP's latency is from Hexagon delegate which directly calls the Qualcomm's binary with less overhead than Android NNAPI; EdgeTPU's latency and APU's latency are obtained by using NNAPI delegate.

TF-Lite models with single-thread and batch size of 1 are used to get all benchmarking results. When getting CPU's latency, only the large cores were used. When benchmarking on CPU uint8, DSP, EdgeTPU and APU, where quantized models are needed, fake quantization is applied [18][2].



(a) CPU float (b) CPU uint8 (c) GPU (d) EdgeTPU (e) DSP

Figure 4: True cost/latency (x-axs) v.s. predicted cost/latency (y-axis) on unseen architectures from the search space for each of the trained cost models. Latency numbers are in millisecond unit.

Cost models: To save hardware communication cost in search, we pre-train linear cost models for target hardware so that the latency of any architecture sampled during search can be inferred from them. When training cost models, 9K pairs of (architecture, latency) data were used to train the cost model for each hardware, except for EdgeTPU where we used 20K to achieve the similar quality. Figure 4 shows that trained cost models have good correlations between predicted and true latency on unseen architectures.

Architecture search and training: We use ImageNet data [9] to search, train and evaluate. Input resolution is 224×224 and ResNet data preprocessing is used. Cloud TPU v2-32 is used in both search and standalone model training, where per core batch size is 128. For standalone model training, we use the same hyper-parameters as in [3], where 0.25 is set as the dropout rate when training models for 360 epochs to get the test accuracy.

For architecture search, we increase the search length from what was used in [3] as it shows some benefits when optimizing DSP and TPU. Specifically, 1) per core learning rate is halved from 0.0825 to 0.04125; 2) the warmup time where only shared model weights are trained without updating RL controller is increased from 25% to 50%; 3) we search for 360 epochs instead of 90 epochs.

Baseline models: To make fair comparisons, we re-implemented all baseline models in the training setup so that they use the same hyper-parameters with multi-hardware models. We have found similar trends as [3] that the re-implemented MobilenetV1 and MobilenetV2 have higher accuracy numbers than the published ones in the original papers.

7.2. Main Results

We conduct two multi-hardware architecture searches using TuNAS to find models that perform well on all hardware in Pixel4, regarding average performance and worst case performance, respectively. Both of them use the same multi-hardware search space proposed in Section 6.1. Reward functions used in these two searches are equation (7)

[2]In order to have consistency across multiple hardware, accuracy in this paper is always measured on the float model.

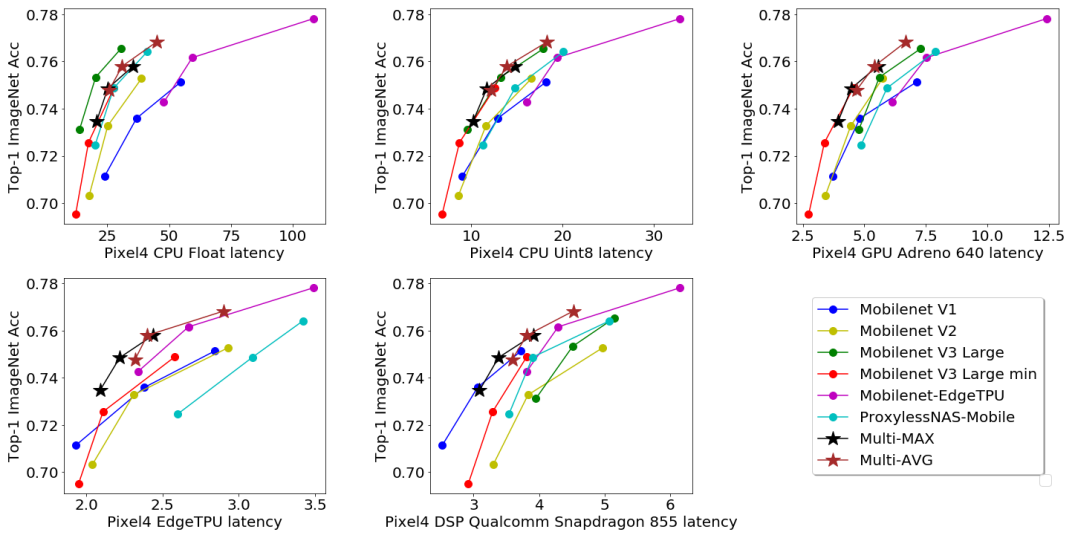


Figure 5: Accuracy-latency trade-offs of multi-hardware model v.s. baseline models on each optimized hardware. The horizontal axis is end-to-end real time latency benchmarks in milliseconds and the vertical axis is test accuracy.

Table 1: Performance of multi-hardware models comparing with baseline models on Pixel4 phone. 'wm' is short for 'width multiplier'. '×' means that the model is not supported on that hardware. 'MN-Norm' is the mobilenet normalized metrics proposed in Section 6.2 where lower is better. Top-1 item within each column has been marked bold.

Model	wm	Accu (%)	Params (M)	MAdds (M)	CPU		GPU	DSP	EdgeTPU	MN-Norm	
					float	uint8				avg	max
MobilenetV1	1.25	75.1	6.25	883	54.7	18.2	7.12	3.72	2.84	1.36	1.49
MobilenetV2	1.25	75.3	5.01	487	38.8	16.6	5.74	4.97	2.93	1.28	1.62
MobilenetV3Large	1.0	75.3	5.45	**217**	**20.3**	13.2	5.61	4.51	×	×	×
MobilenetV3Large min	1.25	74.9	5.73	346	27.7	12.6	4.56	3.81	2.58	1.00	1.25
ProxylessNAS-Mobile	1.0	74.9	**4.05**	321	27.6	14.8	5.92	3.90	3.09	1.14	1.30
Mobilenet-EdgeTPU	1.0	**76.2**	**4.05**	991	59.3	19.4	7.52	4.29	2.67	1.44	1.61
Multi-AVG	1.0	75.8	4.91	433	31.0	13.9	5.40	3.81	2.40	1.06	1.25
Multi-MAX	1.0	74.9	4.39	349	25.2	**11.7**	**4.47**	**3.38**	**2.22**	**0.91**	**1.10**

and (8) respectively. β is set to -0.07.

Figure 5 shows the accuracy-latency pareto curves of the obtained multi-hardware models compared with (re-implemented) baseline models. 'Multi-MAX' and 'Multi-AVG' models are the results from searching over average metric and max metric, respectively. Each model has three points in the plot denoting the performance for width multiplier 0.75, 1 and 1.25.

On CPU float, except MobilenetV3 Large model, which is particularly optimized for CPU but not supported by EdgeTPU, multi-hardware models perform the best among all other baseline models. On the other four hardware platforms, multi-hardware models achieves SoTA trade-off between accuracy and latency.

Specifically, on CPU uint8 and GPU, multi-hardware models perform similarly with MobilenetV3 Large min. However. they are much better than this baseline model on EdgeTPU and DSP. After updating MobilenetV1's accuracy with the better hyper-parameters, we found that this is the best baseline model on DSP. However, multi-hardware models still outperform MobilenetV1 when scaling up and the gap is much larger on other hardware. When comparing with Mobilenet-EdgeTPU, which is optimized particularly for EdgeTPU, multi-hardware models give better results on both EdgeTPU and other hardware.

Figure 1 in the Introduction shows the overall performance where the normalization factors were taken as the latency of MobilenetV1 on examined hardware. As expected, the multi-hardware models are better than all baseline models in both average and worst case performance.

Numerically, we compare multi-hardware models with baseline models on similar accuracy range in Table 1. Multi-MAX model runs the fastest on all examined hardware except on CPU float where it still ranks the second, while its accuracy is only 0.4% lower than the second highest number in baseline models achieved by MobilenetV2 and MobilenetV3 Large. The top accuracy is achieved by Mobilenet-EdgeTPU, which is only 0.4% higher than Multi-AVG model but its latency on CPU float is almost 2× slower and MAdds is 2.29× more. While MobilenetV3

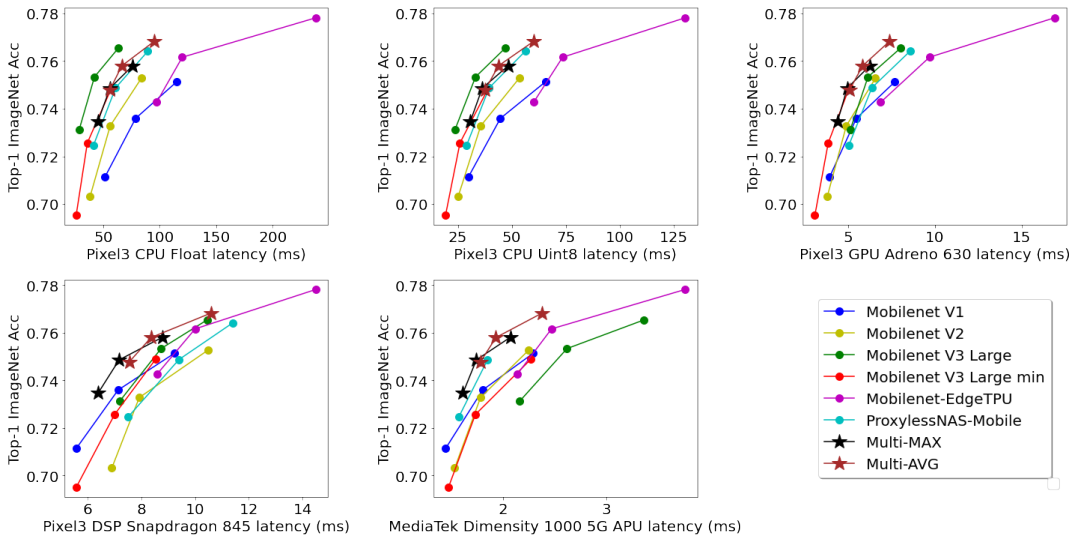


Figure 6: Accuracy-latency trade-offs on unsearched hardware[3], where the discovered Multi-MAX and Multi-AVG models are Pareto-optimal on all hardware except CPU.

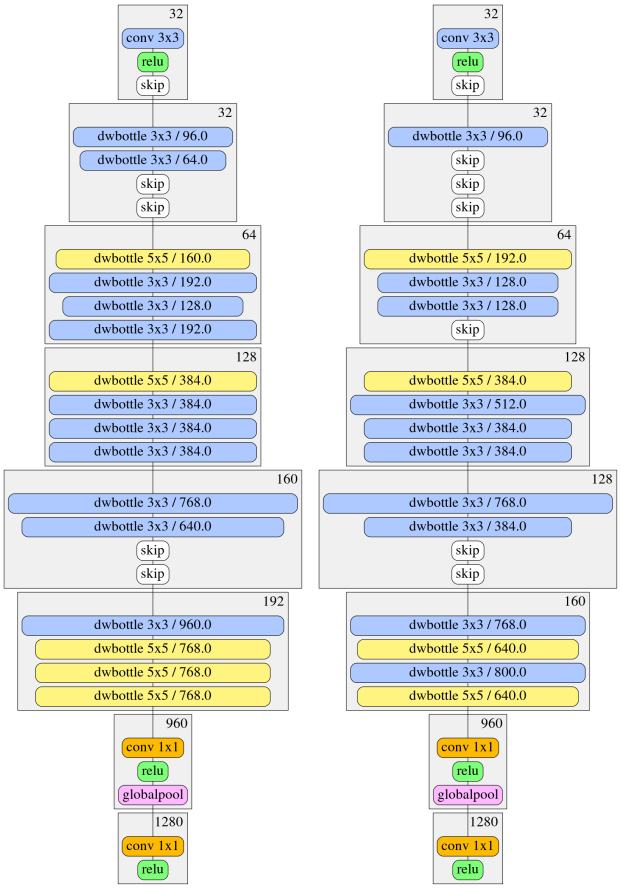


Figure 7: Model visualization of Multi-AVG (left) and Multi-MAX (right) models.

Large achieves the best CPU float latency, it is not supported on EdgeTPU, and runs 18% slower than Multi-AVG on DSP while also 0.5% lower on accuracy. MobilenetV2 is 0.5% lower on accuracy than Multi-AVG and runs also slower on all examined hardware: 25% slower on CPU float and 30% slower on DSP.

To show how multi-hardware models generalize on unsearched hardware, we evaluate their performance on various hardware of Pixel3 and MediaTek phones in Figure 6. Without optimizing for, multi-hardware models achieve SoTA performance on MediaTek accelerators. For Pixel3 hardware, multi-hardware models show similar trends as they are on Pixel4: yield the best results on Pixel3 GPU and DSP while being the second best results on CPU float. However, on CPU uint8, multi-hardware models do not give the best results on Pixel3 as they do for Pixel4. This is because Pixel3 CPU float and uint8 are similar hardware, while Pixel4 CPU uint8 has been particularly accelerated and performs much different from CPU float. The observation above demonstrates that one may only need to pick representative hardware to optimize, as the multi-hardware model will most likely have similar performance on closely related hardware, such as the same type of chips with different versions.

Figure 7 shows the visualization of the discovered multi-hardware models. The number inside each grey box is the output filter size (number of channels) for that stage, which is applied to each of the colored blocks inside. For example, in Multi-AVG model, second box from top, 'dwbottleneck 3x3 / 96.0' indicates an inverted bottleneck block where the kernel size of depthwise conv is 3x3, the filter size of the expanded layer is 96 and the output filter size is 32. 'skip' denotes an identity operation. The $\times 2$ strides on resolution are taken at the same places as the centered model MobilenetV3: at the beginning of 1st, 2nd, 3rd, 4th, 6th stages respectively.

Both of the multi-hardware models have light early lay-

[3]ProxylessNAS-Mobile only has two data points on MediaTek hardware as the model with width multiplier 1.25 is not fully supported by this hardware.

Table 2: Performance of single-hardware models. 'Single-DSP' is the searched model only optimized for DSP. Top-1 item within each column is marked bold.

Model	Accu (%)	CPU		GPU	DSP	EdgeTPU	MN-Norm	
		float	uint8				avg	max
Single-CPU float	**76.5**	39.6	18.0	6.23	4.52	3.32	1.33	1.48
Single-CPU uint8	76.2	38.6	**13.9**	5.85	3.71	2.55	**1.13**	**1.21**
Single-GPU	76.0	**33.6**	15.6	**5.46**	4.10	2.68	1.15	1.34
Single-DSP	76.3	46.7	15.6	6.88	**3.42**	**2.44**	1.21	1.43
Single-EdgeTPU	76.0	44.0	15.4	6.03	3.98	2.47	1.20	1.30

Table 3: Compare computation cost and performance of multi-hardware search and single-hardware search. One unit of search cost is ∼90 hours of Cloud TPU v2-32 usage.

Model	Search Cost	Accu (%)	CPU		GPU	DSP	EdgeTPU	MN-Norm	
			float	uint8				avg	max
Single-Hardware	5×	76.2	38.6	13.9	5.85	3.71	2.55	1.13	1.21
Multi-Hardware	1×	75.8	31.0	13.9	5.40	3.81	2.40	1.06	1.25

ers and heavy later layers, 5x5 kernels also appear later in the network. This indicates that multi-hardware models tend to move the computation to later layers where accelerators may gain more computation advantage. In addition, the observation that fused inverted bottlenecks are not chosen for multi-hardware models indicates that operations only effective on a small subset of examined hardware are not preferable for multi-hardware optimization.

7.3. Multi-Hardware Search v.s. Single-Hardware Search

To show the effectiveness of multi-hardware search comparing with single-hardware search, we conduct single-hardware search, by using reward function in equation (6), for each optimized hardware on the same multi-hardware search space with the multi-hardware search. The results are shown in Table 2. Note that we choose to compare the direct search results without scaling because that represents the best performance from each search. The following comparisons will implicitly consider the minor difference on accuracy.

As expected, the best performance on CPU uint8, GPU and DSP was obtained from searching for corresponding hardware. Since the multi-hardware search space does not contain SE and h-swish which are particularly effectively on CPU float, Single-CPU float model searched on this search space only gives sub-optimal performance. Single-DSP model gives similar or even better results on EdgeTPU than Single-EdgeTPU model does, which may due to the high correlation between DSP and EdgeTPU. By checking the overall latency metrics, Single-CPU uint8 model (highlighted) gives the best results in all single-hardware models.

Taking the best single-hardware search results and comparing with the multi-hardware model (we take Multi-AVG here as they have similar accuracy) in Table 3, we can see that the best single-hardware model performs on par, or even slightly better than multi-hardware model on normalized max metric. However, which hardware would give the best model is unknown until we get all single-hardware models. Therefore, though single-hardware search might get slightly better results than multi-hardware search, its computation cost is $N\times$ of that needed for multi-hardware search, which scales linearly with the number of hardware one wants to optimize on.

8. Conclusions

In this paper, we introduced an important but large ignored factor in hardware-aware neural architecture designs for applications that may be deployed on multiple hardware: model deployment cost. Taking this factor into consideration, we proposed a solution that minimized deployment cost, as well as development cost, while still achieving reasonably good performance across wide variety of hardware. Specifically, the concept of multi-hardware search space that is compatible with all examined hardware has been introduced, as well as the normalized average and max metrics to compare models' overall performance among multiple hardware. The multi-hardware models found in our experiments give SOTA performance on a majority of the examined hardware, as well as closely correlated un-searched hardware. Comparing with single-hardware searches which have to be applied on each target hardware separately, multi-hardware search gives comparable overall performance in a single search/train session.

Acknowledgements: We would like to thank Mark Sandler, Jaeyoun Kim, Jiahui Yu, Mingxing Tan, Ruoming Pang, Quoc V. Le, Hartwig Adam for helpful feedback and discussion; Cheng-Ming Chiang, Guan-Yu Chen, Koan-Sin Tan, Yu-Chieh Lin from MediaTek for useful guidance on MediaTek benchmarks; and QCT (Qualcomm CDMA Technologies) AI SW team for feedback on optimization.

References

[1] Mobilenet github. github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet. 3

[2] Snapdragon neural processing engine sdk reference guide. https://developer.qualcomm.com/docs/snpe/limitations.html. 4

[3] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V. Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1, 2, 3, 4, 5

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020. 1, 2

[5] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *CoRR*, abs/1812.00332, 2018. 3

[6] Yu-Hsin Chen, Joel S. Emer, and Vivienne Sze. Eyeriss v2: A flexible and high-performance accelerator for emerging deep neural networks. *CoRR*, abs/1807.07928, 2018. 3

[7] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Moga: Searching beyond mobilenetv3. *CoRR*, 1908.01314, 2019. 1, 2

[8] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, Peter Vajda, Matt Uyttendaele, and Niraj K. Jha. Chamnet: Towards efficient network design through platform-aware model adaptation. *CoRR*, abs/1812.08934, 2018. 1, 2

[9] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009. 5

[10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey, 2018. 2

[11] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. Squeezenext: Hardware-aware neural network design. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018. 1

[12] Suyog Gupta and Berkin Akin. Accelerator-aware neural network design using automl. *CoRR*, abs/2003.02838, 2020. 3

[13] Suyog Gupta and Mingxing Tan. Efficientnet-edgetpu: Creating accelerator-optimized neural networks with automl. *Google AI Blog*. 4

[14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019. 1, 3

[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 3

[16] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017. 3

[17] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. 1

[18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017. 5

[19] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021. 2

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. 2

[21] Renqian Luo, Fei Tian, Tao Qin, and Tie-Yan Liu. Neural architecture optimization. *CoRR*, abs/1808.07233, 2018. 2

[22] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 2

[23] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. 2

[24] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *CoRR*, abs/1801.04381, 2018. 3

[25] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. 1, 2, 3

[26] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. 1

[27] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *CoRR*, abs/1812.03443, 2018. 1, 2

[28] Tien-Ju Yang, Andrew G. Howard, Bo Chen, Xiao Zhang, Alec Go, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *CoRR*, abs/1804.03230, 2018. 1, 2

[29] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Jianyuan Guo, Wei Zhang, Chao Xu, Chunjing Xu, Dacheng Tao, and Chang Xu. Hournas: Extremely fast neural architecture search through an hourglass lens, 2020. 2

[30] Jiahui Yu and Thomas S. Huang. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *CoRR*, abs/1903.11728, 2019. 1

[31] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 3

[32] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. 2

[33] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. 1, 2