MnasNet: Platform-Aware Neural Architecture Search for Mobile

Mingxing Tan[1] **Bo Chen**[2] **Ruoming Pang**[1] **Vijay Vasudevan**[1] **Mark Sandler**[2] **Andrew Howard**[2] **Quoc V. Le**[1]
[1]Google Brain, [2]Google Inc.
{tanmingxing, bochen, rpang, vrv, sandler, howarda, qvl}@google.com

Abstract

Designing convolutional neural networks (CNN) for mobile devices is challenging because mobile models need to be small and fast, yet still accurate. Although significant efforts have been dedicated to design and improve mobile CNNs on all dimensions, it is very difficult to manually balance these trade-offs when there are so many architectural possibilities to consider. In this paper, we propose an automated mobile neural architecture search (MNAS) approach, which explicitly incorporate model latency into the main objective so that the search can identify a model that achieves a good trade-off between accuracy and latency. Unlike previous work, where latency is considered via another, often inaccurate proxy (e.g., FLOPS), our approach directly measures real-world inference latency by executing the model on mobile phones. To further strike the right balance between flexibility and search space size, we propose a novel factorized hierarchical search space that encourages layer diversity throughout the network. Experimental results show that our approach consistently outperforms state-of-the-art mobile CNN models across multiple vision tasks. On the ImageNet classification task, our MnasNet achieves 75.2% top-1 accuracy with 78ms latency on a Pixel phone, which is 1.8× faster than MobileNetV2 [29] with 0.5% higher accuracy and 2.3× faster than NASNet [36] with 1.2% higher accuracy. Our MnasNet also achieves better mAP quality than MobileNets for COCO object detection. Code is at https://github.com/tensorflow/tpu/tree/master/models/official/mnasnet.

1. Introduction

Convolutional neural networks (CNN) have made significant progress in image classification, object detection, and many other applications. As modern CNN models become increasingly deeper and larger [31, 13, 36, 26], they also become slower, and require more computation. Such increases in computational demands make it difficult to deploy state-of-the-art CNN models on resource-constrained platforms

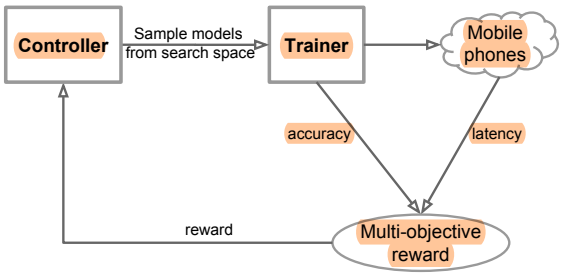


Figure 1: **An Overview of Platform-Aware Neural Architecture Search for Mobile**.

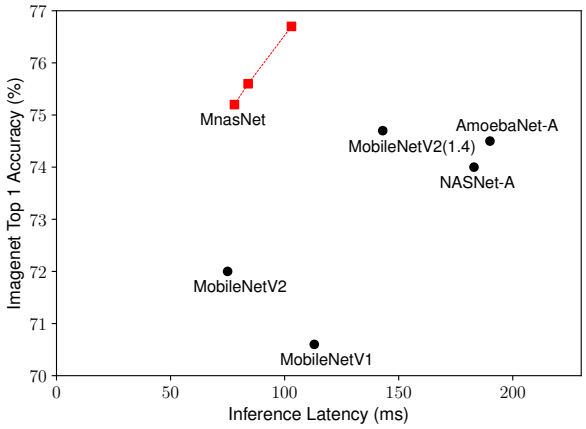


Figure 2: **Accuracy vs. Latency Comparison** – Our MnasNet models significantly outperforms other mobile models [29, 36, 26] on ImageNet. Details can be found in Table 1.

such as mobile or embedded devices.

Given restricted computational resources available on mobile devices, much recent research has focused on designing and improving mobile CNN models by reducing the depth of the network and utilizing less expensive operations, such as depthwise convolution [11] and group convolution [33]. However, designing a resource-constrained mobile model is challenging: one has to carefully balance accuracy and resource-efficiency, resulting in a significantly large design space.

In this paper, we propose an automated neural architecture search approach for designing mobile CNN models. Figure 1 shows an overview of our approach, where the main differences from previous approaches are the latency aware multi-objective reward and the novel search space. Our approach is based on two main ideas. First, we formulate the design problem as a multi-objective optimization problem that considers both accuracy and inference latency of CNN models. Unlike in previous work [36, 26, 21] that use FLOPS to approximate inference latency, we directly measure the real-world latency by executing the model on real mobile devices. Our idea is inspired by the observation that FLOPS is often an inaccurate proxy: for example, MobileNet [11] and NASNet [36] have similar FLOPS (575M vs. 564M), but their latencies are significantly different (113ms vs. 183ms, details in Table 1). Secondly, we observe that previous automated approaches mainly search for a few types of cells and then repeatedly stack the same cells through the network. This simplifies the search process, but also precludes layer diversity that is important for computational efficiency. To address this issue, we propose a novel *factorized hierarchical search space*, which allows layers to be architecturally different yet still strikes the right balance between flexibility and search space size.

We apply our proposed approach to ImageNet classification [28] and COCO object detection [18]. Figure 2 summarizes a comparison between our MnasNet models and other state-of-the-art mobile models. Compared to the MobileNetV2 [29], our model improves the ImageNet accuracy by 3.0% with similar latency on the Google Pixel phone. On the other hand, if we constrain the target accuracy, then our MnasNet models are $\mathbf{1.8}\times$ **faster** than MobileNetV2 and $\mathbf{2.3}\times$ **faster** thans NASNet [36] with better accuracy. Compared to the widely used ResNet-50 [9], our MnasNet model achieves slightly higher (76.7%) accuracy with $\mathbf{4.8}\times$ **fewer** parameters and $\mathbf{10}\times$ **fewer** multiply-add operations. By plugging our model as a feature extractor into the SSD object detection framework, our model improves both the inference latency and the mAP quality on COCO dataset over MobileNetsV1 and MobileNetV2, and achieves comparable mAP quality (23.0 vs 23.2) as SSD300 [22] with $\mathbf{42}\times$ **less** multiply-add operations.

To summarize, our main contributions are as follows:

1. We introduce a *multi-objective* neural architecture search approach that optimizes both accuracy and real-world latency on mobile devices.

2. We propose a novel *factorized hierarchical search space* to enable layer diversity yet still strike the right balance between flexibility and search space size.

3. We demonstrate new state-of-the-art accuracy on both ImageNet classification and COCO object detection under typical mobile latency constraints.

2. Related Work

Improving the resource efficiency of CNN models has been an active research topic during the last several years. Some commonly-used approaches include 1) quantizing the weights and/or activations of a baseline CNN model into lower-bit representations [8, 16], or 2) pruning less important filters according to FLOPs [6, 10], or to platform-aware metrics such as latency introduced in [32]. However, these methods are tied to a baseline model and do not focus on learning novel compositions of CNN operations.

Another common approach is to directly hand-craft more efficient mobile architectures: SqueezeNet [15] reduces the number of parameters and computation by using lower-cost 1x1 convolutions and reducing filter sizes; MobileNet [11] extensively employs depthwise separable convolution to minimize computation density; ShuffleNets [33, 24] utilize low-cost group convolution and channel shuffle; Condensenet [14] learns to connect group convolutions across layers; Recently, MobileNetV2 [29] achieved state-of-the-art results among mobile-size models by using resource-efficient inverted residuals and linear bottlenecks. Unfortunately, given the potentially huge design space, these hand-crafted models usually take significant human efforts.

Recently, there has been growing interest in automating the model design process using neural architecture search. These approaches are mainly based on reinforcement learning [35, 36, 1, 19, 25], evolutionary search [26], differentiable search [21], or other learning algorithms [19, 17, 23]. Although these methods can generate mobile-size models by repeatedly stacking a few searched cells, they do not incorporate mobile platform constraints into the search process or search space. Closely related to our work is MONAS [12], DPP-Net [3], RNAS [34] and Pareto-NASH [4] which attempt to optimize multiple objectives, such as model size and accuracy, while searching for CNNs, but their search process optimizes on small tasks like CIFAR. In contrast, this paper targets real-world mobile latency constraints and focuses on larger tasks like ImageNet classification and COCO object detection.

3. Problem Formulation

We formulate the design problem as a multi-objective search, aiming at finding CNN models with both high-accuracy and low inference latency. Unlike previous architecture search approaches that often optimize for indirect metrics, such as FLOPS, we consider direct *real-world inference latency*, by running CNN models on real mobile devices, and then incorporating the real-world inference latency into our objective. Doing so directly measures what is achievable in practice: our early experiments show it is challenging to approximate real-world latency due to the variety of mobile hardware/software idiosyncrasies.

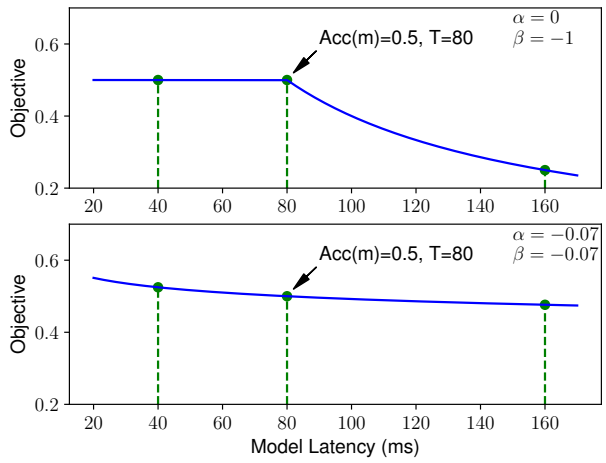


Figure 3: **Objective Function Defined by Equation 2**, assuming accuracy $ACC(m)$=0.5 and target latency T=80ms: (top) show the object values with latency as a hard constraint; (bottom) shows the objective values with latency as a soft constraint.

Given a model m, let $ACC(m)$ denote its accuracy on the target task, $LAT(m)$ denotes the inference latency on the target mobile platform, and T is the target latency. A common method is to treat T as a hard constraint and maximize accuracy under this constraint:

$$\begin{aligned} &\underset{m}{\text{maximize}} \quad ACC(m) \\ &\text{subject to} \quad LAT(m) \leq T \end{aligned} \tag{1}$$

However, this approach only maximizes a single metric and does not provide multiple Pareto optimal solutions. Informally, a model is called Pareto optimal [2] if either it has the highest accuracy without increasing latency or it has the lowest latency without decreasing accuracy. Given the computational cost of performing architecture search, we are more interested in finding multiple Pareto-optimal solutions in a single architecture search.

While there are many methods in the literature [2], we use a customized weighted product method[1] to approximate Pareto optimal solutions, with optimization goal defined as:

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T}\right]^{w} \tag{2}$$

where w is the weight factor defined as:

$$w = \begin{cases} \alpha, & \text{if } LAT(m) \leq T \\ \beta, & \text{otherwise} \end{cases} \tag{3}$$

[1]We pick the weighted product method because it is easy to customize, but we expect methods like weighted sum should be also fine.

where α and β are application-specific constants. An empirical rule for picking α and β is to ensure Pareto-optimal solutions have similar reward under different accuracy-latency trade-offs. For instance, we empirically observed doubling the latency usually brings about 5% relative accuracy gain. Given two models: (1) M1 has latency l and accuracy a; (2) M2 has latency $2l$ and 5% higher accuracy $a \cdot (1 + 5\%)$, they should have similar reward: $Reward(M2) = a \cdot (1 + 5\%) \cdot (2l/T)^{\beta} \approx Reward(M1) = a \cdot (l/T)^{\beta}$. Solving this gives $\beta \approx -0.07$. Therefore, we use $\alpha = \beta = -0.07$ in our experiments unless explicitly stated.

Figure 3 shows the objective function with two typical values of (α, β). In the top figure with $(\alpha = 0, \beta = -1)$, we simply use accuracy as the objective value if measured latency is less than the target latency T; otherwise, we sharply penalize the objective value to discourage models from violating latency constraints. The bottom figure $(\alpha = \beta = -0.07)$ treats the target latency T as a soft constraint, and smoothly adjusts the objective value based on the measured latency.

4. Mobile Neural Architecture Search

In this section, we will first discuss our proposed novel factorized hierarchical search space, and then summarize our reinforcement-learning based search algorithm.

4.1. Factorized Hierarchical Search Space

As shown in recent studies [36, 20], a well-defined search space is extremely important for neural architecture search. However, most previous approaches [35, 19, 26] only search for a few complex cells and then repeatedly stack the same cells. These approaches don't permit layer diversity, which we show is critical for achieving both high accuracy and lower latency.

In contrast to previous approaches, we introduce a novel factorized hierarchical search space that factorizes a CNN model into unique blocks and then searches for the operations and connections per block separately, thus allowing different layer architectures in different blocks. Our intuition is that we need to search for the best operations based on the input and output shapes to obtain better accuracy-latency trade-offs. For example, earlier stages of CNNs usually process larger amounts of data and thus have much higher impact on inference latency than later stages. Formally, consider a widely-used depthwise separable convolution [11] kernel denoted as the four-tuple (K, K, M, N) that transforms an input of size (H, W, M)[2] to an output of size (H, W, N), where (H, W) is the input resolution and M, N are the input/output filter sizes. The total number of multiply-adds can be described as:

[2]We omit batch size dimension for simplicity.

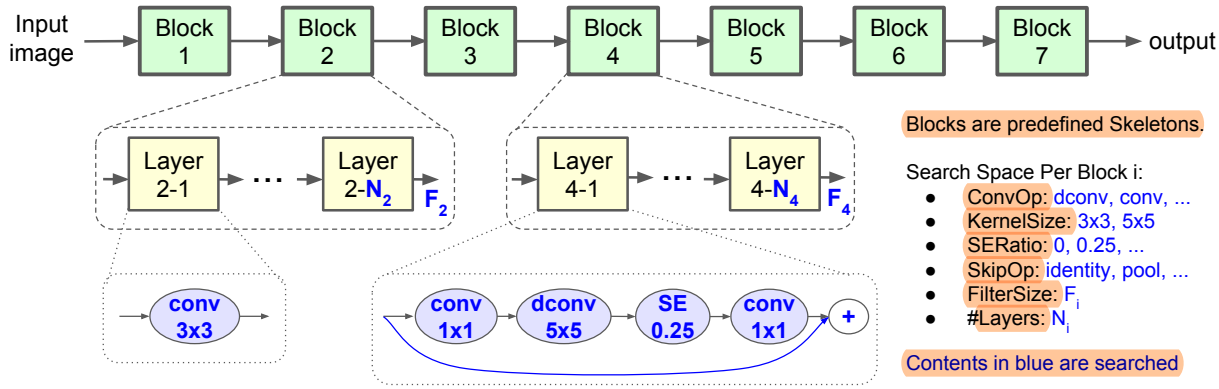


Figure 4: **Factorized Hierarchical Search Space.** Network layers are grouped into a number of predefined skeletons, called blocks, based on their input resolutions and filter sizes. Each block contains a variable number of repeated identical layers where only the first layer has stride 2 if input/output resolutions are different but all other layers have stride 1. For each block, we search for the operations and connections for a single layer and the number of layers N, then the same layer is repeated N times (e.g., Layer 4-1 to 4-N_4 are the same). Layers from different blocks (e.g., Layer 2-1 and 4-1) can be different.

$$H * W * M * (K * K + N) \tag{4}$$

Here we need to carefully balance the kernel size K and filter size N if the total computation is constrained. For instance, increasing the receptive field with larger kernel size K of a layer must be balanced with reducing either the filter size N at the same layer, or compute from other layers.

Figure 4 shows the baseline structure of our search space. We partition a CNN model into a sequence of pre-defined blocks, gradually reducing input resolutions and increasing filter sizes as is common in many CNN models. Each block has a list of identical layers, whose operations and connections are determined by a per-block sub search space. Specifically, a sub search space for a block i consists of the following choices:

- Convolutional ops $ConvOp$: regular conv (conv), depthwise conv (dconv), and mobile inverted bottleneck conv [29].
- Convolutional kernel size $KernelSize$: 3x3, 5x5.
- Squeeze-and-excitation [13] ratio $SERatio$: 0, 0.25.
- Skip ops $SkipOp$: pooling, identity residual, or no skip.
- Output filter size F_i.
- Number of layers per block N_i.

$ConvOp$, $KernelSize$, $SERatio$, $SkipOp$, F_i determines the architecture of a layer, while N_i determines how many times the layer will be repeated for the block. For example, each layer of block 4 in Figure 4 has an inverted bottleneck 5x5 convolution and an identity residual skip path, and the same layer is repeated N_4 times. We discretize all search options using MobileNetV2 as a reference: For #layers in each block, we search for $\{0, +1, -1\}$ based on MobileNetV2; for filter size per layer, we search for its relative size in $\{0.75, 1.0, 1.25\}$ to MobileNetV2 [29].

Our factorized hierarchical search space has a distinct advantage of balancing the diversity of layers and the size of total search space. Suppose we partition the network into B blocks, and each block has a sub search space of size S with average N layers per block, then our total search space size would be S^B, versing the flat per-layer search space with size S^{B*N}. A typical case is $S = 432, B = 5, N = 3$, where our search space size is about 10^{13}, versing the per-layer approach with search space size 10^{39}.

4.2. Search Algorithm

Inspired by recent work [35, 36, 25, 20], we use a reinforcement learning approach to find Pareto optimal solutions for our multi-objective search problem. We choose reinforcement learning because it is convenient and the reward is easy to customize, but we expect other methods like evolution [26] should also work.

Concretely, we follow the same idea as [36] and map each CNN model in the search space to a list of tokens. These tokens are determined by a sequence of actions $a_{1:T}$ from the reinforcement learning agent based on its parameters θ. Our goal is to maximize the expected reward:

$$J = E_{P(a_{1:T};\theta)}[R(m)] \tag{5}$$

where m is a sampled model determined by action $a_{1:T}$, and $R(m)$ is the objective value defined by equation 2.

As shown in Figure 1, the search framework consists of three components: a recurrent neural network (RNN) based controller, a trainer to obtain the model accuracy, and a mobile phone based inference engine for measuring the latency. We follow the well known sample-eval-update loop to train the controller. At each step, the controller first samples a batch of models using its current parameters θ, by

Model	Type	#Params	#Mult-Adds	Top-1 Acc. (%)	Top-5 Acc. (%)	Inference Latency
MobileNetV1 [11]	manual	4.2M	575M	70.6	89.5	113ms
SqueezeNext [5]	manual	3.2M	708M	67.5	88.2	-
ShuffleNet (1.5x) [33]	manual	3.4M	292M	71.5	-	-
ShuffleNet (2x)	manual	5.4M	524M	73.7	-	-
ShuffleNetV2 (1.5x) [24]	manual	-	299M	72.6	-	-
ShuffleNetV2 (2x)	manual	-	597M	75.4	-	-
CondenseNet (G=C=4) [14]	manual	2.9M	274M	71.0	90.0	-
CondenseNet (G=C=8)	manual	4.8M	529M	73.8	91.7	-
MobileNetV2 [29]	manual	3.4M	300M	72.0	91.0	75ms
MobileNetV2 (1.4x)	manual	6.9M	585M	74.7	92.5	143ms
NASNet-A [36]	auto	5.3M	564M	74.0	91.3	183ms
AmoebaNet-A [26]	auto	5.1M	555M	74.5	92.0	190ms
PNASNet [19]	auto	5.1M	588M	74.2	91.9	-
DARTS [21]	auto	4.9M	595M	73.1	91	-
MnasNet-A1	**auto**	**3.9M**	**312M**	**75.2**	**92.5**	**78ms**
MnasNet-A2	**auto**	**4.8M**	**340M**	**75.6**	**92.7**	**84ms**
MnasNet-A3	**auto**	**5.2M**	**403M**	**76.7**	**93.3**	**103ms**

Table 1: **Performance Results on ImageNet Classification** [28]. We compare our MnasNet models with both manually-designed mobile models and other automated approaches – *MnasNet-A1* is our baseline model;*MnasNet-A2* and *MnasNet-A3* are two models (for comparison) with different latency from the same architecture search experiment; *#Params*: number of trainable parameters; *#Mult-Adds*: number of multiply-add operations per image; *Top-1/5 Acc.*: the top-1 or top-5 accuracy on ImageNet validation set; *Inference Latency* is measured on the big CPU core of a Pixel 1 Phone with batch size 1.

predicting a sequence of tokens based on the softmax logits from its RNN. For each sampled model m, we train it on the target task to get its accuracy $ACC(m)$, and run it on real phones to get its inference latency $LAT(m)$. We then calculate the reward value $R(m)$ using equation 2. At the end of each step, the parameters θ of the controller are updated by maximizing the expected reward defined by equation 5 using Proximal Policy Optimization [30]. The sample-eval-update loop is repeated until it reaches the maximum number of steps or the parameters θ converge.

5. Experimental Setup

Directly searching for CNN models on large tasks like ImageNet or COCO is expensive, as each model takes days to converge. While previous approaches mainly perform architecture search on smaller tasks such as CIFAR-10 [36, 26], we find those small proxy tasks don't work when model latency is taken into account, because one typically needs to scale up the model when applying to larger problems. In this paper, we directly perform our architecture search on the ImageNet training set but with fewer training steps (5 epochs). As a common practice, we reserve randomly selected 50K images from the training set as the fixed validation set. To ensure the accuracy improvements are from our search space, we use the same RNN controller as NASNet [36] even though it is not efficient: each architecture search takes 4.5 days on 64 TPUv2 devices. During training, we measure the real-world latency of each sampled model by running it on the single-thread big CPU core of Pixel 1 phones. In total, our controller samples about 8K models during architecture search, but only 15 top-performing models are transferred to the full ImageNet and only 1 model is transferred to COCO.

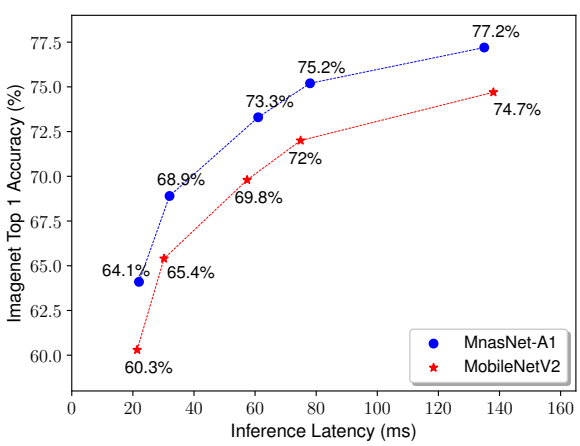
For full ImageNet training, we use RMSProp optimizer with decay 0.9 and momentum 0.9. Batch norm is added after every convolution layer with momentum 0.99, and weight decay is 1e-5. Dropout rate 0.2 is applied to the last layer. Following [7], learning rate is increased from 0 to 0.256 in the first 5 epochs, and then decayed by 0.97 every 2.4 epochs. We use batch size 4K and Inception preprocessing with image size 224×224. For COCO training, we plug our learned model into SSD detector [22] and use the same settings as [29], including input size 320×320.

6. Results

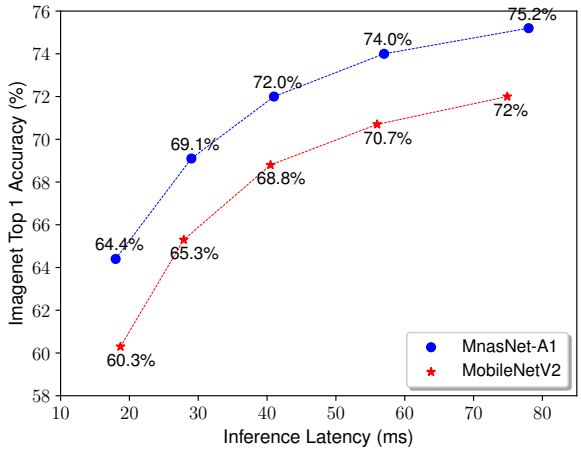
In this section, we study the performance of our models on ImageNet classification and COCO object detection, and compare them with other state-of-the-art mobile models.

6.1. ImageNet Classification Performance

Table 1 shows the performance of our models on ImageNet [28]. We set our target latency as $T = 75ms$, similar



(a) Depth multiplier = 0.35, 0.5, 0.75, 1.0, 1.4, corresponding to points from left to right.



(b) Input size = 96, 128, 160, 192, 224, corresponding to points from left to right.

Figure 5: **Performance Comparison with Different Model Scaling Techniques**. MnasNet is our baseline model shown in Table 1. We scale it with the same depth multipliers and input sizes as MobileNetV2.

		Inference Latency	Top-1 Acc.
w/o SE	MobileNetV2	75ms	72.0%
	NASNet	183ms	74.0%
	MnasNet-B1	77ms	74.5%
w/ SE	MnasNet-A1	78ms	75.2%
	MnasNet-A2	84ms	75.6%

Table 2: **Performance Study for Squeeze-and-Excitation SE [13]** – *MnasNet-A* denote the default MnasNet with SE in search space; *MnasNet-B* denote MnasNet with no SE in search space.

to MobileNetV2 [29], and use Equation 2 with α=β=-0.07 as our reward function during architecture search. Afterwards, we pick three top-performing MnasNet models, with different latency-accuracy trade-offs from the same search experiment and compare them with existing mobile models.

As shown in the table, our MnasNet A1 model achieves 75.2% top-1 / 92.5% top-5 accuracy with 78ms latency and 3.9M parameters / 312M multiply-adds, achieving a new state-of-the-art accuracy for this typical mobile latency constraint. In particular, MnasNet runs $\mathbf{1.8\times}$ **faster** than MobileNetV2 (1.4) [29] on the same Pixel phone with 0.5% higher accuracy. Compared with automatically searched CNN models, our MnasNet runs $\mathbf{2.3\times}$ **faster** than the mobile-size NASNet-A [36] with 1.2% higher top-1 accuracy. Notably, our slightly larger MnasNet-A3 model achieves better accuracy than ResNet-50 [9], but with $\mathbf{4.8\times}$ **fewer** parameters and $\mathbf{10\times}$ **fewer** multiply-add cost.

Given that squeeze-and-excitation (SE [13]) is relatively new and many existing mobile models don't have this extra optimization, we also show the search results without SE in the search space in Table 2; our automated approach still significantly outperforms both MobileNetV2 and NASNet.

6.2. Model Scaling Performance

Given the myriad application requirements and device heterogeneity present in the real world, developers often scale a model up or down to trade accuracy for latency or model size. One common scaling technique is to modify the filter size using a depth multiplier [11]. For example, a depth multiplier of 0.5 halves the number of channels in each layer, thus reducing the latency and model size. Another common scaling technique is to reduce the input image size without changing the network.

Figure 5 compares the model scaling performance of MnasNet and MobileNetV2 by varying the depth multipliers and input image sizes. As we change the depth multiplier from 0.35 to 1.4, the inference latency also varies from 20ms to 160ms. As shown in Figure 5a, our MnasNet model consistently achieves better accuracy than MobileNetV2 for each depth multiplier. Similarly, our model is also robust to input size changes and consistently outperforms MobileNetV2 (increaseing accuracy by up to **4.1%**) across all input image sizes from 96 to 224, as shown in Figure 5b.

In addition to model scaling, our approach also allows searching for a new architecture for any latency target. For example, some video applications may require latency as low as 25ms. We can either scale down a baseline model, or search for new models specifically targeted to this latency constraint. Table 4 compares these two approaches. For fair comparison, we use the same 224x224 image sizes for all

Network	#Params	#Mult-Adds	mAP	mAP_S	mAP_M	mAP_L	Inference Latency
YOLOv2 [27]	50.7M	17.5B	21.6	5.0	22.4	35.5	-
SSD300 [22]	36.1M	35.2B	23.2	5.3	23.2	39.6	-
SSD512 [22]	36.1M	99.5B	26.8	9.0	28.9	41.9	-
MobileNetV1 + SSDLite [11]	5.1M	1.3B	22.2	-	-	-	270ms
MobileNetV2 + SSDLite [29]	4.3M	0.8B	22.1	-	-	-	200ms
MnasNet-A1 + SSDLite	**4.9M**	**0.8B**	**23.0**	**3.8**	**21.7**	**42.0**	**203ms**

Table 3: **Performance Results on COCO Object Detection** – *#Params*: number of trainable parameters; *#Mult-Adds*: number of multiply-additions per image; mAP: standard mean average precision on test-dev2017; mAP_S, mAP_M, mAP_L: mean average precision on small, medium, large objects; *Inference Latency*: the inference latency on Pixel 1 Phone.

	Params	MAdds	Latency	Top1 Acc.
MobileNetV2 (0.35x)	1.66M	59M	21.4ms	60.3%
MnasNet-A1 (0.35x)	1.7M	63M	22.8ms	64.1%
MnasNet-search1	1.9M	65M	22.0ms	64.9%
MnasNet-search2	2.0M	68M	23.2ms	66.0%

Table 4: **Model Scaling vs. Model Search** – *MobileNetV2 (0.35x) and MnasNet-A1 (0.35x)* denote scaling the baseline models with depth multiplier 0.35; *MnasNet-search1/2* denotes models from a new architecture search that targets 22ms latency constraint.

models. Although our MnasNet already outperforms MobileNetV2 with the same scaling parameters, we can further improve the accuracy with a new architecture search targeting a 22ms latency constraint.

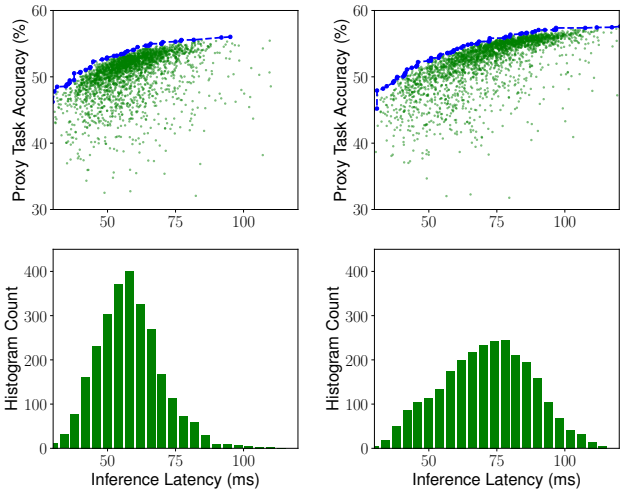
6.3. COCO Object Detection Performance

For COCO object detection [18], we pick the MnasNet models in Table 2 and use them as the feature extractor for SSDLite, a modified resource-efficient version of SSD [29]. Similar to [29], we compare our models with other mobile-size SSD or YOLO models.

Table 3 shows the performance of our MnasNet models on COCO. Results for YOLO and SSD are from [27], while results for MobileNets are from [29]. We train our models on COCO trainval35k and evaluate them on test-dev2017 by submitting the results to COCO server. As shown in the table, our approach significantly improve the accuracy over MobileNet V1 and V2. Compare to the standard SSD300 detector [22], our MnasNet model achieves comparable mAP quality (23.0 vs 23.2) as SSD300 with $\mathbf{7.4\times}$ **fewer** parameters and $\mathbf{42\times}$ **fewer** multiply-adds.

7. Ablation Study and Discussion

In this section, we study the impact of latency constraint and search space, and discuss MnasNet architecture details and the importance of layer diversity.



(a) $\alpha = 0, \beta = -1$ (b) $\alpha = \beta = -0.07$

Figure 6: **Multi-Objective Search Results** based on equation 2 with (a) α=0, β=-1; and (b) α=β=-0.07. Target latency is T=$75ms$. Top figure shows the Pareto curve (blue line) for the 3000 sampled models (green dots); bottom figure shows the histogram of model latency.

7.1. Soft vs. Hard Latency Constraint

Our multi-objective search method allows us to deal with both hard and soft latency constraints by setting α and β to different values in the reward equation 2. Figure 6 shows the multi-objective search results for typical α and β. When $\alpha = 0, \beta = -1$, the latency is treated as a hard constraint, so the controller tends to focus more on faster models to avoid the latency penalty. On the other hand, by setting $\alpha = \beta = -0.07$, the controller treats the target latency as a soft constraint and tries to search for models across a wider latency range. It samples more models around the target latency value at 75ms, but also explores models with latency smaller than 40ms or greater than 110ms. This allows us to pick multiple models from the Pareto curve in a single architecture search as shown in Table 1.

7.2. Disentangling Search Space and Reward

To disentangle the impact of our two key contributions: multi-objective reward and new search space, Figure 5 compares their performance. Starting from NASNet [36], we first employ the same cell-base search space [36] and simply add the latency constraint using our proposed multiple-object reward. Results show it generates a much faster model by trading the accuracy to latency. Then, we apply both our multi-objective reward and our new factorized search space, and achieve both higher accuracy and lower latency, suggesting the effectiveness of our search space.

Reward	Search Space	Latency	Top-1 Acc.
Single-obj [36]	Cell-based [36]	183ms	74.0%
Multi-obj	Cell-based [36]	100ms	72.0%
Multi-obj	**MnasNet**	**78ms**	**75.2%**

Table 5: **Comparison of Decoupled Search Space and Reward Design** – **Multi-obj** denotes our multi-objective reward; **Single-obj** denotes only optimizing accuracy.

7.3. MnasNet Architecture and Layer Diversity

Figure 7(a) illustrates our MnasNet-A1 model found by our automated approach. As expected, it consists of a variety of layer architectures throughout the network. One interesting observation is that our MnasNet uses both 3x3 and 5x5 convolutions, which is different from previous mobile models that all only use 3x3 convolutions.

In order to study the impact of layer diversity, Table 6 compares MnasNet with its variants that only repeat a single type of layer (fixed kernel size and expansion ratio). Our MnasNet model has much better accuracy-latency trade-offs than those variants, highlighting the importance of layer diversity in resource-constrained CNN models.

8. Conclusion

This paper presents an automated neural architecture search approach for designing resource-efficient mobile CNN models using reinforcement learning. Our main ideas are incorporating platform-aware *real-world latency information* into the search process and utilizing a novel *factorized hierarchical search space* to search for mobile models with the best trade-offs between accuracy and latency. We demonstrate that our approach can automatically find significantly better mobile models than existing approaches, and achieve new state-of-the-art results on both ImageNet classification and COCO object detection under typical mobile inference latency constraints. The resulting MnasNet architecture also provides interesting findings on the importance of layer diversity, which will guide us in designing and improving future mobile CNN models.

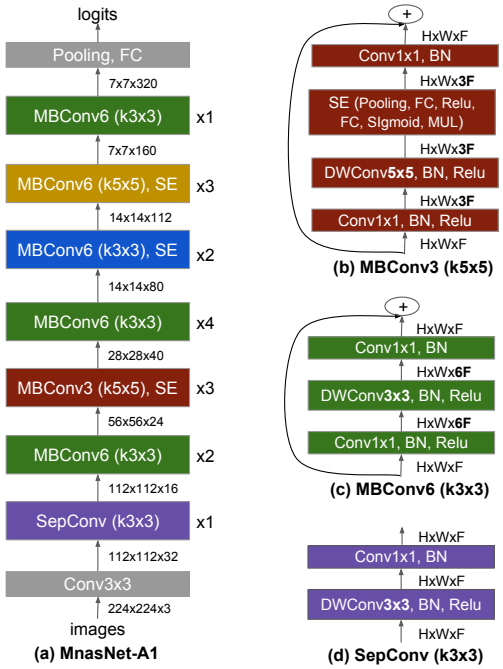


Figure 7: **MnasNet-A1 Architecture** – (a) is a representative model selected from Table 1; (b) - (d) are a few corresponding layer structures. *MBConv* denotes mobile inverted bottleneck conv, *DWConv* denotes depthwise conv, k3x3/k5x5 denotes kernel size, *BN* is batch norm, HxWxF denotes tensor shape (height, width, depth), and $\times 1/2/3/4$ denotes the number of repeated layers within the block.

	Top-1 Acc.	Inference Latency
MnasNet-A1	**75.2%**	**78ms**
MBConv3 (k3x3) only	71.8%	63ms
MBConv3 (k5x5) only	72.5%	79ms
MBConv6 (k3x3) only	74.9%	116ms
MBConv6 (k5x5) only	75.6%	146ms

Table 6: **Performance Comparison of MnasNet and Its Variants** – *MnasNet-A1* denotes the model shown in Figure 7(a); others are variants that repeat a single type of layer throughout the network. All models have the same number of layers and same filter size at each layer.

9. Acknowledgments

We thank Barret Zoph, Dmitry Kalenichenko, Guiheng Zhou, Hongkun Yu, Jeff Dean, Megan Kacholia, Menglong Zhu, Nan Zhang, Shane Almeida, Sheng Li, Vishy Tirumalashetty, Wen Wang, Xiaoqiang Zheng, and the larger device automation platform team, TensorFlow Lite, and Google Brain team.

References

[1] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *ICLR*, 2017.

[2] K. Deb. Multi-objective optimization. *Search methodologies*, pages 403–449, 2014.

[3] 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. *ECCV*, 2018.

[4] T. Elsken, J. H. Metzen, and F. Hutter. Multi-objective architecture search for cnns. *arXiv preprint arXiv:1804.09081*, 2018.

[5] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, and K. Keutzer. Squeezenext: Hardware-aware neural network design. *ECV Workshop at CVPR*, 2018.

[6] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. *CVPR*, 2018.

[7] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[8] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016.

[10] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. *ECCV*, 2018.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[12] C.-H. Hsu, S.-H. Chang, D.-C. Juan, J.-Y. Pan, Y.-T. Chen, W. Wei, and S.-C. Chang. MONAS: Multi-objective neural architecture search using reinforcement learning. *arXiv preprint arXiv:1806.10332*, 2018.

[13] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *CVPR*, 2018.

[14] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. *CVPR*, 2018.

[15] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[16] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CVPR*, 2018.

[17] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. Xing. Neural architecture search with bayesian optimisation and optimal transport. *NeurIPS*, 2018.

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. *ECCV*, 2014.

[19] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. *ECCV*, 2018.

[20] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *ICLR*, 2018.

[21] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. *ICLR*, 2019.

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. *ECCV*, 2016.

[23] R. Luo, F. Tian, T. Qin, and T.-Y. Liu. Neural architecture optimization. *NeurIPS*, 2018.

[24] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *ECCV*, 2018.

[25] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2018.

[26] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019.

[27] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. *CVPR*, 2017.

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, 2018.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI*, 4:12, 2017.

[32] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *ECCV*, 2018.

[33] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CVPR*, 2018.

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

[35] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *ICLR*, 2017.

[36] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *CVPR*, 2018.