MnasNet: Platform-Aware Neural Architecture Search for Mobile

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

arXiv:1807.11626v1 [cs.CV] 31 Jul 2018

Abstract

Designing convolutional neural networks (CNN) models for mobile devices is challenging because mobile models need to be small and fast, yet still accurate. Although significant effort has been dedicated to design and improve mobile models on all three dimensions, it is challenging to manually balance these trade-offs when there are so many architectural possibilities to consider. In this paper, we propose an automated neural architecture search approach for designing resource-constrained mobile CNN models. We propose to explicitly incorporate latency information into the main objective so that the search can identify a model that achieves a good trade-off between accuracy and latency. Unlike in previous work, where mobile latency is considered via another, often inaccurate proxy (e.g., FLOPS), in our experiments, we directly measure real-world inference latency by executing the model on a particular platform, e.g., Pixel phones. To further strike the right balance between flexibility and search space size, we propose a novel factorized hierarchical search space that permits 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 model achieves 74.0% top-1 accuracy with 76ms latency on a Pixel phone, which is $1.5\times$ faster than MobileNetV2 (Sandler et al. 2018) and $2.4\times$ faster than NASNet (Zoph et al. 2018) with the same top-1 accuracy. On the COCO object detection task, our model family achieves both higher mAP quality and lower latency than MobileNets.

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 (Szegedy et al. 2017; Hu, Shen, and Sun 2018; Zoph et al. 2018; Real et al. 2018), 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 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 (Howard et al. 2017) and group convolution (Zhang et al. 2018). 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. Further complicating matters is that each type of mobile devices has its own software and hardware idiosyncrasies and may require different architectures for the best accuracy-efficiency trade-offs.

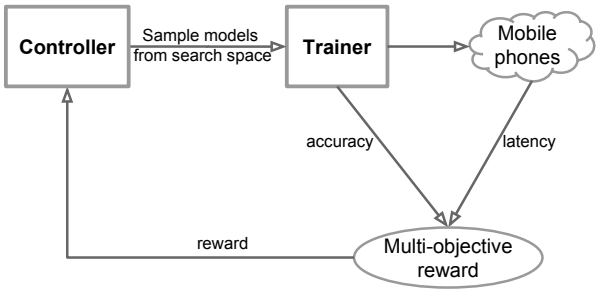


Figure 1: **An Overview of Platform-Aware Neural Architecture Search for Mobile**.

In this paper, we propose an automated neural architecture search approach for designing mobile CNN models. Figure 1 shows an overall view of our approach, where the key differences from previous approaches are the latency aware multi-objective reward and the novel search space. Our approach is inspired by 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. We then use architecture search with reinforcement learning to find the model that achieves the best trade-off between accuracy and latency. Secondly, we observe that previous automated approaches mainly search for a few types of cells and then repeatedly stack the same cells through the CNN network. Those searched models do not take into account that operations like convolution greatly differ in latency based on the concrete shapes they operate on: for instance, two 3x3 convolutions with the same number of theoretical FLOPS but different shapes may not have the same runtime latency. Based on this observation, we propose a *factorized hierarchical search space* composed of a sequence of factorized

Preprint. Work in progress.

blocks, each block containing a list of layers defined by a hierarchical sub search space with different convolution operations and connections. We show that different operations should be used at different depths of an architecture, and searching among this large space of options can effectively be done using architecture search methods that use measured inference latency as part of the reward signal.

We apply our proposed approach to ImageNet classification (Russakovsky et al. 2015) and COCO object detection (Lin et al. 2014). Experimental results show that the best model found by our method significantly outperforms state-of-the-art mobile models. Compared to the recent MobileNetV2 (Sandler et al. 2018), our model improves the ImageNet top-1 accuracy by 2% with the same latency on Pixel phone. On the other hand, if we constrain the target top-1 accuracy, then our method can find another model that is $\mathbf{1.5\times}$ **faster** than MobileNetV2 and $\mathbf{2.4\times}$ **faster** than NASNet (Zoph et al. 2018) with the same accuracy. With the additional squeeze-and-excitation optimization (Hu, Shen, and Sun 2018), our approach achieves ResNet-50 (He et al. 2016) level top-1 accuracy at 76.13%, with $\mathbf{19\times}$ **fewer** parameters and $\mathbf{10\times}$ **fewer** multiply-add operations. We show our models also generalize well with different model scaling techniques (e.g., varying input image sizes), consistently improving ImageNet top-1 accuracy by about 2% over MobileNetV2. 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 MobileNetV1 and MobileNetV2, and achieves comparable mAP quality (22.9 vs 23.2) as SSD300 (Liu et al. 2016) with $\mathbf{35\times}$ **less** computational cost.

To summarize, our main contributions are as follows:

1. We introduce a multi-objective neural architecture search approach based on reinforcement learning, which is capable of finding high accuracy CNN models with low real-world inference latency.
2. We propose a novel factorized hierarchical search space to maximize the on-device resource efficiency of mobile models, by striking the right balance between flexibility and search space size.
3. We show significant and consistent improvements over state-of-the-art mobile CNN models on both ImageNet classification and COCO object detection.

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 (Han, Mao, and Dally 2015; Jacob et al. 2018), or 2) pruning less important filters (Gordon et al. 2018; Yang et al. 2018) during or after training, in order to reduce its computational cost. 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 operations and neural architectures: SqueezeNet (Iandola et al. 2016) reduces the number of parameters and computation by pervasively using lower-cost 1x1 convolutions and reducing filter sizes; MobileNet (Howard et al. 2017) extensively employs depthwise separable convolution to minimize computation density; ShuffleNet (Zhang et al. 2018) utilizes low-cost pointwise group convolution and channel shuffle; Condensenet (Huang et al. 2018) learns to connect group convolutions across layers; Recently, MobileNetV2 (Sandler et al. 2018) 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 quite significant human efforts and are still suboptimal.

Recently, there has been growing interest in automating the neural architecture design process, especially for CNN models. NASNet (Zoph and Le 2017; Zoph et al. 2018) and MetaQNN (Baker et al. 2017) started the wave of automated neural architecture search using reinforcement learning. Consequently, neural architecture search has been further developed, with progressive search methods (Liu et al. 2018a), parameter sharing (Pham et al. 2018), hierarchical search spaces (Liu et al. 2018b), network transfer (Cai et al. 2018), evolutionary search (Real et al. 2018), or differentiable search algorithms (Liu, Simonyan, and Yang 2018). Although these methods can generate mobile-size models by repeatedly stacking a searched cell, they do not incorporate mobile platform constraints into the search process or search space. Recently, MONAS (Hsu et al. 2018), PPP-Net (Dong et al. 2018), RNAS (Zhou et al. 2018) and Pareto-NASH (Elsken, Metzen, and Hutter 2018) attempt to optimize multiple objectives, such as model size and accuracy while searching for CNNs, but they are limited to small tasks like CIFAR-10. In contrast, this paper targets real-world mobile latency constraints and focuses on larger tasks like ImageNet classification and COCO object detection.

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 work which optimizes for indirect metrics such as FLOPS or number of parameters, 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 on proxy inference metrics, including single-core Desktop CPU latency and simulated cost models, show it is challenging to approximate real-world latency due to the variety of mobile hardware/software configurations.

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:

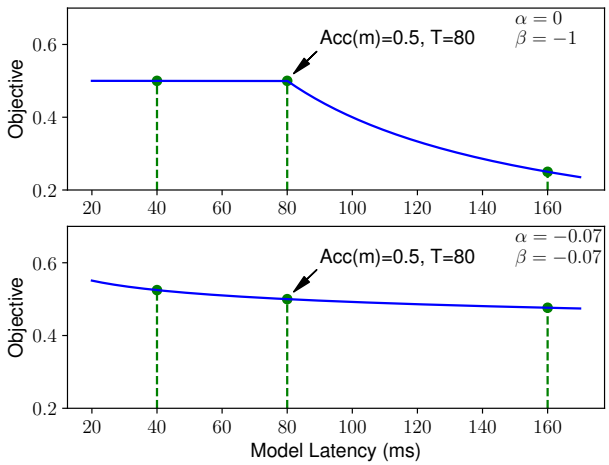


Figure 2: **Objective Function Defined by Equation 2**, assuming accuracy $ACC(m)$=0.5 and target latency T=80ms: (top) shows the objective values with α=0, β=-1, corresponding to the hard latency constraint; (bottom) shows the objective values with α=β=-0.07, corresponding to a soft latency constraint.

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

However, this approach only maximizes a single metric and does not provide multiple Pareto optimal solutions. Informally, a model is called Pareto optimal (Deb 2014) 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 (Deb 2014), we use a customized weighted product method[1] to approximate Pareto optimal solutions, by setting the optimization goal 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:

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

where α and β are application-specific constants. An empirical rule for picking α and β is to check how much accuracy gain or loss is expected if we double or halve the latency. For example, doubling or halving the latency of MobileNetV2 (Sandler et al. 2018) brings about 5% accuracy gain or loss, so we can empirically set $\alpha = \beta = -0.07$, since $2^{-0.07} - 1 \approx 1 - 0.5^{-0.07} \approx 5\%$. By setting (α, β) in this way, equation 2 can effectively approximate Pareto solutions nearby the target latency T.

Figure 2 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. In this paper, we set $\alpha = \beta = -0.07$ in order to obtain multiple Pareto optimal models in a single search experiment. It will be an interesting future direction to explore reward functions that dynamically adapt to the Pareto curve.

Mobile Neural Architecture Search

Search Algorithm

Inspired by recent work (Zoph and Le 2017; Pham et al. 2018; Liu et al. 2018b), we employ a gradient-based 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 search algorithms like evolution (Real et al. 2018) should also work.

Concretely, we follow the same idea as (Zoph et al. 2018) and map each CNN model in the search space to a list of tokens. These tokens are determined by a sequence of actions $a_{i:T}$ from the reinforcement learning agent based on its parameters θ. Our goal is to maximize the expected reward:

$$J = E_{P(a_{1:T};\theta)}[R(m)] \tag{4}$$

where m is a sampled model uniquely 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 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 4 using Proximal Policy Optimization (Schulman et al. 2017). The sample-eval-update loop is repeated until it reaches the maximum number of steps or the parameters θ converge.

Factorized Hierarchical Search Space

As shown in recent studies (Zoph et al. 2018; Liu et al. 2018b), a well-defined search space is extremely important for neural architecture search. In this section, we introduce a novel factorized hierarchical search space that partitions CNN layers into groups and searches for the operations and connections per group. In contrast to previous architecture

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

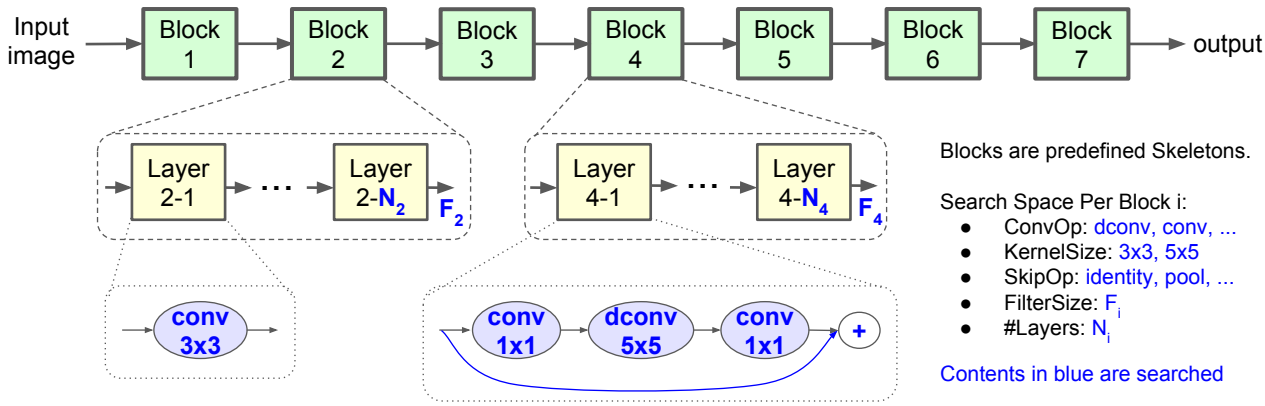


Figure 3: **Factorized Hierarchical Search Space.** Network layers are grouped into a number of predefined skeletons, called blocks, based on their input resolutions and filter sizes. Each block contains a variable number of repeated identical layers where only the first layer has stride 2 if input/output 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.

search approaches (Zoph and Le 2017; Liu et al. 2018a; Real et al. 2018), which only search for a few complex cells and then repeatedly stack the same cells, we simplify the per-cell search space but allow cells to be different.

Our intuition is that we need to search for the best operations based on the input and output shapes to obtain the best accurate-latency trade-offs. For example, earlier stages of CNN models 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 (Howard et al. 2017) 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 computation can be described as:

$$H * W * M * (K * K + N) \tag{5}$$

where the first part, $H * W * M * K * K$, is for the depthwise convolution and the second part, $H * W * M * N$, is for the following 1x1 convolution. Here we need to carefully balance the kernel size K and filter size N if the total computation resources are limited. For instance, increasing the effective 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 3 shows the baseline structure of our search space. We partition a CNN model into a sequence of pre-defined blocks, gradually reducing the input resolution and increasing the filter size 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 with various expansion ratios (Sandler et al. 2018).
- Convolutional kernel size $KernelSize$: 3x3, 5x5.
- Skip operations $SkipOp$: max or average pooling, identity residual skip, or no skip path.
- Output filter size F_i.
- Number of layers per block N_i.

$ConvOp$, $KernelSize$, $SkipOp$, F_i uniquely determines the architecture of a layer, while N_i determines how many times the layer would be repeated for the block. For example, each layer of block 4 in Figure 3 has an inverted bottleneck 5x5 convolution and an identity residual skip path, and the same layer is repeated N_4 times. The final search space is a concatenation of all sub search spaces for each block.

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}. With typical $N = 3$, our search space is orders of magnitude smaller than the flat per-layer search space.

Experimental Setup

Directly searching for CNN models on large tasks like ImageNet or COCO is prohibitively expensive, as each model takes days to converge. Following common practice in previous work (Zoph et al. 2018; Real et al. 2018), we conduct our architecture search experiments on a smaller proxy task, and then transfer the top-performing models discovered during architecture search to the target full tasks. However, finding a good proxy task for both accuracy and latency is non-trivial: one has to consider task type, dataset type, input image size and type. Our initial experiments on CIFAR-10 and the Stanford Dogs Dataset (Khosla et al. 2011) showed

[2]We omit batch size dimension for simplicity.

Model	Type	#Parameters	#Mult-Adds	Top-1 Acc. (%)	Top-5 Acc. (%)	CPU Latency
MobileNetV1 (Howard et al. 2017)	manual	4.2M	575M	70.6	89.5	113ms
SqueezeNext (Gholami et al. 2018)	manual	3.2M	708M	67.5	88.2	-
ShuffleNet (1.5) (Zhang et al. 2018)	manual	3.4M	292M	71.5	-	-
ShuffleNet (x2)	manual	5.4M	524M	73.7	-	-
CondenseNet (G=C=4) (Huang et al. 2018)	manual	2.9M	274M	71.0	90.0	-
CondenseNet (G=C=8)	manual	4.8M	529M	73.8	91.7	-
MobileNetV2 (Sandler et al. 2018)	manual	3.4M	300M	72.0	91.0	75ms
MobileNetV2 (1.4)	manual	6.9M	585M	74.7	92.5	143ms
NASNet-A (Zoph et al. 2018)	auto	5.3M	564M	74.0	91.3	183ms
AmoebaNet-A (Real et al. 2018)	auto	5.1M	555M	74.5	92.0	190ms
PNASNet (Liu et al. 2018a)	auto	5.1M	588M	74.2	91.9	-
DARTS (Liu, Simonyan, and Yang 2018)	auto	4.9M	595M	73.1	91	-
MnasNet	auto	4.2M	317M	74.0	91.78	76ms
MnasNet-65	auto	3.6M	270M	73.02	91.14	65ms
MnasNet-92	auto	4.4M	388M	74.79	92.05	92ms
MnasNet (+SE)	auto	4.7M	319M	75.42	92.51	90ms
MnasNet-65 (+SE)	auto	4.1M	272M	74.62	91.93	75ms
MnasNet-92 (+SE)	auto	5.1M	391M	76.13	92.85	107ms

Table 1: **Performance Results on ImageNet Classification** (Russakovsky et al. 2015). We compare our MnasNet models with both manually-designed mobile models and other automated approaches – *MnasNet* is our baseline model; *MnasNet-65* and *MnasNet-92* are two models (for comparison) with different latency from the same architecture search experiment; *+SE* denotes with additional squeeze-and-excitation optimization (Hu, Shen, and Sun 2018); *#Parameters*: 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; *CPU Latency*: the inference latency with batch size 1 on Pixel 1 Phone.

that these datasets are not good proxy tasks for ImageNet when model latency is taken into account. In this paper, we directly perform our architecture search on the ImageNet training set but with fewer training steps. As it is common in the architecture search literature to have a separate validation set to measure accuracy, we also reserve a randomly selected 50K images from the training set as the fixed validation set. During architecture search, we train each sampled model on 5 epochs of the proxy training set using an aggressive learning schedule, and evaluate the model on the 50K validation set. Meanwhile, we measure the real-world latency of each sampled model by converting the model into TFLite format and run 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 a few top-performing models (< 15) are transferred to the full ImageNet or COCO. Note that we never evaluate on the original ImageNet validation dataset during architecture search.

For full ImageNet training, we use the RMSProp optimizer with decay 0.9 and momentum 0.9. Batch norm is added after every convolution layer with momentum 0.9997, and weight decay is set to 0.00001. Following (Goyal et al. 2017), we linearly increase the learning rate from 0 to 0.256 in the first 5-epoch warmup training stage, and then decay the learning rate by 0.97 every 2.4 epochs. These hyperparameters are determined with a small grid search of 8 combinations of weight decay $\{0.00001, 0.00002\}$, learning rate $\{0.256, 0.128\}$, and batchnorm momentum $\{0.9997, 0.999\}$. We use standard Inception preprocessing and resize input images to 224×224 unless explicitly specified in this paper.

For full COCO training, we plug our learned model architecture into the open-source TensorFlow Object Detection framework, as a new feature extractor. Object detection training settings are set to be the same as (Sandler et al. 2018), including the input size 320×320.

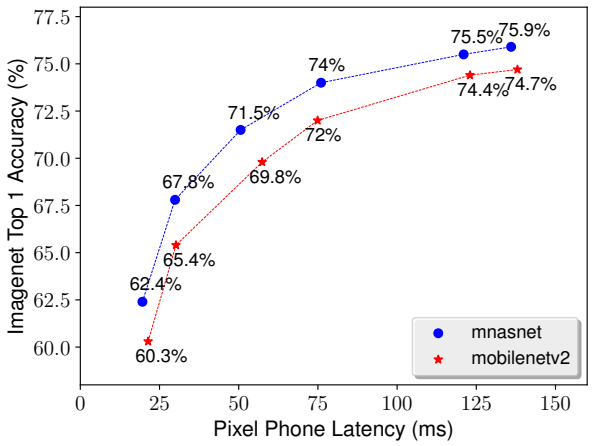
Results

ImageNet Classification Performance

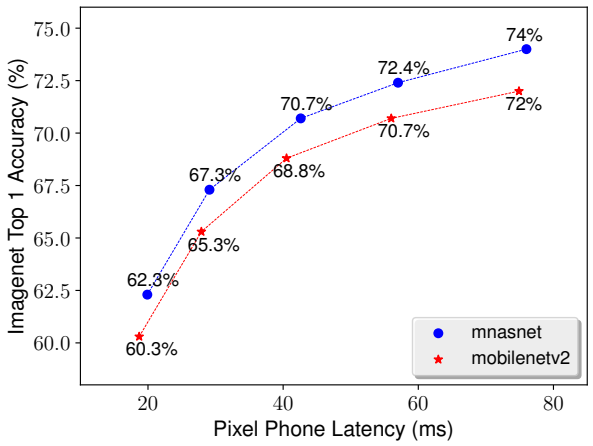
Table 1 shows the performance of our models on ImageNet (Russakovsky et al. 2015). We set our target latency as $T = 80ms$, similar to MobileNetV2 (Sandler et al. 2018), and use Equation 2 with α=β=-0.07 as our reward function during architecture search. Afterwards, we pick three top-performing MnasNet models, with different latency-accuracy trade-offs from the same search experiment and compare the results with existing mobile CNN models.

As shown in the table, our MnasNet model achieves 74% top-1 accuracy with 317 million multiply-adds and 76ms latency on a Pixel phone, achieving a new state-of-the-art accuracy for this typical mobile latency constraint. Compared with the recent MobileNetV2 (Sandler et al. 2018), MnasNet improves the top-1 accuracy by 2% while maintaining the same latency; on the more accurate end, MnasNet-92 achieves a top-1 accuracy of 74.79% and runs $\mathbf{1.55\times}$ **faster** than MobileNetV2 on the same Pixel phone. Compared with recent automatically searched CNN models, our MnasNet runs $\mathbf{2.4\times}$ **faster** than the mobile-size NASNet-A (Zoph et al. 2018) with the same top-1 accuracy.

For a fair comparison, the recent squeeze-and-excitation optimization (Hu, Shen, and Sun 2018) is not included in our baseline MnasNet models since all other models in Table 1 do not have this optimization. However, our approach can

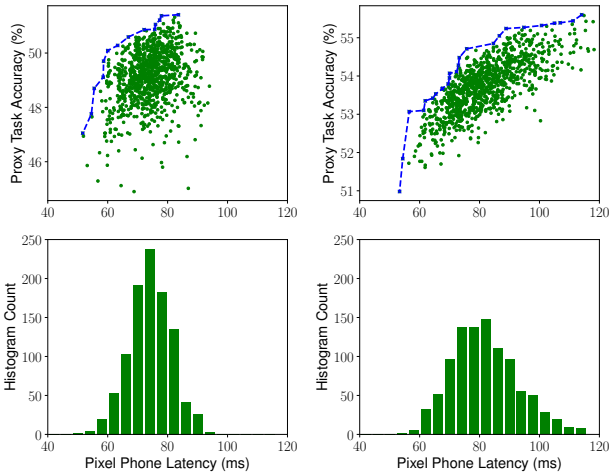


(a) Depth multiplier = 0.35, 0.5, 0.75, 1.0, 1.3, 1.4, corresponding to points from left to right.



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

Figure 4: **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.



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

Figure 5: **Multi-Objective Search Results** based on equation 2 with (a) α=0, β=-1; and (b) α=β=-0.07. Target latency is T=$80ms$. Top figure shows the Pareto curve (blue line) for the 1000 sampled models (green dots); bottom figure shows the histogram of model latency.

take advantage of these recently introduced operations and optimizations. For instance, by incorporating the squeeze-and-excitation denoted as (+SE) in Table 1, our MnasNet-92(+SE) model achieves ResNet-50 (He et al. 2016) level top-1 accuracy at 76.13%, with **19× fewer** parameters and **10× fewer** multiply-add operations.

Notably, we only tune the hyperparameters for MnasNet on 8 combinations of learning rate, weight decay, batch norm momentum, and then simply use the same training settings for MnasNet-65 and MnasNet-92. Therefore, we confirm that the performance gains are from our novel search space and search method, rather than the training settings.

Architecture Search Method

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 reward equation 2.

Figure 5 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 search for models within a very small latency range around the target latency value. 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 80ms, but also explores models with latency smaller than 60ms 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.

Sensitivity to Model Scaling

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 of the network using a depth multiplier (Howard et al. 2017), which modifies the number of filters in each layer with the given ratio. For example, a depth multiplier of 0.5 halves the number of channels in each layer compared to the default, thus significantly reducing the computational resources, latency, and model size. Another common model scaling technique is to reduce the input image size without changing the number of parameters of the network.

Figure 4 compares the performance of MnasNet and MobileNetV2 with different depth multipliers and input image sizes. As we change the depth multiplier from 0.35 to 1.4,

Network	#Parameters	#Mult-Adds	mAP	mAP_S	mAP_M	mAP_L	CPU Latency
YOLOv2 (Redmon and Farhadi 2017)	50.7M	17.5B	21.6	5.0	22.4	35.5	-
SSD300 (Liu et al. 2016)	36.1M	35.2B	23.2	5.3	23.2	39.6	-
SSD512 (Liu et al. 2016)	36.1M	99.5B	26.8	9.0	28.9	41.9	-
MobileNetV1 + SSDLite (Howard et al. 2017)	5.1M	1.3B	22.2	-	-	-	270ms
MobileNetV2 + SSDLite (Sandler et al. 2018)	4.3M	0.8B	22.1	-	-	-	200ms
MnasNet + SSDLite	4.3M	0.7B	22.3	3.1	19.5	42.9	190ms
MnasNet-92 + SSDLite	5.3M	1.0B	22.9	3.6	20.5	43.2	227ms

Table 2: **Performance Results on COCO Object Detection** – *#Parameters*: 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; *CPU Latency*: the inference latency on Pixel 1 Phone.

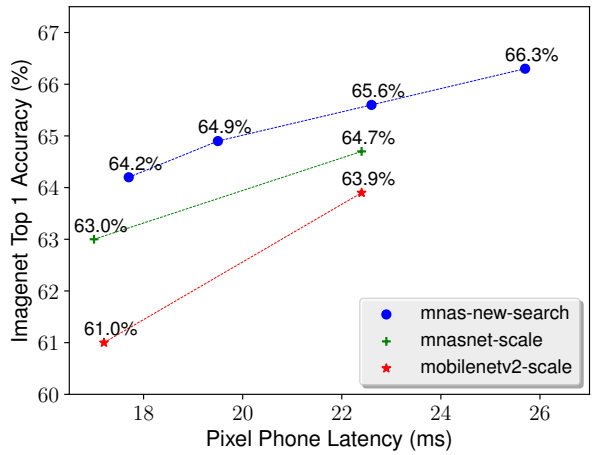


Figure 6: **Model Scaling vs. Model Search** – *mobilenetv2-scale*: scaling MobileNetV2 with (depth multiplier, input size) = (0.5, 160) and (0.5, 192), corresponding to points from left to right; *mnasnet-scale*: scaling the baseline MnasNet with the same depth multipliers and input sizes; *mnas-new-search*: models from a new architecture search with target latency at 23ms.

the inference latency also varies from 20ms to 130ms, but as shown in Figure 4a, our MnasNet model consistently achieves better top-1 accuracy than MobileNetV2 for each depth multiplier. Similarly, our model is also robust to input size changes and consistently outperforms MobileNetV2 across all input image sizes from 96 to 224, as shown in Figure 4b.

In addition to model scaling, our approach also enables us to search a new architecture for any new resource constraints. For example, some video applications may require model latency as low as 25ms. To meet such constraints, we can either scale a baseline model with smaller input size and depth multiplier, or we can also search for models more targeted to this new latency constraint. Figure 6 shows the performance comparison of these two approaches. We choose the best scaling parameters (depth multiplier=0.5, input size=192) from all possible combinations shown in (Sandler et al. 2018), and start a new search with the same scaled input size. For comparison, Figure 6 also shows the scaling parameter (0.5, 160) that has the best accuracy among all possible parameters under the smaller 17ms latency constraint. As shown in the figure, although our MnasNet already outperforms MobileNetV2 under the same scaling parameters, we can further improve the accuracy with a new architecture search targeting a 23ms latency constraint.

COCO Object Detection Performance

For COCO object detection (Lin et al. 2014), we pick the same MnasNet models in Table 1 and use them as the feature extractor for SSDLite, a modified resource-efficient version of SSD (Sandler et al. 2018). As recommended by (Sandler et al. 2018), we only compare our models with other SSD or YOLO detectors since our focus is on mobile devices with limited on-device computational resources.

Table 2 shows the performance of our MnasNet models on COCO. Results for YOLO and SSD are from (Redmon and Farhadi 2017), while results for MobileNet are from (Sandler et al. 2018). We train our MnasNet models on COCO trainval35k and evaluate them on test-dev2017 by submitting the results to COCO server. As shown in the table, our approach improves both the inference latency and the mAP quality (COCO challenge metrics) over MobileNet V1 and V2. For comparison, our slightly larger MnasNet-92 achieves comparable mAP quality (22.9 vs 23.2) as SSD300 (Liu et al. 2016) with **7**× **fewer** parameters and **35**× **fewer** multiply-add computations.

MnasNet Architecture and Discussions

Figure 7(a) illustrates the neural network architecture for our baseline MnasNet shown in Table 1. It consists of a sequence of linearly connected blocks, and each block is composed of different types of layers shown in Figure 7(b) - (f). As expected, it utilizes depthwise convolution extensively across all layers to maximize model computational efficiency. Furthermore, we also observe some interesting findings:

- **What's special about MnasNet?** In trying to better understand how MnasNet models are different from prior mobile CNN models, we noticed these models contain more 5x5 depthwise convolutions than prior work (Zhang et al. 2018; Huang et al. 2018; Sandler et al. 2018), where only 3x3 kernels are typically used. In fact, a 5x5 kernel

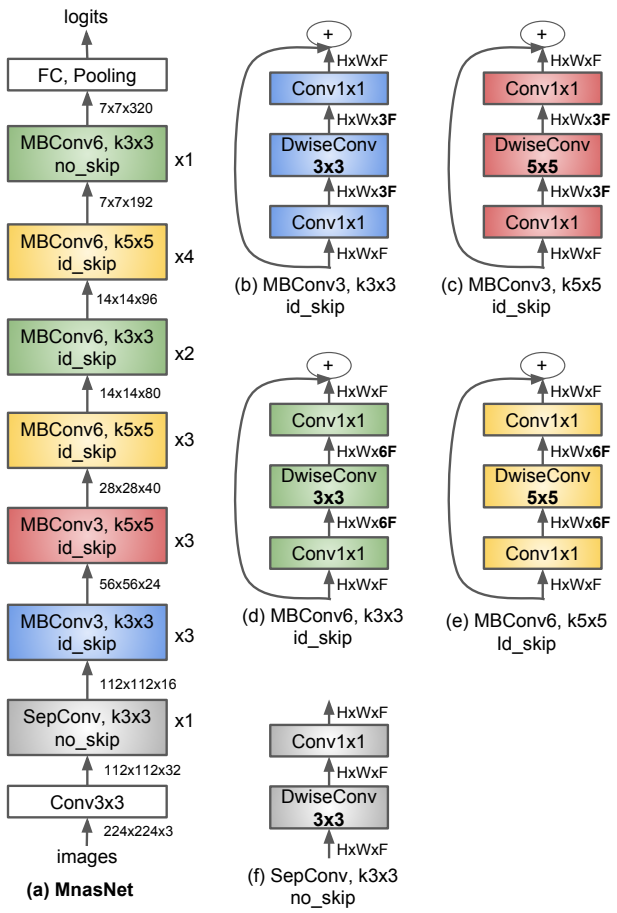


Figure 7: **MnasNet Architecture** – (a) is the MnasNet model shown in Table 1; (b) - (f) are the corresponding layers structure for MnasNet. *MBConv* denotes mobile inverted bottleneck conv, *SepConv* denotes depthwise separable conv, *k3x3* / *k5x5* denotes kernel size 3x3 or 5x5, *no_skip* / *id_skip* denotes no skip or identity residual skip, $H \times W \times F$ denotes the tensor shape of (height, width, depth), and $\times 1/2/3/4$ denotes the number of repeated layers within the block. All layers have stride 1, except the first layer of each block has stride 2 if input/output resolutions are different. Notably, (d) and (f) are also the basic building block of MobileNetV2 and MobileNetV1 respectively.

could indeed be more resource-efficient than two 3x3 kernels for depthwise separable convolution. Formally, given an input shape (H, W, M) and output shape (H, W, N), let $C_{5\times5}$ and $C_{3\times3}$ denote the computational cost measured by number of multiply-adds for depthwise separable convolution with kernel 5x5 and 3x3 respectively:

$$\begin{aligned} C_{5\times5} &= H * W * M * (25 + N) \\ C_{3\times3} &= H * W * M * (9 + N) \\ \implies C_{5\times5} &< 2 * C_{3\times3} \quad \text{if } N > 7 \end{aligned} \tag{6}$$

For the same effective receptive field, a 5x5 kernel has fewer multiply-adds than two 3x3 kernels when the input depth $N > 7$. Assuming the kernels are both reasonably optimized, this might explain why our MnasNet utilizes many 5x5 depthwise convolutions when both accuracy and latency are part of the optimization metric.

- **Is layer diversity important?** Most common mobile architectures typically repeat an architectural motif several times, only changing the filter sizes and spatial dimensions throughout the model. Our factorized, hierarchical search space allows the model to have different types of layers throughout the network, as shown in Figure 7(b), (c), (d), (e), and (f), whereas MobileNet V1 and V2 only uses building block (f) and (d) respectively. As an ablation study, Table 3 compares our MnasNet with its variants that repeat a single type of layer throughout the network. As shown in the table, MnasNet has much better accuracy-latency trade-offs over those variants, suggesting the importance of layer diversity in resource-constrained CNN models.

	Top-1 Acc.	CPU Latency
MnasNet	**74.0**	**76ms**
Figure 7 (b) only	71.3	67ms
Figure 7 (c) only	72.3	84ms
Figure 7 (d) only	74.1	123ms
Figure 7 (e) only	74.8	157ms

Table 3: **Performance Comparison of MnasNet and Its Variants** – *MnasNet* denotes the same model shown in Figure 7(a); *Figure 7(b)-7(e)* denote its 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.

Conclusion

This paper presents an automated neural architecture search approach for designing resource-efficient mobile CNN models using reinforcement learning. The key idea behind this method is to incorporate platform-aware *real-world latency information* into the search process and utilize 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 some interesting findings that will guide us in designing next-generation mobile CNN models.

Acknowledgments

We thank Vishy Tirumalashetty, Xiaoqiang Zheng, Megan Kacholia, and Jeff Dean for their support and valuable inputs, Guiheng Zhou and Wen Wang for help with the mobile infrastructure, Mark Sandler, Andrew Howard, Menglong Zhu, Dmitry Kalenichenko, Barret Zoph, and Sheng Li for helpful discussion, and the larger device automation platform team, TensorFlow Lite team, and Google Brain team.

References

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

Cai, H.; Chen, T.; Zhang, W.; Yu, Y.; and Wang, J. 2018. Reinforcement learning for architecture search by network transformation. *AAAI*.

Deb, K. 2014. Multi-objective optimization. *Search methodologies* 403–449.

Dong, J.-D.; Cheng, A.-C.; Juan, D.-C.; Wei, W.; and Sun, M. 2018. PPP-Net: Platform-aware progressive search for pareto-optimal neural architectures. *ICLR workshop*.

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

Gholami, A.; Kwon, K.; Wu, B.; Tai, Z.; Yue, X.; Jin, P.; Zhao, S.; and Keutzer, K. 2018. Squeezenext: Hardware-aware neural network design. *arXiv preprint arXiv:1803.10615*.

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

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*.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

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

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*.

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

Hu, J.; Shen, L.; and Sun, G. 2018. Squeeze-and-excitation networks. *ICVPR*.

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

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

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

Khosla, A.; Jayadevaprakash, N.; Yao, B.; and Fei-Fei, L. 2011. Novel dataset for fine-grained image categorization. *CVPR workshop*.

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

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

Liu, C.; Zoph, B.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K. 2018a. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*.

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

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*.

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

Redmon, J., and Farhadi, A. 2017. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*.

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

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

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

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

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

Zhang, X.; Zhou, X.; Lin, M.; and Sun, J. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*.

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

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

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