# Searching for MobileNetV3

Andrew Howard[1]     Mark Sandler[1]     Grace Chu[1]     Liang-Chieh Chen[1]     Bo Chen[1]     Mingxing Tan[2]

Weijun Wang[1]     Yukun Zhu[1]     Ruoming Pang[2]     Vijay Vasudevan[2]     Quoc V. Le[2]     Hartwig Adam[1]

[1]Google Research, [2]Google Brain

{howarda, sandler, cxy, lcchen, bochen, tanmingxing, weijunw, yukun, rpang, vrv, qvl, hadam}@google.com

## Abstract

*We present the next generation of MobileNets based on a combination of complementary search techniques as well as a novel architecture design. MobileNetV3 is tuned to mobile phone CPUs through a combination of hardware-aware network architecture search (NAS) complemented by the NetAdapt algorithm and then subsequently improved through novel architecture advances. This paper starts the exploration of how automated search algorithms and network design can work together to harness complementary approaches improving the overall state of the art. Through this process we create two new MobileNet models for release: MobileNetV3-Large and MobileNetV3-Small which are targeted for high and low resource use cases. These models are then adapted and applied to the tasks of object detection and semantic segmentation. For the task of semantic segmentation (or any dense pixel prediction), we propose a new efficient segmentation decoder Lite Reduced Atrous Spatial Pyramid Pooling (LR-ASPP). We achieve new state of the art results for mobile classification, detection and segmentation. MobileNetV3-Large is 3.2% more accurate on ImageNet classification while reducing latency by 15% compared to MobileNetV2. MobileNetV2-Small is 4.6% more accurate while reducing latency by 5% compared to MobileNetV2. MobileNetV3-Large detection is 25% faster at roughly the same accuracy as MobileNetV2 on COCO detection. MobileNetV3-Large LR-ASPP is 30% faster than MobileNetV2 R-ASPP at similar accuracy for Cityscapes segmentation.*
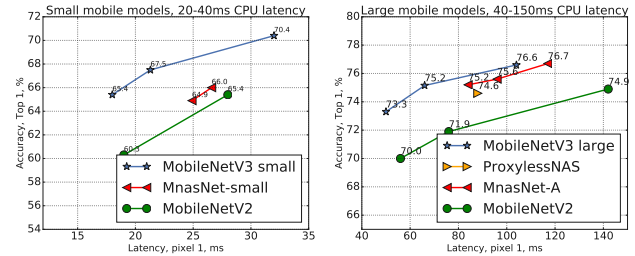
Figure 1. The trade-off between Pixel 1 latency and top-1 ImageNet accuracy. All models use the input resolution 224. V3 large and V3 small use multipliers 0.75, 1 and 1.25 to show optimal frontier. All latencies were measured on a single large core of the same device using TFLite[1]. MobileNetV3-Small and Large are our proposed next-generation mobile models.
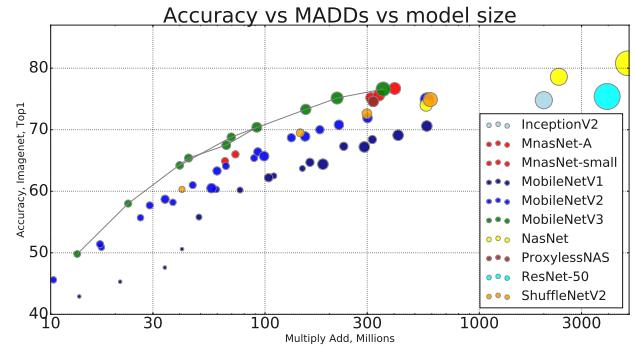


Figure 2. The trade-off between MAdds and top-1 accuracy. This allows to compare models that were targeted different hardware or software frameworks. All MobileNetV3 are for input resolution 224 and use multipliers 0.35, 0.5, 0.75, 1 and 1.25. See section 6 for other resolutions. Best viewed in color.

## 1. Introduction

Efficient neural networks are becoming ubiquitous in mobile applications enabling entirely new on-device experiences. They are also a key enabler of personal privacy allowing a user to gain the benefits of neural networks without needing to send their data to the server to be evaluated. Advances in neural network efficiency not only improve user experience via higher accuracy and lower latency, but also help preserve battery life through reduced power consumption.

This paper describes the approach we took to develop MobileNetV3 Large and Small models in order to deliver the next generation of high accuracy efficient neural network models to power on-device computer vision. The new networks push the state of the art forward and demonstrate how to blend automated search with novel architecture advances to build effective models.

1

The goal of this paper is to develop the best possible mobile computer vision architectures optimizing the accuracy-latency trade off on mobile devices. To accomplish this we introduce (1) complementary search techniques, (2) new efficient versions of nonlinearities practical for the mobile setting, (3) new efficient network design, (4) a new efficient segmentation decoder. We present thorough experiments demonstrating the efficacy and value of each technique evaluated on a wide range of use cases and mobile phones.

The paper is organized as follows. We start with a discussion of related work in Section 2. Section 3 reviews the efficient building blocks used for mobile models. Section 4 reviews architecture search and the complementary nature of MnasNet and NetAdapt algorithms. Section 5 describes novel architecture design improving on the efficiency of the models found through the joint search. Section 6 presents extensive experiments for classification, detection and segmentation in order do demonstrate efficacy and understand the contributions of different elements. Section 7 contains conclusions and future work.

## 2. Related Work

Designing deep neural network architecture for the optimal trade-off between accuracy and efficiency has been an active research area in recent years. Both novel hand-crafted structures and algorithmic neural architecture search have played important roles in advancing this field.

SqueezeNet[20] extensively uses 1x1 convolutions with squeeze and expand modules primarily focusing on reducing the number of parameters. More recent works shifts the focus from reducing parameters to reducing the number of operations (MAdds) and the actual measured latency. MobileNetV1[17] employs depthwise separable convolution to substantially improve computation efficiency. MobileNetV2[37] expands on this by introducing a resource-efficient block with inverted residuals and linear bottlenecks. ShuffleNet[47] utilizes group convolution and channel shuffle operations to further reduce the MAdds. CondenseNet[19] learns group convolutions at the training stage to keep useful dense connections between layers for feature re-use. ShiftNet[44] proposes the shift operation interleaved with point-wise convolutions to replace expensive spatial convolutions.

To automate the architecture design process, reinforcement learning (RL) was first introduced to search efficient architectures with competitive accuracy [51, 52, 3, 25, 33]. A fully configurable search space can grow exponentially large and intractable. So early works of architecture search focus on the cell level structure search, and the same cell is reused in all layers. Recently, [41] explored a block-level hierarchical search space allowing different layer structures at different resolution blocks of a network. To reduce the computational cost of search, differentiable architecture

search framework is used in [26, 5, 43] with gradient-based optimization. Focusing on adapting existing networks to constrained mobile platforms, [46, 14, 12] proposed more efficient automated network simplification algorithms.

Quantization [21, 23, 45, 39, 49, 50, 35] is another important complementary effort to improve the network efficiency through reduced precision arithmetic. Finally, knowledge distillation [4, 15] offers an additional complementary method to generate small accurate "student" networks with the guidance of a large "teacher" network.

## 3. Efficient Mobile Building Blocks

Mobile models have been built on increasingly more efficient building blocks. MobileNetV1 [17] introduced *depthwise separable convolutions* as an efficient replacement for traditional convolution layers. Depthwise separable convolutions effectively factorize traditional convolution by separating spatial filtering from the feature generation mechanism. Depthwise separable convolutions are defined by two separate layers: light weight depthwise convolution for spatial filtering and heavier 1x1 pointwise convolutions for feature generation.

MobileNetV2 [37] introduced the linear bottleneck and inverted residual structure in order to make even more efficient layer structures by leveraging the low rank nature of the problem. This structure is shown on Figure 3 and is defined by a 1x1 expansion convolution followed by depthwise convolutions and a 1x1 projection layer. The input and output are connected with a residual connection if and only if they have the same number of channels. This structure maintains a compact representation at the input and the output while expanding to a higher-dimensional feature space internally to increase the expressiveness of non-linear per-channel transformations.

MnasNet [41] built upon the MobileNetV2 structure by introducing lightweight attention modules based on squeeze and excitation into the bottleneck structure. Note that the squeeze and excitation module are integrated in a different location than ResNet based modules proposed in [18]. The module is placed after the depthwise filters in the expansion in order for attention to be applied on the largest representation as shown on Figure 4.

For MobileNetV3, we use a combination of these layers as building blocks in order to build the most effective models. Layers are also upgraded with modified swish nonlinearities [34]. Both squeeze and excitation as well as the swish nonlinearity use the sigmoid which can be inefficient to compute as well challenging to maintain accuracy in fixed point arithmetic so we replace this with the hard sigmoid [2, 11] as discussed in section 5.2.
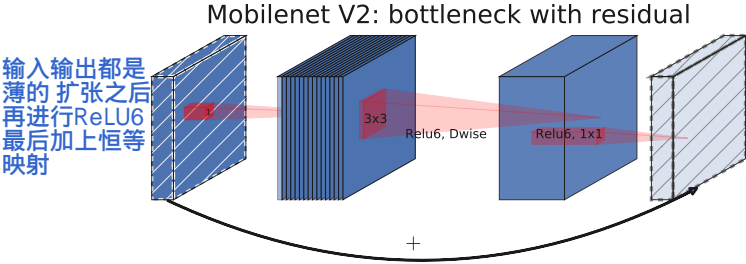
Figure 3. MobileNetV2 [37] layer (Inverted Residual and Linear Bottleneck). Each block consists of narrow input and output (bottleneck), which don't have non-linearity, followed by expansion to a much higher-dimensional space and projection to the output. The residual connects bottleneck (rather than expansion).
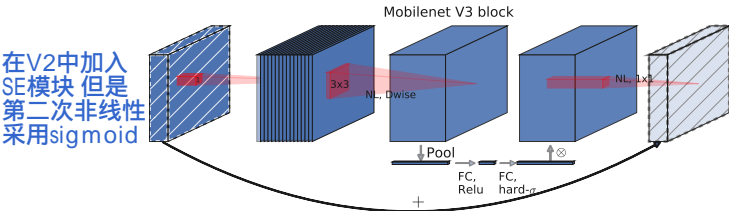


Figure 4. MobileNetV2 + Squeeze-and-Excite [18]. In contrast with [18] we apply the squeeze and excite in the residual layer. We use different non-linearity depending on the layer, see section 5.2 for details.

## 4. Network Search

Network search has shown itself to be a very powerful tool for discovering and optimizing network architectures [51, 41, 5, 46]. For MobileNetV3 we use platform-aware NAS to search for the global network structures by optimizing each network block. We then use the NetAdapt algorithm to search per layer for the number of filters. These techniques are complementary and can be combined to effectively find optimized models for a given hardware platform.

### 4.1. Platform-Aware NAS for Block-wise Search

Similar to [41], we employ a platform-aware neural architecture approach to find the global network structures. Since we use the same RNN-based controller and the same factorized hierarchical search space, we find similar results as [41] for Large mobile models with target latency around 80ms. Therefore, we simply reuse the same MnasNet-A1 [41] as our initial Large mobile model, and then apply NetAdapt [46] and other optimizations on top of it.

However, we observe the original reward design is not optimized for small mobile models. Specifically, it uses a multi-objective reward $ACC(m) \times [LAT(m)/TAR]^w$ to approximate Pareto-optimal solutions, by balancing model accuracy $ACC(m)$ and latency $LAT(m)$ for each model $m$ based on the target latency $TAR$. We observe that ac-

curacy changes much more dramatically with latency for small models; therefore, we need a smaller weight factor $w = -0.15$ (vs the original $w = -0.07$ in [41]) to compensate for the larger accuracy change for different latencies. Enhanced with this new weight factor $w$, we start a new architecture search from scratch to find the initial seed model and then apply NetAdapt and other optimizations to obtain the final MobilenetV3-Small model.

### 4.2. NetAdapt for Layer-wise Search

The second technique that we employ in our architecture search is NetAdapt [46]. This approach is complimentary to platform-aware NAS: it allows fine-tuning of individual layers in a sequential manner, rather than trying to infer coarse but global architecture. We refer to the original paper for the full details. In short the technique proceeds as follows:

1. Starts with a seed network architecture found by platform-aware NAS.

2. For each step:

   (a) Generate a set of new *proposals*. Each proposal represents a modification of an architecture that generates at least $\delta$ reduction in latency compared to the previous step.

   (b) For each proposal we use the pre-trained model from the previous step and populate the new proposed architecture, truncating and randomly initializing missing weights as appropriate. Fine-tune each proposal for $T$ steps to get a coarse estimate of the accuracy.

   (c) Selected best proposal according to some metric.

3. Iterate previous step until target latency is reached.

In [46] the metric was to minimize the accuracy change. We modify this algorithm and minimize the ratio between latency change and accuracy change. That is for all proposals generated during each NetAdapt step, we pick one that maximizes: $\frac{\Delta \text{Acc}}{|\Delta \text{latency}|}$, with $\Delta$latency satisfying the constraint in 2(a). The intuition is that because our proposals are discrete, we prefer proposals that maximize the slope of the trade-off curve.

This process is repeated until the latency reaches its target, and then we re-train the new architecture from scratch. We use the same proposal generator as was used in [46] for MobilenetV2. Specifically, we allow the following two types of proposals:

1. Reduce the size of any expansion layer;

2. Reduce bottleneck in all blocks that share the same bottleneck size - to maintain residual connections.

For our experiments we used $T = 10000$ and find that while it increases the accuracy of the initial fine-tuning of the proposal. However, it does not generally change the final accuracy, when trained from scratch. We set $\delta = 0.01|L|$, where $L$ is the latency of the seed model.

## 5. Network Improvements

In addition to network search, we also introduce several new components to the model to further improve the final model. We redesign the computionally-expensive layers at the beginning and the end of the network. We also introduce a new nonlinearity, h-swish, a modified version of the recent swish nonlinearity, which is faster to compute and more quantization-friendly.

### 5.1. Redesigning Expensive Layers

Once models are found through architecture search, we observe that some of the last layers as well as some of the earlier layers are more expensive than others. We propose some modifications to the architecture to reduce the latency of these slow layers while maintaining the accuracy. These modifications are outside of the scope of the current search space.

The first modification reworks how the last few layers of the network interact in order to produce the final features more efficiently. Current models based on MobileNetV2's inverted bottleneck structure and variants use 1x1 convolution as a final layer in order to expand to a higher-dimensional feature space. This layer is critically important in order to have rich features for prediction. However, this comes at a cost of extra latency.

To reduce latency and preserve the high dimensional features, we move this layer past the final average pooling. This final set of features is now computed at 1x1 spatial resolution instead of 7x7 spatial resolution. The outcome of this design choice is that the computation of the features becomes nearly free in terms of computation and latency.

Once the cost of this feature generation layer has been mitigated, the previous bottleneck projection layer is no longer needed to reduce computation. This observation allows us to remove the projection and filtering layers in the previous bottleneck layer, further reducing computational complexity. The original and optimized last stages can be seen in figure 5. The efficient last stage reduces the latency by 10 milliseconds which is 15% of the running time and reduces the number of operations by 30 millions MAdds with almost no loss of accuracy. Section 6 contains detailed results.

Another expensive layer is the initial set of filters. Current mobile models tend to use 32 filters in a full 3x3 convolution to build initial filter banks for edge detection. Often these filters are mirror images of each other. We experimented with reducing the number of filters and using
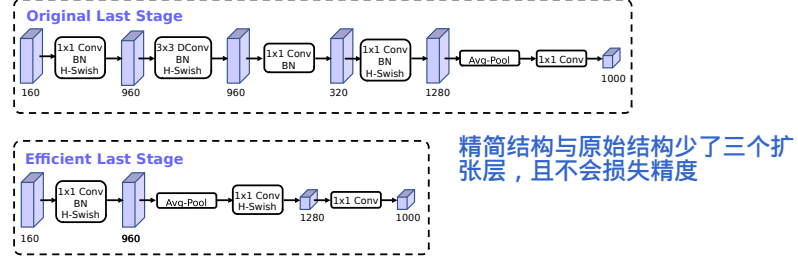


Figure 5. Comparison of original last stage and efficient last stage. This more efficient last stage is able to drop three expensive layers at the end of the network at no loss of accuracy.

different nonlinearities to try and reduce redundancy. We settled on using the hard swish nonlinearity for this layer as it performed as well as other nonlinearities tested. We were able to reduce the number of filters to 16 while maintaining the same accuracy as 32 filters using either ReLU or swish. This saves an additional 3 milliseconds and 10 million MAdds.

### 5.2. Nonlinearities

In [34] a non-linearity called *swish* was introduced that when used as a drop-in replacement for $\mathrm{ReLU}$, that significantly improves the accuracy of neural networks. The non-linearity is defined as

$$\mathrm{swish}\, x = x \cdot \sigma(x) \qquad \text{SE}$$

While this nonlinearity improves accuracy, it comes with non-zero cost in embedded environments as the sigmoid function is much more expensive to compute on mobile devices. We deal with this problem in two ways.

**1.** We replace sigmoid function with its piece-wise linear hard analog: $\frac{\mathrm{ReLU6}(x+3)}{6}$ similar to [11, 42]. The minor difference is we use $\mathrm{ReLU6}$ rather than a custom clipping constant. Similarly, the hard version of swish becomes

$$\mathrm{h\text{-}swish}[x] = x\frac{\mathrm{ReLU6}(x+3)}{6}$$

A similar version of hard-swish was also recently proposed in [2]. The comparison of the soft and hard version of sigmoid and swish non-linearities is shown in figure 6. Our choice of constants was motivated by simplicity and being a good match to the original smooth version. In our experiments, we found hard-version of all these functions to have no discernible difference in accuracy, but multiple advantages from a deployment perspective. First, optimized implementations of $\mathrm{ReLU6}$ are available on virtually all software and hardware frameworks. Second, in quantized mode, it eliminates potential numerical precision loss
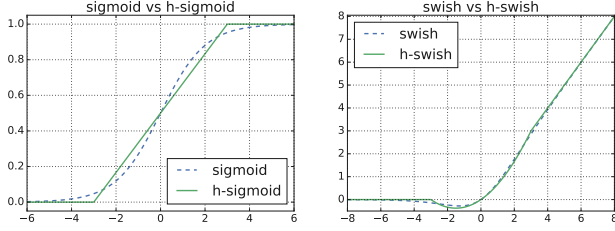
Figure 6. Sigmoid and swish non-linearities and ther "hard" counterparts.

caused by different implementations of the approximate sigmoid. Finally, even optimized implementations of quantized sigmoid tend to be far slower than their ReLU counterparts. In our experiments, replacing h-swish with swish in quantized mode increased inference latency by 15%[1].

**2.** The cost of applying nonlinearity decreases as we go deeper into the network, since each layer activation memory typically halves every time the resolution drops. Incidentally, we find that most of the benefits swish are realized by using them only in the deeper layers. Thus in our architectures we only use h-swish at the second half of the model. We refer to the tables 1 and 2 for the precise layout.

Even with these optimization h-swish still introduces some latency cost. However as we demonstrate in section 6 the net effect on accuracy and latency is positive, and it provides a venue for further software optimization: once smooth sigmoid is replaced by piece-wise linear function, most of the overhead is in memory accesses, which could be eliminated by fusing the nonlinearities with the previous layers.

### 5.2.1 Large squeeze-and-excite

In [41], the size of the squeeze-and-excite bottleneck was relative the size of the convolutional bottleneck. Instead, we replace them all to fixed to be 1/4 of the number of channels in expansion layer. We find that doing so increases the accuracy, at the modest increase of number of parameters, and no discernible latency cost.

### 5.3. MobileNetV3 Definitions

MobileNetV3 is defined as two models: MobileNetV3-Large and MobileNetV3-Small. These models are targeted at high and low resource use cases respectively. The models are created through applying platform-aware NAS and NetAdapt for network search and incorporating the network improvements defined in this section. See table 1 and 2 for full specification of our networks.

---

[1]In floating point mode, memory access dominates the latency cost.

| Input | Operator | exp size | #out | SE | NL | s |
|---|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d | - | 16 | - | HS | 2 |
| $112^2 \times 16$ | bneck, 3x3 | 16 | 16 | - | RE | 1 |
| $112^2 \times 16$ | bneck, 3x3 | 64 | 24 | - | RE | 2 |
| $56^2 \times 24$ | bneck, 3x3 | 72 | 24 | - | RE | 1 |
| $56^2 \times 24$ | bneck, 5x5 | 72 | 40 | ✓ | RE | 2 |
| $28^2 \times 40$ | bneck, 5x5 | 120 | 40 | ✓ | RE | 1 |
| $28^2 \times 40$ | bneck, 5x5 | 120 | 40 | ✓ | RE | 1 |
| $28^2 \times 40$ | bneck, 3x3 | 240 | 80 | - | HS | 2 |
| $14^2 \times 80$ | bneck, 3x3 | 200 | 80 | - | HS | 1 |
| $14^2 \times 80$ | bneck, 3x3 | 184 | 80 | - | HS | 1 |
| $14^2 \times 80$ | bneck, 3x3 | 184 | 80 | - | HS | 1 |
| $14^2 \times 80$ | bneck, 3x3 | 480 | 112 | ✓ | HS | 1 |
| $14^2 \times 112$ | bneck, 3x3 | 672 | 112 | ✓ | HS | 1 |
| $14^2 \times 112$ | bneck, 5x5 | 672 | 160 | ✓ | HS | 1 |
| $14^2 \times 112$ | bneck, 5x5 | 672 | 160 | ✓ | HS | 2 |
| $7^2 \times 160$ | bneck, 5x5 | 960 | 160 | ✓ | HS | 1 |
| $7^2 \times 160$ | conv2d, 1x1 | - | 960 | - | HS | 1 |
| $7^2 \times 960$ | Pool, 7x7 | - | - | - | HS | - |
| $1^2 \times 960$ | conv2d 1x1, NBN | - | 1280 | - | HS | 1 |
| $1^2 \times 1280$ | conv2d 1x1, NBN | - | k | - | - | - |

Table 1. Specification for MobileNetV3-Large. SE denotes whether there is a Squeeze-And-Excite in that block. NL denotes the type of non-linearity used. Here, HS denotes h-swish and RE denotes ReLU. NBN denotes no batch normalization. $s$ denotes stride.

| Input | Operator | exp size | #out | SE | NL | s |
|---|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d, 3x3 | - | 16 | - | HS | 2 |
| $112^2 \times 24$ | bneck, 3x3 | 16 | 16 | ✓ | RE | 2 |
| $56^2 \times 24$ | bneck, 3x3 | 72 | 24 | - | RE | 2 |
| $28^2 \times 24$ | bneck, 3x3 | 88 | 24 | - | RE | 1 |
| $28^2 \times 40$ | bneck, 5x5 | 96 | 40 | ✓ | HS | 1 |
| $14^2 \times 40$ | bneck, 5x5 | 240 | 40 | ✓ | HS | 1 |
| $14^2 \times 40$ | bneck, 5x5 | 240 | 40 | ✓ | HS | 1 |
| $14^2 \times 40$ | bneck, 5x5 | 120 | 48 | ✓ | HS | 1 |
| $14^2 \times 48$ | bneck, 5x5 | 144 | 48 | ✓ | HS | 1 |
| $14^2 \times 96$ | bneck, 5x5 | 288 | 96 | ✓ | HS | 2 |
| $7^2 \times 96$ | bneck, 5x5 | 576 | 96 | ✓ | HS | 1 |
| $7^2 \times 96$ | bneck, 5x5 | 576 | 96 | ✓ | HS | 1 |
| $7^2 \times 96$ | conv2d, 1x1 | - | 576 | ✓ | HS | 1 |
| $7^2 \times 576$ | Pool, 7x7 | - | - | - | HS | 7 |
| $1^2 \times 576$ | conv2d 1x1 | - | 1280 | - | HS | 1 |
| $1^2 \times 1280$ | conv2d 1x1 | - | k | - | HS | |

Table 2. Specification for MobileNetV3-Small. See table 1 for notation.

## 6. Experiments

We present experimental results to demonstrate the effectiveness of the new MobileNetV3 models. We report results on classification, detection and segmentation. We also report various ablation studies to shed light on the effects of various design decisions.

### 6.1. Classification

As has become standard, we use ImageNet[36] for all our classification experiments and compare accuracy versus various measures of resource usage such as latency and multiply adds (MAdds).

| Network | Top-1 | Madds | Params | P-1 | P-2 | P-3 |
|---------|-------|-------|--------|-----|-----|-----|
| V3-Large 1.0 | **75.2** | **219** | 5.4M | **66** | **77** | **52.6** |
| V3-Large 0.75 | 73.3 | 155 | 4M | 50 | 61 | 40.1 |
| Mnasnet-A1 | 75.2 | 315 | 3.9 | 84 | 93 | 64 |
| Proxyless[5] | 74.6 | 320 | 4M | 88 | 98 | 65 |
| V2 1.0 | 72.0 | 300 | 3.4M | 78 | 90 | 61.3 |
| V3-Small 1.0 | 67.4 | 66 | 2.9M | 21.3 | 26 | 17.7 |
| V3-Small 0.75 | 65.4 | 44 | 2.4M | 18 | 21.06 | 15.1 |
| Mnasnet [41] | 64.9 | 65.1 | 1.9M | 24.9 | 28.1 | 18.3 |
| V2 0.35 | 60.8 | 59.2 | 1.6M | 19 | 22 | 14.5 |

Table 3. Floating point performance on Pixel family of phones ("P-$n$" denotes a Pixel-$n$ phone). All latencies are in ms. The inference latency is measured using a single large core with a batch size of 1.

### 6.1.1 Training setup

We train our models using synchronous training setup on 4x4 TPU Pod [22] using standard tensorflow RMSPropOptimizer with 0.9 momentum. We use the initial learning rate of 0.1, with batch size 4096 (128 images per chip), with learning rate decay rate of 0.01 every 3 epochs. We use dropout of 0.8, and l2 weight decay 1e-5. For the same image preprocessing as Inception [40]. Finally we use exponential moving average with decay 0.9999. All our convolutional layers use batch-normalization layers with average decay of 0.99.

### 6.1.2 Measurement setup

To measure latencies we use standard Google Pixel phones and run all networks through the standard TFLite Benchmark Tool. We use single-threaded large core in all our measurements. We don't report multi-core inference time, since we find this setup not very practical for mobile applications.

### 6.2. Results

As can be seen on figure 1 our models outperform the current state of the art such as MnasNet [41], ProxylessNas [5] and MobileNetV2 [37]. We report the floating point performance on different Pixel phones in the table 3. We include quantization results in table 4.

In figure 7 we show the MobileNetV3 performance trade-offs as a function of multiplier and resolution. Note how MobileNetV3-Small outperforms the MobileNetV3-Large with multiplier scaled to match the performance by nearly 3%. On the other hand, resolution provides an even better trade-offs than multiplier. However, it should be noted that resolution is often determined by the problem (e.g. segmentation and detection problem generally require higher resolution), and thus can't always be used as a tunable parameter.

| Network | Top-1 | P-1 | P-2 | P-3 |
|---------|-------|-----|-----|-----|
| **V3-Large 1.0** | **73.8** | 57 | 55.9 | 38.9 |
| V2 1.0 | 70.9 | 62 | 53.6 | 38.4 |
| **V3-Small** | **64.9** | 21 | 20.5 | 13.9 |
| V2 0.35 | 57.2 | 20.6 | 18.6 | 13.1 |

Table 4. Quantized performance. All latencies are in ms. The inference latency is measured using single large core on the respective Pixel 1/2/3 device.
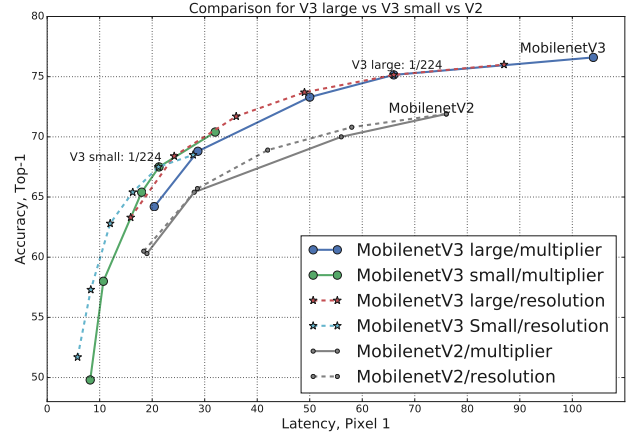


Figure 7. Performance of MobilenetV3 as a function of different multipliers and resolutions. In our experiments we have used multipliers 0.35, 0.5, 0.75, 1.0 and 1.25, with a fixed resolution of 224, and resolutions 96, 128, 160, 192, 224 and 256 with a fixed depth multiplier of 1.0. Best viewed in color.

### 6.2.1 Ablation study

**Impact of non-linearities** In table 5 and figure 8 we show how the decision where to insert h-swish affects the latency. Of particular importance we note that using h-swish on the entire network results in slight increase of accuracy (0.2), while adding nearly 20% in latency, and again ending up under the efficient frontier.

On the other hand using h-swish moves the efficient frontier up compared to ReLU despite still being about 12% more expensive. Finally, we note, that as h-swish gets optimized by fusing it into the convolutional operator, we expect the latency gap between h-swish and ReLU to drop significantly if not disappear. However, such improvement can't be expected between h-swish and swish, since computing sigmoid is inherently more expensive.

**Impact of other components** In figure 9 we show how introduction of different components moved along the latency/accuracy curve.

### 6.3. Detection

We use MobileNetV3 as a drop-in replacement for the backbone feature extractor in SSDLite [37] and compare

| | Top 1 | Latency P-1 |
|---|---|---|
| V3 | 75.2 | 66 |
| 0.85 V3 | 74.3 | 55 |
| ReLU | 74.5 (-.7%) | 59 (-12%) |
| h-swish @16 | 75.4 (+.2 %) | 78 (+20%) |
| h-swish @112 | 75.0 (-.3%) | 64 (-3%) |

Table 5. Effect of non-linearities on MobileNetV3-Large. In h-swish @$N$, $N$ denotes the number of channels, in the first layer that has h-swish enabled.
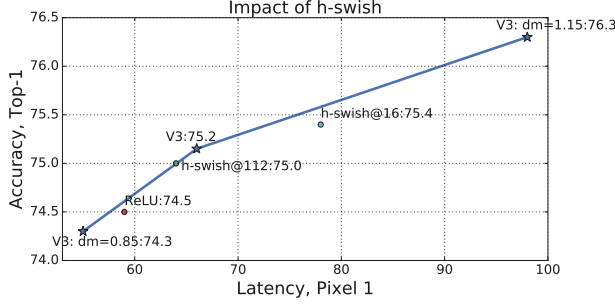


Figure 8. Impact of h-swish vs swish vs ReLU on latency. The curve shows a frontier of using depth multiplier. Note that placing h-swish at all layer with 112 channels or more moves along the optimal frontier.
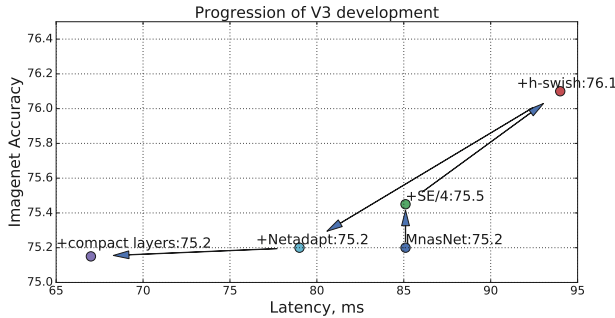


Figure 9. Impact of adding individual components to the network architecture.

with other backbone networks on COCO dataset [24].

Following MobileNetV2 [37], we attach the first layer of SSDLite to the last feature extractor layer that has an output stride of 16, and attach the second layer of SSDLite to the last feature extractor layer that has an output stride of 32. Following the detection literature, we refer to these two feature extractor layers as $C4$ and $C5$, respectively. For MobileNetV3-Large, $C4$ is the expansion layer of the 13-th bottleneck block. For MobileNetV3-Small, $C4$ is the expansion layer of the 9-th bottleneck block. For both networks, $C5$ is the layer immediately before pooling.

We additionally reduce the channel counts of all feature layers between $C4$ and $C5$ by 2. This is because the last few layers of MobileNetV3 are tuned to output 1000 classes, which may be redundant when transferred to COCO with

| Backbone | mAP | Latency (ms) | Params (M) | MAdd (B) |
|---|---|---|---|---|
| V1 | 22.2 | 270 | 5.1 | 1.3 |
| V2 | 22.1 | 200 | 4.3 | 0.80 |
| Mnasnet | 23.0 | 215 | 4.88 | 0.84 |
| V3 | 22.0 | 173 | 4.97 | 0.62 |
| **V3**† | 22.0 | 150 | 3.22 | 0.51 |
| V2 0.35 | 13.7 | 54.8 | 0.93 | 0.16 |
| V2 0.5 | 16.6 | 80.4 | 1.54 | 0.27 |
| Mnasnet 0.35 | 15.6 | 58.2 | 1.02 | 0.18 |
| Mnasnet 0.5 | 18.5 | 86.2 | 1.68 | 0.29 |
| V3-Small | 16.0 | 67.2 | 2.49 | 0.21 |
| **V3-Small**† | 16.1 | 56 | 1.77 | 0.16 |

Table 6. Object detection results of SSDLite with different backbones on COCO test set. †: Channels in the blocks between $C4$ and $C5$ are reduced by a factor of 2.
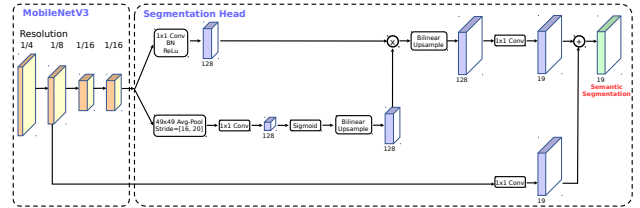


Figure 10. Building on MobileNetV3, the proposed segmentation head, Lite R-ASPP, delivers fast semantic segmentation results.

90 classes.

The results on COCO test set are given in Tab. 6. With the channel reduction, MobileNetV3-Large is $25\%$ faster than MobileNetV2 with near identical mAP. MobileNetV3-Small with channel reduction is also $2.4$ and $0.5$ mAP higher than MobileNetV2 and MnasNet at similar latency. For both MobileNetV3 models the channel reduction trick contributes to approximately $15\%$ latency reduction with no mAP loss, suggesting that Imagenet classification and COCO object detection may prefer different feature extractor shapes.

### 6.4. Semantic Segmentation

In this subsection, we employ MobileNetV2 [37] and the proposed MobileNetV3 as network backbones for the task of *mobile* semantic segmentation. Additionally, we compare two segmentation heads. The first one, referred to as R-ASPP, was proposed in [37]. R-ASPP is a *reduced* design of the Atrous Spatial Pyramid Pooling module [7, 8, 9], which adopts only two branches consisting of a $1 \times 1$ convolution and a global-average pooling operation [27, 48]. In this work, we propose another light-weight segmentation head, referred to as Lite R-ASPP (or LR-ASPP), as shown in Fig. 10. Lite R-ASPP, improving over R-ASPP, deploys the global-average pooling in a fashion similar to the Squeeze-and-Excitation module [18], in which we employ a large pooling kernel with a large stride (to save some

computation) and only one $1 \times 1$ convolution in the module. We apply atrous convolution [16, 38, 31, 6] to the last block of MobileNetV3 to extract denser features, and further add a skip connection [28] from low-level features to capture more detailed information.

We conduct the experiments on the Cityscapes dataset [10] with metric mIOU [13], and only exploit the 'fine' annotations. We employ the same training protocol as [8, 37]. All our models are trained from scratch without pretraining on ImageNet [36], and are evaluated with a *single-scale* input. Similar to object detection, we observe that we could reduce the channels in the last block of network backbone by a factor of 2 without degrading the performance significantly. We think it is because the backbone is designed for 1000 classes ImageNet image classification [36] while there are only 19 classes on Cityscapes, implying there is some channel redundancy in the backbone.

We report our Cityscapes validation set results in Tab. 7. As shown in the table, we observe that (1) reducing the channels in the last block of network backbone by a factor of 2 significantly improves the speed while maintaining similar performances (row 1 vs. row 2, and row 5 vs. row 6), (2) the proposed segmentation head LR-ASPP is slightly faster than R-ASPP [37] while performance is improved (row 2 vs. row 3, and row 6 vs. row 7), (3) reducing the filters in the segmentation head from 256 to 128 improves the speed at the cost of slightly worse performance (row 3 vs. row 4, and row 7 vs. row 8), (4) when employing the same setting, MobileNetV3 model variants attain similar performance while being slightly faster than MobileNetV2 counterparts (row 1 vs. row 5, row 2 vs. row 6, row 3 vs. row 7, and row 4 vs. row 8), (5) MobileNetV3-Small attains similar performance as MobileNetV2-0.5 while being faster, and (6) MobileNetV3-Small is significantly better than MobileNetV2-0.35 while yielding similar speed.

Tab. 8 shows our Cityscapes test set results. Our segmentation models with MobileNetV3 as network backbone significantly outperforms ESPNetv2 [30], CCC2 [32], and ESPNetv1 [30] by 10.5%, 10.6%, 12.3%, respectively while being faster in terms of Madds. The performance drops slightly by 0.6% when not employing the atrous convolution to extract dense feature maps in the last block of MobileNetV3, but the speed is improved to 1.98B (for half-resolution input), which is 1.7, 1.59, and 2.24 times faster than ESPNetv2, CCC2, and ESPNetv1, respectively. Furthermore, our models with MobileNetV3-Small as network backbone still outperforms all of them by at least a healthy margin of 6.2%. Our fastest model variant is 13.6% better than ESPNetv2-small with a slightly faster inference speed.

## 7. Conclusions and future work

In this paper we introduced MobileNetV3 Large and Small models demonstrating new state of the art in mo-

| N | Backbone | RF2 | SH | F | mIOU | Params | Madds | CPU (f) | CPU (h) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | V2 | - | × | 256 | 72.84 | 2.11M | 21.29B | 4.20s | 1.07s |
| 2 | V2 | ✓ | × | 256 | 72.56 | 1.15M | 13.68B | 3.23s | 819ms |
| 3 | V2 | ✓ | ✓ | 256 | 72.97 | 1.02M | 12.83B | 3.16s | 808ms |
| 4 | V2 | ✓ | ✓ | 128 | 72.74 | 0.98M | 12.57B | 3.12s | 797ms |
| 5 | V3 | - | × | 256 | 72.64 | 3.60M | 18.43B | 4.17s | 1.07s |
| 6 | V3 | ✓ | × | 256 | 71.91 | 1.76M | 11.24B | 3.01s | 765ms |
| 7 | V3 | ✓ | ✓ | 256 | 72.37 | 1.63M | 10.33B | 2.96s | 750ms |
| 8 | V3 | ✓ | ✓ | 128 | 72.36 | 1.51M | 9.74B | 2.87s | 730ms |
| 9 | V2 0.5 | ✓ | ✓ | 128 | 68.57 | 0.28M | 4.00B | 1.61s | 402ms |
| 10 | V2 0.35 | ✓ | ✓ | 128 | 66.83 | 0.16M | 2.54B | 1.31s | 323ms |
| 11 | V3-Small | ✓ | ✓ | 128 | 68.38 | 0.47M | 2.90B | 1.38s | 349ms |

Table 7. Semantic segmentation results on Cityscapes *val* set. **RF2**: **R**educe the **F**ilters in the last block by a factor of **2**. V2 0.5 and V2 0.35 are MobileNetV2 with depth multiplier = 0.5 and 0.35, respectively. **SH:** **S**egmentation **H**ead, where × employs the R-ASPP while ✓ employs the proposed LR-ASPP. **F:** Number of **F**ilters used in the Segmentation Head. **CPU (f):** CPU time measured on a single large core of Pixel 3 (floating point) w.r.t. a **full-resolution** input (i.e., $1024 \times 2048$). **CPU (h):** CPU time measured w.r.t. a **half-resolution** input (i.e., $512 \times 1024$). Row 8, and 11 are our MobileNetV3 segmentation candidates.

| Backbone | OS | mIOU | Madds (f) | Madds (h) | CPU (f) | CPU (h) |
|---|---|---|---|---|---|---|
| V3 | 16 | 72.6 | 9.74B | 2.48B | 2.87s | 730ms |
| V3 | 32 | 72.0 | 7.74B | 1.98B | 2.39s | 607ms |
| V3-Small | 16 | 69.4 | 2.90B | 0.74B | 1.38s | 349ms |
| V3-Small | 32 | 68.3 | 2.06B | 0.53B | 1.16s | 290ms |
| ESPNetv2 [30] [†] | - | 62.1 | 13.46B | 3.36B | - | - |
| CCC2 [32] | - | 62.0 | - | 3.15B | - | - |
| ESPNetv1 [29] [†] | - | 60.3 | 17.72B | 4.43B | - | - |
| ESPNetv2 small [30] [†] | - | 54.7 | 2.26B | 0.56B | - | - |

Table 8. Semantic segmentation results on Cityscapes test set. **OS:** **O**utput **S**tride, the ratio of input image spatial resolution to backbone output resolution. When OS = 16, atrous convolution is applied in the last block of backbone. When OS = 32, no atrous convolution is used. **Madds (f):** Multiply-Adds measured w.r.t. a **full-resolution** input (i.e., $1024 \times 2048$). **Madds (h):** Multiply-Adds measured w.r.t. a **half-resolution** input (i.e., $512 \times 1024$). **CPU (f):** CPU time measured on a single large core of Pixel 3 (floating point) w.r.t. a **full-resolution** input (i.e., $1024 \times 2048$). **CPU (h):** CPU time measured w.r.t. a **half-resolution** input (i.e., $512 \times 1024$). [†]: The Madds are estimated from [30] which only provides Madds for input size $224 \times 224$ in their Fig. 7.

bile classification, detection and segmentation. We have described our efforts to harness multiple types of network architecture search as well as advances in network design to deliver the next generation of mobile models. We have also shown how to adapt non-linearities like swish and apply squeeze and excite in a quantization friendly and efficient manner introducing them into the mobile model domain as effective tools. We also introduced a new form of lightweight segmentation decoders called LR-ASPP. While it remains an open question of how best to blend automatic search techniques with human intuition, we are pleased to present these first positive results and will continue to refine methods as future work.

# 8. Acknowledgements

# References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 1

[2] R. Avenash and P. Vishawanth. Semantic segmentation of satellite images using a modified cnn with hard-swish activation function. In *VISIGRAPP*, 2019. 2, 4

[3] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016. 2

[4] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 535–541, New York, NY, USA, 2006. ACM. 2

[5] H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. *CoRR*, abs/1812.00332, 2018. 2, 3, 6

[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015. 8

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2017. 7

[8] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017. 7, 8

[9] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 7

[10] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 8

[11] M. Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015. 2, 4

[12] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha. Chamnet: Towards efficient network design through platform-aware model adaptation. *CoRR*, abs/1812.08934, 2018. 2

[13] M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserma. The pascal visual object classes challenge a retrospective. *IJCV*, 2014. 8

[14] Y. He and S. Han. AMC: automated deep compression and acceleration with reinforcement learning. In *ECCV*, 2018. 2

[15] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. 2

[16] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets: Time-Frequency Methods and Phase Space*, pages 289–297. Springer Berlin Heidelberg, 1989. 8

[17] 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. *CoRR*, abs/1704.04861, 2017. 2

[18] J. Hu, L. Shen, and G. Sun. Squeeze-and-Excitation Networks. *ArXiv e-prints*, Sept. 2017. 2, 3, 7

[19] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. *CoRR*, abs/1711.09224, 2017. 2

[20] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. 2

[21] 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. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

[22] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, R. C. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760, 2017. 6

[23] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018. 2

[24] 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. In *ECCV*, 2014. 7

[25] C. Liu, B. Zoph, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. *CoRR*, abs/1712.00559, 2017. 2

[26] H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. 2

[27] W. Liu, A. Rabinovich, and A. C. Berg. Parsenet: Looking wider to see better. *CoRR*, abs/1506.04579, 2015. 7

[28] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 8

[29] S. Mehta, M. Rastegari, A. Caspi, L. G. Shapiro, and H. Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part X*, pages 561–580, 2018. 8

[30] S. Mehta, M. Rastegari, L. G. Shapiro, and H. Hajishirzi. Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. *CoRR*, abs/1811.11431, 2018. 8

[31] G. Papandreou, I. Kokkinos, and P.-A. Savalle. Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection. In *CVPR*, 2015. 8

[32] H. Park, Y. Yoo, G. Seo, D. Han, S. Yun, and N. Kwak. Concentrated-comprehensive convolutions for lightweight semantic segmentation. *CoRR*, abs/1812.04920, 2018. 8

[33] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. 2

[34] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. 2, 4

[35] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnornet: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016. 2

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, Dec. 2015. 5, 8

[37] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. 2, 3, 6, 7, 8

[38] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv:1312.6229*, 2013. 8

[39] D. Soudry, I. Hubara, and R. Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *NIPS*, pages 963–971, 2014. 2

[40] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. 6

[41] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. 2, 3, 5, 6

[42] S. the Society for Imaging Science, Technology, S. of Photooptical Instrumentation Engineers, and T. A. of the Graphic Arts. *Curves and Surfaces in Computer Vision and Graphics*. Number v. 1610 in Proceedings of SPIE–the International Society for Optical Engineering. SPIE, 1992. 4

[43] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *CoRR*, abs/1812.03443, 2018. 2

[44] B. Wu, A. Wan, X. Yue, P. H. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. *CoRR*, abs/1711.08141, 2017. 2

[45] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. *CoRR*, abs/1512.06473, 2015. 2

[46] T. Yang, A. G. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, 2018. 2, 3

[47] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 2

[48] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *CVPR*, 2017. 7

[49] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *CoRR*, abs/1702.03044, 2017. 2

[50] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. Dorefanet: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016. 2

[51] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. 2, 3

[52] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. 2

# A. Performance table for different resolutions and multipliers

We give detailed table containing multiply-adds, accuracy, parameter count and latency in Table 9.

| Network | Top-1 (%) | Madds (M) | Params (M) | P-1 (ms) |
|---|---|---|---|---|
| large 224/1.25 | 76.6 | 356 | 7.5 | 104.0 |
| large 224/1.0 | 75.2 | 217 | 5.4 | 66.0 |
| large 224/0.75 | 73.3 | 155 | 4.0 | 50.0 |
| large 224/0.5 | 68.8 | 69 | 2.6 | 28.7 |
| large 224/0.35 | 64.2 | 40 | 2.2 | 20.4 |
| large 256/1.0 | 76.0 | 282 | 5.4 | 87.0 |
| large 192/1.0 | 73.7 | 160 | 5.4 | 49.0 |
| large 160/1.0 | 71.7 | 112 | 5.4 | 36.0 |
| large 128/1.0 | 68.4 | 73 | 5.4 | 24.2 |
| large 96/1.0 | 63.3 | 43 | 5.4 | 15.9 |
| small 224/1.25 | 70.4 | 91 | 3.6 | 32.0 |
| small 224/1.0 | 67.5 | 66 | 2.9 | 21.3 |
| small 224/0.75 | 65.4 | 44 | 2.4 | 18.0 |
| small 224/0.5 | 58.0 | 23 | 1.9 | 10.7 |
| small 224/0.35 | 49.8 | 13 | 1.7 | 8.2 |
| small 256/1.0 | 68.5 | 74 | 2.9 | 27.8 |
| small 192/1.0 | 65.4 | 42 | 2.9 | 16.3 |
| small 160/1.0 | 62.8 | 30 | 2.9 | 12.0 |
| small 128/1.0 | 57.3 | 20 | 2.9 | 8.3 |
| small 96/1.0 | 51.7 | 12 | 2.9 | 5.8 |

Table 9. Floating point performance for Large and Small V3 models. P-1 corresponds to large single core performance on Pixel 1.