2019 IEEE/CVF International Conference on Computer Vision (ICCV)

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 AI, [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 20% compared to MobileNetV2. MobileNetV3-Small is 6.6% more accurate compared to a MobileNetV2 model with comparable latency. MobileNetV3-Large detection is over 25% faster at roughly the same accuracy as MobileNetV2 on COCO detection. MobileNetV3-Large LR-ASPP is 34% faster than MobileNetV2 R-ASPP at similar accuracy for Cityscapes segmentation.

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.

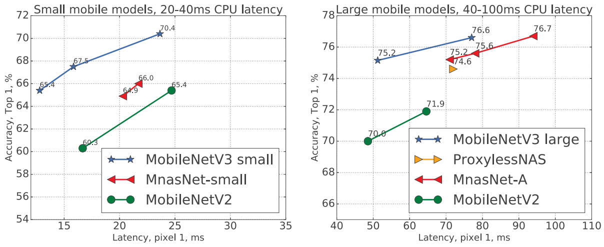

Figure 1. The trade-off between Pixel 1 latency and top-1 ImageNet accuracy. All models use the input resolution 224. V3 large and V3 small use multipliers 0.75, 1 and 1.25 to show optimal frontier. All latencies were measured on a single large core of the same device using TFLite[1]. MobileNetV3-Small and Large are our proposed next-generation mobile models.

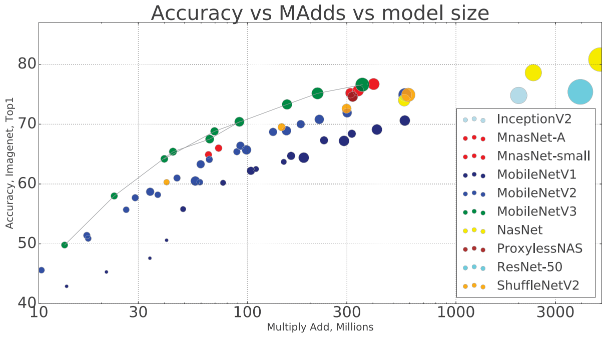

Figure 2. The trade-off between MAdds and top-1 accuracy. This allows to compare models that were targeted different hardware or software frameworks. All MobileNetV3 are for input resolution 224 and use multipliers 0.35, 0.5, 0.75, 1 and 1.25. See section 6 for other resolutions. Best viewed in color.

2380-7504/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCV.2019.00140

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 handcrafted structures and algorithmic neural architecture search have played important roles in advancing this field.

SqueezeNet[22] 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[19] employs depthwise separable convolution to substantially improve computation efficiency. MobileNetV2[39] expands on this by introducing a resource-efficient block with inverted residuals and linear bottlenecks. ShuffleNet[49] utilizes group convolution and channel shuffle operations to further reduce the MAdds. CondenseNet[21] learns group convolutions at the training stage to keep useful dense connections between layers for feature re-use. ShiftNet[46] 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 [53, 54, 3, 27, 35]. 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, [43] 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 [28, 5, 45] with gradient-based optimization. Focusing on adapting existing networks to constrained mobile platforms, [48, 15, 12] proposed more efficient automated network simplification algorithms.

Quantization [23, 25, 47, 41, 51, 52, 37] is another important complementary effort to improve the network efficiency through reduced precision arithmetic. Finally, knowledge distillation [4, 17] 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 [19] 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 [39] 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 nonlinear per-channel transformations.

MnasNet [43] 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 [20]. 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 [36, 13, 16]. 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.

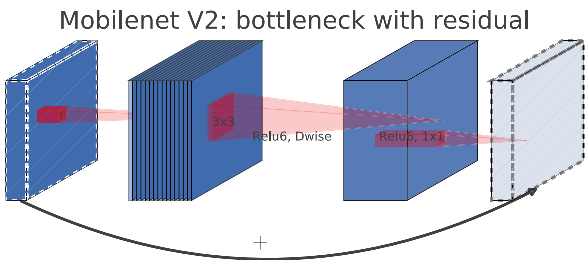

Figure 3. MobileNetV2 [39] layer (Inverted Residual and Linear Bottleneck). Each block consists of narrow input and output (bottleneck), which don't have nonlinearity, followed by expansion to a much higher-dimensional space and projection to the output. The residual connects bottleneck (rather than expansion).

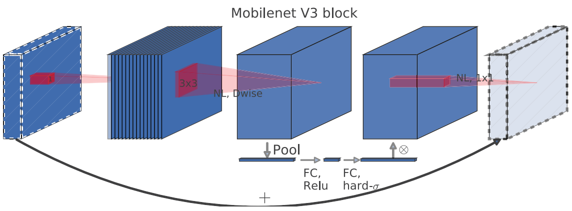

Figure 4. MobileNetV2 + Squeeze-and-Excite [20]. In contrast with [20] we apply the squeeze and excite in the residual layer. We use different nonlinearity depending on the layer, see section 5.2 for details.

4. Network Search

Network search has shown itself to be a very powerful tool for discovering and optimizing network architectures [53, 43, 5, 48]. 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 [43], 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 [43] for Large mobile models with target latency around 80ms. Therefore, we simply reuse the same MnasNet-A1 [43] as our initial Large mobile model, and then apply NetAdapt [48] 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 accuracy 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 [43]) 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 [48]. 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 δ 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. Finetune 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 [48] 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 Δlatency 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 [48] 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 proposals, it does not however, 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 7 milliseconds which is 11% 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 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 2 milliseconds and 10 million MAdds.

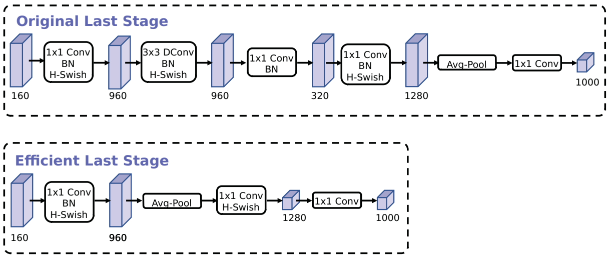

Figure 5. Comparison of original last stage and efficient last stage. This more efficient last stage is able to drop three expensive layers at the end of the network at no loss of accuracy.

5.2. Nonlinearities

In [36, 13, 16] a nonlinearity called *swish* was introduced that when used as a drop-in replacement for ReLU, that significantly improves the accuracy of neural networks. The nonlinearity is defined as

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

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: $\frac{\text{ReLU6}(x+3)}{6}$ similar to [11, 44]. The minor difference is we use ReLU6 rather than a custom clipping constant. Similarly, the hard version of swish becomes

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

A similar version of hard-swish was also recently proposed in [2]. The comparison of the soft and hard version of sigmoid and swish nonlinearities 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 ReLU6 are available on virtually all software and hardware frameworks. Second, in quantized mode, it eliminates potential numerical precision loss caused by different implementations of the approximate sigmoid. Finally, in practice, h-swish can be implemented as

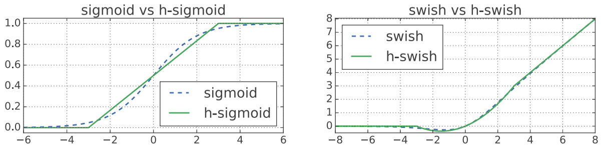

Figure 6. Sigmoid and swish nonlinearities and ther "hard" counterparts.

a piece-wise function to reduce the number of memory accesses driving the latency cost down substantially.

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 optimizations, h-swish still introduces some latency cost. However as we demonstrate in section 6 the net effect on accuracy and latency is positive with no optimizations and substantial when using an optimized implementation based on a piece-wise function.

5.3. Large squeeze-and-excite

In [43], 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.

5.4. 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.

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[38] for all our classification experiments and compare accuracy versus various measures of resource usage such as latency and multiply adds (MAdds).

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	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Table 1. Specification for MobileNetV3-Large. SE denotes whether there is a Squeeze-And-Excite in that block. NL denotes the type of nonlinearity 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
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	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
$14^2 \times 48$	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	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

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

6.1.1 Training setup

We train our models using synchronous training setup on 4x4 TPU Pod [24] 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), and learning rate decay rate of 0.01 every 3 epochs. We use dropout of 0.8, and l2 weight decay 1e-5 and the same image preprocessing as Inception [42]. Finally we use exponential moving average with decay 0.9999. All our convo-

Network	Top-1	MAdds	Params	P-1	P-2	P-3
V3-Large 1.0	**75.2**	**219**	5.4M	**51**	**61**	**44**
V3-Large 0.75	73.3	155	4.0M	39	46	40
MnasNet-A1	75.2	315	3.9M	71	86	61
Proxyless[5]	74.6	320	4.0M	72	84	60
V2 1.0	72.0	300	3.4M	64	76	56
V3-Small 1.0	67.4	66	2.9M	15.8	19.4	14.4
V3-Small 0.75	65.4	44	2.4M	12.8	15.6	11.7
Mnas-small [43]	64.9	65.1	1.9M	20.3	24.2	17.2
V2 0.35	60.8	59.2	1.6M	16.6	19.6	13.9

Table 3. Floating point performance on the Pixel family of phones (P-n denotes a Pixel-n phone). All latencies are in ms and are measured using a single large core with a batch size of one. Top-1 accuracy is on ImageNet.

lutional 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. We contributed an atomic h-swish operator to tensorflow lite, and it is now default in the latest version. We show the impact of optimized h-swish on figure 9.

6.2. Results

As can be seen on figure 1 our models outperform the current state of the art such as MnasNet [43], ProxylessNas [5] and MobileNetV2 [39]. 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.

6.2.1 Ablation study

Impact of non-linearities In table 5 we study the choice of where to insert h-swish nonlinearities as well as the improvements of using an optimized implementation over a naive implementation. It can be seen that using an optimized implementation of h-swish saves 6ms (more than

Network	Top-1	P-1	P-2	P-3
V3-Large 1.0	**73.8**	44	42.5	31.7
V2 1.0	70.9	52	48.3	37.0
V3-Small	**64.9**	15.5	14.9	10.7
V2 0.35	57.2	16.7	15.6	11.9

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

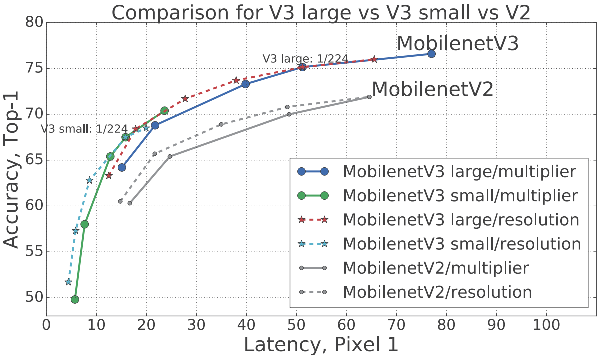

Figure 7. Performance of MobileNetV3 as a function of different multipliers and resolutions. In our experiments we have used multipliers 0.35, 0.5, 0.75, 1.0 and 1.25, with a fixed resolution of 224, and resolutions 96, 128, 160, 192, 224 and 256 with a fixed depth multiplier of 1.0. Best viewed in color. Top-1 accuracy is on ImageNet and latency is in ms.

	Top-1	P-1	P-1 (no-opt)
V3-Large 1.0	75.2	51.4	57.5
ReLU	74.5 (-.7%)	50.5 (-1%)	50.5
h-swish @16	75.4 (+.2%)	53.5 (+4%)	68.9
h-swish @112	75.0 (-.3%)	51 (-0.5%)	54.4

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. The third column shows the runtime without optimized h-swish. Top-1 accuracy is on ImageNet and latency is in ms.

10% of the runtime). Optimized h-swish only adds an additional 1ms compared to traditional ReLU.

Figure 8 shows the efficient frontier based on nonlinearity choices and network width. MobileNetV3 uses h-swish in the middle of the network and clearly dominates ReLU. It is interesting to note that adding h-swish to the entire network is slightly better than the interpolated frontier of widening the network.

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

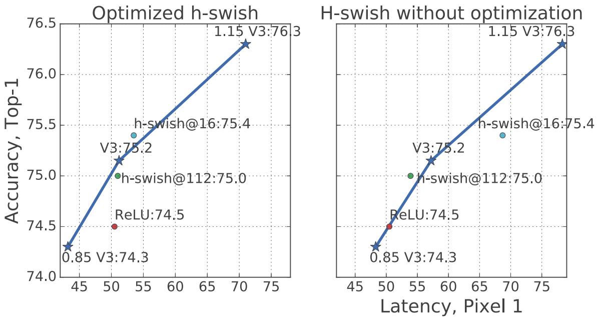

Figure 8. Impact of h-swish vs ReLU on latency for optimized and non-optimized h-swish. The curve shows a frontier of using depth multiplier. Note that placing h-swish at all layers with 80 channels or more (V3) provides the best trade-offs for both optimized h-swish and non-optimized h-swish. Top-1 accuracy is on ImageNet and latency is in ms.

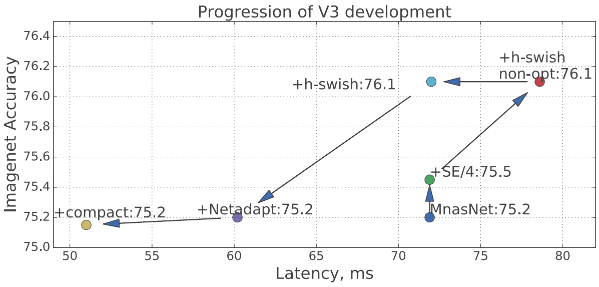

Figure 9. Impact of individual components in the development of MobileNetV3. Progress is measured by moving up and to the left.

6.3. Detection

We use MobileNetV3 as a drop-in replacement for the backbone feature extractor in SSDLite [39] and compare with other backbone networks on COCO dataset [26].

Following MobileNetV2 [39], 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 90 classes.

The results on COCO test set are given in Tab. 6. With the channel reduction, MobileNetV3-Large is 27% 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 while being 35% faster. 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.

Backbone	mAP	Latency (ms)	Params (M)	MAdds (B)
V1	22.2	228	5.1	1.3
V2	22.1	162	4.3	0.80
MnasNet	23.0	174	4.88	0.84
V3	22.0	137	4.97	0.62
V3†	22.0	119	3.22	0.51
V2 0.35	13.7	66	0.93	0.16
V2 0.5	16.6	79	1.54	0.27
MnasNet 0.35	15.6	68	1.02	0.18
MnasNet 0.5	18.5	85	1.68	0.29
V3-Small	16.0	52	2.49	0.21
V3-Small†	16.1	43	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.

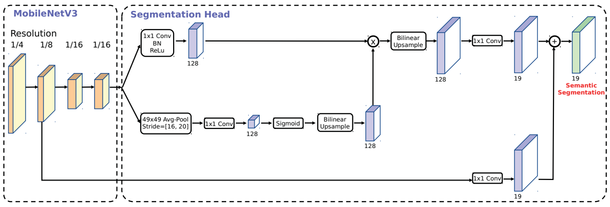

Figure 10. Building on MobileNetV3, the proposed segmentation head, Lite R-ASPP, delivers fast semantic segmentation results while mixing features from multiple resolutions.

6.4. Semantic Segmentation

In this subsection, we employ MobileNetV2 [39] 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 [39]. R-ASPP is a *reduced* design of the Atrous Spatial Pyramid Pooling module [7, 8, 9], which adopts only two branches consisting of a 1×1 convolution and a global-average pooling operation [29, 50]. 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 [20], in which we employ a large pooling kernel with a large stride (to save some computation) and only one 1×1 convolution in the module. We apply atrous convolution [18, 40, 33, 6] to the last block of MobileNetV3 to extract denser features, and further add a skip connection [30] from low-level features to capture more detailed information.

We conduct the experiments on the Cityscapes dataset [10] with metric mIOU [14], and only exploit the 'fine' annotations. We employ the same training protocol as [8, 39]. All our models are trained from scratch without pretraining on ImageNet [38], 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 [38] 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 [39] 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 outperforms ESPNetv2 [32], CCC2 [34], and ESPNetv1 [32] by 6.4%, 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 inputs), which is 1.36, 1.59, and 2.27 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 2.1%.

N	Backbone	RF2	SH	F	mIOU	Params	MAdds	CPU (f)	CPU (h)
1	V2	-	×	256	72.84	2.11M	21.29B	3.90s	1.02s
2	V2	✓	×	256	72.56	1.15M	13.68B	3.03s	793ms
3	V2	✓	✓	256	72.97	1.02M	12.83B	2.98s	786ms
4	V2	✓	✓	128	72.74	0.98M	12.57B	2.89s	766ms
5	V3	-	×	256	72.64	3.60M	18.43B	3.55s	906ms
6	V3	✓	×	256	71.91	1.76M	11.24B	2.60s	668ms
7	V3	✓	✓	256	72.37	1.63M	10.33B	2.55s	659ms
8	V3	✓	✓	128	72.36	1.51M	9.74B	2.47s	657ms
9	V2 0.5	✓	✓	128	68.57	0.28M	4.00B	1.59s	415ms
10	V2 0.35	✓	✓	128	66.83	0.16M	2.54B	1.27s	354ms
11	V3-Small	✓	✓	128	68.38	0.47M	2.90B	1.21s	327ms

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 × 2048). **CPU (h):** CPU time measured w.r.t. a **half-resolution** input (i.e., 512 × 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	2.47s	657ms
V3	32	72.0	7.74B	1.98B	2.06s	534ms
V3-Small	16	69.4	2.90B	0.74B	1.21s	327ms
V3-Small	32	68.3	2.06B	0.53B	1.03s	275ms
ESPNetv2 [32]	-	66.2	-	2.7B	-	-
CCC2 [34]	-	62.0	-	3.15B	-	-
ESPNetv1 [31]	-	60.3	-	4.5B	-	-

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 × 2048). **MAdds (h):** Multiply-Adds measured w.r.t. a **half-resolution** input (i.e., 512 × 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 × 2048). **CPU (h):** CPU time measured w.r.t. a **half-resolution** input (i.e., 512 × 1024). ESPNet [31, 32] and CCC2 [34] take half resolution inputs, while our models directly take full-resolution inputs.

7. Conclusions and future work

In this paper we introduced MobileNetV3 Large and Small models demonstrating new state of the art in mobile classification, detection and segmentation. We have described our efforts to harness multiple network architecture search algorithms as well as advances in network design to deliver the next generation of mobile models. We have also shown how to adapt nonlinearities 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.

Acknowledgements: We would like to thank Andrey Zhmoginov, Dmitry Kalenichenko, Menglong Zhu, Jon Shlens, Xiao Zhang, Benoit Jacob, Alex Stark, Achille Brighton and Sergey Ioffe for helpful feedback and discussion.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang 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] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016. 2

[4] Cristian Buciluǎ, Rich Caruana, and Alexandru 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] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *CoRR*, abs/1812.00332, 2018. 2, 3, 6

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015. 7

[7] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2017. 7

[8] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017. 7, 8

[9] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 7

[10] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 8

[11] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015. 2, 4

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

[13] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *CoRR*, abs/1702.03118, 2017. 2, 4

[14] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserma. The pascal visual object classes challenge a retrospective. *IJCV*, 2014. 8

[15] Yihui He and Song Han. AMC: automated deep compression and acceleration with reinforcement learning. In *ECCV*, 2018. 2

[16] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016. 2, 4

[17] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. 2

[18] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph 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. 7

[19] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 2

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

[21] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. *CoRR*, abs/1711.09224, 2017. 2

[22] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. 2

[23] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

[24] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, Richard C. Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni,

Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760, 2017. 5

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

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 7

[27] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *CoRR*, abs/1712.00559, 2017. 2

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

[29] Wei Liu, Andrew Rabinovich, and Alexander C. Berg. Parsenet: Looking wider to see better. *CoRR*, abs/1506.04579, 2015. 7

[30] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 7

[31] Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda G. Shapiro, and Hannaneh 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

[32] Sachin Mehta, Mohammad Rastegari, Linda G. Shapiro, and Hannaneh Hajishirzi. Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. *CoRR*, abs/1811.11431, 2018. 8

[33] George Papandreou, Iasonas Kokkinos, and Pierre-Andre Savalle. Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection. In *CVPR*, 2015. 7

[34] Hyojin Park, Youngjoon Yoo, Geonseok Seo, Dongyoon Han, Sangdoo Yun, and Nojun Kwak. Concentrated-comprehensive convolutions for lightweight semantic segmentation. *CoRR*, abs/1812.04920, 2018. 8

[35] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. 2

[36] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. 2, 4

[37] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016. 2

[38] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, Dec. 2015. 5, 8

[39] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. 2, 3, 6, 7, 8

[40] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv:1312.6229*, 2013. 7

[41] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *NIPS*, pages 963–971, 2014. 2

[42] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. 5

[43] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. 2, 3, 5, 6

[44] SPSE the Society for Imaging Science, Technology, Society of Photo-optical Instrumentation Engineers, and Technical Association 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

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

[46] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter H. Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. *CoRR*, abs/1711.08141, 2017. 2

[47] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. *CoRR*, abs/1512.06473, 2015. 2

[48] Tien-Ju Yang, Andrew G. Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, 2018. 2, 3

[49] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 2

[50] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017. 7

[51] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *CoRR*, abs/1702.03044, 2017. 2

[52] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016. 2

[53] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. 2, 3

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