Searching for MobileNetV3

CVPR2019

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

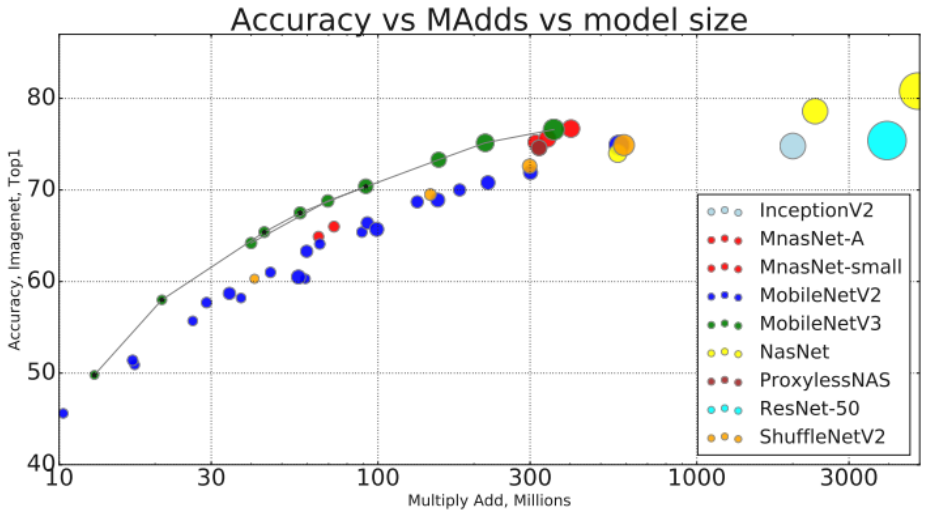
Wonboem Jang

Introduction

Multi-Label Image Recognition

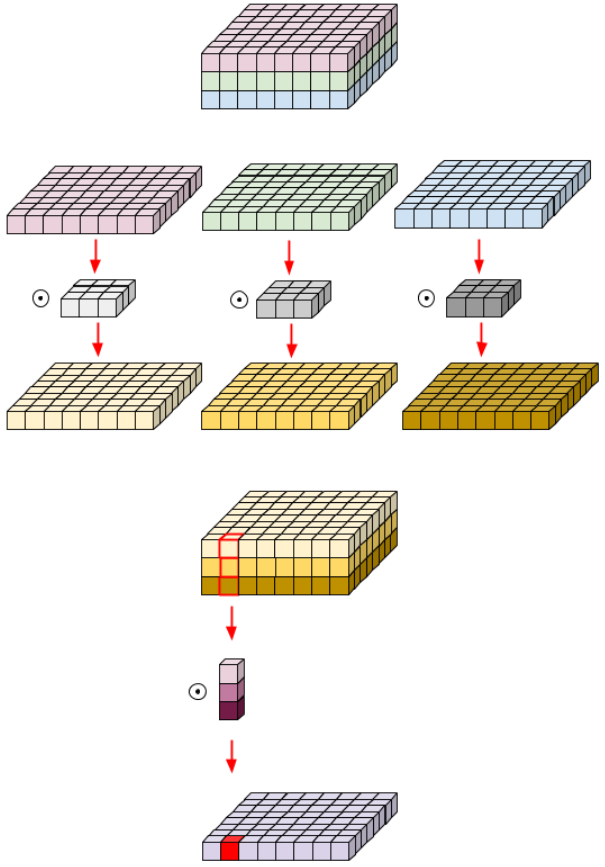
This paper describes the approach we took to develop MobileNetV3 Large and Small models

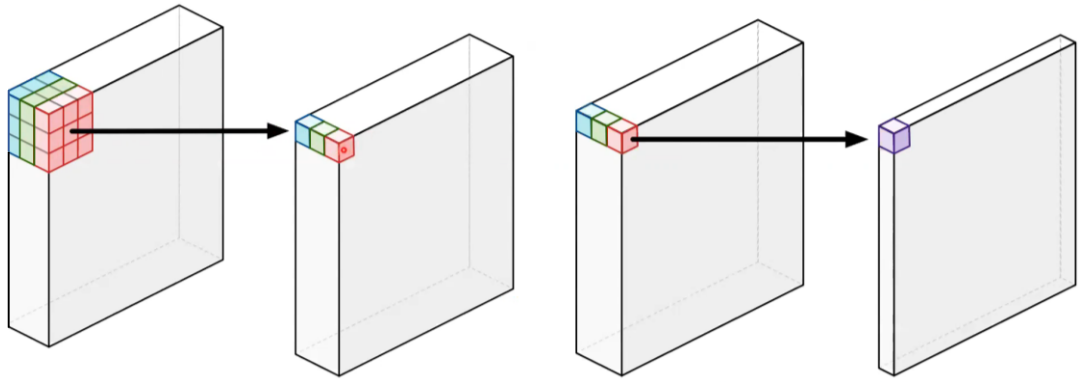
1. complementary search techniques
2. new efficient versions of nonlinearities practical for the mobile setting
3. new efficient network design
4. a new efficient segmentation decoder



Recap: MobileNet V1

Depth-wise separable Convolution





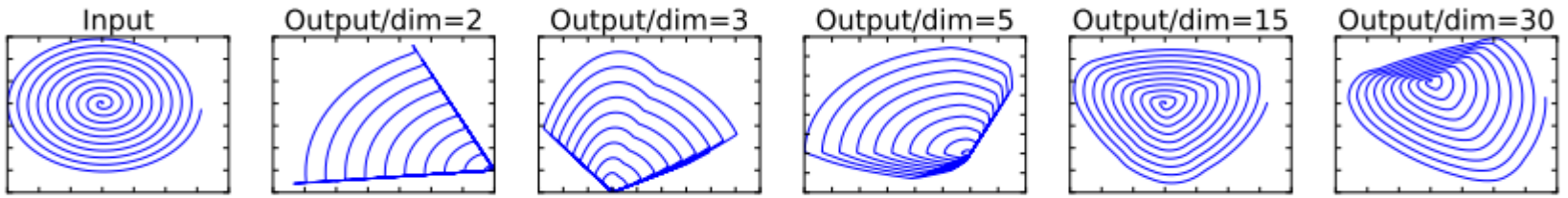
Depthwise Convolution Pointwise Convolution

- Standard Convolution cost:
  $h_i \times w_i \times d_j \times k \times k \times d_i$
- Deptwise seperable convolution cost
  $h_i \times w_i \times d_j \times (k^2 + d_i)$
- Reduction in computation
  $\frac{1}{d_j} + \frac{1}{k^2}$

Recap: MobileNet V2

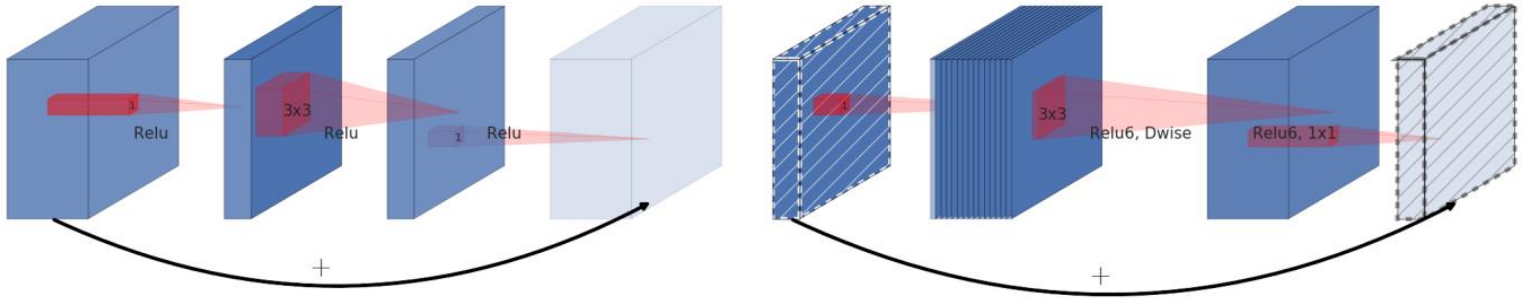
Linear Bottlenecks

- If the manifold of interest remains non-zero volume after ReLU transformation, **it corresponds to a linear transformation**.
- ReLU is capable of preserving complete information about the input manifold, **but only if the input manifold lies in a low-dimensional subspace of the input space**.
- we can capture this by **inserting linear bottleneck layers** into the convolutional blocks.



Recap: MobileNet V2

Inverted Residual

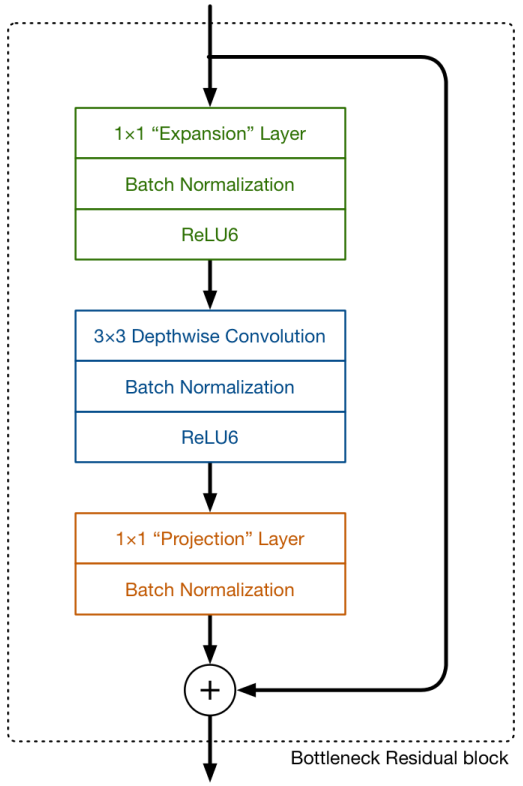


Residual

Inverted Residual

wide → narrow → wide

narrow → wide → narrow



Block of size $h \times w$, expansion factor t, kernel size k, input channel d', output channel d''

$$h \times w \times t \times d' \times d' + h \times w \times t \times d' \times k \times k + h \times w \times d'' \times t \times d''$$

$$= h \times w \times d' \times t \times (d' + k^2 + d'')$$

Recap: MnasNet

Inverted Residual

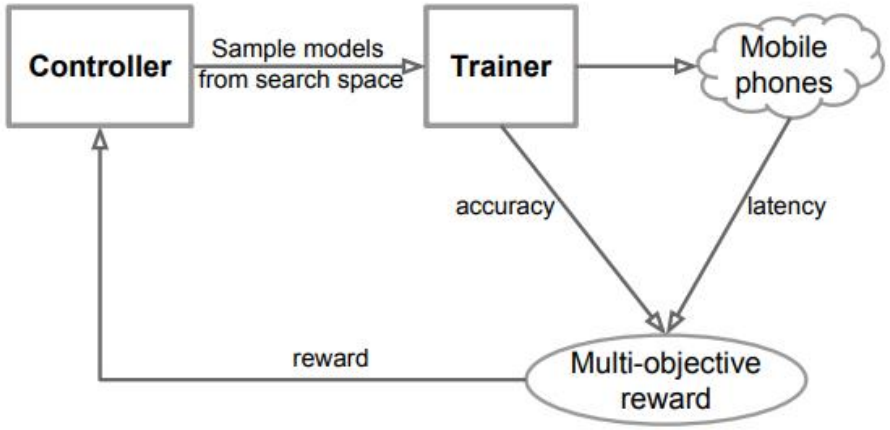
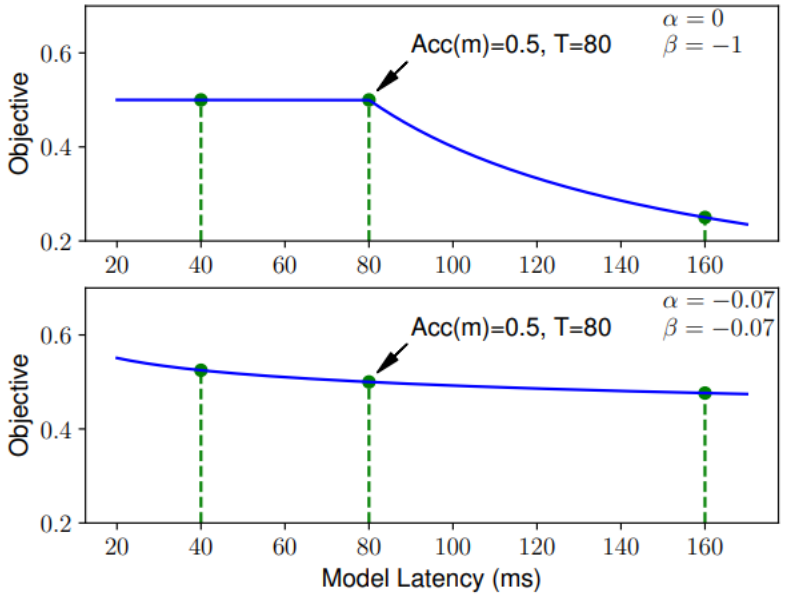


Figure 1: **An Overview of Platform-Aware Neural Architecture Search for Mobile.**



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

where w is the weight factor defined as:

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

Recap: MnasNet

Inverted Residual

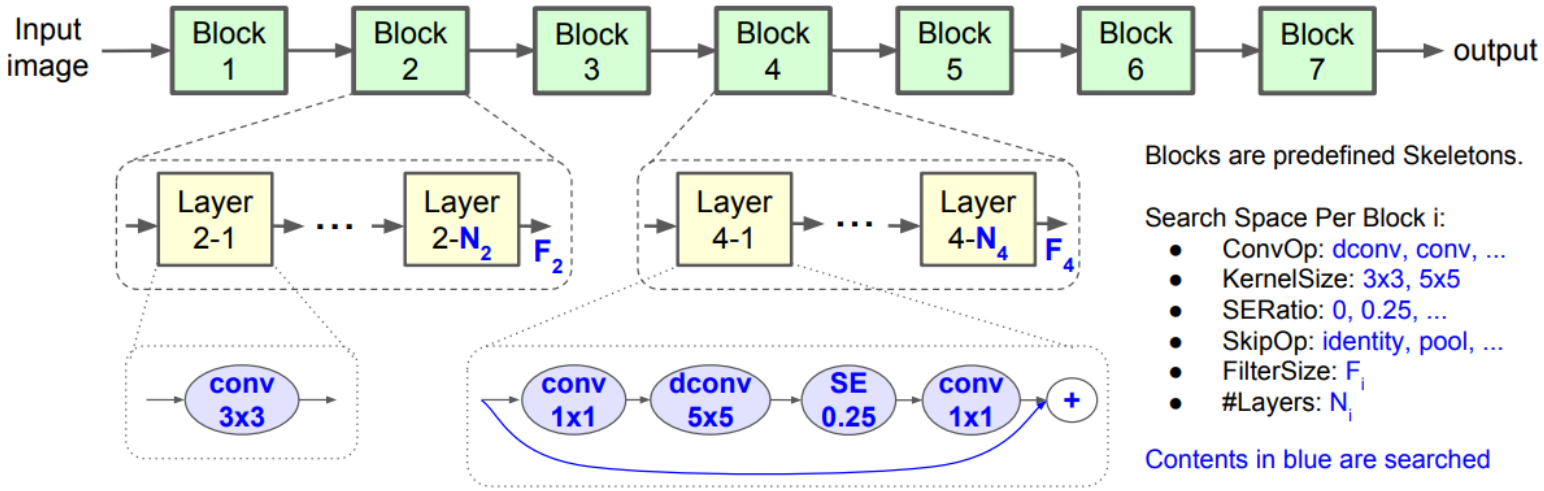
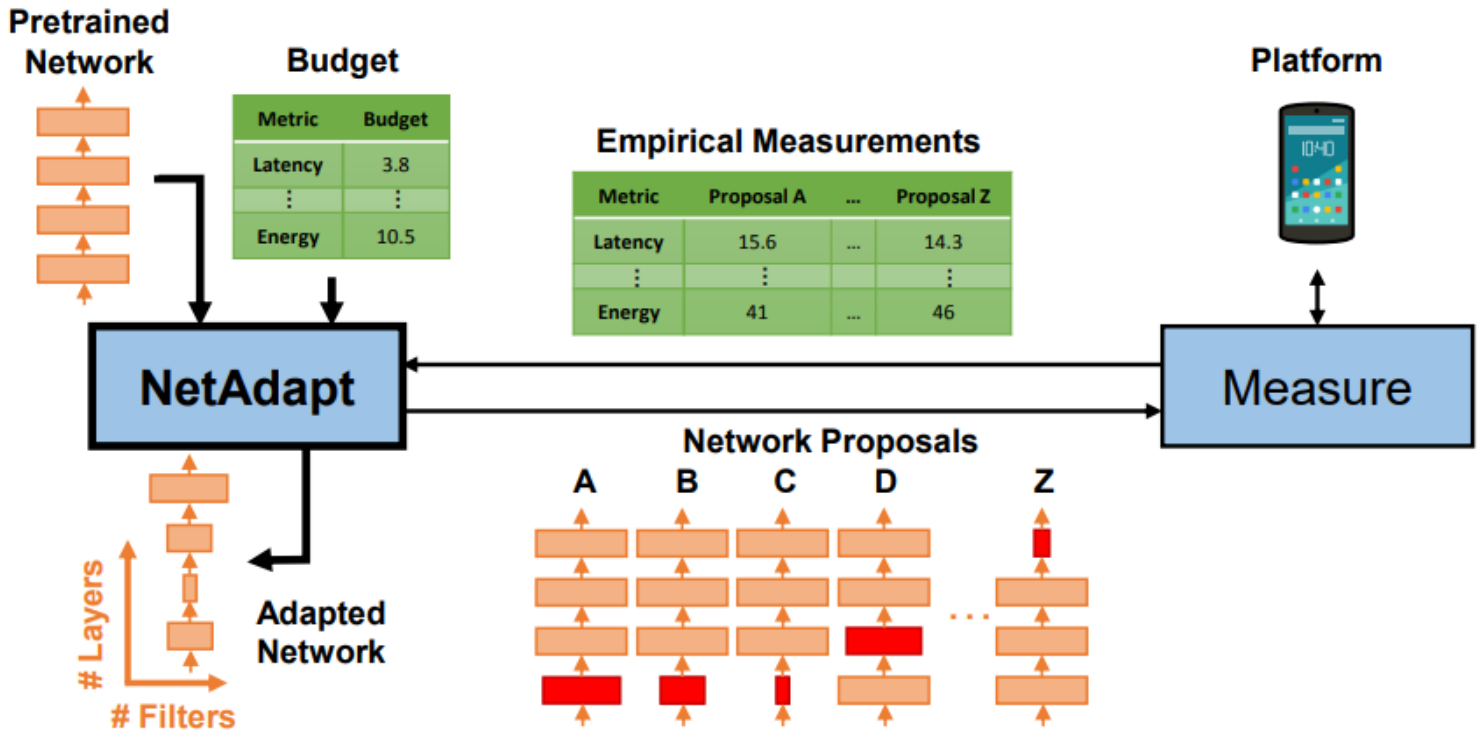


Figure 4: **Factorized Hierarchical Search Space.** Network layers are grouped into a number of predefined skeletons, called blocks, based on their input resolutions and filter sizes. Each block contains a variable number of repeated identical layers where only the first layer has stride 2 if input/output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

Recap: NetAdapt



$$\underset{Net}{\text{maximize}} \quad Acc(Net)$$

$$\text{subject to} \quad Res_j(Net) \leq Bud_j, \; j = 1, \ldots, m,$$

Recap: NetAdapt

Algorithm 1: NetAdapt

Input: Pretrained Network: Net_0 (with K CONV and FC layers), Resource Budget: Bud, Resource Reduction Schedule: ΔR_i

Output: Adapted Network Meeting the Resource Budget: $\hat{Net}$

```
1  i = 0;
2  Res_i = TakeEmpiricalMeasurement(Net_i);
3  while Res_i > Bud do
4      Con = Res_i - ΔR_i;
5      for k from 1 to K do
           /* TakeEmpiricalMeasurement is also called inside
              ChooseNumFilters for choosing the correct number of filters
              that satisfies the constraint (i.e., current budget). */
6          N_Filt_k, Res_Simp_k = ChooseNumFilters(Net_i, k, Con);
7          Net_Simp_k = ChooseWhichFilters(Net_i, k, N_Filt_k);
8          Net_Simp_k = ShortTermFineTune(Net_Simp_k);
9      Net_{i+1}, Res_{i+1} = PickHighestAccuracy(Net_Simp_:, Res_Simp_:);
10     i = i + 1;
11 N̂et = LongTermFineTune(Net_i);
12 return N̂et;
```

Efficient Mobile Building Block

Combination of layers

1. MobileNetV1: depthwise separable convolutions
2. MobileNetV2: linear bottleneck, inverted residual structure
3. MnasNet: lightweight attention modules based on squeeze and excitation into the bottleneck structure

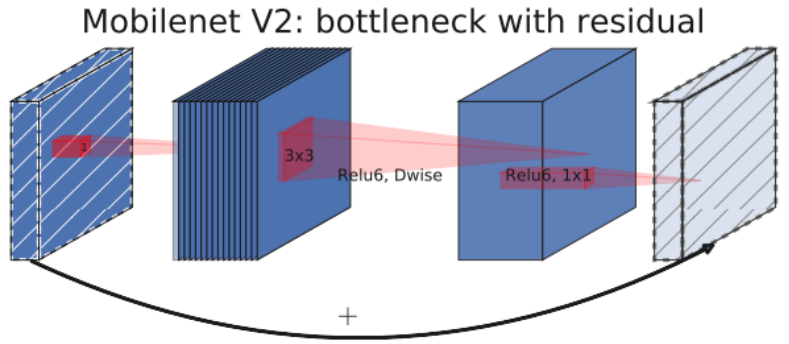


Figure 3. MobileNetV2 [39] layer (Inverted Residual and Linear Bottleneck). Each block consists of narrow input and output (bottleneck), which don't have nonlinearity, followed by expansion to a much higher-dimensional space and projection to the output. The residual connects bottleneck (rather than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

MobileNetV3

we use a combination of these layers as building blocks in order to build the most effective models.

upgraded with modified swish nonlinearities → h-swish

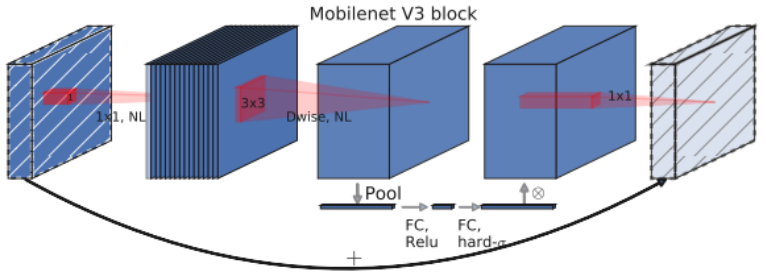


Figure 4. MobileNetV2 + Squeeze-and-Excite [20]. In contrast with [20] we apply the squeeze and excite in the residual layer. We use different nonlinearity depending on the layer, see section 5.2 for details.

Network Search

Platform-Aware NAS for Block-wise Search

We observe that accuracy changes much more dramatically with latency for small models

$$\underset{m}{maximize}\ ACC(m) \times \left[\frac{LAT(m)}{T}\right]^{\omega}$$

$\omega = -0.17$ vs the original $\omega = -0.07$

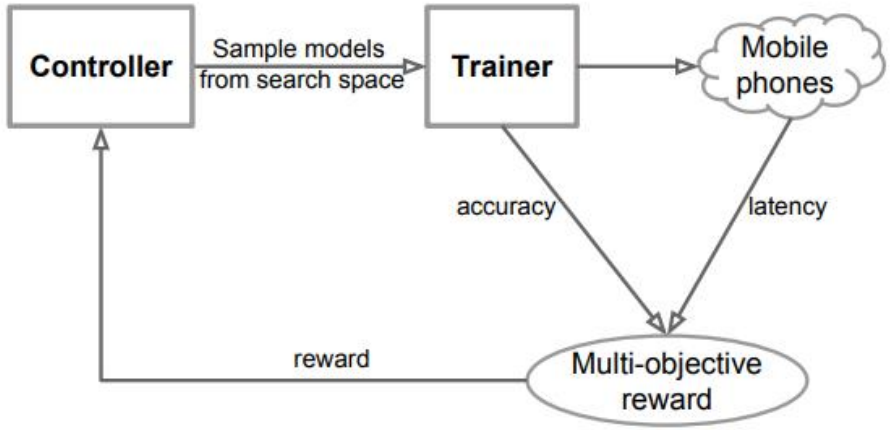


Figure 1: **An Overview of Platform-Aware Neural Architecture Search for Mobile.**

Network Search

NetAdapt for Layer-wise Search

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) (c) Selected best proposal according to some metric.

3. Iterate previous step until target latency is reached.

Network Search

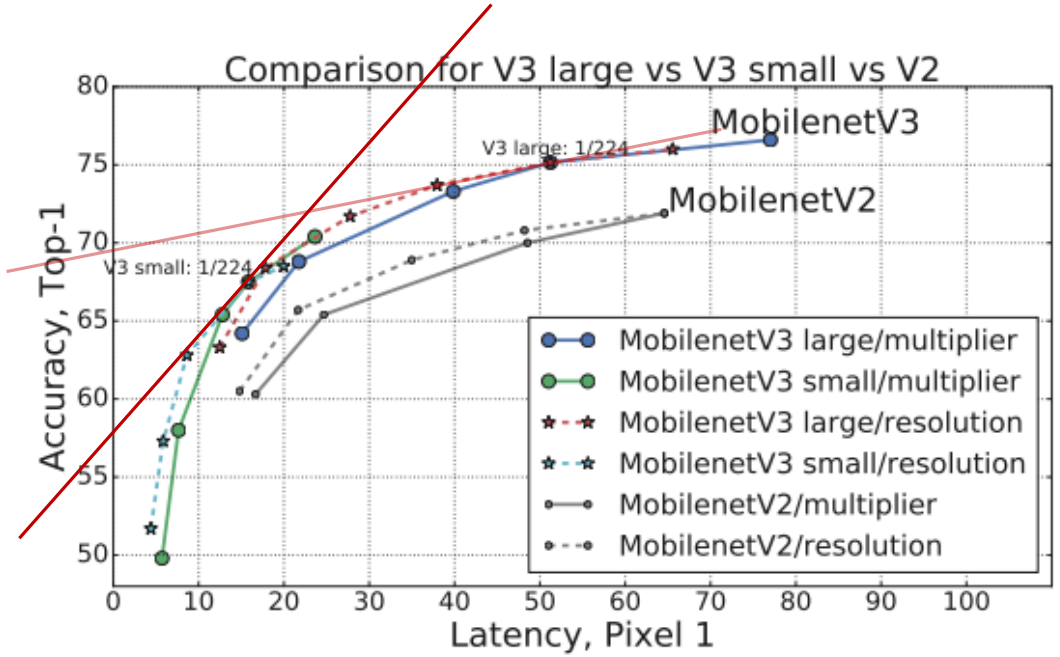
Proposal

1. Reduce the size of any expansion layer;
2. Reduce bottleneck in all blocks that share the same bottleneck size - to maintain residual connections

Evaluation

$$maximize \ \frac{\Delta Acc}{|\Delta latency|}$$

→ maximize tan



Network Improvements

Redesigning Expensive Layers

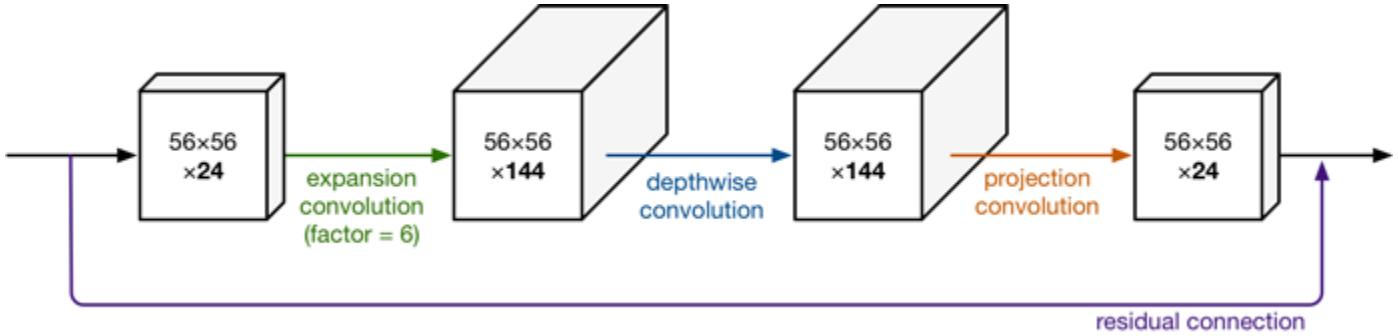
We redesign the computionally-expensive layers at the beginning and the end of the network. These modifications are outside of the scope of the current search space.

1. Current models based on MobileNetV2's inverted bottleneck structure and variants use 1x1 convolution. This layer is critically important in order to have rich features for prediction. However, this comes at a cost of extra latency.

→ Move this layer (1) past the final average pooling

2. The previous bottleneck projection layer is no longer needed to reduce computation.

→ Remove the projection and filtering layers in the previous bottleneck layer



Network Improvements

Redesigning Expensive Layers

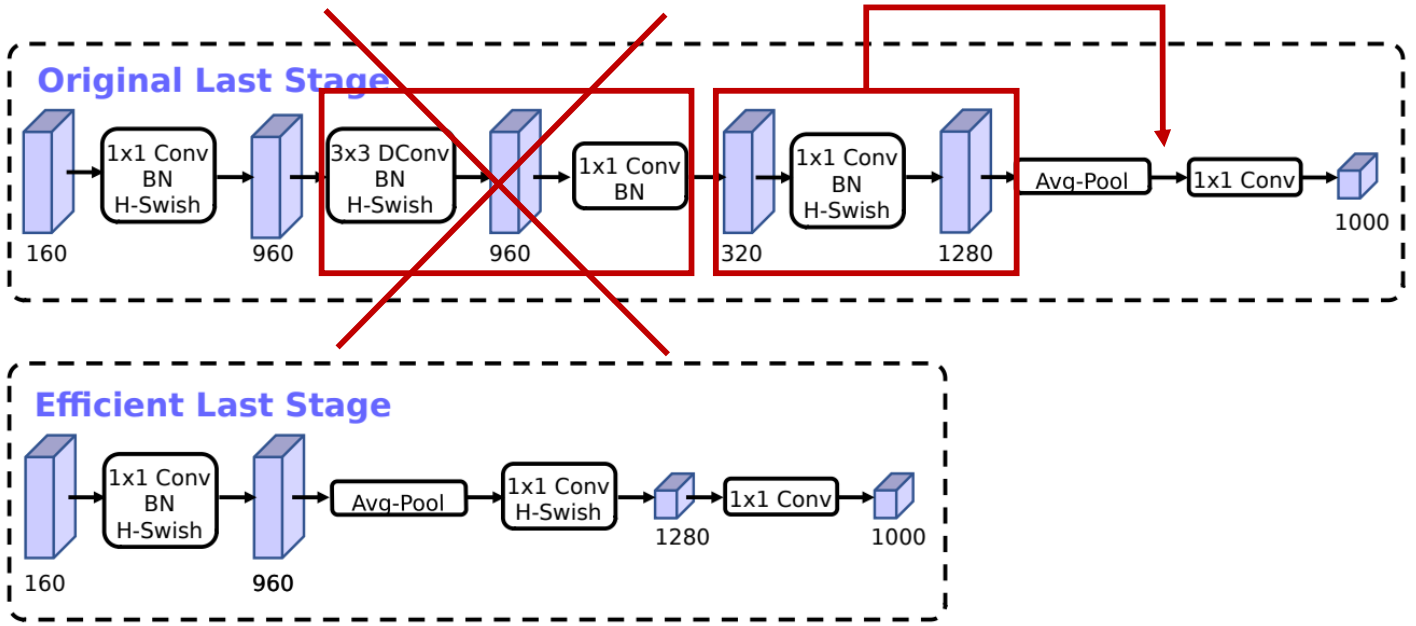


Figure 5. Comparison of original last stage and efficient last stage. This more efficient last stage is able to drop three expensive layers at the end of the network at no loss of accuracy.

Network Improvements

Redesigning Expensive Layers

We experimented with reducing the number of filters and using different nonlinearities to try and reduce redundancy.

We were able to reduce the number of filters to 16 while maintaining the same accuracy as 32 filters using either ReLU or swish.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1

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

Network Improvements

Nonlinearities

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

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.

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

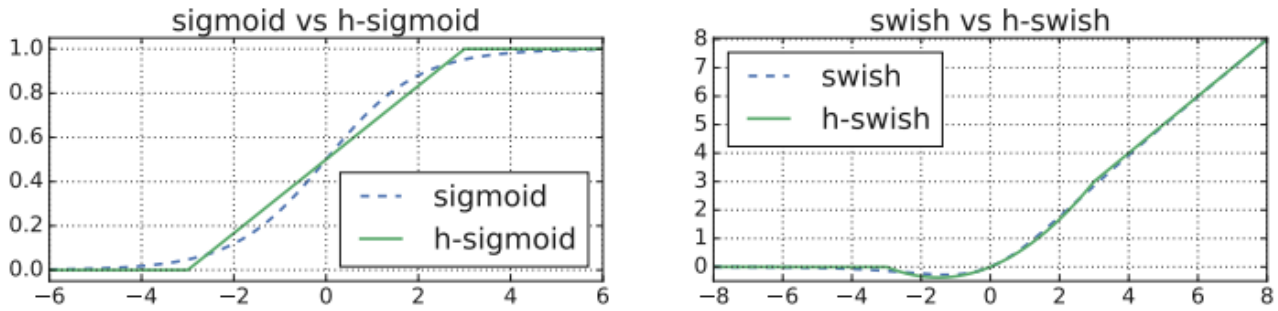


Figure 6. Sigmoid and swish nonlinearities and ther "hard" counterparts.

Network Improvements

Nonlinearities

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

Network Improvements

Nonlinearities

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.

Thus in our architectures we only use h-swish at the second half of the model.

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

Network Improvements

Large squeeze-and-excite

we replace them all to fixed to be 1/4 of the number of channels in expansion layer.

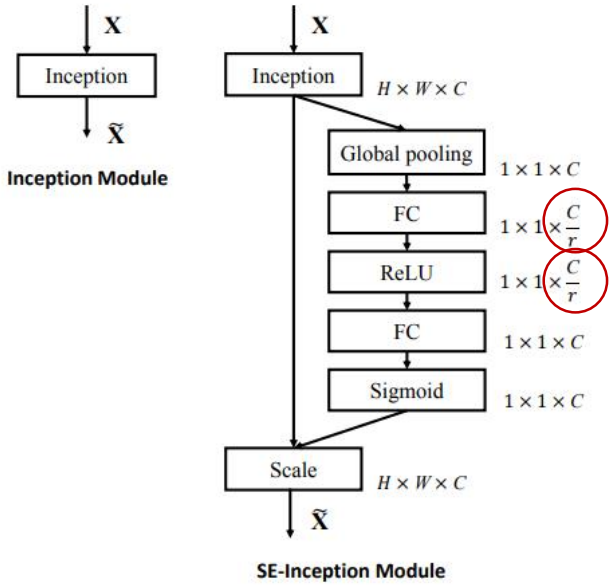


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

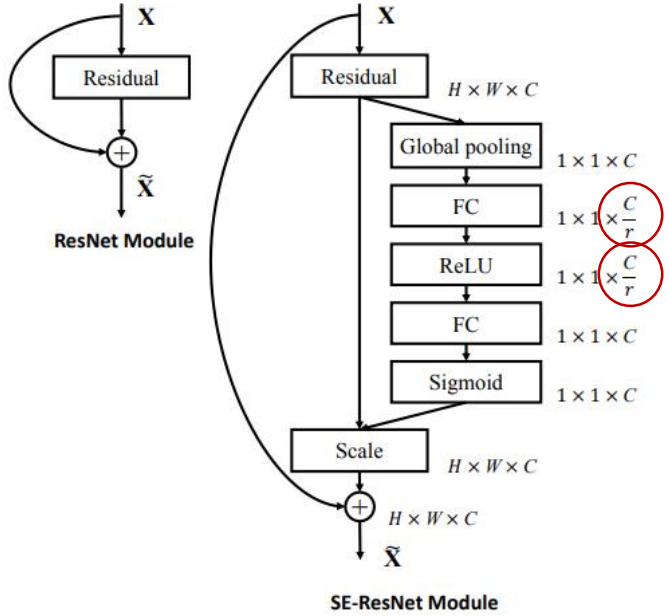


Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

Experiments

Architecture

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	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

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

Experiments

Results - Classification

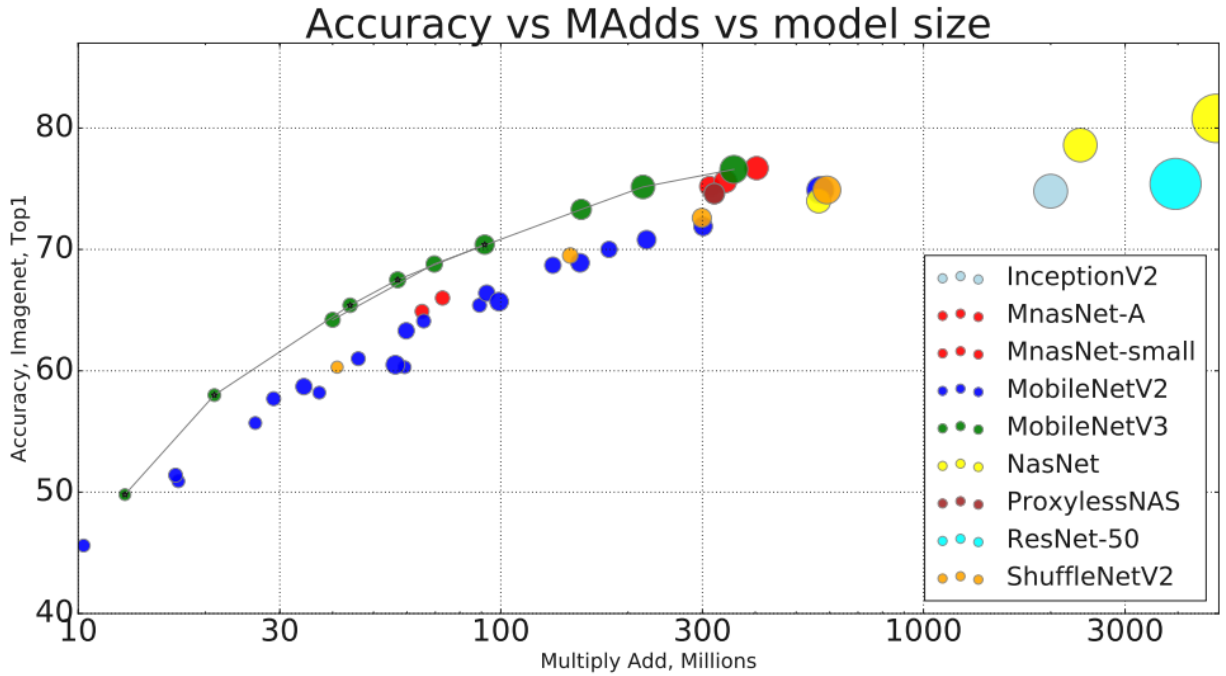


Figure 2. The trade-off between MAdds and top-1 accuracy. This allows to compare models that were targeted different hardware or software frameworks. All MobileNetV3 are for input resolution 224 and use multipliers 0.35, 0.5, 0.75, 1 and 1.25. See section 6 for other resolutions. Best viewed in color.

Experiments

Results - Classification

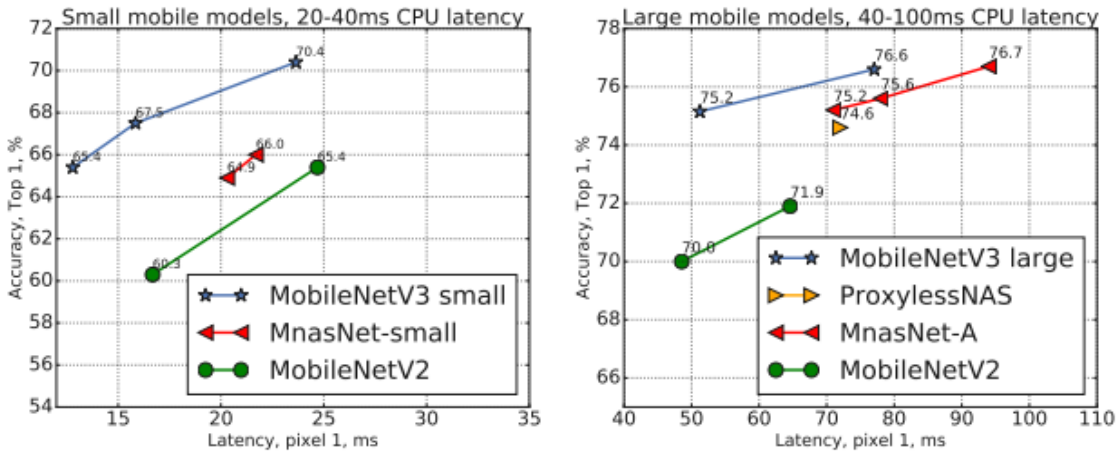


Figure 1. The trade-off between Pixel 1 latency and top-1 ImageNet accuracy. All models use the input resolution 224. V3 large and V3 small use multipliers 0.75, 1 and 1.25 to show optimal frontier. All latencies were measured on a single large core of the same device using TFLite[1]. MobileNetV3-Small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

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**	**56**	2.5M	**15.8**	**19.4**	**14.4**
V3-Small 0.75	65.4	44	2.0M	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.

Experiments

Results - Classification

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.

	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.

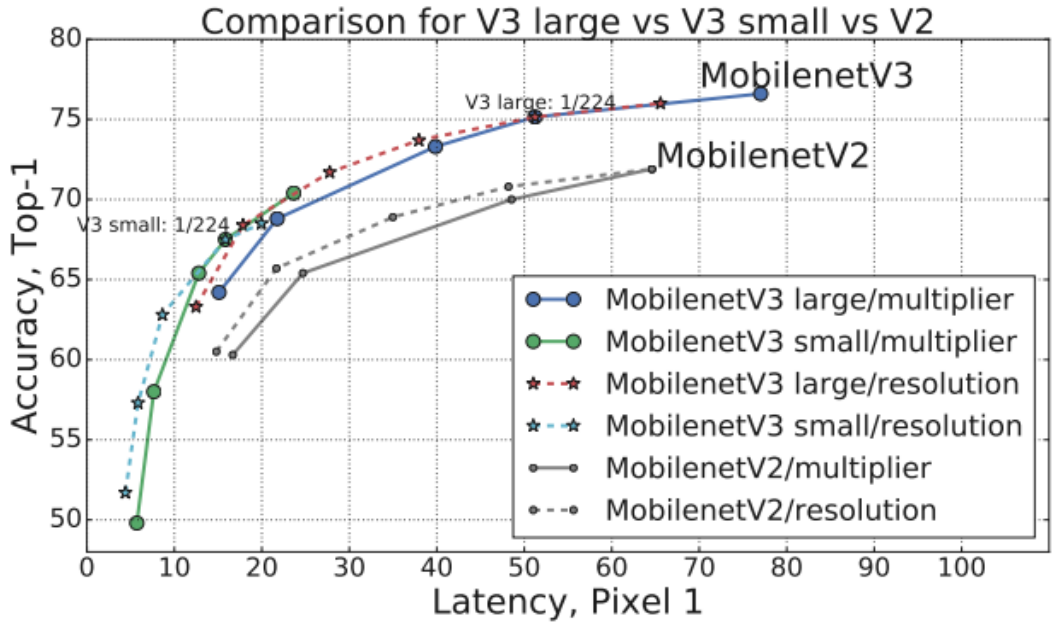


Figure 7. Performance of MobileNetV3 as a function of different multipliers and resolutions. In our experiments we have used multipliers 0.35, 0.5, 0.75, 1.0 and 1.25, with a fixed resolution of 224, and resolutions 96, 128, 160, 192, 224 and 256 with a fixed depth multiplier of 1.0. Best viewed in color.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

Experiments

Results - Classification

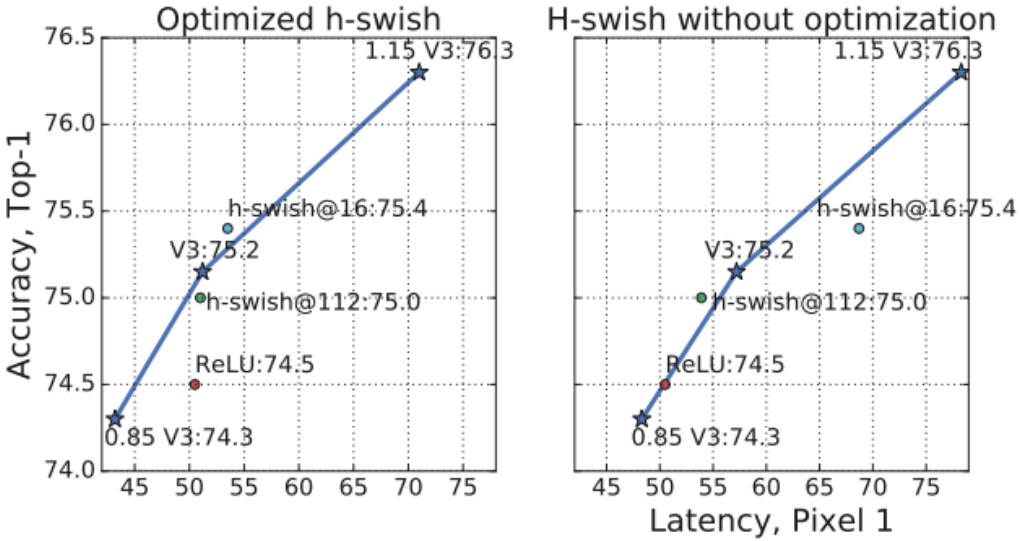


Figure 8. Impact of h-swish vs ReLU on latency for optimized and non-optimized h-swish. The curve shows a frontier of using depth multiplier. Note that placing h-swish at all layers with 80 channels or more (V3) provides the best trade-offs for both optimized h-swish and non-optimized h-swish. Top-1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

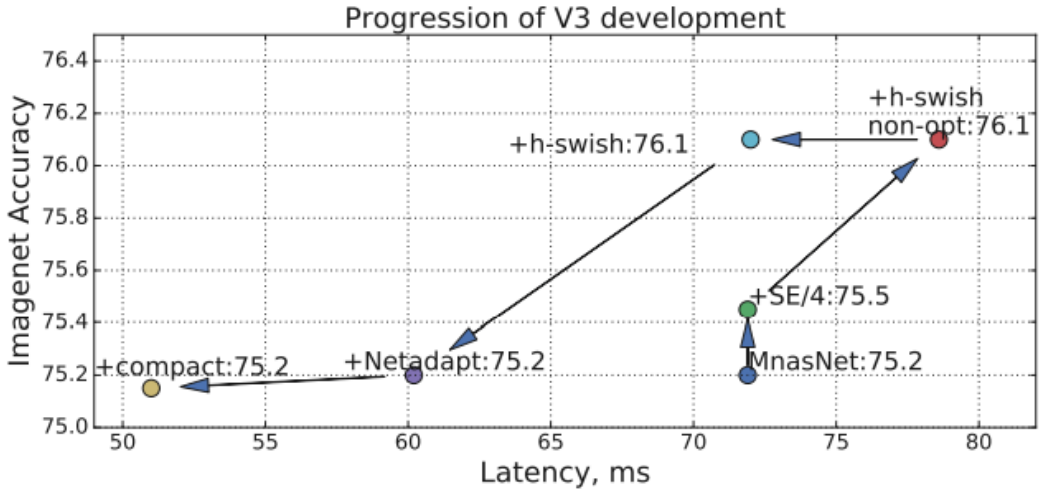


Figure 9. Impact of individual components in the development of MobileNetV3. Progress is measured by moving up and to the left.

Experiments

Results - Detection

SSDLite

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.

Experiments

Results - Semantic Segmentation

R-ASPP

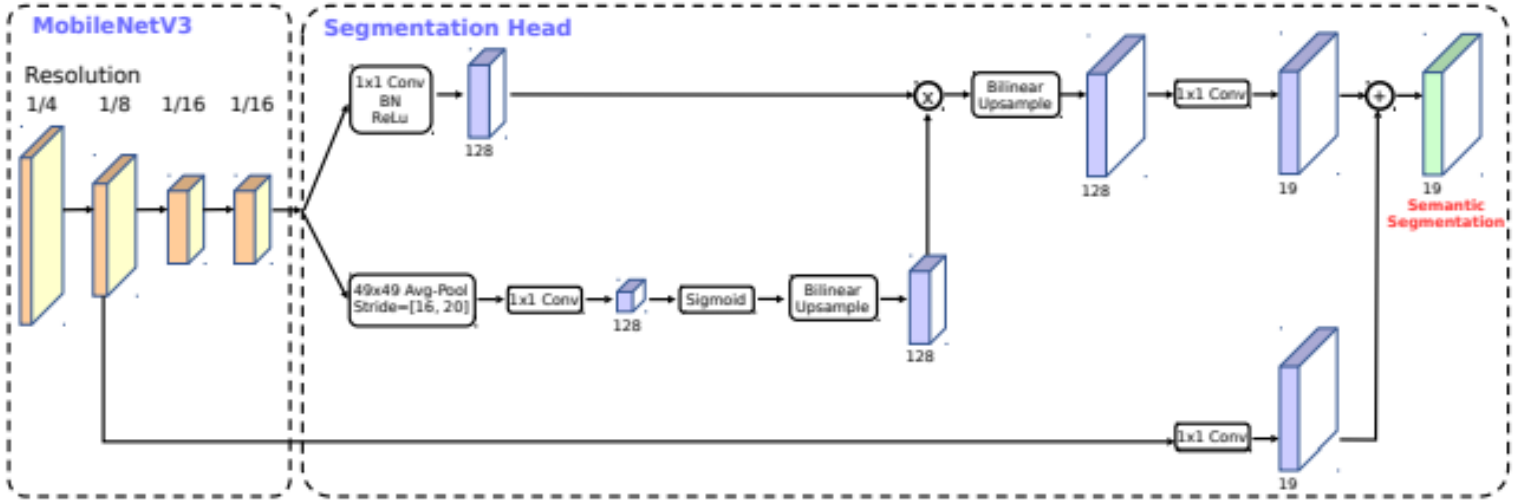


Figure 10. Building on MobileNetV3, the proposed segmentation head, Lite R-ASPP, delivers fast semantic segmentation results while mixing features from multiple resolutions.

Experiments

Results - Semantic Segmentation

R-ASPP

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.

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.