MobileNet

Ho Beom Jeon
UST-ETRI HRI-Lab

2021. 6. 11.

Outline(Contents)

1. Introduction
2. MobileNet V1
3. MobileNet V2
4. Conclusion

1. Introduction

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko
Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam

Google Inc.

{howarda,menglong,bochen,dkalenichenko,weijunw,weyand,anm,hadam}@google.com

MobileNetV2: Inverted Residuals and Linear Bottlenecks

Mark Sandler Andrew Howard Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen
Google Inc.
{sandler, howarda, menglong, azhmogin, lcchen}@google.com

2. MobileNet

1) Abstract
2) Prior Work
3) MobileNet Architecture
4) Experiments
5) Conclusion

2. MobileNet

Abstract

We present a class of efficient models called MobileNets for mobile and embedded vision applications. MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks. We introduce two simple global hyper-parameters that efficiently trade off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem. We present extensive experiments on resource and accuracy tradeoffs and show strong performance compared to other popular models on ImageNet classification. We then demonstrate the effective-ness of MobileNets across a wide range of applications and use cases including object detection, finegrain classifica-tion, face attributes and large scale geo-localization.

1. Depthwise separable Convolution 을 사용하여 경량 딥러닝 네트워크를 설계
2. 2 가지 global hyper parameter를 사용하여 모델을 원하는 어플리케이션에 맞추어 빌드할 수 있다.

2. MobileNet

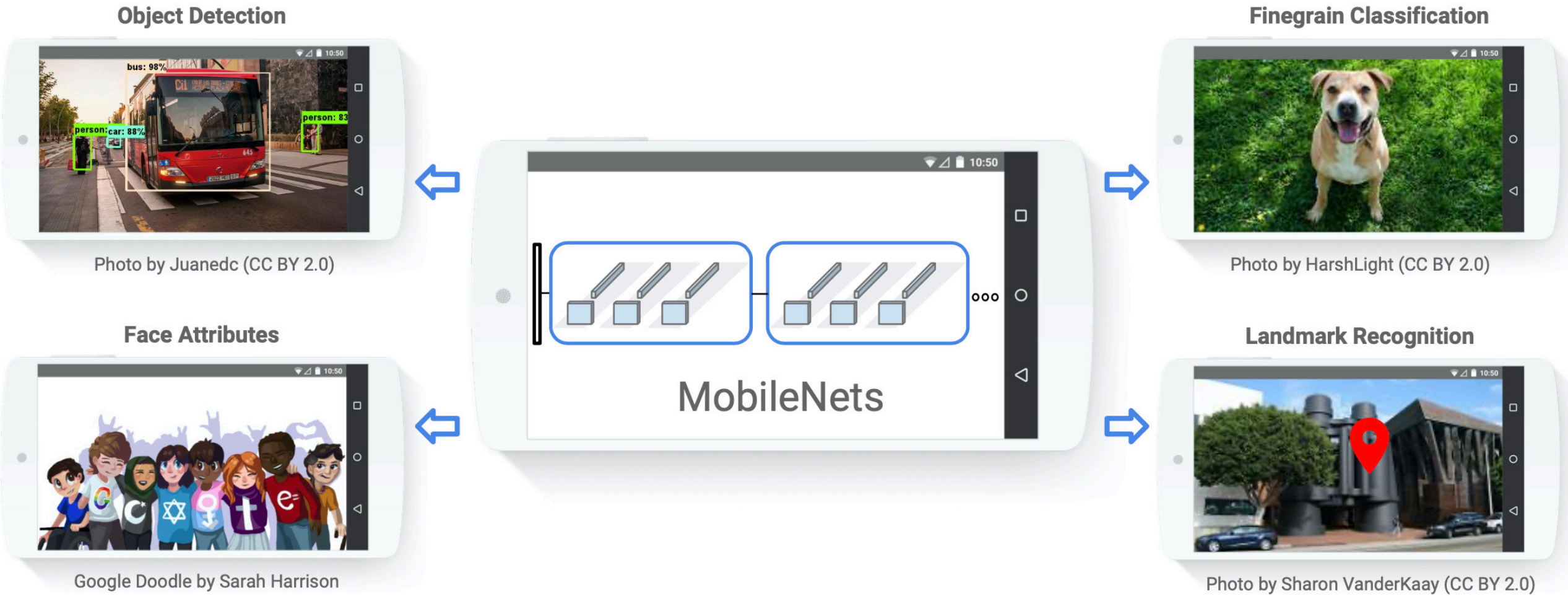


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

2) Prior Work

[26] Rigid-Motion Scattering for Texture Classification arXiv:1403.1687

depthwise separable convolutions initially introduced

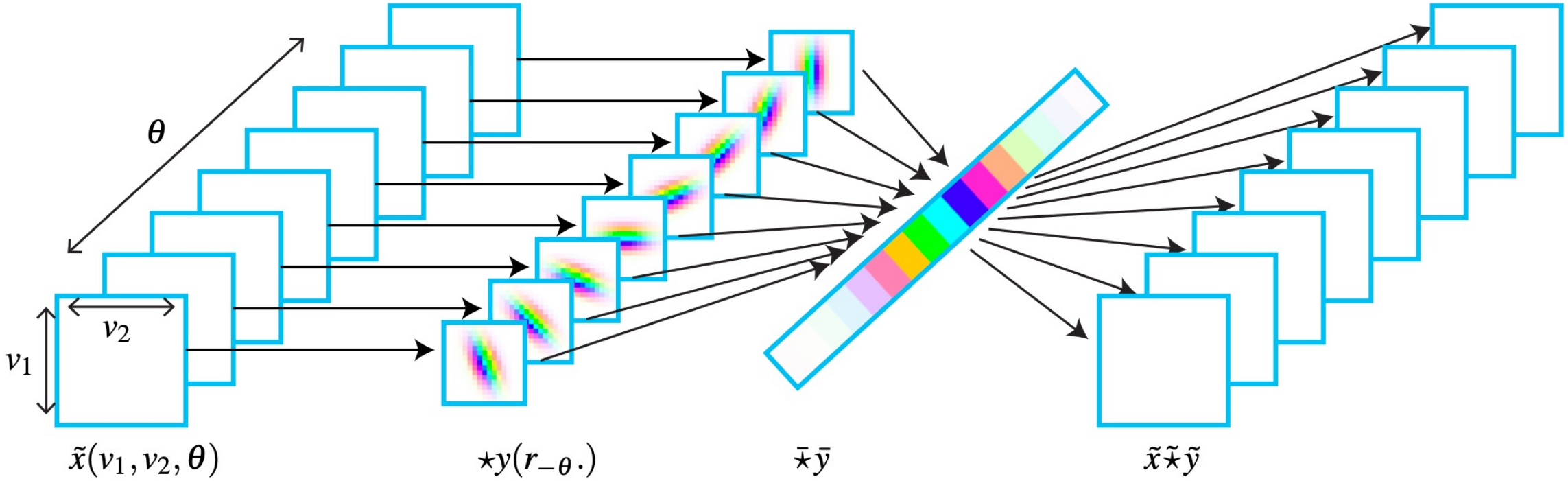


Fig. 6: A rigid-motion convolution (20) with a separable filter $\tilde{y}(v,\theta) = y(v)\bar{y}(\theta)$ in $SE(2)$ can be factorized into a two dimensional convolution with rotated filters $y(r_{-\theta}v)$ and a one dimensional convolution with $\bar{y}(\theta)$.

2) Prior Work

[16] FLATTENED CONVOLUTIONAL NEURAL NETWORKS FOR FEEDFORWARD ACCELERATION arXiv:1412.5474v4

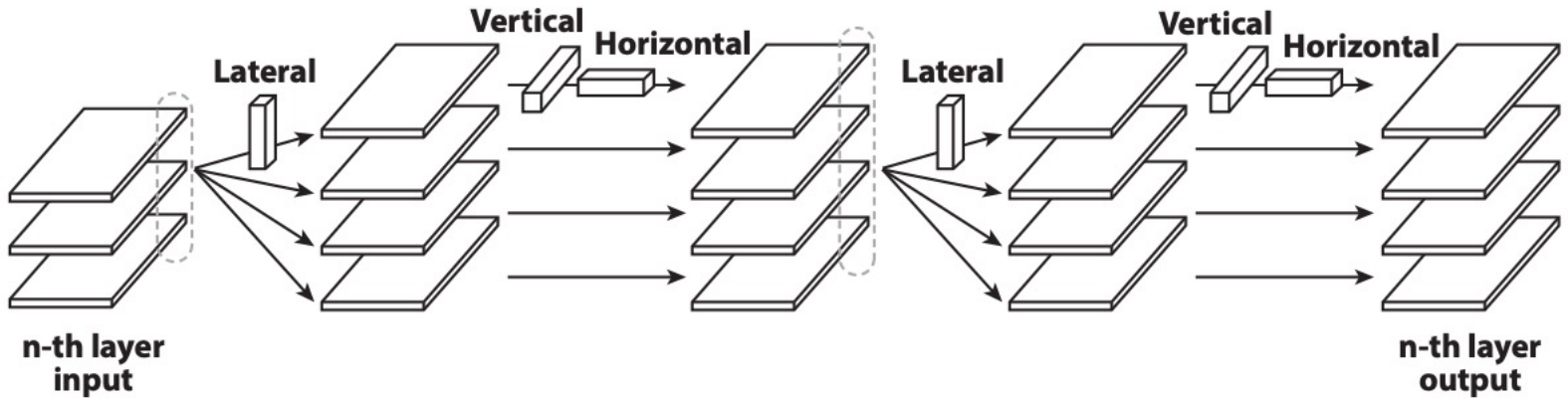
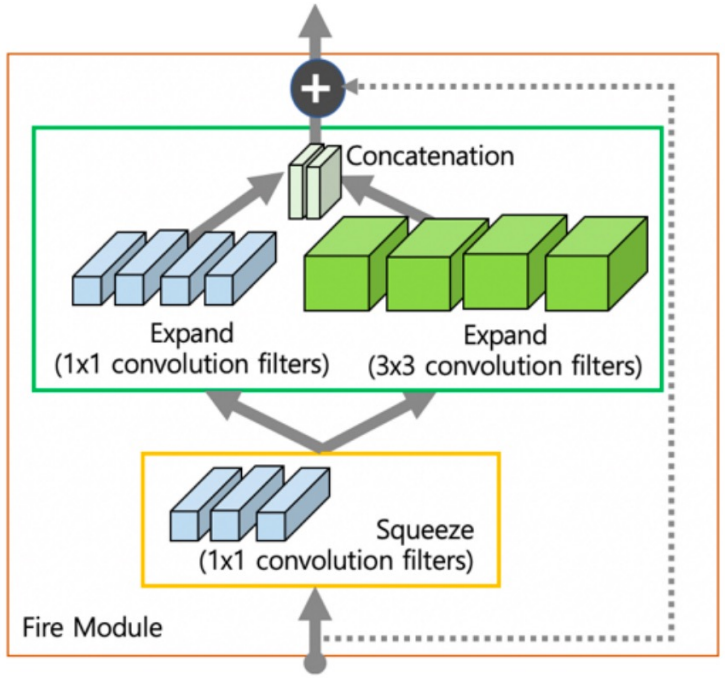


Figure 2: A single layer structure of flattened convolutional networks. Flattened layer includes l sets of 1D-separated convolutions over channels (lateral, L), vertical (V) and horizontal (H) direction. In this work, two stages of LVH combinations ($l = 2$) were chosen by cross-validation and it reported the same accuracy as measured in baseline model. V and H convolutions are operated in *full* mode to preserve the same output dimension as the baseline model. Bias is added after each of 1D convolution, but skipped in this illustration. No non-linear operator is applied within the flattened layer.

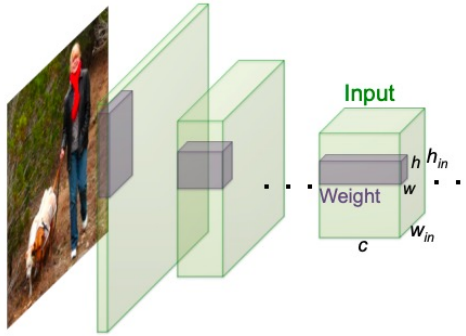
2) Prior Work

[12] SQUEEZENET arXiv:1602.07360v4



(그림 3) 스퀴즈넷(SqueezeNet)의 파이어 모듈(Fire Module)

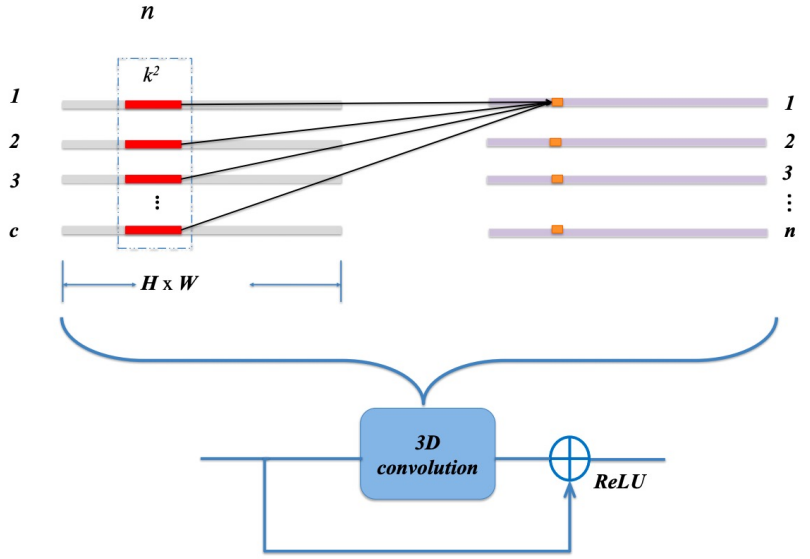
[22] XNOR-Net arXiv:1603.05279v4



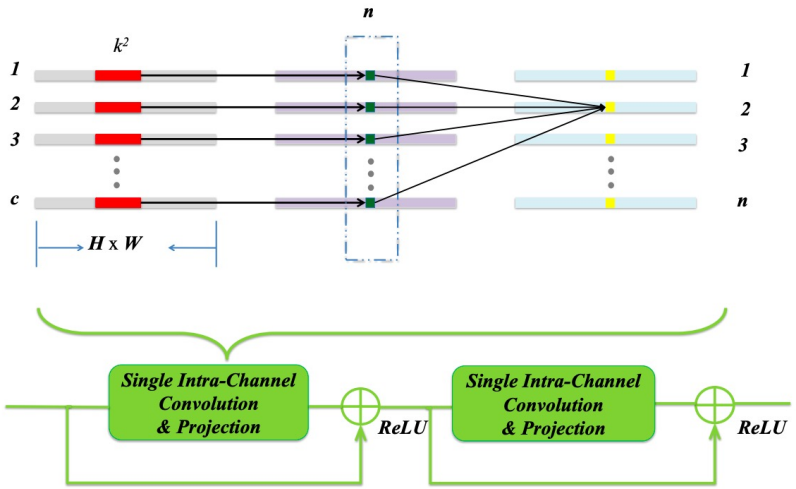
	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52 / Real-Value Weights 0.12 -1.2 ... 0.41 -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52 / Binary Weights 1 -1 ... 1 -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input **(XNOR-Net)**	Binary Inputs 1 -1 ... -1 -1 1 ... 1 / Binary Weights 1 -1 ... 1 -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Fig. 1: We propose two efficient variations of convolutional neural networks. **Binary-Weight-Networks**, when the weight filters contains binary values. **XNOR-Networks**, when both weigh and input have binary values. These networks are very efficient in terms of memory and computation, while being very accurate in natural image classification. This offers the possibility of using accurate vision techniques in portable devices with limited resources.

2) Prior Work

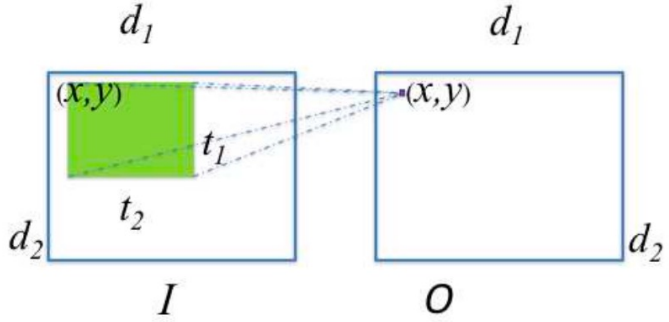


(a) Standard Convolutional Layer with residual connection

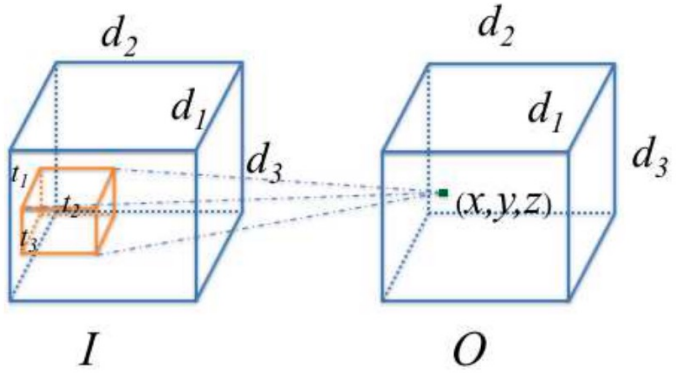


(b) Single Intra-Channel Convolutional Layer

[34] Factorized Convolutional Neural Networks ICCV2017



(a) 2D Topology



(b) 3D Topology

tf.nnseparable_conv2d

```
tf.nn.separable_conv2d(
    input, depthwise_filter, pointwise_filter, strides, padding, data_format=None,
    dilations=None, name=None
)
```

3D conv => 2Dconv + biased 1x1 Conv Projection

2) Prior Work

[3] Xception: Deep Learning with Depthwise Separable Convolutions ICCV2017

Figure 1. A canonical Inception module (Inception V3).

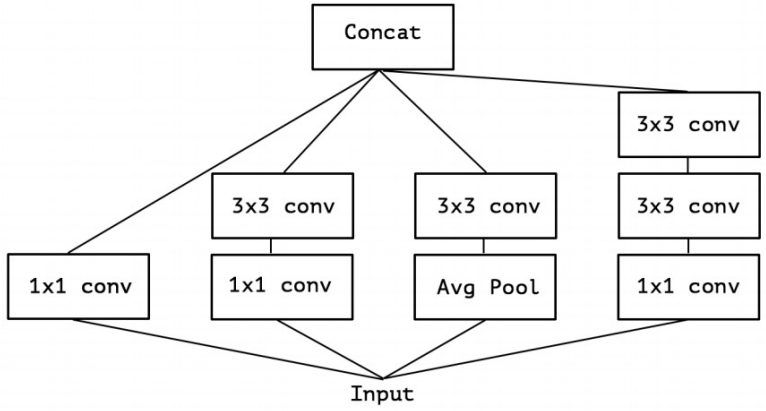


Figure 2. A simplified Inception module.

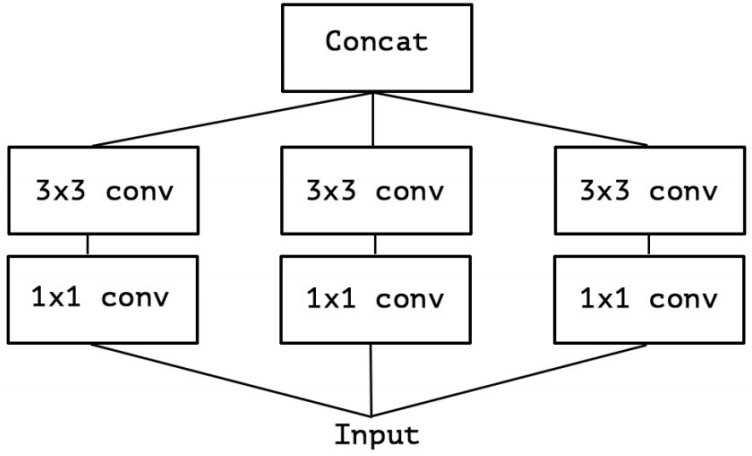


Figure 3. A strictly equivalent reformulation of the simplified Inception module.

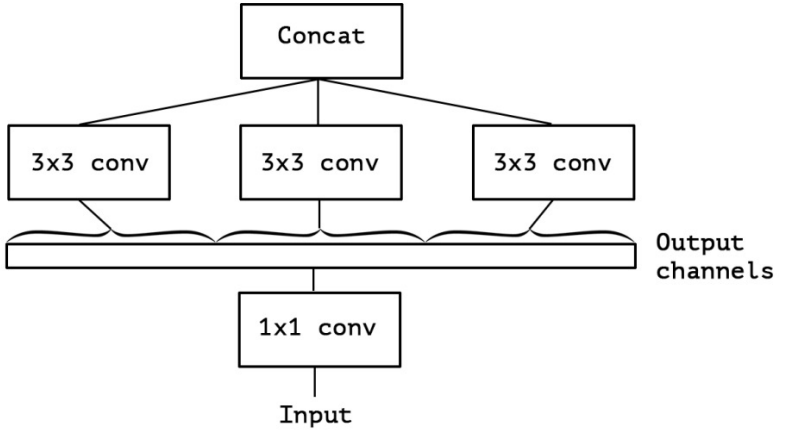
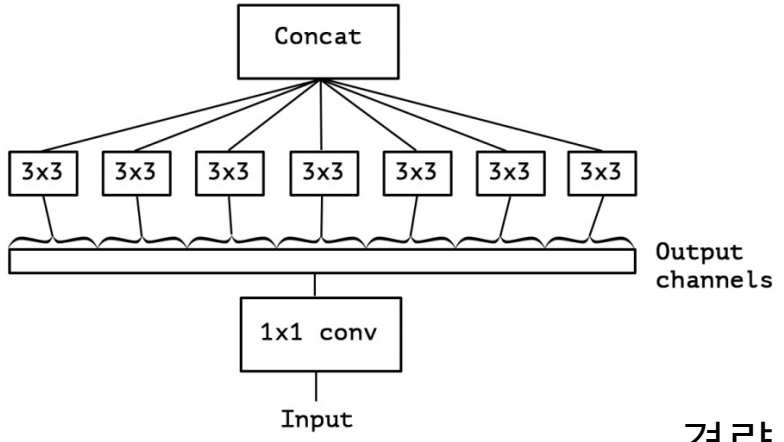


Figure 4. An "extreme" version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.



2) Prior Work

[2] Compressing Neural Networks with the Hashing Trick jmlr.org

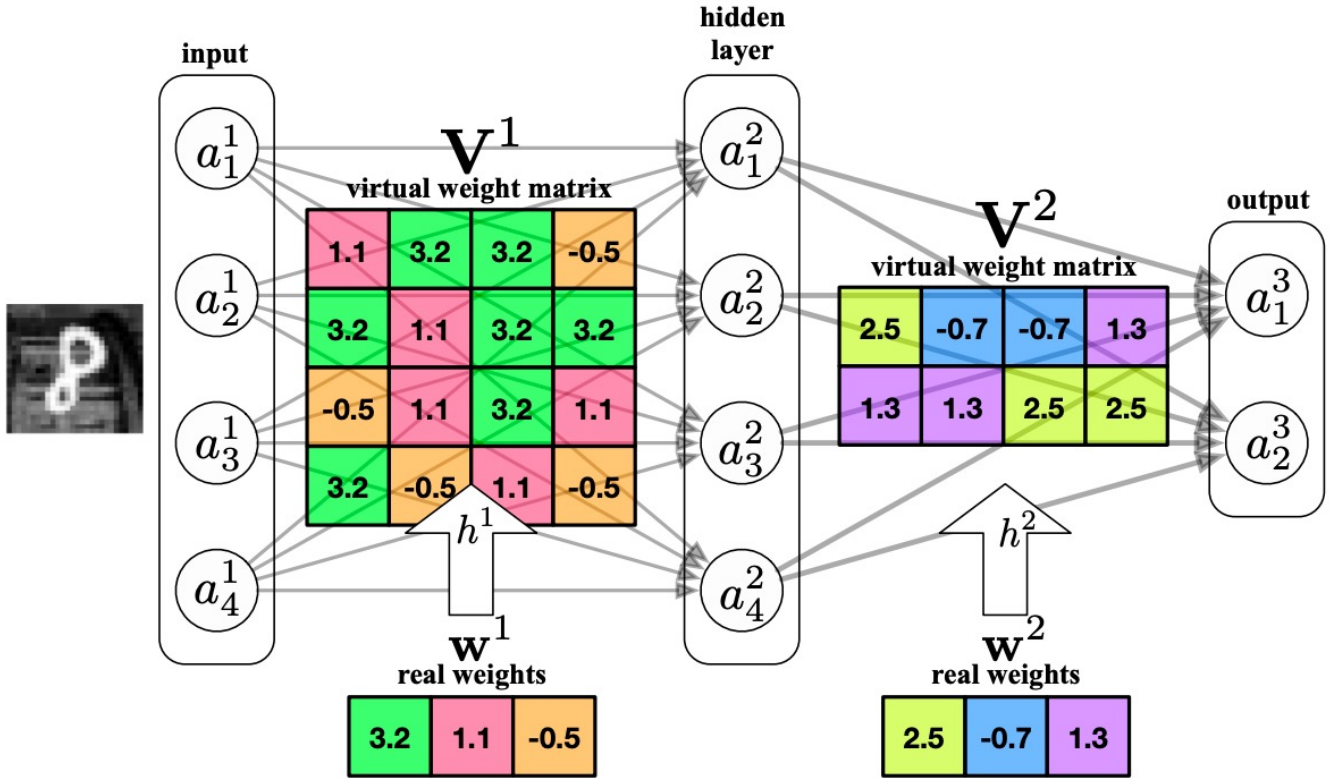


Figure 1. An illustration of a neural network with random weight sharing under compression factor $\frac{1}{4}$. The $16+9=24$ virtual weights are compressed into 6 real weights. The colors represent matrix elements that share the same weight value.

2) Prior Work

[5] DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATIO N AND HUFFMAN CODING arXiv:1510.00149v5

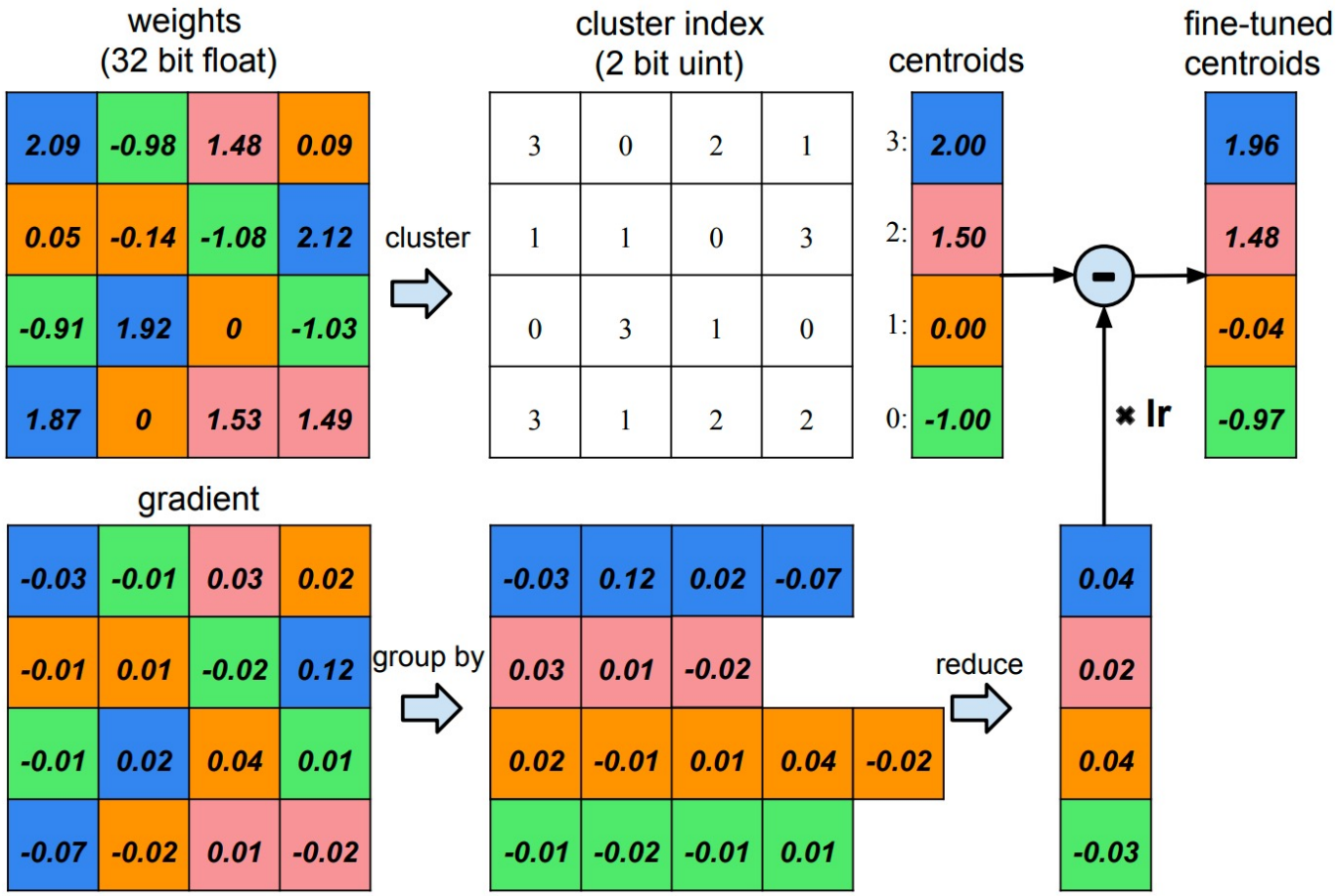


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

2) Prior Work

[36] Quantized Convolutional Neural Networks for Mobile Devices CVPR2016

product quantization

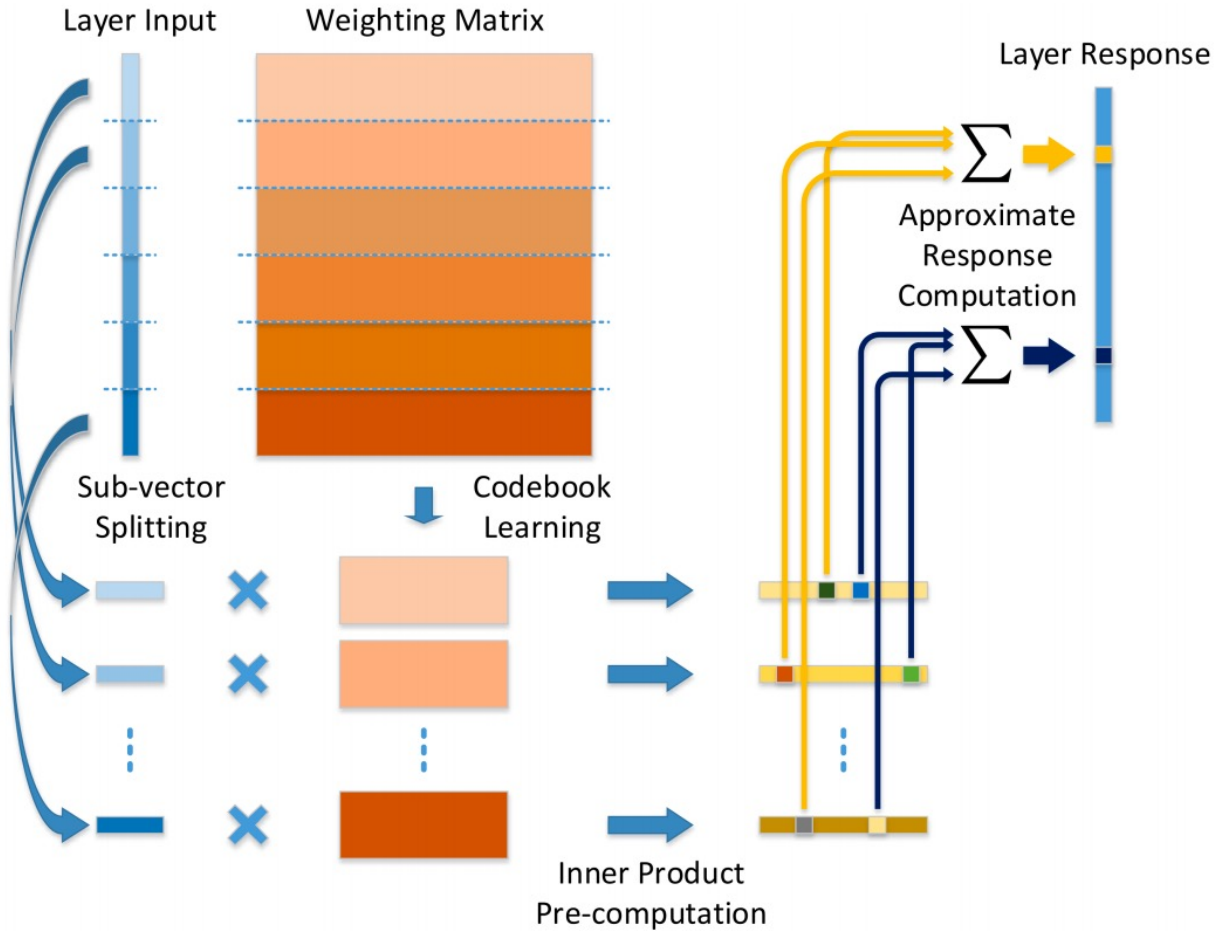
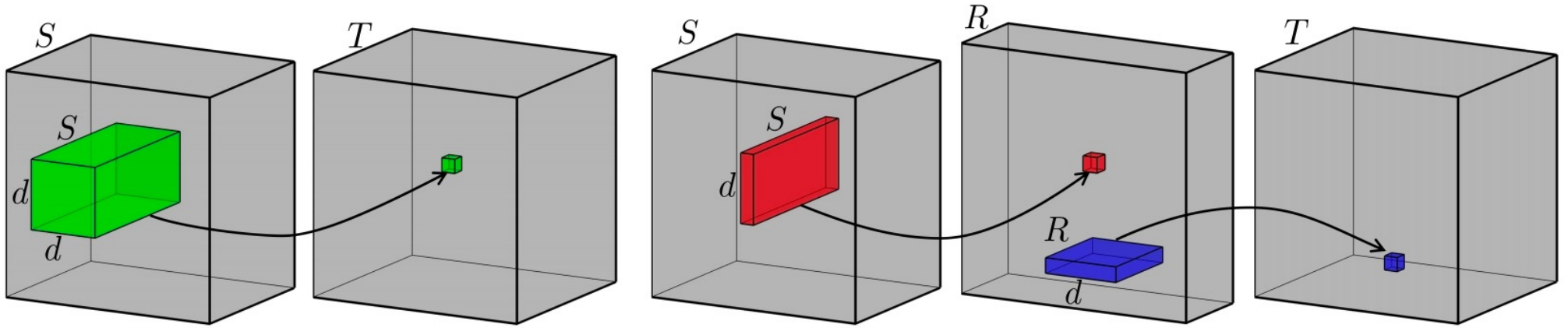


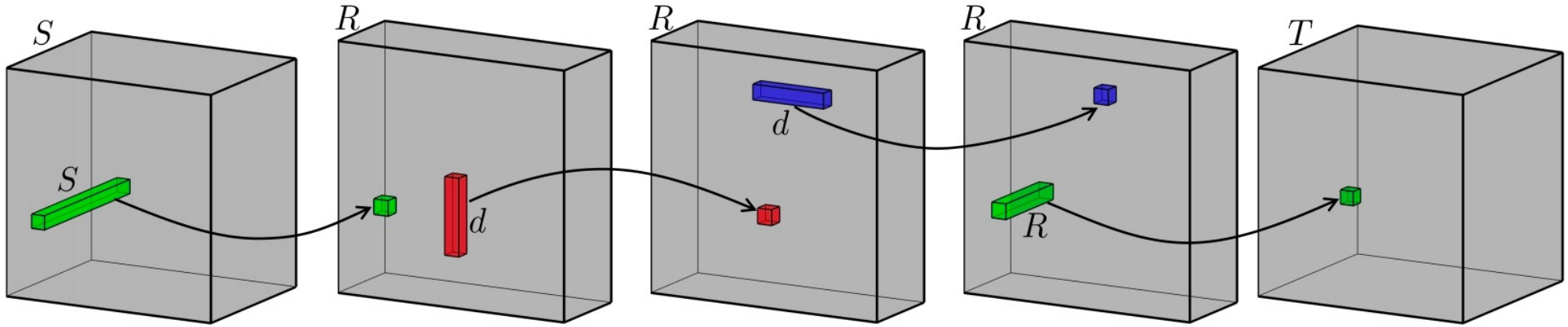
Figure 2. The parameter quantization and test-phase computation process of the fully-connected layer.

2) Prior Work

[14] Speeding up Convolutional Neural Networks with Low Rank Expansions arXiv:1405.3866v1
[20] SPEEDING-UP CONVOLUTIONAL NEURAL NETWORKS USING FINE-TUNED CP-DECOMPOSITION arXiv:1412.6553v3



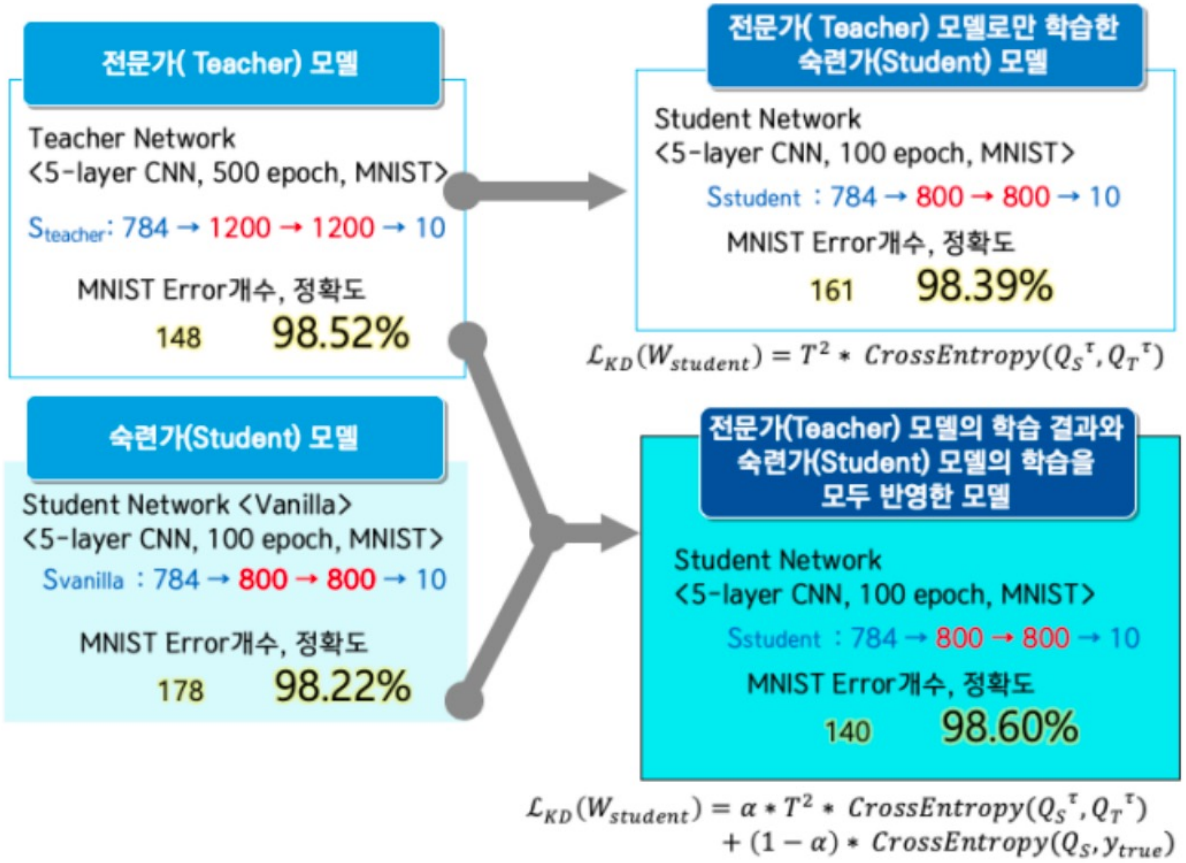
Full convolution [14] Two-component decomposition (Jaderberg et al., 2014a)



[20] CP-decomposition

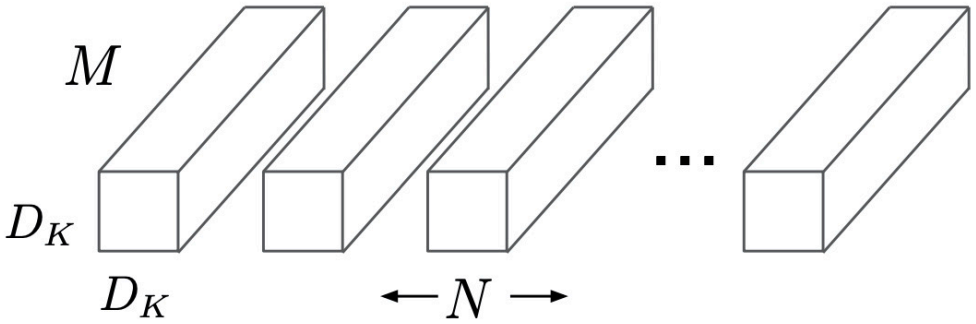
2) Prior Work

[9] **Distilling the Knowledge in a Neural Network** arXiv:1503.02531

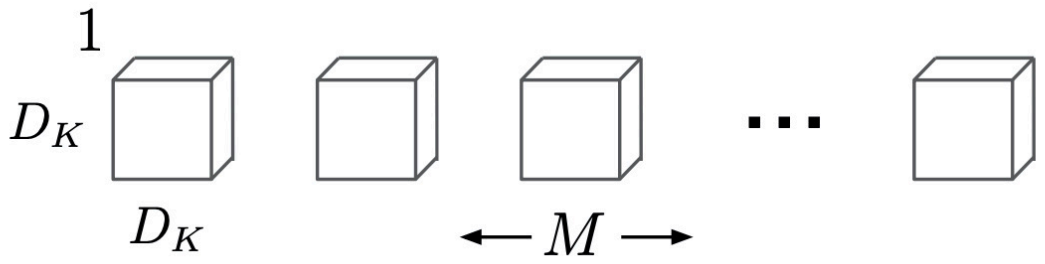


(그림 10) 전문가(Teachers) 모델과 숙련가(Student) 모델의 학습 결과 예

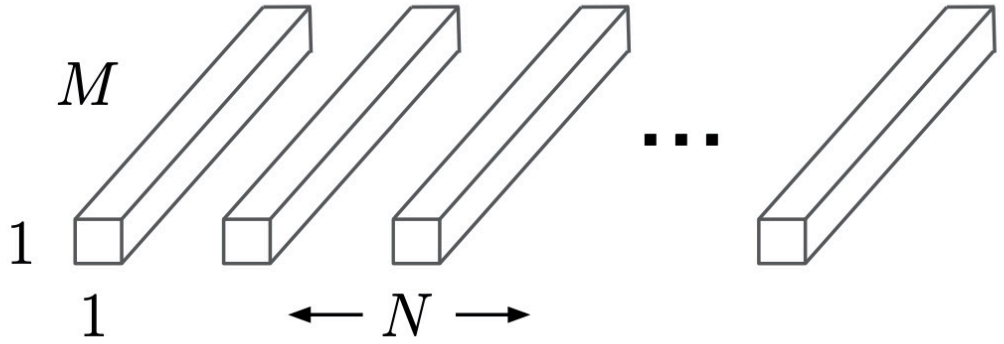
3) MobileNet Architecture



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

3) MobileNet Architecture

Standard Convolution

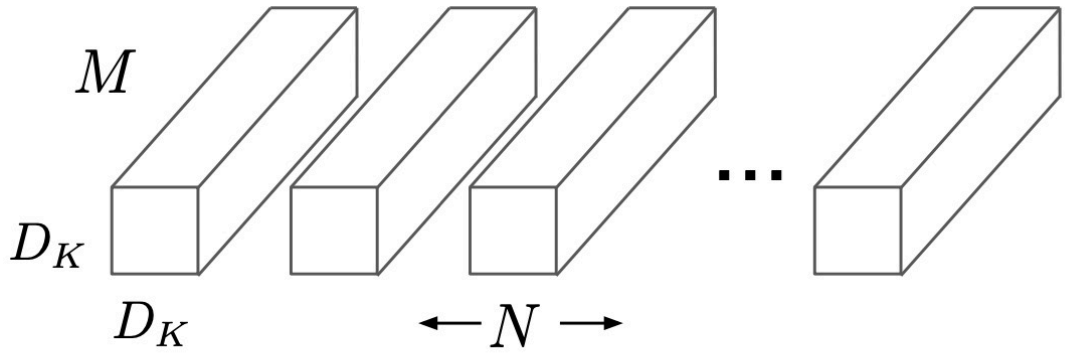
input a $D_F \times D_F \times M$
Feature map F and produces a $D_F \times D_F \times N$
convolution kernel K is $D_K \times D_K \times M \times N$

The output feature map for standard convolution assuming stride one and padding is computed as:

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \qquad (1)$$

Standard convolutions have the computational cost of:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \qquad (2)$$



(a) Standard Convolution Filters

3) MobileNet Architecture

source

Depthwise separable convolution

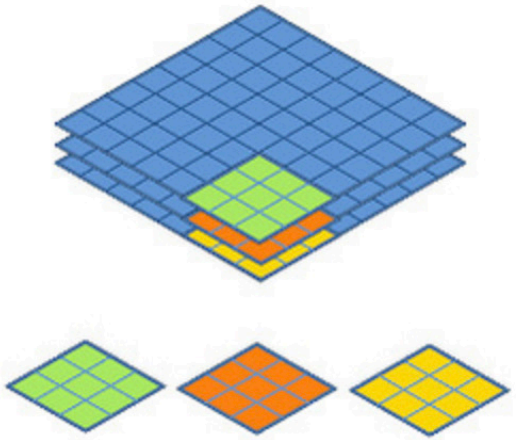
made up of two layers: depthwise convolutions and pointwise convolutions.

Depthwise convolutions

input a $D_F \times D_F \times M$

Feature map F and produces a $D_F \times D_F \times M$

convolution kernel K is $D_K \times D_K \times M$



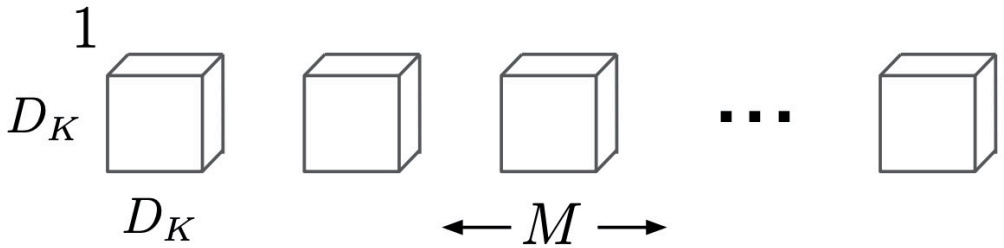
Depthwise Convolutional Filters

Depthwise convolution with one filter per input channel (input depth) can be written as:

$$\hat{\mathbf{G}}_{k,l,m} = \sum_{i,j} \hat{\mathbf{K}}_{i,j,m} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \qquad (3)$$

Depthwise convolution has a computational cost of:

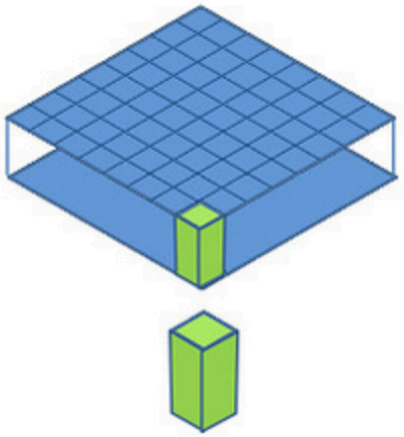
$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \qquad (4)$$



(b) Depthwise Convolutional Filters

3) MobileNet Architecture

source



Pointwise Convolutional Filters

Depthwise separable convolution

made up of two layers: depthwise convolutions and pointwise convolutions.

Pointwise convolutions.

input a $D_F \times D_F \times M$

Feature map F and produces a $D_F \times D_F \times N$

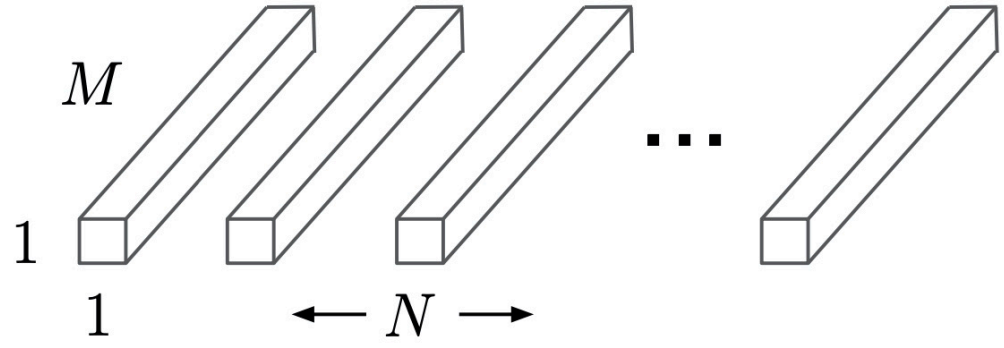
convolution kernel K is $1 \times 1 \times N$

$$M \cdot N \cdot D_F \cdot D_F$$

Depthwise separable convolutions cost:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \quad (5)$$

depthwise convolutions + pointwise convolutions



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

3) MobileNet Architecture

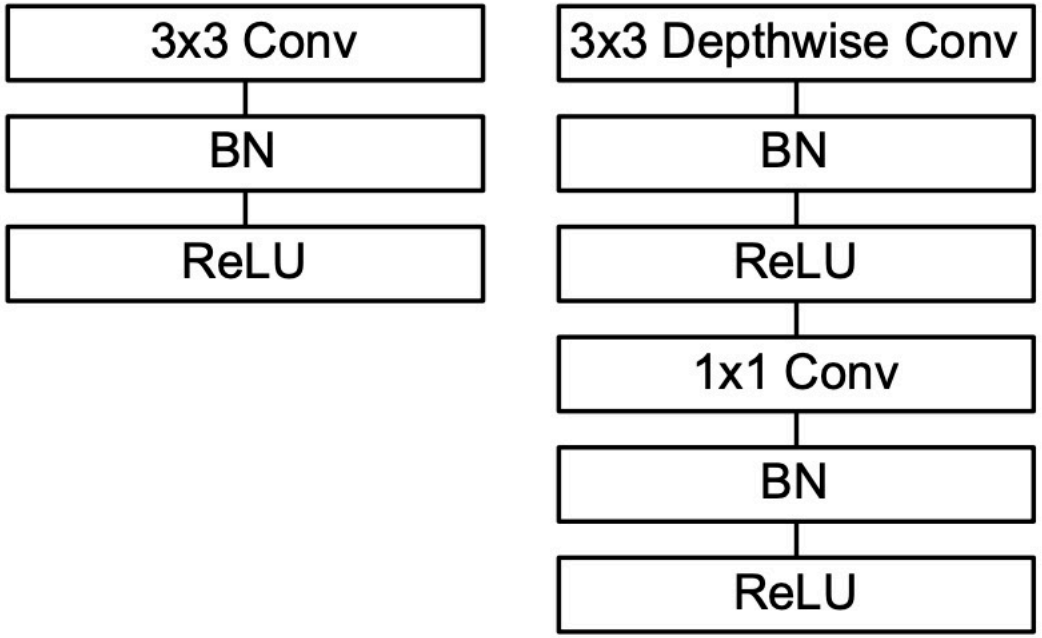


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}$$

$$= \frac{1}{N} + \frac{1}{D_K^2}$$

MobileNet uses 3 × 3 depthwise separable convolutions which uses between 8 to 9 times less computation than standard convolutions

3) MobileNet Architecture

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

3) MobileNet Architecture

Width Multiplier: Thinner Models

The computational cost of a depthwise separable convolution with width multiplier α is:

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F \quad (6)$$

1, 0.75, 0.5 and 0.25.

Resolution Multiplier: Reduced Representation

We can now express the computational cost for the core layers of our network as depthwise separable convolutions with width multiplier α and resolution multiplier ρ:

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F \quad (7)$$

224, 192, 160, and 128.

Table 3. Resource usage for modifications to standard convolution. Note that each row is a cumulative effect adding on top of the previous row. This example is for an internal MobileNet layer with $D_K = 3$, $M = 512$, $N = 512$, $D_F = 14$.

Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$ (160)	15.1	0.15

단일 곱셈-누산기(영어: Fused multiply-add, FMA)는 부동소수점 곱하기와 더하기를 한번에 수행한다. 두 수의 곱한값을 누산기에서 또 다른 값과 더하는 동작이다. 아래의 식에서 b와 c의 곱하기 결과값을 a와 더한 후에 그 결과를 a에 저장됨을 볼 수 있다.

$$a \leftarrow a + (b \times c)$$

3) MobileNet Architecture

stackoverflow.com/how-to-count-multiply-adds-operations

Given a defined Convolutional Neural Network, is there a function or method on how to compute the number of `Multiply-Adds` operations?

Counting the `Multiply-Add` operations is equivalent to calculating the FLOPs of a model. This can be achieved using the profiler from tensorflow.

```
flops = tf.profiler.profile(graph,\
    options=tf.profiler.ProfileOptionBuilder.float_operation())
print('FLOP = ', flops.total_float_ops)
```

Be sure to look at the caveats explained in . Basically:

- The number of FLOPs calculated might include initialisation operations like Multiply-Add from initialising your weights with a Gaussian distribution for instance, you need to freeze the graph to rule those possibly irrelevant Multiply-Add operations,
- The Multiply-Add calculations from TensorFlow are approximate. An issue has been opened on this matter .

4) Experiments

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

4) Experiments

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

4) Experiments

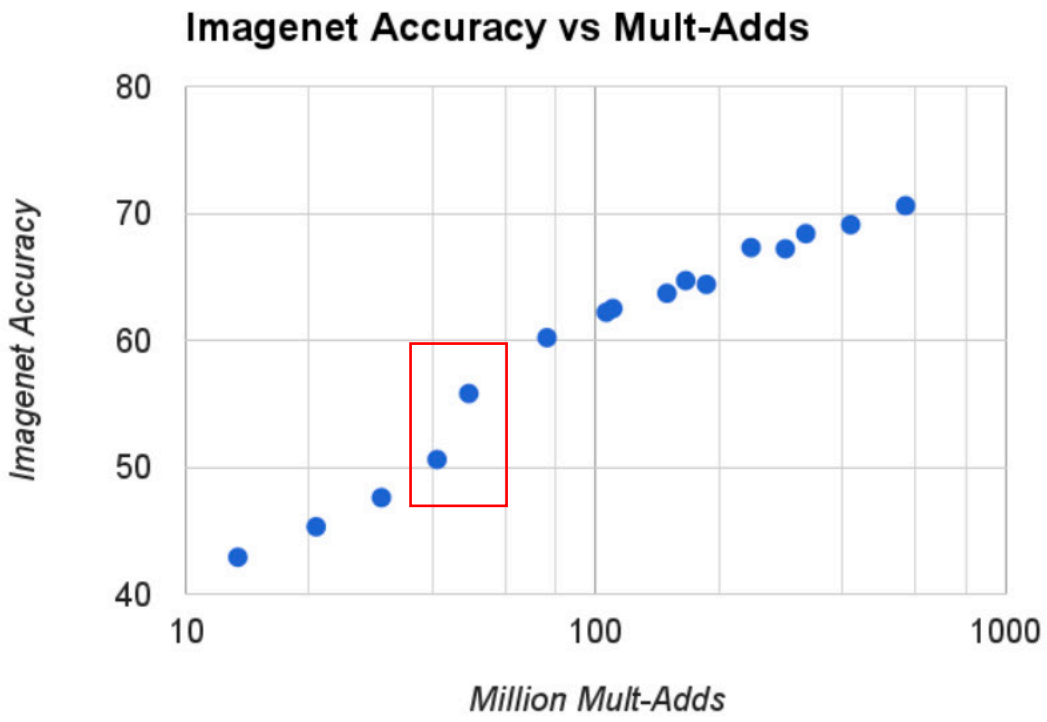
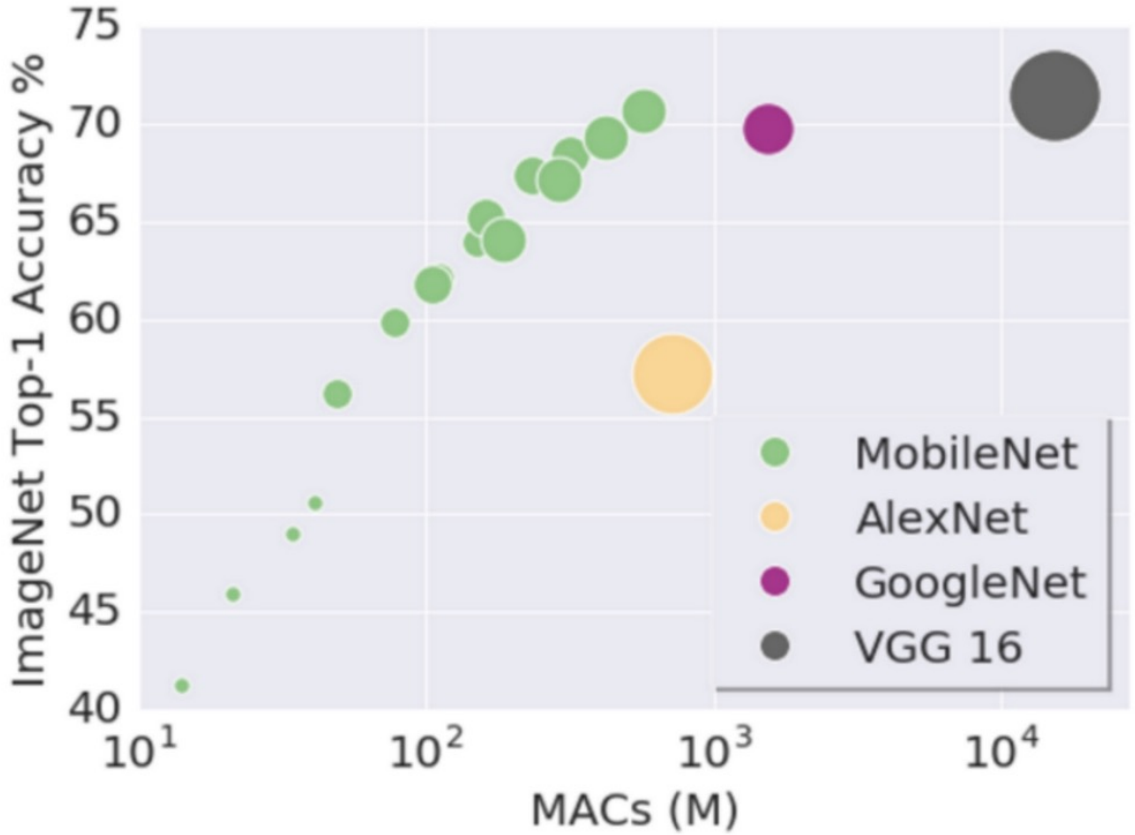


Figure 4. This figure shows the trade off between computation (Mult-Adds) and accuracy on the ImageNet benchmark. Note the log linear dependence between accuracy and computation.



Source

α ∈ {1, 0.75, 0.5, 0.25} and resolutions {224, 192, 160, 128}.

4) Experiments

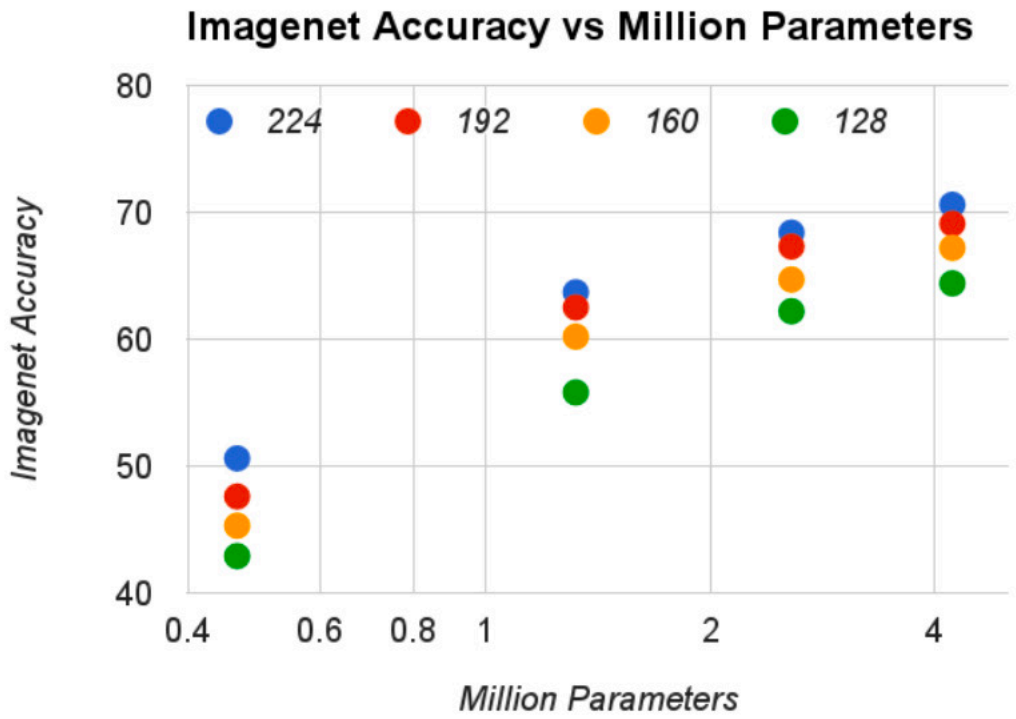


Figure 5. This figure shows the trade off between the number of parameters and accuracy on the ImageNet benchmark. The colors encode input resolutions. The number of parameters do not vary based on the input resolution.

4) Experiments

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

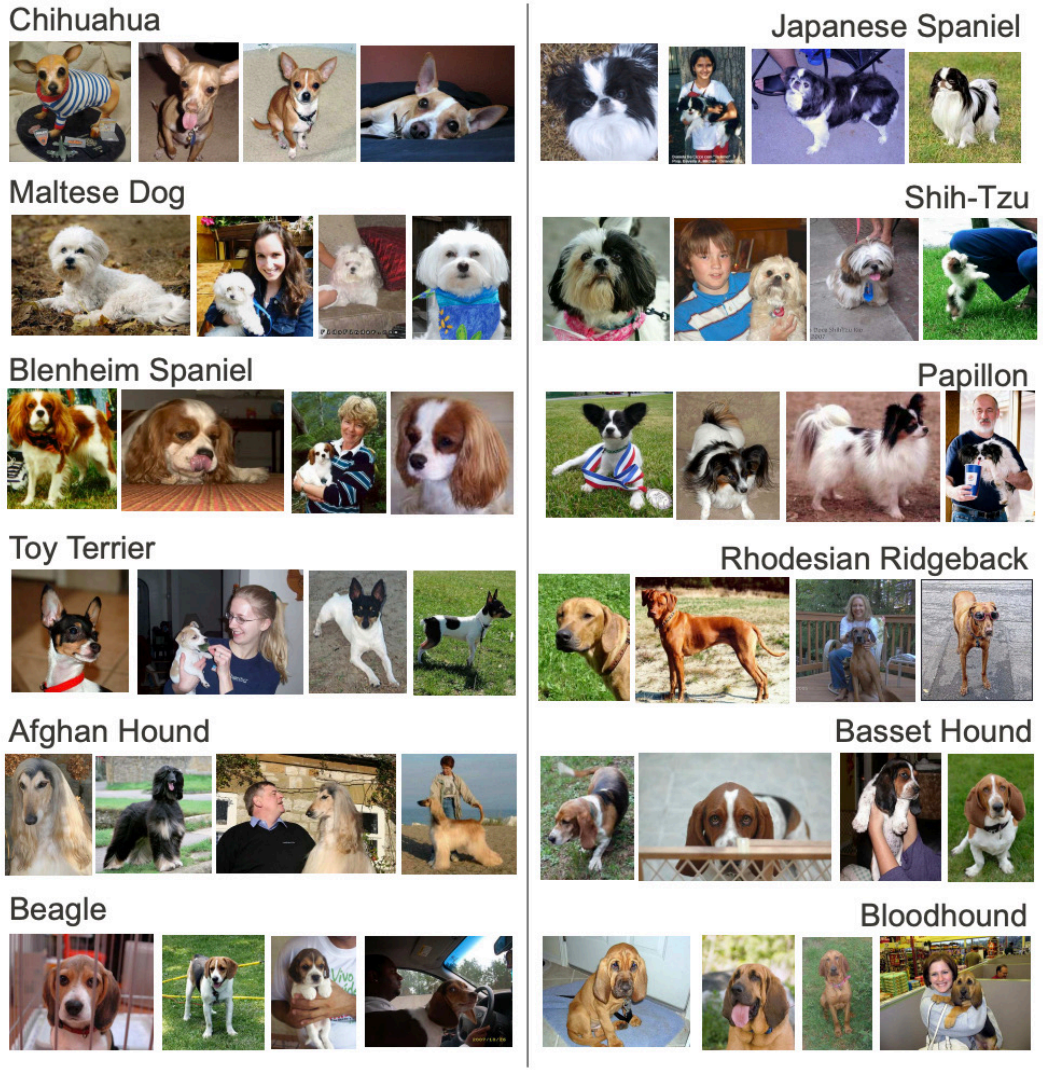
VGG16 while being 32 times smaller and 27 times less compute intensive
GoogleNet while being smaller and more than 2.5 times less computation.

4) Experiments

Table 10. MobileNet for Stanford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

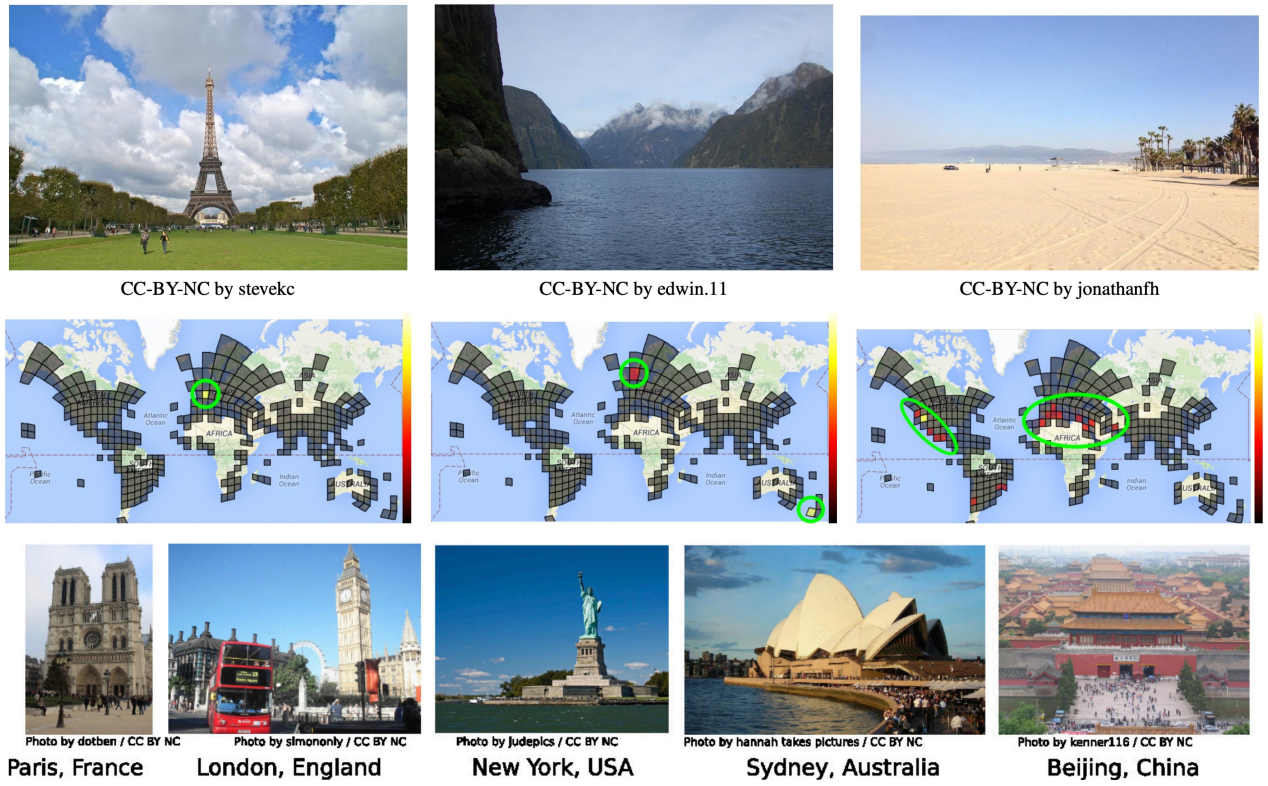
Dataset	No. of classes	No. of images	Images per class	Visibility varies?	Bounding boxes?
CUB-200 [4]	**200**	6033	30	**Yes**	**Yes**
PPMI [5]	24	4800	**200**	No	**Yes**
PASCAL [2]	9	1221	**135**	**Yes**	**Yes**
Stanford Dogs	**120**	**20580**	**180**	**Yes**	**Yes**



4) Experiments

Table 11. Performance of PlaNet using the MobileNet architecture. Percentages are the fraction of the Im2GPS test dataset that were localized within a certain distance from the ground truth. The numbers for the original PlaNet model are based on an updated version that has an improved architecture and training dataset.

Scale	Im2GPS [7]	PlaNet [35]	PlaNet MobileNet
Continent (2500 km)	51.9%	77.6%	79.3%
Country (750 km)	35.4%	64.0%	60.3%
Region (200 km)	32.1%	51.1%	45.2%
City (25 km)	21.9%	31.7%	31.7%
Street (1 km)	2.5%	11.0%	11.4%

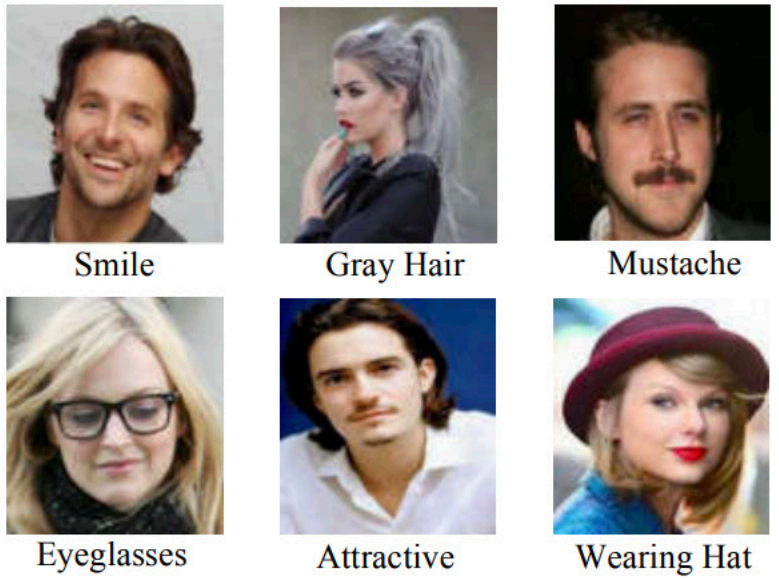


4) Experiments

Table 12. Face attribute classification using the MobileNet architecture. Each row corresponds to a different hyper-parameter setting (width multiplier α and image resolution).

Width Multiplier / Resolution	Mean AP	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	88.7%	568	3.2
0.5 MobileNet-224	88.1%	149	0.8
0.25 MobileNet-224	87.2%	45	0.2
1.0 MobileNet-128	88.1%	185	3.2
0.5 MobileNet-128	87.7%	48	0.8
0.25 MobileNet-128	86.4%	15	0.2
Baseline	86.9%	1600	7.5

[32] YFCC100M



4) Experiments

Table 13. COCO object detection results comparison using different frameworks and network architectures. mAP is reported with COCO primary challenge metric (AP at IoU=0.50:0.05:0.95)

Framework Resolution	Model	mAP	Billion Mult-Adds	Million Parameters
SSD 300	deeplab-VGG	21.1%	34.9	33.1
	Inception V2	22.0%	3.8	13.7
	MobileNet	19.3%	1.2	6.8
Faster-RCNN 300	VGG	22.9%	64.3	138.5
	Inception V2	15.4%	118.2	13.3
	MobileNet	16.4%	25.2	6.1
Faster-RCNN 600	VGG	25.7%	149.6	138.5
	Inception V2	21.9%	129.6	13.3
	Mobilenet	19.8%	30.5	6.1

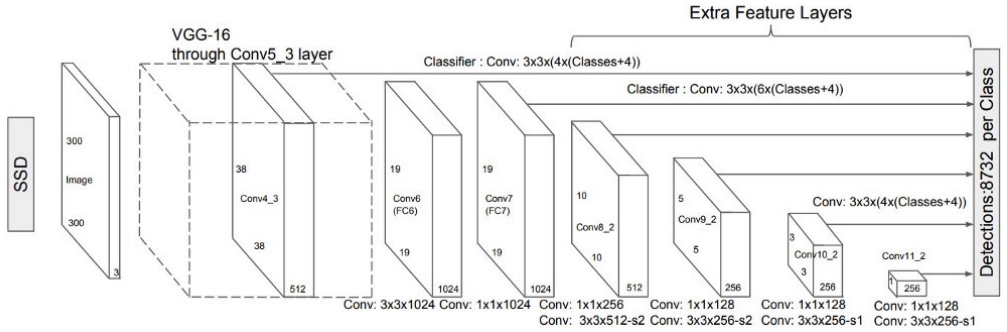


Figure 6. Example objection detection results using MobileNet SSD.

4) Experiments

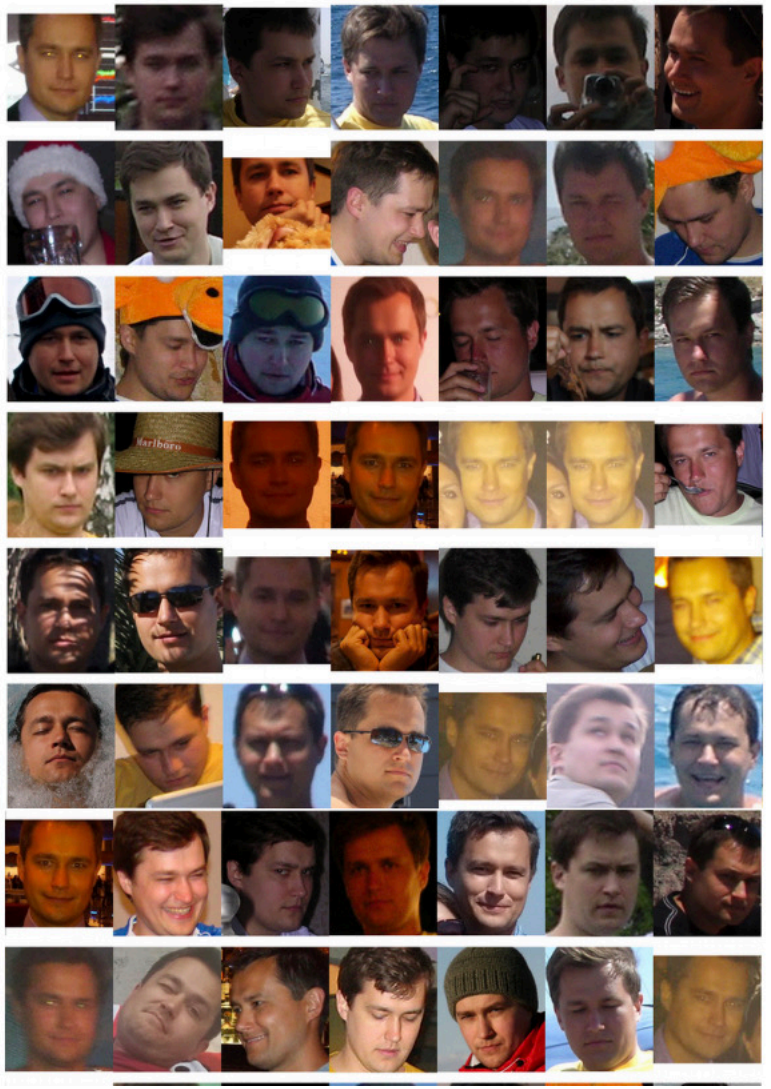
LFW (Labeled Faces in the Wild) dataset

Information:

- 13233 images
- 5749 people
- 1680 people with two or more images

Table 14. MobileNet Distilled from FaceNet

Model	1e-4 Accuracy	Million Mult-Adds	Million Parameters
FaceNet [25]	83%	1600	7.5
1.0 MobileNet-160	79.4%	286	4.9
1.0 MobileNet-128	78.3%	185	5.5
0.75 MobileNet-128	75.2%	166	3.4
0.75 MobileNet-128	72.5%	108	3.8



[25] Figure 7. **Face Clustering.** Shown is an exemplar cluster for one user. All these images in the users personal photo collection were clustered together.

5) Conclusion

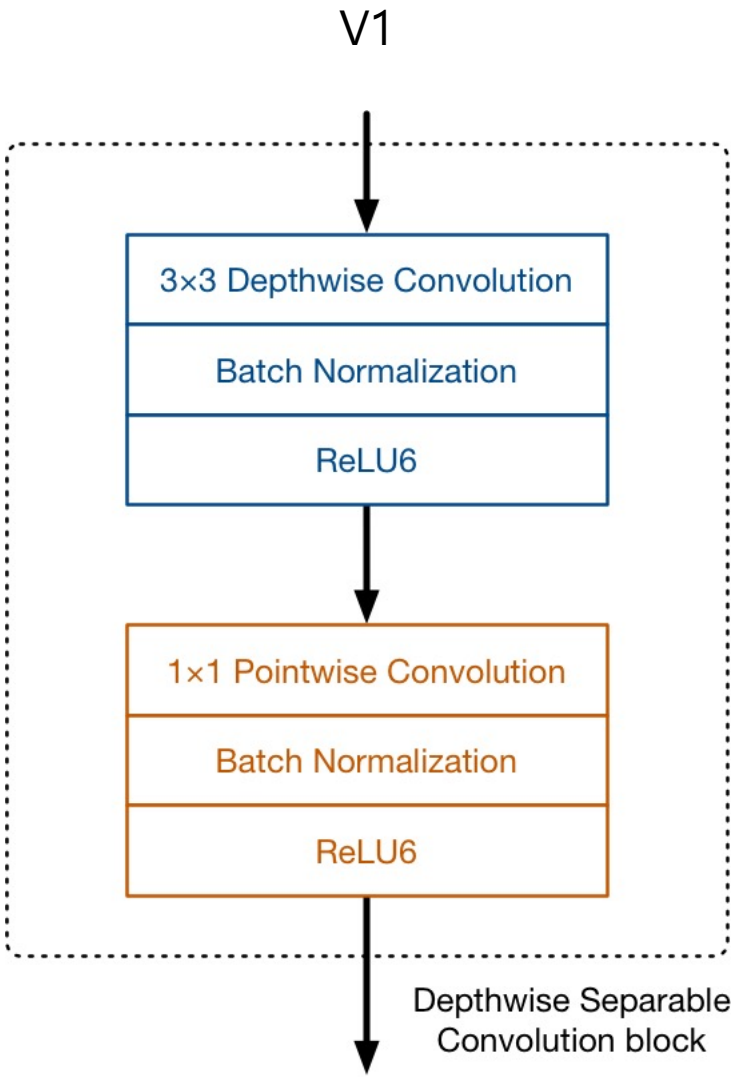
5. Conclusion

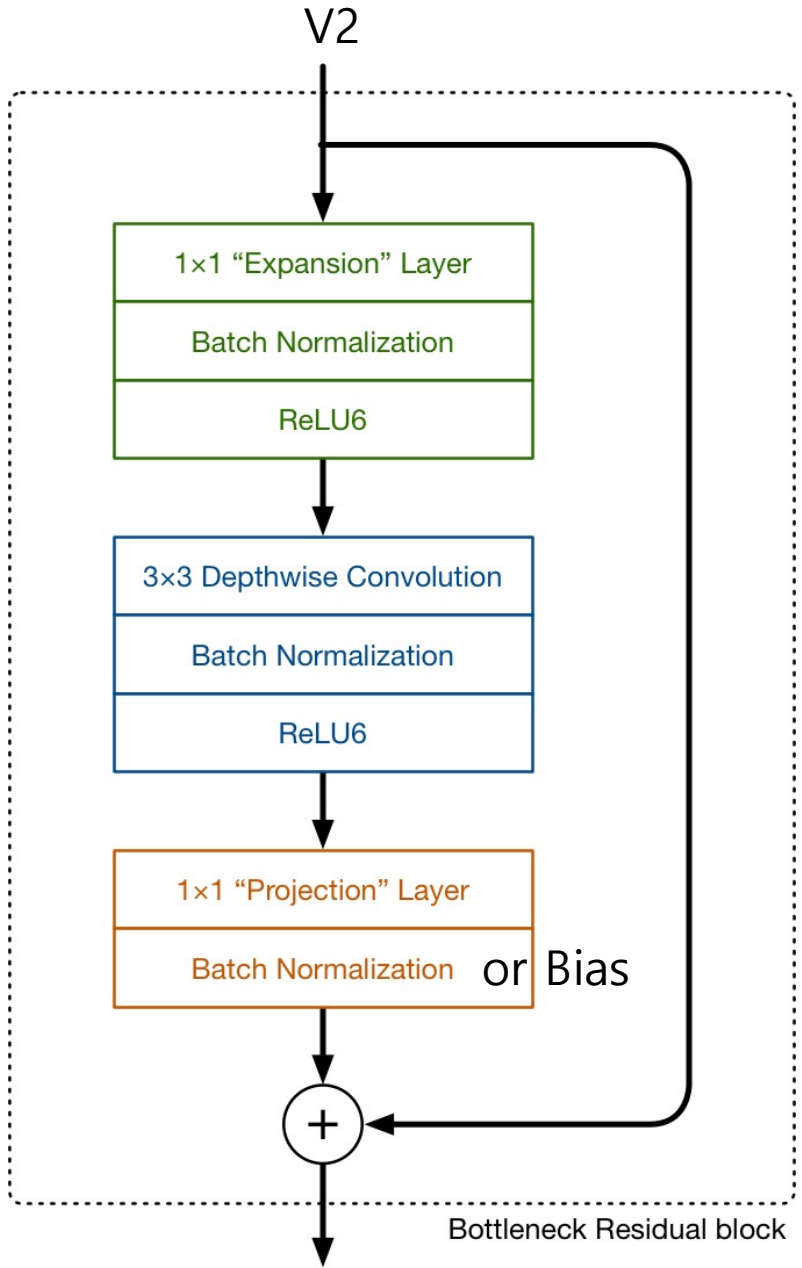
We proposed a new model architecture called MobileNets based on depthwise separable convolutions. We investigated some of the important design decisions leading to an efficient model. We then demonstrated how to build smaller and faster MobileNets using width multiplier and resolution multiplier by trading off a reasonable amount of accuracy to reduce size and latency. We then compared different MobileNets to popular models demonstrating superior size, speed and accuracy characteristics. We concluded by demonstrating MobileNet's effectiveness when applied to a wide variety of tasks. As a next step to help adoption and exploration of MobileNets, we plan on releasing models in Tensor Flow.

3. MobileNet V2

1) Linear Bottlenecks
2) Inverted residual blocks

3. MobileNet V2





http://machinethink.net/blog/mobilenet-v2/

3. MobileNet V2

Linear Bottlenecks

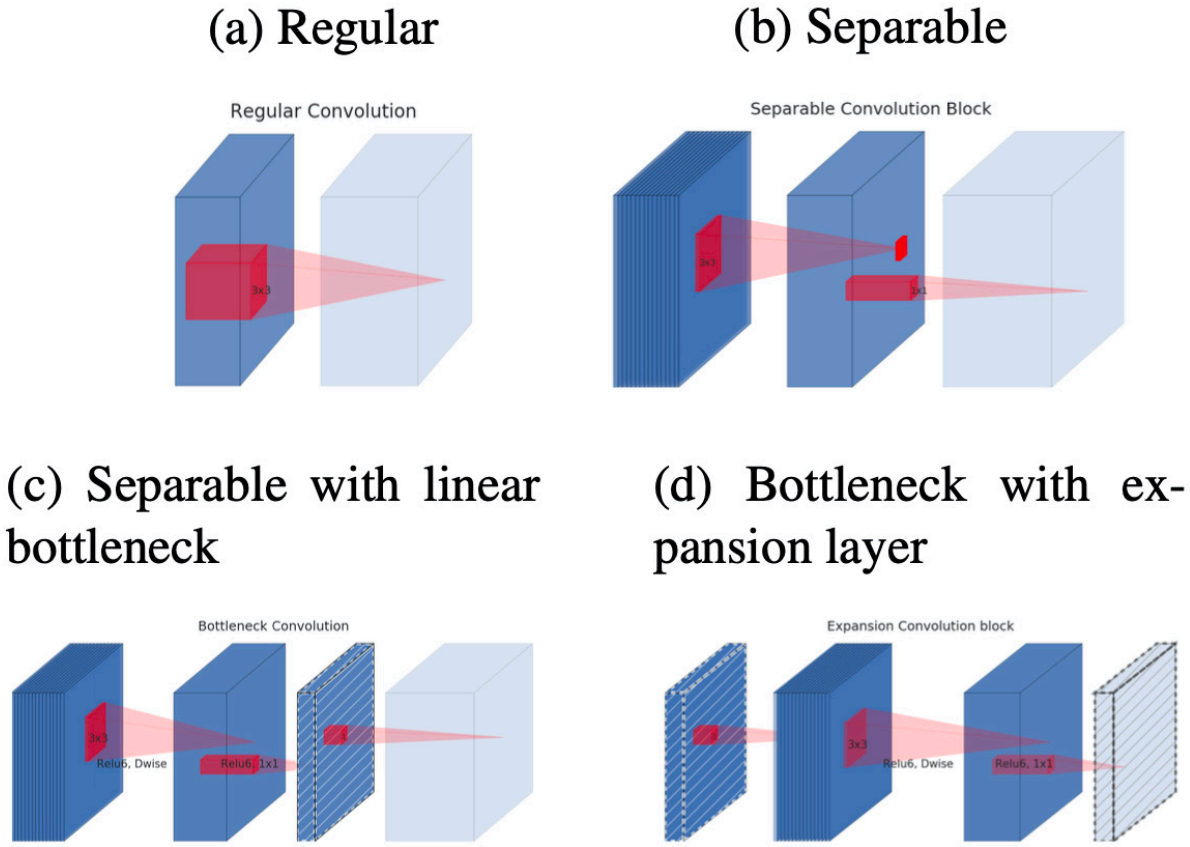


Figure 2: Evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities. The last (lightly colored) layer indicates the beginning of the next block. Note: 2d and 2c are equivalent blocks when stacked. Best viewed in color.

3. MobileNet V2

Linear Bottlenecks

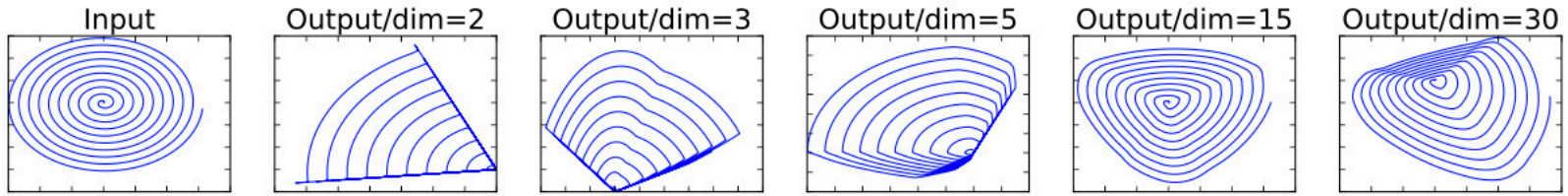


Figure 1: Examples of ReLU transformations of low-dimensional manifolds embedded in higher-dimensional spaces. In these examples the initial spiral is embedded into an n-dimensional space using random matrix T followed by ReLU, and then projected back to the 2D space using T^{-1}. In examples above $n = 2, 3$ result in information loss where certain points of the manifold collapse into each other, while for $n = 15$ to 30 the transformation is highly non-convex.

3. MobileNet V2

Linear Bottlenecks

데이터가 입력되었을 때, 특정 레이어의 활성 값은 “manifold of interest”를 구성합니다. 이 manifold of interest는 어떤 매니폴드 위에 존재하는 입력 데이터가 레이어를 통과하면서 그 실제 매니폴드의 low-dimensional subspace에 임베딩된다고 알려져왔습니다.

얼핏 생각하면 단순히 레이어의 차원을 줄임으로써 이를 달성할 수 있을 것 같고, 실제로 MobileNetV1에서 이런 방식으로 구현에 성공하였습니다. MobileNetV1의 width multiplier를 이용하여 실제 공간 전체로 확장할 수 있도록 활성 함수의 manifold of interest의 차원을 줄이도록 하였습니다. 하지만 딥 컨볼루셔널 뉴럴 네트워크의 ReLU와 같은 것들이 실제로 위치 변환에 대해서도 비선형인 점을 감안하면, 이런 성질은 들어맞지 않게 됩니다.

ReLU가 채널에 적용될 때 필연적으로 채널의 정보를 잃어버리게 됩니다. 하지만, 많은 채널을 가지고 있다면 어떤 채널에서 없어진 정보들이 다른 채널에는 살아있는 구조를 띄게 됩니다. 따라서 입력 매니폴드가 활성값 매니폴드의 lower-dimensional subspace로 충분하게 임베딩 된다면, ReLU 변환을 적용하더라도 정보가 유지된다는 것을 알 수 있습니다.

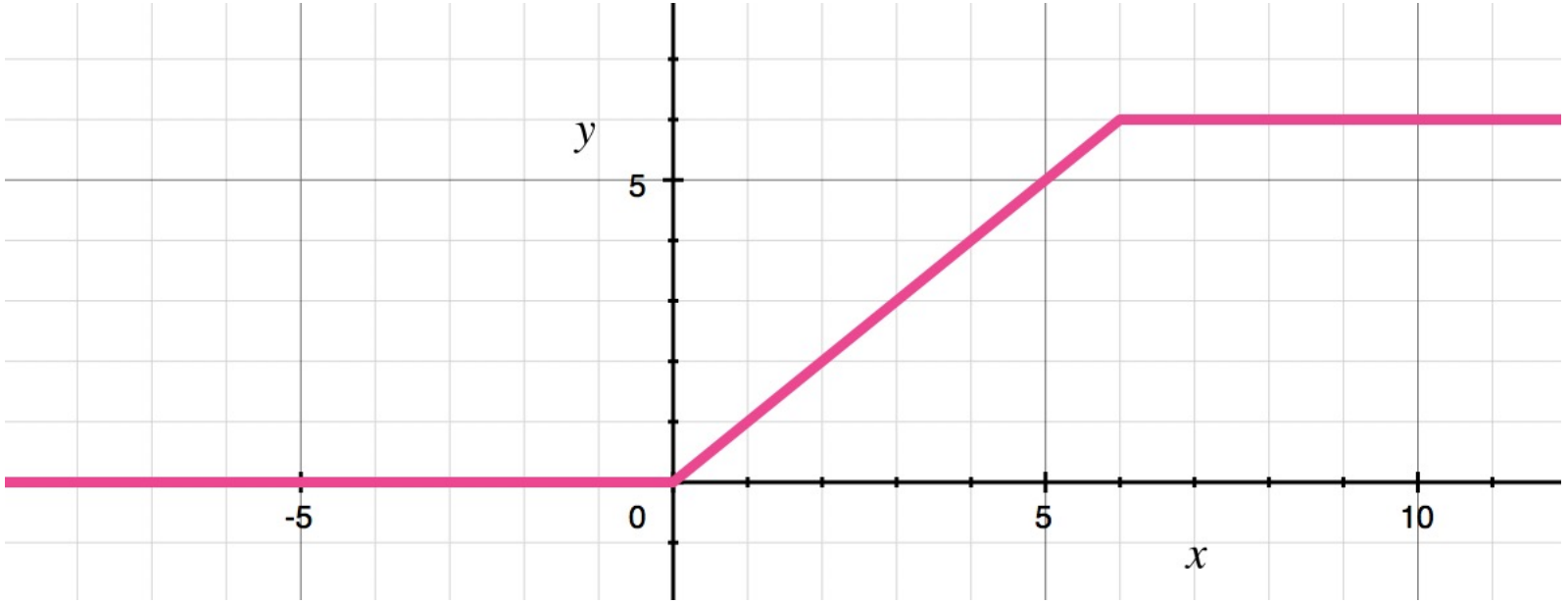
1. 만약 manifold of interest가 ReLU가 적용된 이후 non-zero volume을 가지면 그것들은 linear transformation이다.
2. 입력 manifold가 입력 space의 low-dimensional subspace위에 있다면, ReLU 또한 입력 manifold 위에서의 모든 정보를 보존할 수 잇다.

이 두가지 성질로부터 힌트를 얻었습니다. 만약 manifold of interest가 low-dimensional이라면, convoltion block에 linear bottleneck 레이어를 넣어 이를 알아채게 할 수 있고, 실험 결과 많은 정보의 손실에서 비선형성을 유지하기 위해서는 linear 레이어가 필수적이었습니다.

3. MobileNet V2

Linear Bottlenecks

```
y = min(max(0, x), 6)
```



http://machinethink.net/blog/mobilenet-v2/

3. MobileNet V2

Inverted residuals

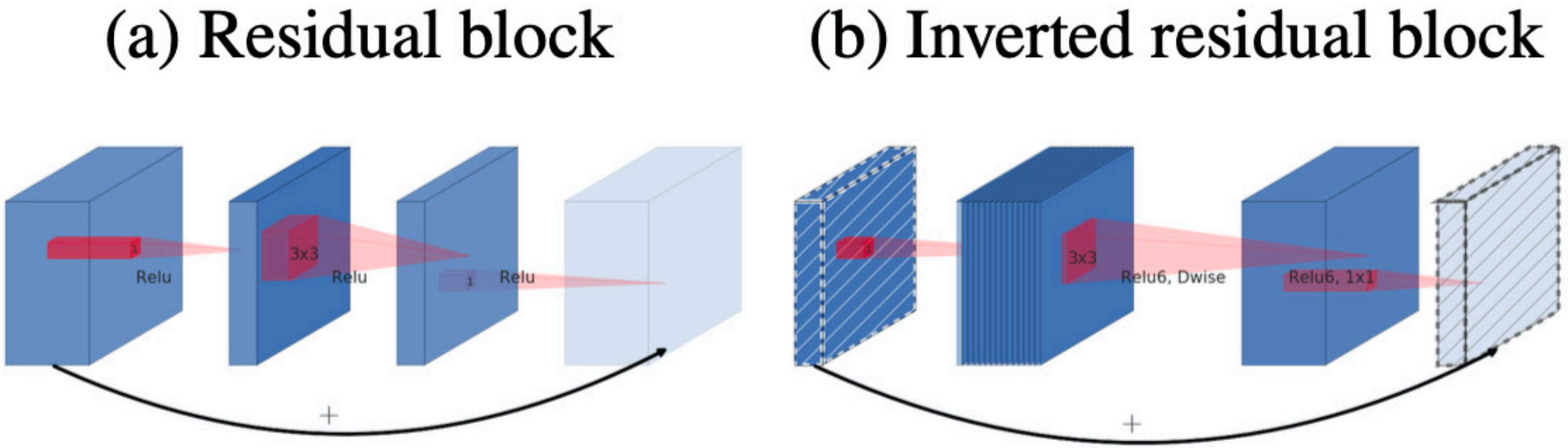
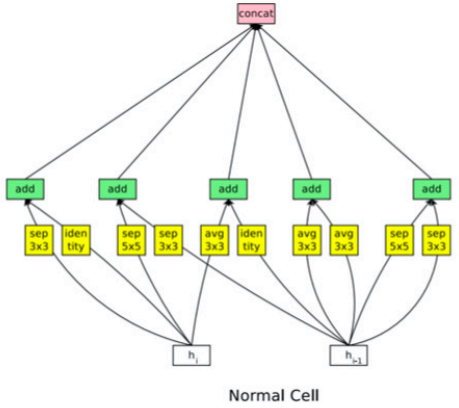


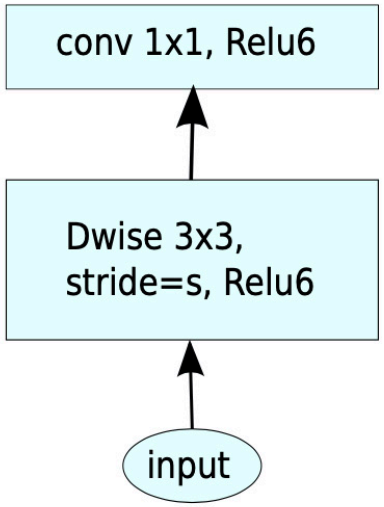
Figure 3: The difference between residual block [8, 30]

3. MobileNet V2

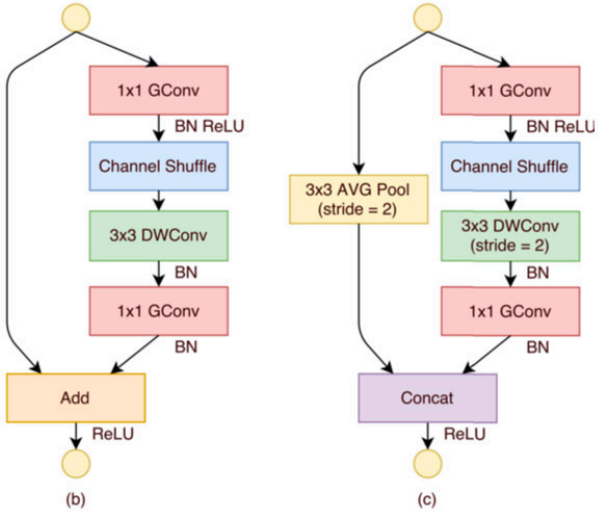
Inverted residuals



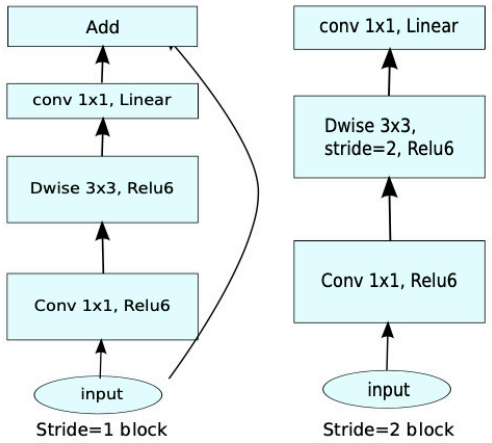
(a) NasNet[23]



(b) MobileNet[27]



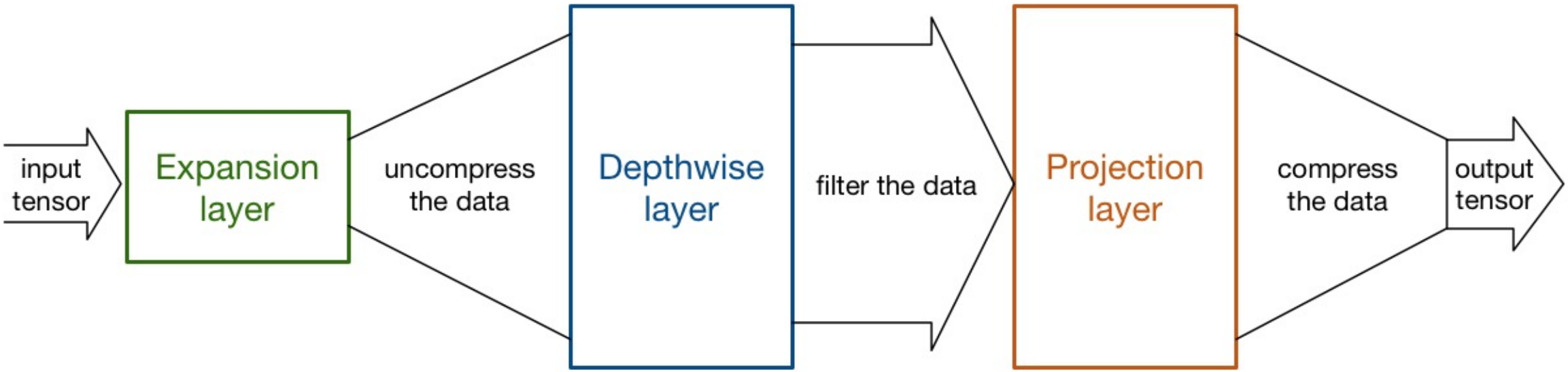
(c) ShuffleNet [20]



(d) Mobilenet V2

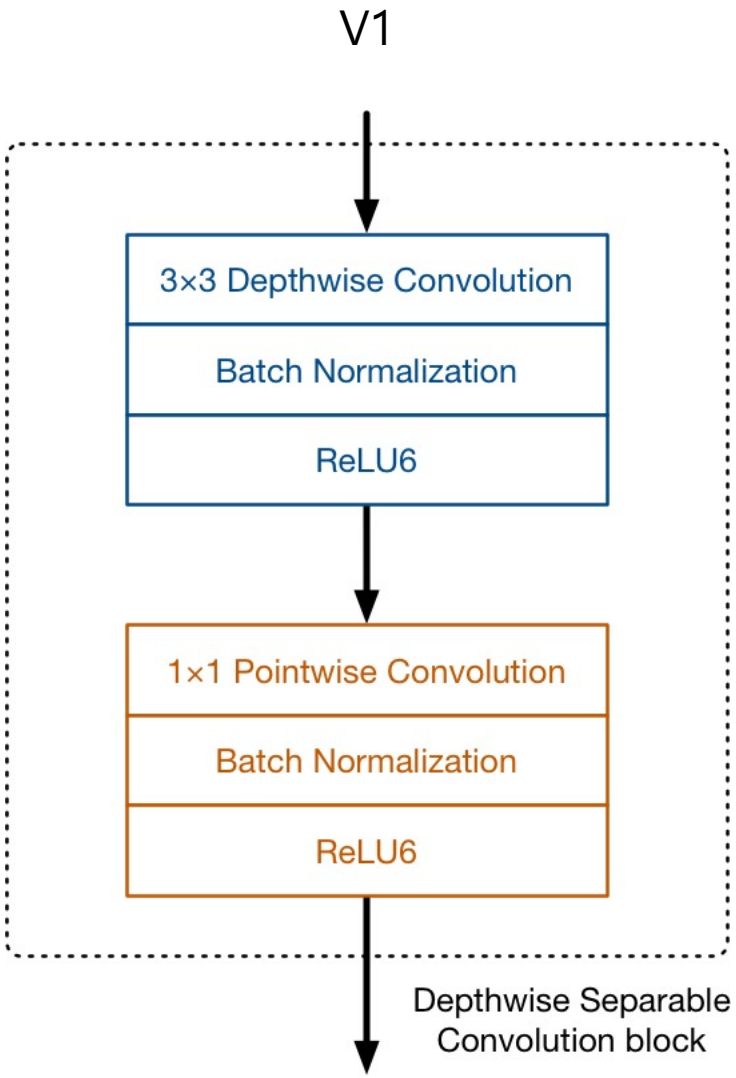
3. MobileNet V2

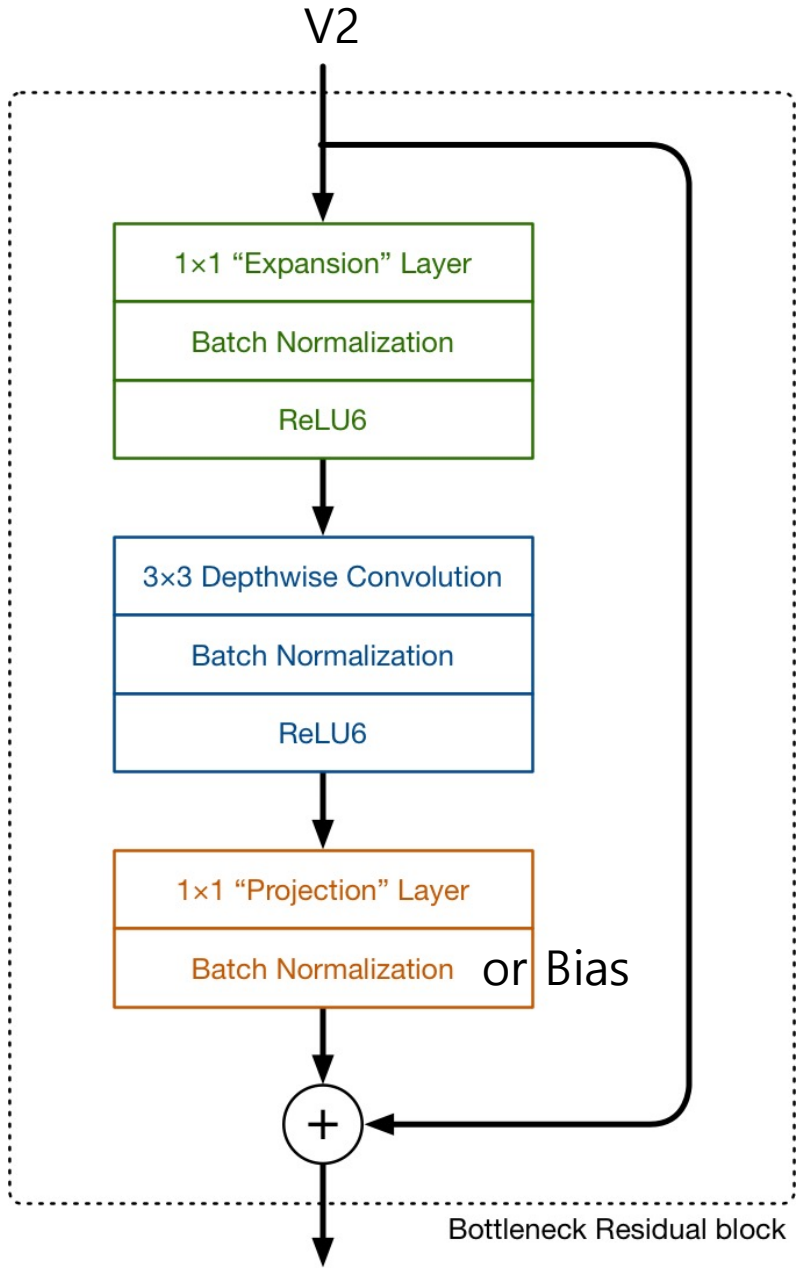
Inverted residuals



http://machinethink.net/blog/mobilenet-v2/

3. MobileNet V2





http://machinethink.net/blog/mobilenet-v2/

3. MobileNet V2

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

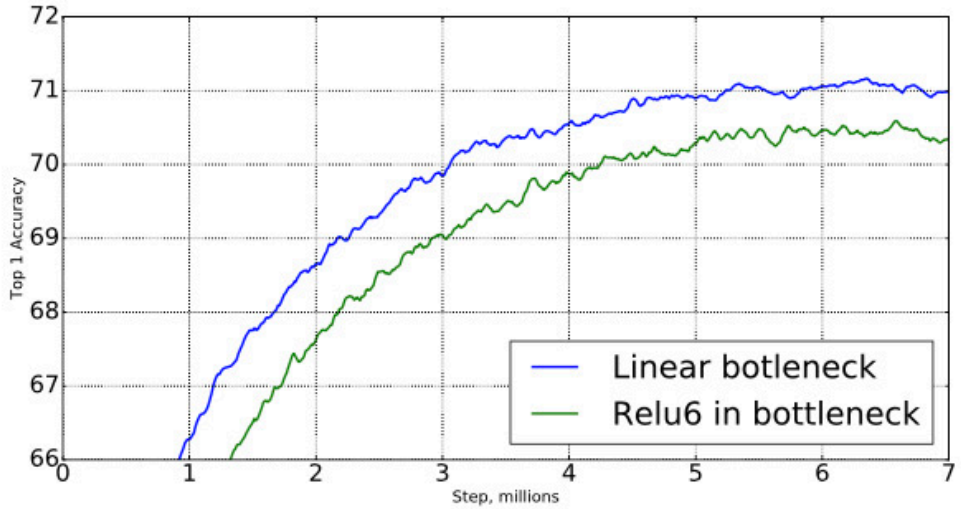
Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

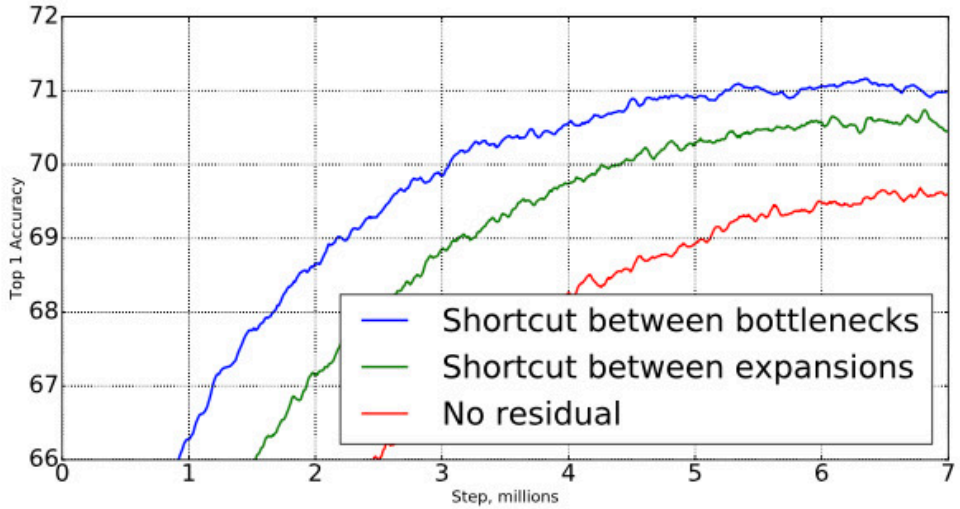
3. MobileNet V2

Size	MobileNetV1	MobileNetV2	ShuffleNet (2x,g=3)
112x112	64/1600	16/400	32/800
56x56	128/800	32/200	48/300
28x28	256/400	64/100	400/600K
14x14	512/200	160/62	800/310
7x7	1024/199	320/32	1600/156
1x1	1024/2	1280/2	1600/3
max	1600K	**400K**	600K

Table 3: The max number of channels/memory (in Kb) that needs to be materialized at each spatial res-

3. MobileNet V2





(a) Impact of non-linearity in the bottleneck layer.

(b) Impact of variations in residual blocks.

Figure 6: The impact of non-linearities and various types of shortcut (residual) connections.

3. MobileNet V2

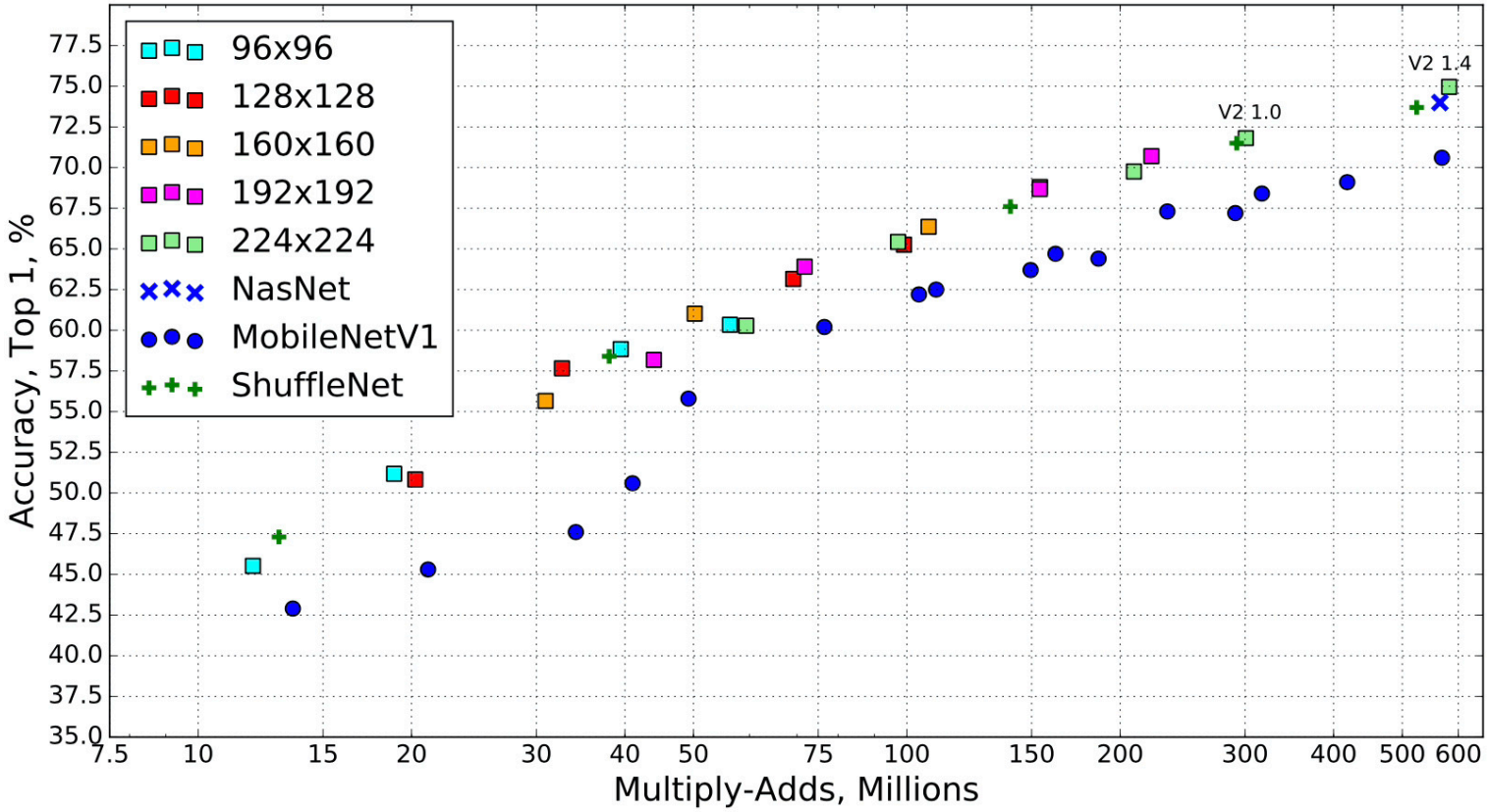


Figure 5: Performance curve of MobileNetV2 vs MobileNetV1, ShuffleNet, NAS. For our networks we use multipliers 0.35, 0.5, 0.75, 1.0 for all resolutions, and additional 1.4 for for 224. Best viewed in color.

3. MobileNet V2

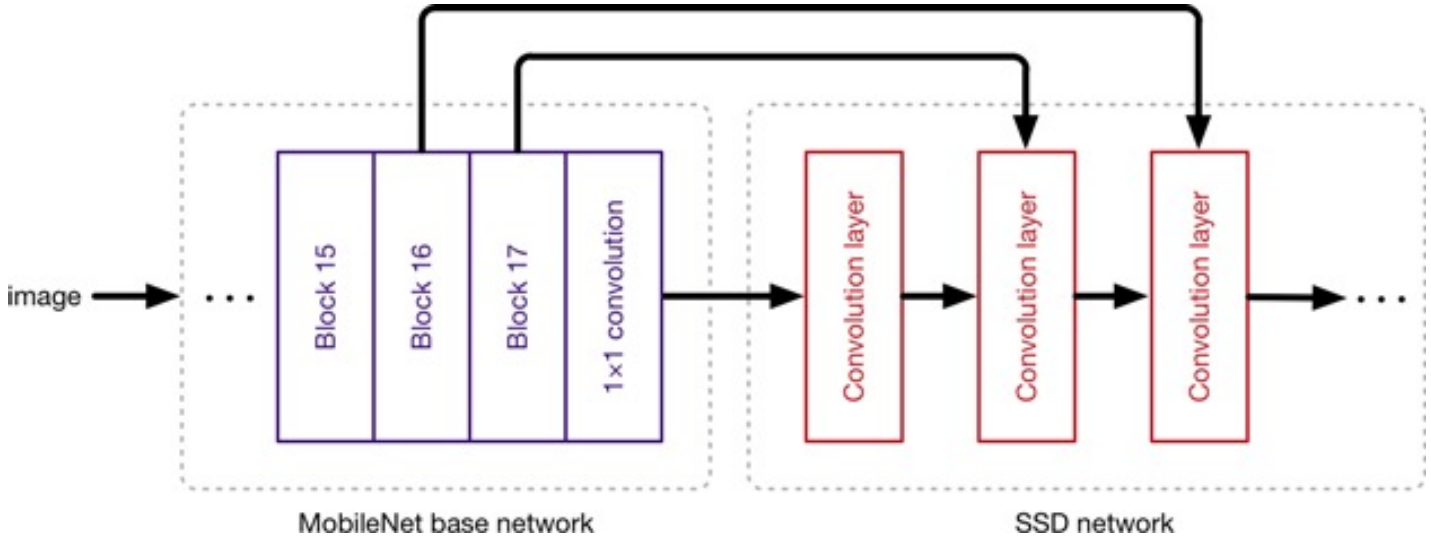
Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Table 4: Performance on ImageNet, comparison for different networks. As is common practice for ops, we count the total number of Multiply-Adds. In the last column we report running time in milliseconds (ms) for a single large core of the Google Pixel 1 phone (using TF-Lite). We do not report ShuffleNet numbers as efficient group convolutions and shuffling are not yet supported.

3. MobileNet V2

	Params	MAdds
SSD[34]	14.8M	1.25B
SSDLite	**2.1M**	**0.35B**

Table 5: Comparison of the size and the computational cost between SSD and SSDLite configured with MobileNetV2 and making predictions for 80 classes.



https://machinethink.net/blog/mobilenet-v2/

3. MobileNet V2

Network	mAP	Params	MAdd	CPU
SSD300[34]	23.2	36.1M	35.2B	-
SSD512[34]	26.8	36.1M	99.5B	-
YOLOv2[35]	21.6	50.7M	17.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
MNet V2 + SSDLite	22.1	**4.3M**	**0.8B**	200ms

Table 6: Performance comparison of MobileNetV2 + SSDLite and other realtime detectors on the COCO dataset object detection task. MobileNetV2 + SSDLite achieves competitive accuracy with significantly fewer parameters and smaller computational complexity. All

4. Conclusion