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

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 effectiveness of MobileNets across a wide range of applications and use cases including object detection, finegrain classification, face attributes and large scale geo-localization.

1. Introduction

Convolutional neural networks have become ubiquitous in computer vision ever since AlexNet [19] popularized deep convolutional neural networks by winning the ImageNet Challenge: ILSVRC 2012 [24]. The general trend has been to make deeper and more complicated networks in order to achieve higher accuracy [27, 31, 29, 8]. However, these advances to improve accuracy are not necessarily making networks more efficient with respect to size and speed. In many real world applications such as robotics, self-driving car and augmented reality, the recognition tasks need to be carried out in a timely fashion on a computationally limited platform.

This paper describes an efficient network architecture and a set of two hyper-parameters in order to build very small, low latency models that can be easily matched to the design requirements for mobile and embedded vision applications. Section 2 reviews prior work in building small models. Section 3 describes the MobileNet architecture and two hyper-parameters width multiplier and resolution multiplier to define smaller and more efficient MobileNets. Section 4 describes experiments on ImageNet as well a variety of different applications and use cases. Section 5 closes with a summary and conclusion.

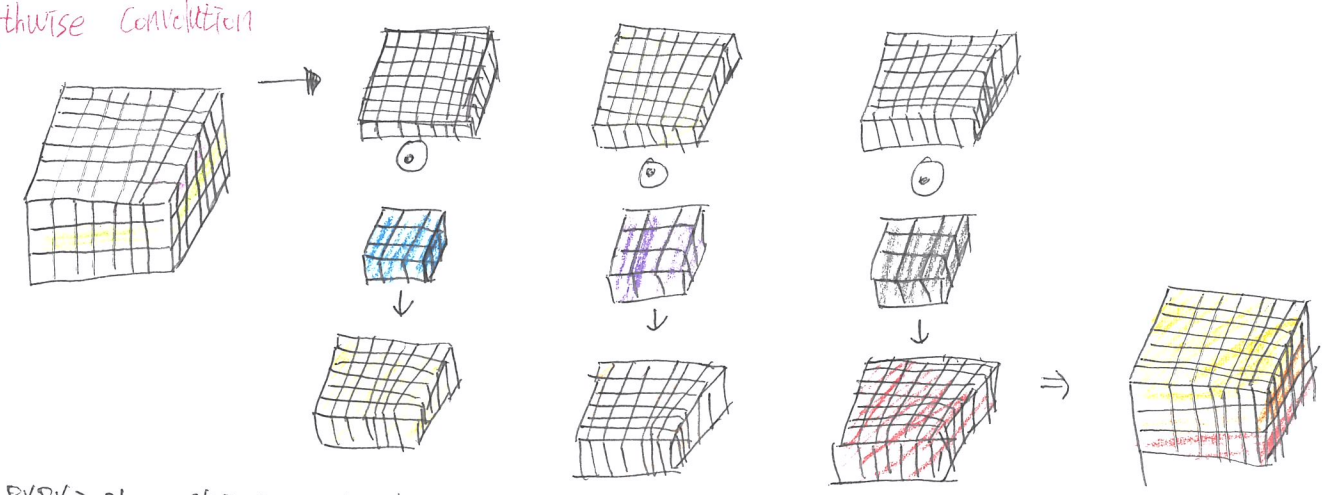
2. Prior Work

There has been rising interest in building small and efficient neural networks in the recent literature, e.g. [16, 34, 12, 36, 22]. Many different approaches can be generally categorized into either compressing pretrained networks or training small networks directly. This paper proposes a class of network architectures that allows a model developer to specifically choose a small network that matches the resource restrictions (latency, size) for their application. MobileNets primarily focus on optimizing for latency but also yield small networks. Many papers on small networks focus only on size but do not consider speed.

MobileNets are built primarily from depthwise separable convolutions initially introduced in [26] and subsequently used in Inception models [13] to reduce the computation in the first few layers. Flattened networks [16] build a network out of fully factorized convolutions and showed the potential of extremely factorized networks. Independent of this current paper, Factorized Networks[34] introduces a similar factorized convolution as well as the use of topological connections. Subsequently, the Xception network [3] demonstrated how to scale up depthwise separable filters to out perform Inception V3 networks. Another small network is Squeezenet [12] which uses a bottleneck approach to design a very small network. Other reduced computation networks include structured transform networks [28] and deep fried convnets [37].

A different approach for obtaining small networks is shrinking, factorizing or compressing pretrained networks. Compression based on product quantization [36], hashing

* Depthwise Convolution

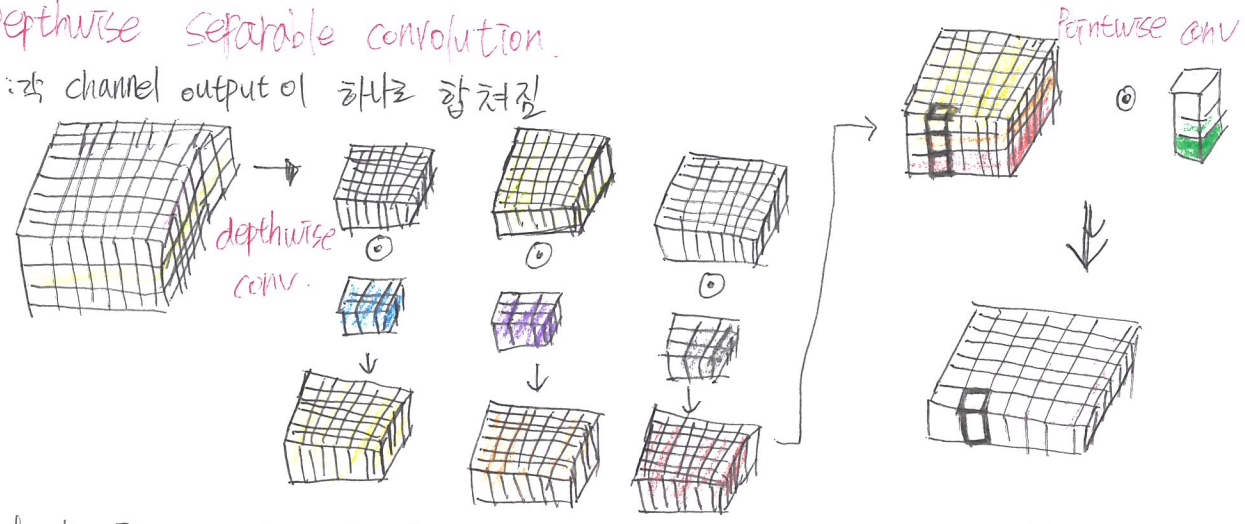


⇒ 8X8X3의 matrix를 depthwise convolution 하기 위해 3X3X3 의 커널 사용.
이때, 3X3X3 커널은 각 채널별로 분리되어 3X3의 2차원 커널을 8X8 크기의 각 분리된 matrix에 붙여 각 convolution 진행하고 다시 합쳐지는 구조.

: 채널 방향의 convolution은 하지 않고 공간 방향의 convolution만 진행.

* Depthwise separable convolution

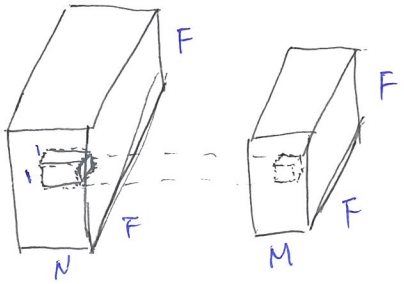
: 각 channel output이 하나로 합쳐짐



depth wise convolution을 진행한 결과물에 각 채널을 1개의 채널로 압축할 수 있는 추가 conv. 진행

* Pointwise convolution (1X1 conv)

: depthwise conv. depthwise separable conv는 공간 방향의 conv 진행 후, channel 방향에 conv. 진행함/하지X
Pointwise conv. 는 공간 방향의 conv는 진행 X. 채널방향의 conv. 진행



1X1XC의 크기 kernel을 사용해 input의 특징을 1개의 채널로 압축시키는 효과 O
input의 채널을 압축시키는 효과 → 연산속도 ↑
압축되면서 없어지는 데이터는 연산속도와 trade-off 관계. 어느정도 조정

* distillation (증류)
: 미리 잘 학습된 큰 네트워크 (teacher Network)의
지식을 실제로 사용하고자 하는 작은 네트워크 (student network)
에게 전달하는 것.

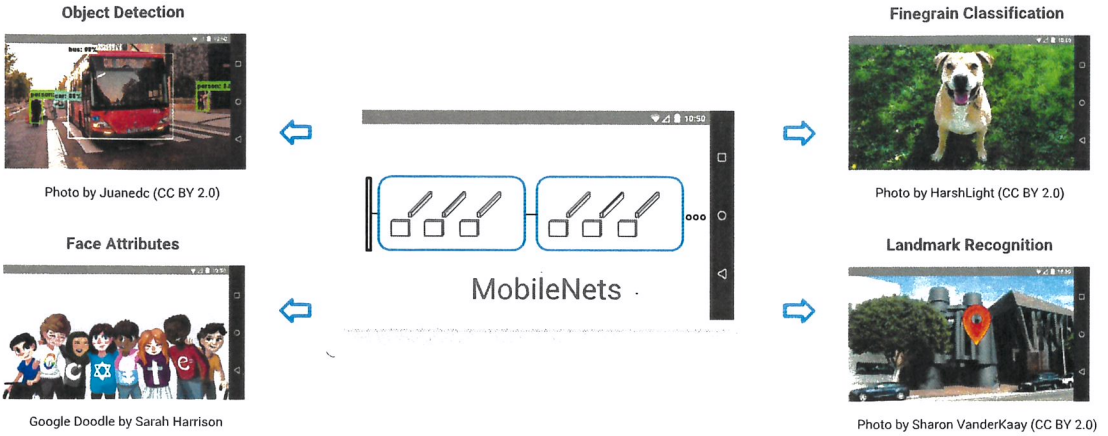


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

small network 연구
1. shrinking
2. factorizing
3. compressing
4. distillation
Speed up

[2], and pruning, vector quantization and Huffman coding [5] have been proposed in the literature. Additionally various factorizations have been proposed to speed up pre-trained networks [14, 20]. Another method for training small networks is distillation [9] which uses a larger network to teach a smaller network. It is complementary to our approach and is covered in some of our use cases in section 4. Another emerging approach is low bit networks [4, 22, 11].

3. MobileNet Architecture

core layer: depthwise separable filters 기반

In this section we first describe the core layers that MobileNet is built on which are depthwise separable filters. We then describe the MobileNet network structure and conclude with descriptions of the two model shrinking hyper-parameters width multiplier and resolution multiplier.

3.1. Depthwise Separable Convolution

The MobileNet model is based on depthwise separable convolutions which is a form of factorized convolutions which factorize a standard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution. For MobileNets the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs the depthwise convolution. A standard convolution both filters and combines inputs into a new set of outputs in one step. The depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size. Figure 2 shows how a standard convolution 2(a) is factorized into a depthwise convolution 2(b) and a 1×1 pointwise convolution 2(c).

A standard convolutional layer takes as input a $D_F \times D_F \times M$ feature map $\mathbf{F}$ and produces a $D_F \times D_F \times N$ feature map $\mathbf{G}$ where D_F is the spatial width and height of a square input feature map[1] M is the number of input channels (input depth), D_G is the spatial width and height of a square output feature map and N is the number of output channel (output depth).

The standard convolutional layer is parameterized by convolution kernel $\mathbf{K}$ of size $D_K \times D_K \times M \times N$ where D_K is the spatial dimension of the kernel (assumed to be square) and M is number of input channels and N is the number of output channels as defined previously.

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

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

Standard convolutions have the computational cost of:

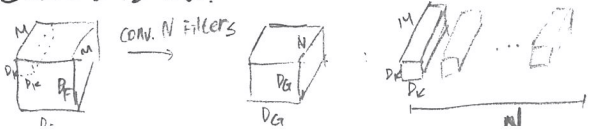
$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \quad (2)$$

where the computational cost depends multiplicatively on the number of input channels M, the number of output channels N the kernel size $D_k \times D_k$ and the feature map size $D_F \times D_F$. MobileNet models address each of these terms and their interactions. First it uses depthwise separable convolutions to break the interaction between the number of output channels and the size of the kernel.

The standard convolution operation has the effect of filtering features based on the convolutional kernels and combining features in order to produce a new representation. The filtering and combination steps can be split into two steps via the use of factorized convolutions called depthwise

[1] We assume that the output feature map has the same spatial dimensions as the input and both feature maps are square. Our model shrinking results generalize to feature maps with arbitrary sizes and aspect ratios.

depthwise separable conv: factorized conv. 형태
→ Standard conv를 depthwise conv와 pointwise conv로 분리.
* depthwise conv: 각 채널에 하나의 필터 적용
* pointwise conv: 1×1 conv 적용. ∵ depth wise conv 출력값 결합 위해.
* Standard conv: filter와 출력의 새로운 세트의 입력을 결합. (한번에!)



separable convolutions for substantial reduction in computational cost.

Depthwise separable convolution are made up of two layers: depthwise convolutions and pointwise convolutions. We use depthwise convolutions to apply a single filter per each input channel (input depth). Pointwise convolution, a simple 1×1 convolution, is then used to create a linear combination of the output of the depthwise layer. MobileNets use both batchnorm and ReLU nonlinearities for both layers.

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

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

where $\hat{\mathbf{K}}$ is the depthwise convolutional kernel of size $D_K \times D_K \times M$ where the m_{th} filter in $\hat{\mathbf{K}}$ is applied to the m_{th} channel in $\mathbf{F}$ to produce the m_{th} channel of the filtered output feature map $\hat{\mathbf{G}}$.

Depthwise convolution has a computational cost of:

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

Depthwise convolution is extremely efficient relative to standard convolution. However it only filters input channels, it does not combine them to create new features. So an additional layer that computes a linear combination of the output of depthwise convolution via 1×1 convolution is needed in order to generate these new features.

The combination of depthwise convolution and 1×1 (pointwise) convolution is called depthwise separable convolution which was originally introduced in [26].

Depthwise separable convolutions cost:

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

which is the sum of the depthwise and 1×1 pointwise convolutions.

By expressing convolution as a two step process of filtering and combining we get a reduction in computation of:

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

MobileNet uses 3×3 depthwise separable convolutions which uses between 8 to 9 times less computation than standard convolutions at only a small reduction in accuracy as seen in Section 4.

Additional factorization in spatial dimension such as in [16, 31] does not save much additional computation as very little computation is spent in depthwise convolutions.

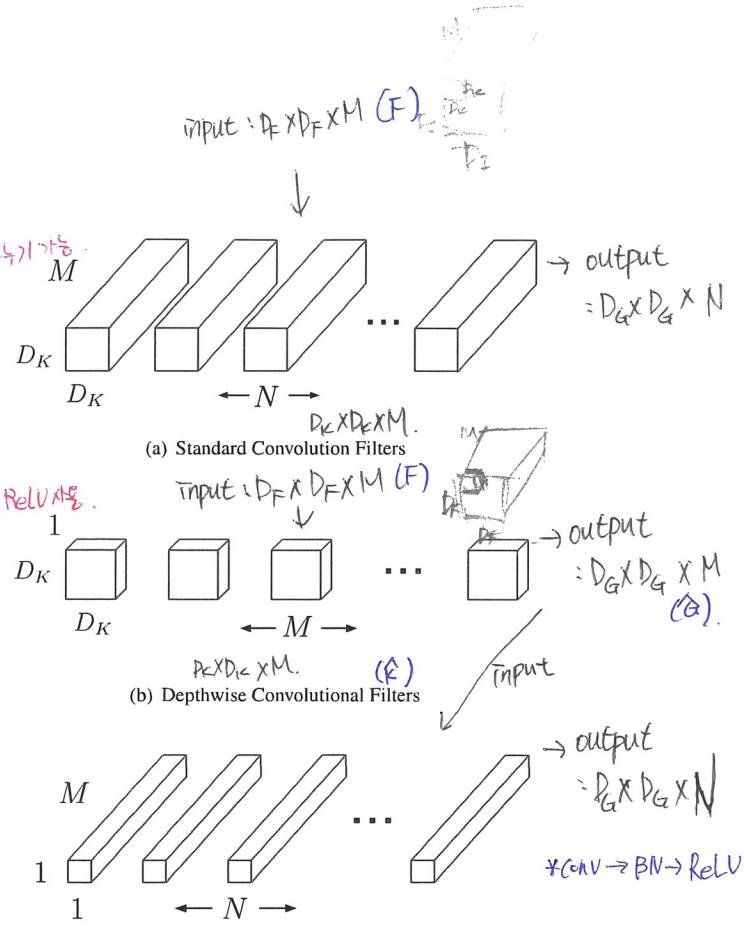


Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

3.2. Network Structure and Training

The MobileNet structure is built on depthwise separable convolutions as mentioned in the previous section except for the first layer which is a full convolution. By defining the network in such simple terms we are able to easily explore network topologies to find a good network. The MobileNet architecture is defined in Table 1. All layers are followed by a batchnorm [13] and ReLU nonlinearity with the exception of the final fully connected layer which has no nonlinearity and feeds into a softmax layer for classification. Figure 3 contrasts a layer with regular convolutions, batchnorm and ReLU nonlinearity to the factorized layer with depthwise convolution, 1×1 pointwise convolution as well as batchnorm and ReLU after each convolutional layer. Down sampling is handled with strided convolution in the depthwise convolutions as well as in the first layer. A final average pooling reduces the spatial resolution to 1 before the fully connected layer. Counting depthwise and pointwise convolutions as separate layers, MobileNet has 28 layers.

It is not enough to simply define networks in terms of a small number of Mult-Adds. It is also important to make sure these operations can be efficiently implementable. For

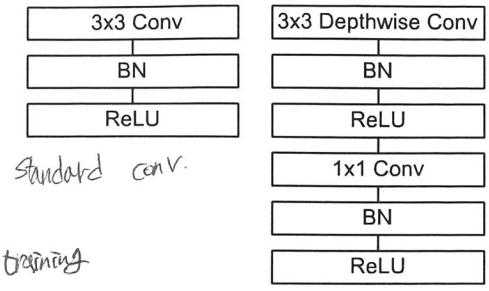


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

instance unstructured sparse matrix operations are not typically faster than dense matrix operations until a very high level of sparsity. Our model structure puts nearly all of the computation into dense 1×1 convolutions. This can be implemented with highly optimized general matrix multiply (GEMM) functions. Often convolutions are implemented by a GEMM but require an initial reordering in memory called im2col in order to map it to a GEMM. For instance, this approach is used in the popular Caffe package [15]. 1×1 convolutions do not require this reordering in memory and can be implemented directly with GEMM which is one of the most optimized numerical linear algebra algorithms. MobileNet spends 95% of it's computation time in 1×1 convolutions which also has 75% of the parameters as can be seen in Table 2. Nearly all of the additional parameters are in the fully connected layer.

MobileNet models were trained in TensorFlow [1] using RMSprop [33] with asynchronous gradient descent similar to Inception V3 [31]. However, contrary to training large models we use less regularization and data augmentation techniques because small models have less trouble with overfitting. When training MobileNets we do not use side heads or label smoothing and additionally reduce the amount image of distortions by limiting the size of small crops that are used in large Inception training [31]. Additionally, we found that it was important to put very little or no weight decay (l2 regularization) on the depthwise filters since their are so few parameters in them. For the ImageNet benchmarks in the next section all models were trained with same training parameters regardless of the size of the model.

3.3. Width Multiplier: Thinner Models

Although the base MobileNet architecture is already small and low latency, many times a specific use case or application may require the model to be smaller and faster. In order to construct these smaller and less computationally expensive models we introduce a very simple parameter α called width multiplier. The role of the width multiplier α is to thin a network uniformly at each layer. For a given layer and width multiplier α the number of input channels M becomes αM and the number of output channels N becomes αN.

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

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

where $\alpha \in (0, 1]$ with typical settings of 1, 0.75, 0.5 and 0.25. $\alpha = 1$ is the baseline MobileNet and $\alpha < 1$ are reduced MobileNets. Width multiplier has the effect of reducing computational cost and the number of parameters quadratically by roughly α^2. Width multiplier can be applied to any model structure to define a new smaller model with a reasonable accuracy, latency and size trade off. It is used to define a new reduced structure that needs to be trained from scratch.

3.4. Resolution Multiplier: Reduced Representation

The second hyper-parameter to reduce the computational cost of a neural network is a resolution multiplier ρ. We ap-

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%

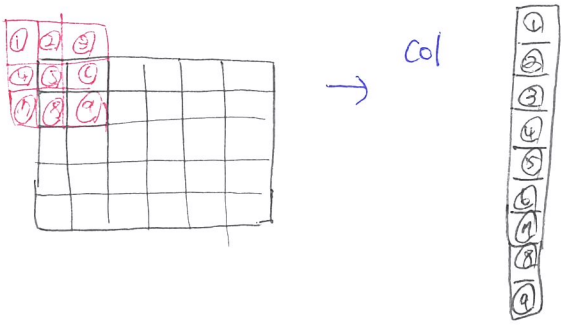
*im2 col

: CNN은 1989년 고안. but 범용화 X. (Why? conv은 특정 size의 window를 입력 이미지 각 pixel 마다 size의 제곱 개씩 만큼 곱셈 연산 수행 → 대부분의 Feature Map
→학습해야할 parameter 적음, 연산시간 ↓↓.)

빠르게 conv 연산 위해 이미지 데이터를 conv하기 좋은 연속된 데이터로 변환 (im2 col)

ex) 3x3 window 사용해 conv 수행.

source image → col



: 입력데이터 양에 커지게 됨.
Conv. 연산 빠른 대신 연산 메모리↑

*RMSProp

: AdaGrad는 최소값에 도달하기 전에 학습률을 0에 수렴하게 만들 수 O.
AdaGrad가 간단한 convex function에서는 잘 작동 but, 복잡한 다차원 곡면함수를 잘 탈색 X.
기울기의 단순한 누적만으로 충분 X.

⇒ RMSProp : 기울기 단순 누적 X. 지수 가중 이동 평균 (Exponentially weighted moving average) 사용해 최신 기울기들이 더 크게 반영.

$$h_i \leftarrow \rho h_{i-1} + (1-\rho)\frac{\partial L_i}{\partial w} \odot \frac{\partial L_i}{\partial w}$$

이전 AdaGrad의 h에 새로운 hyper parameter ρ 추가.
h가 무한히 커지지 X. ρ가 작을수록 가장 최신의 기울기 더 크게 반영.

*분산 병렬 딥러닝 구조

동기식 (synchronous) : 상대적으로 빠른 worker들이 느린 worker의 학습 진도에 맞춰 학습. (진행속도가 parameter server로부터 항상 일관된 최신 global-parameter 계산. 다음 단계 학습 진행)

비동기식 (Asynchronous) : worker들이 독립적 학습 진행. parameter server로부터 동일 X global parameter 제공받아 다음의 학습과정 진행.

⇒ 동기식 : 학습 정확도 보장. 속도 느림.
비동기식 : 학습속도 빠르지만 정확도 보장 X.

*label smoothing

: 이전 분류 문제에서 label은 0 또는 1.
But, 반드시 그렇지 않을 수 있음
why? 사람이 annotation 하기 때문에 실수 가능성 존재
(label smoothing X) 이때 발생할 수 있는 잘못된 loss의 영향을 줄이기 위하여 label을 0 또는 1이 아니라 smooth하게 부여.

→ label smoothing는 mislabeling 된 데이터 고려위해 사용
추가적으로 regularization 에 도움
model generalization, calibration (수정) 에 도움

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
$\rho = 0.714$	15.1	0.15

ply this to the input image and the internal representation of every layer is subsequently reduced by the same multiplier. In practice we implicitly set ρ by setting the input resolution.

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:

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

where $\rho \in (0, 1]$ which is typically set implicitly so that the input resolution of the network is 224, 192, 160 or 128. $\rho = 1$ is the baseline MobileNet and $\rho < 1$ are reduced computation MobileNets. Resolution multiplier has the effect of reducing computational cost by ρ^2.

As an example we can look at a typical layer in MobileNet and see how depthwise separable convolutions, width multiplier and resolution multiplier reduce the cost and parameters. Table 3 shows the computation and number of parameters for a layer as architecture shrinking methods are sequentially applied to the layer. The first row shows the Mult-Adds and parameters for a full convolutional layer with an input feature map of size $14 \times 14 \times 512$ with a kernel K of size $3 \times 3 \times 512 \times 512$. We will look in detail in the next section at the trade offs between resources and accuracy.

4. Experiments

In this section we first investigate the effects of depthwise convolutions as well as the choice of shrinking by reducing the width of the network rather than the number of layers. We then show the trade offs of reducing the network based on the two hyper-parameters: width multiplier and resolution multiplier and compare results to a number of popular models. We then investigate MobileNets applied to a number of different applications.

4.1. Model Choices

First we show results for MobileNet with depthwise separable convolutions compared to a model built with full convolutions. In Table 4 we see that using depthwise separable convolutions compared to full convolutions only reduces

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

Table 6. MobileNet Width Multiplier (α)

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 (ρ)

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

accuracy by 1% on ImageNet was saving tremendously on mult-adds and parameters.

We next show results comparing thinner models with width multiplier to shallower models using less layers. To make MobileNet shallower, the 5 layers of separable filters with feature size $14 \times 14 \times 512$ in Table 1 are removed. Table 5 shows that at similar computation and number of parameters, that making MobileNets thinner is 3% better than making them shallower.

4.2. Model Shrinking Hyperparameters

Table 6 shows the accuracy, computation and size trade offs of shrinking the MobileNet architecture with the width multiplier α. Accuracy drops off smoothly until the architecture is made too small at $\alpha = 0.25$.

Table 7 shows the accuracy, computation and size trade offs for different resolution multipliers by training MobileNets with reduced input resolutions. Accuracy drops off smoothly across resolution.

Figure 4 shows the trade off between ImageNet Accuracy and computation for the 16 models made from the cross product of width multiplier $\alpha \in \{1, 0.75, 0.5, 0.25\}$ and resolutions $\{224, 192, 160, 128\}$. Results are log linear with a jump when models get very small at $\alpha = 0.25$.

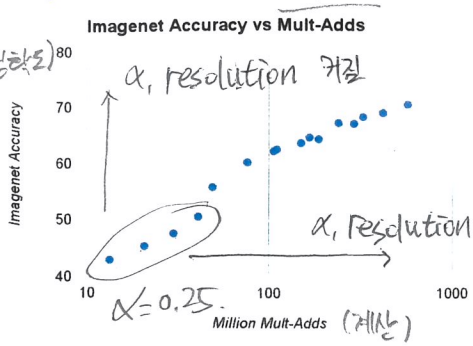


Figure 4. This figure shows the trade off between computation (Mult-Adds) and accuracy on the ImageNet benchmark. Note the log linear dependence between accuracy and computation.

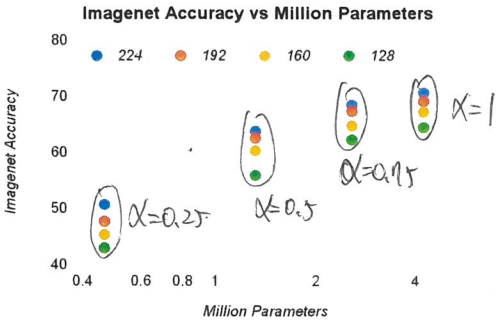


Figure 5. This figure shows the trade off between the number of parameters and accuracy on the ImageNet benchmark. The colors encode input resolutions. The number of parameters do not vary based on the input resolution.

Figure 5 shows the trade off between ImageNet Accuracy and number of parameters for the 16 models made from the cross product of width multiplier $\alpha \in \{1, 0.75, 0.5, 0.25\}$ and resolutions $\{224, 192, 160, 128\}$.

Table 8 compares full MobileNet to the original GoogleNet [30] and VGG16 [27]. MobileNet is nearly as accurate as VGG16 while being 32 times smaller and 27 times less compute intensive. It is more accurate than GoogleNet while being smaller and more than 2.5 times less computation.

Table 9 compares a reduced MobileNet with width multiplier $\alpha = 0.5$ and reduced resolution 160×160. Reduced MobileNet is 4% better than AlexNet [19] while being $45\times$ smaller and $9.4\times$ less compute than AlexNet. It is also 4% better than Squeezenet [12] at about the same size and $22\times$ less computation.

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

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

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%

4.3. Fine Grained Recognition

We train MobileNet for fine grained recognition on the Stanford Dogs dataset [17]. We extend the approach of [18] and collect an even larger but noisy training set than [18] from the web. We use the noisy web data to pretrain a fine grained dog recognition model and then fine tune the model on the Stanford Dogs training set. Results on Stanford Dogs test set are in Table 10. MobileNet can almost achieve the state of the art results from [18] at greatly reduced computation and size.

4.4. Large Scale Geolocalizaton

PlaNet [35] casts the task of determining where on earth a photo was taken as a classification problem. The approach divides the earth into a grid of geographic cells that serve as the target classes and trains a convolutional neural network

* Pre training

: 선행학습, 사전 훈련, 전처리 과정

Multi layered Perception (MLP)에서 weight와 BIAS를 잘 초기화 시키는 방법

⇒ 효과적으로 layer 쌓아 여러개의 hidden layer도 효율적 훈련하는 목적.

또한, unsupervised learning 이 가능함 → fine-tuning

(물론 가중치, 편향 초기화 끝나면 레이블 된 데이터로 supervised learning 해야함)

레이블 X 인 큰 데이터 넣어 훈련 시킬 수 있다.

Drop out, Mini-batch 방식으로 Pre-traing 생략 가능.

* Fine-tuning

: 기존에 학습되어져 있는 모델 기반으로 아키텍처를 새로운 목적 (내 이미지 데이터에 맞게) 변형하고 이미 학습된 모델 weight로부터 학습 업데이트 하는 방법.

- 모델의 파라미터 미세하게 조절하는 행위

: 특히 DL에서 이미 존재하는 모델에 추가 데이터를 투입해 파라미터 업데이트

⇒ "정교한" "파라미터" 튜닝.

ex) 고양이, 개 분류기 VGG16 모델.

1000개의 카테고리 학습 → 고양이, 개 2개의 카테고리만 필요.

(모든 레이어 쓸 필요 X)

↓

데이터 를 해당 모델로 예측해 bottleneck feature 만 뽑아내고, 이를 이용하여 어파인 레이어 (fully-connected layer) 만 학습 시켜 사용하는 방법

→ 파인 튜닝 X 피처 추출 레이어 파라미터 업데이트 안함.

파인튜닝 → 기존에 학습이 된 레이어 내 데이터를 추가로 학습시켜 파라미터 업데.

단, 정교 해야함.

완전히 랜덤한 초기 파라미터 쓰거나 가장 아래 레이어의 (일반적인 피처 학습한 덜 추상화된 레이어)

파라미터 학습시 오버피팅 or 전체 파라미터 망가질.

on millions of geo-tagged photos. PlaNet has been shown to successfully localize a large variety of photos and to outperform Im2GPS [6, 7] that addresses the same task.

We re-train PlaNet using the MobileNet architecture on the same data. While the full PlaNet model based on the Inception V3 architecture [31] has 52 million parameters and 5.74 billion mult-adds. The MobileNet model has only 13 million parameters with the usual 3 million for the body and 10 million for the final layer and 0.58 Million mult-adds. As shown in Tab. 11, the MobileNet version delivers only slightly decreased performance compared to PlaNet despite being much more compact. Moreover, it still outperforms Im2GPS by a large margin.

4.5. Face Attributes

Another use-case for MobileNet is compressing large systems with unknown or esoteric training procedures. In a face attribute classification task, we demonstrate a synergistic relationship between MobileNet and distillation [9], a knowledge transfer technique for deep networks. We seek to reduce a large face attribute classifier with 75 million parameters and 1600 million Mult-Adds. The classifier is trained on a multi-attribute dataset similar to YFCC100M [32].

We distill a face attribute classifier using the MobileNet architecture. Distillation [9] works by training the classifier to emulate the outputs of a larger model[2] instead of the ground-truth labels, hence enabling training from large (and potentially infinite) unlabeled datasets. Marrying the scalability of distillation training and the parsimonious parameterization of MobileNet, the end system not only requires no regularization (e.g. weight-decay and early-stopping), but also demonstrates enhanced performances. It is evident from Tab. 12 that the MobileNet-based classifier is resilient to aggressive model shrinking: it achieves a similar mean average precision across attributes (mean AP) as the in-house while consuming only 1% the Multi-Adds.

4.6. Object Detection

MobileNet can also be deployed as an effective base network in modern object detection systems. We report results for MobileNet trained for object detection on COCO data based on the recent work that won the 2016 COCO challenge [10]. In table 13, MobileNet is compared to VGG and Inception V2 [13] under both Faster-RCNN [23] and SSD [21] framework. In our experiments, SSD is evaluated with 300 input resolution (SSD 300) and Faster-RCNN is compared with both 300 and 600 input resolution (Faster-RCNN 300, Faster-RCNN 600). The Faster-RCNN model evaluates 300 RPN proposal boxes per image. The models are trained on COCO train+val excluding 8k minival images and evaluated on minival. For both frameworks, MobileNet achieves comparable results to other networks with only a fraction of computational complexity and model size.

[2]The emulation quality is measured by averaging the per-attribute cross-entropy over all attributes.

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

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.

4.7. Face Embeddings

The FaceNet model is a state of the art face recognition model [25]. It builds face embeddings based on the triplet loss. To build a mobile FaceNet model we use distillation to train by minimizing the squared differences of the output

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

of FaceNet and MobileNet on the training data. Results for very small MobileNet models can be found in table 14.

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.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*, 1, 2015. 4

[2] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *CoRR, abs/1504.04788*, 2015. 2

[3] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357v2*, 2016. 1

[4] M. Courbariaux, J.-P. David, and Y. Bengio. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014. 2

[5] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015. 2

[6] J. Hays and A. Efros. IM2GPS: estimating geographic information from a single image. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2008. 7

[7] J. Hays and A. Efros. Large-Scale Image Geolocalization. In J. Choi and G. Friedland, editors, *Multimodal Location Estimation of Videos and Images*. Springer, 2014. 6, 7

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 1

[9] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2, 7

[10] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv preprint arXiv:1611.10012*, 2016. 7

[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016. 2

[12] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 1, 6

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 1, 3, 7

[14] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 2

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 4

[16] J. Jin, A. Dundar, and E. Culurciello. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014. 1, 3

[17] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011. 6

[18] J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev, T. Duerig, J. Philbin, and L. Fei-Fei. The unreasonable effectiveness of noisy data for fine-grained recognition. *arXiv preprint arXiv:1511.06789*, 2015. 6

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 6

[20] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 2

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. Ssd: Single shot multibox detector. *arXiv preprint arXiv:1512.02325*, 2015. 7

[22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016. 1, 2

[23] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 7