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 depthwise separable convolutions to build light weight deep neural networks. We introduce two simple global hyperparameters that efficiently trade off between latency and accuracy. These hyperparameters 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 outperform 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
[2], and pruning, vector quantization and Huffman coding [5] have been proposed in the literature. Additionally various factorizations have been proposed to speed up pretrained 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

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 hyperparameters 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 $1 \times 1$ convolution called a pointwise convolution. For MobileNets the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a $1 \times 1$ 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 $1 \times 1$ pointwise convolution 2(c).

A standard convolutional layer takes as input a $D_F \times M$ feature map $F$ and produces a $D_F \times D_F \times N$ feature map $G$ 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 $K$ 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:

$$G_{k,l,n} = \sum_{i,j,m} K_{k,i,j,m,n} \cdot F_{k+i-1,l+j-1,m}$$

Standard convolutions have the computational cost of:

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

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 separable convolutions.

\(^1\)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.
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 $1 \times 1$ 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:

$$G_{k,l,m} = \sum_{i,j} K_{i,j,m} \cdot F_{k+i-1,l+j-1,m}$$

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

Depthwise convolution has a computational cost of:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$$

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 $1 \times 1$ convolution is needed in order to generate these new features.

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

Depthwise separable convolutions cost:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

which is the sum of the depthwise and $1 \times 1$ 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 $3 \times 3$ 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.

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, $1 \times 1$ 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
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 $1 \times 1$ 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]. $1 \times 1$ 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 its computation time in $1 \times 1$ 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 $\alpha$ called width multiplier. The role of the width multiplier $\alpha$ is to thin a network uniformly at each layer. For a given layer

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

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 $\alpha^2$. 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 $\rho$. We ap-
In practice we implicitly set every layer is subsequently reduced by the same multiplier. Apply this to the input image and the internal representation of every layer is subsequently reduced by the same multiplier. In practice we implicitly set \( \rho \) 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 \( \alpha \) and resolution multiplier \( \rho \):

\[
D_K \cdot D_K \cdot \alpha M \cdot \rho F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F \cdot (7)
\]

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 \( \rho^2 \).

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 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 \( \alpha \). 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 \( \alpha \in \{0.25, 0.5, 0.75, 1.0\} \) 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 on the ImageNet benchmark. The colors encode input resolutions. The number of parameters do not vary based on the input resolution.

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 $160 \times 160$. 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 10. MobileNet for Stanford Dogs

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 Accuracy</th>
<th>Million Mult-Adds</th>
<th>Million Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception V3 [18]</td>
<td>84%</td>
<td>5000</td>
<td>23.2</td>
</tr>
<tr>
<td>1.0 MobileNet-224</td>
<td>83.3%</td>
<td>569</td>
<td>3.3</td>
</tr>
<tr>
<td>0.75 MobileNet-224</td>
<td>81.9%</td>
<td>325</td>
<td>1.9</td>
</tr>
<tr>
<td>1.0 MobileNet-192</td>
<td>81.9%</td>
<td>418</td>
<td>3.3</td>
</tr>
<tr>
<td>0.75 MobileNet-192</td>
<td>80.5%</td>
<td>239</td>
<td>1.9</td>
</tr>
</tbody>
</table>

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 Geolocalization

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
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.

### 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

---

\(^2\)The emulation quality is measured by averaging the per-attribute cross-entropy over all attributes.
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


Table 14. MobileNet Distilled from FaceNet

<table>
<thead>
<tr>
<th>Model</th>
<th>1e-4 Accuracy</th>
<th>Million Mult-Adds</th>
<th>Million Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>FaceNet [25]</td>
<td>83%</td>
<td>1600</td>
<td>7.5</td>
</tr>
<tr>
<td>1.0 MobileNet-160</td>
<td>79.4%</td>
<td>286</td>
<td>4.9</td>
</tr>
<tr>
<td>1.0 MobileNet-128</td>
<td>78.3%</td>
<td>185</td>
<td>5.5</td>
</tr>
<tr>
<td>0.75 MobileNet-128</td>
<td>75.2%</td>
<td>166</td>
<td>3.4</td>
</tr>
<tr>
<td>0.75 MobileNet-128</td>
<td>72.5%</td>
<td>108</td>
<td>3.8</td>
</tr>
</tbody>
</table>


