Disclaimer: This paper is a work of fiction written from the perspective of a 2020 researcher traveling back in time to late 2012 to share some 2020 network design, training and implementation ideas; references to credit the actual inventors of the various ideas is provided at the end

# MobileNet: Creating Faster Networks using Depthwise Separable Convolutions

Anshul Pardhi
anshul.pardhi@utdallas.edu
Dec 31, 2012

## Abstract

MobileNets[1] are discussed in detail, providing efficient means for image classification. Depthwise separable convolutions are introduced, which is the backbone of this network. These depthwise separable layers help in reducing the compute time to a large extent as compared to the previously built convolutional neural networks. Data movement and compute time is also calculated, to get the predicted performance time.

## 1 Introduction

Neural networks have been developed for a long time now. They are extremely efficient in predicting newer outputs based on the trained data. But much of the prediction done till now has been on the textual data. Applying neural networks on images (and videos) has been a hot research topic in the recent years. Various methodologies have been developed and far newer methods are being developed every year to perform various operations on images, such as image classification, object detection, to name a few.

Like the two sides of a coin, applying neural networks on images have their drawbacks as well. The biggest (and the most challenging) limitation is the size of the input. Images are represented as a set of matrices and the various computations involved are not that efficient. There is generally a trade-off for the size of the input image and the time it takes to classify the image. We need to classify good resolution images in the least practically possible time but not many methods have been developed to overcome this trade-off.

This paper demonstrates a better neural network design to reduce the above-mentioned trade-off. Section 2 describes the prior work done in this domain in the past. Section 3 describes the MobileNet design and architecture. Section 4 describes the results obtained by training MobileNet on CIFAR-10[2] dataset. Section 5 describes the performance prediction strategy and the various computations required by the network. Section 6 closes with the conclusion of the network.

## 2 Related Work

### 2.1 Design

The design for MobileNet is inspired from the historic LeNet[3], which was the very first proper convolutional neural network, providing accurate results on a small MNIST dataset. Earlier this year, the development of AlexNet[4], the winner of the ImageNet challenge: ILSVRC 2012[5], provided a new motivation to develop far efficient networks. This network was based on serial design, where there are a sequential set of operations from input to output. It helped reduce the top-1 and top-5 error rates significantly during ILSVRC 2012, however the network contained far more parameters, due to the fully connected layers, making the computations significantly slower. Like AlexNet, MobileNet is also based on sequential design, but it uses far lesser parameters.

### 2.2 Training

In order to come up with the optimum parameters or weights, the most common technique used is gradient descent. Various optimizations have been made to the traditional gradient descent over time, such as batch gradient descent and stochastic gradient descent (SGD) [6]. A combination of the prior two, mini batch SGD has become quite popular recently. It uses small batches of fixed number of training examples as input and continuously apply the gradient descent on these small batches. The average of all the examples within a batch

is taken as the final value contributing to the weight update. The technique, however has its limitations, the most prevalent being choosing the optimum batch size.

### 2.3 Implementation

Initially, neural networks were trained on CPUs (Central Processing Units). They are typically multi-core (having 2 to 10 cores in general), but they seem too slow while training deep neural networks and have been mostly replaced by GPUs (Graphics Processing Units). They contain much more cores, which are efficient in operating in parallel. They perform far better than CPUs, but are expensive and have their compute limitations as well. Most modern deep learning systems are a mix of CPUs and GPUs where GPU does most of the training, while CPU is responsible for loading the data to and from the memory.

## 3 Design

The basic building blocks of the encoder (body) design of MobileNet are depthwise separable convolutions. These are composed of depthwise convolution and pointwise convolution. The depthwise convolution applies a single filter to each input channel. The pointwise convolution applies a 1X1 convolution and then adds the result with the result of the depthwise convolution to obtain the net result. Depthwise separable convolution drastically reduces the computation and model size.

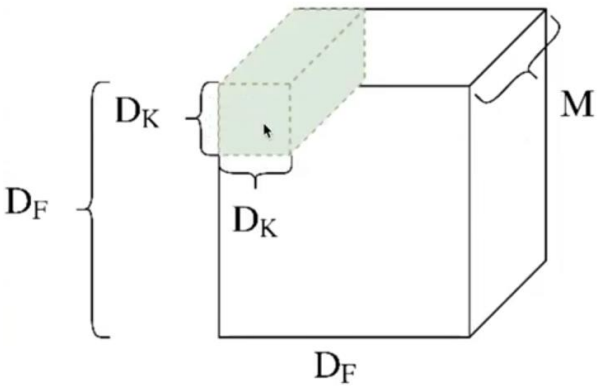


Figure 1. Input feature map

Suppose the input feature map has dimensions $D_F$ x $D_F$ x M, as shown in Figure 1. Let this input feature map produce the output feature map with dimensions $D_F$ x $D_F$ x N. $D_F$ is the spatial dimension of the input and output feature maps. Appropriate padding is performed to make sure the input and output feature maps' height and width dimensions are the same. M is the number of input channels while N is the number of output channels.

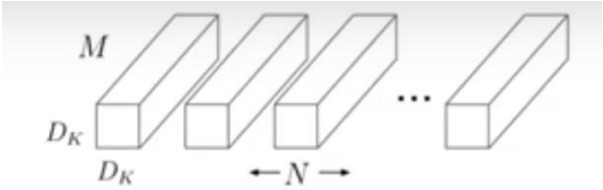


Figure 2. Standard convolution filters

Figure 2 shows the dimensions of the standard convolution filters, $D_K$ x $D_K$ x M x N, where $D_K$ is the spatial dimension of the filter. The computational cost of standard convolution is

$$D_K.D_K.M.N.D_F.D_F$$

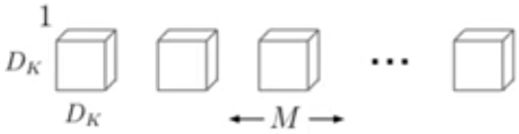


Figure 3. Depthwise convolution filters

Figure 3 shows the dimensions of the depthwise convolution filters, $D_K$ x $D_K$ x 1 x N, where $D_K$ is the spatial dimension of the filter.

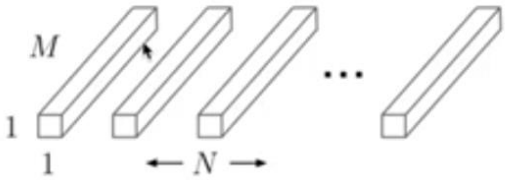


Figure 4. Pointwise convolution filters

Figure 4 shows the dimensions of the pointwise convolution filters, 1 x 1 x M x N.

Depthwise separable convolution consists of the addition of depthwise as well as pointwise convolution filters. So, the computational cost of depthwise separable convolution is

$$D_K.D_K.M.D_F.D_F + M.N.D_F.D_F$$

Comparing the computational costs of depthwise separable and standard convolutions, we get

$$\frac{D_K.D_K.M.D_F.D_F + M.N.D_F.D_F}{\mathrm{D_K}.D_K.M.N.D_F.D_F}$$

i.e., $\frac{1}{N} + \frac{1}{D_K 2}$

Thus, there is a drastic decrease in the computational cost when using depthwise separable convolution over standard convolution.

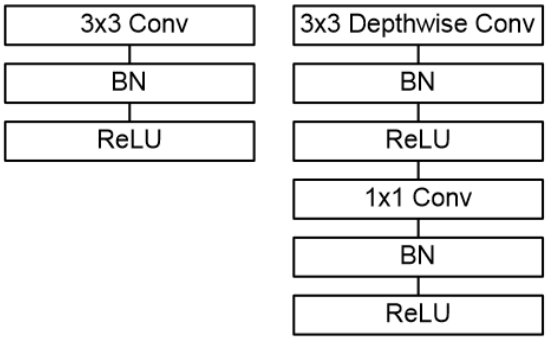


Figure 5. Left: Standard Convolution Layer; Right: Depthwise Separable Convolution Layer

Figure 5 shows the basic architecture of the standard and depthwise separable convolution layers, consisting of the convolution, followed by batch normalization[7] and ReLU[8] nonlinearity.

| Type / Stride | Filter Shape | Input Size |
|---|---|---|
| Conv / s2 | 3 x 3 x 3 x 32 | 28 x 28 x 3 |
| Conv dw / s1 | 3 x 3 x 32 dw | 14 x 14 x 32 |
| Conv / s1 | 1 x 1 x 32 x 64 | 14 x 14 x 32 |
| Conv dw / s1 | 3 x 3 x 64 dw | 14 x 14 x 64 |
| Conv / s1 | 1 x 1 x 64 x 128 | 14 x 14 x 64 |
| Conv dw / s1 | 3 x 3 x 128 dw | 14 x 14 x 128 |
| Conv / s1 | 1 x 1 x 128 x 256 | 14 x 14 x 128 |
| Conv dw / s1 | 3 x 3 x 256 dw | 14 x 14 x 256 |
| Conv / s1 | 1 x 1 x 256 x 512 | 14 x 14 x 256 |
| 5 x Conv dw / s1 | 3 x 3 x 512 dw | 14 x 14 x 512 |
| 5 x Conv / s1 | 1 x 1 x 512 x 512 | 14 x 14 x 512 |
| Conv dw / s1 | 3 x 3 x 512 dw | 14 x 14 x 512 |
| Conv / s1 | 1 x 1 x 512 x 1024 | 14 x 14 x 512 |
| Conv dw / s2 | 3 x 3 x 1024 dw | 14 x 14 x 1024 |
| Conv / s1 | 1 x 1 x 1024 x 1024 | 7 x 7 x 1024 |
| Avg Pool / s1 | Pool 7 x 7 | 7 x 7 x 1024 |
| FC / s1 | 1024 x 10 | 1 x 1 x 1024 |
| Softmax / s1 | Classifier | 1 x 1 x 10 |

Table 1: MobileNet Architecture

As shown in Table 1, the MobileNet body (encoder) consists of a standard convolution layer, followed by 11 layers of depthwise separable convolutions. For the head (decoder) part, this is followed by a global average pool. A fully connected dense layer is then added to the network. A softmax function is applied to this layer to generate results among the 10 classes of CIFAR-10 dataset.

## 4 Training

MobileNet is trained on CIFAR-10 dataset. The dataset consists of 60000 32 x 32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. For data preprocessing, firstly, the input image is normalized by subtracting the mean and dividing by standard deviation. The image is then randomly flipped horizontally from left to right. Then, a random crop is applied to the image to convert it from 32 x 32 to 28 x 28.

After the input images are preprocessed, they are sent to the MobileNet body, where firstly a standard convolution layer is applied, followed by 11 layers of depthwise separable convolutions. Inside the depthwise separable convolutions, after applying the convolution, batch normalization, followed by ReLU nonlinearity is applied, as shown in the right chart of Figure 5. In batch normalization, the outputs are scaled by subtracting mean and dividing my standard deviation for fixed sized batches. It allows each layer of a network to learn by itself a little bit more independently of other layers. ReLU (Rectified Linear Unit) provides a non-linearity to the network by making all negative values zero and passing all positive values unchanged.

A softmax classifier classifies the outputs into the 10 classes of CIFAR-10. Cross Entropy loss is used to calculate the loss over the 60 epochs on which the network is trained. Furthermore, RMSProp is used for updating the weights.

A hyperparameter, depth multiplier was used, which factors the number of input channels (M) and number of output channels (N) by a factor α. It helps in faster computation, but reduces the accuracy obtained. Different experiments were performed using different values of this depth multiplier, and the results are specified in Table 2.

| Depth Multiplier | Training Accuracy | Validation Accuracy |
|---|---|---|
| 1 | 92.50% | 84.52% |
| 0.75 | 90.82% | 80.69% |

Table 2: Results of Experiments Performed

Since CIFAR-10 is not that big of a dataset, having depth multiplier to reduce accuracy while not reducing much of time does not seem apt. As a result, depth multiplier=1 is chosen and the curves showing training and validation accuracy and training and validation loss are plotted. The curves are shown in Figure 6.

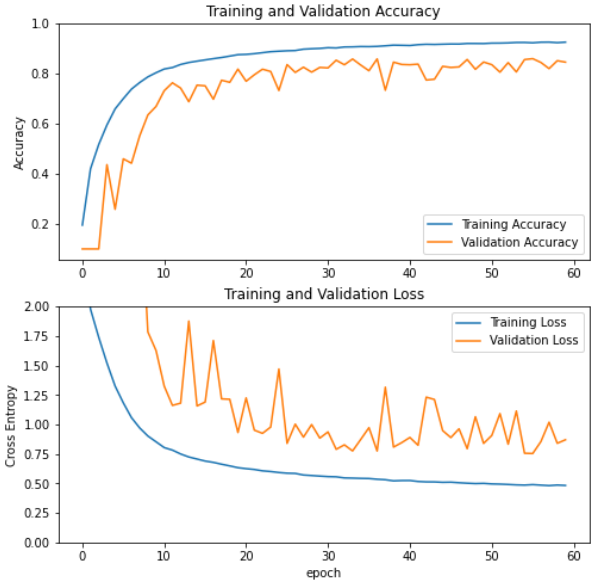


Figure 6. Training & Validation Accuracy & Loss Curves

The final validation accuracy thus obtained is 84.52%.

## 5 Implementation

The input feature map is initially stored in external memory. It is then moved to the internal memory, where the computations are performed. For every layer as shown in Table 1, the filter coefficients are brought to the internal memory from the external memory via the DDR bus, and are moved back to the external memory once the computations are performed. The interaction between internal and external memories, using the system bus is illustrated in Figure 7.

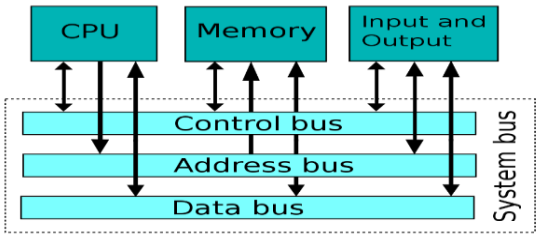


Figure 7. Interaction between Internal & External Memory Using Bus

Firstly, the input and output feature map computations are calculated by multiplying the underlying dimensions of the feature maps. Total internal memory is 1 MB, which is equivalent to 8388608 bits. Total internal memory in use is computed by adding the current layer's output feature map computation size to the earlier result. It is made sure that the total internal memory in use never exceeds 1 MB. Table 3 shows the sizes of the input and output feature map as well as the total internal memory in use in bits.

| Layer | Input Feature Map Size | Output Feature Map Size | Total Internal Memory in Use |
|---|---|---|---|
| Conv / s2 | 18816 | 50176 | 68992 |
| Conv dw / s1 | 6272 | 6272 | 75264 |
| Conv / s1 | 6272 | 12544 | 87808 |
| Conv dw / s1 | 12544 | 12544 | 100352 |
| Conv / s1 | 12544 | 25088 | 125440 |
| Conv dw / s1 | 25088 | 25088 | 150528 |
| Conv / s1 | 25088 | 50176 | 200704 |
| Conv dw / s1 | 50176 | 50176 | 250880 |
| Conv / s1 | 50176 | 100352 | 351232 |
| 5 x Conv dw / s1 | 100352 | 100352 | 451584 |
| 5 x Conv / s1 | 100352 | 100352 | 551936 |
| Conv dw / s1 | 100352 | 100352 | 652288 |
| Conv / s1 | 100352 | 200704 | 852992 |
| Conv dw / s2 | 200704 | 50176 | 903168 |
| Conv / s1 | 50176 | 50176 | 953344 |
| FC / s1 | 1024 | 10 | 953354 |

Table 3: Memory Computations

Every time total data movement consists of the total bits moved when moving the filter coefficients from the external memory to internal memory and back to external memory. Data movement time in seconds for each layer is calculated by dividing this total data movement by 1 GB, as we have 1 GB/s DDR bus. Data movement computations are shown in Table 4.

| Layer | Total Data Movement | Data Movement Time |
|---|---|---|
| Conv / s2 | 2352 | 0.000002352 |
| Conv dw / s1 | 576 | 0.000000072 |
| Conv / s1 | 4096 | 0.000000512 |
| Conv dw / s1 | 1152 | 0.000000144 |
| Conv / s1 | 16384 | 0.000002048 |
| Conv dw / s1 | 2304 | 0.000000288 |
| Conv / s1 | 65536 | 0.000008192 |
| Conv dw / s1 | 4608 | 0.000000576 |
| Conv / s1 | 262144 | 0.000032768 |
| 5 x Conv dw / s1 | 9216 | 0.000001152 |
| 5 x Conv / s1 | 524288 | 0.000065536 |
| Conv dw / s1 | 9216 | 0.000001152 |
| Conv / s1 | 1048576 | 0.000131072 |
| Conv dw / s2 | 18432 | 0.000002304 |
| Conv / s1 | 2097152 | 0.000262144 |
| FC / s1 | 20480 | 0.00000256 |

Table 4: Data Movement Time

MACs are computed and compute time is calculated by dividing twice the MACs by 1 TFLOPS. All other operations are divided by 10 GFLOPS. Predicted performance time is then calculated by adding the data movement time and the compute time. The calculated results are shown in Table 5.

| Layer | MACs | Compute Time | Predicted Performance Time |
|---|---|---|---|
| Conv / s2 | 677376 | 0.000001354 | 0.000003706 |
| Conv dw / s1 | 1806336 | 3.61267E-06 | 3.68467E-06 |
| Conv / s1 | 401408 | 8.02816E-07 | 1.31482E-06 |
| Conv dw / s1 | 7225344 | 1.44507E-05 | 1.45947E-05 |
| Conv / s1 | 1605632 | 3.21126E-06 | 5.25926E-06 |
| Conv dw / s1 | 28901376 | 5.78028E-05 | 5.80908E-05 |
| Conv / s1 | 6422528 | 1.28451E-05 | 2.10371E-05 |
| Conv dw / s1 | 115605504 | 0.000231211 | 0.000231787 |
| Conv / s1 | 25690112 | 5.13802E-05 | 8.41482E-05 |
| 5 x Conv dw / s1 | 462422016 | 0.000924844 | 0.000925996 |
| 5 x Conv / s1 | 51380224 | 0.00010276 | 0.000168296 |
| Conv dw / s1 | 462422016 | 0.000924844 | 0.000925996 |
| Conv / s1 | 102760448 | 0.000205521 | 0.000336593 |
| Conv dw / s2 | 1849688064 | 0.003699376 | 0.00370168 |
| Conv / s1 | 51380224 | 0.00010276 | 0.000364904 |
| FC / s1 | 10240 | 2.048E-08 | 2.58048E-06 |

Table 5: Compute Time & Predicted Performance Time

From the predicted performance time, it is evident that depthwise convolution takes lesser time than pointwise colvolution because there is more data movement in the case of pointwise convolution as well as it results in lesser compute time.

## 6 Conclusion

There is a big trade-off between the size of the input image and the time it takes to train the neural network. Depthwise separable convolutions introduced in MobileNets seems promising enough to reduce this trade-off. Some important design decisions were made, providing enough number of depthwise separable layers to increase validation accuracy, making sure the network doesn't take that longer time. Depth multiplier is used as a hyperparameter to increase the speed of the network. Data movement and compute times were also calculated to predict the performance time of the network.

## References

Note: As stated above, this paper is a work of fiction. The following are the actual inventors of the ideas described in this paper.

[1] A. Howard, et. al., "MobileNets: Efficient Convolutional Neural Networks for Mobile VisionApplications," arXiv:1704.04861, 2017.

[2] A. Krizhevsky, "Convolutional Deep Belief Networks on CIFAR-10," Unpublished manuscript, 2010.

[3] Y. LeCun, et. al., "Gradient-based Learning Applied to Document Recognition," Proceedingsof the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[4] A. Krizhevsky, et. al., "ImageNet Classification with Deep ConvolutionalNeural Networks," In Advances in neural information processing systems, pages 1097–1105, 2012.

[5] O. Russakovsky, et. al., "ImageNet Large Scale Visual Recognition Challenge" International Journal of Computer Vision, 115(3):211–252, 2012.

[6] L. Bottou, "Large-scale Machine Learning with Stochastic Gradient Descent," inProc. 19th Int. Conf. Comput. Statist., 2010.

[7] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv:1502.03167, 2015.

[8] V. Nair and G. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines", ICML, 2010.

## Code Link

https://colab.research.google.com/drive/1OjMyY1Z7_XayQ62Uzbg7wEQXyNAuZPsH