Understanding and Improving Image Classification methods using MobileNets

Ruchi Singh
rxs180057@utdallas.edu
Dec 31, 2012

Abstract

Recently, Image classification has been evolved as a powerful application in the field of machine learning and computer vision. We present an importance of image classification, its benefits and the limitations of existing methods. We explain about the first two network architectures (LeNet and AlexNet) which has become very popular for image classification applications. Many problems have been solved by using these network architectures but they had some limitations in terms of computational resources, accuracy and complex network size. Considering the limitations of previous networks, we present a class of efficient models called MobileNets for mobile and embedded vision applications. We present extensive experiments on resource and accuracy tradeoffs and show strong performance compared to previous networks. We then discuss the training methods we used and implementation section which includes the performance prediction strategy, training loss and accuracy for MobileNet network architecture.

1 Introduction

Image classification refers to a process in computer vision that can classify an image according to its visual contents. For example, an image classification algorithm may be designed to tell if an image contains a particular figure or not. In other words, Image classification is a process of taking an input (like a picture) and outputting a class (like a "cat") or a probability that input belongs to a particular class ("there's 90% probability that this input is a cat").

It is perhaps the most important part of digital image analysis. It is being used for websites with large visual databases. Using Image Classification algorithms, companies can easily organize and categorize their database because it allows for automatic classification of images in large quantities. Visual recognition on social media is already a fact.

There are various classification methods involves Support Vector Machine, Artificial Neural Networks, Decision Trees etc. Each has its pros and cons when it comes to classify an image. Support Vector Machine delivers unique solution, very efficient than other methods and it avoids overfitting. But on the other hand, it has high complexity algorithm, so it is slower in speed than other. Artificial Neural Networks are robust to noisy training dataset and very efficient for large datasets. But it gives high computational cost and they are lazy learners. Decision Trees are also used for image classification. In case of decision trees, it requires little efforts from users and they are easy to interpret and explain but on the other hand, splits are very sensitive to training dataset and it, sometimes, gives high classification error rates.

In this paper, we describe a neural network design for image classification which focuses on the encoder building block structure and efficient classification decoder. Following the design section, we describe some training methods to train the above described network. And detailed description of 'training data loss' and 'validation data accuracy' curves with different experiments and their results. The implementation section describes the model's performance prediction strategy. It also describes the external memory used, DDR bus, internal memory used, matrix compute and per layer memory location assignment and predicted data movement with compute time on host machine.

2 Related Work

2.1 Design

LeNet-5 (1998) was one of the simplest architectures for image classification. It has 2 convolutional layers and 3

fully-connected layers. The average pooling layer as we know it now was called sub-sampling layer and it had trainable weights. This architecture has about 60,000 parameters. The LeNet architecture was was straight and small, it can even run on the CPU. The image features are distributed across the entire image, and convolutions with learnable parameters are an effective way to extract similar features at multiple locations with few parameters. Therefore, being able to save parameters and computation was key advantage in LeNet. However, this network design was not designed to work on large images. LeNet was designed to work on fixed-size input so the ability to process high resolution images required larger and more convolution layers, this network was constrained by the availability of computing resources.

In 2012, Alex krizhevsky released AlexNet which was a deeper and much wider version of the LeNet . AlexNet scaled the insights of LeNet to a much larger network that could be used to learn much more complex objects object hierarchies. With 60M parameters, lexNet has 8 layers – 5 convolutional and 3 full-connected layers. It was first time that ReLU has been used as activation functions. The use of dropout technique to selectively ignore single neurons during training, a way to avoid overfitting of the model. One interesting non -standard thing is overlapping pooling layers. The use of GPUs NVIDIA GTX 580 to reduce training time. AlexNet suffers from the issues of many free parameters in larger filters, data locality for local response norm and issues of memory in fully connected layers. Real time implementation is implicitly a memory bandwidth test.

2.2 Training

Gradient descent is an optimization algorithm often used for finding the weights or coefficients of machine learning algorithms, such as artificial neural networks and logistic regression.[3] Stochastic gradient descent (SGD) is a variation of gradient descent algorithm that calculates the error and update the model for each example in the training dataset. Batch gradient descent is a variation of the gradient descent algorithm that calculates the error for each example in the training dataset but updates the model after all training examples have been evaluated.

Batch gradient descent can used for smoother curves. SGD can used when the dataset is large. Batch gradient descent converges directly to minima. SGD converges faster for larger datasets. But, since in SGD we use only one example at a time, we cannot implement the vectorized implementation on it. This can slow down the computation. To tackle this problem, we use a mixture of batch gradient descent and SGD which is called mini batch SGD.

In mini batch SGD, neither we use all the dataset at once nor we use single example at a time. We use a batch of fixed number of training examples than the actual dataset and call it a mini-batch. For one epoch, we pick a mini batch, feed it to neural network, calculate the mean gradient of the mini batch and use the calculated mean gradient to update the weights.[6]

However, Just like SGD, the average cost over the epochs in mini batch gradient descent fluctuates because we average a small number of examples at a time which sometimes gives the low final accuracy. It also requires the configuration of an additional 'mini-batch size' hyperparameter for the learning algorithm. Error information must be accumulated across mini batches of training examples like batch gradient descent.

2.3 Implementation

X86 CPUs are central processing units, the ordinary processers we are used to. They are multi-core usually from 2 to 10 cores in modern Core i3-i9 Intel CPUs and extends upto 18 cores in high-end Intel CPUs like i9 on desktop market. On the server market, there are Intel Xeon, usually having more cores such as 56 cores or 64 cores in AMD EPYC 7742. But CPUs are not the current choice for training of deep learning models. In the experiments, there are some attempts to use clusters of CPUs for deep learning, optimizing DL libraries for CPUs, but they didn't look very promising when it came to training of complex models having high complex matrix multiplications.

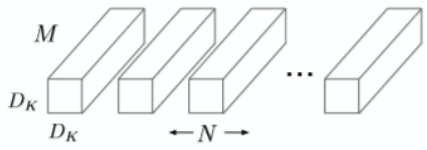
GPUs, Graphics Processing Units, are specialized processors originally created for computer graphics tasks. Modern GPUs contain a lot of processors and are hihgly parallel, which makes them very effective for deep learning models training. x86 GPUs have much more specialized cores (upto 5120 in the latest NVIDIA Volta V100 GPUs), and matrix operations are parallelized much better on GPUs. The modern deep learning systems are a mix of CPU and GPU, where the GPU does the heavy lifting and CPU is responsible for loading the data into/from the memory and orchestrating the calculations.

3 Design

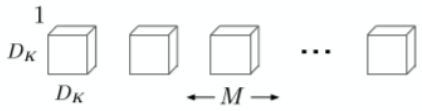
We present a class of efficient models called MobileNets for mobile and embedded vision applications. MobileNets are based on a streamlined architecture that used depth-seperable convolutions to build light weight deep neural networks. In this design, we propose a class of network architectures that allows a model developer to specifically choose a small network that matches the resources restrictions (latency, speed) for their applications.

3.1 Depthwise Seperable Convolutions

The MobileNet model is based on depthwise seperable convolutions which is a form of factorize convolutions which is a combination of a depthwise convolution and 1x1 convolution called a pointwise convolution.[1] The depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1x1 convolution to combine the outputs of depthwise convolution. The depthwise seperable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorizing has the effect of drastically reducing computation and model size. Figure 1 shows how a standard convolution is factorized into a depthwise convolution and a 1x1 pointwise convolution.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

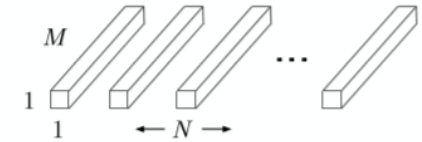


Figure 1. The standard convolutional filters in two layers: (a) a depthwise convolution (b) a pointwise convolution (c) to build a depthwise seperable filter.

A standard convolution layer takes as input a D_f x D_f x M feature map F and produces a D_g x D_g x N feature map G where D_f is the special width and height of a square input feature map. M is the number of input channels D_g is the special width and height of a square output feature map and N is the number of output channel. The standard convolutional layer is parameterized by convolution kernel of size Dk X Dk X M X N where Dk is the spatial dimension of the kernel assumed to be square and M is the number of input channels and N is the number of output channels.

Standard convolutions have the computational cost of:

D_k x D_k x M x N x D_f x D_f

MobileNet model uses depthwise seperable convolutions to break the interaction between the number of output channels and the size of the kernel.

Depthwise seperable convolution is the combination of two layers: depthwise convolution and pointwise convolution. We use depthwise convolution to apply a single filter per each input channel. A simple 1x1 pointwise convolution is used to create a linear combination of the output of depthwise layers. Depthwise convolution is extremely efficient than a standard convolution. However, It only filters input channels, it does not combine them to create new features. So a 1x1 pointwise convolutional layer is used.

Depthwise seperable convolutions cost:

D_k x D_k x M x D_f x D_f + M x N x D_f x D_f

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

(D_k x D_k x M x D_f x D_f + M x N x D_f x D_f)/
(D_k x D_k x M x N x D_f x D_f) = 1/N+1/D^2_k

MobileNet uses 3 x 3 depthwise seperable convolutions which uses between 8 to 9 times less computation than standard convolutions at only a small reduction in accuracy.

3.2 Network Structure

The MobileNet architecture is based on depthwise seperable layers as mentioned in the previous section with an exception of first layer which is a full convolution. The MobileNet architecture (body and head architecture) is defined in Table 1. In the body of the

model, All layers are followed by a batchnorm and ReLU nonlinearity with the exception of the final fully connected layer which has no nonlinearity and feeds into a softmax layer for classification. One basic block of MobileNet architecture can be represented in Figure 2.

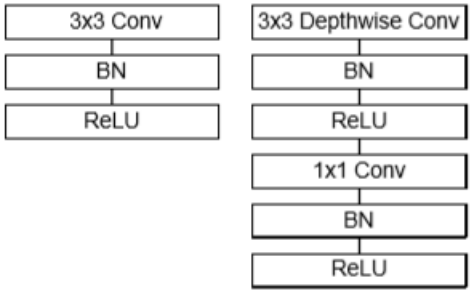


Figure 1: Left: Standard convolution layer with batchnorm and ReLU. Right: Depthwise Separable convolution with Depthwise and pointwise layers by batchnorm and ReLU.

Downsampling is handled with strided convolution in the depthwise convolutions as well as in the first layer. A final average pooling layer reduces the spatial resolution to 1 before the fully connected layer. Couting depthwise and pointwise convolutions as separate layers, MobileNet has 28 layers.

Table 1. MobileNet Body and Head Architecture

Type/Stride	Filter Shape	Input size
Body		
Conv / s1	3 x 3 x 3 x 32	28 x 28 x 3
Conv dw /s1	3 x 3 x 32 dw	28 x 28 x 32
Conv / s1	1 x 1 x 32 x 64	28 x 28 x 32
Conv dw / s2	3 x 3 x 64 dw	28 x 28 x 64
Conv / s1	1 x 1 x 64 x 64	14 x 14 x 64
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 256	14 x 14 x 256
Conv dw / s1	3 x 3 x 256 dw	14 x 14 x 256
Conv / s1	1 x 1 x 256 x 256	14 x 14 x 256
5x (Conv dw/s1, conv/s1)	3 x 3 x 256 dw 1 x 1 x 256 x 256	14 x 14 x 256 14 x 14 x 256
Conv dw / s1	3 x 3 x 256 dw	14 x 14 x 256
Conv / s1	1 x 1 x 256 x 256	14 x 14 x 256
Conv dw / s2	3 x 3 x 256 dw	14 x 14x 256
Conv / s1	1 x 1 x 256 x 256	7 x 7 x 256
Head		
Avg Pool / s1	Pool 7 x 7	7 x 7 x 256
FC / s1	256 x 10	1 x 1 x 256
Softmax / s1	Classifier	1 x 1 x 10

4 Training

The model is trained on CIFAR10 dataset using tensorflow and keras libraries using RMSprop for the weigth update. We used less regularization and data augmentation techniques because small models tend has lesser tendency for the overfitting in comparison to large models. We normalized the CIFAR10 dataset by subtracting the mean and dividing by variance. We used random left-right flip and image cropping to reduce the resolution of the images from 32x32 to 28x28.

After data preprocessing, the data is passed through the body of the model through all layers followed by a batchnorm and ReLU nonlinearity. Batch norm is used to normalize the input layer by adjusting and scaling the activations. It reduces the amount by what the hidden unit values shift around.[7] This layer normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. ReLU layer applies an elementwise activation function max(0, x), which turns negative values to zeros. This layer does not change the size of the volume and there are no hyperparameters. We used cross entropy loss to compute the loss per epoch.

The basic mobilenet architecture is already small and low, many times a specific usecase needs a small and fast model. In order to construct these smaller and less computationally models, we introduce two parameters: a) width multiplier b) resolution multiplier. The role of width multiplier is to thin a network uniformly at each layer. For a given layer and a width multiplier *a,* the number of input channels M becomes aM and the number of output channels N becomes aN. The computational cost of a depthwise seperable convolution with width multiplier a is:

$D_k \times D_k \times aM \times D_f \times D_f + aM \times aN \times D_f \times D_f$

The resolution multiplier hyperparameter is used to reduce the computational cost of a neural network. We apply this to input image and the internal representation of every layer is subsequently reduced by the same multiplier.

With lower values of width multiplier and resolution multiplier, we might get significantly low accuracy but it increases the network speed and reduces the requirements of computational resources. For CIFAR10 dataset, we set the width multiplier value as 1.

Table 2: Validation accuracy and loss for different values of width multipliers

Width Multiplier	Test Accuracy	Test Loss
1	0.84	0.72
0.75	0.81	0.75

Table 2 shows the training experiments done on width multiplier value for CIFAR10 dataset. For width multiplier value as 1, we get 84% test accuracy and 92% training accuracy. We get a decrease in loss when we set the width multiplier value as 1 instead of 0.75.

Figure 3 and 4 below represents training and validation data accuracy and loss during training for 60 epochs. The X axis represents epochs and Y axis represent accuracy and loss respectively. We get 92% training accuracy and 84% validation accuracy on CIFAR10 dataset.

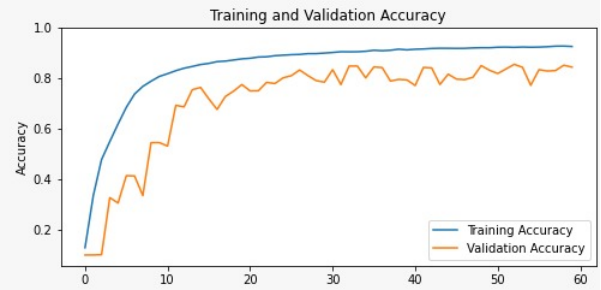


Figure 3: Training and validation accuracy curve on CIFAR10 dataset

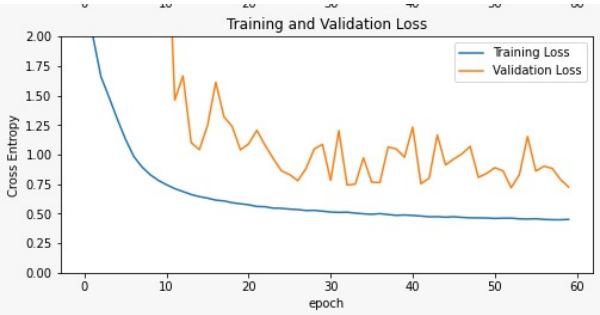


Figure 4: Training and validation loss curve on CIFAR10 dataset

5 Implementation

The mobilenet network model contains 27 layers excluding batch norm, ReLU and global average pooling layers. As CIFAR10 dataset is already a small dataset, the generated input and output feature map are also small and can be fit into internal memory of size 1 MB. Filter coefficients move from internal memory to external memory at each layer to perform the convolution operation. We presented the table 3 describes the total size of internal memory at each layer and data movement.

Table 3: I/O feature map size and location with internal memory used and data movement size each layer.

Layers	Input feature map size (bits)	Output feature map size (bits)	Internal memory used (bits)	Data Movement size (bits)
1	18816	200704	220384	20544
2	200704	200704	421376	576
3	200704	401408	824832	4096
4	401408	401408	1226816	1152
5	401408	200704	1435712	16384
6	200704	100352	1536640	1152
7	100352	200704	1745536	16384
8	200704	200704	1947392	2304
9	200704	200704	2164480	32768
10	200704	200704	2367488	4608
11	200704	200704	2633728	131072
12	200704	200704	2836736	4608
13	200704	404544	3306816	131072
14	404544	401408	3710528	4608
15	401408	401408	4177472	131072
16	401408	401408	4581184	4608
17	401408	401408	5048128	131072
18	401408	401408	5451840	4608
19	401408	401408	5918784	131072
20	401408	401408	6322496	4608
21	401408	401408	6789440	131072
22	401408	401408	7193152	4608
23	401408	401408	7660096	131072
24	401408	401408	8063808	8066112
			Memory Limit exceeded	
25	401408	100352	567296	8604224
26	100352	100352	203008	4608
27	100352	100352	368896	131072

After the 25th layer, the limit (1 MB) of internal memory storage exceeded so the data is moved to external memory by which the data movement size is increased.

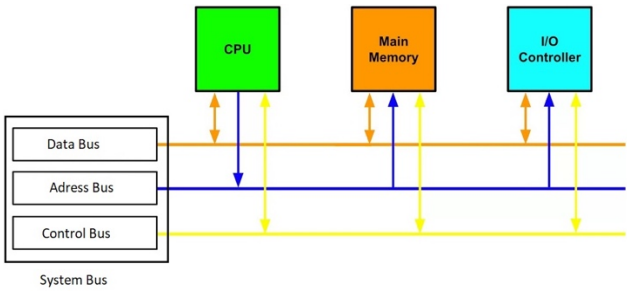


Figure 5: DDR 3 bus architecture.

To move the data from internal memory to external memory, we use 1 GB/s DDR bus. Table 4 describes the time taken in the data movement from internal memory to external memory, compute time and the total performance time at each layer by summing up the data movement time and compute time.

Table 4: I/O feature map size and location with internal memory used and data movement size each layer.

Lay--ers	Data Movement time (sec)	Compute Time (sec)	Performance Time (sec)
1	0.000002568	1.35475E-06	3.92275E-06
2	0.000000072	4.51584E-07	5.23584E-07
3	0.000000512	3.21126E-06	3.72326E-06
4	0.000000144	9.03168E-07	1.04717E-06
5	0.000002048	1.60563E-06	3.65363E-06
6	0.000000144	2.25792E-07	3.69792E-07
7	0.000002048	3.21126E-06	5.25926E-06
8	0.000000288	4.51584E-07	7.39584E-07
9	0.000004096	1.28451E-05	1.69411E-05
10	0.000000576	9.03168E-07	1.47917E-06
11	0.000016384	2.56901E-05	4.20741E-05
12	0.000000576	9.03168E-07	1.47917E-06
13	0.000016384	2.56901E-05	4.20741E-05
14	0.000000576	9.03168E-07	1.47917E-06
15	0.000016384	2.56901E-05	4.20741E-05
16	0.000000576	9.03168E-07	1.47917E-06
17	0.000016384	2.56901E-05	4.20741E-05
18	0.000000576	9.03168E-07	1.47917E-06
19	0.000016384	2.56901E-05	4.20741E-05
20	0.000000576	9.03168E-07	1.47917E-06
21	0.000016384	2.56901E-05	4.20741E-05
22	0.000000576	9.03168E-07	1.47917E-06
23	0.000016384	2.56901E-05	4.20741E-05
24	0.001008264	9.03168E-07	0.001009167
		Memory limit Exceeded	
25	0.001075528	6.42253E-06	0.001081951
26	0.000000576	2.25792E-07	8.01792E-07
27	0.000016384	6.42253E-06	2.28065E-05
			0.002461038

The total performance time calculated is 0.002461038. The performance time at layer 25 is considerably more than any other layers of the model due to large data movement as the limit of internal memory exceeded (1 MB). It also happened because of the large filter coefficient size compared to other layers.

6 Conclusion

We proposed an efficient model architecture called MobileNet based on depthwise seperable convolutions. We discussed about some of the important design decisions how to build an efficient, smaller and faster network considering the computational resources being used and trading off a reasonable amount of accuracy to reduce size and latency. We then presented the training methods and experiments we did in the design part and discussed about the internal memory, external memory and data movement happened at each layer for input feature maps, filter coefficients and output feature maps in the model. We concluded by demonstrating MobileNet's effectiveness when applied to a wide variety of tasks. As a next step, we plan to release this network design for various applications of image classification and computer vision.

The mobilenet network model architecture code can be found on the below link:
https://colab.research.google.com/drive/1EsiXLz10Tt2fg9-h7Pc5cVO7jO06q3g4

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.