MobileNets

Chengjian Tao ct2891, Abhijeet Mishra aam2285, Abhinandan Srinivasan as5630

Columbia University in the city of New York

Abstract

In this work, we use a Deep Neural Network with Imagenet dataset to develop and then compare the MobileNet Architecture to perform image Classification. MobileNets are Neural Networks with relatively less parameters, but which can still perform very well on image classification tasks [1]. Furthermore, we investigated the dependence of the output image on various network parameters such as image quality, trainable parameters, epochs, hidden layers, and number of classes among others. This paper is a comparison of the performance of MobileNets vs other models such as GoogleNet (InceptionV3) and VGG16. The authors have also attempted to use GAN to augment data and got mixed results.

1. Introduction

Computer Vision and convolutional neural networks have come hand in hand for quite some time now. It all started with AlexNet [2] and the ImageNet Challenge [5]. Generally, a commonly considered way to make the network more powerful has been to increase the number layers and the size making it more and more complex. However, in our project, we try to reiterate the MobileNets architecture to find a more optimal means of executing the task for different datasets. MobileNets was among the first models to substitute 2D Convolution layers with Depth-wise convoluted layers, which reduced the parameters required for each filter drastically and allowed us to have channels with large depths. MobilNets proved that using Depthwise Convoluted layers resulted in very low loss of accuracy but allowed us to achieve models with very low memory, which could be even run on small less powerful machines such as our cell phones.

2. Summary of the original paper

In this section, we will discuss the methodology employed by Andrew, et al. to work on the architecture of the developing the MobileNet architecture and how it holds ground with various cases.

2.1 Methodology of the original paper

The paper presents the architecture employed and the algorithm used in the original paper. The MobileNet makes use of depth wise separable convolution. It is a factorized convolution and a pointwise convolution. The process involves sequential use of single filter to each input channel and then the pointwise convolution. This helps reduce computation as it divides it into two different layers for filtering and combining.

Figure 1a: Depthwise Convolution [3]

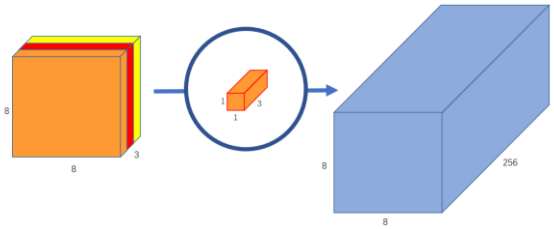


Figure 1b: Pointwise convolution [3]

For standard convolutional layer

$Input\ feature\ map\ F$: D_F x D_F x M

Output feature map G: D_F x D_F x N

D_F: spatial width and height of input

M: number of Input channels

D_G: spatial width and height of output

N: number of Output channels

Standard convolutional layer kernelizations:

$Kernel$: D_K x D_K x M x N

Output feature map with standard stride and padding

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

The cost of computation for standard convolutions:

$$D_K. D_K. M. N. D_F. D_F \quad \text{..... (1)}$$

MobileNet breaks the inter dependency of these factors on each other by using separable convolution.

The depthwise separable convolution uses both pointwise convolution and depthwise convolutions. Each filter is applied with depthwise convolution and the pointwise convolution in the next step. This creates a linear combination of the output. The nonlinearities used are batchnorm and ReLU. Below is the depthwise convolution with one filter input per channel.

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

$\hat{K}$: depthwise convolutional kernel of size

Kernel size: D_K x D_K x M where the:

m: Filter index

$\hat{G}$: output feature map

The cost of computation of depthwise convolution:

$$D_K. D_K. M. D_F. D_F \quad \text{.....(2)}$$

Depthwise convolution does not create any new features, it focuses only on filtering input channels. To counter this, an additional layer is introduced. This layer does a 1x1 convolution to compute a linear combination of the output. The combination of these two processes is called depthwise separable convolution [4].

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

This is nothing but the sum of (1) and (2) i.e depthwise convolutions and 1x1 pointwise convolutions.

The reduction in computation because of the use of this two-stage process is:

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

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

2.2 Network Structure and Training

Depthwise separable convolutions are the basis for the MobileNet architecture. Only the first layer is a fully convoluted layer. The architecture for the same is defined in Table 1 below.

Table 1: MobileNet 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	$1 \times 1 \times 1000$

2.3 Key Results of the Original Paper

The paper made many groundbreaking conclusions, that paved way for many networks that based off this.

- Innovative Depthwise separable convolutions
- Directions to increasing efficiency of the model.
- Use of width multiplier and resolution multiplier and the relation between these and accuracy reduction
- Comparison with existing networks and showing its performance, size, accuracy and speed.
- Analysis across different tasks.

3. Methodology

The main of the project was to use the 3 models given below and to compare its performance across two different data sets

Models:

1. MobileNet
2. VGG16
3. Inception

Data Sets:

(i) Tiny Image Net: (100k photos; 64 x 64 x 3; 200 Classes)

(ii) Stanford Dogs: (21k images; High res photos; 120 Classes)

(i) Usage of Tiny Imagenet

For Tiny Imagenet, we first used our written MobileNet architecture and completely trained the entire network and determined the accuracy. We also tried various width multipliers on the network (0.75, 0.5, 0.25).

Hoping to get better results, we used MobileNet's pre-trained network and added 2 layers to satisfy out class classification and retrained the network. We then constructed the VGG-16 and the Inception V3 network with weights pre-trained on ImageNet. All the results from these tests will be discussed in detail in the following sections.

(ii) Use of Stanford Dogs

The Stanford dogs dataset is a much better dataset in terms of resolution of the image, which resulted in approximately a 50% accuracy with MobileNet, 22% with VGG16 and 10% with inception. We first trained the network with 64 x x64 images and then with the 128 x 128 images. The latter proved to be much better in terms of accuracy, but the training time was compromised.

3.1 Objectives and Technical Challenges

The training data, would easily overload the RAM of the system, thereby making the system crash. Thus, it was extremely important that that we took the size of the data into consideration while performing calculations. We could also overcome this challenge by only loading the data just before we needed to process it. I.e. loading only one batch at a time, just before passing it through the training set.

Due to the size of the data, we would have ideally wanted to perform the calculations on multiple GPUs. however, due to technical restrictions, we could only use one GPU. This limited the speed of training and thus training the entire layer took a lot of time.

The quality of data also played a key role while training. We realized that if the size of data is smaller than 224 pixels, it severely compromises the performance of the neural networks. This affects deep neural networks even more as the size of network reduces with layers.

3.2 Problem Formulation and Design

To accomplish our objectives, we adhered to the mathematical and conceptual formulation provided by Andrew et al.

3.2.1 Width Multiplier

Sometimes, the model has a necessity to be smaller and less expensive to compute in order to meet the specifications of the application. Andrew wt al [1] proposes α, the width multiplier. This multiplier works on each layer and thins it uniformly. Both the input and the output channels are multiplied by a factor of α, thus making $\alpha\mathrm{M}\ (\mathrm{input})$ and $\alpha\mathrm{N}(\mathrm{output})$

The updated computational cost with width multiplier α is:

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

where α ϵ (0; 1]. α = 1 is the MobileNet with no reduction in any layer and any value of $\alpha < 1$ indicates a thinning (reduced MobileNet).

Width multiplier reduces the number of parameters quadratically by almost α^2, thereby drastically reducing cost of computation. The paper then discusses about using different α values and defining a smaller model. We have used α = 1, 0.75, 0.5 and 0.25, to see the change in accuracy, latency and reductions in size.

3.2.2 Resolution Multiplier: Reduced representation

Apart from the width multiplier, the resolution multiplier ρ also plays a huge part in lowering the cost

of computation. Andrew et al applied this to the input image, by doing so, each layer of the network is reduced by the same factor of ρ.

The updated computational cost with width multiplier α and resolution multiplier ρ is:

$$D_K . D_K . \alpha M . \rho D_F . \rho D_F + \alpha M . \alpha N . \rho D_F . \rho D_F$$

where ρ ϵ (0; 1] ρ = 1 is the baseline MobileNet and any value of ρ < 1 represents a reduced MobileNet. The resolution multiplier also, much like width multiplier reduces the computational cost by ρ^2. We have used ρ values of 0.8125, 0.6875, 0.5625 to see the effects on the network, these correspond to the typical input resolutions that we give as inputs for the network (224, 192, 160 or 128).

3.2.3 Using GANs to develop images

Additionally, we use GANs to develop images of dogs that we save and then pass through the MobileNet network to see if it can find the breed of the animal.

The GAN algorithm consists of two neural networks. The generator network takes input of a noise vector and outputs a fake image. We then input these fake images and also the real images into the discriminator network which predicts whether the images are fake or real. Thus, the goal of the generator network is to develop images which can trick the discriminator network into believing they are real images. These two networks compete and grow together. However, a key issue could be that the Discriminator network becomes too powerful and the generator network simply cannot trick it. This leads to divergence between the loss values of the two networks. We also experienced this issue and thus our generator network could not produce extremely rich images. The networks also take a long time to train.

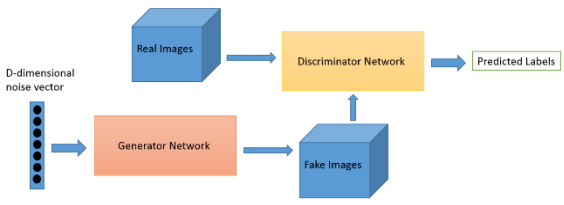


Figure 3a: GAN Network [8]

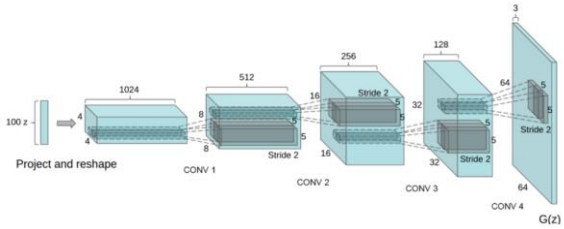


Figure3b: DCGAN generator structure

The GAN algorithm we used was able to train and develop photos was able to produce images of dogs, but the improvement of images stopped at a certain point because the classifier's network overpowered the generator. Below in Figure 4 of the 2nd epoch of the algorithm and the 430th epoch,

Figure 4: Images generated from the GAN

4. Implementation

In this section, we will discuss the implementation of this architecture in code. We will begin with discussing the software design and follow it up with the process.

4.1 Software Design

The code for this project was written using the TensorFlow package for Python. The training data is loaded from the google cloud, Jupyter Notebooks have been used to train the entire algorithm as it offers easiest access to hyperparameters and allows us to run parts of the code, thereby saving a lot of effort. Initially the data is loaded and pre-processed. This includes normalization and scaling the data. We also convert the y-data to one hot array data. Following this we design the entire model in using keras and TensorFlow. Thus, we are able to define the layers, activation functions, normalization layers etc. Following this, the loss function and optimizer functions are defined. We then define the parameters such as epochs, batch size, and also set up the data generator and load the training data in it. Once this

has been completed, we begin the training session and trying to understand how the training and validation accuracy vary. We then reiterate through the process by optimizing the parameters and storing the results for future use.

5. Results

In the end, we trained the three networks (MobileNet, VCG, Inception) on two different data sets, compared the results of those and also created a GAN to spit out images that we tested on out trained network.

5.1 Project Results

10 Classes ImageNet

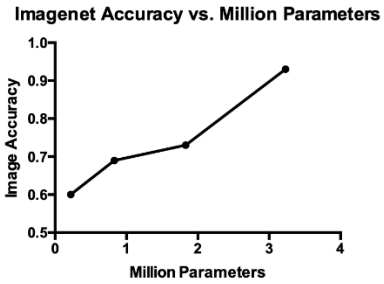


Figure 5. This figure shows the tradeoff between the number of parameters and accuracy on the 10-classes-ImageNet.

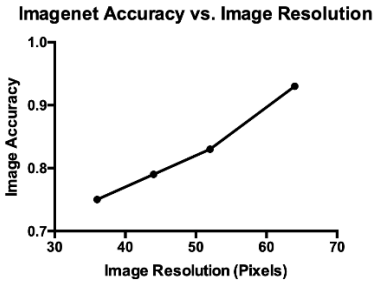


Figure 6: This figure shows the tradeoff between the image resolution & accuracy on the 10-classes ImageNet.

5.2 Comparison of Results

We compared thinner models with width multiplier of 0.75 to shallower models with less layers. Table 2 shows that with similar number of parameters, making MobileNets thinner is 6% better than making MobileNets shallower.

Table 2: Narrow vs. Shallow MobileNet

Model	ImageNet Accuracy	Million Parameters
0.75 MobileNet-64	0.73	1.83
Shallow MobileNet-64	0.67	1.88

Table 3 shows the ImageNet accuracy and computation tradeoffs of making the MobileNet narrower with the width multiplier alpha. ImageNet accuracy drops off smoothly as the MobileNet becomes narrower.

Table 3:MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Parameters
1.0 MobileNet-64	0.93	3.23
0.75 MobileNet-	0.73	1.83
0.5 MobileNet-64	0.69	0.83
0.25 MobileNet-	0.6	0.22

Table 4 shows the ImageNet accuracy and image resolution tradeoffs by training MobileNets with reduced input resolution. ImageNet accuracy drops off smoothly as the input resolution decreases.

Width Multiplier	ImageNet Accuracy	Million Parameters
1.0 MobileNet-64	0.93	3.23
1.0 MobileNet-52	0.83	3.23
1.0 MobileNet-44	0.79	3.23
1.0 MobileNet-36	0.75	3.23

Table 4: MobileNet Resolution

Table 5 compares our MobileNet to the VGG16 and GoogleNet. Our MobileNet performs better than both VGG16 and GoogleNet while being 27 times and 2.5 times less compute intensive.

Table 5: MobileNet Comparison to Popular Models

Width Multiplier	ImageNet	Million Parameters
1.0 MobileNet-64	0.93	3.23
VGG 16	0.886	14.9
GoogleNet	0.112	22.4

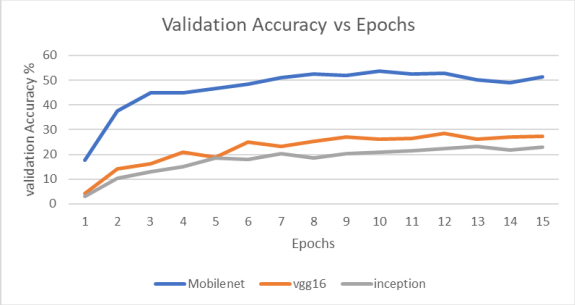
The training time for 10-classes-ImageNet is about 20 minutes for each MobileNet model, 4 minutes for GoogleNet and VGG16. We terminated the training process for GoogleNet in advance because the training accuracy never increases. The training time for VGG16 was short because most part of VGG16 were already trained and we only need to train the last few layers, while the training for our MobileNet was from scratch.

For 200-classes-ImageNet, the accuracy of our MobileNet was 22.44%, and the training time was about 2 hours and 20 minutes. The accuracy of pretrained MobileNet from keras was about 24.86%.

The Stanford Dogs data set consists of approximately 120 classes of dogs, with each class having about 150 images. Thus, the total number of images is about 21000. The images are large in size and thus they can achieve a higher classification rate. We sampled down all images to the size 128*128*3 while training. After training the models, we found that MobileNet had the highest accuracy of about 50%. It was followed by VGG16 and then by InceptionV3. InceptionV3 seems to have performed badly, because there was probably not enough data. Moreover, we only tweaked the last few layers of each pretrained model and thus we were able to train the model quickly. Although we trained for only 15 epochs, the models very quickly reached their peak accuracy and the validation accuracy remained stable or in some cases, even dropped a little. Thus, we can say that the models were tending towards overfit. However, on increasing the size of the models, the accuracy would stop to increase. This suggested the data needed to be larger. This could be in two ways, either we could increase the size of each image or increase the total number of datasets. However, it was very clear that the quality of the data was key in this case, as it very heavily altered the training of the neural networks.

Dogs Data	classes	Image_Size	Trainable Parameters	Time	Accuracy
Mobilenet	200	128*128*3	443000	945	49.4
VGG16	200	128*128*3	395000	975	25.8
Inception	200	128*128*3	425000	975	20.78

Figure 2: Parameters and accuracy vs Epoch graphs for different models on Stanford Dogs dataset



5.3 Discussion of Insights Gained

We were unable to use the original ImageNet since it was too large and unable to be uploaded to the Google Cloud. So we downloaded a tiny ImageNet dataset which contained 200 classes and 100000 images. The size of each image was 64*64 rather than 224*224.

10 Class Classification

There was an overfitting issue in most of our models since the validation and testing accuracy were always less than the training accuracy. We have tried to reduce overfitting by normalizing each image, adding dropout layers, and using data augmentation. By normalizing each image, the training process became much slower and the testing accuracy dropped dramatically. We suspected that this is due to poor image quality since the original image size is 224*224, while ours were 64*64. Also, adding dropout layers did not help. However, after we implemented data augmentation, the testing accuracy increased from 50% to around 90% for 10-classes-ImageNet.

200 Class Classification

For 200-classes-ImageNet, the testing accuracy was a little bit above 20% even if we used data augmentation. We suspected that we did not have enough images to train the model with 200 classes, and the image quality is too low. The only way to

optimize the result is to use the original ImageNet to train the model.

Compared to the results in the paper, we got higher ImageNet accuracy (93%) than the paper (70.6%). This is expected because we only trained on 10 classes, while the paper trained on 1000 classes. For MobileNet Width Multiplier and MobileNet resolution, our results were consistent with the paper's results. The accuracy decreases as width multiplier decreases, and the accuracy decreases as image resolution decreases.

For GoogleNet, we only achieved accuracy of 11.2%, while the paper had accuracy of 69.8%. This might have also been caused by the poor image quality we had since the resolution we had was 64*64, and we only had 100000 images to train on.

6. Conclusion

With the methods proposed in the paper, we were able to compare the different models and prove that MobileNets performs as well as others, if not better. We believe that our results are comparable with some of the tests in the actual paper. Additionally, we noted the importance of having a proper dataset on the performance. Moreover, we also tried to implement a GAN network.

7. Acknowledgement

We thank Dr. Zoran Kostic for teaching Neural Networks and Deep Learning this semester. We also thank the course TAs, Arjun DCunha, Aijia Gao, Zhen Wang, Chengrui Wu, Huixiang Zhuang, Deepak Ravishankar, for their help and responsiveness when questions arose throughout the semester.

8. References

[1] Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam: MobileNets- Efficient Convolutional Neural Networks for Mobile Vision Applications.: Google Inc.

[2] 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.

[3] A basic Introduction to separable Convolutions – Chi-Feng Wang: Medium https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

[4] L. Sifre. Rigid-motion scattering for image classification. PhD thesis, Ph. D. thesis, 2014.

[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 115(3):211–252, 2015.

[6] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[7] DCGANs – Generating Dog Images with Tesnorflow and Keras: Medium https://towardsdatascience.com/dcgans-generating-dog-images-with-tensorflow-and-keras-fb51a1071432

[8] A Beginner's Guide to Generative Adversarial Networks (GANs) – PathMind https://pathmind.com/wiki/generative-adversarial-network-gan

9. Appendix

9.1 Individual Student contributions in fractions – table

UNI	ct2891	aam2285	as5630
Name	Chengjian Tao	Abhijeet Mishra	Abhinandan Srinivasan
Fraction of total contribution	**1/3**	**1/3**	**1/3**
Contribution	1) Code for training 2) Testing mobilenet under different conditions. 3) Report: 1-2	1) Code for loss simulation, 2) GAN networks. 3) Report 5-7	1) Procured Dataset. 2) Data parsing for I/O 3) Report 3-5