MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

E4040.2022Fall.AHSN.report

Safeyah Alshemali ska2145, Hannah Portes hsp2122, Aparna Muraleekrishnan am5964
Columbia University

Abstract

Today, there are many applications for AI on mobile and edge devices that require lighter models which can work with limited resources. The original paper presents a class of efficient models called MobileNets. These use depth-wise convolutions to build deep neural networks that are thinner and lighter by using two hyperparameters that trade off between resource use and accuracy. We try to recreate the experiments and findings of the original paper by reconstructing the MobileNet structure using depth and pointwise convolution, and the same model structure using regular convolution. Comparing the results of the same two datasets through these two models we show that a similar accuracy can be achieved, except with fewer parameters and resource allocation. After extensive testing with multiple datasets split in different ways we concluded our datasets and MobileNet were not compatible. We then attempt to make a simpler CNN and MobileNet to still prove that between the two models similar accuracy can be achieved with CNN using more resources and taking significantly more time to train. We conduct experiments on three datasets with varying results described in the report below. We build our own MobileNet models and CNN with the same number of layers to demonstrate the varying latency and accuracy. Our computational complexity and latency are consistent with the findings of the paper. Our models produce lower accuracy for Tiny ImageNets because of the large number of classes. Our accuracy for Stanford Dogs does not match the results of the original paper. Our model appears to be overfitting given the high training accuracy and very low validation accuracy. We believe this is because the model we are replicating is tested on 14 million images belonging to ImageNet and our dataset is 100,000 images. We implement the width multiplier with ImageNette with differing results from the original paper. This is again a result of the complexity of our model and small dataset we use to train. When varying the resolution parameter of the model and using the same Imagenettes, we produced findings that are aligned with the original paper. Our paper details the approach, challenges, implementation, and findings of our team.

1. Introduction

Convolutional neural networks had a resurgence in 2012 when AlexNet won the ImageNet challenge in 2012. The general direction of convolutional neural networks in computer vision has been focused on accuracy. This focus to achieve higher accuracy results in deeper and more complex networks that are inefficient. In real world applications tasks need to be completed in a timely fashion. The original paper presents an approach to building networks with equivalent accuracy that are less computationally taxing.

Our paper covers the concepts and findings of the original paper in section 2. Section 3 covers our approach and the challenges we encountered as we were carrying out our experiments. Section 4 details our implementation, and section 5 covers our results and conclusions.

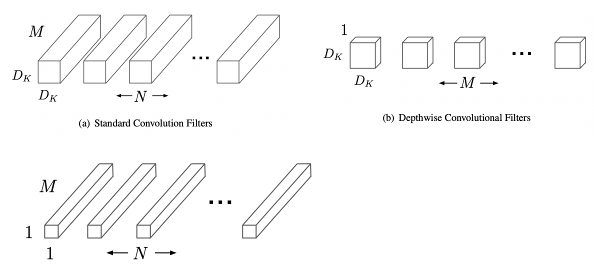
2. Summary of the Original Paper

2.1 Methodology of the Original Paper

Prior approaches covered in the paper generally consist of training small networks or compressing pretrained networks. The paper proposes a new model architecture focused on depth-wise separable convolutions, where the model developer may customize the network to match resource restrictions. They continue by exploring using width and resolution multipliers to demonstrate constructing smaller and faster Mobilenets by trading accuracy for speed.

The main component of MobileNets are the depthwise separable convolutions to reduce the computation required which was first introduced in Inception models. The original paper briefly covers Flattened networks, factorized networks, the Xception network, deep fried convnets, structured transform networks, and Squeezenet as examples of papers that focus on reducing computation to produce smaller and more efficient networks. Other approaches covered include factorizing, shrinking, or compressing pretrained networks resulting in smaller networks desired, as well as distillation and low bit networks.

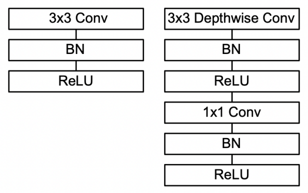
The core layers of MobileNet are depthwise separable convolutions. These are a form of factorized ions that separate standard convolutions into a depthwise convolution and a 1 x 1 pointwise convolution. Factorization into this structure reduces the computation model and size greatly. Figure (1a) shows a standard convolution factorized into a depthwise (1b) and pointwise convolution (1c) [5].


M
D_K
D_K
← N →
(a) Standard Convolution Filters
1
D_K
D_K
← M →
(b) Depthwise Convolutional Filters
M
1
1
← N →

Figure(1): Standard convolution layer vs Depthwise and Pointwise layer

The MobileNet structure used in the original paper is pictured in Table 1. This is the exact structure we modeled with our own experiments to compare MobileNet to a standard Convolutional Neural Network. The first layer is a full convolution, followed by depthwise separable convolutions. Each layer is followed by batchnorm and ReLu, until the last layer which is a fully connected layer that connects to a softmax classification layer. This results in a total of 28 layers.

Figure 2 shows a standard convolutional layer followed by batchnorm and ReLu and the depthwise and pointwise convolutions each followed by batchnorm and ReLu [5].



Figure(2) Left: Standard convolution layer with batchnorm and ReLU. Right: Depthwise Separable convolution followed by batchnorm and ReLU.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5× 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	$1 \times 1 \times 1000$

The original authors introduce a method to even more reduce computational complexity and time. The parameter α is a width multiplier used to uniformly thin the network every layer. The computation cost of a depthwise separable convolution with α is shown in figure 6, with the number of input channels M and number of output channels N. $\alpha \in (0, 1]$, typically increments of 0.25. α = 1 is the baseline and α < 1 are reduced MobileNets [5].

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

The second parameter the original authors introduce to reduce computational complexity and time is the parameter ρ, a resolution multiplier to reduce the internal representation of each layer by the same value. In practice ρ is set to the input resolution.

The computation cost of a depthwise separable convolution with α and ρ is shown in figure 7, with the number of input channels M and number of output channels N. $\rho \in (0, 1]$, ρ = 1 is the baseline and ρ < 1 are reduced MobileNets. This resolution multiplier reduces computation cost by ρ2 [5].

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

2.2 Key Results of the Original Paper

The original paper first showed that using depthwise separable convolutions compared to a standard model using full convolutions reduces accuracy by only 1% and saves a great amount on parameters and mult-adds (Table 4 from the paper) [5].

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

They also demonstrated the results of making a thinner model using a width multiplier compared to a narrower model using fewer layers, concluding that making MobileNets thinner is 3% better than making MobileNets shallower (Table 5 from the paper) [5].

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

Using the width multiplier α, Table 6 shows the comparison of shrinking the architecture by this α. The original paper concluded accuracy decreases slightly and consistently until α is 0.25 resulting in an architecture that is too small [5].

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 10 shows consistent results on the Stanford Dogs dataset. The original paper trained a Mobilenet for fine grained recognition using a large and noisy dataset, fine tuned on the Stanford Dogs data, and showed the results are 0.7% away from the state of the art results of the Inception V3 model with significantly fewer parameters and operations [5].

Table 10. MobileNet for Stanford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

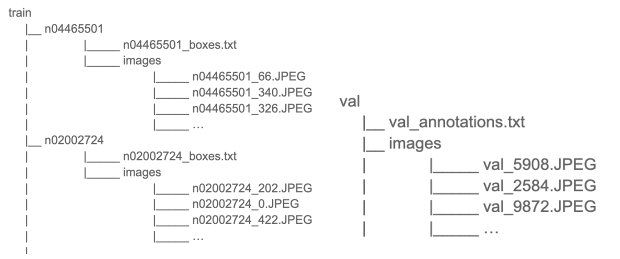
3. Methodology

3.1. Objectives and Technical Challenges

Our primary objective was to design a MobileNet that can give high accuracy when trained on a dataset of our choice. MobileNets are useful since they are lighter and use significantly less compute to attain similar accuracy as a bigger network. To test this, we planned on running our model on a GPU system and CPU to provide a timing analysis.We also planned to build a similar CNN model to serve as a basic model to compare computation parameters and accuracy with. Additionally, we vary the hyperparameters to study the effect on our model. We planned on training our MobileNet model on images of different resolutions and comparing the effect on accuracy. We also planned on comparing the results of using four different values for the width multiplier (alpha). Our other objective was to compare the performance of our model to that of other existing networks like AlexNet. For this, we needed to select an AlexNet of similar size so the results would be comparable.

3.1.2. : Data Processing Challenges:

To begin recreating the experiments conducted in the original paper we first had to pick an appropriate dataset. Initially we looked into using Fashion MNIST but decided to work with a dataset that already had three channels instead of choosing to modify this dataset. The Imagenets that original paper used was too large a dataset for us to train on given time and resource constraints. So we settled on Tiny ImageNet, which we loaded from Stanford University's public resources. The directory structure of the training data is pictured in Figure (3-up) and the directory structure of the validation data is pictured in Figure 3-down. The training dataset consists of 100,000 images categorized into 200 folders corresponding to classes. The validation data consists of 10,000 images in a single image folder, and a .txt file containing validation image file names and their corresponding classes. This requires mapping the .txt file to a dataset within a notebook and then reading through the image file names to map each to its true class. This is how we compiled our training and validation data. The provider of tiny imagenet does not publish the labels belonging to the test dataset so the testing file was not extremely relevant to our experiment.



```
train
  |__ n04465501
  |         |_____ n04465501_boxes.txt
  |         |_____ images
  |                    |_____ n04465501_66.JPEG
  |                    |_____ n04465501_340.JPEG
  |                    |_____ n04465501_326.JPEG
  |                    |_____ ...
  |__ n02002724
  |         |_____ n02002724_boxes.txt
  |         |_____ images
  |                    |_____ n02002724_202.JPEG
  |                    |_____ n02002724_0.JPEG
  |                    |_____ n02002724_422.JPEG
  |                    |_____ ...
  |

val
  |__ val_annotations.txt
  |__ images
  |        |_____ val_5908.JPEG
  |        |_____ val_2584.JPEG
  |        |_____ val_9872.JPEG
  |        |_____ ...
```

Figure(3): Up: training data structure, Down: validation data structure

The original paper covers residing the images to 224, 192, 160 and 128. We resized our images to 128 x 128 to help reduce training time for our experiments. To assist with data augmentation we use ImageDataGenerators, as well as define our own training and validation split with the ratio 80:20 from the training folder. On this, initial models produced validation accuracy much less than training accuracy. To tackle this, data preprocessing was modified, taking proportions of each class and shuffling the data to ensure that the training and validation data are randomly distributed and not skewed towards one class, which would produce biased results.

As a second experiment for our models we chose to use another dataset used by the original authors as well - Stanford Dogs. This is a dataset of 120 dog breeds, 12,000 training images, and 8,580 test images. This dataset exists as part of tenforflow's dataset module and is imported already split into train and test data. To have a more equal split between train and validation, we decided to split the 12,000 images into training and validation with a ratio of 70:30. We set the images to 124 height and width, the smallest dimensions our MobileNet can accommodate, and defined functions to resize the image, one-hot encode the labels, and shuffle and prepare the data for training. We also decided to split this data 80:20 from the training set to create our own training and validation data of size . Our objective was to run this additional dataset through our MobileNet and our standard CNN to verify that very similar accuracy can be achieved by both models, while the time and resources used are significantly smaller in training the MobileNet compared to the required time and resources used by the standard CNN.

3.1.3. : Model Challenges:

We started experimentation by first recreating the MobileNet model as listed by the original paper. The description table has an error in layer 14 where the stride has been mistyped as 2. We confirmed that the stride should be 1 [6] and designed a replica of the model. Initial experimentation with this model and the Tiny ImageNet dataset presented severe overfitting. The original model was trained on ImageNets which has millions of images as opposed to the tiny imagenet of 10000 images. We reduced the number of layers and added dropout to create more optimal models.

On designing a shallower model to work with tiny Imagenets, we tried to further improve accuracy by using different optimizers and finally concluded that the Adam optimizer produced the best results. We also introduced dropout into our different models to reduce overfitting. Our shallow models produced validation accuracy in the range of 0.39 to 0.46 for Tiny ImageNets.

Resource constraints caused us to use another dataset, Imagenettes with 10 classes. Since the new dataset was simpler, we developed a shallower model and used dropout to generate the architecture version we used for further experimentation with hyperparameters. With imagenettes, we obtained better results and a lower training time for our model.

3.2. Problem Formulation and Design Description

To accomplish this project, two models have been built to demonstrate the effectiveness of the proposal in this paper, which are the Standard CNN model and the MobileNet model. Both models have been built by using primitive components of Keras and Tensorflow. To have a fair comparison between the performance of the two models, we built equivalent structures considering that CNN model uses standard convolution layers while the MobileNet uses Depthwise Separable Convolution layers .The standard CNN model consists of 14 convolutional layers (Conv2D) of kernel size (3,3) and changeable stride value between 1 and 2. According to the paper and as shown in Figure (2- left), each standard convolution is followed by batch normalization and “Relu” activation function.

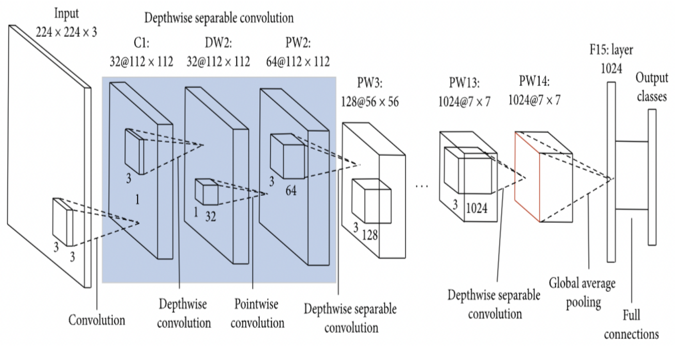


Figure (4): MobileNet Block Arcichture [7]

The MobileNet architecture as described in the paper in Table (1) in section 2 and in the Figure (4). The first layer is a standard convolution layer of kernel size (3,3) and stride 2. Then a series of Depthwise Separable Convolution layers which consist of the depth-wise and the pointwise layers as shown in the image below. The depth-wise layer handles the down-sampling using strided Conv2D layer. Thus, the stride varies between 1 and 2 in these series of layers. The point-wise convolution is used to combine the output of the depthwise convolution output. Each of these two layers is followed by batch Normalization and “Relu” activation function. The architecture of the Depthwise Separable Convolution layer is illustrated in Figure (2 - right side) along with the block diagram in Figure (5). The total number of DwSC layers are 13 and 1 standard convolution layer. So , the total number of core layers is 14 equivalent to the Standard CNN model.

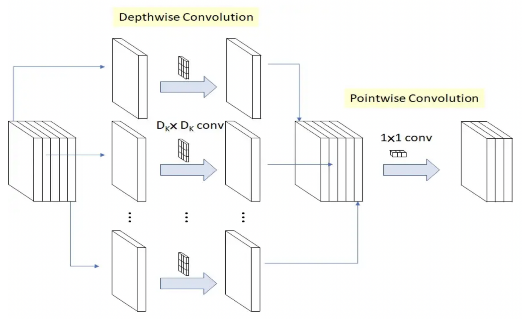


Figure (5): Block diagram of the Depthwise Separable Convolution layer [8]

4. Implementation

In this section, we will present the process of implementing the project. Section 4.1, covers the information about the dataset we used to train and test our models and do the experiments. Section 4.2 illustrates the Deep Neural Network Models that we built to examine our objectives as mentioned in Section 3.1. Section 4.3, we present the software design of the project along with the source code of our project.

4.1 Data

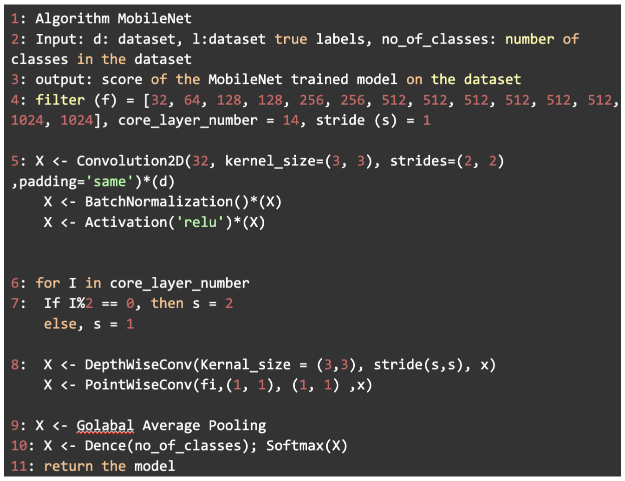
As we encountered difficulty with the first 2 datasets, we turned to Imagenette [2]. This is a tensorflow dataset subset of ImageNet containing 10 classes, with 9,469 training images and 3,925 validation images. We chose to use the 160px size of this dataset and resize the images down to 128 using a mapping function. We again use ImageDataGenerator for data augmentation to rescale, rotate, and vertically shift the images before training. We ran our experiments using the predefined training and validation data splits.

4.2 Software Design

In section 4.2, we will demonstrate different models that we implemented in order to accomplish this project. The implementations will be explained along with the Pusedcode that represents the algorithm behind it and a link to the course code of each.

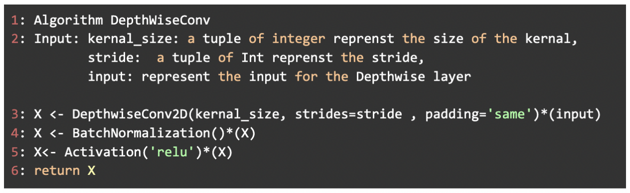
4.2.1: Original MobileNet and Standard CNN Models

The first implementation we did is reconstruct the MobileNet suggested by the authors which is shown in the Pseudocode (1). After processing the data, the model will be created according to the size of the images, number of classes in the dataset, and width parameter (alpha). In line 4, an array of filters, number of Depthwise Separable Convolution layer layers and stride variables have been initialized. Line 5, is an implementation of the first standard convolution layer. Then in line 6, the series of DepthWiseConv and PointWiseConv is implemented. As the stride is different according to the layer, an if statement is used to set appropriate stride in line 7. The pseudocodes of the DepthWiseConv and PointWiseConv function is shown in [Pseudocode 2]. Note that DepthwiseConv2D is a layer block provided by Keras library. Then, GlobalAveragePooling is used to reduce the spatial resolution and produce an average feature map.The final layer is a Dense layer with softmax activation. A layer structure was used to create a standard CNN model by replacing line 8 with the standard Conv2D layer. The source of the code can be found in our group github in the following link.

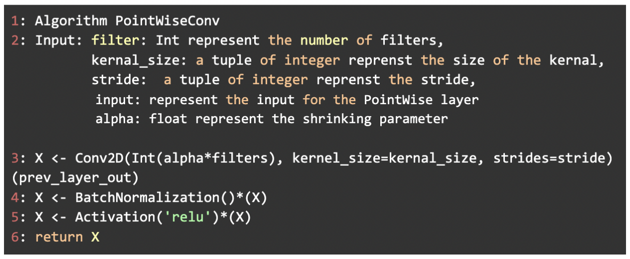


```
1: Algorithm MobileNet
2: Input: d: dataset, l:dataset true labels, no_of_classes: number of
classes in the dataset
3: output: score of the MobileNet trained model on the dataset
4: filter (f) = [32, 64, 128, 128, 256, 256, 512, 512, 512, 512, 512,
1024, 1024], core_layer_number = 14, stride (s) = 1

5: X <- Convolution2D(32, kernel_size=(3, 3), strides=(2, 2)
,padding='same')*(d)
    X <- BatchNormalization()*(X)
    X <- Activation('relu')*(X)


6: for I in core_layer_number
7:  If I%2 == 0, then s = 2
    else, s = 1

8:  X <- DepthWiseConv(Kernal_size = (3,3), stride(s,s), x)
    X <- PointWiseConv(fi,(1, 1), (1, 1) ,x)

9: X <- Golabal Average Pooling
10: X <- Dence(no_of_classes); Softmax(X)
11: return the model
```

Pseudocode (1)



```
Algorithm DepthWiseConv
Input: kernal_size: a tuple of integer reprenst the size of the kernal,
       stride:  a tuple of Int reprenst the stride,
       input: represent the input for the Depthwise layer

X <- DepthwiseConv2D(kernal_size, strides=stride , padding='same')*(input)
X <- BatchNormalization()*(X)
X<- Activation('relu')*(X)
return X
```



```
Algorithm PointWiseConv
Input: filter: Int represent the number of filters,
       kernal_size: a tuple of integer reprenst the size of the kernal,
       stride:  a tuple of integer reprenst the stride,
        input: represent the input for the PointWise layer
        alpha: float represent the shrinking parameter

X <- Conv2D(Int(alpha*filters), kernel_size=kernal_size, strides=stride)
(prev_layer_out)
X <- BatchNormalization()*(X)
X <- Activation('relu')*(X)
return X
```

Pseudocode (2)

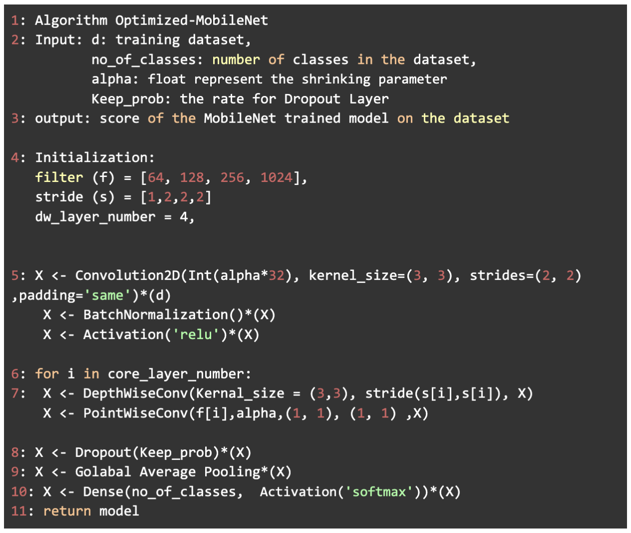
4.2.2: Reduced MobileNet Model:

The original model is complex and is suitable for large datasets like the ImageNet dataset that it was trained on. However, as we are using a different dataset and are limited by resources, the original model overfits and does not converge. As a result we reduced the complexity of the model to fit the dataset we chose (Imagenette, as specified in section 4.1).

Thus, the second implementation removes the 5 repetitive Depthwise Separable Layers with 512 filters. So, this model has 8 Depthwise Separable layers instead of 13. The source of the code can be found in our group github in the following link.

4.2.3: Optimized MobileNet Model:

From the second experiment and the good improvement we observed regarding the accuracy and the overfitting problem , we decided to do more reduction in the complexity of the model. The reduction has been done systematically by taking off layers which have redundant filters. The model has been optimized later by adding a dropout layer as a regularization technique before the pooling process. Moreover, the data augmentation is used for the data to avoid generalization and increase the validation accuracy. So, our model that we adapted for our project is illustrated in the Pseudocode (3). The source of the optimized model can be found in our group github in the following link.



```
1: Algorithm Optimized-MobileNet
2: Input: d: training dataset,
        no_of_classes: number of classes in the dataset,
        alpha: float represent the shrinking parameter
        Keep_prob: the rate for Dropout Layer
3: output: score of the MobileNet trained model on the dataset

4: Initialization:
   filter (f) = [64, 128, 256, 1024],
   stride (s) = [1,2,2,2]
   dw_layer_number = 4,


5: X <- Convolution2D(Int(alpha*32), kernel_size=(3, 3), strides=(2, 2)
,padding='same')*(d)
    X <- BatchNormalization()*(X)
    X <- Activation('relu')*(X)

6: for i in core_layer_number:
7:  X <- DepthWiseConv(Kernal_size = (3,3), stride(s[i],s[i]), X)
    X <- PointWiseConv(f[i],alpha,(1, 1), (1, 1) ,X)

8: X <- Dropout(Keep_prob)*(X)
9: X <- Golabal Average Pooling*(X)
10: X <- Dense(no_of_classes,  Activation('softmax'))*(X)
11: return model
```

Pseudocode (3)

5. Results

Section 5, is a demonstration of results we get from different experiments we conduct for this project along with some plots.

5.1 Project Results

5.1.1: Original Model Vs Reproduced Ones Model:

The experiment has been done using the original CNN and MobileNet model as proposed in the paper. The table below shows the number of parameters of the reproduced CNN and MobileNet in comparison with the result in the original paper [Table 4 in the paper]. We could observe the difference in the number of parameters generated by the two reproduced model as the number of parameters have been rescued by 7 times by using MobileNet. This almost close to proposed results in the paper.So, this prove that MobileNet is much lighter than the CNN in terms of the number of operations, Multi-Ads operations and faster which make it suitable for mobile applications.

Model	CNN	MobileNet
Paper: MobileNet-224	29.3(Million)	4.2 million
Replicated: MobileNet-224	28.4(Million)	3.4 million

Table (7): Original MobileNet

5.1.2: Reduced MobileNet Model:

On training the reduced model with tiny ImageNets, we were able to attain a validation accuracy of 0.46 in 15 epochs. But beyond this, the model started overfitting. Given that the dataset has 20 classes, this was a decent result. But we decided to train on Imagenettes and optimize the model further based on our observations. The result can be shown in the following link.

5.1.3: Optimized Model vs Equivalent CNN:

The table below shows the result of the optimized-MobileNet model that has been trained with 128 resolution and alpha one. Imagenette dataset vs the equivalent CNN model. The model was trained over 25 epochs and with batch size 32. The results as expected as accuracy is almost similar between optimized MobileNet and the equivalent CNN. Moreover, the number of parameters and the time duration to train the model in the MobileNet model is much less than the CNN.

	Training Accuracy	Validation Accuracy	Number of parameters	Time
Optimized MobileNet	0.9314	0.6892	0.73 million	7939.37s
Equivalent CNN	0.8620	0.6739	6.3 million	18454.90s

Table (8): MobileNet vs CNN

5.1.4: Optimized Model with Various Width Multiplier:

On changing alpha from 0.75 to 0.50 and 0.25 we observe that accuracy drops off as expected. These values are in accordance with the original results presented in the paper Figures (6a-d) But on reducing alpha from 1.0 to 0.75 there was an increase in accuracy [Table 9]. This could be because our current model is not the most optimal for Imagenettes. Since the dataset is small with only 10 classes, a simpler network could provide better training results. At an alpha of 0.75 the model is further simplified and becomes more optimal and hence more accurate.

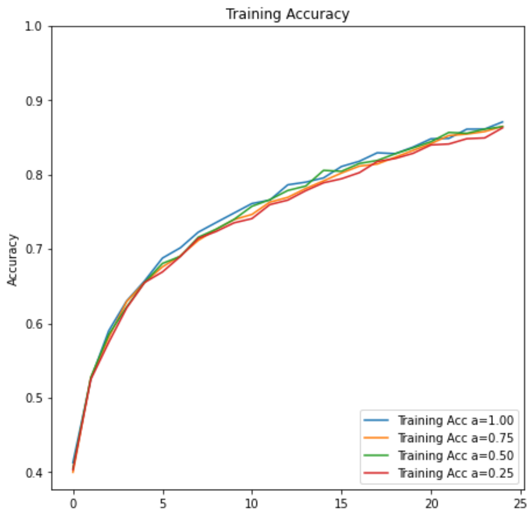


Figure (6a): Training accuracy for varying alpha

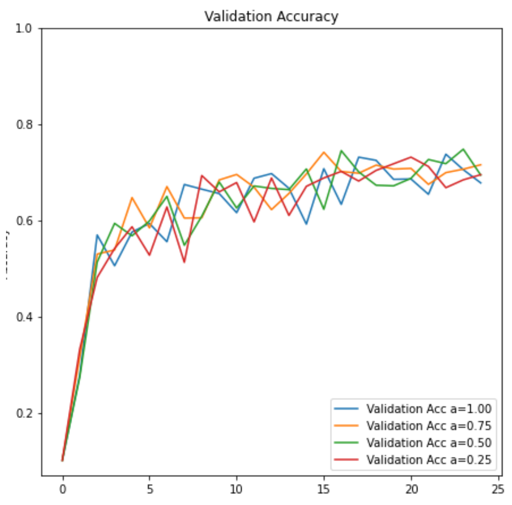


Figure (6b): Validation accuracy for varying alpha

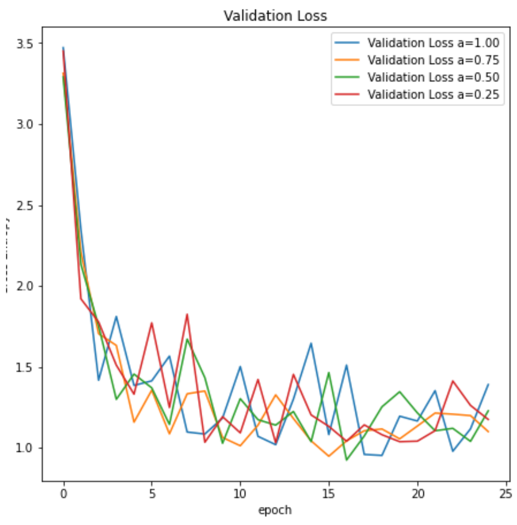


Figure (6c): Validation Loss for varying alpha

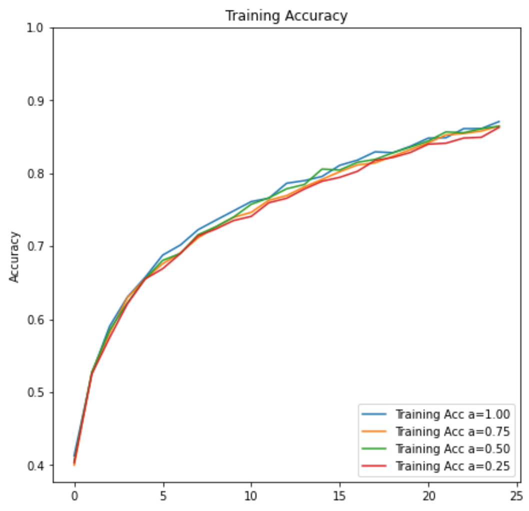


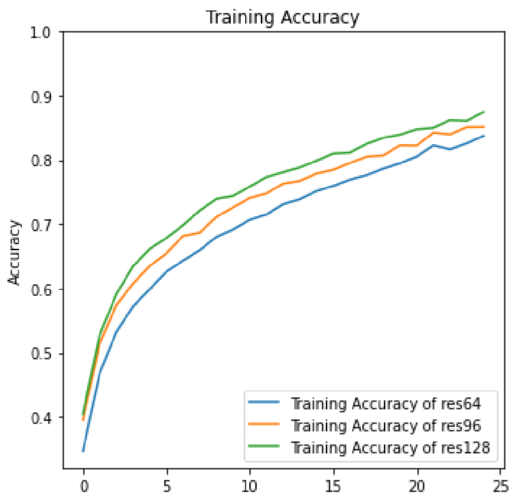
Figure (6d): Training Loss for varying alpha

Model	Training Accuracy	Validation Accuracy	Time
1.0 MobileNet 128	.8663	.6986	950s
0.75 MobileNet 128	.8437	.7205	950s
0.50 MobileNet 128	.7977	.7190	950s
0.25 MobileNet 128	.7111	.6629	950s

Table (9): MobileNet varying width multiplier

5.1.5: Optimized Model with Resolution Multiplier:

On varying the input resolution from 160 x 160 to 64 x 64, we observe faster training times for the smaller models. The accuracy of the model drops off with decreasing resolution.[Fig 8 a and b] For the first 3 resolutions, the drop off is minimal but on reducing the resolution to 64, we observe a significant drop in validation accuracy.[Table 11] Our findings align with the original paper in that reducing resolution creates a trade-off between the number of parameters and the accuracy of the model.



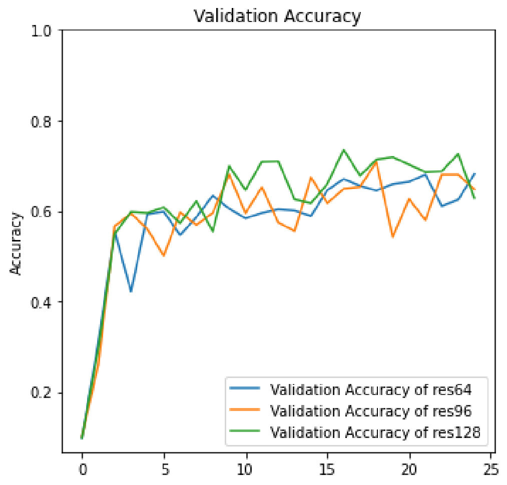


Figure (8a and b): Training and Validation Loss for varying resolutions

Model	Training Accuracy [25 epochs]	Validation Accuracy [25 epochs]	Time
MobileNet 160	0.8787	0.7060	10907s
MobileNet 128	0.8663	0.6986	1850s
MobileNet 96	0.8678	0.6945	625s
MobileNet 64	0.8425	0.6211	375s

Table (11): MobileNet with varying input resolutions

5.1.6: Optimized Model with AlexNet Model:

We compare the performance of our optimized MobileNet with an AlexNet model design with more parameters. Our model achieves 6% more accuracy than the AlexNet model while 25% smaller and using less compute than it. Our results follow the same trend as the original MobileNet but we do not obtain the performance to size ratio seen in the original paper.

Model [50 epochs]	Training Accuracy	Validation Accuracy	Num params
Optimized MobileNet 128	0.9494	0.7603	733,418
AlexNet 128	0.8350	0.7098	1,026,954

Table (12): MobileNet vs AlexNet

5.2 Discussion / Insights Gained

The results of our MobileNet and corresponding CNN demonstrate the great reduction in computation resources gained by implementing depthwise separable convolutions compared to a standard CNN. We also compared our models to the pre trained keras MobileMet to demonstrate our similarity in timing and total parameters. The results of these comparisons are shown in the table below (Table 13). We successfully show our standard network and MobileNet of equal depth have drastically different numbers of parameters. Our MobileNet is also shown to be faster than our standard convolutional neural network. Initially we were working with Stanford Dogs after we had downsized from Tiny ImageNet, but the models we were working with were not achieving satisfactory accuracy. Table 13 below shows the summary of the standard network (28,404,056 params) and the summary of our MobileNet (3,362,776 params).

Stanford Dogs	Runtime - 15 epochs (s)	Val accuracy	Params
Keras MobileNet	929.61117	0.4240	3,351,864
Our MobileNet	846.8893	0.1024	3,362,776
CNN	1060.9336	0.0418	28,404,05

Table (13): MobileNet on Stanford Dogs

We did learn a great deal from working with this dataset that made working with our even smaller Imagenette seamless. The MobileNet we replicated from the paper appeared to be overfitting the data as the training accuracy was very good and the validation accuracy plateaued at 0.1.

We tried reducing the number of layers to make the network less complex without success. We added a Gaussian noise layer with varying standard deviations as was done in the paper for this dataset [7]. We added a dropout layer at the end of the model with dropout rates of both 0.2 and 0.5. We tried running with different optimizers (Adam, Adamax, RMSprop) with varying learning rates (0.01, 0.001, 0.0001). We used data augmentation and resized the images to varying pixel sizes. Despite every approach we could not reach a good validation accuracy with the Stanford dogs dataset. Through research we found the dataset is very challenging as it is 120 classes and classifying the breed instead of the simpler object.

When we turned to Imagenette we had gone through the experimentation on Stanford dogs and had a direct and clear approach to modifying and optimizing the model for a smaller dataset.

6. Conclusion / Future Work

In this report we studied the performance of a MobileNet model on various datasets and compared the accuracy achieved when varying hyperparameters such as the input resolution and the depth (number of filters) of each layer on the network. The key takeaway from the experiments is the significant reduction in number of parameters that makes MobileNets much lighter than a comparable CNN that achieves a similar accuracy. The time taken to train the MobileNet on a CPU only system was low, which justifies the use of MobileNets in edge and embedded applications where resources are limited. We observed that even though our model is significantly shallower than the original MobileNet, we achieved a high accuracy by training for a relatively short time as the Imagenettes dataset has only 10 classes. When the resolution multiplier parameter was varied, we observed that the accuracy drops for lower resolutions. Varying the width multiplier gave more interesting results since we achieved highest accuracy for 0.75 model width. This is because the dataset is small and an optimal model for Imagenettes could be made smaller. For lower values of the width multiplier, accuracy drops as expected. From our experiments, we observe that there is scope to fine tune the models to further reduce the parameters.

Our existing model can be fine tuned by varying learning rates and varying filter sizes of existing layers, as was observed for the 0.75 width multiplier model.

Outside of this the blocks themselves can be modified. In particular, there is known literature on modified versions of MobileNets for achieving better accuracy. By replacing the MobileNet blocks with inverted residual blocks, ReLu activation layers in the range 0 to 6 and linear data transfer at bottlenecks of the residual block, MobileNets v2 obtains higher accuracy comparable to state of the art NasNets.[9] There is scope for us to further extend our experiments to include various other applications of image classifications and try attain higher accuracy using architecture proposed in the later versions of MobileNet

6. Acknowledgement

We would like to extend our sincere gratitude to every team member who put up great effort to prepare this report. We would especially like to thank Prof. Zoran Kostic and the TA team for their assistance in coordinating our study through their stimulating recommendations and support.

7. References

[1]github repo:
https://github.com/ecbme4040/e4040-2022Fall-Project-AHSN-am5964-hsp2122-ska2145

[2] Imagenette, Tensorflow.
https://www.tensorflow.org/datasets/catalog/imagenette (Dec 12 2022).

[3] A. Angioi, "Transfer learning made easy: let's build a dog breed classifier!" angioi.com, https://www.angioi.com/dog-breed-classification/ (Dec 14 2022)

[4] stanford_dogs, Tensorflow.
https://www.tensorflow.org/datasets/catalog/stanford_dogs (Dec 11 2022)

[5] Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto and Hartwig Adam. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." ArXiv abs/1704.04861 (2017): n. Pag.

[6] MobileNet source-code from Tenserflow:
https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.py

[7] Wang, W., Li, Y., Zou, T., Wang, X., You, J., & Luo, Y. (2020). A novel image classification approach via dense-mobilenet models. Mobile Information Systems, 2020, 1–8. https://doi.org/10.1155/2020/7602384

[8]I. Junejo, N. Ahmed, "Depthwise Separable Convolutional Neural Networks for Pedestrian." SN Computer Science. 2. 10.1007/s42979-021-00493-z (2021)

[9]Sandler, M., Howard, A., Zhu, M., et al. (2018) Mobilenetv2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, 18-23 June 2018, 4510-4520.https://doi.org/10.1109/CVPR.2018.00474

8. Appendix

Extended Results from Original Paper

2.2c Large Scale Geolocalizaton

The same MobileNet approach is applied to geolocalization. The original authors retrain PlaNet using the MobileNet architecture and show an impressive reduction in the number of parameters and operations, with a very slight decrease in performance compared to PlaNet and a significant outperformance compared to Im2GPS (Table 11).

2.2d Face Attributes

The original paper demonstrates a complementary relationship between MobileNet and distillation. Using distillation to train the classifier on the outputs of a larger model, as opposed to the ground truth labels, this model can be trained on infinite and unlabeled datasets. Table 12 shows the MobileNet based classifier can tolerate shrinking, achieves similar precision across attributes, and uses only 1% of mult-adds.

2.2e Object Detection

MobileNet is also explored with object detection and compared to VGG and Inception V2 under Faster-RCNN and SSD frameworks. These experiments were run using COCO data. MobileNet is shown to achieve similar results to both other models, again with significantly smaller complexity and size (Table 13).

2.2f Face Embeddings

The last experiment shown in the original paper is applied to facial recognition. The original authors used distillation to minimize the squared differences of the output of FaceNet and MobileNet during training (Table 14).

8.1 Individual Student Contributions in Fractions

	ska2145	hsp2122	am5964
Last Name	Alshemalis	Portes	Muraleekrishnan
Fraction of total contribution	1/3	1/3	1/3
What I did 1	Built MobileNet and CNN, Build the optimized MobileNetModel, test the MobileNet model Vs CNN	Gathered data sets, prepared for training	Update MobileNe with the two hyperparameterst, ran hyperparameter comparison, CPU/GPU timing analysis and AlexNet Comparison
What I did 2	Trained all datasets (Tiny ImageNet, Stanford Dogs, Imagenette) on models with modifications attempting to better results	Trained Stanford Dogs dataset on models with modifications attempting to better results	Trained 2 datasets (Tiny ImageNet, Imagenette) on models with modifications attempting to better results
What I did 3	Wrote sections of paper - section 3.2, intro section 4, section 4.2, intro section 5, 5.1.1, 5.1.2, 5.1.3, section 6	Wrote sections of paper - Abstract, Intro, section 2.1, 2.2, 4.1, 3.1.2.1: Data Challenges, 5.2, Appendix,	Wrote sections of paper - section 3.1, 5.1.4, 5.1.5, 5.1.6, Conclusion

Table (13): Individual Contributions