MobileNet: Efficient Convolutional Neural Networks for Mobile Vision Applications

E4040.2022Fall.XSWL.report
Yingqi Ma ym2926, Zhan Shu zs2584, Zihao Xiao zx2407
Columbia University

Abstract

For this project we want to reproduce the MobileNet structure of the original article and test its performance on image classification. We first reviewed the structure of the MobileNet model and the related computational principles in the original article. Then we successfully reproduced MobileNet and trained the model according to the original article. Next, we use two hyperparameters from the original paper to make the model smaller and faster. In the next section, we perform many sets of comparison experiments, including comparisons with some popular models, and our new model shows higher accuracy and fewer parameters in different cases. The conclusions we draw are generally consistent with the original paper and achieve our initial goal.

1. Introduction

Among the various deep neural network structures, convolutional neural network is the most widely used one, which was proposed by LeCun in 1989 [1]. Convolutional neural networks were successfully applied to handwritten character image recognition in the early days.The deeper AlexNet network [2] was a big success in ImageNet Challenge in 2012, and since then convolutional neural networks have flourished and been widely used in various fields. Increasingly deeper and complex neural networks have been built in order to achieve higher accuracy.The problem arises, however, in some real application scenarios such as mobile or embedded devices, such large and complex models are difficult to be applied. First, too large size of the models will cause insufficient memory, and second, these scenarios require low latency, or fast response time. Therefore, it is crucial to study small and efficient CNN models in these scenarios.

To cope with the above problems, the original paper built a neural network structure called MobileNet, which has both small and low latency properties.Two hyperparameters width multiplier and resolution multiplier are also defined to optimize the performance of MobileNet.

In this project, we intend to reproduce the MobileNet structure from the original article and optimize it for efficiency with two hyperparameters. The performance of our constructed model is compared with the performance of the original article model to analyze the reasons.

2. Summary of the Original Paper [3]

2.1 Methodology of the Original Paper

(1) As the original paper described, the basic unit of MobileNet is depthwise separable convolution. It can be decomposed into two smaller operations: depthwise convolution and pointwise convolution.(Figure 1.)

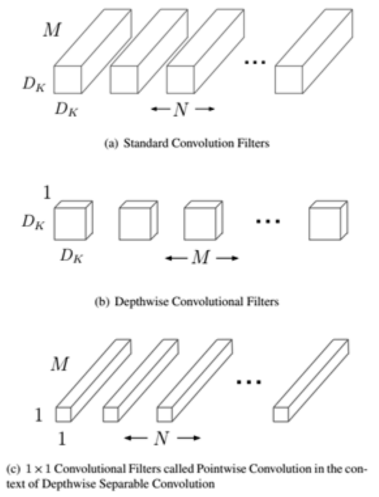

M
D_K
D_K
N
(a) Standard Convolution Filters
1
D_K
D_K
M
(b) Depthwise Convolutional Filters
M
1
1
N
(c) 1 × 1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 1. (a) Standard Convolution Filters; (b) Depthwise Convolutional Filters; (c) Pointwise Convolution Filters

Depthwise convolution differs from standard convolution in that for standard convolution the convolution kernels are used for all input channels, while depthwise convolution uses different convolution kernels for each input channel, with one convolution kernel corresponding to one input channel.The pointwise convolution is similar to the normal convolution, which uses a 1x1 convolution kernel.These two operations are shown in Figure 2.

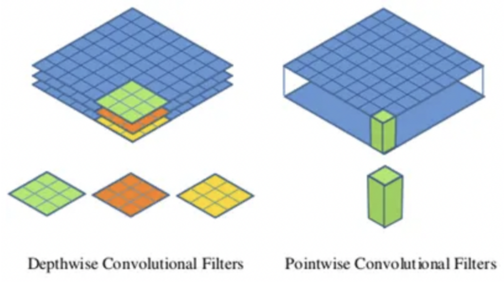

Depthwise Convolutional Filters
Pointwise Convolutional Filters

Figure 2. Depthwise Convolutional Filters and Pointwise Convolutional Filters

For depthwise separable convolution, firstly, depthwise convolution is used to convolve the different input channels separately, and then pointwise convolution is used to combine the output of the previous step, so that the overall effect is similar to a standard convolution, but the computation time and the number of model parameters are greatly reduced.

The difference between depthwise separable convolution and standard convolution in terms of computational effort:

They assuming that the input feature map size is:

$$D_F \times D_F \times M$$

The output feature map size is:

$$D_F \times D_F \times N$$

Among them D_F is the width and height of the feature map.

It is also assumed here that the input and output feature map sizes (width and height) are the same.

For the standard convolution the computational effort will be:

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

For depthwise convolution its computation will be:

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

The pointwise convolution computation will be:

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

So the total computation of depthwise separable convolution will be:

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

Comparing the depthwise separable convolution with the standard convolution is as follows:

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

In general, if N is large, the depthwise separable convolution can reduce the computational effort by about 8-9 times compared to the standard convolution if a 3x3 convolution kernel is used.

(2) MobileNet Network Structure:

In the real application depthwise separable convolution will join batchnorm and use ReLU activation function, so the basic structure of depthwise separable convolution in the original paper is shown in Figure 3 below:

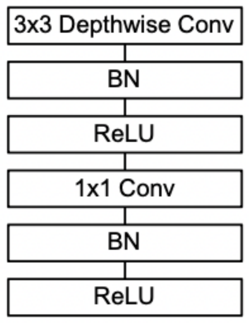


Figure 3. Basic structure of depthwise separable convolution

The network structure of MobileNet is shown in Table 1:

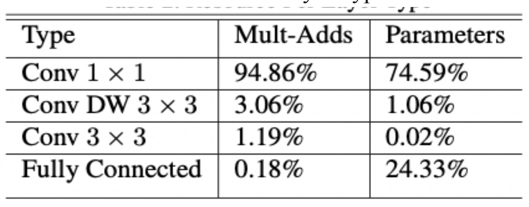
Table1. MobileNet Body Architecture

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 first is a standard convolution of 3x3, followed by a multilayer depthwise separable convolution. then average pooling is used to turn the feature into 1x1, a fully connected layer is added according to the predicted class size, and finally a softmax layer.

The distribution of parameters and computational effort for the whole network, is shown in Table 2：

Table2. Resource Per Layer Type



Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

It can be seen that the entire computational effort is basically concentrated on the 1x1 convolution; for the parameters are also mainly concentrated in the 1x1 convolution, in addition to that there is a fully connected layer accounting for part of the parameters.

(3) MobileNet Thinning：In order to further reduce the benchmark model of MobileNet, MobileNet needs to be slimmed down.

The original article introduces two hyperparameters: width multiplier and resolution multiplier. the first parameter, width multiplier, is mainly to reduce the number of channels proportionally, which is denoted as α, and its value range is (0,1], then the number of input and output channels will become αM and αN , for depthwise separable convolution, the computation becomes：

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

The second parameter, resolution multiplier, mainly reduces the size of the feature map proportionally, and is denoted as ρ , for example, the original input feature map is 224x224, which can be reduced to 192x192, together with the resolution multiplier, the computation of depthwise separable convolution will be:

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

2.2 Key Results of the Original Paper

The original paper proposed a new model architecture called MobileNets based on depthwise separable convolutions. The main results can be concluded as: (1) Using depthwise separable convolutions compared to full convolutions only reduces small accuracy in ImageNet, but saving tremendously on mult-adds(computational times) and parameters; (2) At similar computation and number of parameters, that making MobileNets thinner(using width multiplier) is better than making them shallower(using less layers) in accuracy; (3) Compared with popular model such as Google Net and VGG16, MobileNet shows nearly the same accuracy while smaller size and less computation time. (4) MobileNet has shown efficient performance in a variety of practical scenarios (Face Attributes, Object Detection, Face Embeddings).

3. Methodology (of the Students' Project)

In this project, we reproduce the MobileModel model of the original article by code and apply two hyperparameters provided in the article to shrink models. And we test the image classification performance of the MobileModel model. Due to the large workload of the original article, this project only reproduces parts of the original article, excluding the second half of the project, which includes practical applications such as facial recognition and object recognition. Unlike the original article, this project uses CIFAR-100 from Keras as the dataset. And we made some changes to the MobileModel model by adding dropout layers to alleviate overfitting conditions. So the final result is a bit different from the original article.

The main platform we use is GCP, and the libraries we use are TensorFlow[4], numpy, matplotlib.

3.1. Objectives and Technical Challenges

3.1.1 Objectives:

Building a Depthwise separable convolutional layer with code and reproduce the structure of MobileModel; Applying two hyperparameters, width multiplier and resolution multiplier, to shrink the model and explore the changes in computation time and accuracy; Comparing the performance of other models in terms of computation time, number of parameters and accuracy; Comparing and analyze the differences between the results and the original article; Summarizing the use of MobileModel and the proposed conditions of application.

3.1.2 Technical Challenge:

MobileModel is a neural network structure with nearly 30 layers. Building such a huge network structure with code requires understanding the structure of Depthwise separable convolution net. And the principle of MobileNet involves a lot of mathematical formulas;

Due to the large data set and limited computation time, we had to select only the first 5000 data of CIFAR-100 for the training test. This means that our algorithm may be trained with noisy and biased data, which will affect its overall performance;

In our actual training process, overfitting often occurs, and we have to try various methods such as regularization and adding dropout layers to solve this problem, which takes a lot of time.

3.2.Problem Formulation and Design Description

The following flow chart (Figure4) illustrates the process of this project, and the pseudo-code for each step of the software part will be detailed in the Software Design section in 4.2.

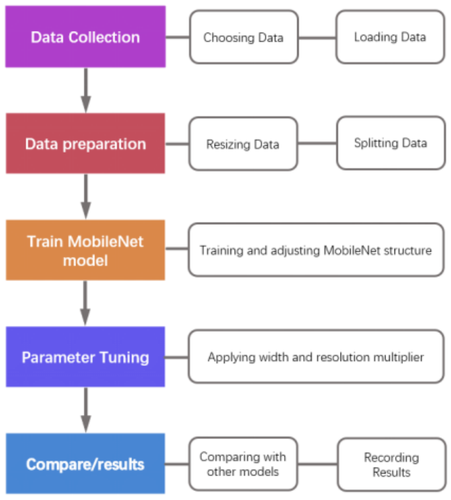


Figure 4.. Flowchart of the process.

(1) Data Collection: We download the data set from keras.datasets and load it.

(2) Data preparation:After importing the data, we select part of them as our training set and test set. Then resize the image size to (224,224,3).

(3) Train MobileNet model: We built the structure of MobileNet by referring to the original article in the previous section, and started training. However, the initial performance results were poor, and the validation set correct rate did not increase with the increase of epoch, but kept fluctuating between some values. We speculated that this is caused by overfitting. Trying to fix this, we added a dropout layer before the last convolutional layer of the model, and this operation is consistent with the original one. But the training results were still unsatisfactory after we tried many parameters. Finally, we decided to apply dropout layers to the whole model. We added a dropout layer after each combination of Depthwise and Pointwise.

(4) Parameter Tuning: To shrink the MobileNet architecture, we apply two global-hyper parameters(width multiplier α and resolution) to the model.

We perform two sets of experiments with reference to the original article. In the first group, we set α to 1, 0.75, 0.5, and 0.25 to test the performance of the model in terms of computation time, accuracy, and number of parameters, respectively. In the other group, we applied the resolution settings of 224, 192, 160, and 128 to the model and recorded the same three values. Next we compare thinner models with width multiplier to shallower models using less layers.

(5) Compare/results: In this section we compare the trained MobileNet model with other models. First we compare MobileNet with depthwise separable convolutions with full convolutions. Second we compare MobileNet model with a widely used VGG16 and Inception V3[5] model. We make a comparison of the accuracy, computation time and number of parameters. Finally, we record all the above results for a summary analysis.

4. Implementation

4.1 Data

The data we use is from CIFAR100 small images classification dataset, which contains 50,000 32x32 color training images and 10,000 test images, labeled over 100 fine-grained classes that are grouped into 20 coarse-grained classes [6]. Because of the memory limitation of Google Cloud Platform and to shorten the training time of the model, we have decided to pick only the first 10 classes, which results in 5,000 32x32 color training images and 1,000 test images.

Since the input shape of MobileNet needs to be (224, 224, 3), (192, 192, 3), (160, 160, 3) or (128, 128, 3), we resize each image to the above sizes depending on different experiments.

4.1 Implementation of MobileNet

The MobileNet we implemented contains 5 parameters:

1. alpha: which corresponds to the width multiplier as stated in the original paper. alpha is a number between 0 and 1. $\alpha = 1$ is the baseline MobileNet and $\alpha < 1$ are reduced MobileNets [3]. The default alpha is set to 1.
2. imgsize: imgsize represents the shape of input images. For example, if each input image has a shape of (224, 224, 3), then imgsize should be set to 224. The parameter has a similar function as the resolution multiplier ρ in the original paper [3], which shrinks the input resolution of the network. The default imgsize is set to 224.
3. num_classes: specifies the number of classes of the data. The default num_class is set to 1000.
4. dropout_rate: specifies the dropout rate of each dropout layer. The default dropout_rate is set to 1e-3.
5. shallow: a boolean to determine whether a shallower MobileNet should be built. When shallow is set to True, 5 hidden depthwise convolution layers with filter shape 3x3x512 and 5 convolution layers with filter shape 1x1x512x512 will be excluded from the model, and a shallow MobileNet would be built. The default shallow parameter is set to False.

The MobileNet structure is built on depthwise separable convolutions except for the first layer which is a full convolution. Each depthwise separable convolution contains a 3x3 depthwise convolution layer, followed by batchnorm and ReLU, and a 1x1 pointwise convolution layer, followed by batchnorm, ReLU and Dropout.

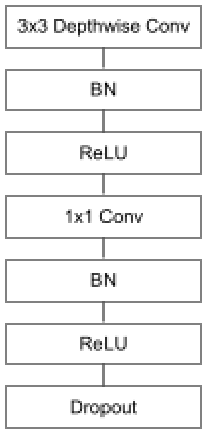


Figure 5. Basic structure of depthwise separable convolution in this project.

The structure of the MobileNet is shown in Table 3. The first layer is a 2x2 pointwise convolution layer, followed by 13 depthwise separable convolution blocks. Then the output is fed to a Global Average Pooling layer to reduce the spatial resolution to 1. Although it is not stated in the original paper, we took a look at the source code of MobileNet and found that the Global Average Pooling Layer is followed by a Reshape layer which reshapes the input to a size of 1 x 1 x 1024. Then a Dropout Layer is added with default dropout rate set to 1e-3 and a pointwise convolution layer with number of filters equal to number of classes. Finally, another Reshape layer is added to reshape the input to match the number of classes and an Activation layer at the output layer. Excluding batchnorm, ReLU and Dropout layers, the MobileNet we implemented has a total of 32 layers, which is the same as the MobileNet structure described in the original paper.

Table 3. Main Structure of MobileNet

Type / Stride	**Output Shape**	**Filter Shape**
Conv / s2	112 × 112 × 32	3 × 3 × 3 × 32
Conv dw / s1	112 × 112 × 32	3 × 3 × 32 dw
Conv / s1	112 × 112 × 64	1 × 1 × 32 × 64
Conv dw / s2	56 × 56 × 64	3 × 3 × 64 dw
Conv / s1	56 × 56 × 128	1 × 1 × 64 × 128
Conv dw / s1	56 × 56 × 128	3 × 3 × 128 dw
Conv / s1	56 × 56 × 128	1 × 1 × 128 × 128
Conv dw / s2	28 × 28 × 128	3 × 3 × 128 dw
Conv / s1	28 × 28 × 256	1 × 1 × 128 × 256
Conv dw / s1	28 × 28 × 256	3 × 3 × 256 dw
Conv / s1	28 × 28 × 256	1 × 1 × 256 × 256
Conv dw / s2	14 × 14 × 256	3 × 3 × 256 dw
Conv / s1	14 × 14 × 512	1 × 1 × 256 × 512
Conv dw / s1	14 × 14 × 512	3 × 3 × 512 dw
Conv / s1	14 × 14 × 512	1 × 1 × 512 × 512
Conv dw / s1	14 × 14 × 512	3 × 3 × 512 dw
Conv / s1	14 × 14 × 512	1 × 1 × 512 × 512
Conv dw / s1	14 × 14 × 512	3 × 3 × 512 dw
Conv / s1	14 × 14 × 512	1 × 1 × 512 × 512
Conv dw / s1	14 × 14 × 512	3 × 3 × 512 dw
Conv / s1	14 × 14 × 512	1 × 1 × 512 × 512
Conv dw / s1	14 × 14 × 512	3 × 3 × 512 dw
Conv / s1	14 × 14 × 512	1 × 1 × 512 × 512
Conv dw / s2	7 × 7 × 512	3 × 3 × 512 dw
Conv / s1	7 × 7 × 1024	1 × 1 × 512 × 1024
Conv dw / s1	7 × 7 × 1024	3 × 3 × 1024 dw
Conv / s1	7 × 7 × 1024	1 × 1 × 1024 × 1024
Global Avg Pool	1024	
Reshape	1 × 1 × 1024	
Dropout	1 × 1 × 1024	
Conv / s1	1 × 1 × 1000	1 × 1 × 1024 × 1000
Reshape	1000	
Softmax	1000	Classifier

4.2 Software Design

We have implemented a total of 4 experiments:

1. Compare accuracy, size and computation trade offs by shrinking the MobileNet architecture with the width multiplier alpha [3].
   Algorithm:
   1) Fetch CIFAR100 dataset.
   2) Pick training data and validation data with labels from 0-9.
   3) Resize each image to a shape of (224, 224, 3).
   4) Create a MobileNet with alpha = 1, imgsize = 224, num_classes = 10, dropout_rate = 0.1.
   5) Train the model and plot training accuracy and validation accuracy vs. number of epochs.
   6) Create a MobileNet with alpha = 0.75, imgsize = 224, num_classes = 10, dropout_rate = 0.1.
   7) Train the model and plot training accuracy and validation accuracy vs. number of epochs.
   8) Create a MobileNet with alpha = 0.5, imgsize = 224, num_classes = 10, dropout_rate = 0.1.
   9) Train the model and plot training accuracy and validation accuracy vs. number of epochs.
   10) Create a MobileNet with alpha = 0.25, imgsize = 224, num_classes = 10, dropout_rate = 0.1.
   11) Train the model and plot training accuracy and validation accuracy vs. number of epochs.

2. Compare accuracy, size and computation trade offs for different resolution multipliers by training the MobileNet with reduced input resolutions [3].
   Algorithm:
   1) Resize each image to a shape of (224, 224, 3).
   2) Create a MobileNet with alpha = 1, imgsize = 224, num_classes = 10, dropout_rate = 0.1.
   3) Train the model and plot training accuracy and validation accuracy vs. number of epochs.
   4) Resize each image to a shape of (192, 192, 3).
   5) Create a MobileNet with alpha = 1, imgsize = 192, num_classes = 10, dropout_rate = 0.1.
   6) Train the model and plot training accuracy and validation accuracy vs. number of epochs.
   7) Resize each image to a shape of (160, 160, 3).
   8) Create a MobileNet with alpha = 1, imgsize = 160, num_classes = 10, dropout_rate = 0.1.
   9) Train the model and plot training accuracy and validation accuracy vs. number of epochs.
   10) Resize each image to a shape of (128, 128, 3).
   11) Create a MobileNet with alpha = 1, imgsize = 128, num_classes = 10, dropout_rate = 0.1.
   12) Train the model and plot training accuracy and validation accuracy vs. number of epochs.

3. Compare thinner MobileNet with alpha = 0.75 to shallower MobileNet with less layers [3].
   Algorithm:
   1) Resize each image to a shape of (224, 224, 3).
   2) Create a MobileNet with alpha = 0.75, imgsize = 224, num_classes = 10, dropout_rate = 0.1.
   3) Train the model and plot training accuracy and validation accuracy vs. number of epochs.
   4) Create a MobileNet with alpha = 1, imgsize = 224, num_classes = 10, dropout_rate = 0.1, shallow = True.
   5) Train the model and plot training accuracy and validation accuracy vs. number of epochs.

4. Compare full MobileNet with other popular models such as VGG16 and InceptionV3.
   Algorithm:
   1) Resize each image to a shape of (224, 224, 3).
   2) Create a base model of VGG16 with input shape of (224, 224, 3), weights = ‘imagenet’ and include_top = False
   3) Set the layers in the base model as untrainable.
   4) let x = output of base layer
   5) Add a Flatten layer with input of x.
   6) Add a Dense layer with num_filters = 4096 and activation function = ‘relu’.
   7) Add another Dense layer with num_filters = 1024 and activation function = ‘relu’.

8) Add a dropout layer with dropout rate = 0.2.
9) Add an output layer with 10 classes and activation function = ‘softmax’.
10) Train the model and plot training accuracy and validation accuracy vs. number of epochs.
11) Create a base model of InceptionV3 with input shape of (224, 224, 3), weights = ‘imagenet’ and include_top = False
12) Set the layers in the base model as untrainable.
13) let x = output of base layer
14) Add a GlobalAveragePooling layer with input of x.
15) Add a Dense layer with num_filters = 1024 and activation function = ‘relu’.
16) Add a dropout layer with dropout rate = 0.2.
17) Add an output layer with 10 classes and activation function = ‘softmax’.
18) Train the model and plot training accuracy and validation accuracy vs. number of epochs.

Link to code in github:

1. https://github.com/ecbme4040/e4040-2022fall-project-xswl-ym2926-zs2584-zx2407/blob/main/mobilenet.py
2. https://github.com/ecbme4040/e4040-2022fall-project-xswl-ym2926-zs2584-zx2407/blob/main/E4040_project_fianlmodel.ipynb

5. Results

5.1 Project Results

The results we observed are shown in the following tables. First we compare shrinking the architecture of the model in different ratios. Table 4 shows the comparison between different width multipliers in training accuracy, validation accuracy, number of parameters and overall training time. We change the width multiplier alpha of the model from 1 to 0.25 every 0.25. As shown in the table, Training accuracy drops down fast from 0.5 to 0.25. However, the training time does not save a lot. Whereas the comparison between 1.0 and 0.75 is not as we expected. The training and validation accuracy is higher than a thicker model. The training and validation accuracy plot of every epoch for each model is shown in the appendix 10.2.

Table 4. MobileNet Width Multiplier Compare

Width Multiplier	Train accuracy (%)	Validation accuracy (%)	Million param	Train time (s)
1.0-224	96.25	74.00	3.25	1455
0.75-224	96.28	74.10	1.85	1124
0.5-224	94.14	73.7	0.84	803
0.25-224	89.08	70.9	0.22	714

Table 5 below shows the comparison between different resolutions. We train the model with different image sizes of 224, 192, 160, 128 and compare their performance. We can observe that for the top three rows of data, it decreases slowly as resolution drops. However, for the 128 image size model, the validation accuracy happens to be the highest. This is due to the fact that we want to make sure the model converges so we train 50 epochs for the model. Therefore, it is reasonable to have the highest accuracy.

Table 5. MobileNet Resolution Compare

Resolution	Train accuracy (%)	Validation accuracy (%)	Million param	Train time (s)
1.0-224	96.25	74.00	3.25	1455
1.0-192	95.13	74.6	3.25	1054
1.0-160	95.3	73.6	3.25	794
1.0-128	95.00	75.3	3.25	714

Table 6 is the comparison between the model with 0.75 width multiplier to the shallower model with less layers. This also shows the same four aspects of the model, and we want to observe the trade-off between accuracy and parameters number. When building the model we set shallow = True to make the model shallower. The observation shows that both accuracy are 1% higher in shallow MobileNet, but the 0.75 width model has lower training time and parameters.

Table 6. 0.75 width multiplier vs. shallow MobileNet

Model	Train accuracy (%)	Validation accuracy (%)	Million param	Train time (s)
0.75-224	96.28	74.10	1.85	1124
Shallow MobileNet	97.32	75.80	1.89	1184

Table 7 below shows the comparison between MobileNet and VGG16 [3]. This shows the train and validation accuracy and parameter numbers and train time. As shown in the table, the validation accuracy is 3.2% higher than MobileNet but for this 3% accuracy it costs 121 millions of parameters and this is because we only run 20 epochs for this or it may stop the kernel. Therefore, MobileNet is still an ideal model as we expected. InceptionV3 has the highest accuracy and lowest training time. MobileNet has achieved almost the same training accuracy as VGG16 and InceptionV3, but validation accuracy is still lower than InceptionV3 13.9%.

Table 7. MobileNet vs. VGG16 vs. InceptionV3

Model	Train accuracy (%)	Validation accuracy (%)	Million param	Train time (s)
1.0-224	96.25	74.00	3.25	1455
VGG16	96.85	77.2	121.68	612
InceptionV3	99.21	87.9	23.91	304

5.2 Comparison of the Results Between the Original Paper and Students' Project

In this section, we are going to compare our results with the original research. We discovered some interesting results and made several progress.

Table 8 [3] below is the comparison between the original performance of the model and our new model as the width multiplier varies. Our new model has all accuracy higher than the original work. And we also lower the parameter numbers to lower the run time of the model. Now for each epoch of the model, the run times are between 10s-50s.

Table 8. Original MobileNet vs. New MobileNets

Width Multiplier	Original accuracy (%)	Validation accuracy (%)	Original Params (million)	New Params (million)
1.0-224	70.6	74.00	4.2	3.25
0.75-224	68.4	74.10	2.6	1.85
0.5-224	63.7	73.7	1.3	0.84
0.25-224	50.6	70.9	0.5	0.22

Table 9 [3] below is the comparison between original and new model result observation of the model as the image size varies. We can observe that all the accuracy over these four image sizes are higher in our validation accuracy column. And we also minimize the number of parameters.

Table 9. Original MobileNet vs. New MobileNet Resolution

Resolution	Original accuracy (%)	Validation accuracy (%)	Original Params (million)	New Params (million)
1.0-224	70.6	74.00	4.2	3.25
1.0-192	69.1	74.6	4.2	3.25
1.0-160	67.2	73.6	4.2	3.25
1.0-128	64.4	75.3	4.2	3.25

Table 10 [3] below is the comparison between original and new 0.75 MobileNet and Shallow MobileNet. We discover different results of the two models. In the original research, they state that shallow MobileNet has lower accuracy, but ours has higher. This may happen because the dataset we use is smaller than the original study, therefore, different models may perform differently.

Table 10. Original vs. New 0.75 MobileNet/Shallow

Model	Original accuracy (%)	New accuracy (%)	Original Params (million)	New Params (million)
0.75-224	68.4	74.10	2.6	1.85
Shallow MobileNet	65.3	75.80	2.9	1.89

Table 11 below is the comparison between original and new comparison between MobileNet and VGG16 and InceptionV3. We can observe that the total trend of these three comparisons is the same. Which means our model is ideal. Accuracy is near VGG16 and far from InceptionV3. We have less parameters so the overall training time is saved.

Table 11. Original vs. New 0.75 MobileNet/Shallow

Model	Original accuracy (%)	New accuracy (%)	Original Params (million)	New Params (million)
1.0-224	70.6	74.00	4.2	3.25
VGG16	71.5	77.2	138.0	121.68
InceptionV3	84.0	87.9	23.2	23.91

5.3 Discussion / Insights Gained

In the original paper, they did not specify the learning rate and dropout rate for our model. To optimize the model's performance, we adjusted the hyper parameters of the model. We use a learning rate of 1e-3 and a dropout rate of 0.1.

Another major change to the structure is that instead of adding only one dropout layer as previous work did, we add a dropout layer after each convolution. Therefore, the overfitting problem has decreased more.

We used 40 epochs for every model we trained to let the model converge and not wait too long to process.

For the reason our results and performance are better, the major cause is the dataset we use is smaller than the original dataset. We have only tested ten classes of the data. To test the whole dataset is not applicable for our computer right now, so currently we are using the optimal choices and parameters we have.

We were facing two major problems during the modeling process. The first one is long run time when training the model. Initially, the model only has one dropout layer, the run time is over 300s per epoch. Furthermore, the model is heavily overfitting when we apply the dataset. Therefore, we add more dropout layers. Now the run time decreases drastically on GCP. Second problem we were facing is the large volume of the dataset. We first tested the dataset with 20 classes and the kernel just crashed and disconnected. So eventually we tested 10 classes and adjusted the parameters to make the model run properly.

The insight of the project is the fast training speed and high validation accuracy. We improve the model and fit the dataset better with the adjustment of hyper parameters and reshaping of the data. Our model also saves parameters to further increase the learning speed.

6. Future Work

The experiments we did in this project is based on a relatively small dataset with 10 classes. A possible improvement would be to train the MobileNet on a larger dataset and compare our results with the original paper. Furthermore, more experiments on parameter tuning could be conducted, for example, adjust the learning rate of the optimizer and the dropout rate, try to add regularization to some layers, add or remove some dropout layers, in order to accelerate the training process, prevent overfitting, and produce a better result.

7. Conclusion

In this project we reproduce the MobileNet structure based on depthwise separable convolutions of the original article. We also use the two hyperparameters from the original article to make the model smaller and faster. We then performed many sets of comparison experiments, including comparisons with some popular models, and the new model we built showed higher accuracy and fewer parameters in different cases. The conclusions we reached were generally consistent with the original paper and met our expected goals. Through the project we learned to effectively trade-off model accuracy, computation time, and storage space, which has a wide range of real-world applications. We believe that Mobile Net still has great potential in the field of image vision, and future research can focus on how to further improve the accuracy of the model while keeping a low number of parameters.

8. Acknowledgement

During this project, we want to express our acknowledgement to the TAs and professor. It is your help that this project could stay on track. We also want to thank the previous research team. Their paper has been a huge inspiration on our project and we reference their paper a lot. In addition, online resources also give us some different angle solutions for the problems we are facing. Thank you for helping on the project.

9. References

[0] Project github link: https://github.com/ecbme4040/e4040-2022fall-project-xswl

[0] Papers link: A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for Mobile Vision Applications," *arXiv.org*, 17-Apr-2017. [Online]. Available: https://arxiv.org/abs/1704.04861. [Accessed: 18-Dec-2022].

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[2] J. E. Espinosa, S. A. Velastin, and J. W. Branch, "Vehicle detection using Alex Net and faster R-CNN deep learning models: A comparative study," *Advances in Visual Informatics*, pp. 3–15, 2017.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for Mobile Vision Applications," *arXiv.org*, 17-Apr-2017. [Online]. Available: https://arxiv.org/abs/1704.04861. [Accessed: 18-Dec-2022].

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv.org*,

16-Mar-2016. [Online]. Available: https://arxiv.org/abs/1603.04467. [Accessed: 18-Dec-2022].

[5] J. Krause, H. Jin, J. Yang, and L. Fei-Fei, "Fine-grained recognition without part annotations," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

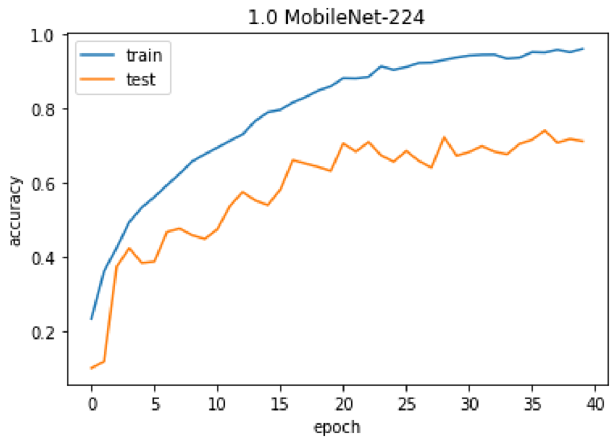
[6] https://keras.io/api/datasets/cifar100/

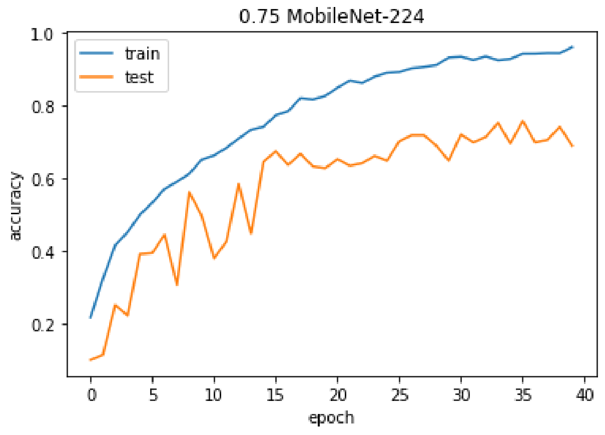
10. Appendix

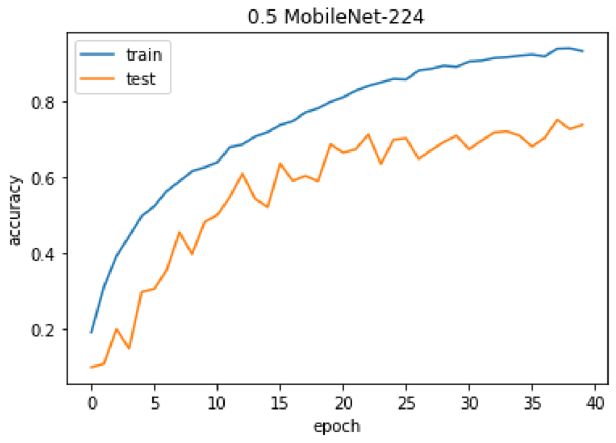
10.1 Individual Student Contributions in Fractions

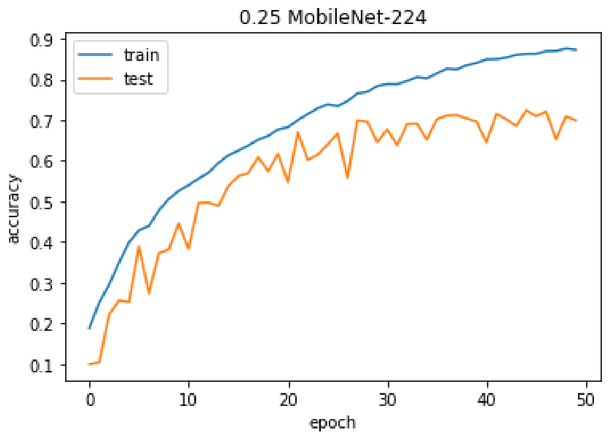
	ym2926	zs2584	zx2407
Last Name	Ma	Shu	Xiao
Fraction of (useful) total contribution	25%	40%	35%
What I did 1	Report: Results demonstration, Acknowledgement, Appendix plot.	Code: Implement MobileNet structure and test code	Report: Abstract, Introduction, Summary of the Original Paper, Methodology, Conclusion
What I did 2	Help to tuning parameters and training models.	Parameter tuning and adjustment of MobileNet architecture	Help to parameter tuning and training mode; README file.
What I did 3		Report: Implementation, Future works	

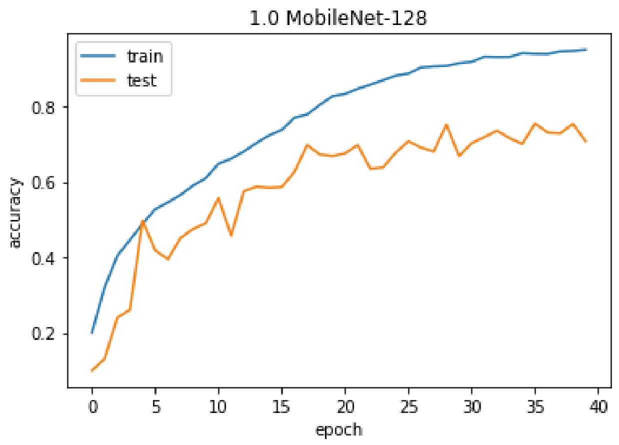
10.2 Support Material

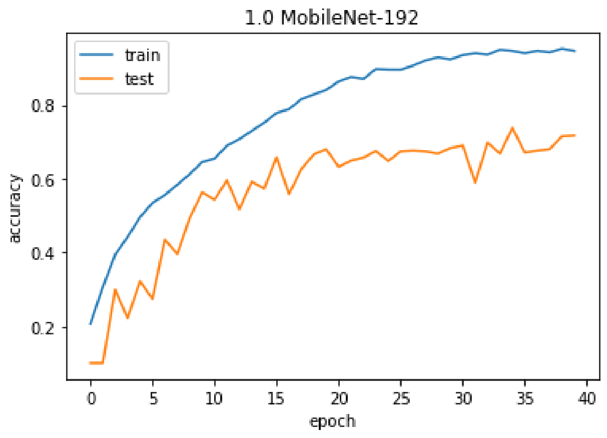


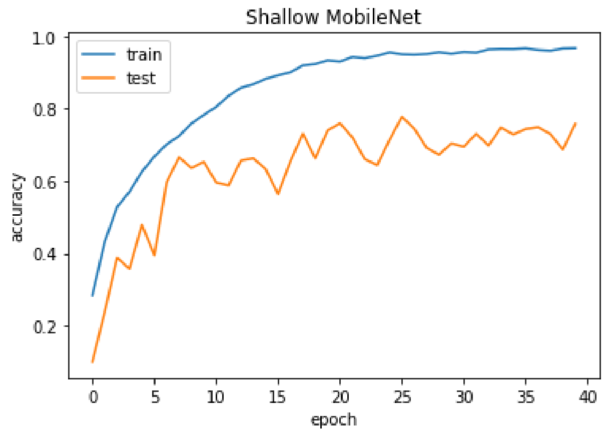


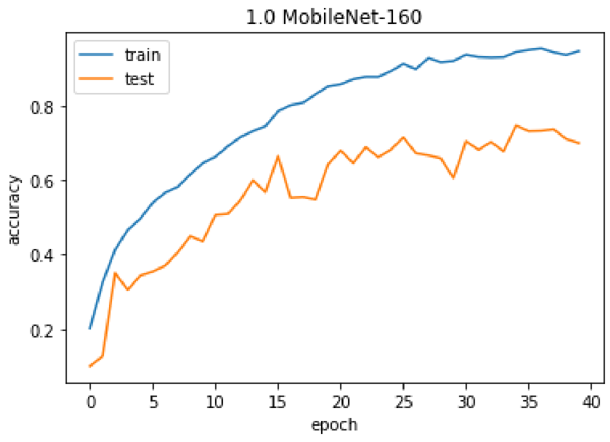


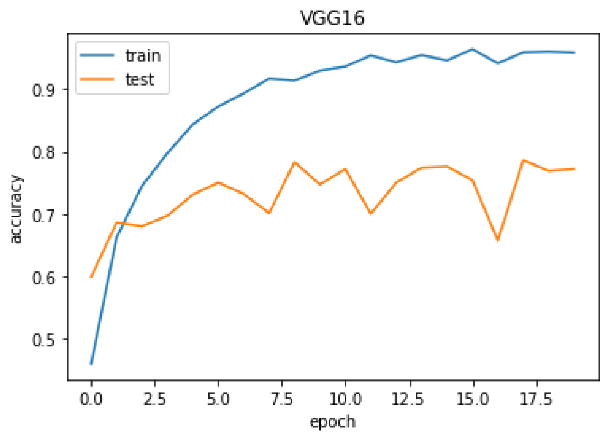

0.25 MobileNet-224
train
test
accuracy
epoch

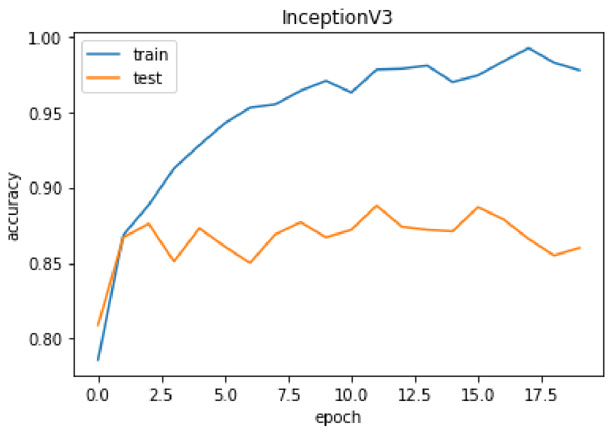

1.0 MobileNet-128
train
test
accuracy
epoch


1.0 MobileNet-192
train
test
accuracy
epoch


Shallow MobileNet
train
test
accuracy
epoch


1.0 MobileNet-160
train
test
accuracy
epoch


VGG16
train
test
accuracy
epoch



InceptionV3
train
test
accuracy
epoch
1.00
0.95
0.90
0.85
0.80
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5