A Deep Dive into MobileNet: Mathematical Explanation and Experiments

Xin Huang
Department of Electrical Engineering
Columbia University
New York, NY 10025
xh2510@columbia.edu

Qimeng Tao
Department of Electrical Engineering
Columbia University
New York, NY 10025
qt2139@columbia.edu

Andreas Knaupp
Department of Biomedical Engineering
Columbia University
New York, NY 10025
andreas.knaupp@columbia.edu

Ling Sun
Department of Data Science
Columbia University
New York, NY 10025
ls3759@columbia.edu

Abstract

What is the key that makes MobileNet a mainstream for mobile vision with fewer parameters but great performance? A general answer is the use of depth-wise separable convolutions, inverted residual blocks, and squeeze and excitation blocks. These elements play a critical role in building lightweight, low latency deep neural networks for mobile and embedded devices.

In this paper, we look into mathematical foundations for depth-wise separable convolutions and inverted residual blocks. We state out theorems and provide explanations of the key elements in MobileNet and conduct experiment for image classification task on CIFAR-10, Flower and other datasets.

The experiment results show that MobileNet achieves considerate accuracy in image classification tasks comparing to other neural networks with greater parameters, which justify its ability of maintaining good performance on mobile devices with less computational resource and less energy consumption.

Keywords: MobileNet, Depthwise Separable Convolution, Residual Block, PCA, Squeeze-and-excitation block, ODE, Adjoint Models

1 Introduction

Convolutional Neural Networks (CNN) have been widely studied and used in the field of computer vision and have achieved good results (1). In recent years, scholars generally use CNN to deal with problems such as image recognition and use deeper models to fit more complex recognition problems or classification problems. It can be seen that in order to pursue classification accuracy, the model depth is getting deeper and deeper, and the model complexity is getting higher and higher. Beginning with the VGG network architecture (2), scholars have begun to pay extensive attention to the excellent effects of large models, VGG builds a deep convolutional neural network by using a series of small convolution kernels and pooling layers of size 3×3, and achieves good results. The VGG model is widely appreciated by researchers because of its simple structure and strong applicability, especially its network structure design method, which provides a direction for building deep neural networks.

The emergence of ResNet (3) has also made the training of deep models easier. Based on the larger and larger model structure and the introduction of more novel concepts, with the support of sufficient

computing power, scholars have achieved better and better results in the classification of ImageNet and other datasets.

However, in some real application scenarios, such as mobile or embedded devices, a large and complex model is challenging to apply. The first is that the model is too large, and the mobile devices face the problem of insufficient memory. Secondly, these scenarios require low latency, fast response speed, or low consumption of resources (power). The considerable model and parameters make the traditional CNN The network structure does not work on the mobile side.

Based on this, research on small and efficient CNN models is crucial in these scenarios, at least for now, although hardware will get faster and computational resource will become greater in the future. The current research is divided into two directions: to compress the complex neural network model to become a smaller one; the other is to design and train the small model directly. Regardless, the goal is to reduce model size while maintaining model performance and increasing model speed.

MobileNets (4)(5)(6), which will be discussed in this project, is an efficient model for mobile and embedded devices based on an architecture that uses depthwise separable convolutions to build light weight deep neural networks (4). The model introduces two simple global hyperparameters that effectively balance size and accuracy. These two hyperparameters allow us to choose an appropriately model size corresponding to their application based on the constraints of the problem. The authors conduct extensive experiments on resource and accuracy trade-offs, and MobileNets show strong performance compared to other popular network models on ImageNet classification. Effectiveness of MobileNets in a wide range of application scenarios, including object detection, image classification, computational photography, etc.

With the development of deep learning, convolutional neural networks are becoming more and more common. The current general trend is to achieve higher accuracy through deeper and more complex networks, but such networks often do not have much advantage in model size and running speed. Some applications on embedded platforms, such as robotics and autonomous driving, have limited hardware resources and require lightweight, low-latency (and acceptable accuracy) network models, which is the main job of MobileNet.

2 Mathematical Analysis of MobileNet

2.1 Depth-wise Separable Convolution

2.1.1 Separable Convolution Intuition

Convolutions networks improve the performance of dense networks by modeling correlations only within a small scope of resolutions instead of the whole picture. For pictures with channels, a regular convolution kernel attempts to simultaneously capture correlations along the width, height (spatial correlations), and channel dimensions (cross-channel correlations). With increasing requests for high performing convolution networks, a branch of work is on separable convolutions arises. This idea is first proposed in (7), (8) and later extended in (9), (10).

Techniques for improving parameter efficiency have been extensively explored to boost neural network performance. A wide range of applications justify their ability to save memory and time resource for expanding models in more efficient directions. The line of research on reducing parameter redundancy in this report, instead of pursuing better performance, aims to achieve higher efficiency with a mild or no performance drop.

Inception (11) proposes a prototype of separable convolutions based on the assumption that cross-channel correlations and spatial correlations are sufficiently decoupled. Xception (9) makes a stronger hypothesis that cross-channel correlations and spatial correlations can be learnt completely separately, and raises a standard form of depth-wise separable convolutions. MobileNet adopts the same design of depth-wise separable convolutions as in Xception. As displayed in Figure 1, for 3-D inputs, they perform one spatial convolution for each input channel independently and then a list of point-wise convolutions, each maps the intermediate results to a new channel space. This configuration will be referred as the depth-wise first version in the following. Another version of depth-wise separable convolution, studied by (10), swaps the order of the above two components. It will be referred as the point-wise first version.

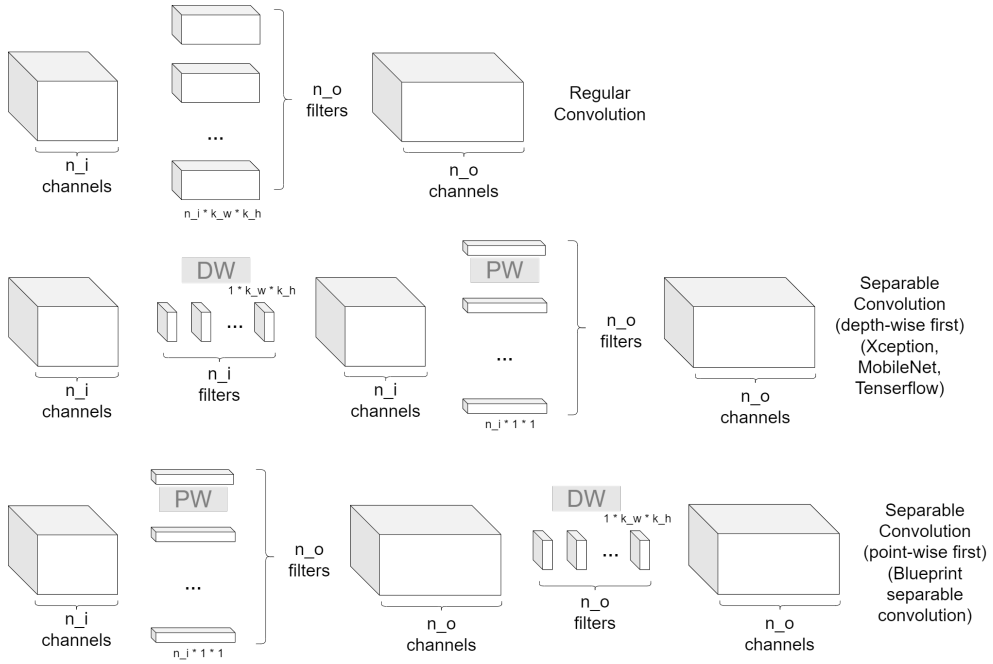

Figure 1: Regular convolution (upper), depth-wise separable convolution 1 (depth-wise first) (middle), and depth-wise separable convolution 2 (point-wise first) (bottom).

Separable convolutions bring significant advantages in reducing model sizes. A regular convolution block contains $O\left(H \times W \times n_i \times k_h \times k_w \times n_o\right)$ parameters. One depth-wise first separable convolution block contains $O\left(H \times W \times \left(n_i \times k_h \times k_w + n_i \times n_o\right)\right)$ parameters and one point-wise first version $O\left(H \times W \times \left(n_o \times k_h \times k_w + n_i \times n_o\right)\right)$ parameters.

To achieve the best effect of parameter reduction, a single block of depth-wise separable convolutions is often used to approach the whole block of regular convolutions in practice. Therefore, the size ratio of between models equipped with depth-wise separable convolutions and regular convolutions is approximately $\frac{1}{k_h \times k_w}$ for both versions, given $n_i, n_o \gg k_h k_w$ in most cases. Precisely, the point-first version may preserve slightly more parameters, given the output feature usually has more channels than the input features $(n_o > n_i)$.

Equipped with depth-wise first separable convolutions, MobileNet maintains within 5% top-1 accuracy drop using approximately 10% of parameters in Inception V3 and within 2% top-1 accuracy drop using less than 3% of parameters in VGG-16.

2.1.2 Relation to PCA Decomposition

Inspired by a theorem in a 2018 paper (12), we observe a connection between depth-wise separable convolutions and PCA decomposition. Denote the numbers of input and output channels as n_i and n_o, height and width of inputs as h and w, convolution operation as $*$, and composition of operators as $\circ$. The theorem and its proof are paraphrased as follows.

Theorem (Guo et al. (12)). $\forall$ block of n_o regular 3-D convolutions $\mathbf{W} = (W_1, W_2, \ldots, W_{n_o}) \in \mathbb{R}^{n_o \times n_i \times k_h \times k_w}$, $\exists$ a set of depth-wise separable convolutions characterized by point-wise and depth-wise kernels $\left\{\mathbf{P}^k, \mathbf{D}^k\right\}_{k=1}^{K}$, where $\mathbf{D}^k = (D_1^k, D_2^k, ..., D_{n_i}^k) \in \mathbb{R}^{n_i \times 1 \times h \times w}$, $\mathbf{P}^k = (P_1^k, P_2^k, ..., P_{n_o}^k) \in \mathbb{R}^{n_o \times n_i \times 1 \times 1}$ for the depth-wise first version, or $\mathbf{D}_k = (D_1^k, D_2^k, ..., D_{n_o}^k) \in \mathbb{R}^{n_o \times 1 \times h \times w}$, $\mathbf{P}_k = (P_1^k, P_2^k, ..., P_{n_o}^k) \in \mathbb{R}^{n_o \times n_i \times 1 \times 1}$ for the point-wise first version. These kernels satisfy that $K \leq hw$ and

$$\mathbf{W} = \begin{cases} \sum_{k=1}^{K} \mathbf{P}_k \circ \mathbf{D}_k \\ \sum_{k=1}^{K} \mathbf{D}_k \circ \mathbf{P}_k \end{cases} \tag{1}$$

where the first equation is for the depth-wise first version and the second for the point-wise first version.

Proof. Given an input of n_i channels, $\mathbf{x} = (x_1, x_2, \ldots, x_{n_i}) \in \mathbb{R}^{n_i \times h \times w}$. For the depth-wise first case,

$$\left\| \left(\mathbf{W} - \sum_{k=1}^{K} \mathbf{P}_k \circ \mathbf{D}_k \right) * x \right\|_2^2 = \left\| \sum_{i=0}^{n_i} \left(W_{:,i} - \sum_{k=1}^{K} P_{:,i}^k D_i^k \right) * x_i \right\|_2^2, \tag{2}$$

The key point is to show that: $\exists P_{:,i}^k, D_i^k$, s.t $\left(W_{:,i} - \sum_{k=1}^{K} P_{:,i}^k D_i^k \right) = 0, \forall i \in \{1, 2, ..., n_i\}$.

Let $\tilde{W}_{:,i}$ be the slice of the regular tensor $W_{:,i,:,:}$ reshaped into a matrix of $\mathbb{R}^{n_o \times (k_h \times k_w)}$, $\tilde{D}_i^k$ be the slice of the regular tensor $D_{i,:,:,:}$ reshaped into a vector in $\mathbb{R}^{(k_h \times k_w)}$, and $\tilde{P}_{:,i}^k$ be the slice of the regular tensor $P_{:,i,:,:}$ reshaped into a vector in $\mathbb{R}^{n_o}$. By singular value decomposition, $\tilde{W}_{:,i} = USV^T$. $rank(\tilde{W}_{:,i}) \leq min\{n_o, k_h \times k_w\}$.

Let $\tilde{P}_{:,i}^k = U_k S(k)$, i.e. the k-th column of U multiplying is the k-th singular value, and $\tilde{D}_i^k = V_k$, i.e. the k-th column of V. Therefore, $W_{:,i} = \sum_{k=1}^{K} P_{:,i}^k D_i^k$ for $K \leq rank(\tilde{W}_{:,i}) \leq min\{n_o, k_h \times k_w\}$.

The proof is similar for the point-wise first case.

The theorem says a block of regular convolutions can be written as a sum of depth-wise separable convolutions without loss of information. By looking into the main idea of this proof, we provide an explanation for this theorem as demonstrated in Figure 2. With a single block of separable convolution, the depth-first version consists of the first principal components extracted from all vertical slices in a regular kernel, and the point-first version consists of the first principal components from all horizontal slices. This observation leads to further analysis of the foundation of separable convolutions.

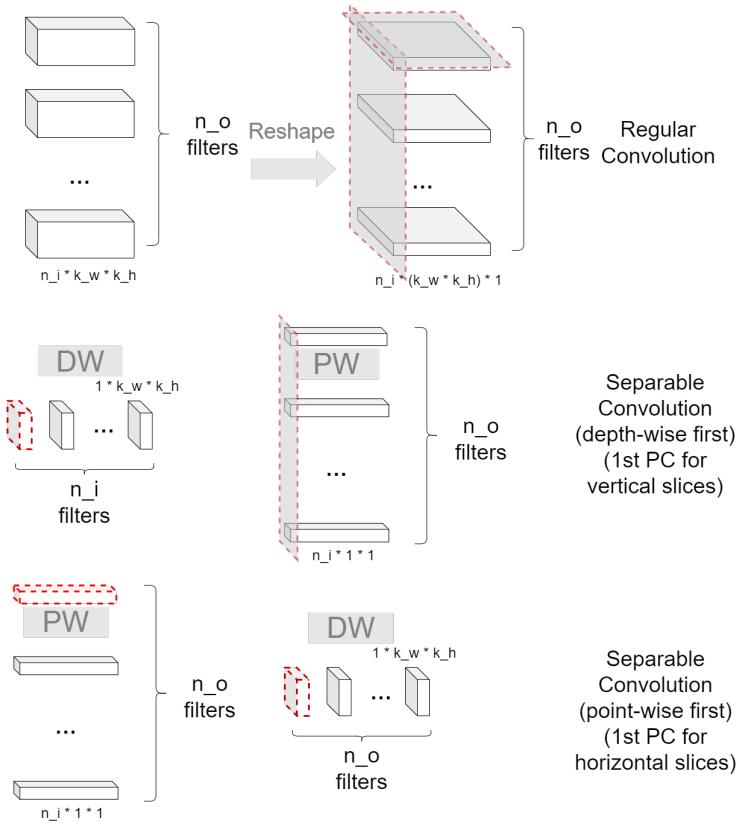

Figure 2: A connection between depth-wise separable convolution and PCA decomposition

2.1.3 Investigation into two versions

A natural question is whether it is possible to find evidence to support the fairness of separable convolutions. We observe that different evidence may intuitively lead to different configurations (depth-wise first or point-wise first) in the end.

Chollet (9) claims that the order of depth-wise and point-wise convolutions in Xception modules is of little importance because from a whole point of view, they can be viewed as alternately placed to each other. MobileNet V1, V2 and V3 (4), (5), (6) adopt the same configuration for their separable convolution blocks as Xception does. However, this explanation might not apply to MobileNet, as there are residual blocks and SE blocks in between.

Later research provides further discussion about the order of these two components. Guo, et al. (12) bring up that the point-wise first configuration performs better probably because for most convolution modules, the output channels are usually deeper than the input channels. Therefore, the point-wise first configuration maintains of more parameters than the depth-wise first configuration does. Haase and Amthor (10) design an experiment to reveal to which extend of parameters are redundant within horizontal slices of regular kernels. Based on the experiment, they propose blueprint separable convolutions (BSConv) accordingly, adopting the point-wise first configuration and observe that MobileNet V1, V2, V3 equipped with BSConv outperform their original versions by approximately 1% on average.

Aware of the possible effect of model sizes (numbers of parameters) raised by Guo, et al., one shortcoming of Haase and Amthor's experiment is lack of counterpart results showing that depth-wise first configuration might be less reasonable. We supplement Haase and Amthor's experiment in the following taking VGG-19 as an example.

The VGG-19 network has 16 convolution layers. Their kernels can be cut into 4995 vertical slices or 5504 horizontal slices. Vertical slices have 43.15% of the variance concentrated on the 1st PC on average, and horizontal slices have 48.24%. The horizontal slices display a slightly heavier concentration on the first few principal components, which coincides with Haase and Amthor's results.

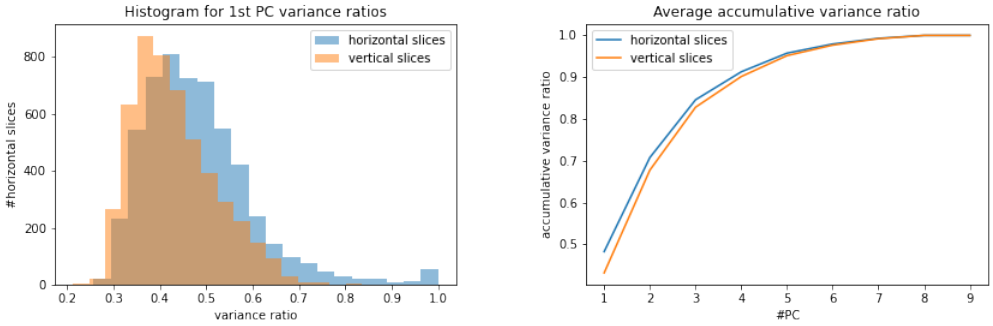

Figure 3: Analysis on VGG-19 kernels

This posterior analysis on model parameters cannot be taken as an exact bound for model capacity of depth-wise separable convolutions. The point is to better understand convolution kernels that neural networks have learned and come up with more efficient architectures. This analysis provides evidence to support the usage of either depth-wise first or point-wise first version for training. From another perspective, it also suggests a method to compress large pre-trained networks for lightweight devices at the deployment stage.

2.2 Inverted Residual Block

2.2.1 Deep Neural Networks and ResNet

With the full exploration of shallow neural networks in academia and industry, scholars have begun to turn their attention to neural networks with more layers, such as AlexNet (13) and VGG (2), to enhance the fitting ability of complex problems such as multi-classification problems. However, with

the deepening of the network, the accuracy of the training set decreases, and the proposal of ResNet (3) solves the problem of the degradation of the deep network to a certain extent, and provides a theoretical basis and tool support for the complex deep network with more training parameters and larger layers.

2.2.2 Inverted Residuals

In ResNet, the most important structure is the residual block, the residual block connects the start and end of the convolution block through shortcut connections.

By adding these two states, the network has access to earlier activations that were not modified in the convolutional block. This approach has proven to be critical for building deep networks.

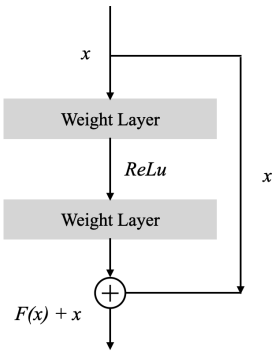

Figure 4: Shortcut Connection.

Figure 1 demonstrates the shortcut connection structure where x stands for the input and $F(x)$ is the output after the weight layer, combined with the shortcut x, where we obtain an output from that block $F(x) + x$.

For the general residual block He et al. introduces, it basically follows a $wide \rightarrow narrow \rightarrow wide$ path, shown in figure 5 below.

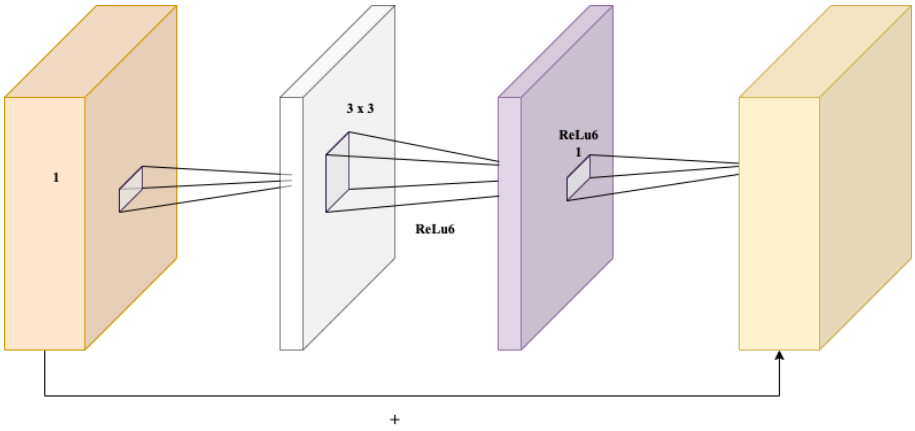

Figure 5: General Residual Block.

Here wide and narrow stand for the amount of channels. In this case, the input is with a great amount of channels, compressed with an inexpensive 1×1 convolution, followed by a 3×3 convolution after with far fewer parameters, finally by using a 1×1 convolution we have an output with wide channels.

Different from the general residual block, MobileNet (5) uses a similar structure with the same convolution path but with inverse channel parameter scaling. As the figure 6 shown below, the inverted residual block follows a convolution path: $narrow \rightarrow wide \rightarrow narrow$.

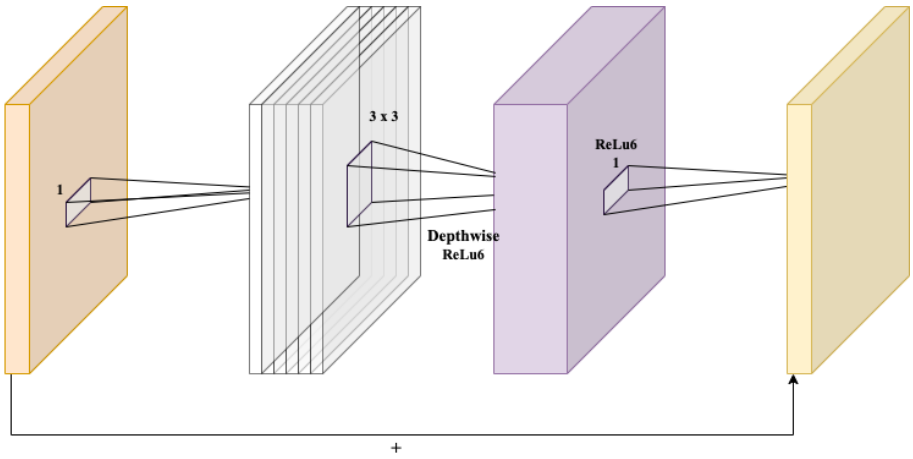

Figure 6: Inverted Residual Block, which is used in MobileNet.

Firstly, it uses a 1×1 convolution expansion network, since the subsequent 3×3 depthwise convolutions have greatly reduced the number of parameters. Then another 1×1 convolutional compression network to match the initial number of channels.

The intuitive changes of the channel parameters of general residual block and inverted residual block are shown in the following figure 7.

Figure 7: General and inverted residual block: the left one denotes the general residual block: $wide \rightarrow narrow \rightarrow wide$, and the right denotes the inverted residual block: $narrow \rightarrow wide \rightarrow narrow$.

In MobileNet, this idea is called inverted residual blocks because there are skip connections between narrow parts of the network, which is the opposite of how the original residual connections work. In the follow-up experiment section, we can also see that the inverted block has fewer parameters.

2.2.3 Inverted Residual Block as Ordinary Differential Equation

In last section, we explain the difference between general residual block and inverted residual block by demonstrating the different convolution paths. The inverted residual block helps to build up the MobileNet, in this section, we are going to talk about the general ideas of why residual block assists in resolving degradation problem in deeper networks, He at al. have proved the effectiveness of the shortcut structure in their work (14), which is briefly stated in Appendix A.1.

An important work to explain the great results brought by Ricky et al. (15). Here, we state and extend their work and check the characteristics of inverted residual block.

In ResNet, for a general shortcut connection structure, we can model the transformation from input to output as:

$$x^{(t+1)} = x^{(t)} + F(x) \tag{3}$$

where $x^{(t)}$ and $x^{(t+1)}$ stand for the states information at time t and $t+1$, and $F(x)$ denotes the transformation of the weight layer.

Similar to the transformation in the residual block structure, here we model the continuous dynamics of the residual structure by applying Euler's Rule with step size 1.

$$\mathbf{h}^{(t+1)} = \mathbf{h}^{(t)} + \sigma\left(\mathbf{h}^{(t)}, \theta^{(t)}\right) \tag{4}$$

where the $\sigma(*)$ is the layer transformation function.

We can think about taking the current hidden state and output the future hidden state through this transformation.

Basically, we are trying to figure out what happened between the current and future hidden states and if there are many layers but with a smaller step, by using ODE, we have:

$$\frac{d\mathbf{h}(t)}{dt} = \sigma(\mathbf{h}(t), t, \theta). \tag{5}$$

We write the residual block in a form of ODE, which is modeled to be a continuous process where the neural network has discrete and natural number of layers, which is to say, there can only be a transformation between two layers n and $n+1$, where $n \in \mathbb{N}$.

The figure from Ricky et al.'s work (15) also illustrates that, shown below.

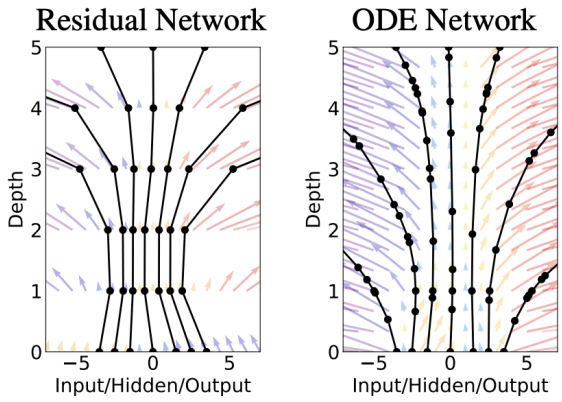

Figure 8: The sequence of transformations in ResNet and ODE Network. (image source: Ricky et al., Neural Ordinary Differential Equations (15))

The forward propagation is straight-forward in inverted residual block and residual block as ODE, where we call it Neural ODE.

For the back-propagation and scale factor in ResNet, He et al. gives detailed proof (14), see Appendix A.1.

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}\left(\mathbf{x}_i, \mathcal{W}_i\right)\right) \tag{6}$$

The equation above helps to illustrate the use of shortcut connection in network degradation issue and prevent the gradient from being extremely large or small from the backward propagation.

Let's get back to the neural ODE, we can use adjoint sensitivity method (16) to compute the gradient in the neural ODE.

To model the backward propagation, let us consider a process that $d\mathbf{z}/dt = \sigma(\mathbf{z}(t), t)$ which we want to approximate, and we have some observations $\{(z_0, t_0), (z_1, t_1), \ldots, (z_M, t_M)\}$, we pick

two points $(z(t), t)$, $(z(t+\tau), t+\tau)$ from the observations and calculate the movement from $(z(t), t)$ to $(z(t+\tau), t+\tau)$,

$$L\left(\mathbf{z}\left(t+\tau\right)\right) = L\left(\mathbf{z}\left(t\right) + \int_{t}^{t+\tau} \sigma(\mathbf{z}(t), t, \theta)dt\right) \tag{7}$$

where $t+\tau$ refers to a small change in time corresponding to t, $L(*)$ is the loss function and $\mathbf{z}(*)$ denotes the hidden state in back-propagation.

Here, our goal is to optimize the loss function, where we define the adjoint term to be:

$$\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{z}(t)}, \tag{8}$$

and a brief recall of the use of adjoint models can be found in Appendix A.2.

From the work of Pontrjagin et al. (16), we have:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\top} \frac{\partial \sigma(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}. \tag{9}$$

By solving the preceding equation in back-propagation, we can obtain the adjoint $\mathbf{a}(t)$. This requires us to compute $\mathbf{z}(t)$ backward in time together with adjoint $\mathbf{a}(t)$.

Now we take the integral of equation (10) to calculate adjoint $\mathbf{a}(t)$,

$$\frac{\partial L}{\partial \mathbf{z}\left(t+\tau\right)} = -\int_{t}^{t+\tau} \mathbf{a}(t)^{\top} \frac{\partial \sigma(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} \, \mathrm{d}t. \tag{10}$$

Similarly, we can compute the gradients corresponding to θ,

$$\frac{dL}{d\theta} = -\int_{t}^{t+\tau} \mathbf{a}(t)^{\top} \frac{\partial \sigma(\mathbf{z}(t), t, \theta)}{\partial \theta} dt \tag{11}$$

In every back-propagation, we update the gradient $\frac{\partial L}{\partial \theta}$ of parameter θ, and adjoint $\mathbf{a}$.

Thus, we have the model of forward and backward propagation for the Neural ODE, and we analyse the its performance versus the performance of a small residual network, shown in next section.

2.3 Comparison between ResNet and Neural ODE Model

Ricky et al. (15) have experiments on Neural ODE and achieve 99.58% accuracy on the MNIST dataset with a relative small network model. Here we show the work on CIFAR-10 dataset and compare the performance between Neural ODE and a small network built up with general residual block.

Table 1: Performance on CIFAR-10.

	Validation Accuracy	# Params
ResNet	85.14%	0.27M
Neural ODE	84.61%	0.27M

As shown in table 1 above, the Neural ODE model achieves similar performance for a small ResNet with general residual block.

2.4 Squeeze-and-excitation

MobileNet (6) applies the squeeze-and-excitation block which is originally an outstanding work from Hu et al. (17) and we briefly state their work in Appendix A.3, here we will mainly focus on the application of squeeze-and-excitation block in MobileNet (6), where the block in MobileNet becomes as follows.

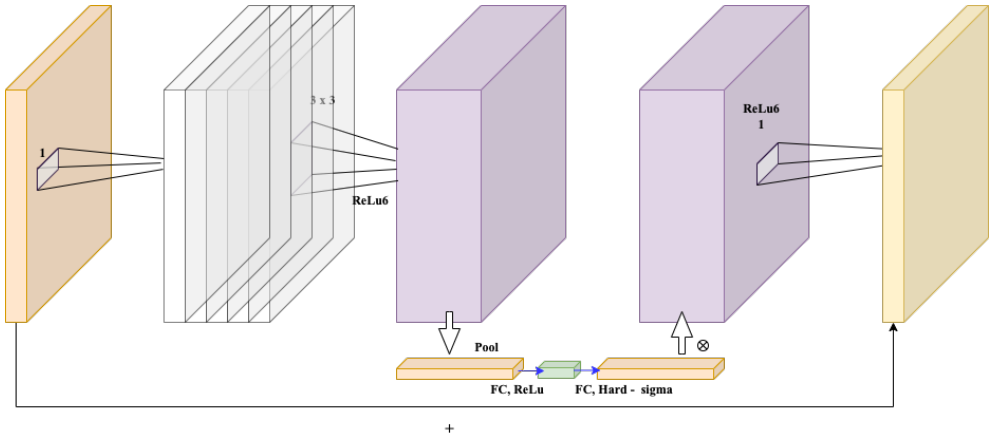

Figure 9: MobileNet with squeeze-and-excitation block.

Squeeze and Excitation include Squeeze (compression) and Excitation (activation) two parts. Among them, the squeeze operation obtains the compressed feature map by performing global average pooling on the feature map, and the excitation obtains the weight of each channel in the feature map through the two-layer full connection, and uses the weighted feature map as the next layer of neural network.

2.5 Activation Function and Linear Bottleneck

2.5.1 From ReLu to ReLu6

From VGG to other CNN architectures, the convolution module is always composed with batch normalization and ReLu, which brings non-linear properties to the network model. Different from $ReLu$, MobileNet (5) uses $ReLu6$, which can be expressed as:

$$ReLu(6) = \min(\max(0, x), 6) \tag{12}$$

MobileNet uses ReLu6 instead of ReLu, the basic idea coming from the transfer is the robustness.

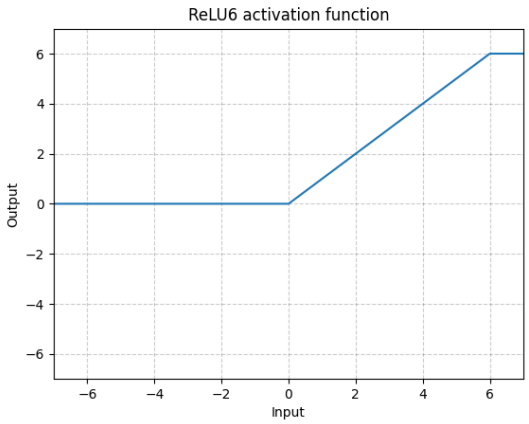

Figure 10: Activation function of ReLu6 (image source: PyTorch 1.11.0 Documentation).

ReLu6 has a feature of limitation of the maximum output, shown in the figure 10 above, which is used to control the magnitude of the numerical value, and it is easy to explode numerically by simply using ReLu in a large network.

3 Architecture of MobileNetV2 and MobileNetV3

3.1 MobileNets: Background

MobileNetV2 and MobileNetV3 are the 2nd and 3rd iteration of a general class of models called MobileNets, first developed in 2017 for use in mobile and embedded system applications. Because of their intended use case, all MobileNets are designed to be efficient, lightweight models. The original MobileNet, sometimes referred to as MobileNetV1, introduced the use of depthwise-separable convolutions convolutions to significantly reduce the computational complexity required to utilize the network (4). This first version demonstrated strong performance when compared to other popular networks at the time, such as VGG16 and AlexNet, in spite of the fact that these models often contained over 10x the number of parameters as in MobileNetV1.

3.2 MobileNetV2

The second generation MobileNet, MobileNetV2, was first published in 2019 (4). In this iteration, the authors introduced the inverted residual block structure and combined it with the depthwise-separable convolution style previously implemented in MobileNetV1. MobileNetV2 further improved upon the classification performance of MobileNetV1, and also greatly improved its efficiency, reducing the number of parameters, number of operations, and execution time.

The primary building block of MobileNetV2 combines several key features, including residual connections, depthwise-separable covolutions, and a bottleneck structure. These blocks make use of a linear bottleneck structure, which is similar to a normal bottleneck, but with the final activation layer removed. This is done to preserve information which otherwise would be destroyed by the nonlinear activation, and the authors discuss how this has been empirically shown to improve performance. The blocks also make use of the depthwise-separable convolutions featured in MobileNetV1, and are thus afforded similar benefits. The block also features inverted residual connections, which are more memory efficient and provide improved performance. The overall structures of these blocks is shown in Table 2, and are depicted visually in Figure 11.

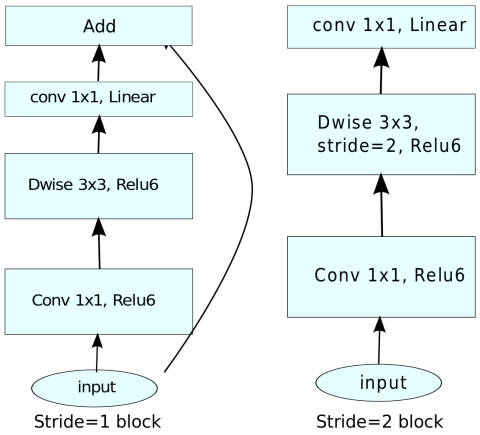

Figure 11: Illustration of the MobileNetV2's basic bottleneck block, featuring depthwise convolutions and an inverse residual connection (image source: MobileNet V2 (5)).

In the complete MobileNetV2 architecture, shown in Table 3, 7 of these bottleneck blocks are combined in sequence. The first layer of the network is a basic 2d convolution, as is the layer immediately following the bottleneck sequence. The network then feeds through an average-pooling layer and a final 2d convolution.

Table 2: The *bottleneck residual block* introduced in MobileNet v2 (5), which transforms from k to k' channels, with stride s, and expansion factor t.

Input	Operator	Output
$h \times w \times k$	1x1 ,	$h \times w \times (tk)$
$h \times w \times tk$	3x3 s=s,	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 3: MobileNetV2 (5): Complete model structure, featuring 7 bottleneck convolutional/residual blocks.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

3.3 MobileNetV3

3.3.1 Bottleneck Architecture

Figure 9 9 shows the architecture of each bneck, which is also the inverted residual architecture described above. First, we use a 1×1 convolution kernel to increase the dimension. After the convolution layer, there will be batch normalization and ReLu6 activation functions. Then there is a 3×3 DepthWise Convolution, which still has batch normalization and ReLu6 activation function. Then there is a 1×1 convolution kernel, which reduces the channels of the feature map. It should be noted that it only has batch normalization and does not use ReLu6.

It can also be seen from Figure 8 that bneck has a shortcut branch, which is similar to the shortcut connection of the Resnet network. When stride in DepthWise Convolution is equal to 1 and the channels of the input feature matrix and the output feature matrix are consistent, it will perform a shortcut connection, that is: the input feature matrix and the output feature matrix are added in the same dimension.

3.3.2 SE Module

Compared with MobileNetV2, MobileNetV3 adds an attention mechanism to bneck. Its idea is also very simple. For example, when we are doing the image classification task, there is a dog walking in the vast desert, the model can easily to recognize the dog, because the background of the picture is very simple. If a dog is walking in a dense forest with plants, trees, birds, streams, etc., the model may not recognize it. Because the model focuses on the whole image, not a certain area, and the background in the forest is complex. So once we change the simple background to a complex background, the model is likely to misclassify. As a human, it is easy to classify because we only care if there is a dog in the scene, we don't care if the background is desert or forest. In order to solve the problem of complex background, the author uses the SE module.

First, pool each channel of each feature matrix we get, and then get our output vector through two fully connected layers. The output vector is a weight relationship analyzed for each channel of each feature matrix. When the model thinks this channel is important, it gives it a larger weight. And vice versa, when the model thinks the channel is not important, it gives it a smaller weight. Figure 12 is

the calculation flow of a simple SE module. We assume that the shape of the feature map is $2 \times 2 \times 2$, after the SE module, we can get a new feature matrix.

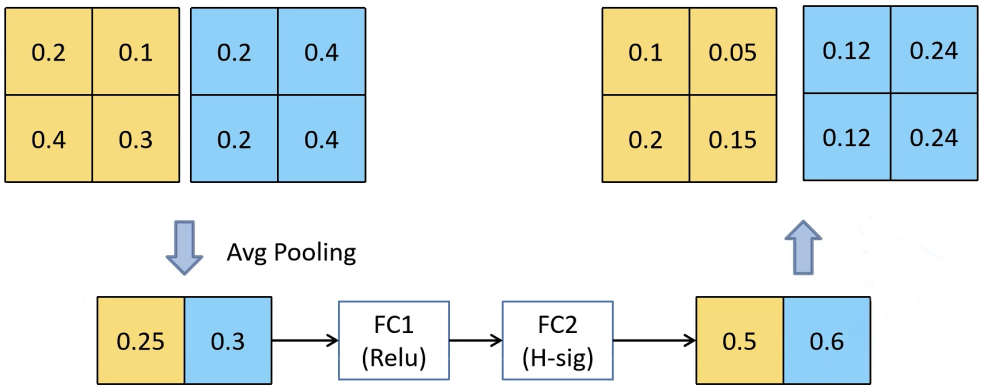

Figure 12: Calculation process of SE module (image source: (18))

3.3.3 Redesigning Expensive Layers

The purpose of MobileNet is to reduce the computing time as much as possible while maintaining high accuracy, so that the model can be run on mobile devices. In order to achieve this purpose, the author proposes Redesigning Expensive Layers. First, the author found through experiments that in the first convolution layer, the results obtained by using 16 convolution kernels and using 32 convolution kernels are consistent, so by adjusting the number of convolution kernels in the first convolutional layer to 16, it can be found that the model can reduce the training time by 2 milliseconds(6). Second, the author simplified the Last Stage, as shown in Figure 13 below.

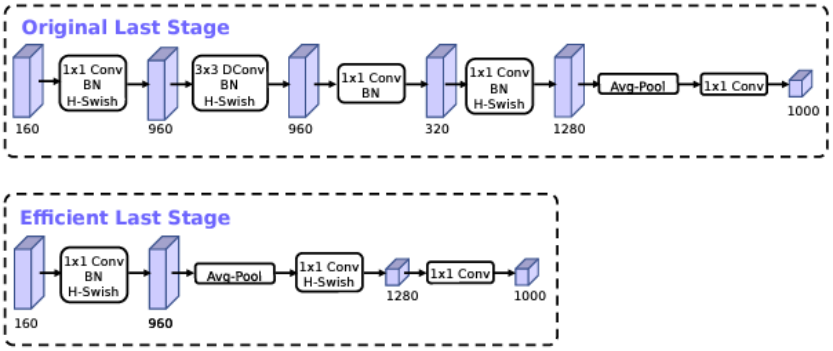

Figure 13: Redesigning Expensive Layers (image source: MobileNet V3 (6)).

Original Last Stage is a very time-consuming structure, and the author simplifies its structure. As shown in Figure 13, the adjusted Expensive Layers obviously have a lot less convolutional layers than the original result. Experiments show that the accuracy of the adjusted model does not change significantly, but saves 7 milliseconds, which 11% of the training time(6).

3.3.4 MobileNetV3 Architecture

Table 4 is the Architecture of MobileNetV3 Large. The exp size in the table represents the dimension required to be increased in the first ascending dimensional convolution in each bneck. After passing through the DepthWise convolution and the SE module, the channel will not change, and then the 1×1 convolution is used for dimensionality reduction. The channel after dimensionality reduction is out in Table 4, SE represents whether the current layer uses the attention mechanism, NL represents the activation function used by the current layer, HS represents hard swish, RE represents ReLu 6, and s represents stride.

Table 4: Specification for MobileNetV3-Large

Input	Operator	exp size	# out	SE	NL	s
$224^2 \times 3$	conv $2d$	–	16	–	HS	2
$112^2 \times 16$	bneck, 3×3	16	16	–	RE	1
$112^2 \times 16$	bneck, 3×3	64	24	–	RE	2
$56^2 \times 24$	bneck, 3×3	72	24	–	RE	1
$56^2 \times 24$	bneck, 5×5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5×5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5×5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3×3	240	80	–	HS	2
$14^2 \times 80$	bneck, 3×3	200	80	–	HS	1
$14^2 \times 80$	bneck, 3×3	184	80	–	HS	1
$14^2 \times 80$	bneck, 3×3	184	80	–	HS	1
$14^2 \times 80$	bneck, 3×3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3×3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5×5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5×5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5×5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1×1	–	960	–	HS	1
$7^2 \times 960$	pool, 7×7	–	–	–	–	1
$1^2 \times 960$	conv2d 1×1, NBN	–	1280	–	HS	1
$1^2 \times 1280$	conv2d 1×1, NBN	–	k	–	–	1

4 Experiments

4.1 Training and Environment Setup

In this Section, we will experiment with all the above models using the flower dataset and a batch size of 16, the models are MobileNetV2 trained from scratch and MobileNetV2 trained from scratch, MobileNetV3 Small trained from scratch and MobileNetV3 Small trained from scratch, MobileNetV3 Large trained from scratch and Using the pretrained model MobileNetV3 Large, six models. We will conduct experiments on GCP, the graphics card is NVIDIA Tesla T4, the optimization function of all models is Adam, the learning rate is 0.0001, and the loss function is CrossEntropyLoss, and the Epoch is 25.

For dataset, we use the "flowers" dataset from TensorFlow, a dataset comprised of 3,670 images of 5 types of flowers (daisies, dandelions, roses, sunflowers, tulips), shown in figure 14 below.

Figure 14: Sample Dataset.

4.2 Experimental Results

4.2.1 MobileNetV2

We firstly have experiment on MobileNetV3, which is constructed with depthwise separable convolution and inverted residual block but without squeeze-and-excitation block.

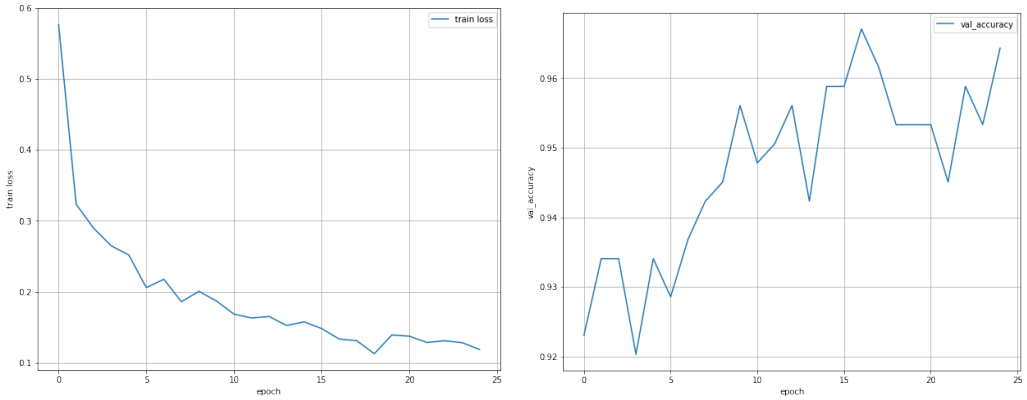

Figure 15: Performance of MobileNetV2 when training from scratch

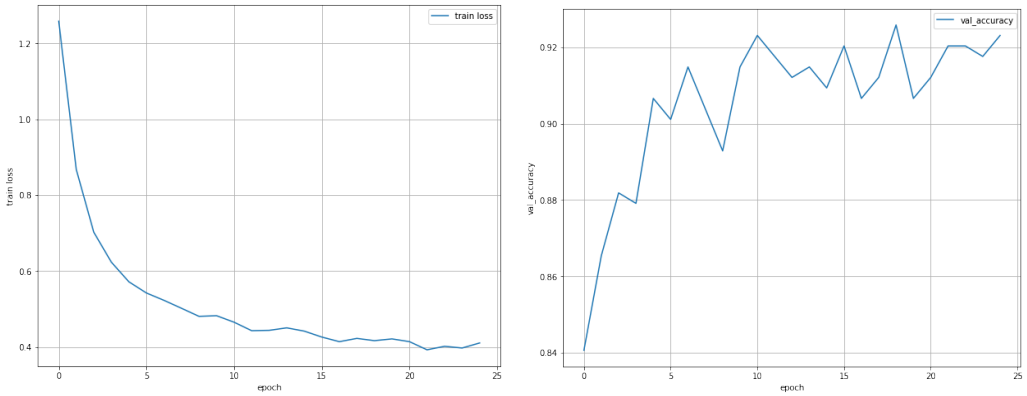

Figure 16: Performance of MobileNetV2 when applying pre-trained weights learned on ImageNet

MobileNetV2 showed strong performance both when using randomly-initialized weights (Figure 15) and when using pre-trained weights learned via classification of the ImageNet database (Figure 16). In both cases, we achieve an accuracy rate of .95 in roughly 5 epochs. Ultimately, MobileNetV2, without pre-training, achieved a Top-1 Accuracy score of 96.4%, and the pre-trained model achieved a Top-1 Accuracy score of 92.3%, as seen in Table 5.

Another trend which can be found through the experiment of MobileNet-V2 is the convergence speed, MobileNet has a major advantage, which is size over other network architectures such as VGG (2), etc, MobileNet-V2 has some new features, depthwise seperable convolution and inverted residual block, those features build up a lighter and easy trainable network architecture.

4.2.2 MobileNetV3

In previous section, we have experiments focusing on MobileNetV2, here, we add the squeeze-and-excitation block to the MobileNetV2 and redesign the expensive layers, which brings features like attention mechanism and computing time reduction.

We maintain the same experiment setting, but add new features on, shown in the figure 17 below:

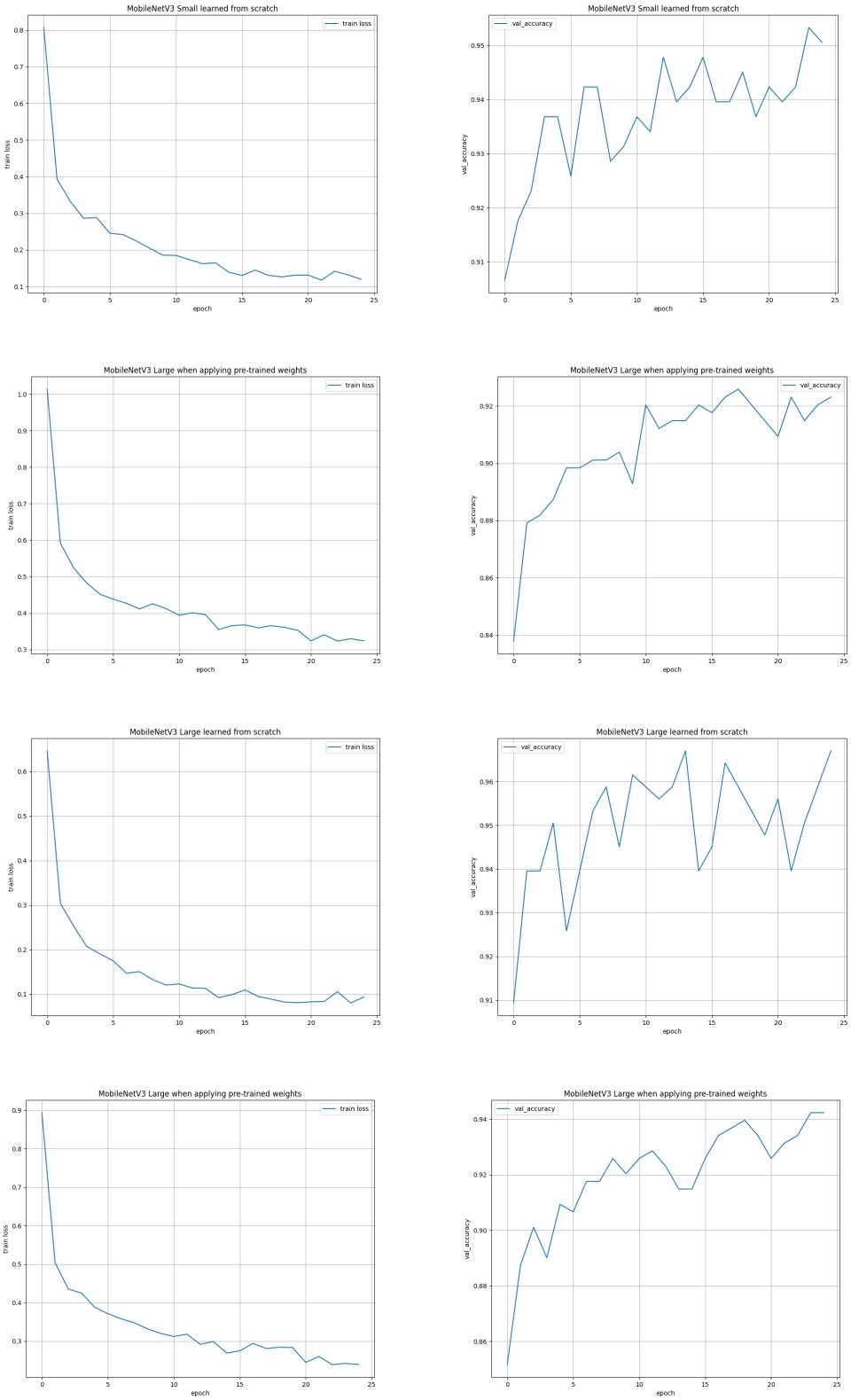

Figure 17: Performance of MobileNetV3. From top to bottom, we have the performance (loss and validation accuracy) of (a) MobileNetV3 Small learned from scratch; (b) MobileNetV3 Large with pre-trained weights; (c) MobileNetV3 Large learned from scratch; (d) MobileNetV3 Small with pre-trained weights.

We train the model using two methods, learning from scratch and using pretrained weights, and the experimental results are shown in Figure 16. All of them can achieve more than 95% accuracy, especially the MobileNetV3 Large learned from scratch, which achieves 97.3% accuracy.

Another fact we observe from the experiment results on MobileNet V3 is the convergence trend, He et al. (3) show an outstanding work on make the network easier than conventional neural network architectures. Here, we also show that MobileNet has the same fact, using inverted residual block and depthwise separable convolution, MobileNet V3 resolves the issue network degradation problem and comes up with a model with much less parameters, where we may compare it with other architectures in following section (6). If we compare MobileNet V3 with V2 in previous section, MobileNet V3 shows a great potential in fitting model which can be reasonable result by the use of squeeze-and-excitation block, which analyzes the correlation of channels and reinforce some critical features from all features to improve accuracy.

4.3 Model Prediction Show

Figure 18 shows the predictions we get using the weights trained from scratch on MobileNetV3 Large, with tulips on the left and sunflowers on the right. As you can see from the results, they are 100% accurate.

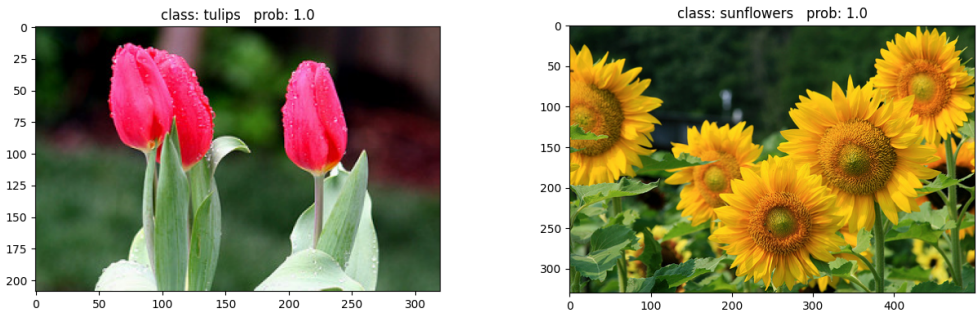

Figure 18: Model Predictions

4.4 Model Comparison

The most important contribution of MobileNet is to provide a model that allows us to run on mobile devices, so the training time and parameters of the model are a very important factor, and we will compare the experiments. It can be seen from Table 5 that MobileNetV3 Small has 0.9M fewer parameters than MobileNetV2. The result is a reduction in training time. MobileNetV3 Large has 2M more parameters than MobileNetV2 and 2M more parameters than MobileNetV3 Large, but the accuracy is improved by 2.6%.

Table 5: Comparative Results

Model	Top1 - Acc	Parameters	Training Time
MobileNetV2	96.4%	2.2M	19s / epoch
MobileNetV2 Pretrain	92.3%	6405	14s / epoch
MobileNetV3 Small	95.1%	1.52M	17s / epoch
MobileNetV3 Large	97.3%	4.2M	19s / epoch
MobileNetV3 Small Pretrain	92.3%	0.59M	14s / epoch
MObileNetV3 Large Pretrain	94.2%	1.2M	15s / epoch

It can be seen from Table 5 that using the pre-training model, that is, freezing all weights except the last two fully connected layers, can reduce the training time and still have a high accuracy

rate. Especially MobileNetV2, using pre-training weights, freezing all layers except the last fully connected layer, the training parameters of the model are only 6000. We found experimental results trained on ImageNet from both the ResNet (3) and the MobileNetV3 (6) paper. As can be seen from Figure 19, MobileNet greatly reduces training parameters while maintaining high accuracy, making it possible to run on mobile devices.

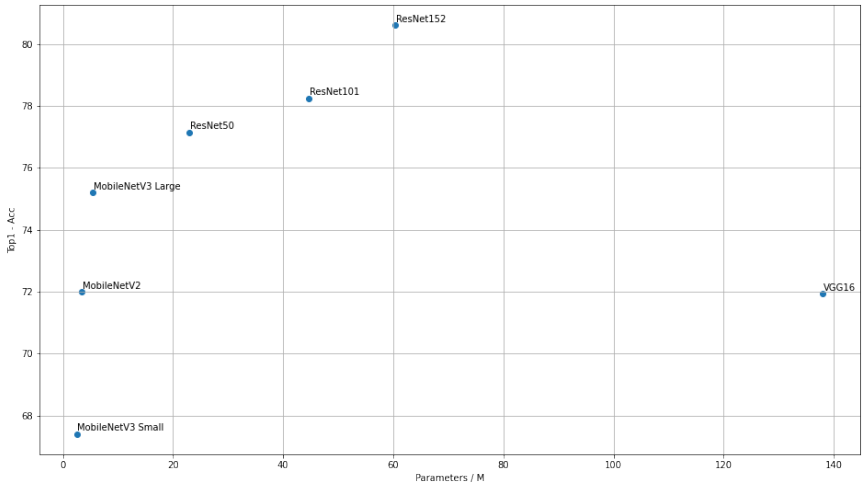

Figure 19: Comparative Results.

As a fact, MobileNet family forms a cluster with parameters less than 5.4M and achieves a classification accuracy over 67.4% (6) on ImageNet, as a comparison, conventional neural network models form a cluster with parameters no less than 25M, but no obvious lead over MobileNet family.

MobileNetV3 Large, outperforms than all other models in MobileNet family, with around 5.4M parameters, MobileNetV3 Large achieves a performance over 75.2%, where 5.4M (6) parameters are acceptable for some mobile devices and the model stands out in size in MobileNet family is the MobileNetV2 Pre-trained model, with a total training parameters of 6405 (we only train the last layer and use the pre-trained weights on ImageNet, without pre-train, the size of MobileNet is also relatively small comparing to other network architectures), but still with a classification performance over 66% when using our training set.

From table 5 and figure 19 previously in this section, MobileNet shows its advantage in cost-efficiency of performance and parameters.

5 Conclusion

In this project, we first discuss the notable technical details of MobileNet, and try to combine the knowledge learned in class with it to explain its properties, such as depthwise structure, inverted residual block, and non-linear activation function characteristic.

Following this, we examine the performance of MobileNetV2 and MobileNetV3 to see the effects of these architectural features. We conducted our experiments on the "flowers" dataset included with TensorFlow, and our best-performing model, MobileNetV3-Large, achieved a top1-accuracy of 97.3%. All of our models either met or exceeded the performance of VGG16 and VGG19, which we used as benchmarks. Our results demonstrate the impressive efficiency gains of MobileNet architectural features, which can outperform standard models over 10 times their size, and further prove the value of these networks for mobile and embedded-system applications.

As a summary of the EECS 6699 course project experience itself, in this project, we extend the key structure of MobileNet, and use the knowledge taught in the class and in-depth discussions by Prof. Predrag R. Jelenkovic in this semester's course to refine and understand the relevant knowledge , and combining mathematical theoretical foundations with experimental demonstrations consolidates our understanding of MobileNet and related knowledge in the course.

Acknowledgement

We sincerely thanks Prof. Predrag R. Jelenkovic for his teaching, and appreciate Prof. Predrag R. Jelenkovic and Sam Fieldman for the guidance this semester.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[4] 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," 2017. cite arxiv:1704.04861.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2018. cite arxiv:1801.04381.

[6] A. G. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1314–1324, 2019.

[7] L. Sifre and S. Mallat, "Rigid-motion scattering for texture classification," *arXiv preprint arXiv:1403.1687*, 2014.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[9] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

[10] D. Haase and M. Amthor, "Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14600–14609, 2020.

[11] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[12] J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li, "Network decoupling: From regular to depthwise separable convolutions," *arXiv preprint arXiv:1808.05517*, 2018.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 630–645, Springer International Publishing, 2016.

[15] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[16] L. Pontrjagin, V. Boltyanskii, R. Gamkrelidze, E. Mishchenko, and D. Brown, *The Mathematical Theory of Optimal Processes*. International series of monographs in pure and applied mathematics, Wiley, 1962.

[17] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *CoRR*, vol. abs/1709.01507, 2017.

[18] "GitHub - WZMIAOMIAO/deep-learning-for-image-processing: deep learning for image processing including classification and object-detection etc. — github.com." `https://github.com/WZMIAOMIAO/deep-learning-for-image-processing`. [Accessed 07-May-2022].

[19] R. M. Errico, "What is an adjoint model," *Bulletin of the American Meteorological Society*, vol. 78, pp. 2577–2591, 1997.

A Appendix

A.1 Back-propagation in Residual Network

From the work of He et al., we can have the identity mapping in deep residual networks (14).

The residual block (forward) performs a transformation as:

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}\left(\mathbf{x}_l, \mathcal{W}_l\right). \tag{13}$$

By making a recursive call, we have:

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}\left(\mathbf{x}_i, \mathcal{W}_i\right). \tag{14}$$

For the back-propagation in residual network, by applying the chain rule, we have:

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}\left(\mathbf{x}_i, \mathcal{W}_i\right)\right) \tag{15}$$

where $\mathcal{E}$ denotes the loss function.

He et al. illustrates the importance of shortcut connection by stating a similar equation to equation (12), which is,

$$\mathbf{x}_{l+1} = \lambda_l \mathbf{x}_l + \mathcal{F}\left(\mathbf{x}_l \mathcal{W}_l\right) \tag{16}$$

where λ_l stands for the modulating scalar.

By making a recursive call, we can obtain,

$$\mathbf{x}_L = \left(\prod_{i=l}^{L-1}\lambda_i\right)\mathbf{x}_l + \sum_{i=l}^{L-1}\hat{\mathcal{F}}\left(\mathbf{x}_i, \mathcal{W}_i\right) \tag{17}$$

which yields a back-propagation in the following,

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(\left(\prod_{i=l}^{L-1}\lambda_i\right) + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\hat{\mathcal{F}}\left(\mathbf{x}_i, \mathcal{W}_i\right)\right) \tag{18}$$

From the equation above, we can figure out for a neural network with deep structure, if $\lambda_i > 1$, the gradient can be quite large and yields issues like gradient explosion but with $\lambda_i < 1$ the term will vanish in the back-propagation, if we restate the equation (14) in the following form:

$$\mathbf{x}_{l+1} = \lambda_l \mathbf{x}_l + \mathcal{F}\left(\mathbf{x}_l, \mathcal{W}_l\right). \tag{19}$$

The analysis from equation above will be a great proof for λ_l to be 1, which verifies the correctness and effectiveness of the scale of shortcut connection.

A.2 Adjoint Models

Here we state the work of Ronald M. Errico (19) as a supplement to illustrate the adjoint in section 2.2.3.

For a general input-output system, we can denote it as:

$$\mathbf{y} = F(\mathbf{x}), \tag{20}$$

where F stands for a model operator, if the system has many sub-models and they are similar to the in-and-out relationship to the model above, we can make a recursive call and denote the model as:

$$F(\mathbf{x}) = F_N\left(F_{N-1}\left(\cdots\left(F_2\left(F_1\left(F_0(\mathbf{x})\right)\right)\right)\right)\right) \tag{21}$$

where we can apply a sequential formulation to obtain:

$$y_j^{(n)} = \sum_k \frac{\partial y_j^{(n)}}{\partial y_k^{(n-1)}} y_k^{((n-1)}, \quad for\ N \in [1, N] \tag{22}$$

To calculate the gradient, we assume $J = J(\mathbf{y}) = J(F(\mathbf{x}))$, by chain rule, we have:

$$\frac{\partial J}{\partial x_j} = \sum_k \frac{\partial y_k}{\partial x_j}\frac{\partial J}{\partial y_k} \tag{23}$$

A.3 Squeeze and Excitation Network

The squeeze-and-excitation is introduced by Jie Hu et al., which is considered as an effective method to extract weights in neural networks (a model attention-like mechanism).

Here, we conclude the idea in the work of Jie Hu et al. in brief. We can write the model of squeeze-and-excitation (Jie Hu et al., 2019), which we can build up a transformation $\mathbf{F}_{tr}$ from the input to output, firstly, we have:

$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s \tag{24}$$

where $\mathbf{U} = [u_1, u_2, \cdots, u_C]$ is the output and X is the input, and $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_C]$ is earned set of filter kernels, $*$ denotes the convolution operation.

We can denote **global information embedding** (squeeze), formally, $\mathbf{z} \in \mathbb{R}^C$ is calculated by down-sampling U through its spatial dimensions $H \times W$, thus, c-th element of z is calculated by:

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j) \tag{25}$$

After squeezing, the network manages to have an excitation process.

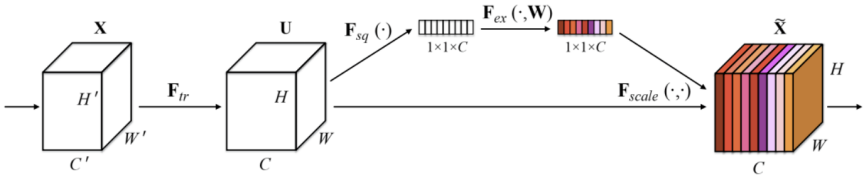

Figure 20: Squeeze and excitation (image source: Jie Hu et al., 2019 (17))

Excitation is implemented with **2 fully-connected layers** (FC). The first FC compresses C channels into C/r channels to reduce the total amount of parameters, the second FC restores it back to C channels.

$$\tilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \mathbf{u}_c \tag{26}$$