In The Name of God

Assignment 3

Course Instructor:Dr. Kheradpisheh

Author : Mohamadreza Khanmohamadi

Lesson : Neural Network

December 16, 2022

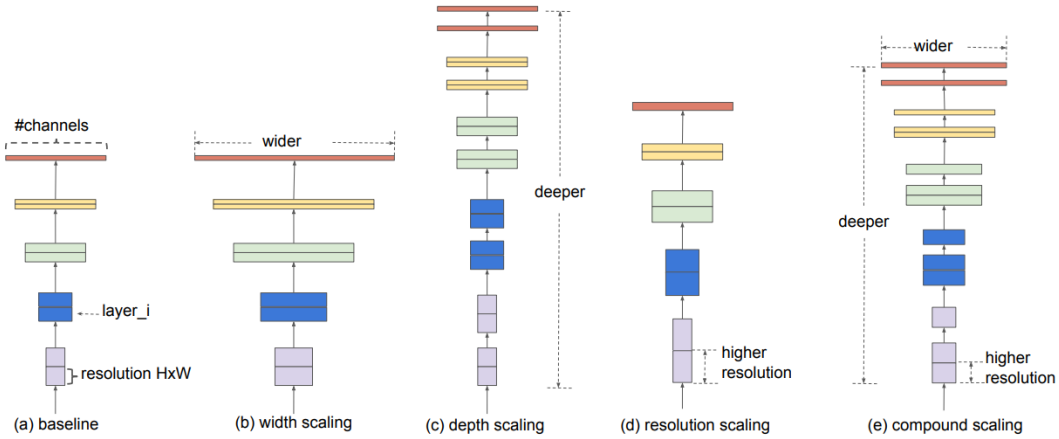
Problem 1

The EfficientNet architecture needs to be described in detail and thoroughly explain all its concepts and innovations.

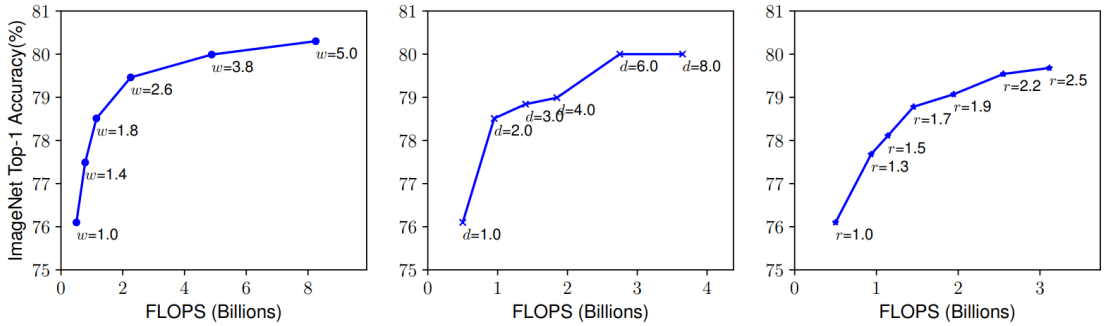
Solution.

1 EfficientNet Articel Overview

The main idea and result of this article is rescaling the main and state-of-the-art models like Resnet and ... to achieve the main model with lower parameters (Smaller) and faster without losing the main model results. In the article some baselines for model scaling: 1. Wide 2. Depth 3. Resolution



in this article, another scaling used is the compound scaling method.when the input image resolution increase the depth and wide for getting the better results. One of the problems that scaling up the network can cause is the vanishing gradient problem, which can be solved with skip connection.



As we see above the accuracy after some scaling up the parameteres doesn't change it shows that the model scaling can not effeciently get the better result.So we now use the main idea of paper : Can we compound the above scaling tricks together to get better results?

1.1 Compound Scaling

As we can see in the below fig we can understand that diffrenet dimension of the scaling are not independent. The results demonstrate that when two scaling parameters increase we can increase the another one. As example when the resolution of the image increase we can use deeper layers.

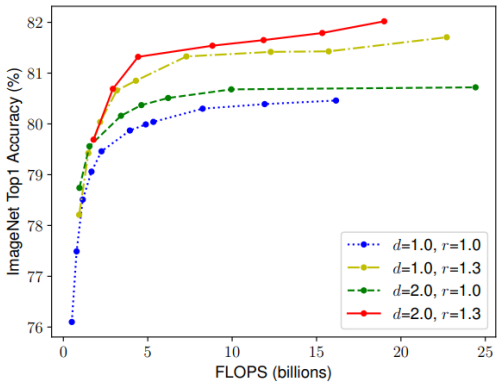


Figure 4. **Scaling Network Width for Different Baseline Networks**. Each dot in a line denotes a model with different width coefficient (w). All baseline networks are from Table 1. The first baseline network (d=1.0, r=1.0) has 18 convolutional layers with resolution 224x224, while the last baseline (d=2.0, r=1.3) has 36 layers with resolution 299x299.

If we want to formalize this method in math we can represent that as an optimization problem.

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

Where the α, β, γ are constants that show the scaling method parameters change and φ is a user-specified coefficient that controls how many more resources are available. In this paper there is no baseline model for compare. Every model that get state-of-the-art results like Alexnet , Resnet , ... are comparing with themselves after using scaling methods.

1.2 Diffrenet Architectures

The base EfficientNet-B0 network is based on the inverted bottleneck residual blocks of MobileNetV2, in addition to squeeze-and-excitation blocks.

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

For the finding and searching for the best solution (solving the optimization problem) we can first fix some paremeters and find others that have done by the writers in two step.

1.3 Results

As we can see the results of the efficient net is better than the base models.

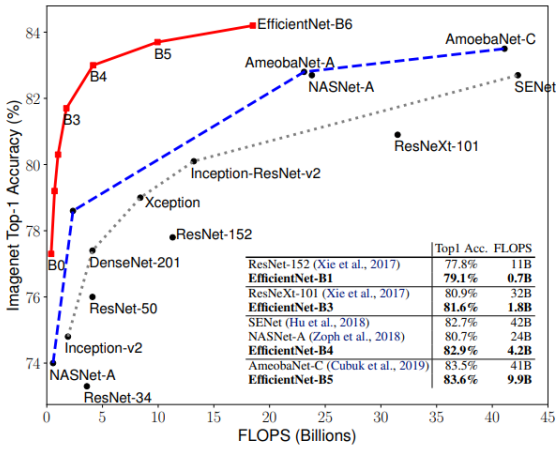


Figure 5. **FLOPS vs. ImageNet Accuracy –** Similar to Figure 1 except it compares FLOPS rather than model size.

Problem 2

The ResNet and the Inception-ResNet should be compared and discussed in terms of their advantages and disadvantages.

Solution.

2 Resnet

2.1 Introduction

Resnet model was introduce in 2015 by Kaiming He, Xiangyu Zhang Shaoqing Ren, Jian Sun.This aricle is most cited article in the deep learning. The reason for the success of the Resnet model is that it allowed us to train very deep neural networks with more than 150 layers. Before Resnet, very deep neural networks were in trouble due to the problem of gradient fading.

When the plain models(such as Alexnet , Fully Connected, ...) becomes deeper for reaching the better results on the complex tasks the vanishing or exploding gradients happens.

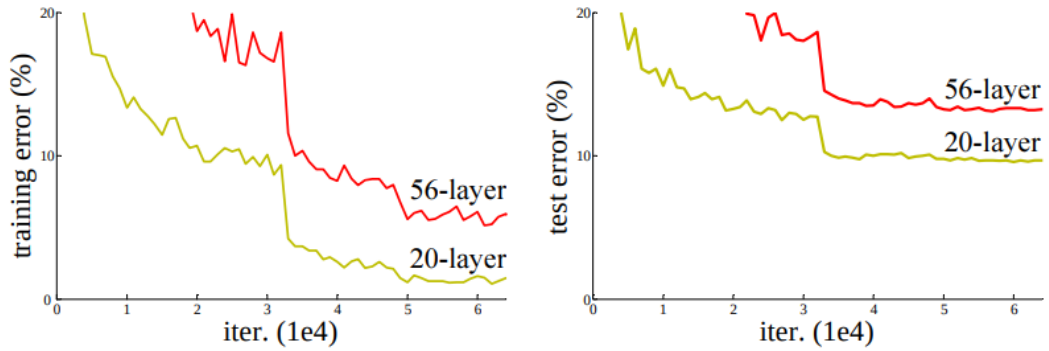
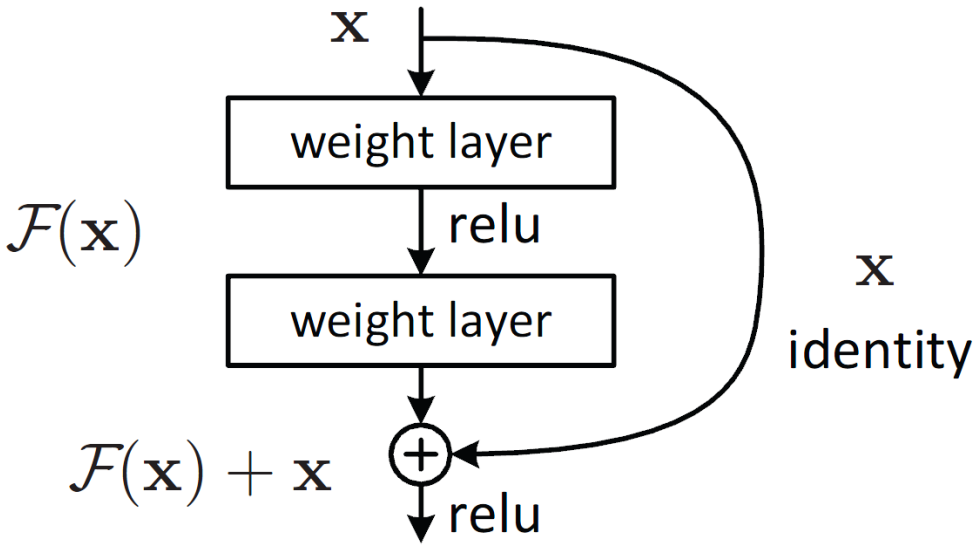


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

The idea of the paper is quite simple. ResNet's main idea is to create an "identity shortcut link" that skips one or more layers.When the model layers increase after some layers data information lose and the deeper layers doesnt get the main features and infos.

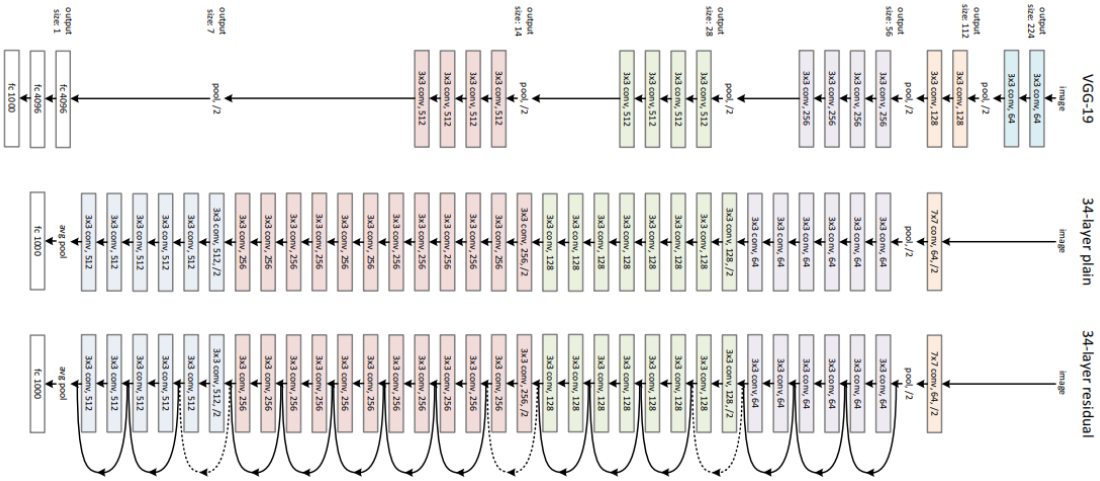


Lets see how its works : $H(x) = f(wx + b)$

but in the resnet we have skip connection that make the equation such : $H(x) = f(wx+b)+x$

Note : if we have different dimension size in the last layer we can use some approach that allow us to make dimensions same(such as zero padding , ...). In this architecture, the output of each block is the sum of the output of the previous block and the output of the layers inside this block. Each new layer will only help to improve the network. In the back propagation phase, due to the fact that the gradient can be propagated in the process of collecting, the gradient is well spread in the layers of very deep networks and the problem of vanishing gradient is solved to a large extent.

2.2 Architecture



We can see the model architecutre of resnet in above fig. As layer increasing the time complexity of the model will increase too. to addressing this problem the paper use CONV1*1 for reducing the model parameter without effecting on the results. So as we see in the below fig the designers decide to use CONV1*1 for the first and last in each bettleneck.

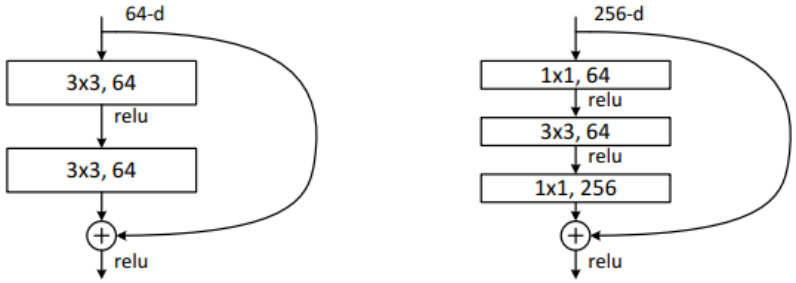


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

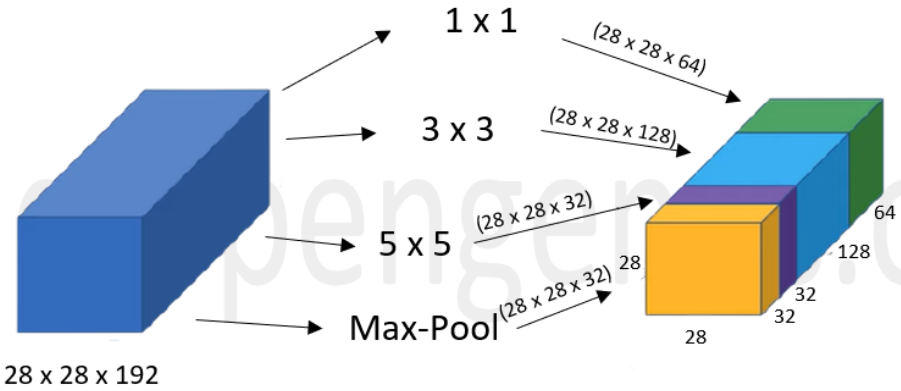
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

3 Inception Resnet

3.1 Introduction

Inception is a convolutional neural network for assisting in image analysis and object detection, and got its main idea from Googlenet model. The idea of the Inception module is to use filters of different dimensions simultaneously. In this way, several filters with different dimensions (convolution and pooling filters) are applied to the input. Then their output is concatenated. In this case, feature maps will contain different features. Each module contains several convolution and parallel pooling operations. The figure below shows a block diagram of a custom Inception module.In short, it means a layer consisting of several layers, each of which tries to obtain different information with different filter sizes and simultaneously provide this information to the next layer for processing. Again, the idea here is to get rich information from every level as much as possible.

The idea of the authors originated from here, we know that large convolutional networks are true that they have a lot of power, but due to having a large number of parameters, they are subject to overfitting. If you want to uniformly increase the size of the network like VGG, the calculations will increase in the same way, and if we want to go in the opposite direction, although thin networks can be used in theory, they are very non-optimal in practice. That's why they came to try to create an approximation of this thin structure that doesn't have the problems mentioned above. Their idea was to say that convolution filters with different sizes can cover different parts of information. Therefore, different layers with different sizes of filters are created, each of them starts working on the data and each of them extracts some information from it, then finally all these are integrated together and presented to the next layer. Throwing the pooling layer in the middle just because it got a good response after seeing the previous architectures!



After while the inception idea compound with resnet model.

3.2 Architercutre

Inception-ResNet-v2-B is an image model block for a 17 x 17 grid used in the Inception-ResNet-v2 architecture. It largely follows the idea of Inception modules - and grouped convolutions - but also includes residual connections.

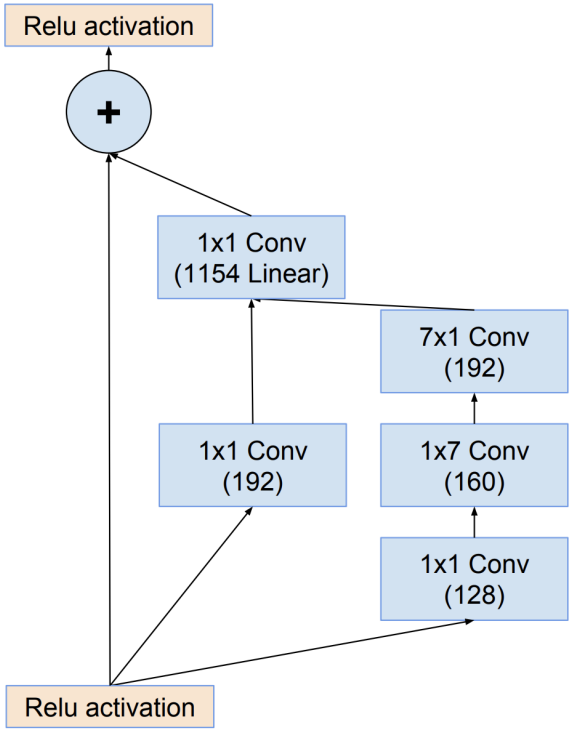


Figure 17. The schema for 17 × 17 grid (Inception-ResNet-B) module of the Inception-ResNet-v2 network.

4 Compare

While Inception focuses on computational cost, ResNet focuses on computational accuracy. Intuitively, deeper networks should not perform worse than the shallower networks, but in practice, the deeper networks performed worse than the shallower networks, caused not by overfitting, but by an optimization problem. The Inception-ResNet network is inspired by ResNet, which introduces the residual structure of ResNet in the Inception module. Adding the residual structure does not significantly improve the model effect. But the residual structure helps to speed up the convergence and improve the calculation efficiency. The calculation amount of Inception-ResNet-v1 is the same as that of Inception-V3, but the convergence speed is faster.

Residual network with addition tries to learn difference in learned features and if the learned features are not useful in final decision then the weight will become zero hence it will not overfit. Increased complexity of architecture Implementation of Batch normalization layers since ResNet heavily depends on it Adding skip level connections for which you have take into account the dimensionality between the different layers which can become a headache.

Network	N	Top-1 Error	Top-5 Error
ResNet (He et al. 2015)	6	N/A	3.6%
Inception-v3 (Szegedy et al. 2015b)	4	17.3%	3.6%
Inception-v4(+Residual)	4	16.4%	3.1%

Problem 3

In this part, you will use a ResNet architecture to solve an imagery classification problem. You have to work with the Intel Image Classification, which consists of 6 classes with 25k samples. You are allowed to utilize a pre-trained model and apply the transfer learning technique. Also, you are free to use other alternatives to ResNet architectures, such as Inception-ResNet, ResNext, Wide Residual Networks (WRNs), and DenseNet. An extra mark will be awarded to the student who achieves the highest accuracy on the test data in the class (+20 pts)

Solution. In all of the models that will discuss for the Intel Image dataset the Adam optimizer used with the LEARNING RATE=0.001.
Note: In every fig the red line is the train data and the blue line is considered as the valid data information.

5 Preprocessing transforms

As we know for model improvement and better generalization and for avoiding overfitting problems there are a lot of solutions such transform during the train. this transformation will help the model for watching images from another view in each epoch and learn them better for reaching better results. we used such transform for train data and test data(train

transform is different from test data): (Note: The resize transform used because some images didn't had (150,150)pixel size and it makes problem for model input so we used this method.)

5.1 Train Transform

```
        transforms.Resize((150,150)),
        transforms.RandomHorizontalFlip(p=0.5), # randomly flip and rotate
        transforms.ColorJitter(0.3,0.4,0.4,0.2),
        transforms.ToTensor(),
        transforms.Normalize((0.425, 0.415, 0.405), (0.205, 0.205, 0.205))

```

5.2 Test Transform

```
        transforms.Resize((150,150)),
        transforms.ToTensor(),
        transforms.Normalize((0.425, 0.415, 0.405), (0.255, 0.245, 0.235))

```

6 Resnet50

6.1 Model Architecture

In this model, we used the Resnet50 with pre train weights. we add some linear layer at the end of the model :

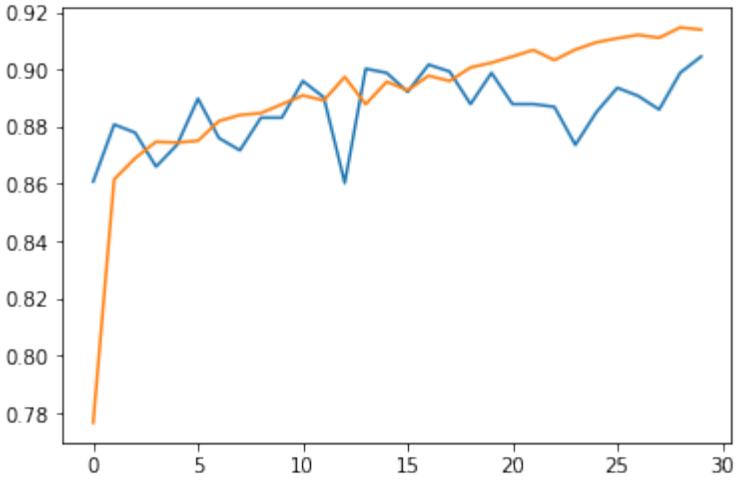
```
    Linear(2056,1024)
    ReLU()
    Dropout(0.2)
    Linear(1024,256)
    ReLU()
    Linear(256,64)
    ReLU()
    Linear(64,6)

```

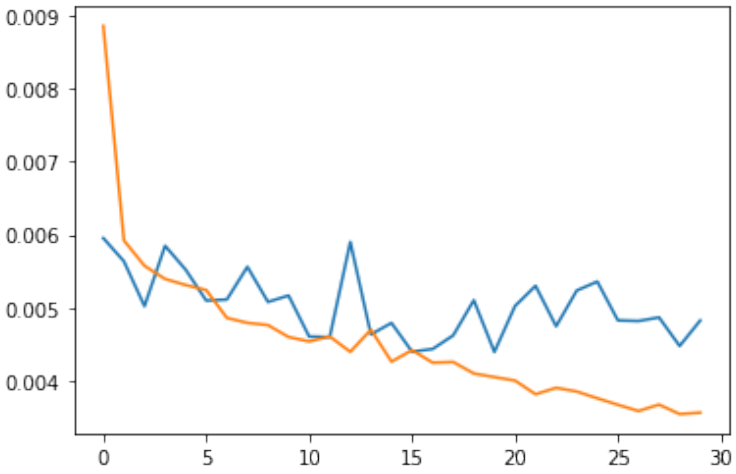
6.2 Results

Lets see the results :

6.2.1 Accuracy



6.2.2 Loss



7 Densenet121

7.1 Model Architecture

In this model, we used the denseness with pre train weights. we add some linear layer at the end of the model :

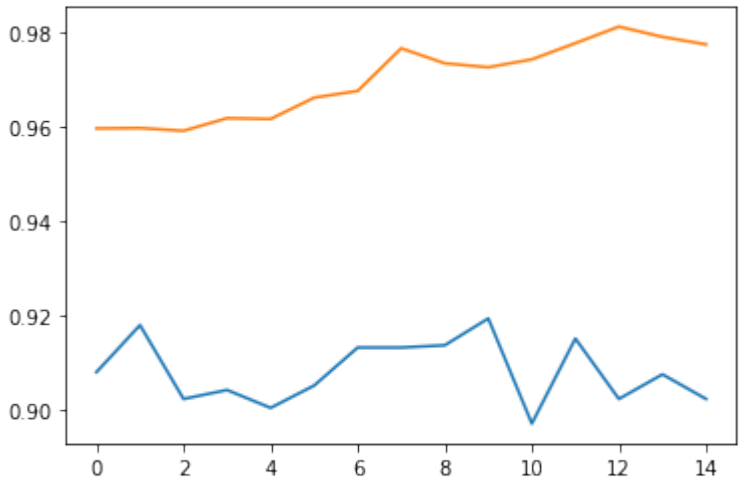
```
    Linear(1024,128)
    ReLU()
    Linear(128,6)

```

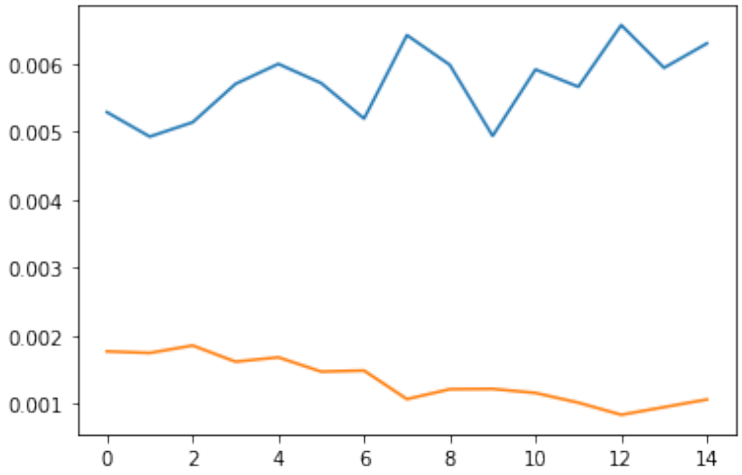
7.2 Results

Lets see the results :

7.2.1 Accuracy



7.2.2 Loss



8 WideResnet

8.1 Model Architecture

In this model, we used the WideResnet with pre train weights. we add some linear layer at the end of the model :

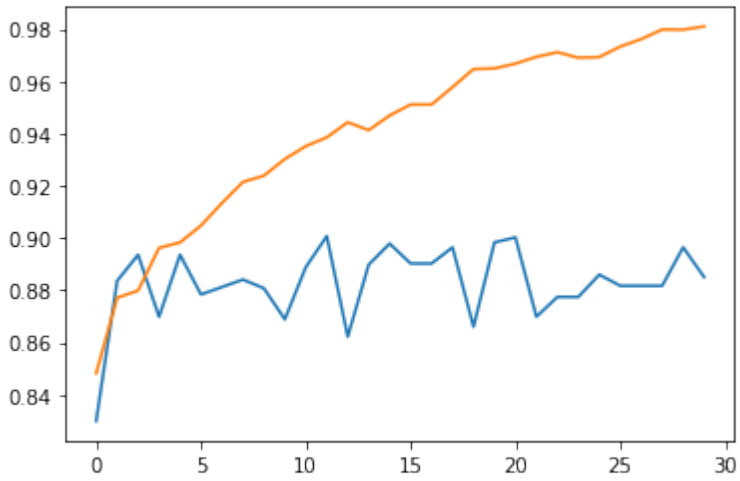
```
    Linear(2056,1024)
    ReLU()
    Dropout(0.2)
    Linear(1024,256)
    ReLU()
    Linear(256,64)
    ReLU()
    Linear(64,6)

```

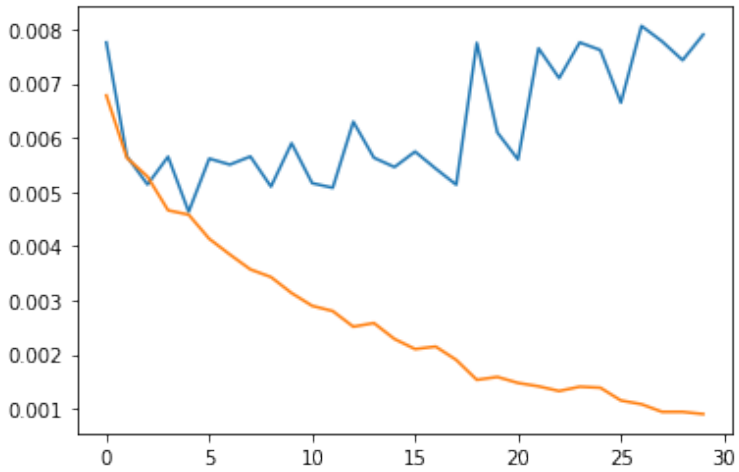
8.2 Results

Let's see the results :

8.2.1 Accuracy



8.2.2 Loss



9 Resnext101

9.1 Model Architecture

Using WideResnet with pre-train weights, we developed this model. At the end of the model, we add some linear layers:

```
    Linear(2056,1024)
    ReLU()
    Dropout(0.2)
    Linear(1024,256)
    ReLU()
```

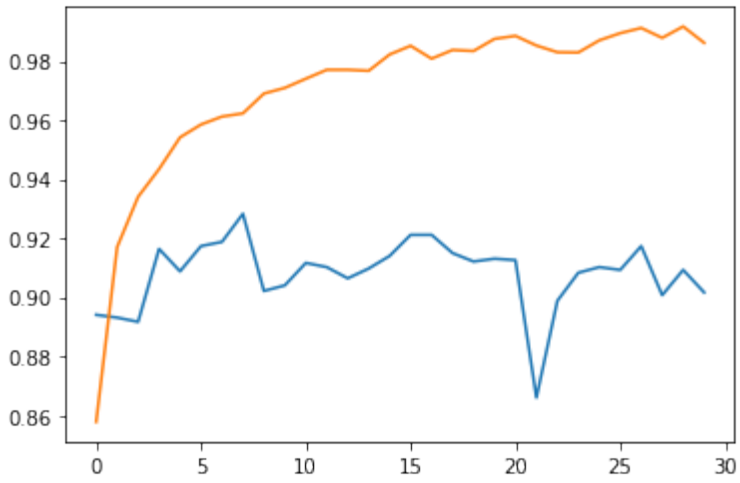
```
    Linear(256,64)
    ReLU()
    Linear(64,6)

```

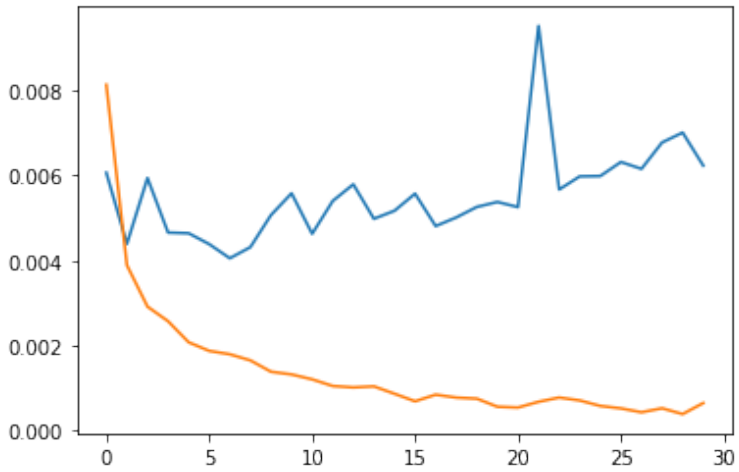
9.2 Results

Let's see the results :

9.2.1 Accuracy



9.2.2 Loss



10 Resnext34

10.1 Preprocessing

Before talking aboat the model arch we used some extra preprocessing for searching the better result: we use 4 augmentation for train data and increased train data 4 times.we used

cv2 library.

```
        rotate(img, angle=45, mode = 'wrap') # rotate the image
        np.fliplr(img) #flip right the image
        np.flipud(img) #filp down the image
        random_noise(img,var=0.2**2) # adding noise to the image

```

each image have been 4 time augmented and train data length became aboat 60k.

10.2 Model Architecture

Using WideResnet with pre-train weights, we developed this model. At the end of the model, we add some linear layers:

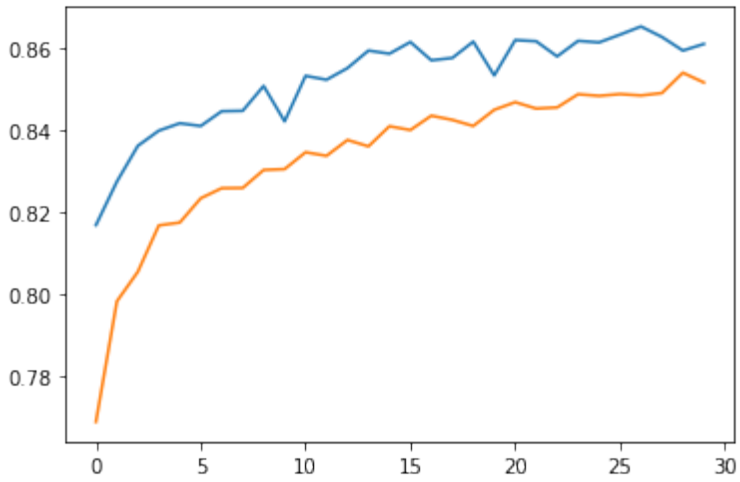
```
    Linear(512,1024)
    ReLU()
    Dropout(0.2)
    BatchNorm1d(1024)
    Linear(1024,512)
    ReLU()
    BatchNorm1d(512)
    Linear(512,256)
    ReLU()
    Dropout(0.2)
    BatchNorm1d(256)
    Linear(256,128)
    ReLU()
    Dropout(0.2)
    BatchNorm1d(128)
    Linear(128,64)
    ReLU()
    Linear(64,6)

```

10.3 Results

Let's see the results :

10.3.1 Accuracy



10.3.2 Loss