The L3BNet, Computer Vision Project Report

CS7-GV1
Jinwei Yuan
17306137

Introduction

This project is aimed to make myself understand the different aspects and characteristics of deep learning and convolutional neural network(CNN). Using CNN in classification problems (image classification for instance) is a good start and a good practice to gain knowledge and experience about deep learning. In this report, I descripts the model I make, L3BNet, which means a line-shape network model with 3 (convolutional) blocks. I try some parameters on my model, and also do comparison with two state-of-art models, AlexNet [1] and ResNet18 [5].

Theoretical background

The LeNet5 by LeCun [6] (1998) is considered as one of the earliest paper for deep learning. **Filters**, for example, an edge detection filter can extract edge **features** from images, training using multiple **convolution layer** with a lot of filters is able to get suitable parameters to do image classification. The LeNet5 introduces methods like **convolution**, **down sampling**, non-linear **activation functions** such as tanh and sigmoid.
The AlexNet by Alex Krizhevsky(2012) has more depth (7 layers) than LeNet5, and it reaches a high accuracy compared to other classification algorithms. The AlexNet introduces **ReLU** as the activation function to save time for calculation, **dropout** [1][3] to prevent over-fitting and **max-pooling**.
Network-in-network (2013) introduces that using **1x1** convolution kernel can improve reduce the error rate.
Using multiple 3x3 kernels instead of large-size kernels [2] (VGG, 2014) and bottleneck from GoogleNet [4] (2014), **Batch-normalize** (Google, 2015) are also useful methods to build deep learning models. The addition operation of two convolution matrix outputs is useful as well (ResNet,2015)

Implementation

The deep learning framework I use is Pytorch. I use the sample code and modify it to build my own model (L3BNet), resume training, load AlexNet and ResNet18 to train and evaluate the Top1 and Top5 accuracy.
This is the L3BNet model I made, I use some concepts from VGG (using only 3x3 kernel), and some ideas from GoogleNet (Batch-normalization), and basic ideas from LeNet5 and AlexNet, combines them together and generade my own model.

```
y = self.conv_1(x)   # 3x3 , 32
y = self.relu_T(y)
y = self.batch_norm_32(y)
y = self.pool_2_2(y)


y = self.conv_32_64_3(y)
y = self.conv_64_64_3(y)
y = self.relu_T(y)
y = self.batch_norm_64(y)
y = self.pool_2_2(y)

y = self.conv_64_128_3(y)
y = self.conv_128_128_3(y)
y = self.relu_T(y)
y = self.batch_norm_128(y)
y = self.pool_2_2(y)

# add new
y = self.conv_128_256_3(y)
y = self.conv_256_256_3(y)
y = self.relu_T(y)
y = self.batch_norm_256(y)
y = self.pool_2_2(y)

y = y.view(y.size(0), -1)
y = self.fc_1(y)
y = self.relu_4(y)

y = self.batch_norm_4(y)
y = self.dropout(y)
y = self.fc_2(y)
```

Input Image
conv3, 32
ReLU
Batch-Normalize
Maxpool2
conv3, 64
conv3, 64
ReLU
Batch-Normalize
Maxpool2
conv3, 128
conv3, 128
ReLU
Batch-Normalize
Maxpool2
conv3, 256
conv3, 256
ReLU
Batch-Normalize
Maxpool2
Fully Connect, 2048
ReLU
Batch-Normalize
Dropout 0.5
Fully Connect, 200

Experiments & Results

Add the beginning, I combine some knowledge gained from different deep learning networks (LeNet5, AlexNet, VGG, GoogleNet, ResNet…), I make two base models, one is a line-shape models with one middle convolution block, the other is with 2 middle convolution blocks then do addition. The dataset I use is tiny ImageNet dataset. There are 100,000 images of 200 classes (500 for each class) in the training dataset, there are 10,000 images of 200 classes (50 for each class) in the validation dataset.

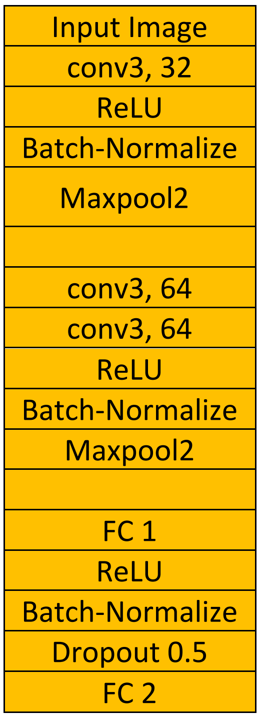

Fig 1: L1BNet Model

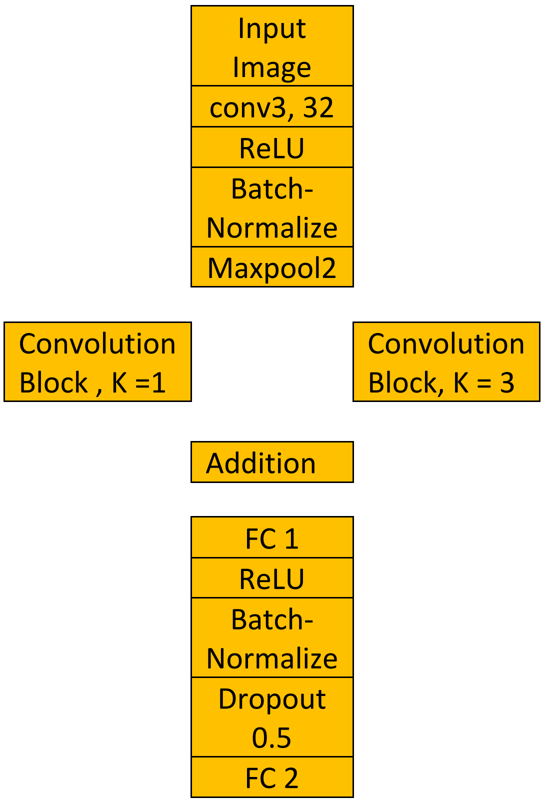

Fig2: Simple Addition Model

A. Shape / Structure of a small model: Split with addition later OR Line-Shape (from top to bottom straightly)
B. Depth and Width of a network
C. Data augmentation strategies (cropping, scaling …etc.)
D. Initial learning rate
E. Effect of dropouts
F. Filter size, number of filter, stride, padding etc.
G. Activation function
H. Loss function
I. Optimizer
J. Other

(A) I run the above Simple Addition Model (Fig 2), for 35 epochs, the best top1 accuracy is 0.32, the training time is around 0.9 minute for each epoch. For L1BNet Model (Fig 1), the result is 0.37, the training time is around 0.36 minute for each epoch. In this case, I pick the Line-Shape one(L1BNet).

(B) I add one more middle block (con3, con3, ReLU, Batch-Normalize, Maxpool2) to the line-shape model, the best top1 accuracy becomes 0.391, 11.5 mins for 25 epochs. Now it becomes L2BNet (line shape with 2 blocks) (Fig 3)

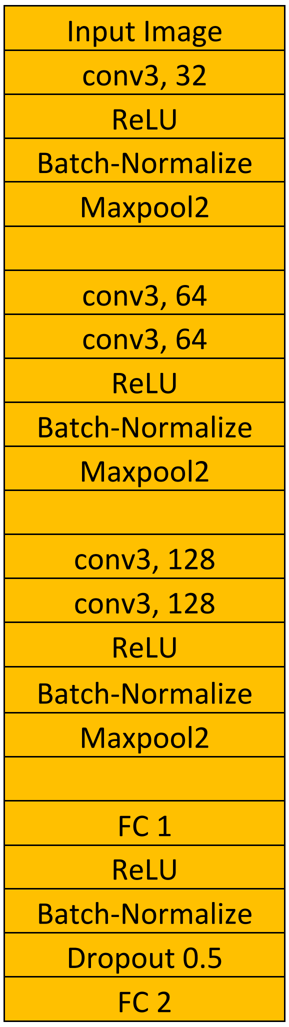

Fig 3: L2BNet

(J) Then I change the output parameter of the first fully connect layer from 1024 to 2048, the result is 0.393, it is very similar to the previous result 0.391 with a slight increase. So I keep the changes.

(I) I am using Adam optimizer, I want to compare with SGD. I use SGD and get the best top1 accuracy which is 0.38, time for each epoch is similar, so I decide to use Adam optimizer.

(B) I want to test whether adding one more middle convolution block (con3, con3, ReLU, Batch-Normalize, Maxpool2) is beneficial to the top1 accuracy, I try running 50 epochs (around 30 mins) and the result is 0.45 (best accuracy). I keep this change and my model is now becoming **L3BNet**.

(C) Augmentation like horizontal flip is useful to avoid over-fitting, if not using it the accuracy of my model will decrease from 0.45 to 0.39. Adding a random vertical flip or cropping decreases the accuracy by 0.15 and 0.20, so vertical flip is not good. I try adding a random rotation (-45,45) to the data augmentation list and test for 100 epochs in 59 mins, the top1 accuracy reaches 0.47. Scaling image (or blurring images) does not make sense to me, because my network works ok with 64*64 images. So in my data augmentation list, there are a random horizontal flip and a random rotation (-45,45).

(E) Using dropout helps a lot to avoid over-fitting. Without dropout in the last fully connect block, the top1 accuracy is only 0.41, 6 percent lower compared to the one with dropout, 0.47.

(D) To determine what initial learning rate is suitable for my model, I experiment with 0.01 and 0.001. Applying 0.01 gives me a 0.26 top1 accuracy, and I get a 0.47 top1 accuracy with 0.001. According to the result, 0.001 is ideal for L3BNet.

(F) For filter size, I try using filter size 5 and 3, the top1 accuracy from size 5 is a bit lower 0.46 compared to the size 3 which is 0.47, I think using size 3 is ok. For the number of filters, and I try not to double number of filters in the last middle convolution block (from 128 to 256), the top1 accuracy just fall by 6%.

(H) Loss Function. I test with nearly all the build-in loss functions in Pytorch, most of other does not work for my model, NLL loss. So I only compare NLL loss and Cross Entropy. (Fig4)

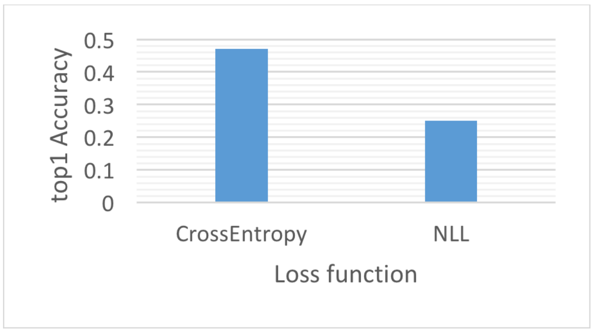

Fig 4

(G) I test different activation functions such as ReLU, ReLU6, ELU for my model, the top1 accuracy is ReLU>ELU>ReLU6, I have also try applying a additional Softmax in the end of fully connect block, but it just slow down the time to converge and it can not increase the top1 accuracy. （Fig 5）

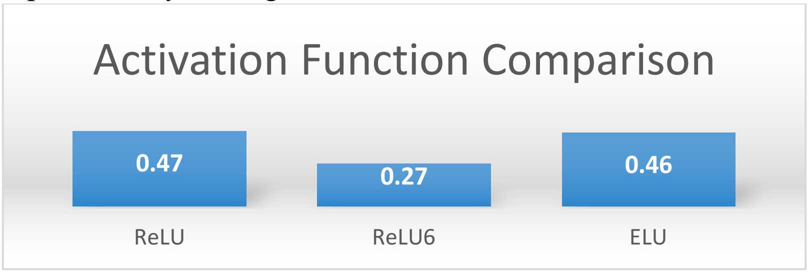

Fig 5

(J) I also set the factor of Reduce Learning Rate on Plateau Scheduler to 0.2, patience to 4, which means if there are no accuracy increase in 4 epochs, decrease the learning by multiplying the factor. This helps the mode converge faster without losing accuracy.

Results of L3BNet and Comparison with AlexNet and ResNet18

Here are the results of top1 and top5 accuracy in 50 epochs from L3BNet, not pre-trained AlexNet, pre-trained AlexNet, not pre-trained ResNet18 and pre-trained ResNet18.（Fig 6, 7,8,9,10,11）

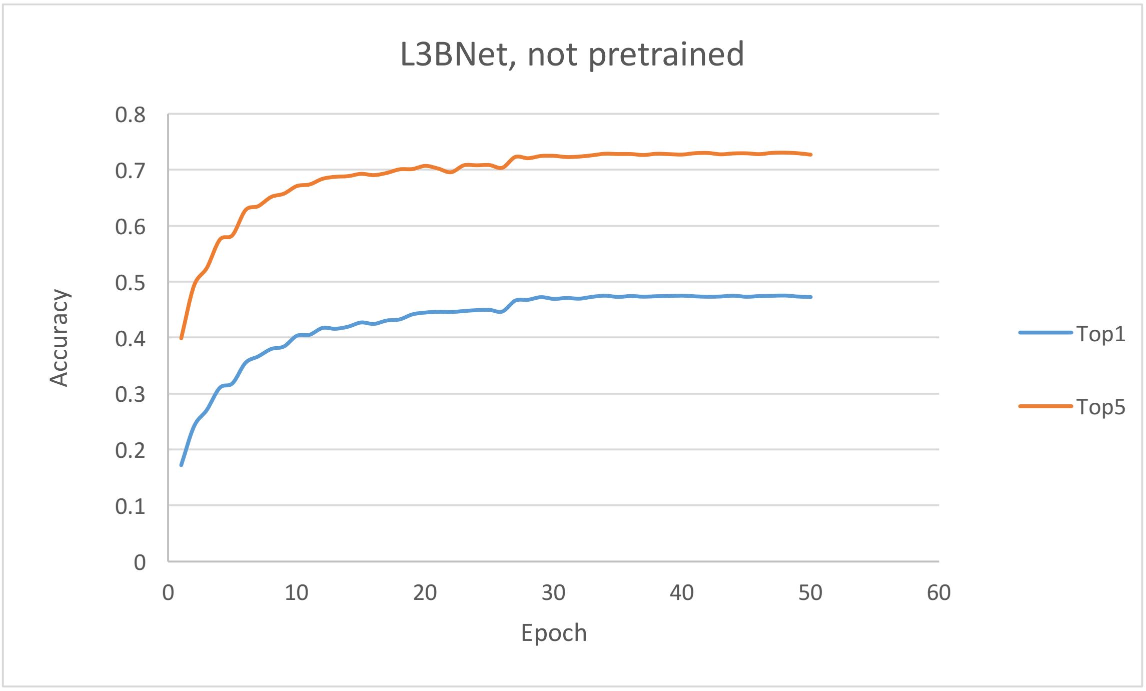

Fig 6

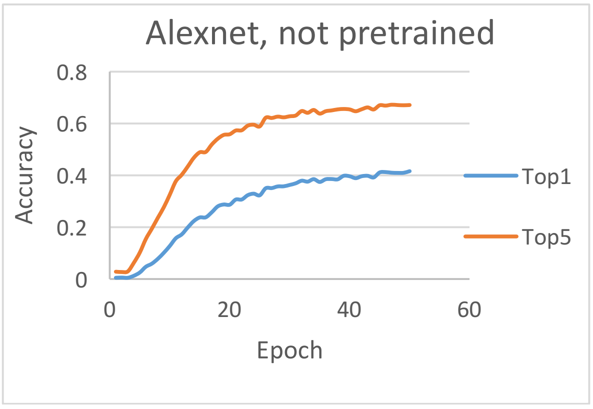

Fig 7

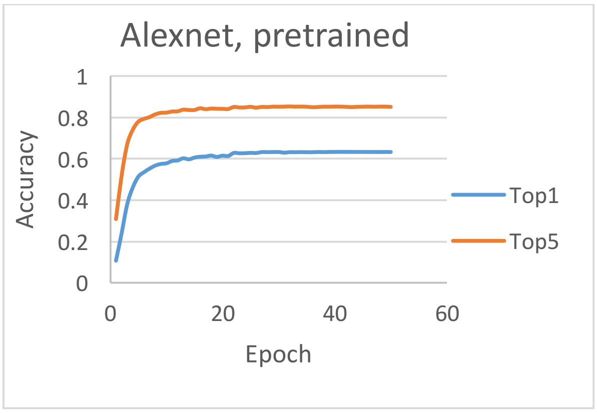

Fig 8

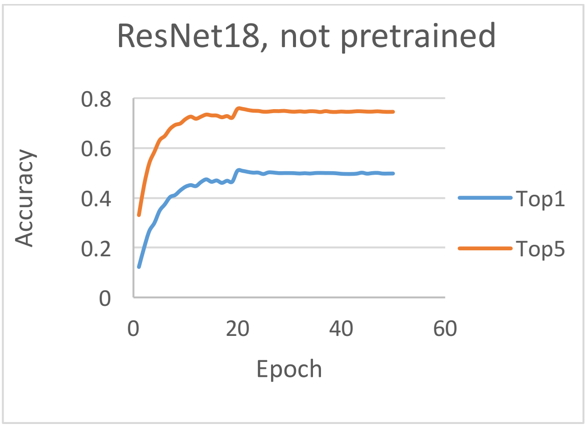

Fig 9

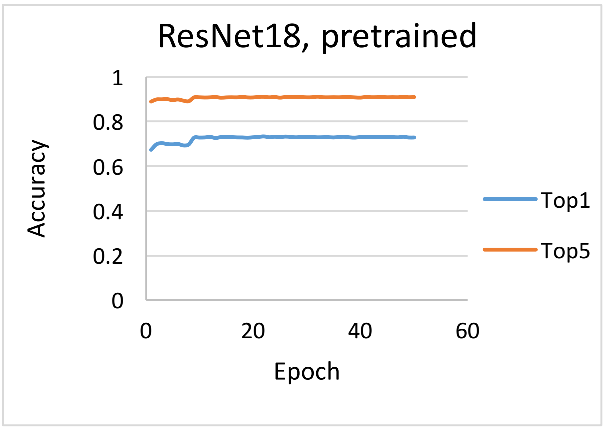

Fig 10

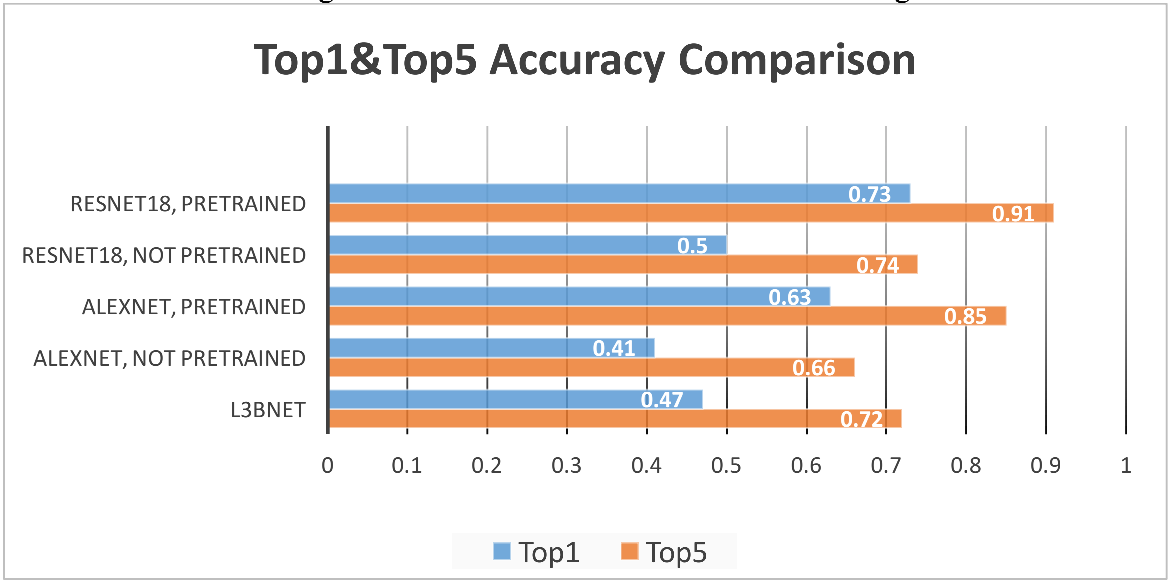

Fig 11

From top1 and top5 accuracy among L3BNet, not pre-trained AlexNet, pre-trained AlexNet, not pre-trained ResNet18, pre-trained ResNet18, the pre-trained ResNet18 performs the best both in top1 and top5 accuracy (Fig11). L3BNet has a close top1 (0.47) and top5 (0.72) comparing to not pre-trained ResNet18 (0.50, 0.74) and it performs a bit better (6%) than not pre-trained AlexNet when using 50 epochs. AlexNet needs around 45 epochs to converge, which is the largest among the five. Time consumption for training and validating in 50 epochs: L3BNet (30 mins) < AlexNet (76 mins) < ResNet18 (267 mins).

Confusion Matrix of 20 random classes for validation dataset. I create this by creating the confusion matrix of all 200 classes in validation set, and use the sklearn confusion matrix method to generate 200-class confusion matrix and use the parameter ‘labels’ to pick 20 classes randomly. From Fig 12 and 13, the L3BNet performs well in validating only 20 classes. There are only two diagonal elements are under 0.8 which means the accuracy is high when considering only 20 classes.

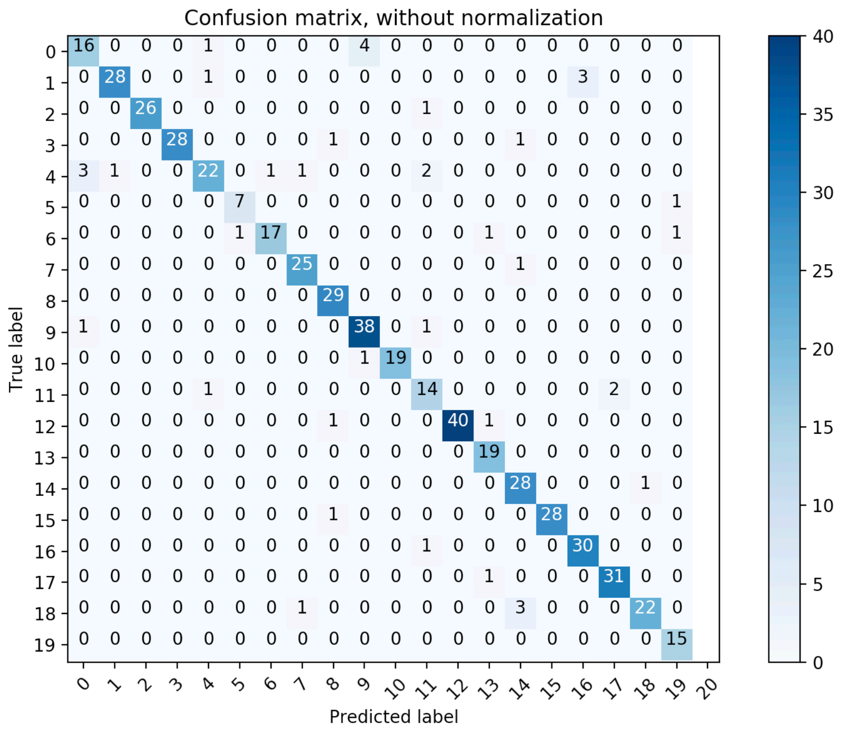

Fig 12

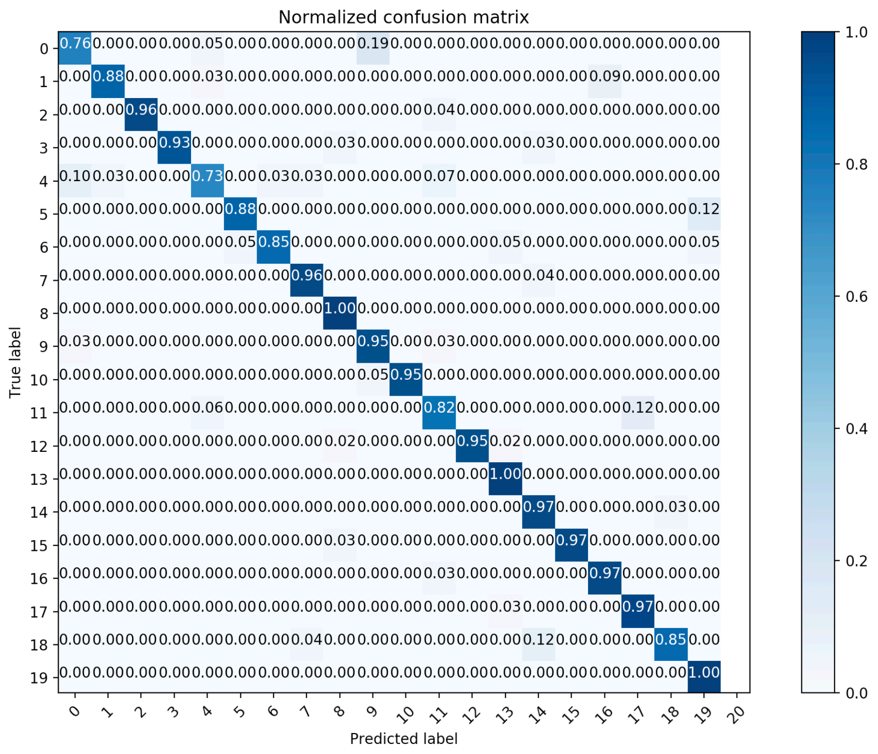

Fig 13

Analysis

From top1 and top5 accuracy in L3BNet, it converges near epoch 30. According to the experiments and results:

- The **shape or base architecture** influences the top1 and top5 accuracy the most. Because having a branch or not strongly influences the result of doing a convolution calculation.
- The depth of network: when a **convolution block** is defined, for instance, the convolution block is like (con3, con3, ReLU, Batch-Normalize, Maxpool2), the block of GoogleNet is like Fig 14 using concatenation, the block of ResNet is like Fig 15 addition. The addition operation takes a long time so one should consider carefully about whether it is worth to use addition operation. If stacking two blocks together improves the accuracy, increasing the model depth by adding more blocks should rise the accuracy, but the time will arise linearly.

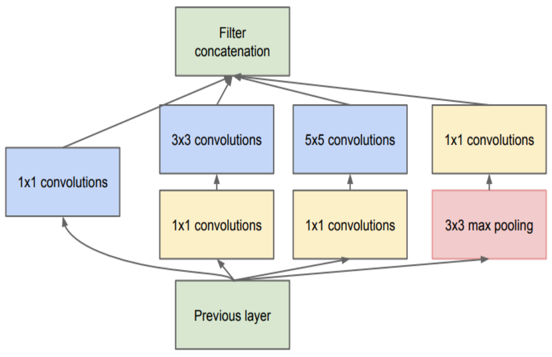

Fig 14

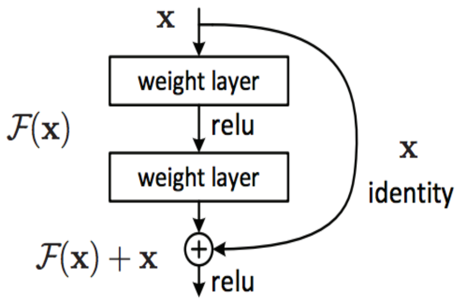
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Fig 15

- The filter size and number influences the accuracy by affecting the output of each convolution block. Filter size 3 is enough for my model, and it is better than size 5 because larger may fit the training set better, but having a more ‘blur’ output does not mean the result is good, having a larger filter size of 5 does not improve the accuracy.
- The filter number takes an important part in affecting the accuracy. The larger the filter number is, the better the accuracy will be, at the same time, the number of calculation and period of time will rise. This is because number of parameters will be larger significantly when the number of filters increase.
- Augmentation: The top1 and top5 accuracy increase obviously when I applying random horizontal flip and random rotation to training images. The reason is that the number of images is not enough, doing augmentation like flip and rotation is equal to generating more training images.
- Dropout: The accuracy increase significantly (6%) when a dropout is applied to the fully connect block. This is because the backward propagation is used to adjust the weights based on reducing the training error rates (loss), which makes the weights fit the training set too much. Dropout which set output of connections to zero by a pre-defined probability reduces over-fitting.

Summary and Future

In conclusion, I generate a deep learning network named L3BNet based on doing experiment and test on different hyper-parameters, the shape/ architecture, depth of network, optimizer, data augmentation, dropout and filter numbers take more important part in determination of the top1 and top5 accuracy than other parameter does. When building a CNN network, theses parameters should be considered carefully.
In the future, I will improve the performance, mostly the top1 accuracy, of L3BNet by making it deeper, trying more shape and filter number on it.

References

[1] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks (2012) ALexnet
[2] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. (2014). VGG
[3] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. (2014). Dropout
[4] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. (2014) GoogleNet
[5] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. (2014) ResNet
[6] Yann, L., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. (1998)