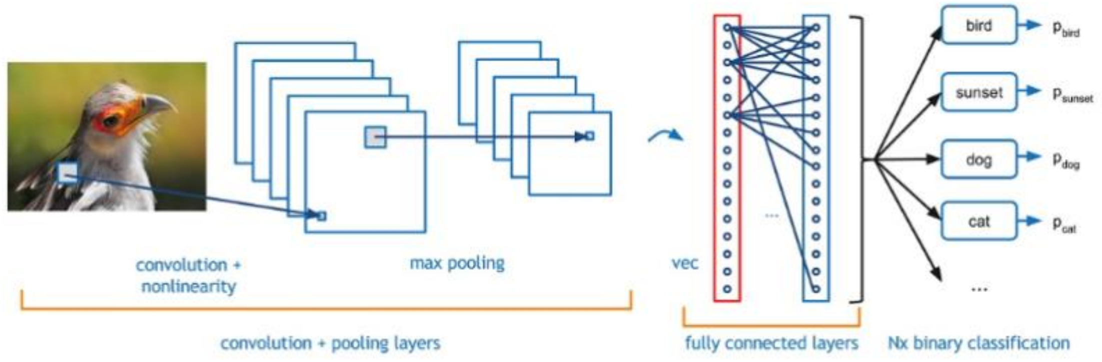
EE 569: Homework 5

1. CNN Architecture and Training

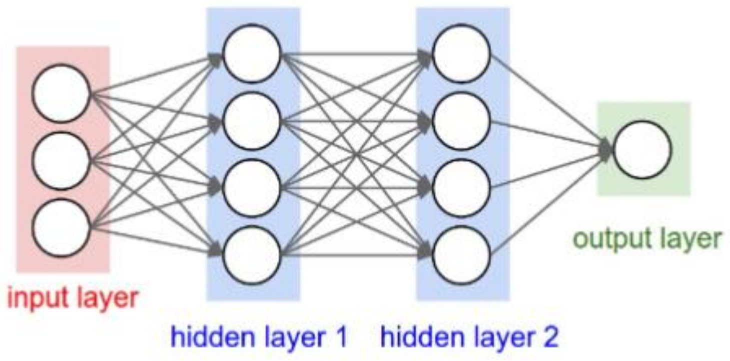
1.1. CNN Architecture



1.1.1. Fully Connected Layer

Fully connected layer play as a classifier in whole neural network. There will be no less than one fully connected layer after convolutional layer and max-pooling layer. The function for fully connected layer is to map all "distributed feature representation" obtained from convolutional layer and max-pooling layer to the sample markup space. In this layer all input data will be weighted and plus a bias then pass an activation function, so this layer can be regarded as multi-layer perception. In order to improve FC layer performance, the activation function chosen by FC is ReLU function.

Due to heavy number of parameters in FC layers (in some complex network which may take about 80% of the total parameters), some of the efficient network model use Global Average pooling (GAP) to replace FC layers. Comparing to FC the GAP have a better prediction ability.

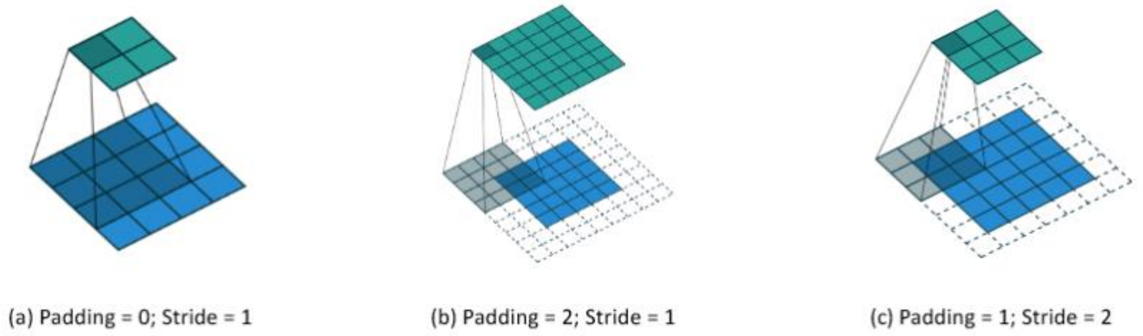


1.1.2. Convolutional Layer

Convolutional layer normally is the first layer in neural network. It is used to extract features from input. It is a feed-forward neural network. The response from artificial neurons are only based on coverage areas so this layer need less parameters than MLP, because of this it has a better performance than MLP when dealing with large scale images.

A convolutional network contains more than one convolutional layer. It is also followed by some pooling layers. Because of the structure like this, CNN could utilize 2-dimentional structure of input image to get a better result. The parameters of this layer can be trained by back propagation algorithms. Comparing with other deep learning structures, CNN can give better results in image and speech recognition.

The work done by this layer is to use several convolution filters and do convolution to the input data and product some outputs with the same number of filters.


(a) Padding = 0; Stride = 1
(b) Padding = 2; Stride = 1
(c) Padding = 1; Stride = 2

1.1.3. Max-pooling layer

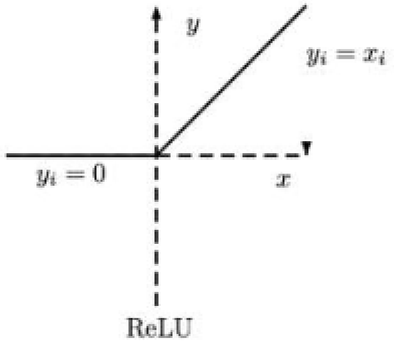
Pooling is a useful method to reduce output spatial size and preserve original information. In this task we will use max-pooling of stride = 1 and patch-size = 2. Max pooling can improve image offset, reduce the number of parameters to be trained of the entire network, and obtain more overall features.

Besides there are some pooling method, for example min-pooling, max-pooling and average pooling. In this homework, I will use max-pooling.

1.1.4. Activation Function

Activation function is a useful tool. This function is used to correct the result from convolutional layers and fully connected layers. This is a nonlinear function and it will improve the performance of the network. If we don not use this activation function, the multi-layer linear structure will be equivalent to a layer of perceptron. This kind of perceptron cannot handle XOR logic, so it cannot achieve good results.

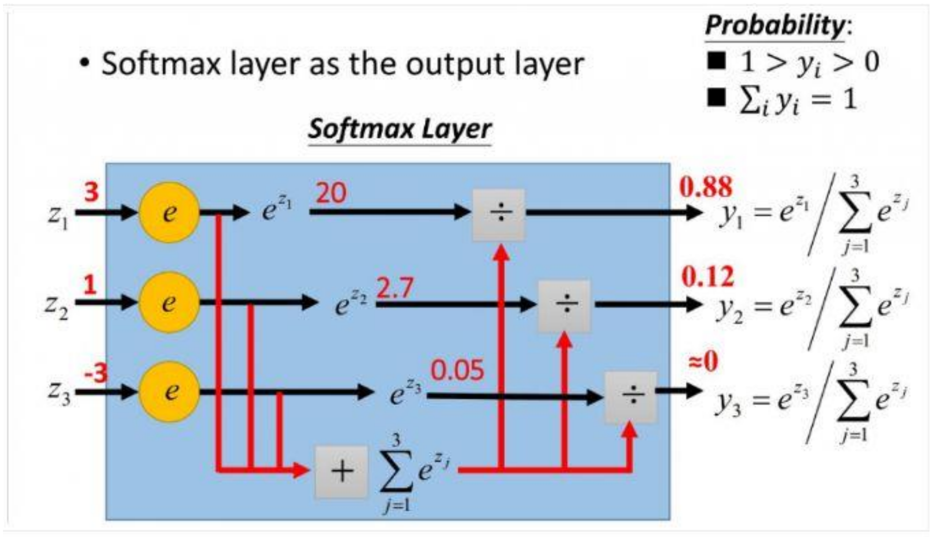
According to C.C. Jay Kuo's lecture, he introduced three different kinds of activation functions, Sigmoid ReLU and LeakyReLU, in this homework I will use ReLU.



ReLu is popular in image processing, this kind of function is close to how real neural cell in human brain work. But it is far from perfect, since it is non- zero-mean, so small learning rate is necessary and some neurons may died. ELU may help with this issue.

1.1.5. Softmax Function

Softmax is a useful tool, it is used at the last layer of the CNN. Because the results obtain from FC layers may vary greatly. Because when making classification, people want to obtain probability rate rather than some positive or negative numbers. Then additivity is very important in this operation. Then we can use exponential function to deal with it. This function is shown below:



• Softmax layer as the output layer

Probability:

■ $1 > y_i > 0$

■ $\sum_i y_i = 1$

Softmax Layer
z_1
3
z_2
1
z_3
-3
e
e^{z_1}
20
e^{z_2}
2.7
e^{z_3}
0.05
$\sum_{j=1}^{3} e^{z_j}$
0.88
$y_1 = e^{z_1} / \sum_{j=1}^{3} e^{z_j}$
0.12
$y_2 = e^{z_2} / \sum_{j=1}^{3} e^{z_j}$
≈0
$y_3 = e^{z_3} / \sum_{j=1}^{3} e^{z_j}$

For each out put $\mathrm{y_i} = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$. This function can normalize result from CNN and larger the different between different classes. The total sum of yi will equal to 1, and the maximum yi will be result class for classification.

1.2. Over-fitting

Over-fitting is common when training neural network. This phenomenon is mainly caused by not enough training data input. Under this circumstance, the network is trying to fit all training data with high accuracy. But for the testing data, the predict result may get worsen. There are several causes for over-fitting:

a. Incorrect selection of modeling samples, such as not enough samples, wrong selection for samples, fault sample labels. These insufficient sample data are not able to represent the predetermined classification rules;
b. Noise in samples will misguide network to treat noise as features;
c. The hypothetical model does not exist or the condition for hypothesis are not true;
d. Too many parameters in the network;
e. In neural network model, there may be some classification decision hyper-plane that are not unique, and the BP algorithm converge to a more complexity decision hyper-plane.

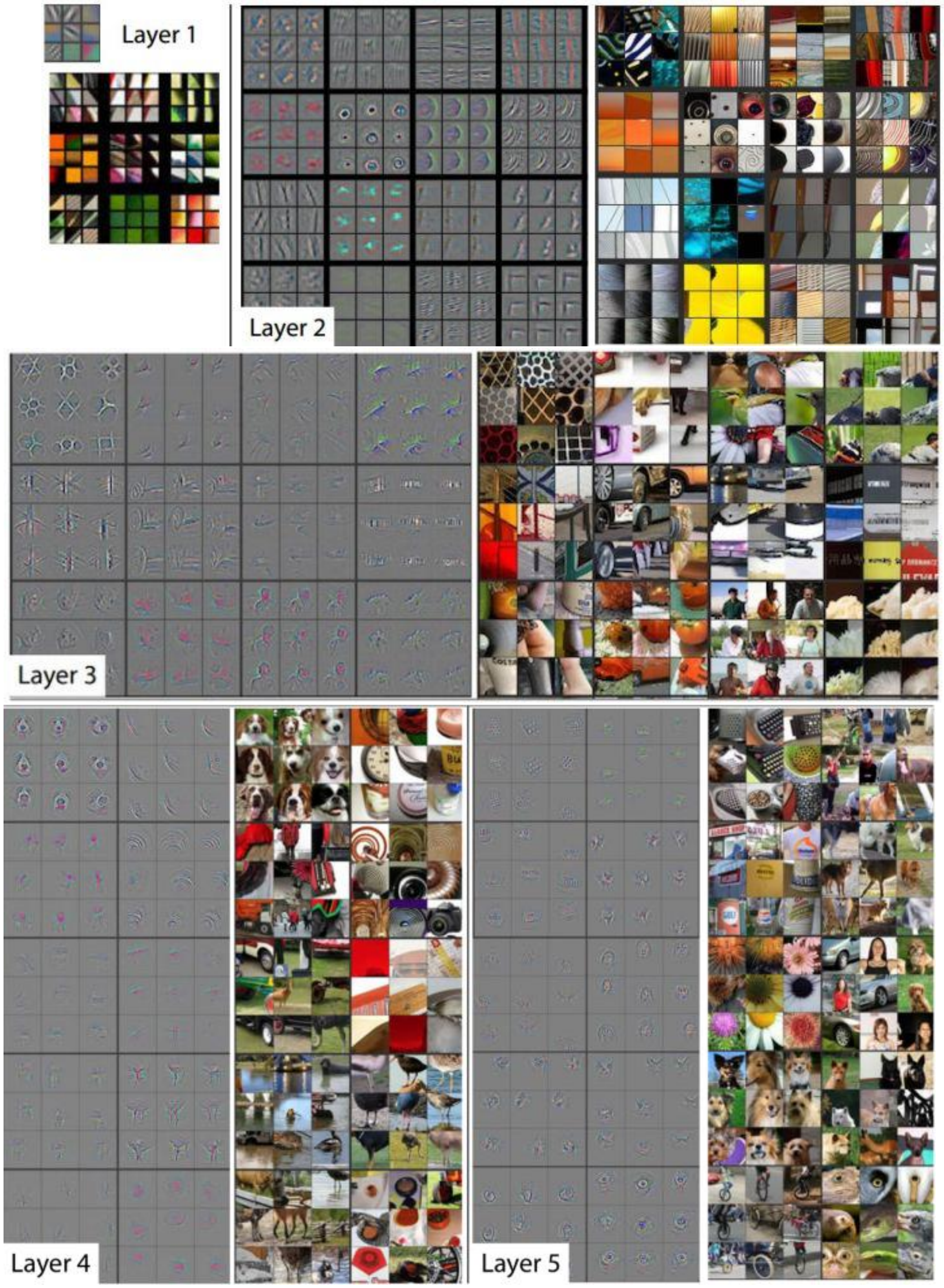
Ways to overcome it:

a. Get more data;
b. Use a proper network model, reduce the number of layer and the number of neurons;
c. Dropout;
d. Regularization, limit the weight during training before it is getting too large;
e. Correct the wrong label or delete wrong data.

1.3. Why CNN is better?

The reason for this is that CNN could extra useful features from training set automatically, comparing to previous method they only utilize so called hand-craft features. Also the number of feature extra by CNN is much larger and accuracy than traditional method. The image data can be directly used as input, not only does not require manual preprocessing of the image and additional feature extraction and other complex operations, but also its unique fine-grained feature extraction method

For example, when training the ImageNet, here it utilized anti-convolution to do feature visualization.


Layer 1
Layer 2
Layer 3
Layer 4
Layer 5

This visualization shows us the way neural network do. The second layer concentrate on corners and other edge or color information. The third layer has captured similar textures, and layer 4 is more category specific. At the fifth layer shows the entire object.

This kind of process is human like process and will get a better result.

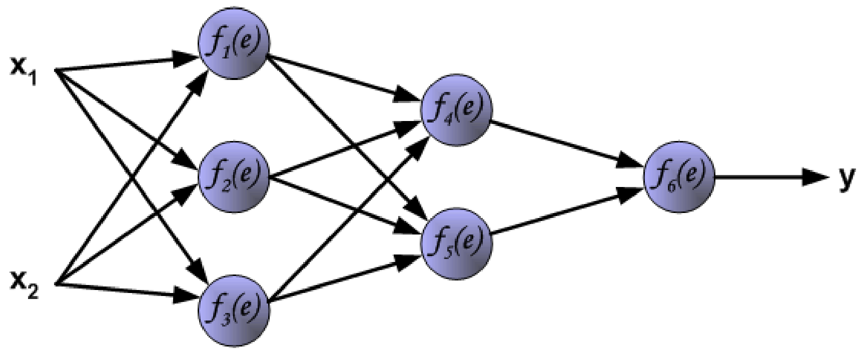
1.4. Backpropagation

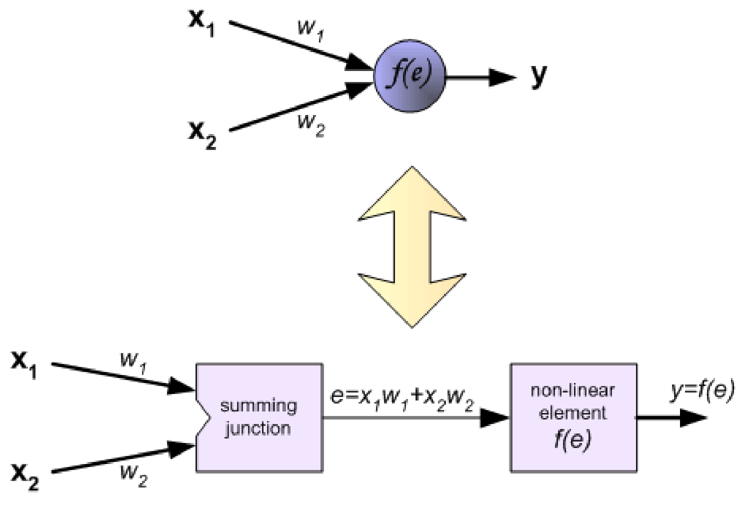
At first we will talk about loss function, one of the wildly used loss function is cross entropy.

$$\mathrm{L} = -\sum y(i)\log\left(a(i)\right)$$

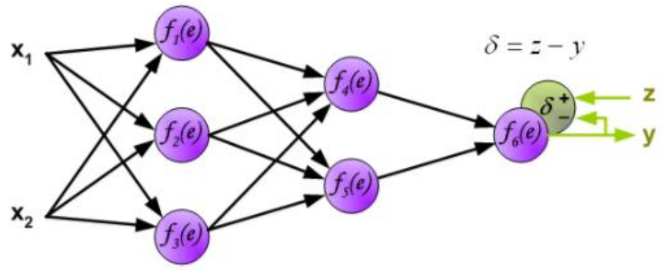
$y(i)$ is ground truth label and $a(i)$ is output from the soft max. Ideal loss function should be non-convex.

Backpropagation is a useful tool to train neural network, the following I will introduce how it work. For a more intuitive explanation, I used the following neural network:

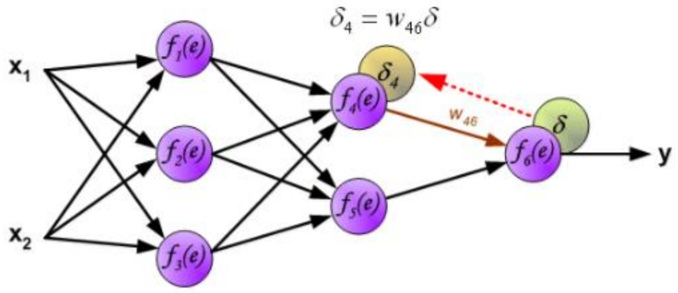


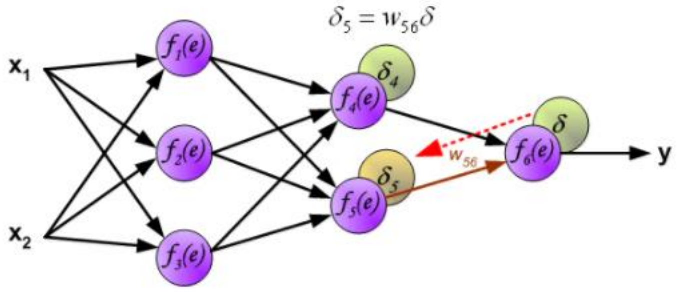


After the forward propagation, the next step is to calculate the output and then calculate the error. This error is called error signal, then this signal will pass back to previous layers and make modification to previous layers' weights.

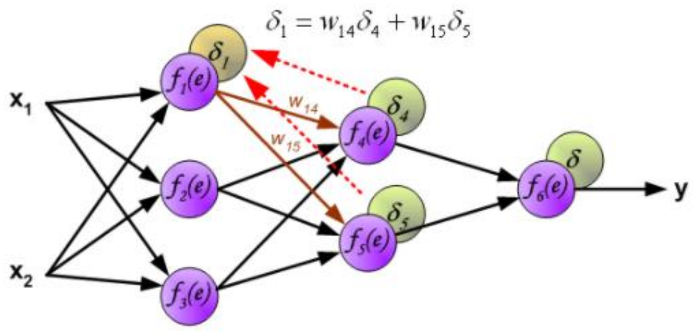


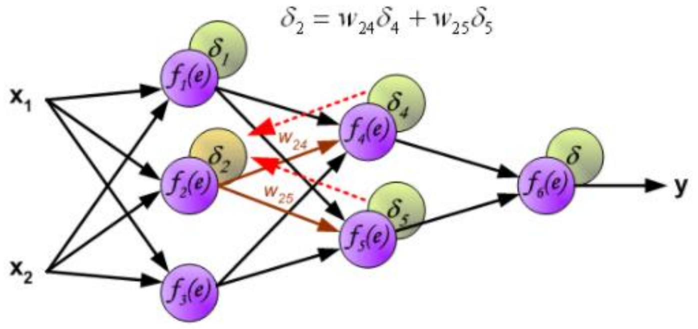
BP will transfer δ to all previous nodes, and weights will modified based on these δ.

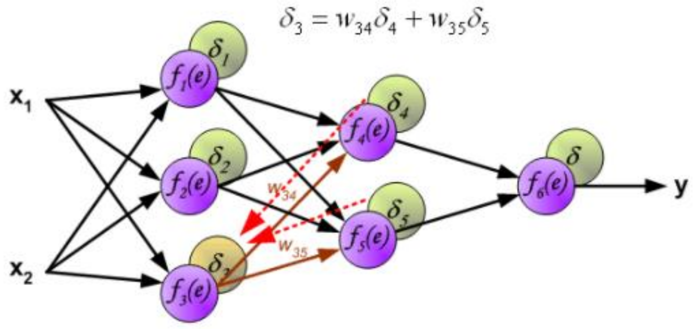




The weight coefficient w_{ij} is used to propagate the error signal δ. Only the direction of the data flow is changed (the signal propagation is weighted one by one from the output to the input). This technique is used for all network layers. The error δ according to the previous weight will obtain the individual error of each neuron $\delta_5, \delta_4, \dots, \delta_1$



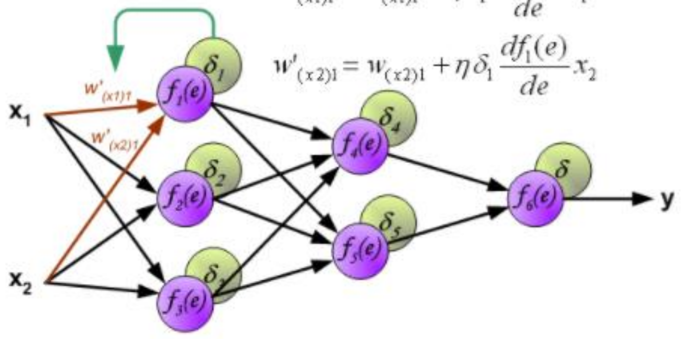




After calculation the errors of every neuron, w_{ij} will be updated with the following method: at here η is the learning rate

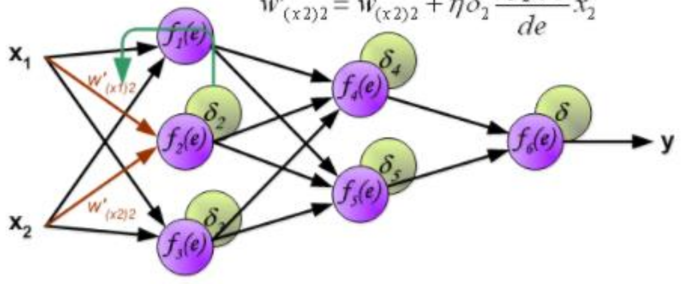
$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



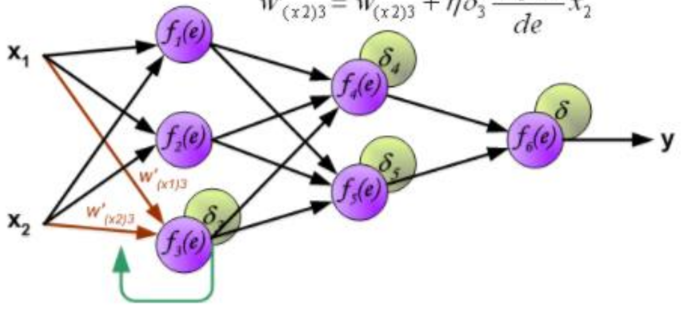
$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

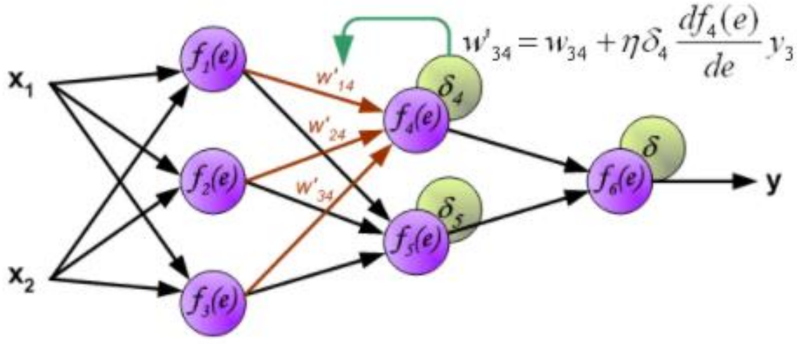
$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$



$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

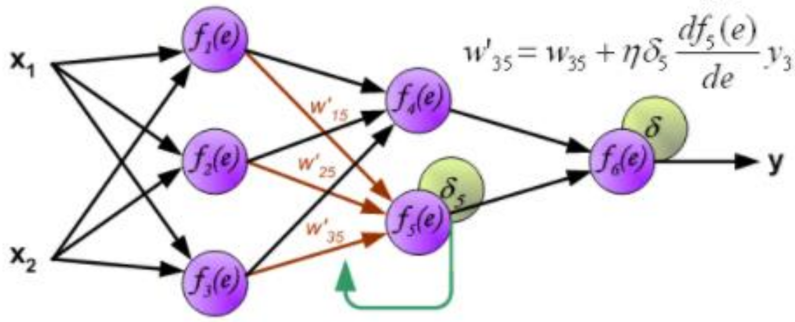
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

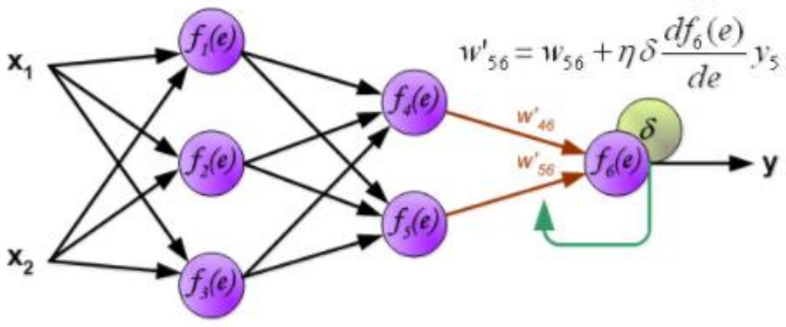
$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$



$$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$$

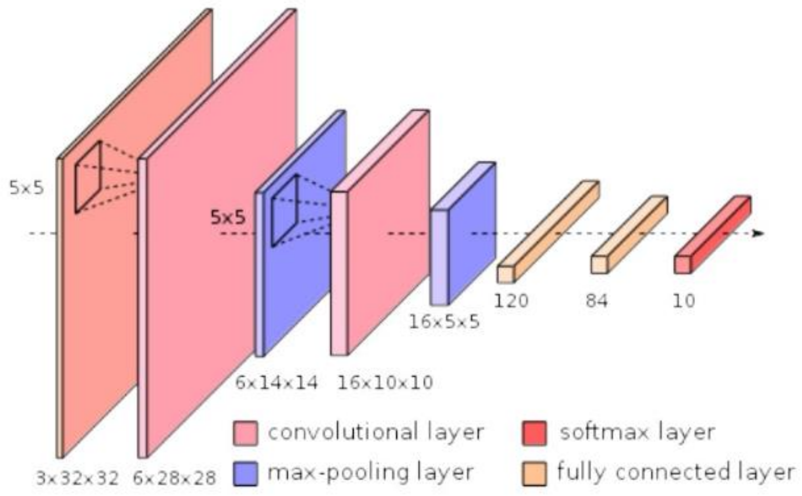
$$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$$



These w_{ij} are updated in each epoch.

2. Results and best parameters

2.1. Architecture of this LeNet 5



2.2. Result curves and parameter chosen

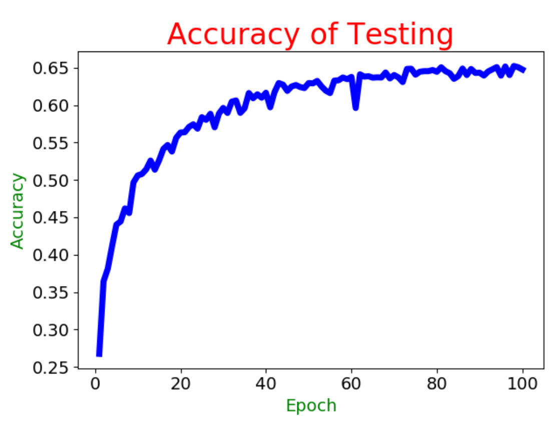
The 5*5 filter number of first and second convolutional layers are 6 and 16, the stride is 1 and no padding is used.

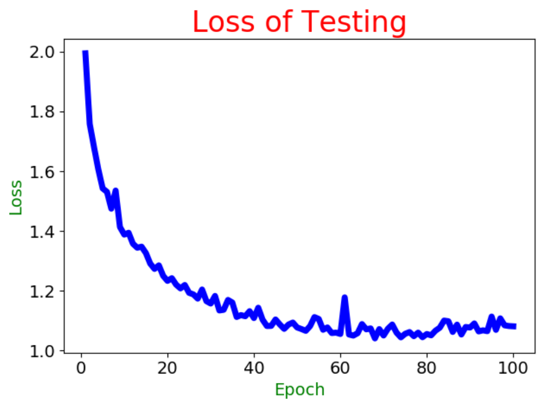
At here I choose the number of epochs equal to 100

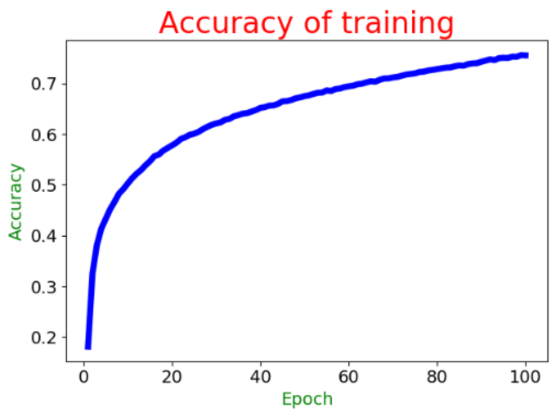
Number of test	Testing accuracy 100 epochs	Training accuracy 100 epochs	Learning rate	Batch size	momentum	decay	padding
1	0.6477	0.7551	0.00055	64	0.9	0.000001	valid
2	0.6195	0.7190	0.00055	64	0.9	0.00001	Valid
3	0.6241	0.7150	0.001	128	0.9	0.000001	Valid
4	0.5986	0.8692	0.003	128	0.9	0.000001	Valid
5	0.6129	0.6578	0.001	128	0.8	0.000001	Valid
6	0.6206	0.9041	0.00055	64	0.9	0.000001	Same
7	0.5501	0.5759	0.0001	64	0.9	0.000001	Valid
8	0.5859	0.6473	0.00055	128	0.9	0.000001	valid

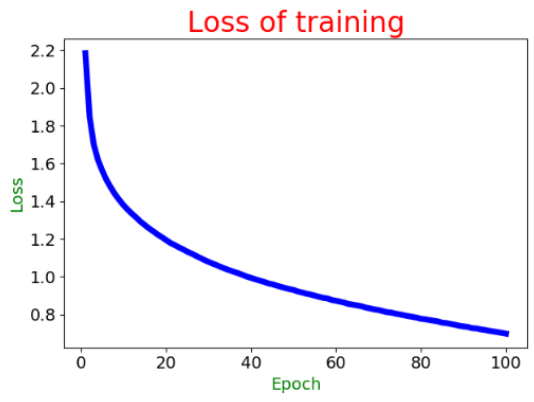
Results:

Test 1

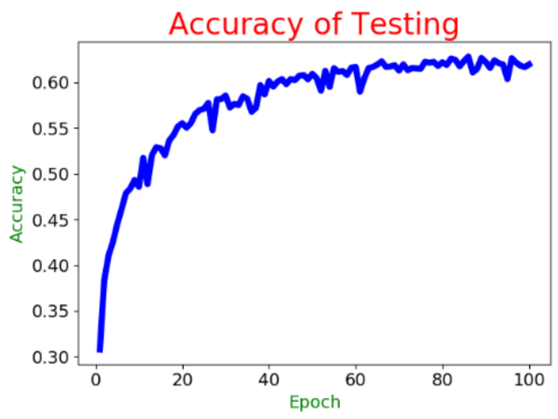


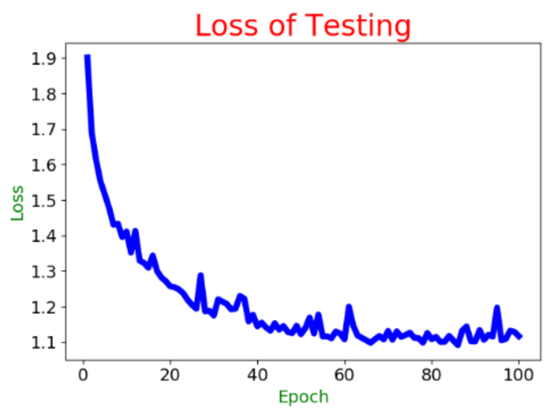


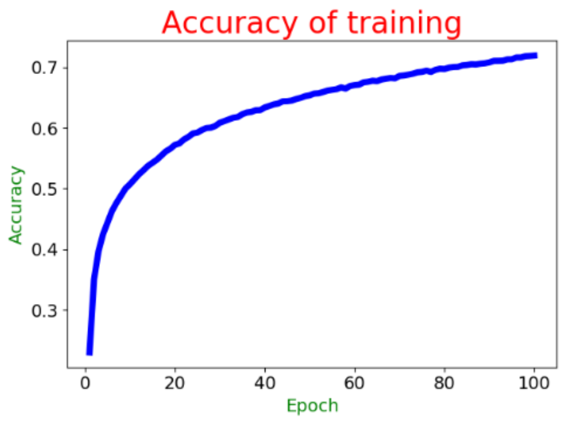


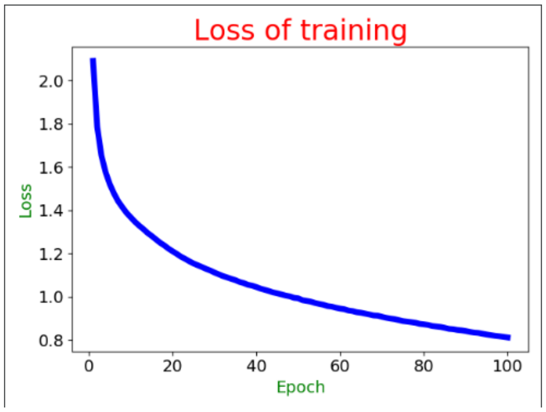


Test 2:

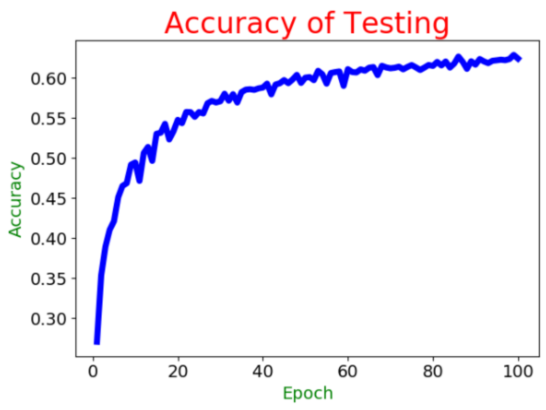


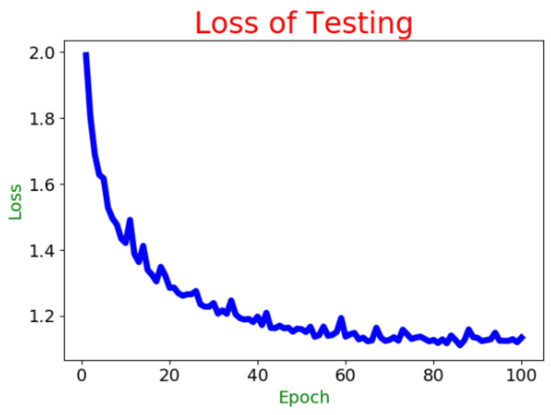


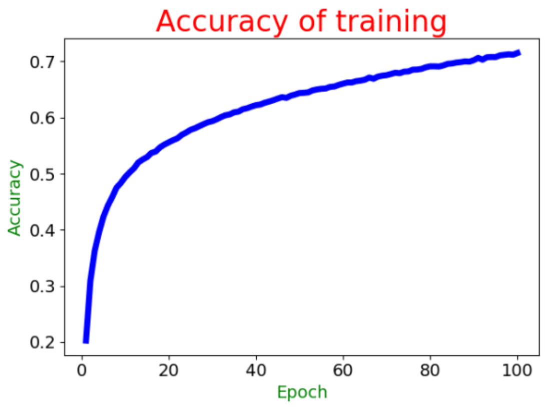

Accuracy of training
Accuracy
Epoch
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80
100

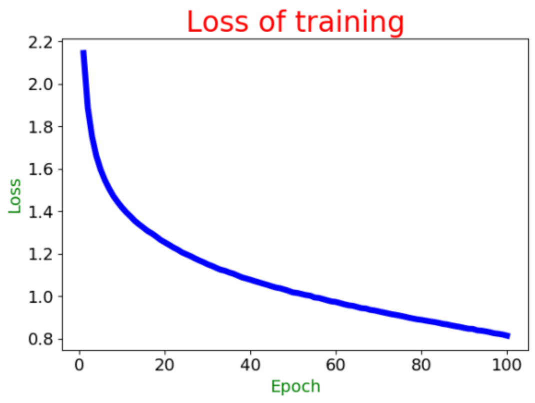

Loss of training
Loss
Epoch
0.8
1.0
1.2
1.4
1.6
1.8
2.0
0
20
40
60
80
100

Test 3

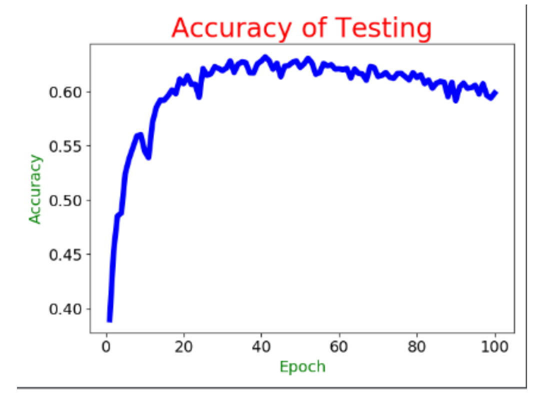

Accuracy of Testing
Accuracy
Epoch
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0
20
40
60
80
100

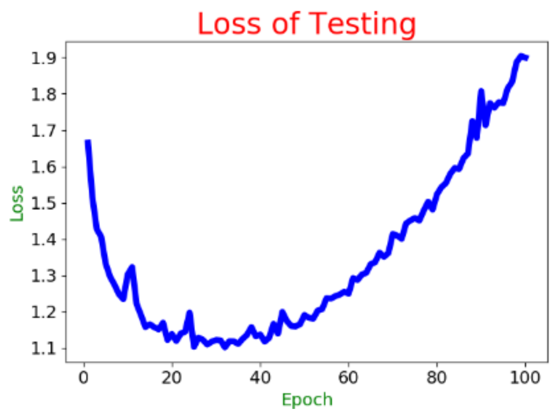

Loss of Testing
Loss
Epoch
1.2
1.4
1.6
1.8
2.0
0
20
40
60
80
100

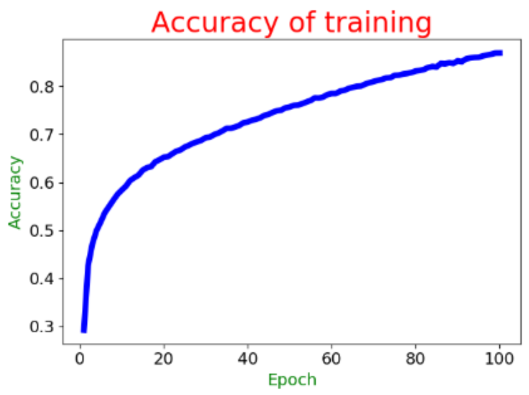

Accuracy of training
Accuracy
Epoch
0.2
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80
100

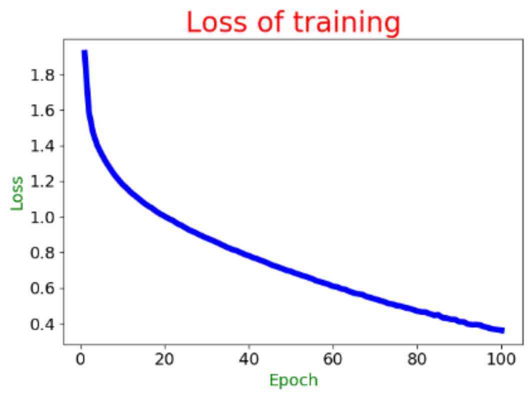

Loss of training
Loss
Epoch
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
0
20
40
60
80
100

Test 4

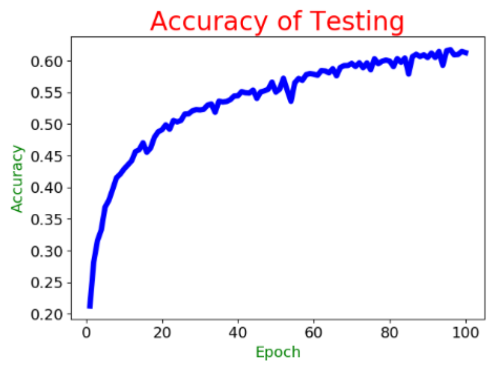

Accuracy of Testing
Accuracy
Epoch

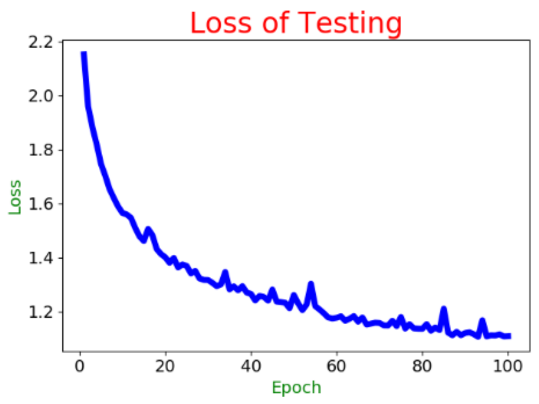

Loss of Testing
Loss
Epoch

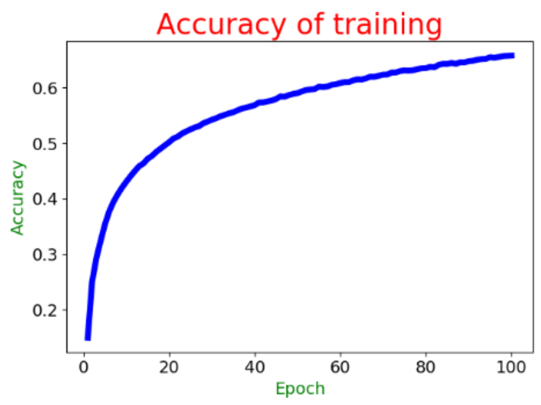

Accuracy of training
Accuracy
Epoch

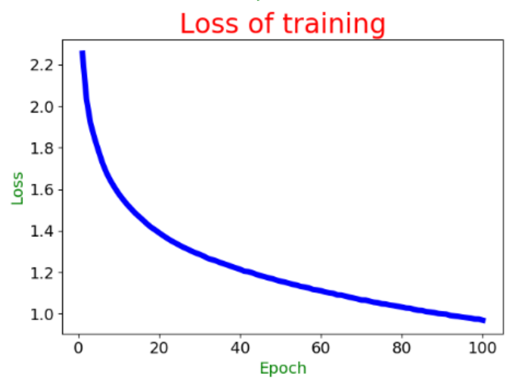

Loss of training
Loss
Epoch

Test 5

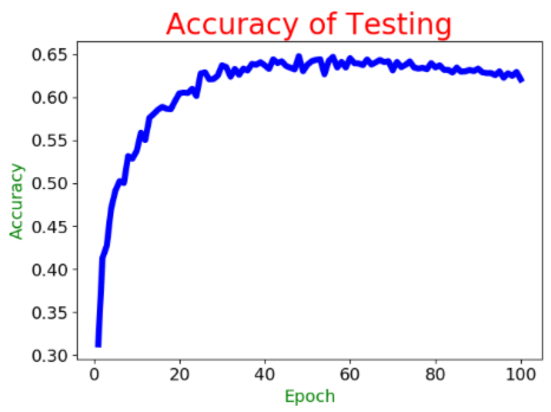

Accuracy of Testing
Accuracy
Epoch

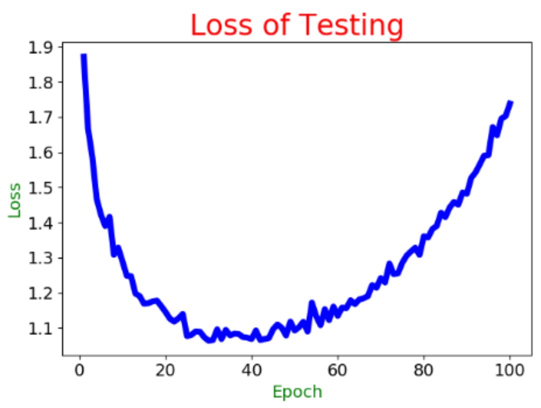

Loss of Testing
Loss
Epoch

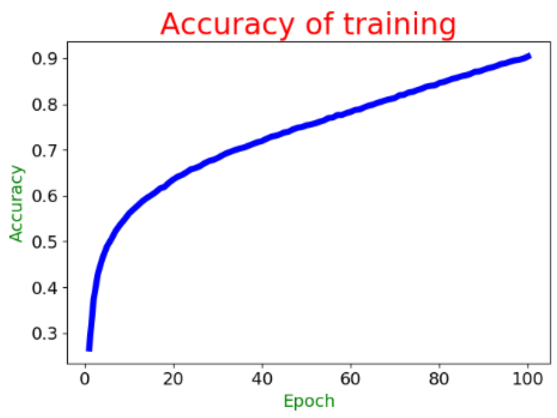

Accuracy of training
Accuracy
Epoch

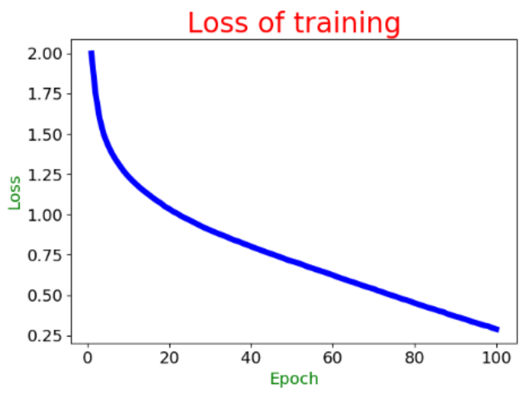

Loss of training
Loss
Epoch

Test 6

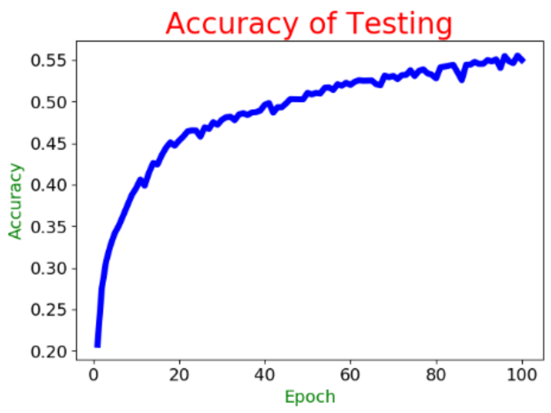

Accuracy of Testing
Accuracy
Epoch

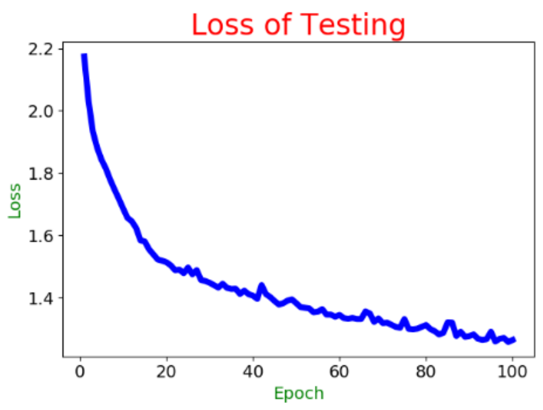

Loss of Testing
Loss
Epoch

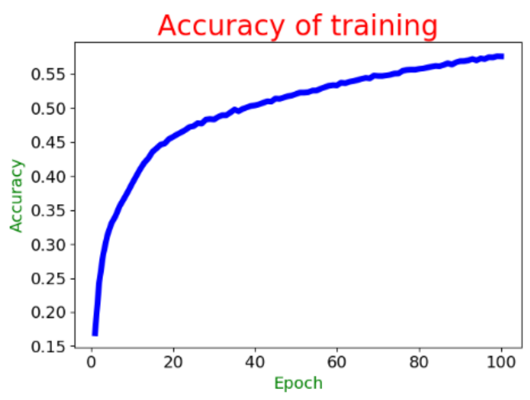

Accuracy of training
Accuracy
Epoch

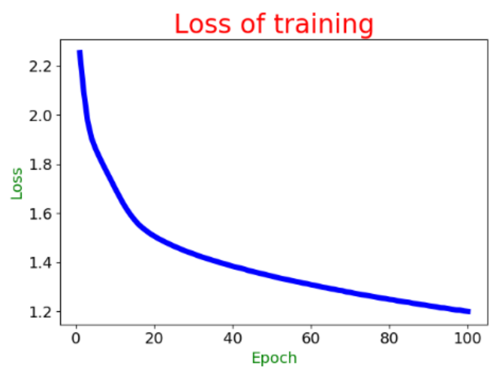

Loss of training
Loss
Epoch

Test 7

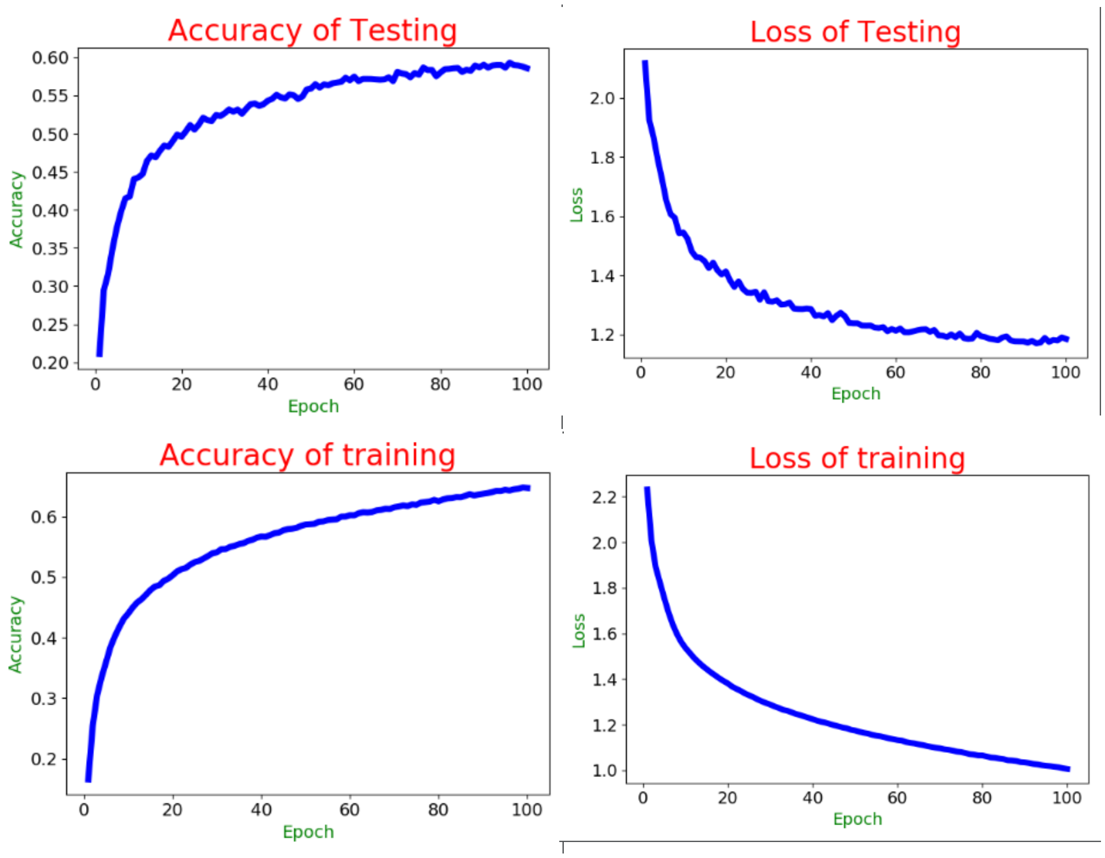

Accuracy of Testing
Accuracy
Epoch


Loss of Testing
Loss
Epoch


Accuracy of training
Accuracy
Epoch


Loss of training
Loss
Epoch

Test 8



2.3. Parameters explanation

In this homework, I have chosen total number of epochs for training is 100, SGD for optimizer, and ReLU for activation function, He normal distribution to initialize the network.

SGD is a method for optimizer, which full name is stochastic gradient descent. As we know in traditional batch gradient descent, when a new parameter needs to be renewed, it will utilize values from the whole network. This will slow down the total learning speed when training samples is too large. The SGD is set to solve this problem. It utilize loss function of each sample to do partial deviation to θ to get corresponding gradience then update θ.

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)})$$

Initializer is important since we cannot get result from symmetric initialized network after training since this net work will degenerate to a one neuron network.

As the talk above, ReLU function could introduce non-linear function to the network which will improve classification ability dramatically.

Parameters:

- Learning rate: learning rate is a rate for gradient descent, if the learning rate is too small, then it is like to trap into some bad local minimum. For example, in test 7, the learning rate is lower to 0.0001, then the output result of testing accuracy is below 60%. On the contrast, if the learning rate is too large, then will happen to over fitting and the curve of testing accuracy will not be smooth. The example is test 4. A proper learning rate is important.

- Patch size: patch size is the total number of training image feed to the training. Larger patch size will product a smoother testing output curve but will slow down training process and will take more memory. If patch size is small, it can produce overfitting. The example is test6 and test8
- Momentum: One shortcoming of SGD is that it updates direction only based on its current batch, so this is not stable. Momentum borrow the concept from physics which stimulate the inertia of object when moving. The previous update value and direction will make influence on present update.

$$\mathrm{v_t} = \gamma \mathrm{v_{t-1}} - \alpha \nabla_\theta J\left(\theta; x^{(i)}, y^{(i)}\right)$$
$$\theta = \theta - v_t$$

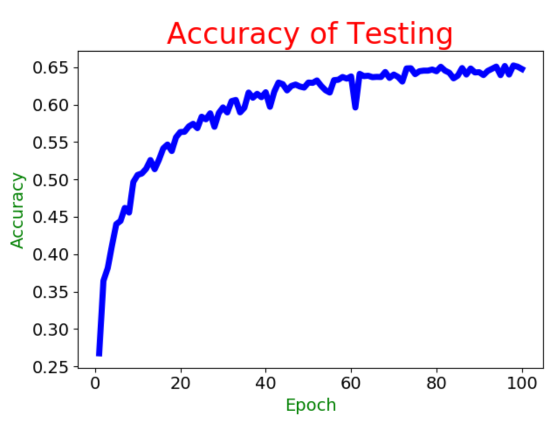
  Larger momentum will speed the training. For example, test 3 is faster than test 5
- Decay: in the loss function, weight decay is a coefficient which is utilized before regularization. Weight decay balance the impact of the model complexity on the loss function. If weight decay is large, the value of the complex model loss function is also large. This is a tool to prevent overfitting.
- Padding: there are two padding methods used in this homework. Valid padding and same padding. Padding will not introduce features, but same padding will bring more parameters to be trained into network. Comparing to test 1 and test 6, test 6 run slower than 1.

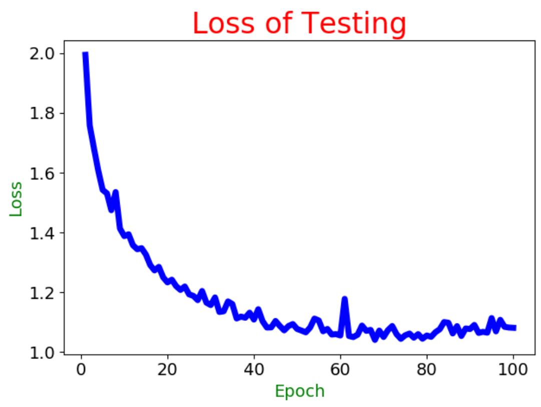
2.4. Best result;

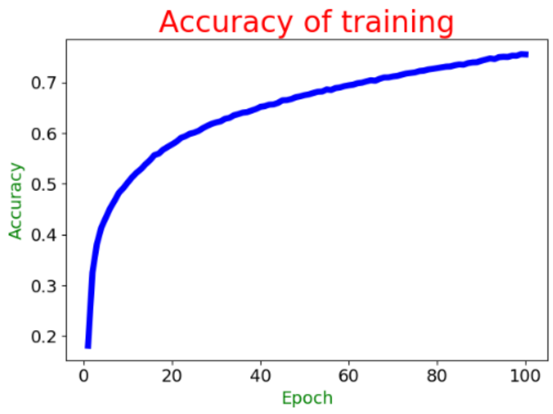
According to 2.2, the best test is test1.

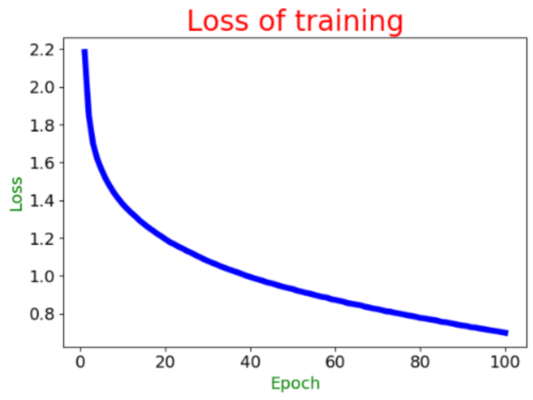
Number of test	**Testing accuracy 100 epochs**	Training accuracy 100 epochs	Learning rate	Batch size	momentum	decay	padding
1	**0.6477**	0.7551	0.00055	64	0.9	0.000001	valid

Test 1


Accuracy of Testing
Accuracy
Epoch
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0
20
40
60
80
100


Loss of Testing
Loss
Epoch
1.0
1.2
1.4
1.6
1.8
2.0
0
20
40
60
80
100


Accuracy of training
Accuracy
Epoch
0.2
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80
100


Loss of training
Loss
Epoch
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
0
20
40
60
80
100

3. State-of-the-Art CIFAR-10 Classification

3.1. Paper chosen

The paper I have chosen is Deep Convolutional Neural Networks as Generic Feature Extractors. The accuracy rate for this paper over CIFAR-10 is 89.67%

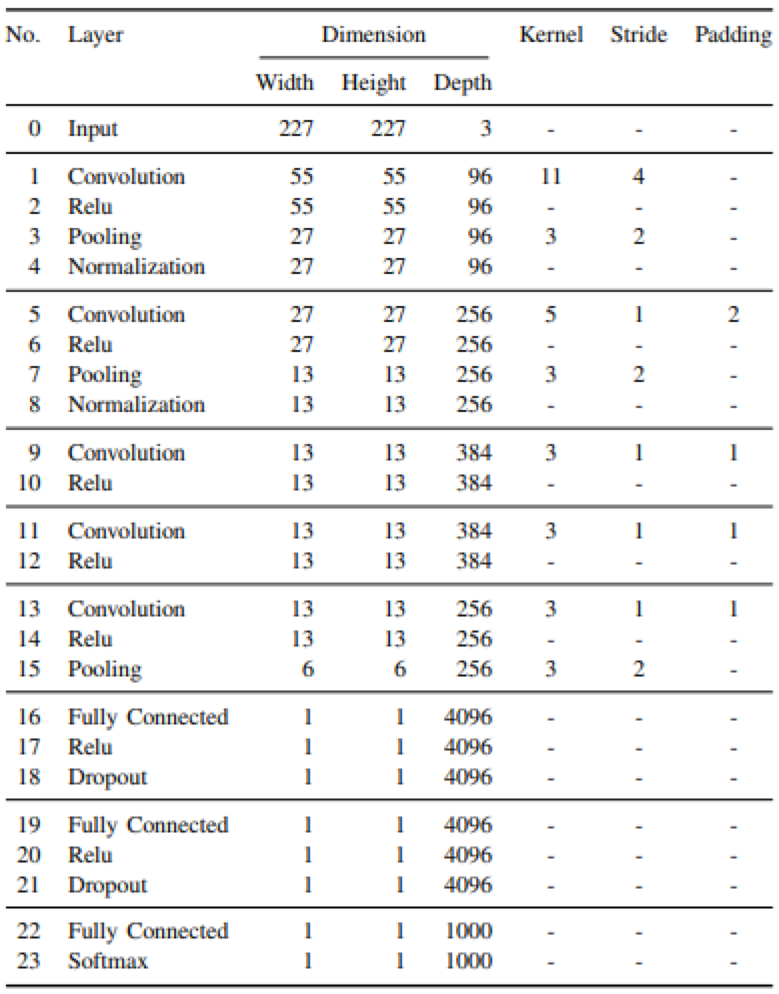
3.2. How the author achieves the result

According to the authors, deep networks behave good on different benchmark, but it needs a long time to train this network. So here, the author provides a way to train the network, it keeps learned feature extract part and train the classification layer.

They maintained the feature extraction part, learned convolutional kernels. And they only train the classification layers.

The architecture for this network is shown below:

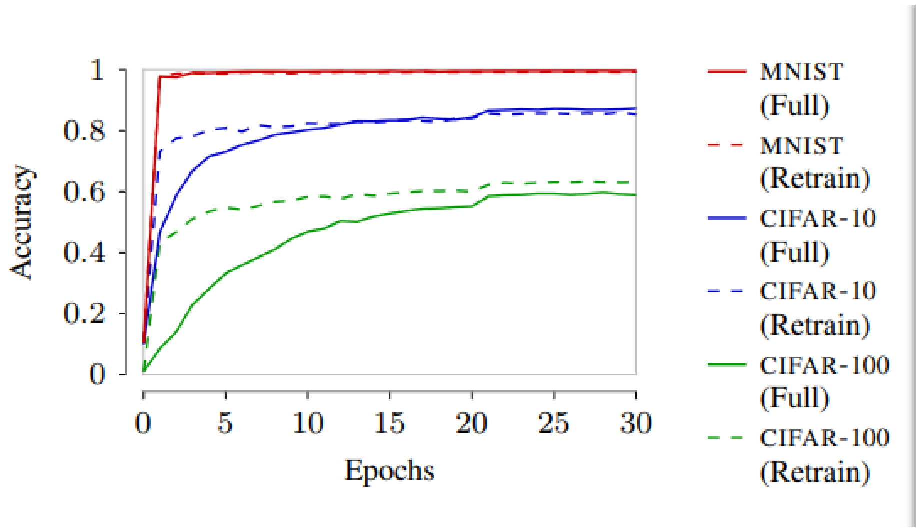
TABLE II. ARCHITECTURE OF OUR IMPLEMENTED CONVOLUTIONAL NETWORK.



No.	Layer	Dimension			Kernel	Stride	Padding
		Width	Height	Depth			
0	Input	227	227	3	-	-	-
1	Convolution	55	55	96	11	4	-
2	Relu	55	55	96	-	-	-
3	Pooling	27	27	96	3	2	-
4	Normalization	27	27	96	-	-	-
5	Convolution	27	27	256	5	1	2
6	Relu	27	27	256	-	-	-
7	Pooling	13	13	256	3	2	-
8	Normalization	13	13	256	-	-	-
9	Convolution	13	13	384	3	1	1
10	Relu	13	13	384	-	-	-
11	Convolution	13	13	384	3	1	1
12	Relu	13	13	384	-	-	-
13	Convolution	13	13	256	3	1	1
14	Relu	13	13	256	-	-	-
15	Pooling	6	6	256	3	2	-
16	Fully Connected	1	1	4096	-	-	-
17	Relu	1	1	4096	-	-	-
18	Dropout	1	1	4096	-	-	-
19	Fully Connected	1	1	4096	-	-	-
20	Relu	1	1	4096	-	-	-
21	Dropout	1	1	4096	-	-	-
22	Fully Connected	1	1	1000	-	-	-
23	Softmax	1	1	1000	-	-	-

They started with a learning rate η = 0.001 and decreased it to η = 0.0001 after the 20-th epochs. In addition, they selected a momentum value μ = 0.9, weight decay λ = 0.0005, and a batch size of β = 80. These parameters were determined based on a validation set.

3.3. Comparation



For my network, there will be 62,006 parameters to be trained, but for this network, it only has (4096+4096+1000)*2= 18384 parameters to be train if the feature extraction part has already been train. This network is more efficient and more accurate.