VAN-HUNG, LE (黎文雄)

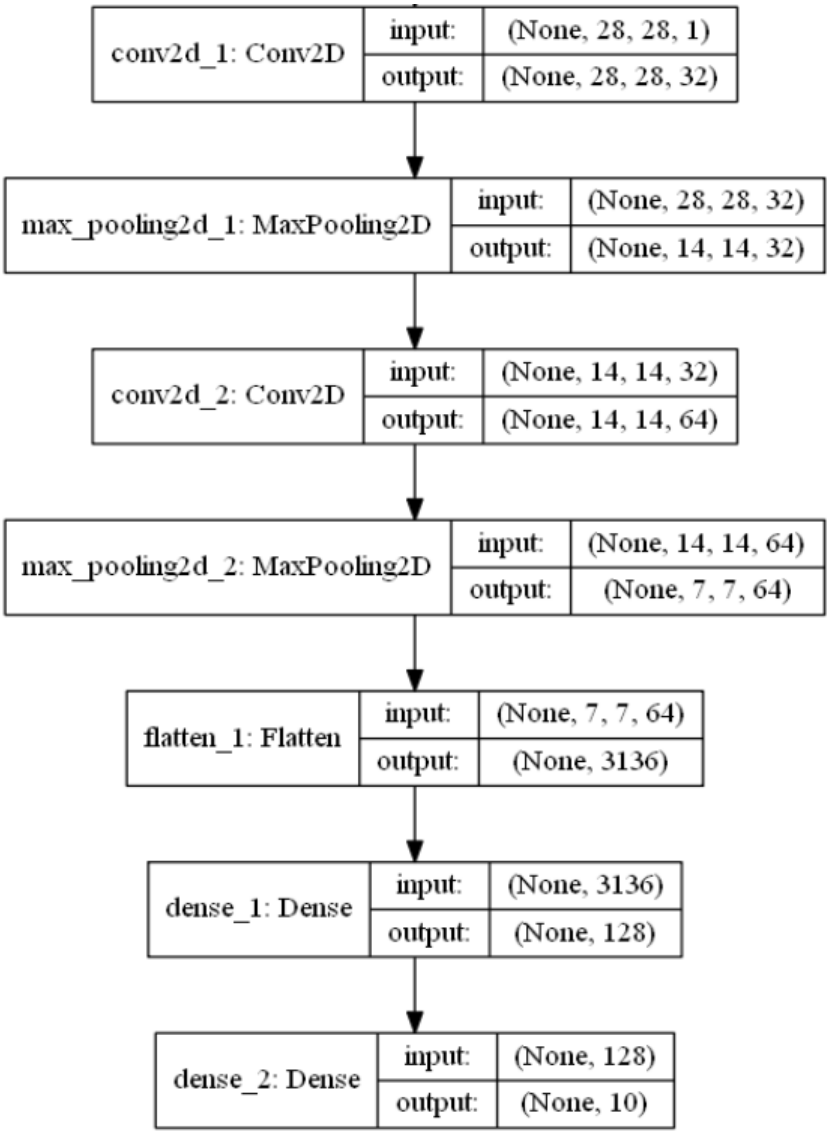
Student ID: 0860831

DEEP LEARNING – HOMEWORK 2

1. Using Convolutional Neural Network for Image Recognition

In this exercise, I use Keras library to create a convolutional neural network. Besides, I also implement the forward computation by yourself and compare it with Keras.

Network structure:



Conv2D: convolution layer with kernel_size = (3,3), strides = (1,1), activation ='relu' and padding = 'same' (in this model padding = 1) for same convolution.

Maxpooling2D: maxpooling layers, I setup that layer with pool_size = (2,2)

Flatten: flatten the previous maxpooling layer to flatten vector

Dense: fully connected layer with activation = 'relu' in hidden layer and activation = 'softmax' in output layer.

1.1 Implement feed forward

I implement step by step of feed forward manually

a. Convolution layer

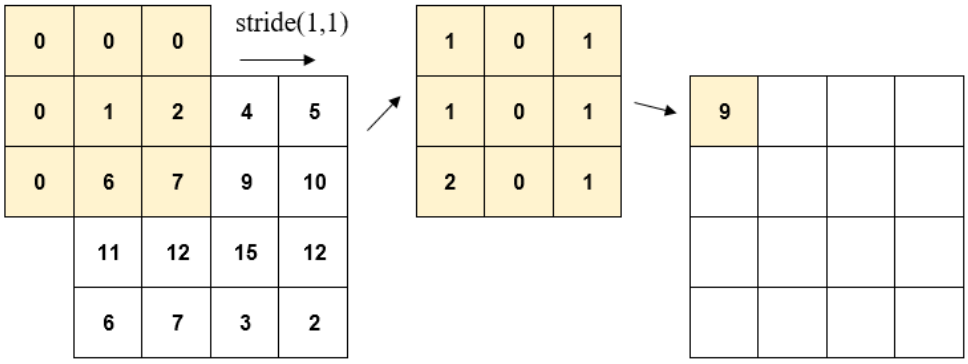


Fig1: convolution operator

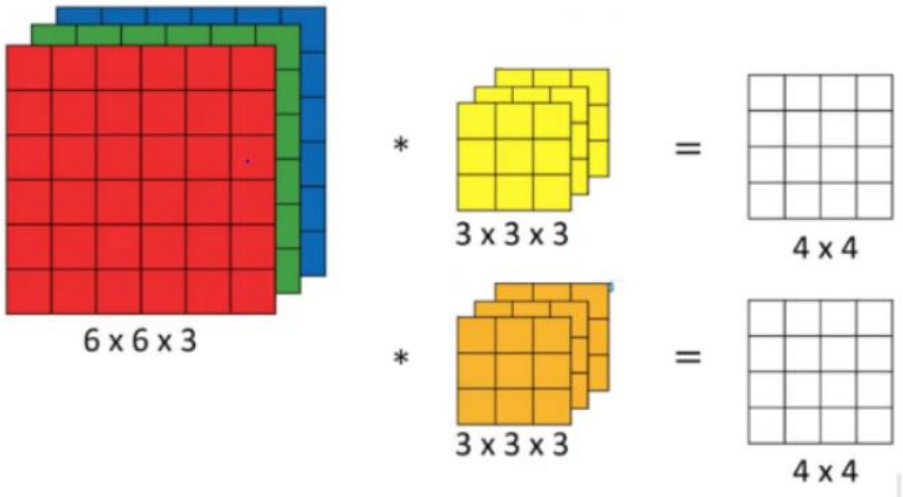


Fig2: how does a convolution layer work

First, I add zero paddings (p=1) for I can get the output the same scales with input.

After that, I using a convolution operator to do with all of the slices of input and take the sum of them to calculate the value of the next layer (Fig2)

b. Maxpooling

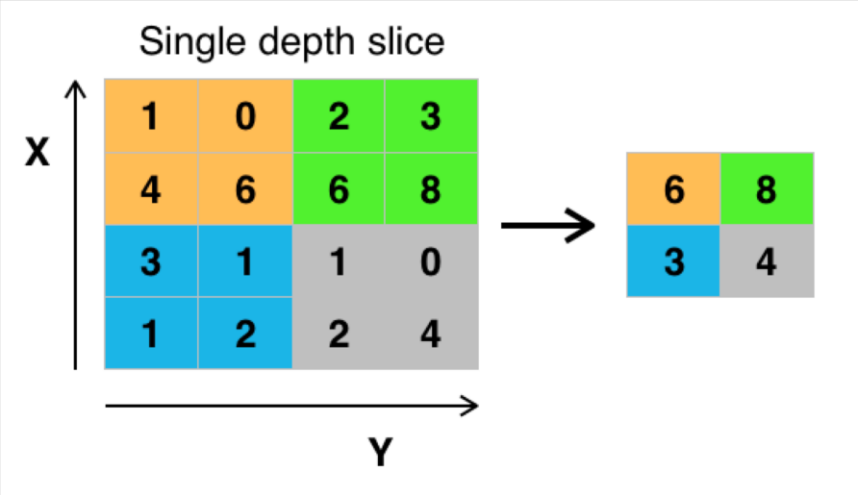


Fig3: Max pooling

For each feature map of current lasyer, we select the maximum value inside the (2x2) slide windows with the stride is (1,1) for every pixel.

c. Flatten

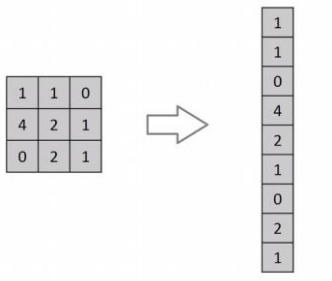


Fig4: flatten

Flatten the last maxpooling layer by using numpy.reshape() function, I will get one column vector to use in fully connected layer.

d. Fully connected

e. Check feed forward

Double check with the feed forward from keras, there is no mistake (100% match). I use plot_model_MNIST.check_forward() function to check it. You can see the check_flag bool variable (line 263 of file plot_model_MNIST.py) to see the result.

1.2 Accuracy of Training, Validation and Testing

- Accuracy of training: 99.98
- Accuracy of validation: 99.28
- Accuracy of testing: 99.25

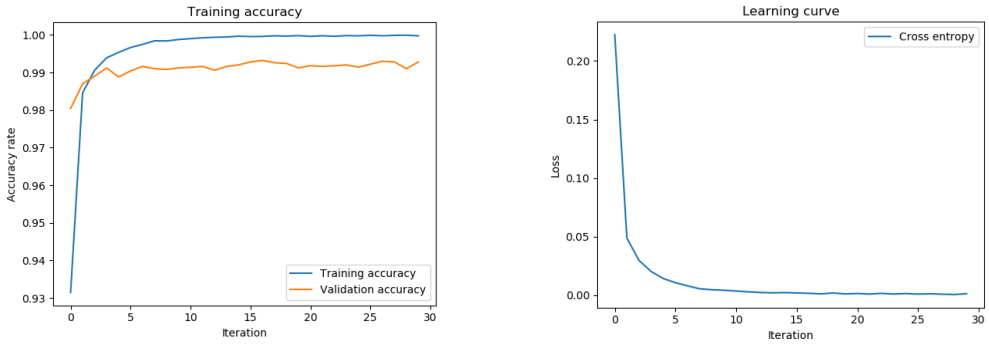


Fig5: accuracy of model with No l2 regularization

1.3 Histogram layers

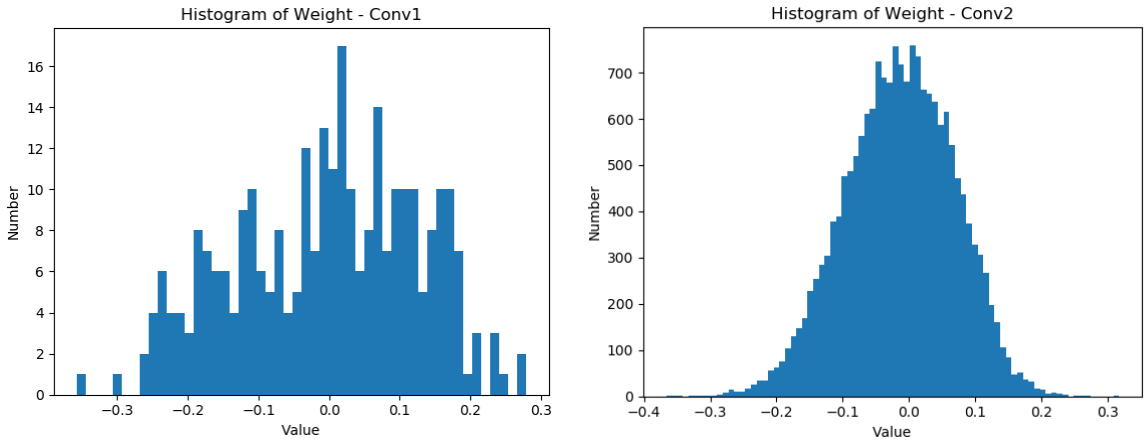


Fig6: Histogram of Weight of Convolution layer

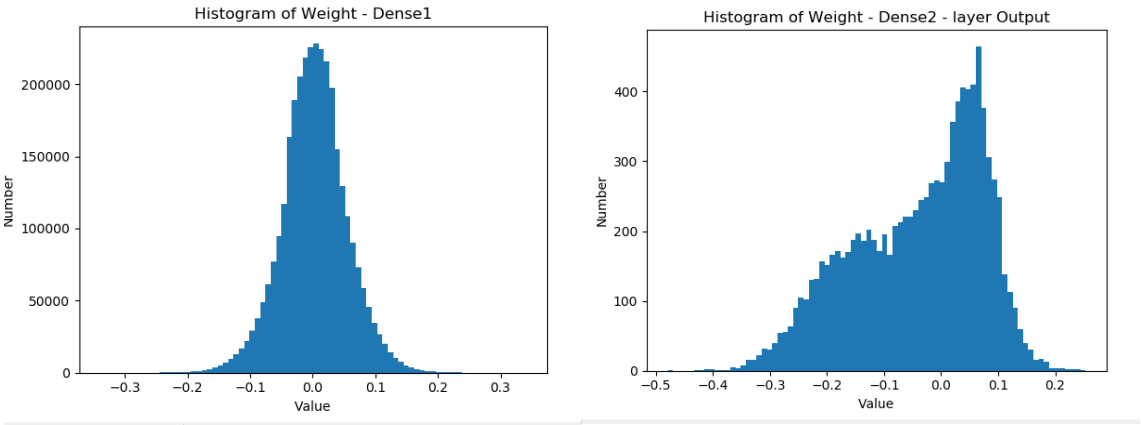
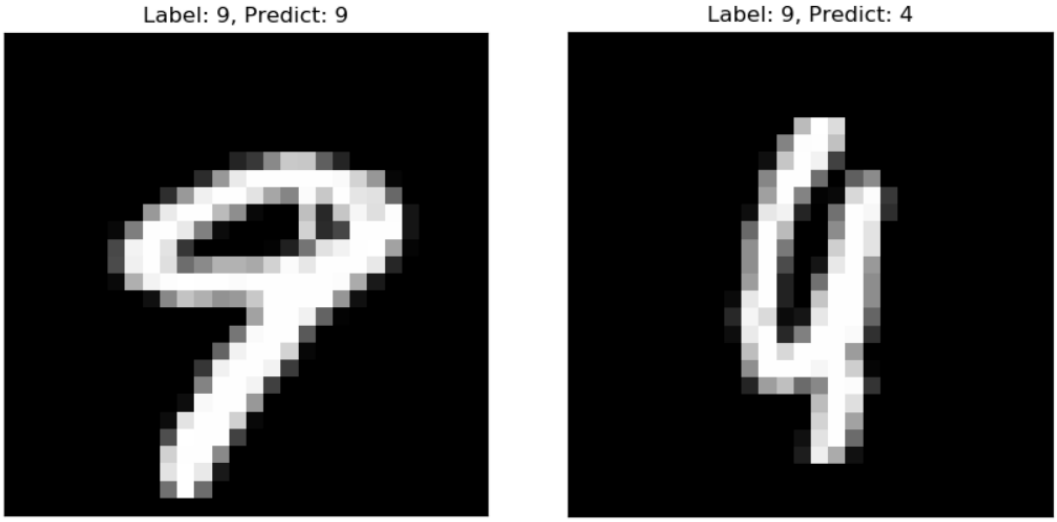


Fig7:Histogram of Weight of Fully Connected layer

1.4 Correctly and miss-classification

There are **75** data are fail to classify, you can refer to folder incorrect_classification for more review.

I also show some examples of miss-classification pictures and correctly classified picture below:



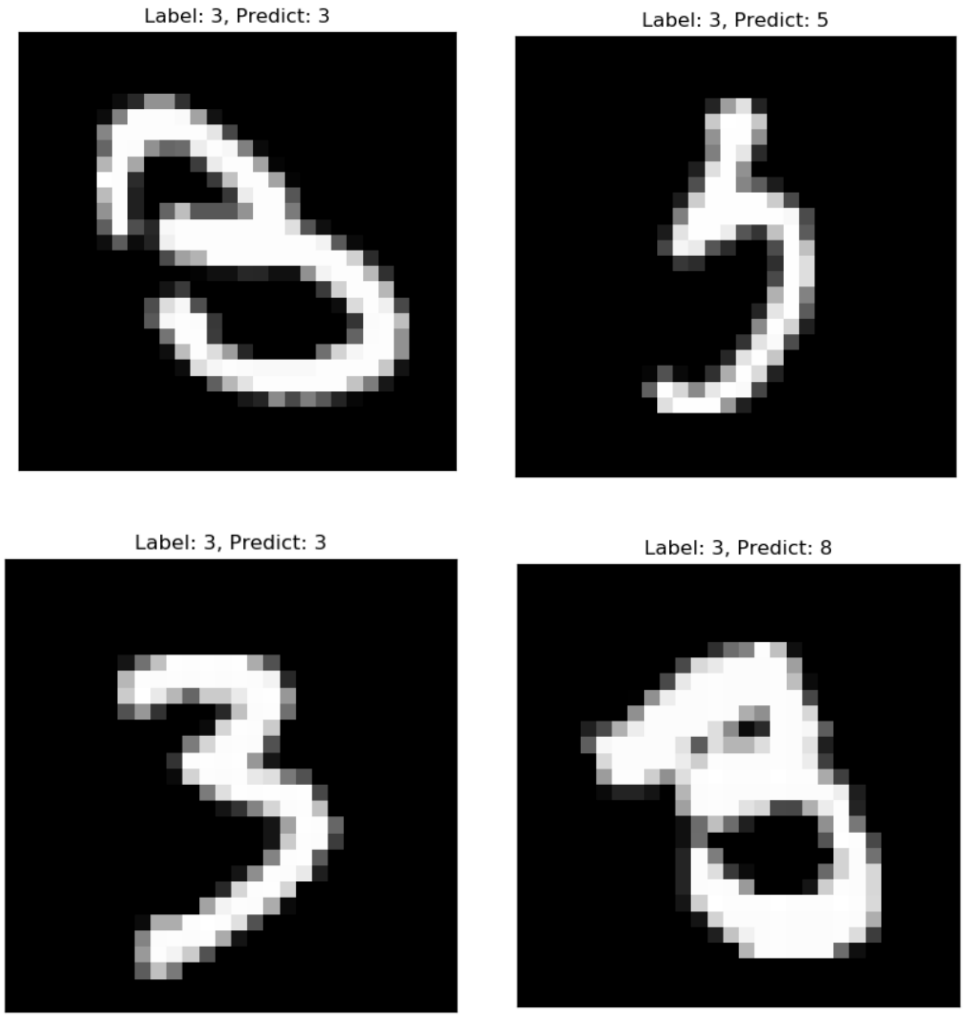
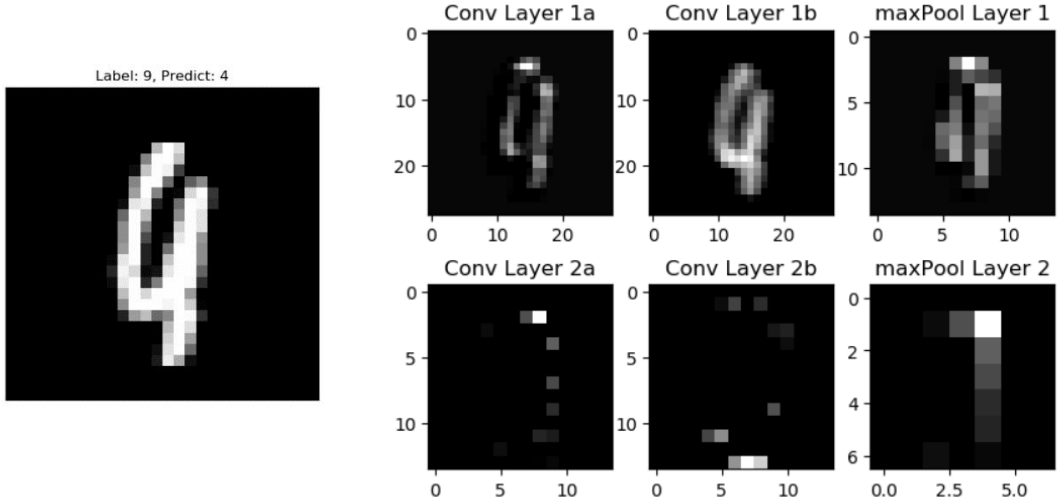


Fig8: Examples of correctly classified and miss-classified images

1.5 Visualize feature map



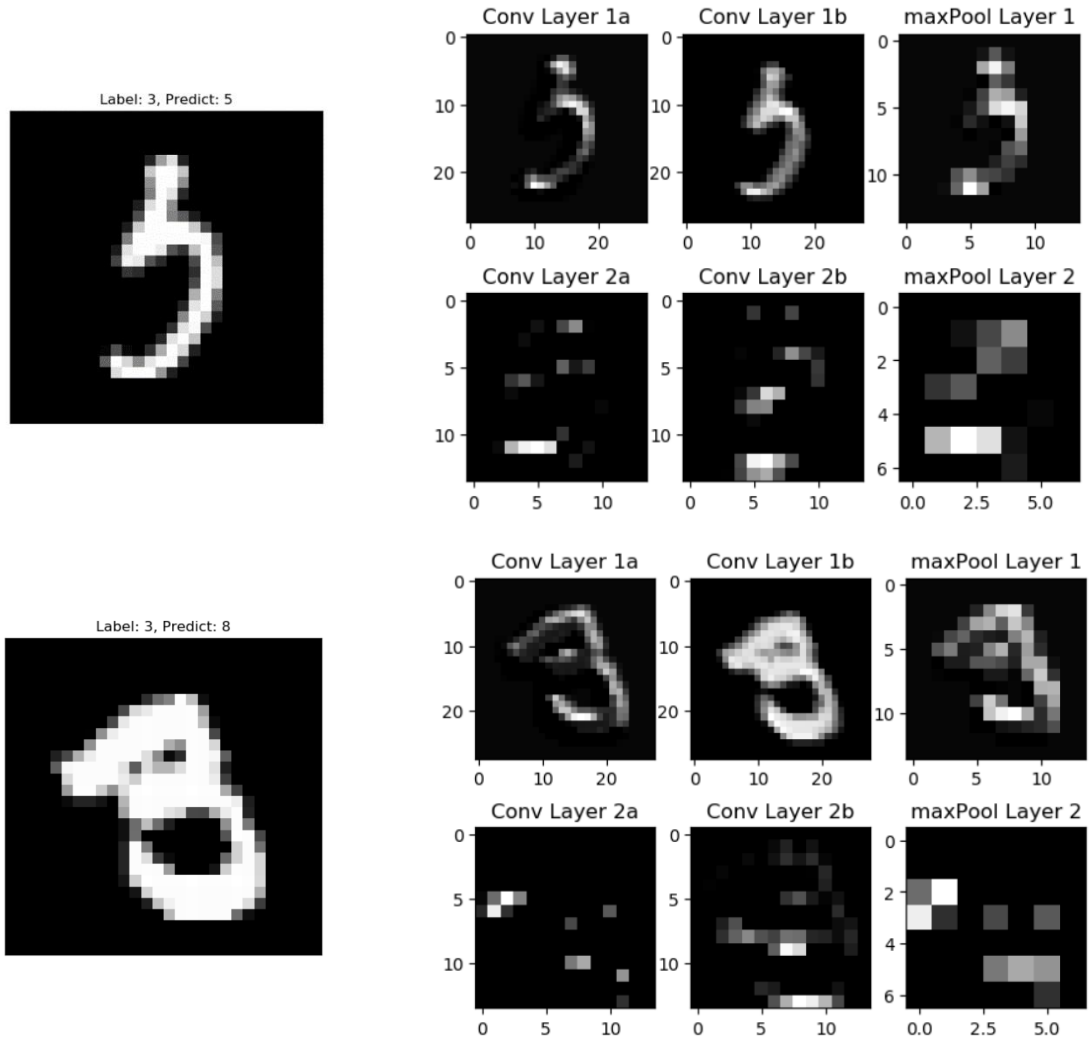


Fig9: Visualize feature map at different layer of network

1.6 Adding regularization l2

Objective function:

$$E = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \ln t_{nk} + \alpha||w||_2^2$$

Alpha is regularization parameter, in this task, I set **alpha = 0.001**

- Accuracy of training: 99.46
- Accuracy of validation: 98.9
- Accuracy of testing: 0.9913

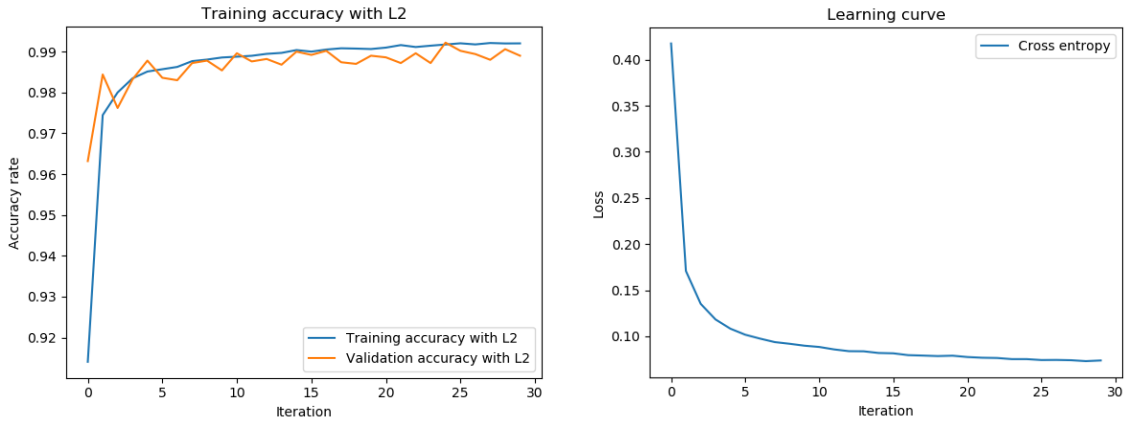


Fig10: accuracy of model with l2 regularization

L2 regularization base on limiting capacity of the model by adding a parameter norm penalty to the objective function E. It forces the weights to decay towards zero but not exactly zero, show on the histogram below.

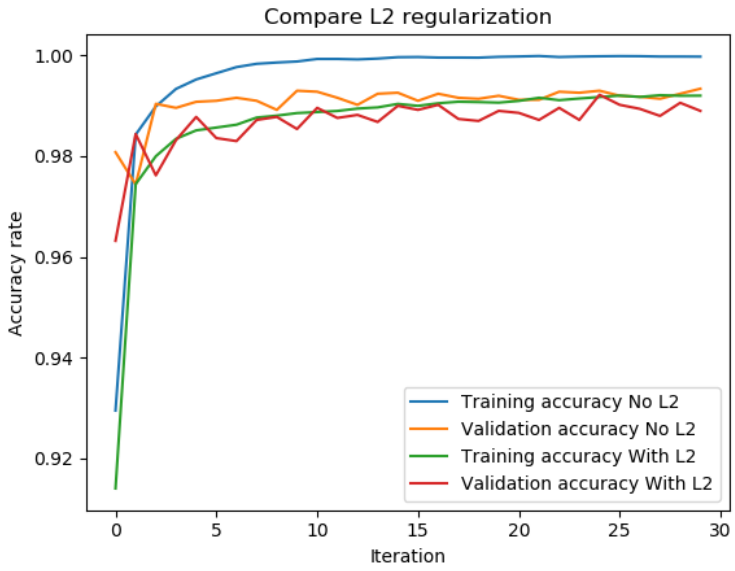


Fig11: compare accuracy of model with L2 regularization

- Histogram layers after adding L2 regularization

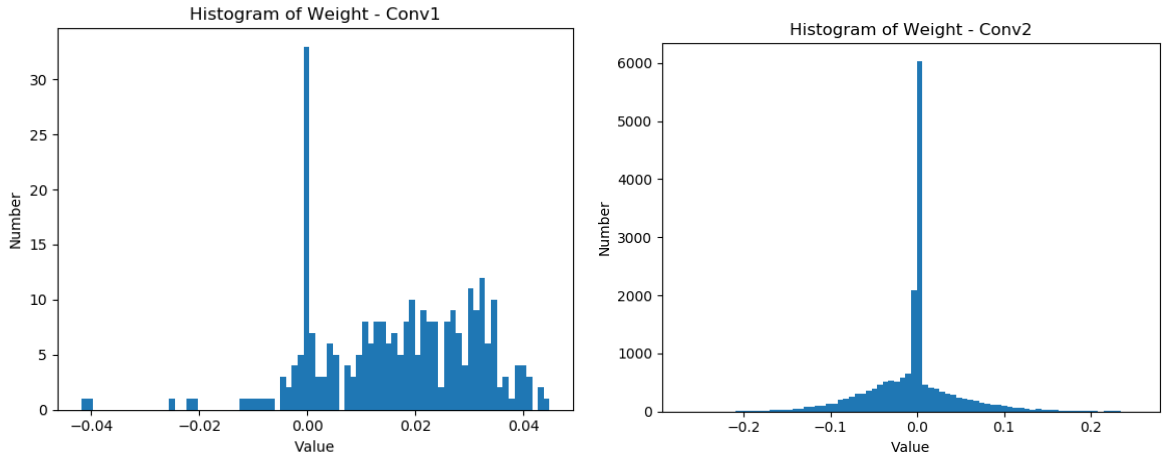


Fig12: Histogram of Weight of Convolution layer with L2 regularization

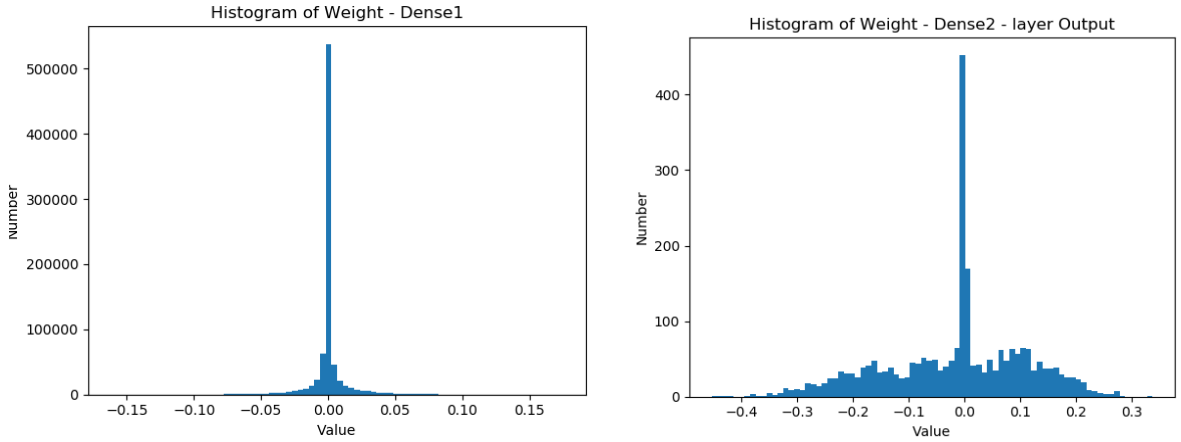
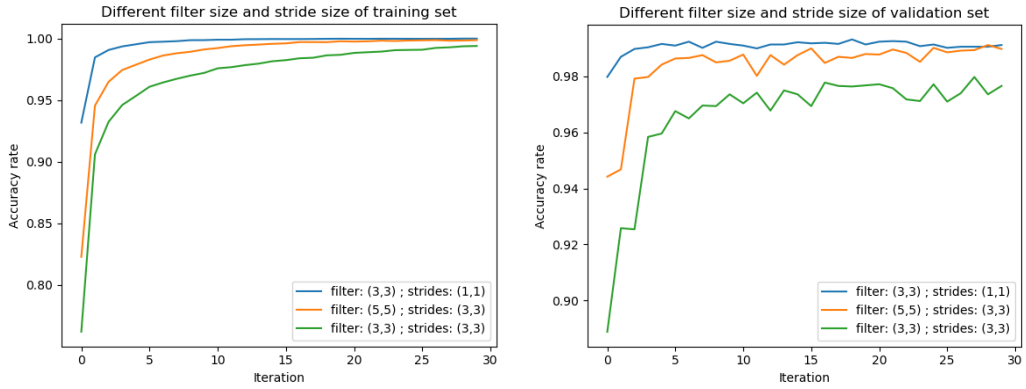


Fig13: Histogram of Weight of Fully connected layer with L2 regularization

1.7 Analyze the effect of stride size, filler size, network structure

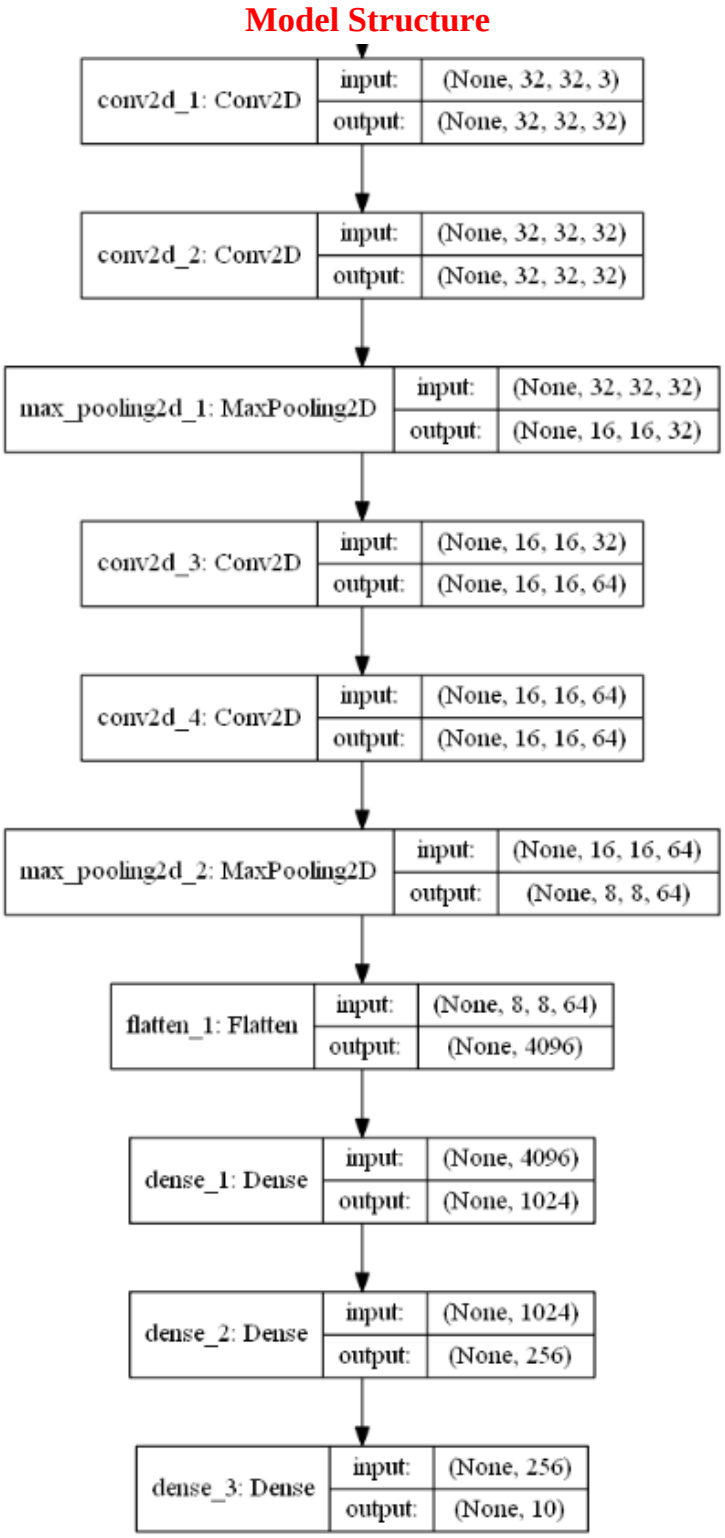


- I changed the filter size to see the different of the accuracy.
- Big filter size gives the good result of accuracy with the same stride.
- Small strides size gives the good result of accuracy with the same filter.
- When we run more time of epoch, Small strides size give the bad result of validation accuracy.

Python file for run:

myself_forward.py	Implement feed forward manually
traning_model_MNIST.py	Training model using Keras library
plot_model_MNIST.py	Check myself_forward and plot figures
incorect_classification	Data miss classification (folder)
analyze_size_strides_filter.py	Analyze size of filter and strides

2. Preprocessing before Using Convolutional Neuron Network for Image Recognition



2.1 Preprocessing CIFAR dataset

The CIFAR-10 dataset consists of 5 batches for training, named data_batch_1 to data_batch_5 and 1 batch for testing named test_batch. The original one batch data is (10000, 3072) **maxtrix** expressed in numpy array. Which mean 10,000 images and 3072=32x32x3 value. This row represents a color image of 32x32 pixel with 3 channels RGB. We need to reshape the matrix (10000x3072) to the tensor with shape (10000,32,32,3)

I will step by step to preprocessing data:

- Using pickle to open each file, using [‘lable’] and [‘data’] to get raw data.
- Using reshape function: each_batch.reshape(10000, 3, 32, 32))
- Using transpose function each_batch.transpose(0, 2, 3, 1)
- Normalize data using Min-Max Normalization y = (x-min)/(max-min)
- Concatenate 5 batches together for training data and use 10% of training data as a validation set.
- Finally, one-hot encode label. CIFAR-10 provides 10 different classes of the image, so I need a vector in size of 10 as well.

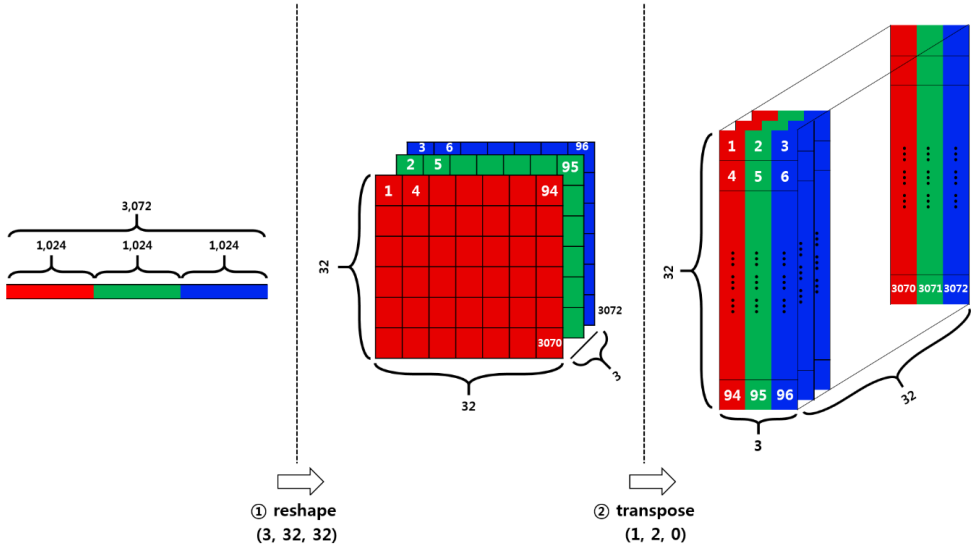


Fig14: reshape and transpose row data

index	label
0	airplane (0)
1	automobile (1)
2	bird (2)
3	cat (3)
4	deer (4)
5	dog (5)
6	frog (6)
7	horse (7)
8	ship (8)
9	truck (9)
...	...
...	...

original label data

label	index											
	0	1	2	3	4	5	6	7	8	9	...	...
airplane	**1**	0	0	0	0	0	0	0	0	0	...	...
automobile	0	**1**	0	0	0	0	0	0	0	0	...	...
bird	0	0	**1**	0	0	0	0	0	0	0	...	...
cat	0	0	0	**1**	0	0	0	0	0	0	...	...
deer	0	0	0	0	**1**	0	0	0	0	0	...	...
dog	0	0	0	0	0	**1**	0	0	0	0	...	...
frog	0	0	0	0	0	0	**1**	0	0	0	...	...
horse	0	0	0	0	0	0	0	**1**	0	0	...	...
ship	0	0	0	0	0	0	0	0	**1**	0	...	...
truck	0	0	0	0	0	0	0	0	0	**1**	...	...

one-hot-encoded label data

Fig15: one-hot encode process

2.2 Accuracy of Training, Validation and Testing

- Accuracy of training: 99.25
- Accuracy of validation: 70.80
- Accuracy of testing: 71.14

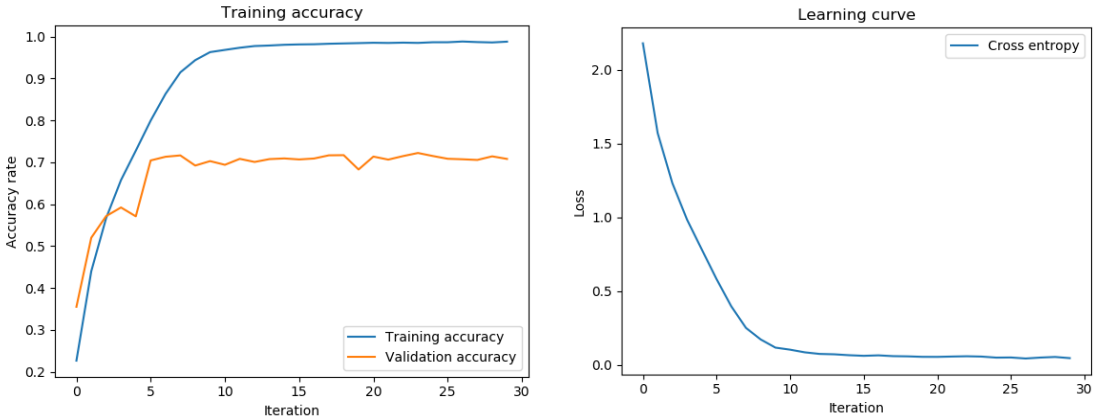


Fig16: accuracy of model with No l2 regularization

We can see that from the epoch 12^{th} the model is about over-fitting to the data. Using L2 regularization will solve this problem.

2.3 Histogram layers

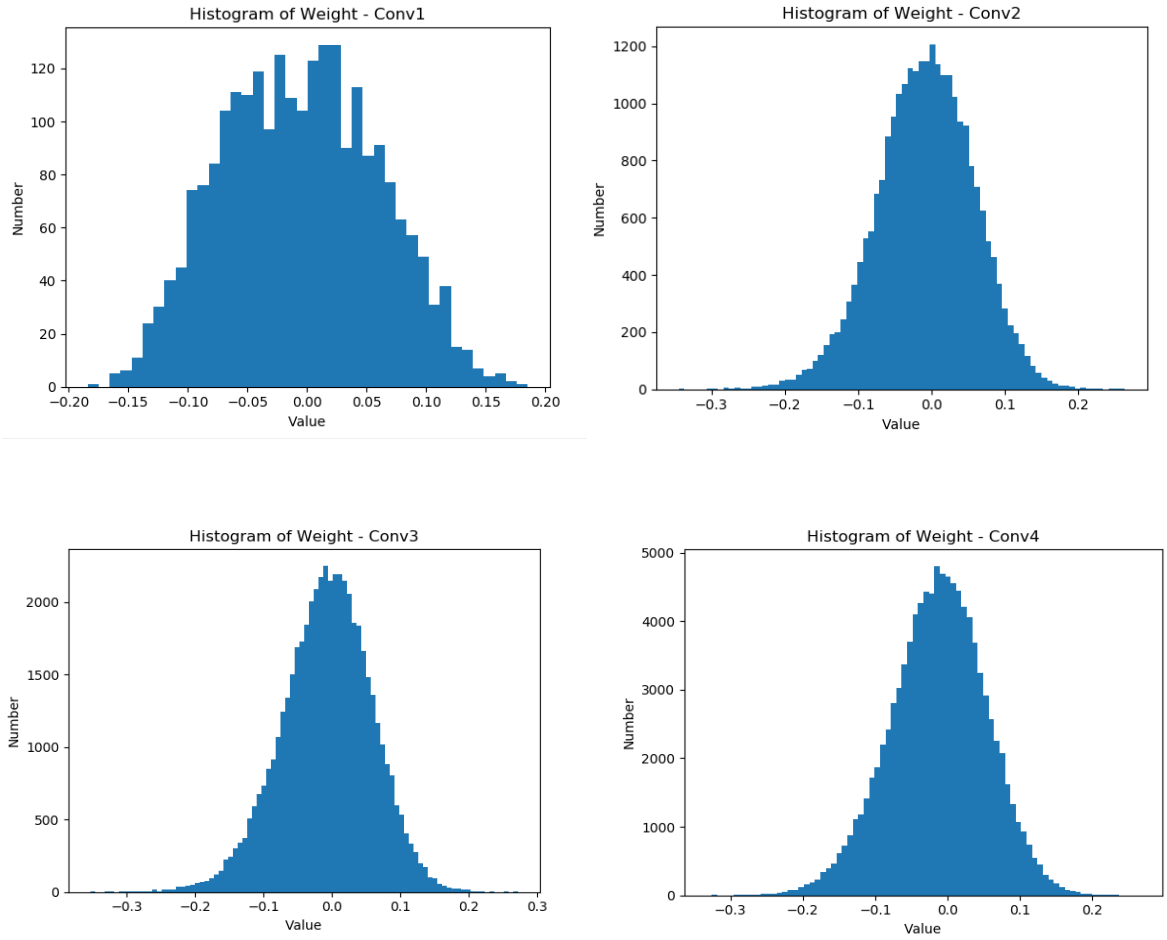


Fig17: Histogram of Weight of Convolution layer

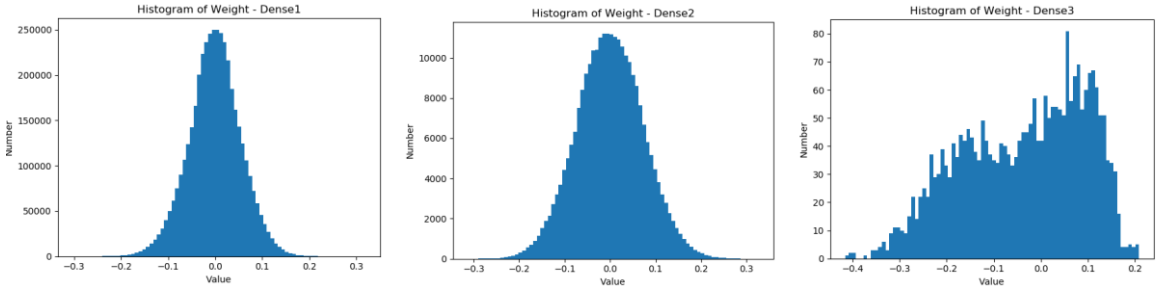


Fig18: Histogram of Weight of Fully connected layer

2.4 Correctly and miss-classification

There are **2856** data are fail to classify, you can refer to folder incorrect_classification_CIFAR for more review.

I also show some examples of miss-classification pictures and correctly classified picture below:

Fig19: Examples of miss-classified images

Fig20: Examples of correctly classified

The accuracy is just 71.14%, since CIFAR is much more difficult than MNIST, the examples above show some “understandable” miss-classification by my network. Say: horse to dog, airplane to bird, dog to bird, automobile to truck, bird to airplane.

2.5 Visualize feature map

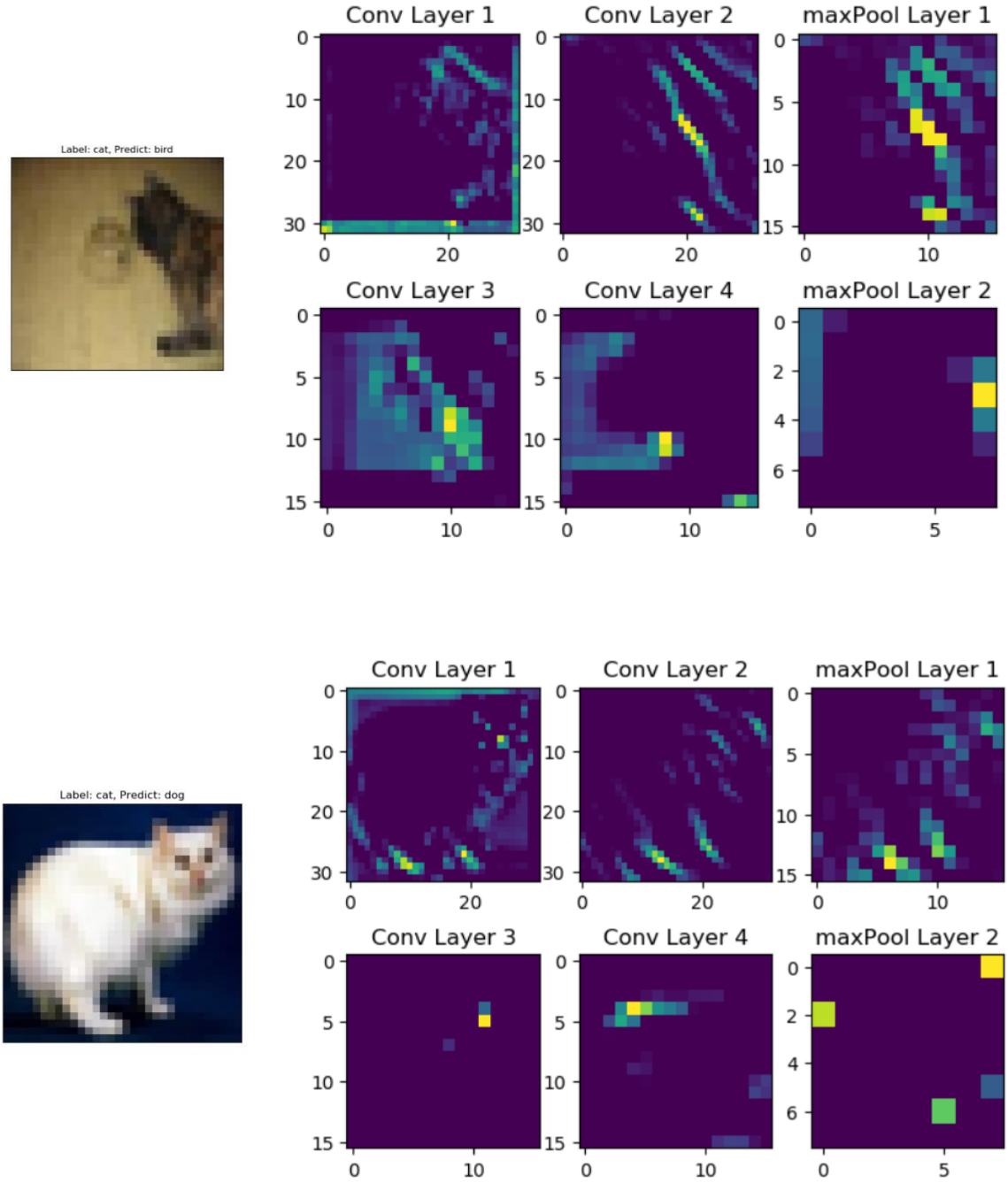


Fig21: Visualize feature map at different layer of network

2.6 Adding regularization l2

Objective function:

$$E = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \ln t_{nk} + \alpha||w||_2^2$$

Alpha is regularization parameter, in this task, I set **alpha = 0.001**

- Accuracy of testing: 73.89

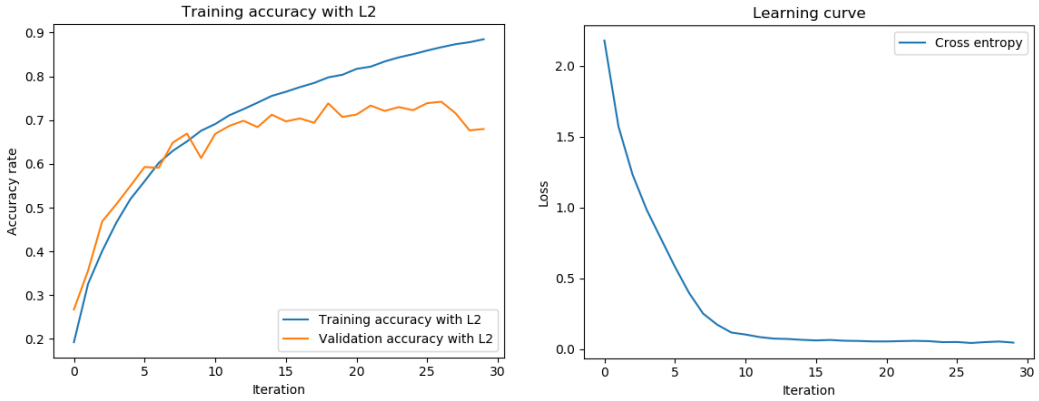


Fig22: accuracy of model with l2 regularization

L2 regularization base on limiting capacity of the model by adding a parameter norm penalty to the objective function E. It forces the weights to decay towards zero but not exactly zero, show on the histogram below.

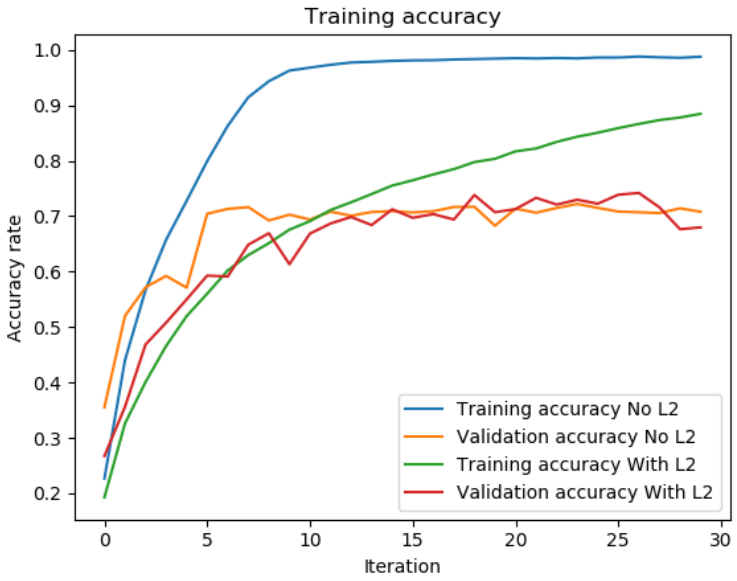


Fig23: compare accuracy of model with L2 regularization

- Histogram layers after adding L2 regularization

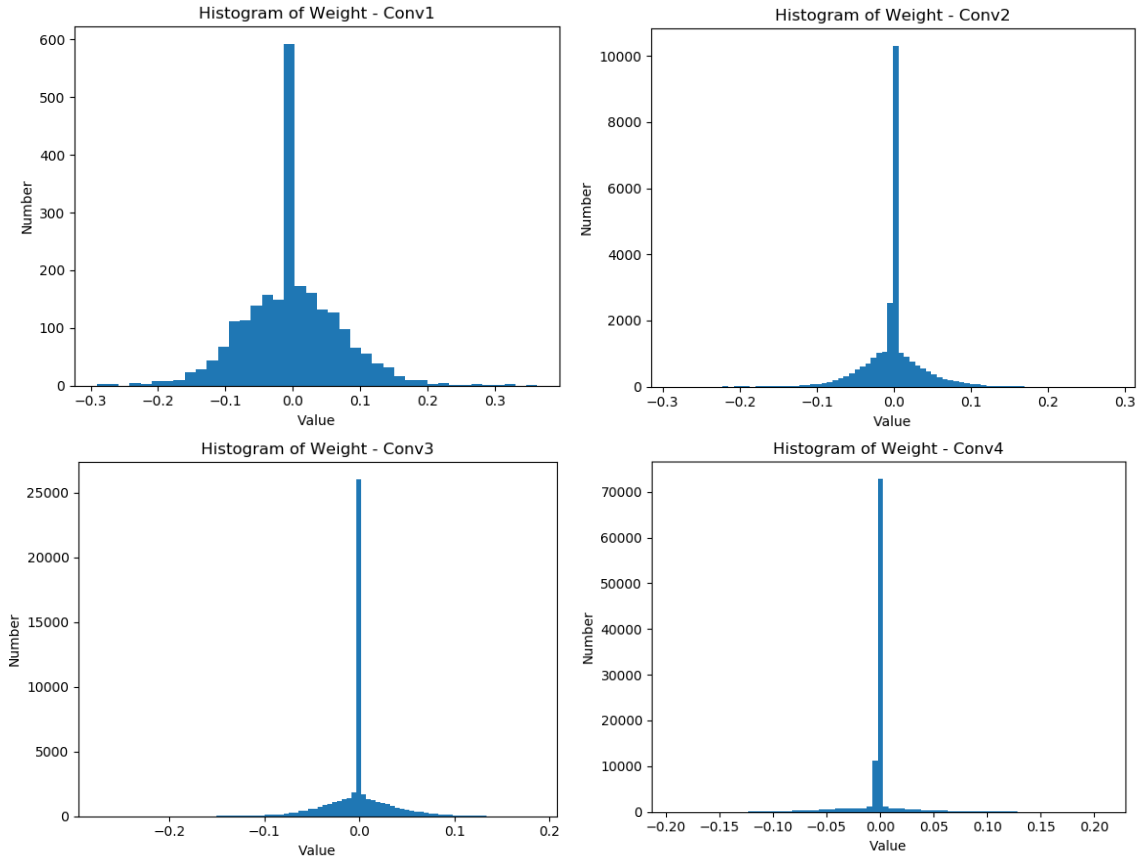


Fig24: Histogram of Weight of Convolution layers with L2 gegularization

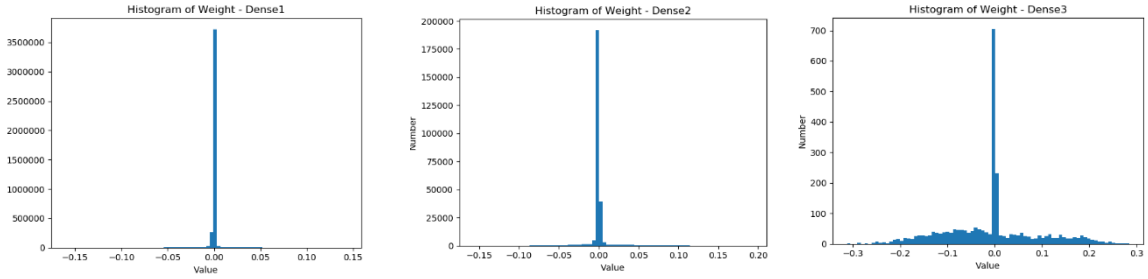


Fig25: Histogram of Weight of Dense layers with L2 gegularization

File python for run:

traning_model_CIFAR.py	Training model and save weight
plot_model_CIFAR.py	Plot figures
my_preprocessing_data.py	Library for preprocessing data
cifar-10-python	Data for training and testing (folder)
incorrect_classification_CIFAR	Data miss classification (folder)