Submission of the assignment 3 of big data course

Xiaoke(Jimmy) Shen

April 14, 2017

1 Summary of the submission of this assignment

In this submission, single layer, multiple layer neural network and the CNN (Convolutional Neural Networks) are used to do the classify the CIFAR10 dataset provided in the big data course.
Results are shown in figure 1. Details are given in the following sections.

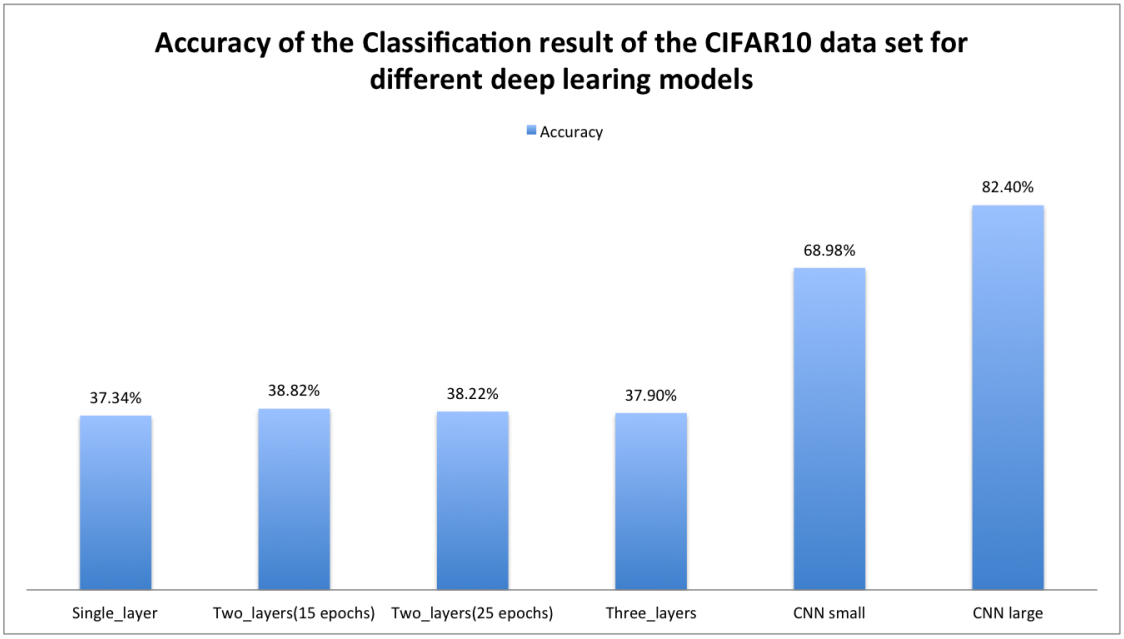


Figure 1: The result of the classification result of the CIFAR10 dataset on different models used in this submission.

2 Part 1

2.1 Description of the assignment part 1

Develop a simple one layer neural network to classify the CIFAR10 dataset. Play with learning rates, number of training steps, training sample size, etc and see how well you can do. This is a harder dataset than MNIST, and the downsampling to greyscale reduces the image information even further, so anything above 50% is acceptable in this step(in the requirement of the updated email, above 30% is acceptable).

2.2 Submission of the part 1

The **one hidden layer** neural network is used here to classify the CIFAR10 dataset. The number of nodes of this hidden layer is the same as the pixel number of each image which is $3 * 32 * 32 = 3072$. The activity function is "relu". The output layer has 10 nodes and the activity function is "softmax". The accuracy of this one layer network is **37.34**% under 10 epochs.
The Screenshots of the result is given in figure 2.

The code of this part is provided in the file
`big_data_assignment3_single_layer.py`.

```
xiaokes-MacBook-Pro:big_data jimmy$ python big_data_assignment3_single_layer.py
Using TensorFlow backend.
('type of dict', <type 'dict'>)
('type of features', <type 'numpy.ndarray'>)
('shape of features', (50000, 3072))
('lbl[0]', array([ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.], dtype=float32))
('type of lbl', <type 'numpy.ndarray'>)
('shape of lbl', (50000, 10))
Train on 45000 samples, validate on 5000 samples
Epoch 1/10
104s - loss: 2.0894 - acc: 0.2827 - val_loss: 1.8441 - val_acc: 0.3568
Epoch 2/10
92s - loss: 1.5619 - acc: 0.4850 - val_loss: 1.7895 - val_acc: 0.3660
Epoch 3/10
87s - loss: 1.3259 - acc: 0.5635 - val_loss: 1.8365 - val_acc: 0.3644
Epoch 4/10
87s - loss: 1.1354 - acc: 0.6361 - val_loss: 1.8760 - val_acc: 0.3654
Epoch 5/10
87s - loss: 0.9434 - acc: 0.7121 - val_loss: 1.9326 - val_acc: 0.3666
Epoch 6/10
87s - loss: 0.7482 - acc: 0.7928 - val_loss: 1.9998 - val_acc: 0.3670
Epoch 7/10
87s - loss: 0.5682 - acc: 0.8658 - val_loss: 2.0782 - val_acc: 0.3650
Epoch 8/10
87s - loss: 0.4119 - acc: 0.9230 - val_loss: 2.1457 - val_acc: 0.3646
Epoch 9/10
87s - loss: 0.2884 - acc: 0.9601 - val_loss: 2.2196 - val_acc: 0.3702
Epoch 10/10
87s - loss: 0.1976 - acc: 0.9806 - val_loss: 2.3018 - val_acc: 0.3734
Baseline Accuracy: 37.34%
xiaokes-MacBook-Pro:big_data jimmy$
```

Figure 2: The Screenshots of the result of single layer neural network: input:3072, first hidden layer:$3 * 32 * 32 = 3072$ nodes, output:10 nodes. The loss means the loss of the training data, acc means the accuracy of the training data. The val_loss means the loss of the test data, val_acc means the accuracy of the test data.

3 Part 2

3.1 Description of the assignment part 2

Add a hidden layer to the network. So, instead of going from 3072 inputs to 10 outputs, you'll be going from 3072 inputs to N intermediate neurons, then from N intermediate outputs to 10 outputs. You'll need a second set

of weights and biases, and you?ll have to think a bit about what size your weight/bias matrices need to be, but once you?ve got that right the network itself is just two separate matrix multiplication steps (plus the biases) instead of one. Remember to apply a nonlinearity to the output of the hidden layer; tf.nn.relu() is a reasonable choice for this. See how much of an improvement you can get over your single layer network. In addition to learning rate etc. you now have a new hyperparameter to play with: N, the size of your hidden layer. If you?re feeling a bit more ambitious, try adding more layers and see what combination of layer numbers and layer sizes works best for you.

3.2 Submission of the part 2

3.2.1 Two hidden layer neural network

The **two hidden layer** neural network is used here to classify the CIFAR10 dataset. The number of nodes of the first hidden layer is the same as the pixel number of each image which is $3 * 32 * 32 = 3072$. The activity function is "relu". For the second hidden layer, the number of nodes is 128 and the activity function is also "relu". The output layer has 10 nodes and the activity function is "softmax".
The accuracy of this one layer network is **38.82**% under 10 epochs(Screenshots of the result is given in figure 3).
The accuracy of this one layer network is **38.22**% under 25 epochs(Screenshots of the result is given in figure 4).
The code of this part is provided in the file
`big_data_assignment3_two_layers.py`.

From the result we can get the conclusion below:

- The result of two hidden layer network is slight better than the single layer network.
- When the epoch is bigger enough, there is almost no improvement of the result on the test data
- The accuracy of the training part is approaching to 100% when the number of the epoch increases, however, the result of the accuracy of the test part is far away from 100%. The reason is overfitting. In order to fix the overfitting problem. Dropout is used for this two layers network and the overfitting still exists.

```
xiaokes-MacBook-Pro:big_data jimmy$ python big_data_assignment3_two_layers.py
Using TensorFlow backend.
('type of dict', <type 'dict'>)
('type of features', <type 'numpy.ndarray'>)
('shape of features', (50000, 3072))
('lbl[0]', array([ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.], dtype=float32))
('type of lbl', <type 'numpy.ndarray'>)
('shape of lbl', (50000, 10))
Train on 45000 samples, validate on 5000 samples
Epoch 1/10
98s - loss: 2.0824 - acc: 0.2414 - val_loss: 1.8715 - val_acc: 0.3274
Epoch 2/10
104s - loss: 1.5991 - acc: 0.4420 - val_loss: 1.7774 - val_acc: 0.3688
Epoch 3/10
106s - loss: 1.2367 - acc: 0.5785 - val_loss: 1.8043 - val_acc: 0.3844
Epoch 4/10
102s - loss: 0.8294 - acc: 0.7310 - val_loss: 1.9697 - val_acc: 0.3892
Epoch 5/10
100s - loss: 0.4449 - acc: 0.8741 - val_loss: 2.1971 - val_acc: 0.3878
Epoch 6/10
93s - loss: 0.1941 - acc: 0.9592 - val_loss: 2.4371 - val_acc: 0.3926
Epoch 7/10
102s - loss: 0.0825 - acc: 0.9885 - val_loss: 2.7116 - val_acc: 0.3942
Epoch 8/10
99s - loss: 0.0427 - acc: 0.9960 - val_loss: 2.8861 - val_acc: 0.3972
Epoch 9/10
100s - loss: 0.0280 - acc: 0.9974 - val_loss: 3.0222 - val_acc: 0.3908
Epoch 10/10
97s - loss: 0.0250 - acc: 0.9970 - val_loss: 3.1868 - val_acc: 0.3882
Baseline Accuracy: 38.82%
```

Figure 3: The Screenshots of the result of **two hidden layer** neural network: input:3072, first hidden layer: 3072 nodes, second hidden layer: 128 nodes, output:10 nodes. **10 epochs** is used here.

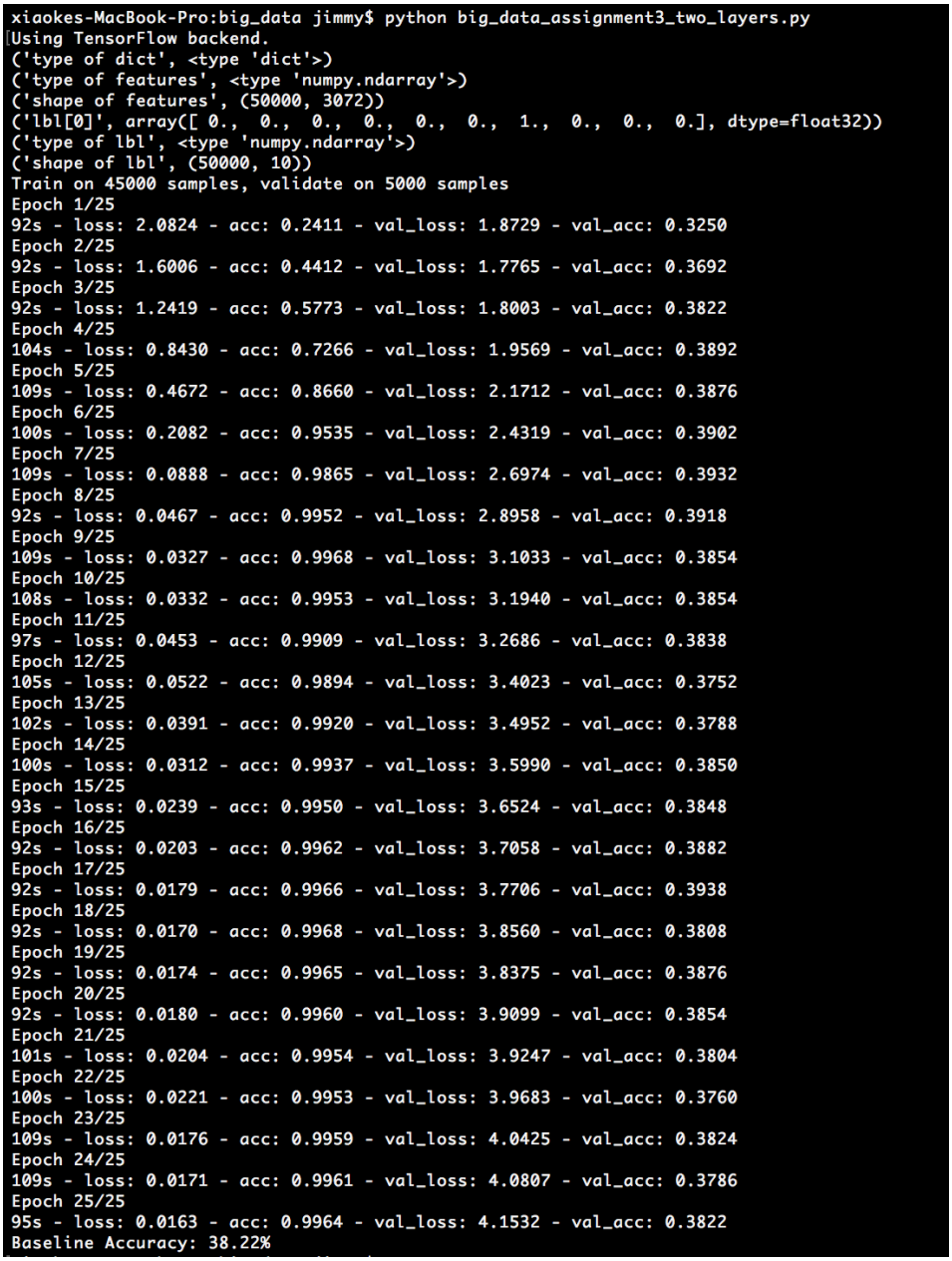


Figure 4: The Screenshots of the result of **two hidden layer** neural network: input:3072, first hidden layer: 3072 nodes, second hidden layer: 128 nodes, output:10 nodes. **25 epochs** is used here.

3.2.2 Three hidden layer neural network

The **three hidden layer** neural network is used here to classify the CIFAR10 dataset. The first two hidden layers are the same as the two layer model. The third hidden layer has 64 nodes with an activity function of "relu". The output layer has 10 nodes and the activity function is "softmax".
The accuracy of this network is **37.90**%(Screenshots of the result is given in figure 5).

The code of this part is provided in the file
`big_data_assignment3_three_layers.py`.

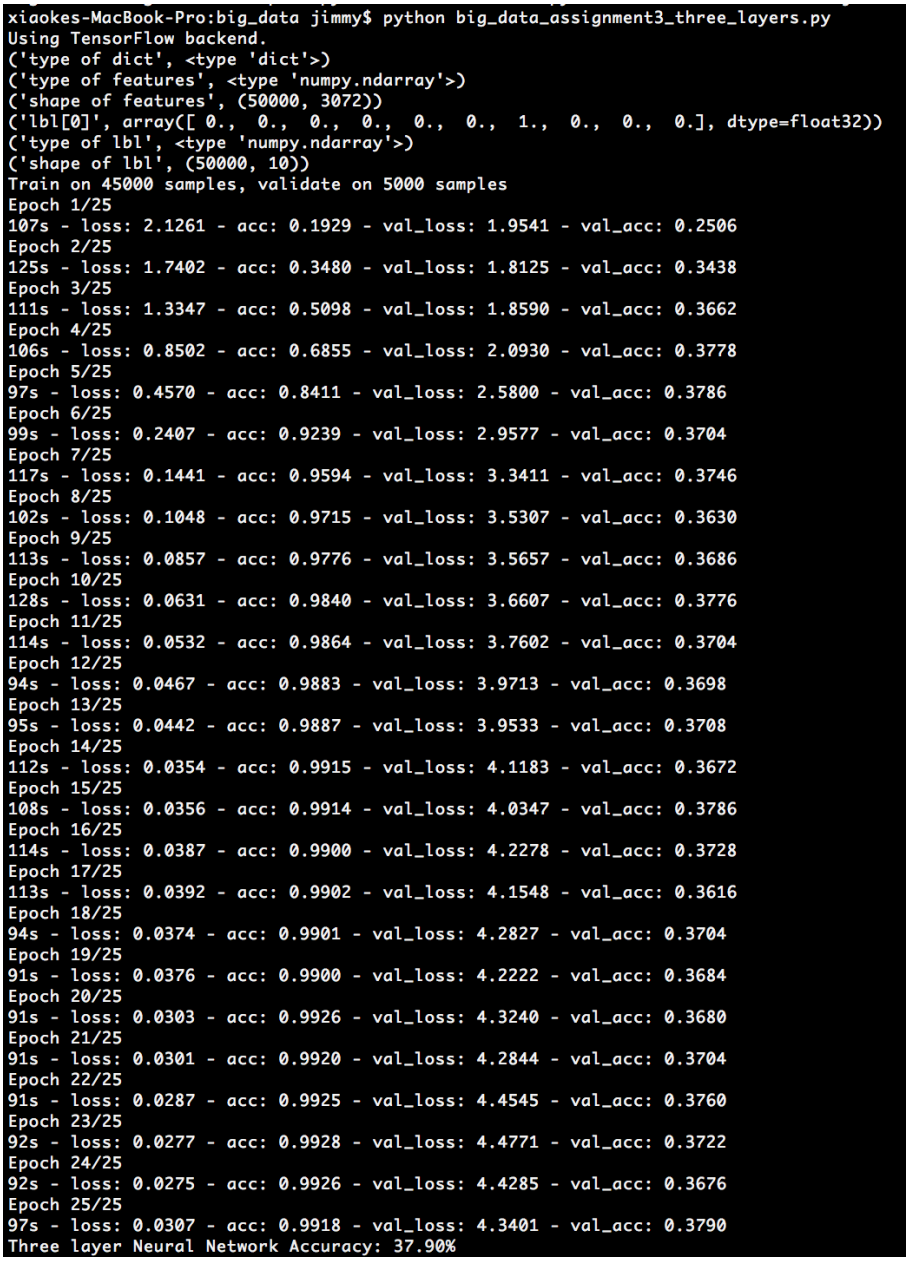


Figure 5: The Screenshots of the result of **three hidden layer** neural network: input:3072, first hidden layer: 3072 nodes, second hidden layer: 128 nodes, third hidden layer: 64 nodes, output:10 nodes. 25 epochs is used here.

From the result we can get the conclusion below:

- The result of the three layers network is not improved and even a bit worse than the two layer network. The reason may be the parameters are not well chosen such as the number of nodes per layer. At the same time, it shows that for the choosing of the deep learning parameters, doing the experiments to choose a better model is an important technique to achieve a better performance.

4 Extra submission

This part is the extra submission for this problem by using the CNN(Convolutional Neural Networks) algorithm.

The original training data of the color image is stored in a 3072 1-D array. In order to do the CNN, it has to be reshaped to the size of $3 * 32 * 32$, where 3 is the three color(RGB) channel, 32 is the number of row pixel and the other 32 is the number of the column pixel. In order to verity whether the data is correctly reshaped, the first 9 images are plotted and shown in figure 6. From the result, we can see the image is not clearly plotted. I printed out the data and found the range of the value of each pixel is not from 0 to 255. It is also not from 0 to 1 as a normalized version of the 0 to 255. It contains some negative value and some positive value. The pixel value of the first image before reshape and after reshape is given in figure 7.

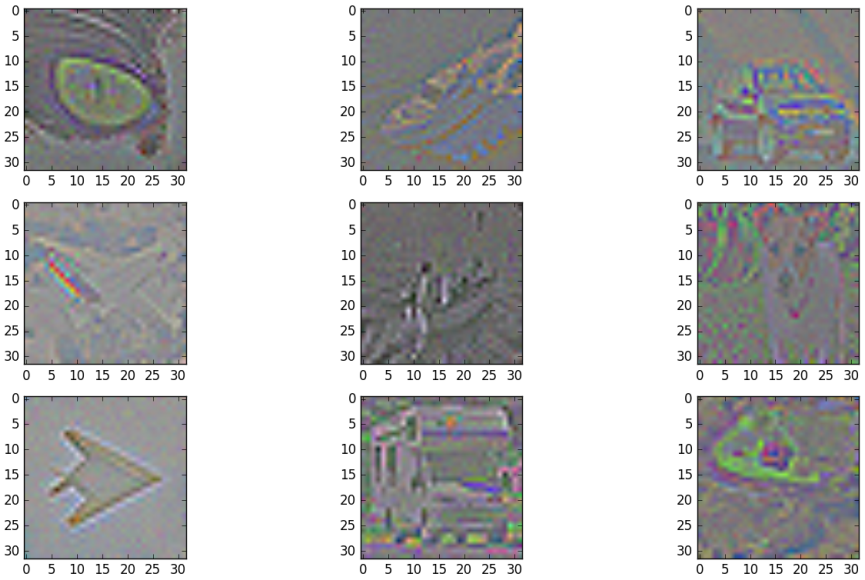


Figure 6: The plot result of the first 9 images. The original 3072 vector are reshaped to $3 * 32 * 32$. To verify whether the image is correctly reshaped, the plotted result is shown here.

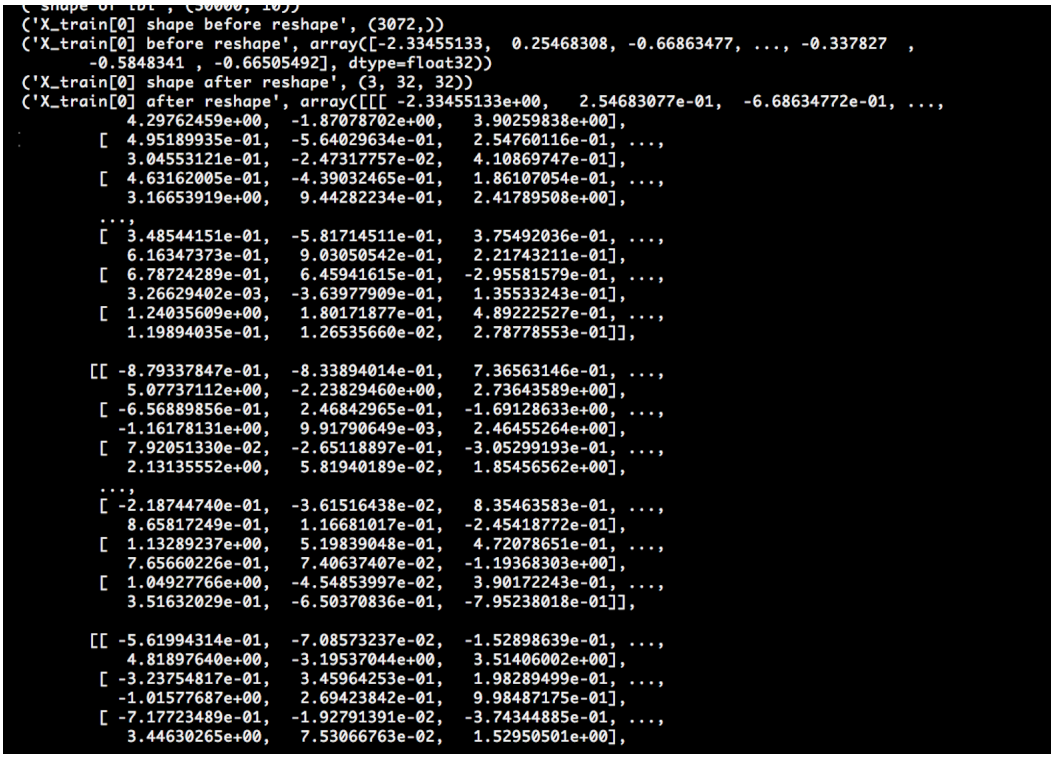
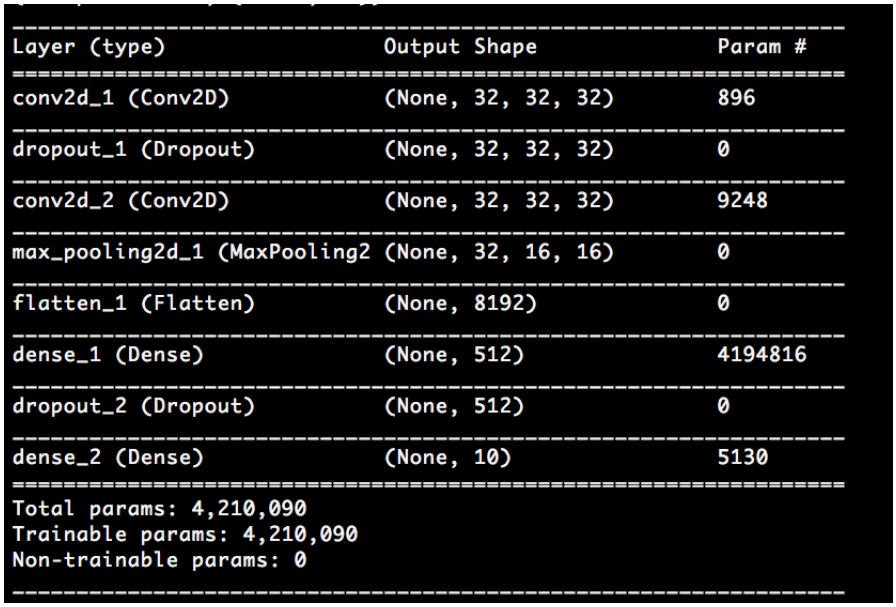


Figure 7: The pixel value of the first image before reshape and after reshape.

As from the plotted result, the image can still be recognized such as the top right is a vehicle and the left bottom is an airplane. The data is used for the CNN algorithm to check the performance.

4.1 Small size CNN

The configuration of this small size CNN is given in figure 8.
The accuracy of this CNN network is **68.98**% under 25 epochs(Screenshots of the result is given in figure 9).
The code of this part is provided in the file
`big_data_assignment3_cnn.py`.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 16, 16)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               4194816
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: The summary of the large CNN network

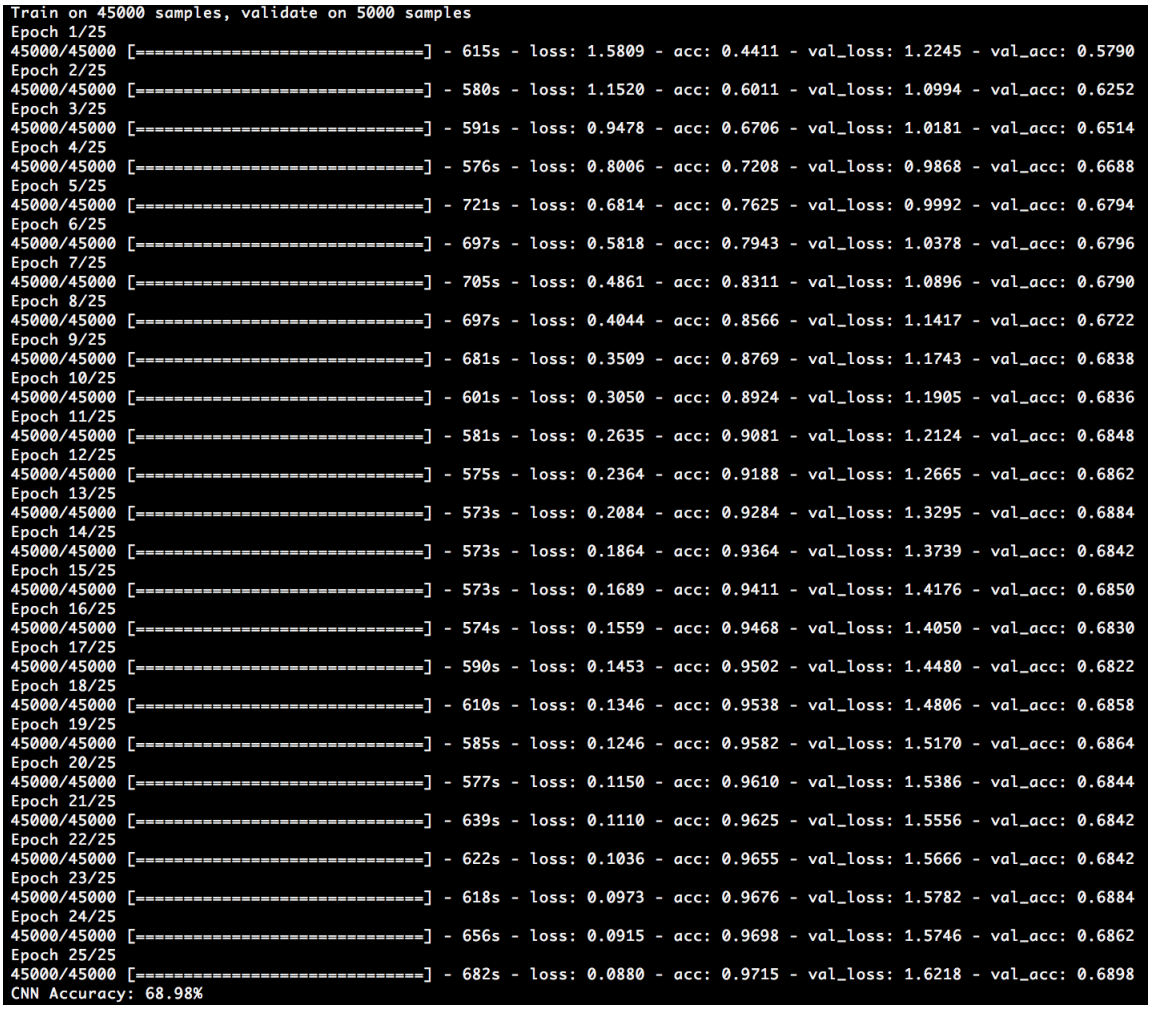
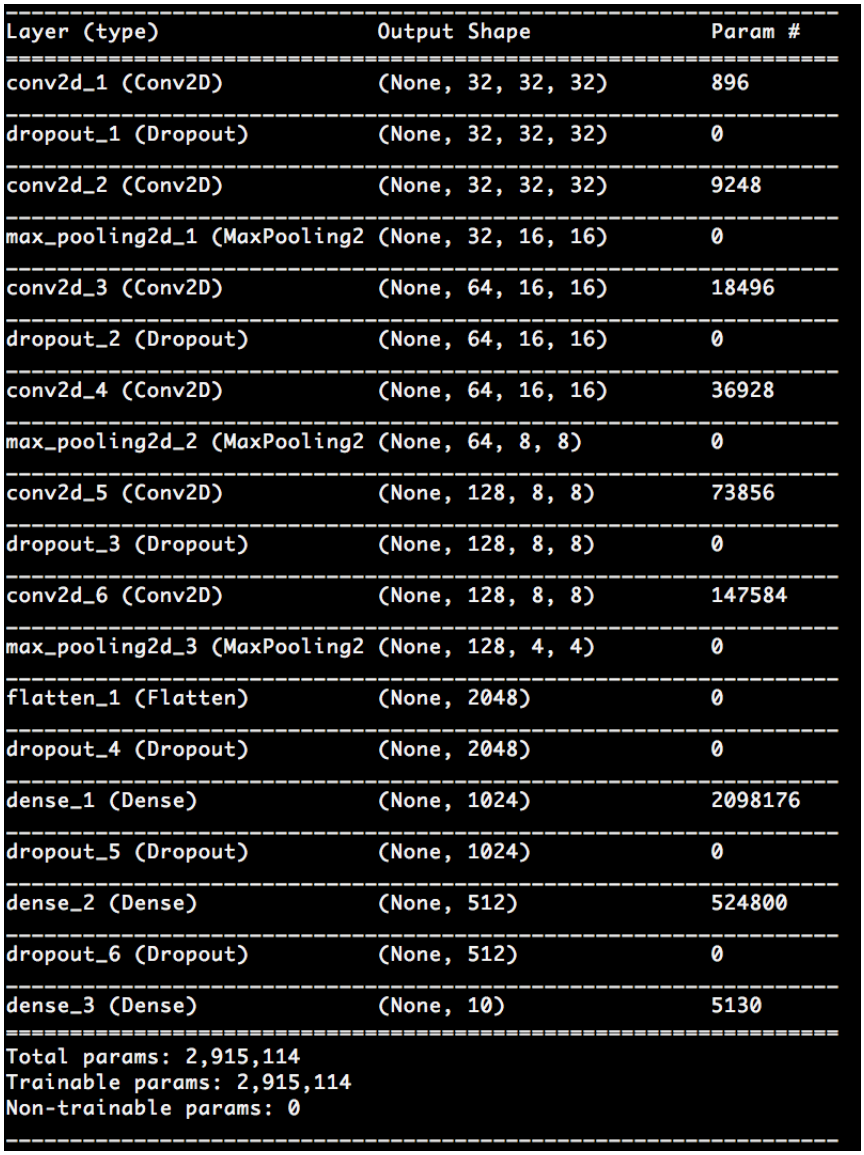


Figure 9: The result of the large CNN network for 25 epochs

4.2 Large size CNN

The configuration of this large size CNN is given in figure 10.
The accuracy of this CNN network is **82.40**% under 25 epochs(Screenshots of the result is given in figure 11).
The code of this part is provided in the file `big_data_assignment3_cnn_large.py`.



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
dropout_1 (Dropout)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 32, 16, 16)	0
conv2d_3 (Conv2D)	(None, 64, 16, 16)	18496
dropout_2 (Dropout)	(None, 64, 16, 16)	0
conv2d_4 (Conv2D)	(None, 64, 16, 16)	36928
max_pooling2d_2 (MaxPooling2	(None, 64, 8, 8)	0
conv2d_5 (Conv2D)	(None, 128, 8, 8)	73856
dropout_3 (Dropout)	(None, 128, 8, 8)	0
conv2d_6 (Conv2D)	(None, 128, 8, 8)	147584
max_pooling2d_3 (MaxPooling2	(None, 128, 4, 4)	0
flatten_1 (Flatten)	(None, 2048)	0
dropout_4 (Dropout)	(None, 2048)	0
dense_1 (Dense)	(None, 1024)	2098176
dropout_5 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
dropout_6 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130

Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0

Figure 10: The summary of the large CNN network

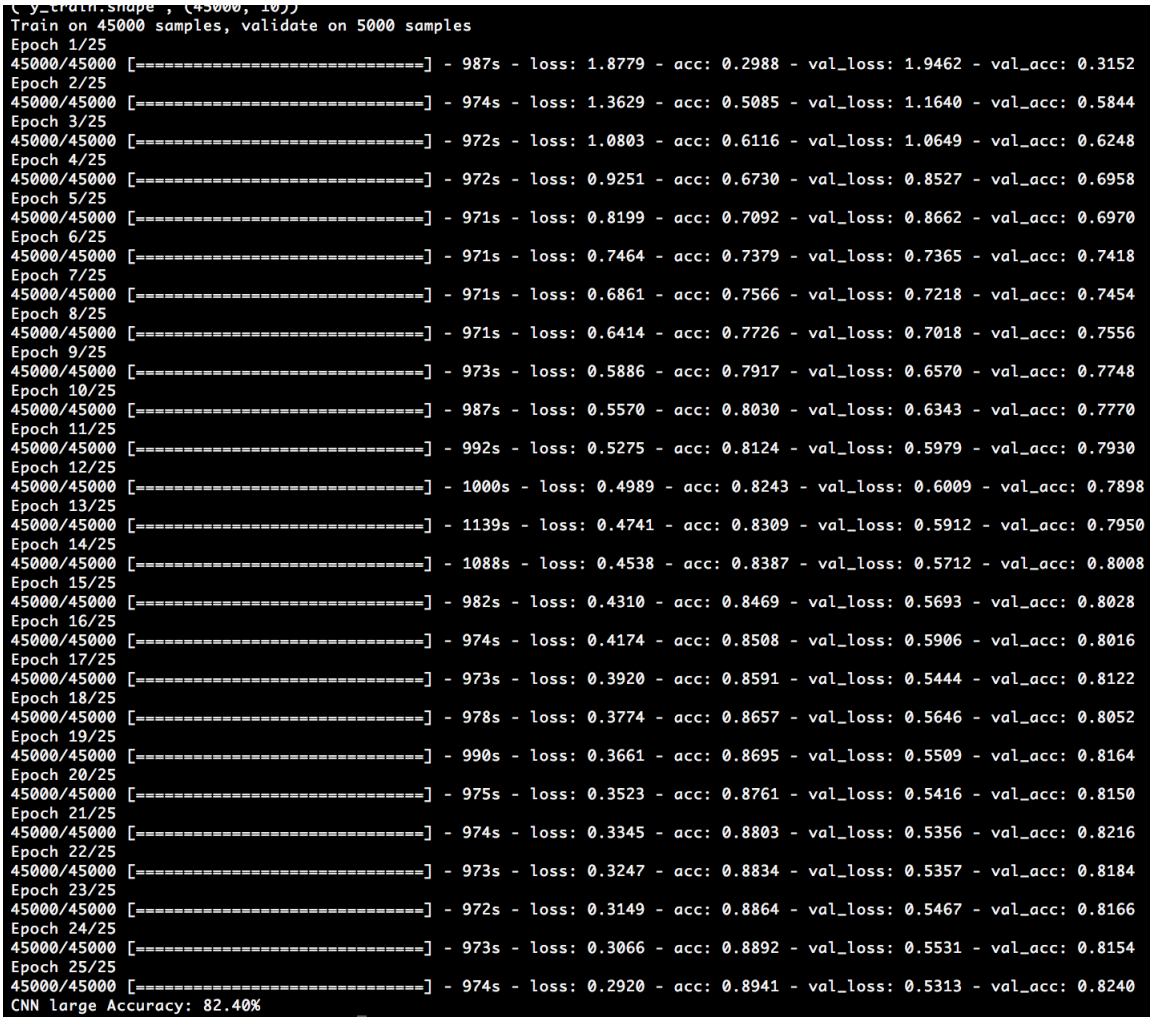


Figure 11: The result of the large CNN network for 25 epochs

4.3 Conclusion

- The CNN can have a better performance than the simply single or multiple layer neural network as it can collect features from the data set.

- The large size CNN can have a better performance than the small size CNN.