Problem 1 : (1)

I use the file “mnist_cnn” in Keras. It trains 60000 data and test 10000 data. There are eight layers.

The first one is convolutional layer (Conv2D), tensor is 32, the kernel size is 3*3 and the activation function is ‘relu’ that is a common function and the input_shape is 28*28 and single color.

The second layer is convolutional layer. Only the tensor is 64 and others thing are same with first one.

The third layer is pooling. Size 2 * 2 means every dimension cut in half.

The forth layer is dropout. It is scaled down to 25 percent.

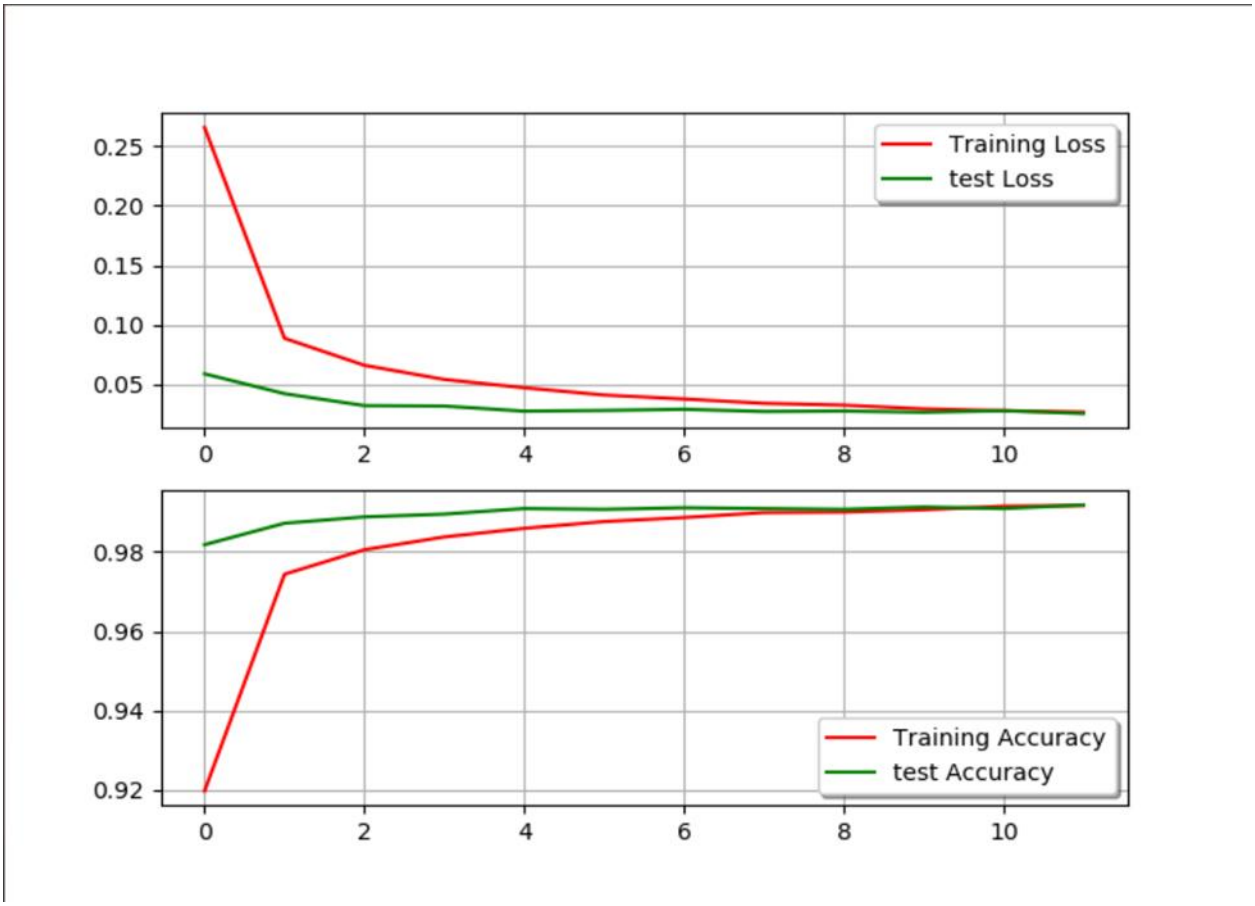
The fifth layer is flatten.

The sixth layer is dense. The parameter is 128. Function is relu.

The seventh layer is dropout too. It is scaled down to 50 percent.

The eighth layer is dense. The parameter is 10. Function is softmax that means classification.

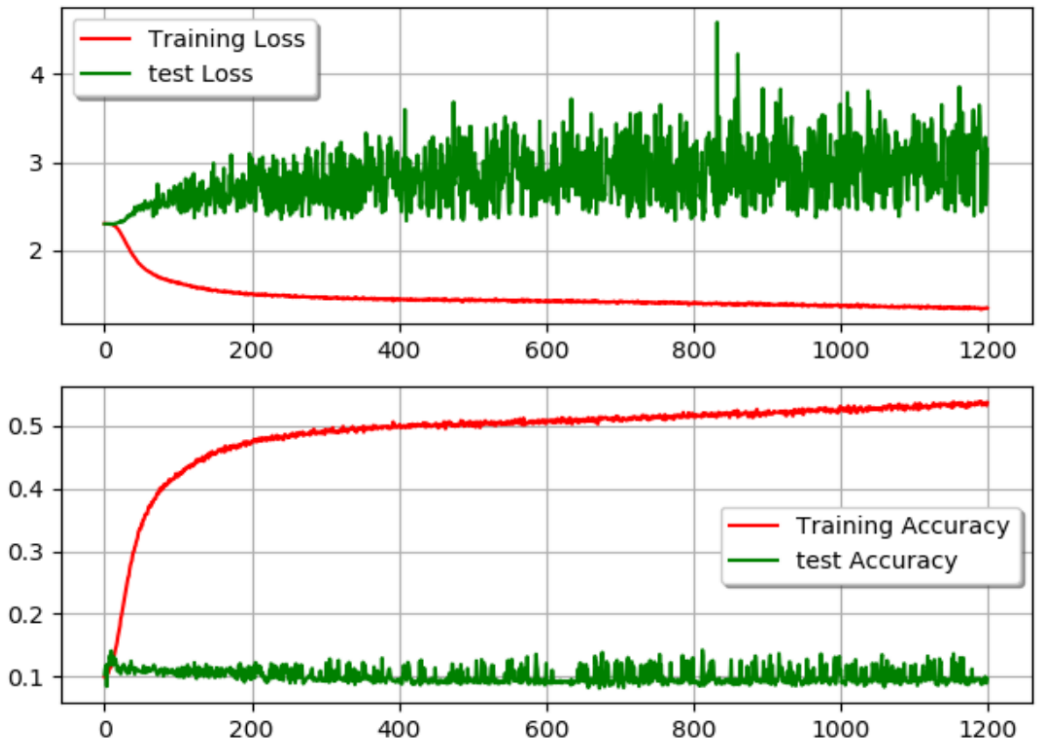
(2)



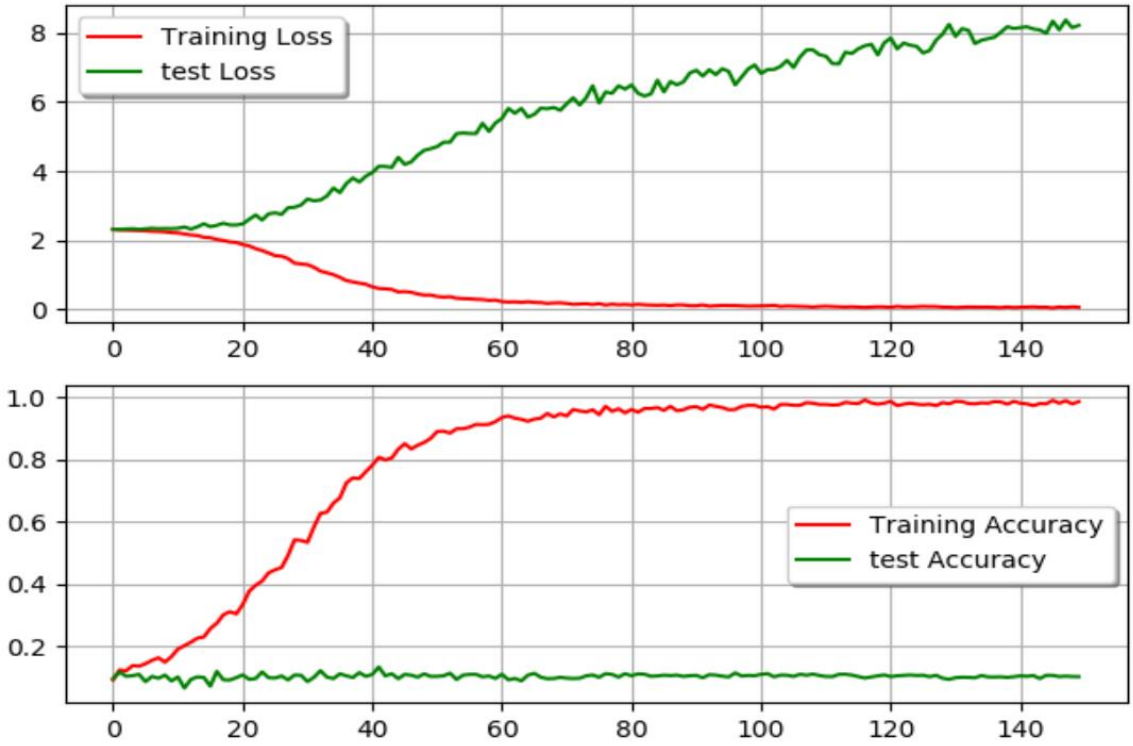
I plot the loss- epochs and accuracy-epochs. It shows when iteration is going to 12, the training accuracy is 0.9906 and loss is close to 0.029463. the val_accuracy is 0.9918, loss is 0.0295. the The performance is good.

Problem 2:

(1)



The first time, I set the epochs as 1200 and there are 60000 trainset and 10000 test set. The training accuracy is 0.54 and loss is 1.25 in final. The val_accuracy is 0.13 and the loss is about 2.45. It is hard for training accuracy to close to 100 because the set is huge we need to do more epochs.



The second time, I reduce the training set to 1000 and set epochs is 150. The training accuracy is close to 1 and loss is close to 0. The val_accuracy is about 0.10 and loss is about 8.21. It is normal because the label is random (0-9).

(2) according to the two results I get, there are very different between accuracy on the training set. In problem 1, the training accuracy starts at 0.92 that already have high value. After the randomizing label, the accuracy starts at 0.18. I think this phenomenon is because at the beginning, the neural network is remembering the data set. it is a progress, so the accuracy at problem 2 starts very low. And in problem 1, the label is not change. When the model trains one time, the model already is pretty good because of correct label. Therefore, it will increase significantly after one epochs.

Problem 3

No matter how randomness is introduced into the model, random noise is added to the image, random shuffling of pixels is carried out, or the image is composed of randomly generated pixels. Even if the labels of the image are randomly generated, the model can also minimize the error in the training set. That's probably because the neural networks are enough for memorizing the entire data set, and it's easy to optimize on random labels. Moreover, randomizing labels is solely a data transformation.

The challenge of achieving good generalization. The first one is Rademacher complexity and VC-dimension. A trivial upper bound on the Rademacher complexity that does not lead to useful generalization bounds in realistic settings. The second one is uniform stability. It is solely a property of the algorithm, which does not take into account specifics of the data or the distribution of the labels.

There are some ways to improve generalization. The first one is early stopping to avoid overfitting. (a) early stopping could potentially improve generalization when other regularizes are absent. (b) early stopping is not necessarily helpful on CIFAR10, but batch normalization stabilizes the training process and improves generalization. The second one is explicit and implicit regularizes. The normalization operator helps stabilize the learning dynamics, but the impact on the generalization performance is only 3%-4%.

The conclusion from the experiment from this paper is that optimization is empirically easy even if the resulting model cannot be generalized.