HOMEWORK#5 Report

Issued: 3/29/2021 **Due: 11:59PM, 4/14/2021**
Name: Siyu Li **ID:2455870216**
E-mail: lisiyu@usc.edu

Problem 1: CNN Training on LeNet-5 (100%)

(a) CNN Architecture (Basic: 20%)

1. Describe CNN components in your own words:

1) Fully connected layer:

Fully connection layer which is also known as Multilayer Perceptron (MLP) is composed of several layers of perceptron and activation function. A perceptron is connected with all the perceptron in the previous layer with different weight and the input of each perceptron is determined by the result of activation function on the linear combination of all perceptron in the previous layer. As is indicated by Universal Approximation Theorem, any function can be approximated with multilayer perceptron activated by non-linear activation function. Therefore, in Convolutional Neural Network, the Fully Connection Layer plays a roll of classifier to the features extracted by the previous convolutional layer and classify them into different classes. The function is shown below:

$$a^{(i+1)} = h(W_{i+1} * a^i + b_{i+1})$$

Here, a^{i+1} represents the activated linear combination of previous ith layer perceptron values while h() denotes the activation function. W^{i+1} is the weight matrix which specifies the way ith layer and (i+1) th layer are connected. And b_{i+1} denotes the bias applied in the (i+1) th layer.

2) Convolutional layer:

Convolutional layer is applied to do feature extraction by using different filters in different convolution kernel. And the weights in the convolutional kernel are trainable parameters. When the training process converges, the weights distribution in a convolution kernel can denote an important feature that needs to be extracted from input image. The size of kernel is usually 3*3 or 5*5 in each layer and remains the same in all the layers. Because with the down-sampling process image is becoming smaller and smaller, even if the size of kernel is the same in every layer they can observe and extract features from different scales.

Similarly, convolutional layer also has activation function but they are not fully connected. The weights are shared by different perceptron in the same channel because

it is a process that convolution kernel slides on an image to extract features. If convolution layer is represented as fully connected layer, there will be lots of zero weight. And another feature is that convolutional layer will not require the size of input.

3) Max pooling layer:

Max Pooling Layer is applied to down-sample the feature map extracted from the previous convolutional layer to do dimension reduction, which can alleviate the computational cost and size of the network effectively. When doing the max pooling, the pixel with the max value in a certain region is maintained while other pixels are discarded. The reason of doing so is that the max value in the region usually represents the max response to a certain feature.

Moreover, Max Pooling is also able to help the network achieve invariance under slight transition, scaling or rotation. Even if some pixels' locations are slightly changed, there will be very likely that the pixel remains after max pooling will show the same extracted feature.

The Max Pooling Layer will also make it possible to extract features from different scales. Because when the image is down-sampled and the convolution kernel size remains the same, the kernel is able to observe features on a larger region in the previous image. This means the network can extract features from multiple scales.

4) Activation function:

Activation Function is a very essential part in CNN because CNN can approximate any function with its non-linearity incurred. Otherwise, without activation function, the network will be composed of a huge linear combination of elements in previous layer, which makes it impossible to approximate any complicated nonlinear function. In CNN usually there are two activation function, one is ReLu function and Softmax function. The form of ReLu function is shown below.

$$h(x) = \max(0, x)$$

This function will let positive part remains meanwhile eliminate the negative one. This property will incur some non-linearity in the network while its positive part will also exhibit linearity which is very useful in preventing gradient explosion and gradient vanishing in relatively deep neural network. The reason is that the weight value back propagation is implemented by chain rule of partial derivative and the derivative of a function's linear part is a constant. This means weight value will not be changed very drastically as that of nonlinear function after each back propagation step. Owning to the properties discussed above, activation function will help CNN achieve a better classification accuracy.

5) Softmax function:

Softmax function is applied as the active function in the fully connected layer of CNN which is used to do classification. The classification criteria is Cross Entropy which measures the similarity between each class label and corresponding weighted

combination of output values in the previous layer. In order to make effective comparison at the same scale, the different output combinations from the previous layer which lead to different class will be normalized and form a probability distribution. Softmax function will transform a vector of different scale values which includes the different output combinations from the previous layer to a vector containing probability values ranging from 0 to 1. After the Softmax Function transform, cross-entropy can be calculated easily with the vector of probability. The form of Softmax Function is shown below.

$$S_i = \frac{e^{x_i}}{\sum j\, e^{x_j}}$$

Here Si represents the i th output of vector which is a probability value, xi represents the i th value in the original unnormalized vector and the denominator represent the sum of all the score values. The larger the score xi is, the larger probability the classification result it will be i th class.

2. What is the over-fitting issue in model learning? Explain any technique that has been used in CNN training to avoid the over-fitting.

Over-fitting issue is a very common issue in machine learning which is usually reflected by the phenomenon that one model achieves very high accuracy on training set while performs poorly on test set. The major reason is that the parameters in the model are trained to follow the distribution of data in the training set while the distribution of test set may be different from that of the training set due to the noise in the training set and the disparity between training set and test set. Therefore, if a model totally fit the training set, it may mistakenly learn the noise in the training set as useful features and it lose the generalization capability to tolerate the disparity between the training set and test set.

There are several techniques that can be applied to avoid the over-fitting in CNN.

Weight decay is realized by adding regularizer term multiplied by a weight decay parameter to the cost function. L1 regularizer tends to introduce sparsity while L2 regularizer tend to shrink relatively not useful weights. With this technique, the network tends to minimize the model weights in each iteration and the overfitting issue is alleviated.

Another method is called data augmentation. One of the reasons that a model will go over-fitting is that the training set is too small and can provide less information than the needs of the model. Usually, a model with very larger VC dimension which usually means more complexity and parameters needs a huge amount of data to be trained effectively. And fitting a complex model with very small training set can lead to the situation where the model has very high complexity but can only tell little information about new sample for classification. However, data is very expensive to acquire. Therefore, operation could be done on existing data to generate some new data to exploit the existing data better, which equals to introduce some new data in some sense. In the preprocessing of image classification neural network, images rotation, gray scale

inverse and transition are usually applied in the training set to generate some new data.

Then, cross-validation is another method to avoid overfitting by choosing the proper model parameters via the validation set. Validation set will not be applied to train model and so it can evaluate the performance of a model precisely. Cross-validation is an improvement on validation which can make better use of the valuable training set. In this setting, early stop can be applied to help training process stop at a proper epoch where the training accuracy continues to increase while the validation set accuracy start to drop.

Dropout which means some weights are randomly set to zero during training can also alleviate overfitting issue as model can be regarded as a combination of different model and have more diversity after the operation. It is like the idea of boosting method which directly combines some models together in some way.

3. Explain the difference among different activation functions including ReLU, LeakyReLU and ELU.

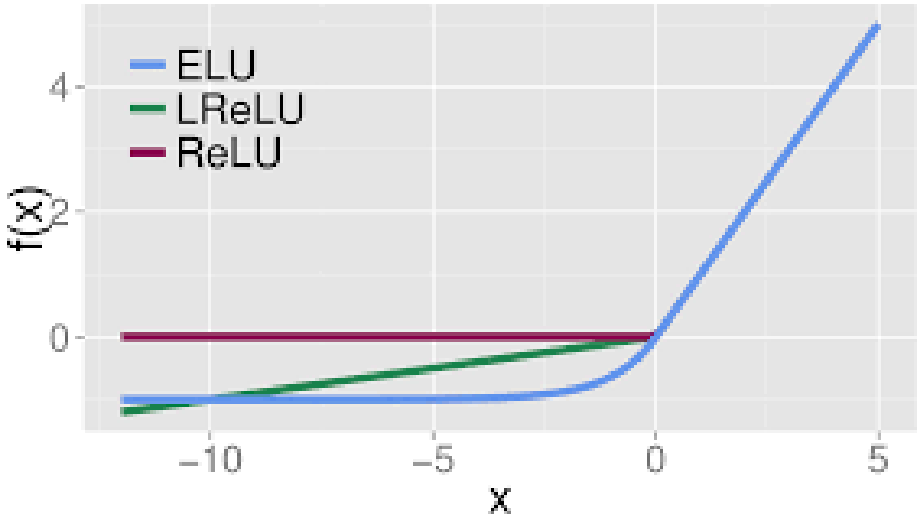


Fig 1.1 Plot of ReLU, LeakyReLU and ELU

ReLU is an activation function in the neural network which will introduce non-linearity and enable the network to approximate any complicated engineering function.

$$ReLU(x) = \max(0, x)$$

This function will let positive part remains meanwhile eliminate the negative one. This property will incur some non-linearity in the network while its positive part will also exhibit linearity which can prevent gradient explosion and gradient vanishing in relatively deep neural network. The reason is that the weight value back propagation is implemented by chain rule of partial derivative and the derivative of a function's linear part is a constant. This indicates that weight value will not be changed very drastically as that of nonlinear function such as sigmoid after each back propagation step.

Moreover, the linearity of ReLU function means it has less computational complexity

compared to nonlinear activation function such as Sigmoid and tanh.

However, the major flaw of ReLU is the dead ReLU problem which means ReLU will fail to work if the input is negative. This can hardly be a problem in forward propagation meanwhile in back propagation the gradient update will fail with negative input.

Moreover, ReLU is not a function centered with zero and this will incur a slower rate of convergence.

To solve the dead ReLU issue and the non-zero centered problem, **Leaky ReLU** is introduced and it is shown in Fig 1.1.

$$Leaky\ ReLU(yi) = \begin{cases} yi, & if\ yi > 0 \\ ai * yi, & otherwise \end{cases}$$

Leaky ReLU solves the issue of dead ReLU problem by introduce very small linear component to negative input to eliminate zero gradient during back propagation. And the range of Leaky ReLU is extended from $(0, \infty)$ to $(-\infty, \infty)$.

ELU also address the issue of dead ReLU by giving negative input values calculated by a certain function rather than zero only in back propagation. Its function form is shown below and its plot is in Fig 1.1.

$$ELU(x) = \begin{cases} x, & x > 0 \\ a * (e^x - 1), & otherwise \end{cases}$$

ELU's output average approximates more to zero and it can converge the cost to zero faster. And the negative part of input will converge to a certain negative value when it goes negative while in the Leaky ReLU the output of negative input will not converge at all. This will help reduce the negative value forward propagation and the uncertainties in the weights update.

The major drawback of ELU is the non-linearity of its negative part function, which will lead to more computational complexity and cost.

4. Read official documents of different loss functions including L1Loss, MSELoss and BCELoss. List applications where those losses are used, and state why do you think they are used in those specific cases?

L1LOSS which is also called least average error constructs the loss function by taking the difference between actual value and estimated value and then taking absolute value of the difference.

$$S = \sum_{i=1}^{n} |Y_i - f(X_i)|$$

This function has very stable solution because it has stable gradient no matter what the input is, which will solve the issue of gradient explosion. However, the disadvantage is that it not derivable in the center as there is an abrupt change caused by the symmetric

folding of the plot. **Therefore, L1LOSS function should be applied when there are serious gradient explosion problems.**

MSELOSS which is also known as L2LOSS is acquired by taking the difference between actual value and estimated value and then taking power of the difference.

$$S = \sum_{i=1}^{n} (Y_i - f(x_i))^2$$

The function is smooth and derivable everywhere and has relatively stable solution. However, when the input value is far from the center, the gradient can be very large when solution is acquired by gradient descent method and it may result in gradient explosion.

Both of L1LOSS and MSELOSS should be applied in **regression problem** only because it is only meaningful in calculating continuous value. **And smooth L1 loss function whose center is smooth and derivable is proposed to combine the advantage of both L1LOSS and MSELOSS.**

For classification problem, we should use BCELoss.

BCE loss which represents Binary Cross Entropy loss is a special case of Cross Entropy loss. It is can be and can only be used for binary classification. The formula is shown below.

$$-y * \log(yhat) - (1 - y) * \log(1 - yhat)$$

Here y is the indicator and yhat is predicted value. Let us assume the case that indicator y is 0, which means ground truth is 1 class while y=1 means ground truth is 0. And yhat reflected the predicted value of ground truth 0 which is usually probability while (1-yhat) reflect the predicted value or probability of ground truth 1. With y and (1-y) as multiplied parameter, they play a role of indicator function because when y is 0 the remained part is -log(1-yhat). So, we are actually evaluating the loss of ground truth 1. When (1-yhat) is larger, the loss is smaller. This means that the predicted value or possibility corresponding to a certain class is larger, the better the classifier performs. That is how binary cross entropy loss criteria works.

In the multiple label classification case, the function is called **Cross Entropy loss function**. The multiplier parameter y and (1-y) in the formula are replaced by one-hot code and when the position of a one-hot code vector is 1 and the rest is zero, we are actually observing the loss of a certain class -log(yhat(certain_class)). And terms for other classes are set to zero when multiplied with zero-valued one-hot code vector element. Thus, the Cross Entropy loss function logic that the larger predicted value or probability and the smaller the loss of this class is realized.

(b) Compare classification performance on different datasets (Basic: 50%)

I. Approach and Procedures

The theoretical part of CNN has already been discussed above in detail, here only the impact of batch size and learning rate will be introduced.

Minimizing and finding the gradient direction of loss function has always been an essential issue of neural network training.

The first method is to traverse the entire training set and only calculate the loss function once and update the gradient and do back propagation to change weights. This method which is called Batch Gradient Descent is computationally expensive and cannot support online learning. It will also be less accurate due to the too many parameters caused by the huge amount of data.

On the contrary, updating the loss function after traversing only one sample in the training set and then calculating gradient and doing weights update by back propagation is called stochastic gradient descent. It is far less computational-intensive but it is not stable in convergence. The loss function oscillates drastically during convergence.

To overcome the disadvantage and reach a compromise between the two methods above, mini-batch gradient descent is proposed. This method classifies the data into several batches of certain size and update parameters every batches. It has the advantage of both computationally efficient and high stability in convergence. Larger mini-batch size will result in faster convergence time while smaller mini-batch size will make it slower. Proper mini-batch size can strike a very good balance between accuracy and convergence rate. **Therefore, mini-batch size is a very important parameter for turning.**

Another parameter for turning is learning rate, smaller learning rate will make it converge slower while object optimal point can be approximated better. Larger learning rate can make it converge much faster meanwhile it is easier to miss the optimal point.

Initial parameter will also affect the result of convergence, here we tried Ones(), he_normal() and GlorotNormal() as initial parameter.

L1 and L2 regularizer are also used to avoid overfitting in the convolutional and fully connected layer.

In the model building and training step based on **tensorflow framework in python**, **Sequential()** from **keras.models** is used to specify the detail of the CNN. Two **Conv2D** layers and two **MaxPooling2D** layers are used to build the feature extraction part. Then,

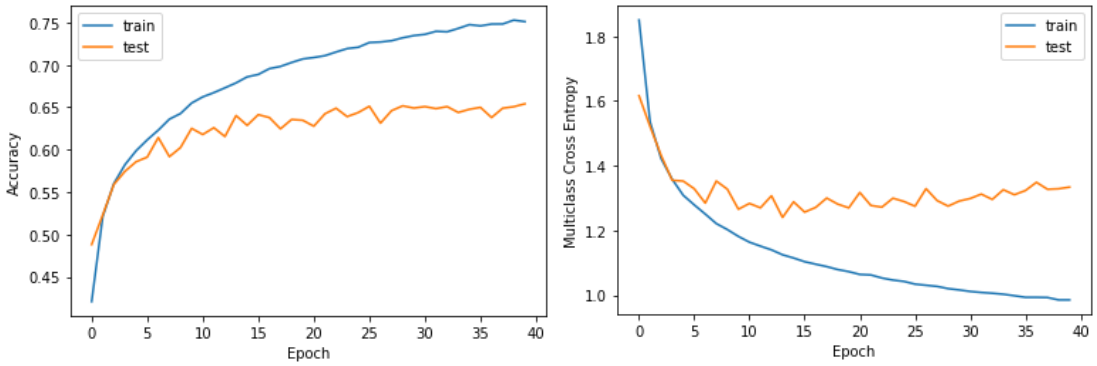
Flatten() is used to reshape the extracted feature map and **Dense()** is used to construct the fully connected layer. In order to make the **mnist-fashion** and **mnist** dataset which is 28*28*1 compatible with the 32*32 input requirement of **Lenet5**, padding is also applied by function **np.pad()** from **numpy**.

Then the model is compiled and type of loss criteria (categorical cross entropy), optimizer (Adam, SGD etc. respectively) and metrics (accuracy) are specified. Then in the model evaluation part, test set data is used as validation to evaluate the performance.

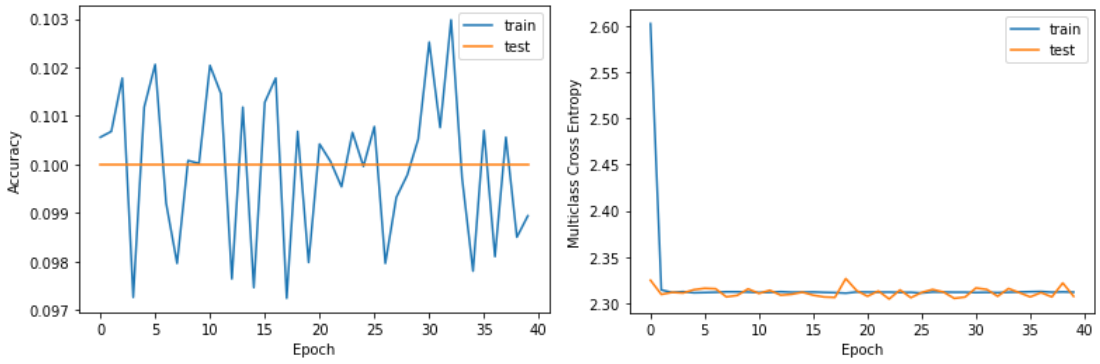
Finally, **model.history** is applied to plot the performance curve revealing the relationship between epoch and accuracy of both training set and test set.

II. Experimental Results

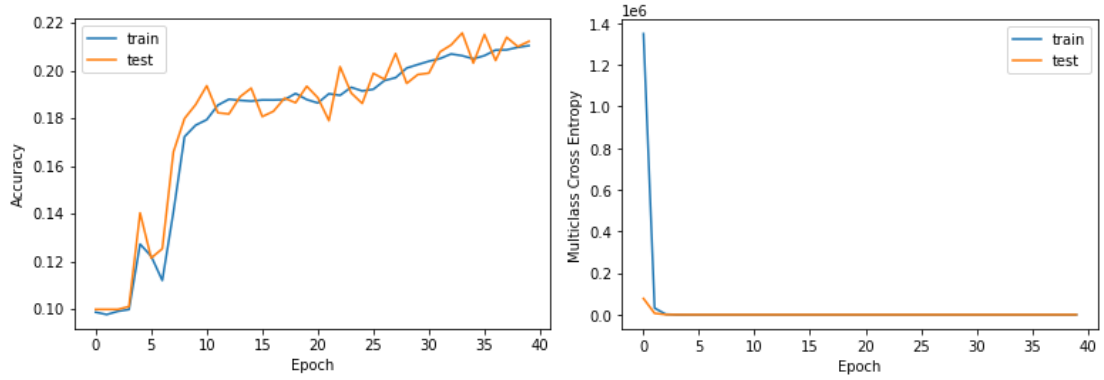
1.Cifar-10 Classification Result



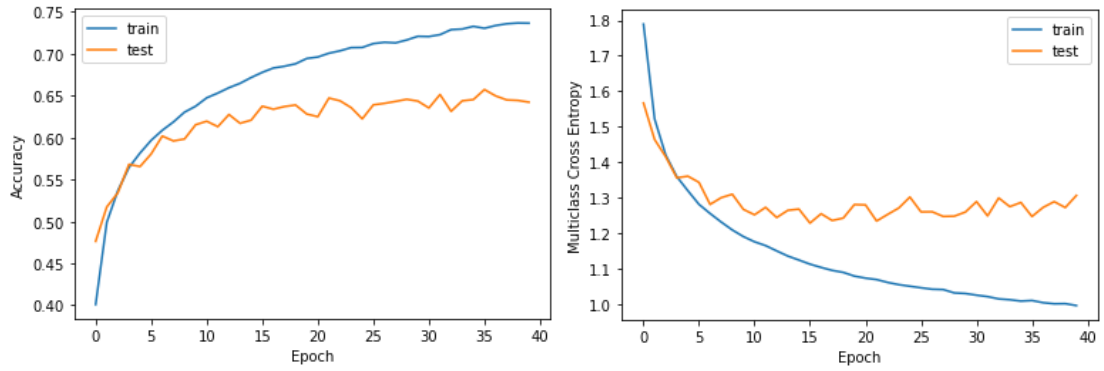
(a) learning rate=0.0005, mini-batch size=16, Number of epochs = 40, regularizer=L2(0.001), initializer= he_normal(), optimizer='Adam'



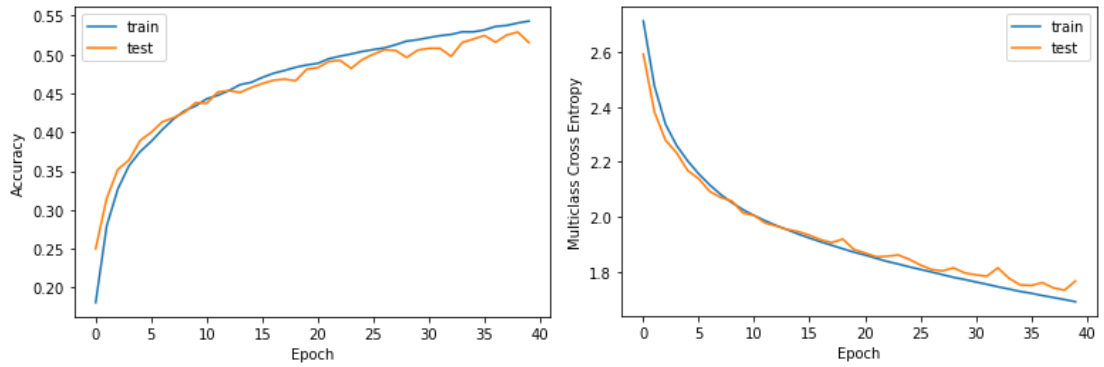
(b) learning rate=0.05, mini-batch size=16, Number of epochs = 40, regularizer=L2(0.001), initializer= he_normal(), optimizer='Adam'



(c) learning rate=0.0005, mini-batch size=16, Number of epochs = 40, regularizer=L2(0.001), initializer= Ones(), optimizer='Adam'



(d) learning rate=0.0005, mini-batch size=16, Number of epochs = 40, regularizer=L2(0.001), initializer= GlorotNormal(), optimizer='Adam'



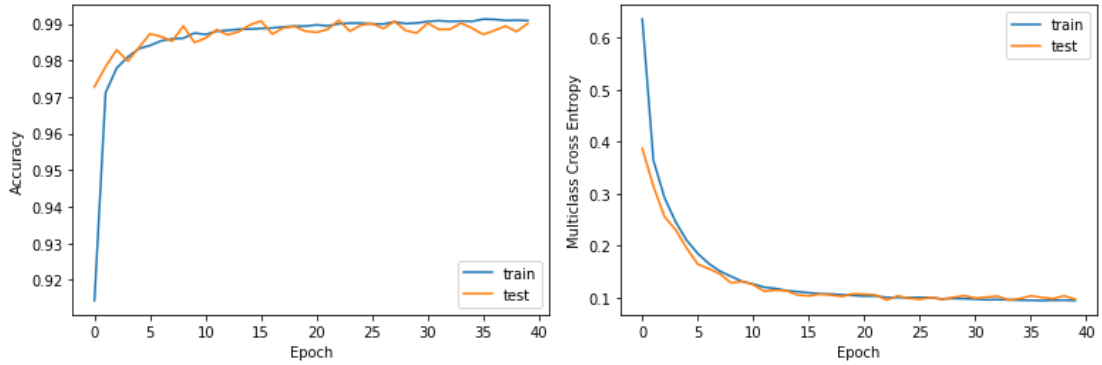
(e) learning rate=0.0005, mini-batch size=16, Number of epochs = 40, regularizer=L2(0.001), initializer= he_normal(), optimizer='SGD'

Fig 1.b.1 Performance Curve Plot of Cifar10 under Different Parameter Setups

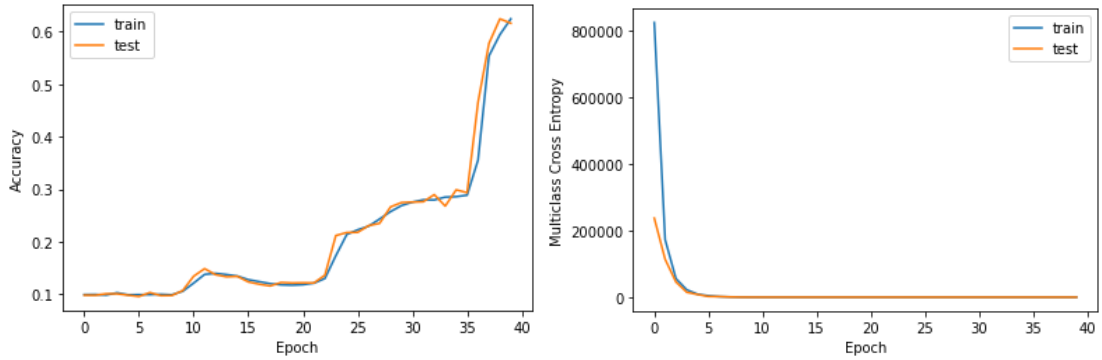
In the training process in **cifar10** dataset, convolution kernel size 5*5 is used. **The best result setting is shown in Fig 1.b.1 (a), we get the best test accuracy above 65% when the training process is fully converged. This meets the requirements on the homework pdf document.** When we compare Fig 1.b.1 (a) and (e), we can conclude that Adam optimizer can achieve better test set accuracy but the convergence process

of SGD optimizer is more stable. There is less fluctuation on the curve of SGD result. When we compare (a), (c) and (d), we can conclude that parameter initialization is very important as the performance of (c) where the initializer is Ones() is much lower than that of (a) and (d). The performance of he_normal() and GlorotNormal() is nearly identical but the test set accuracy of he_normal() result is higher and its convergence process is more stable. When compare (a) and (b), we can conclude that learning rate of 0.0005 performs much better than 0.05 and learning rate setting can make a decisive difference to the performance. With too big learning rate as the 0.05 case, it is very difficult for the optimizer to find the optimal point and the convergence result will oscillate back and force and will not be stable. Under this situation, the test set accuracy is also very low. **To see the exact number of accuracy and loss in each epoch in iteration, you can open the EE569HW5Proplem1bCifar10.ipynb document and set corresponding parameters to see the result.**

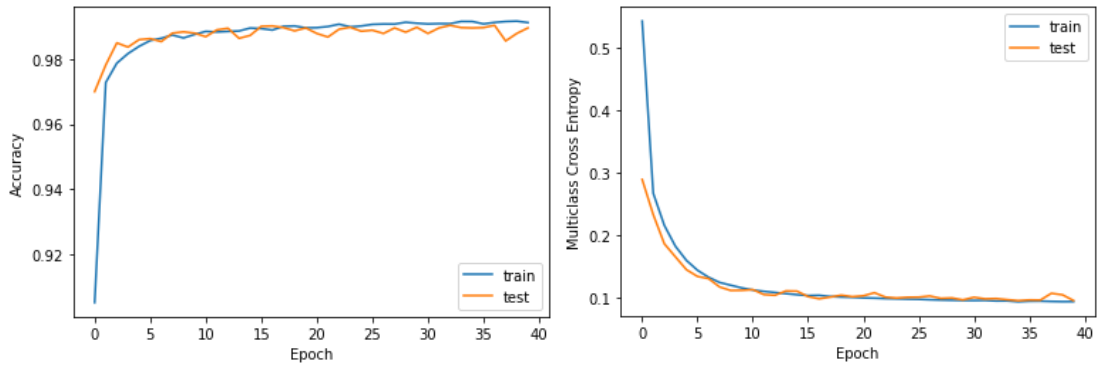
2. Mnist Classification Result



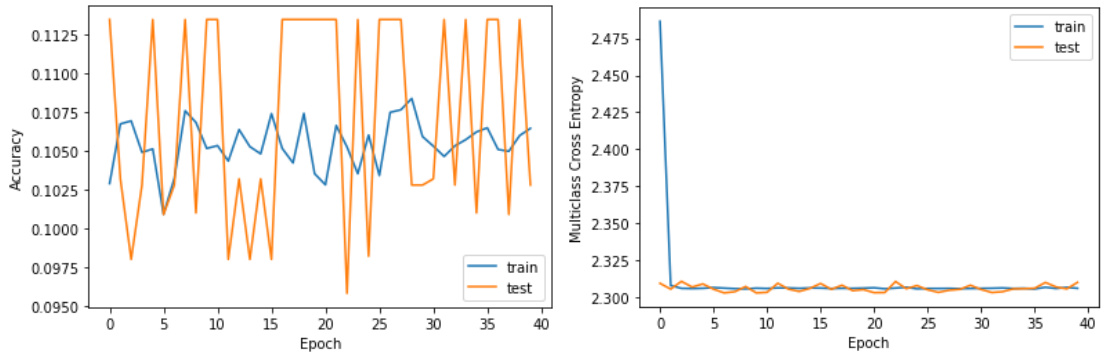
(a) learning rate=0.0005, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= he_normal(), optimizer='Adam'



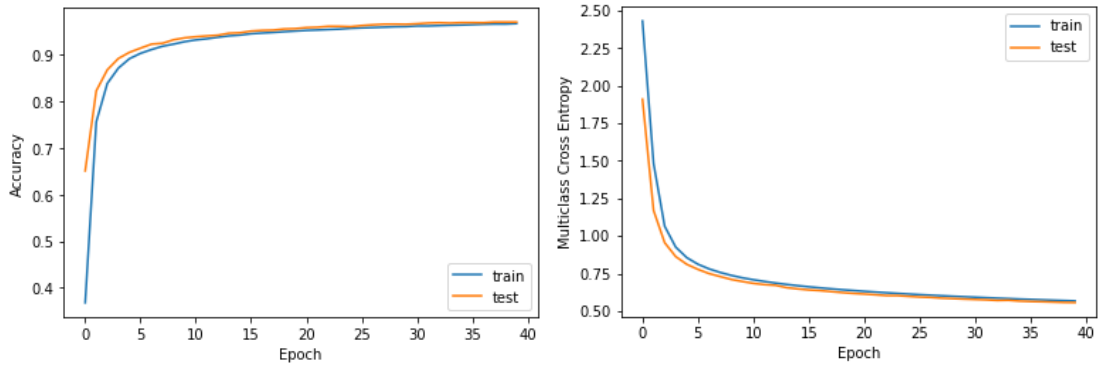
(b) learning rate=0.0005, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= Ones(), optimizer='Adam'



(c) learning rate=0.0005, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= GlorotNormal(), optimizer='Adam'



(d) learning rate=0.05, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= he_normal(), optimizer='Adam'



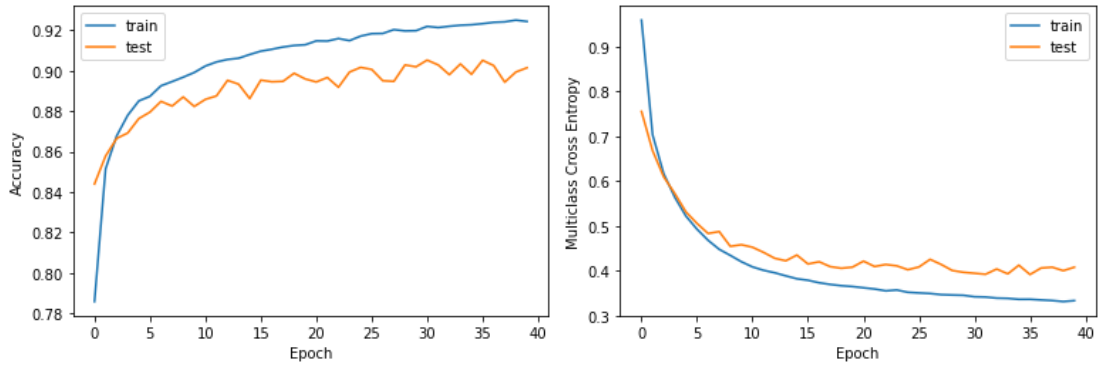
(e) learning rate=0.0005, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= he_normal(), optimizer='SGD'

Fig 1.b.2 Performance Curve Plot of Mnist under Different Parameter Setups

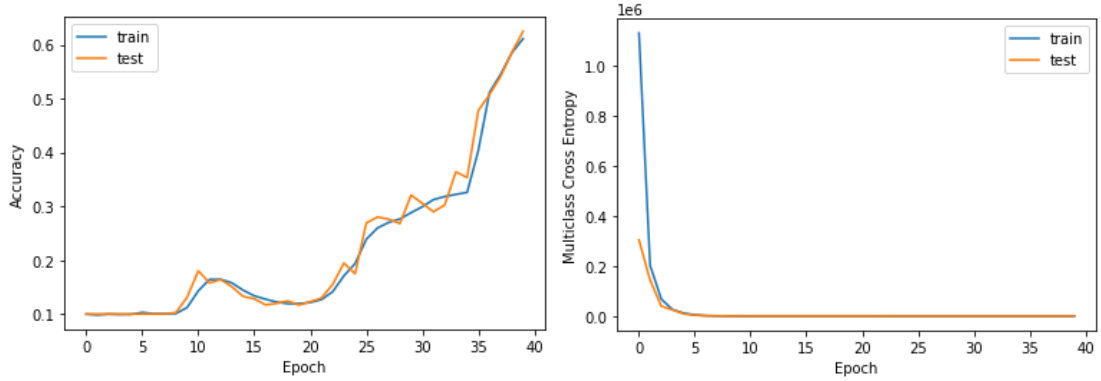
In the training process in **mnist** dataset, convolution kernel size 5*5 is used. **The best result setting is shown in Fig 1.b.2 (a), we get the best test accuracy above 99% when the training process is fully converged. This meets the requirements on the homework pdf document.** When we compare Fig 1.b.2 (a) and (e), we can conclude that Adam optimizer can achieve better test set accuracy but the convergence process

of SGD optimizer is smoother and more stable. There is less fluctuation on the curve of SGD result. When we compare (a), (b) and (c), we can conclude that parameter initialization is very important as the performance of (b) where the initializer is Ones() is much lower than that of (a) and (c). The performance of he_normal() and GlorotNormal() is nearly identical but the test set accuracy of he_normal() result is higher and its convergence process is more stable. When compare (a) and (d), we can conclude that learning rate of 0.0005 performs much better than 0.05 and learning rate setting can make a decisive difference to the performance. With too big learning rate as the 0.05 case, it is very difficult for the optimizer to find the optimal point and the convergence result will oscillate back and force and will not be stable. Under this situation, the test set accuracy is also very low. **To see the exact number of accuracy and loss in each epoch in iteration, you can open the EE569HW5Proplem1bMnist.ipynb document and set corresponding parameters to see the result.**

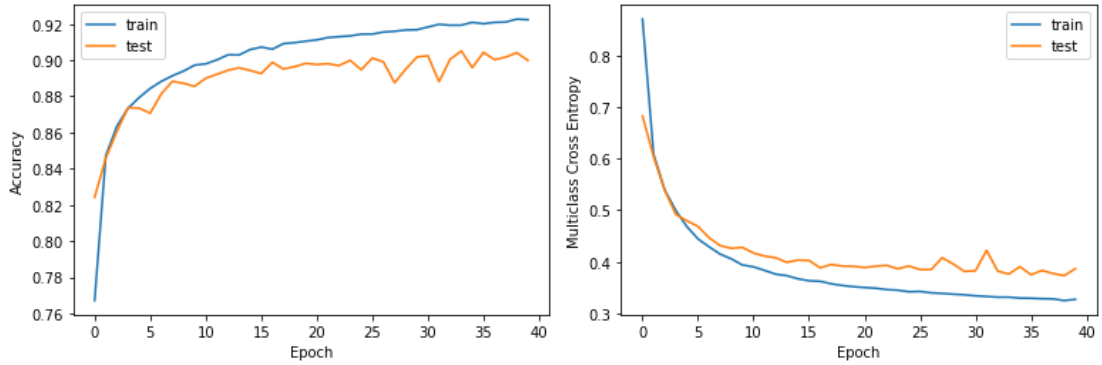
3. Mnist-fashion Classification Result



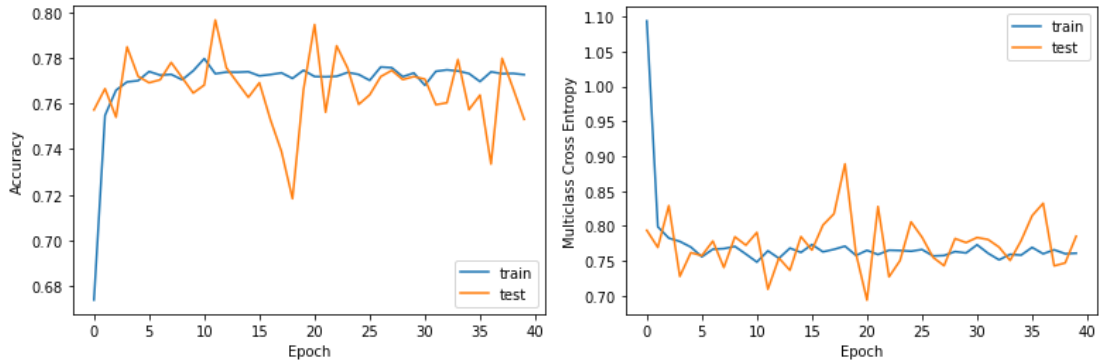
(a) learning rate=0.0005, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= he_normal(), optimizer='Adam'



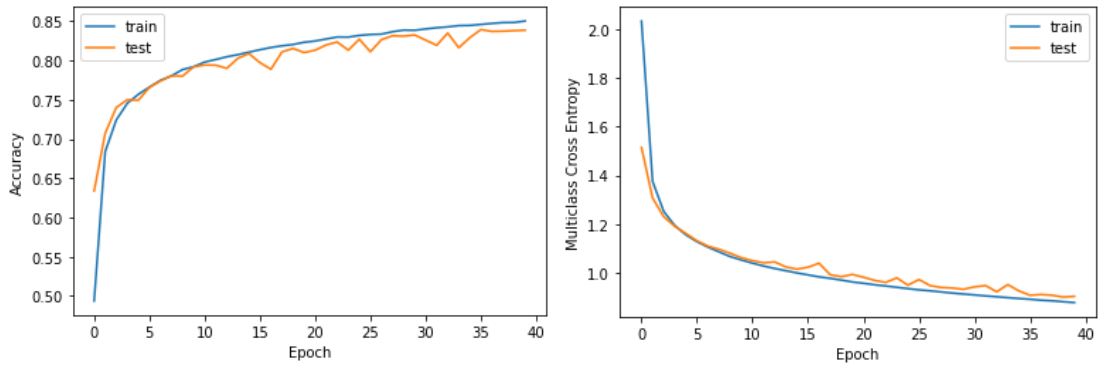
(b) learning rate=0.0005, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= Ones(), optimizer='Adam'



(c) learning rate=0.0005, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= GlorotNormal(), optimizer='Adam'



(d) learning rate=0.05, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= he_normal(), optimizer='Adam'



(e) learning rate=0.0005, mini-batch size=64, Number of epochs = 40, regularizer=L2(0.001), initializer= he_normal(), optimizer='SGD'

Fig 1.b.3 Performance Curve Plot of Mnist-fashion under Different Parameter Setups

In the training process in **mnist-fashion** dataset, convolution kernel size 5*5 is used. **The best result setting is shown in Fig 1.b.3 (a), we get the best test accuracy above 65% when the training process is fully converged. This meets the requirements on the homework pdf document.** When we compare Fig 1.b.3 (a) and (e), we can conclude that Adam optimizer can achieve better test set accuracy but the convergence

process of SGD optimizer is smoother and more stable. There is less fluctuation on the curve of SGD result. When we compare (a), (b) and (c), we can conclude that parameter initialization is very important as the performance of (b) where the initializer is Ones() is much lower than that of (a) and (c). The performance of he_normal() and GlorotNormal() is nearly identical but the test set accuracy of he_normal() result is higher and its convergence process is more stable. When compare (a) and (d), we can conclude that learning rate of 0.0005 performs much better than 0.05 and learning rate setting can make a decisive difference to the performance. With too big learning rate as the 0.05 case, it is very difficult for the optimizer to find the optimal point and the convergence result will oscillate back and force and will not be stable. Under this situation, the test set accuracy is also very low. **To see the exact number of accuracy and loss in each epoch in iteration, you can open the EE569HW5Proplem1bMnistFashion.ipynb document and set corresponding parameters to see the result.**

5. Compare your best performances on three datasets. How do they differ and why do you think there is such difference?

The best performance in cifar10 on test set accuracy is over 65% while the corresponding values of mnist and mnist fashion are respectively over 99% and over 90%. The major difference I think is the complex of dataset. For the cifar10, it has 3 channels and the objects in the image are very complicated objects such as bird, cat, plane, deer and ship. A huge number of features are needed to separate these complicated objects from each other. Meanwhile in mnist and fashion mnist, there is only one channel and the objects in the images are mainly handwritten characters. Therefore, it is relatively easier to have better test set accuracy. In general, in order to do classification on complicated items, more data and more complicated network which usually means more effective features are needed to get better test set accuracy rate.

(c) Apply trained network to negative images (Advanced: 30%)

1. Describe how you can get negatives of the testing set. Implement your idea, then use statistics and sample images to show that you correctly reverse the intensity.

If the image has not been normalized to 0-1, implementing (255-x_test_data) in python directly can reverse the intensity of test set to get negative image. Otherwise, implementing (1-x_test_data) directly in python can reverse the intensity. To prove that intensity is reversed correctly, we draw the same sample image with both the original one and the reversed one to make a comparison.

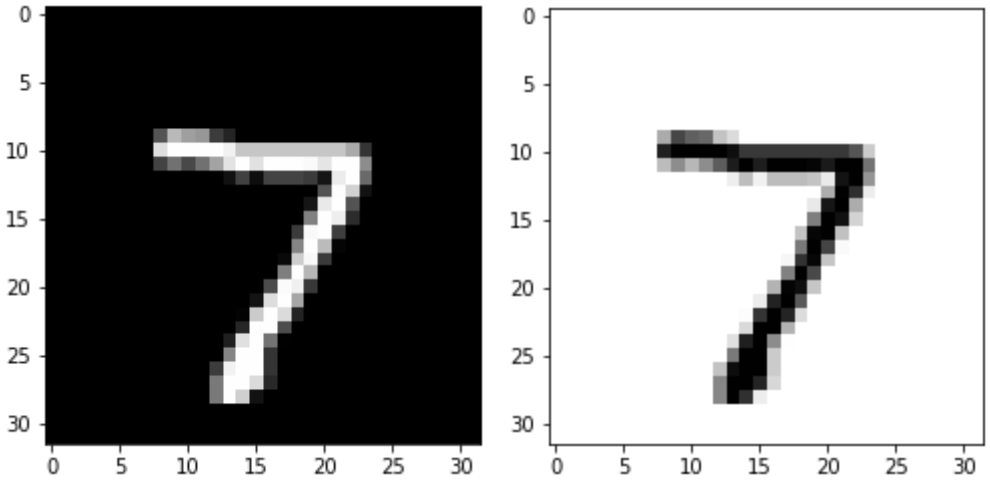


Fig 1.c.1 Original Image (left) and Intensity Reversed Image (right) of the Same Sample Image Chosen Randomly from the Test Set

From Fig 1.c.1, it can be proved that the image intensity is inversed correctly.

2. Report the accuracy on the negative test images using the LeNet-5 trained in part b). Discuss your result.

The accuracy on the negative test images performs terrible, which means human has no trouble recognizing negative test images while the **CNN trained by original training set images and optimal parameters concluded from above cannot recognize them at all.**

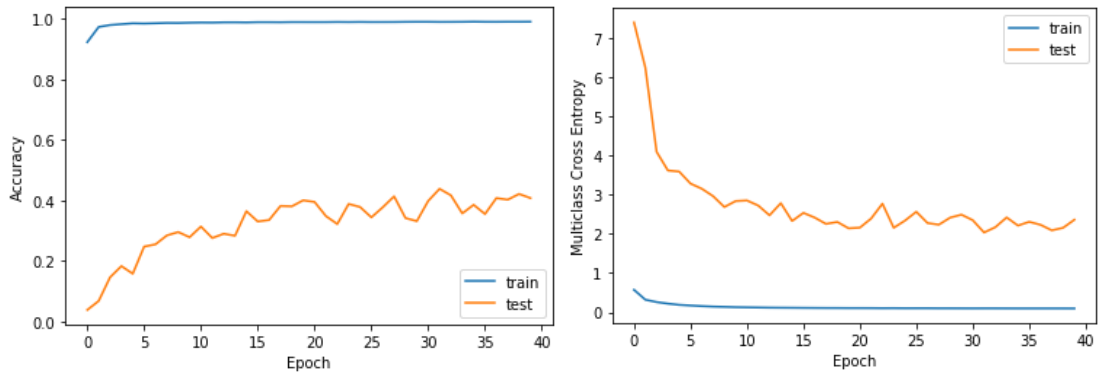


Fig 1.c.2 Performance Curve of Accuracy (left) and Multiclass Cross Entropy Loss (right) of Training Set and Intensity Reversed Test Set

As is shown in Fig 1.c.2, when the intensity of test image set is reversed, the test set accuracy is much lower than the training set accuracy. Its highest test set accuracy is just a little bit more than 0.4. And the test set loss is much high than the training set loss. This indicates that when trained with original training set only **without any data augmentation, the model will have very poor generalization ability** when there is some disparity between the training set and test set. Therefore, data augmentation method needs to be introduced to solve the issue. **To see the exact number of accuracy and loss in each epoch in iteration, you can open the EE569HW5Problem1cMnistReverse.ipynb document to see the result.**

3. Design and train a new network that can recognize both original and negative images from the MNIST test dataset. Test your proposed network, report the accuracy and make discussion.

To solve the issue of very low test set accuracy when test set is reversed in intensity, data augmentation by reversing intensity of half of the training set images is introduced here. The training set is split by half, and we reverse the intensity of one half and maintain the original data of the other half. Then the two datasets are combined together to form a new refined training set. We test the new CNN on both of the reversed test set and the original test set, both shows very good result as is shown in Fig 1.c.3 and Fig 1.c.4 below.

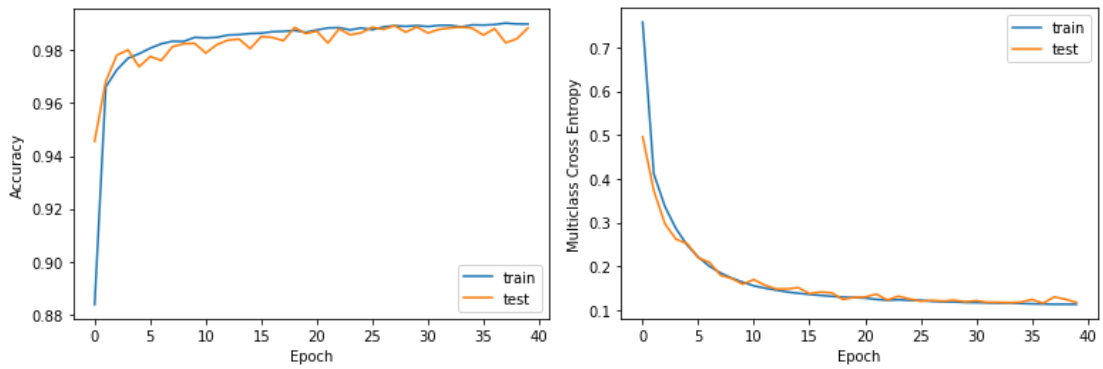


Fig 1.c.3 Performance Curve of Accuracy (left) and Multiclass Cross Entropy

Loss (right) of Refined Training Set and Intensity Reversed Test Set

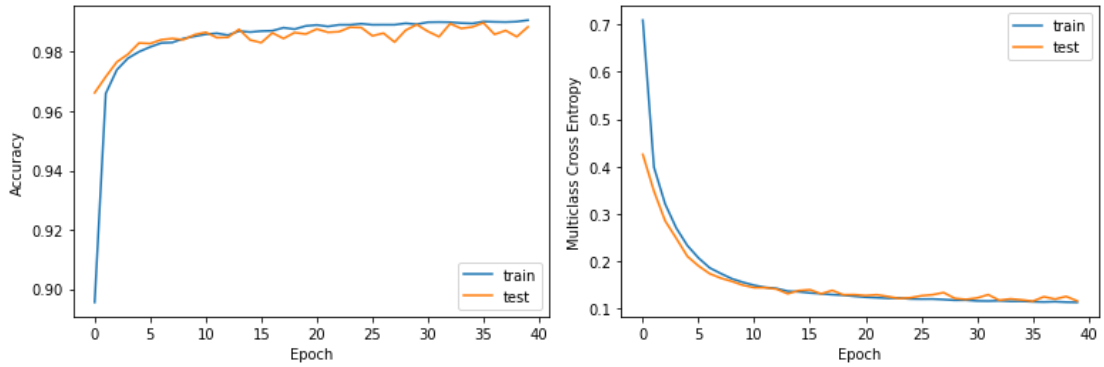


Fig 1.c.4 Performance Curve of Accuracy (left) and Multiclass Cross Entropy Loss (right) of Refined Training Set and Original Test Set

It is proved that with data augmentation method as preprocessing, the model will have much better generalization capability on intensity reversed test set. Besides, it also maintains very good performance on the original test set. **To see the exact number of accuracy and loss in each epoch in iteration, you can open the EE569HW5Problem1cReversedSolution.ipynb document to see the result.**