HW 5

Problem1

a) CNN Architecture

1.

Fully connected layer is the layer that connects all nodes for the two different cluster of nodes, which usually uses in the end of pipeline to connect and reduce NN dimension and use as classifier.

Convolution layer is the layer that use the convolution kernel to convolve with the input and get the output map. Convolution layer is normally used as extracting feature of image

Max pooling layer is the layer that use the max value in a pool to represent the pool region. The layer is normally used to reduce the size of feature map.

The Activation function is an element-wise non-linear function used after some layers, which can add non-linear to network and increase the complexity.

The SoftMax layer is normally used in the end of network for use classification.

2.

The overfitting problem is caused by limited training data with a large number of parameters. The overfitting problem is that training model has high training accuracy verse low validation or test accuracy. The model cannot get good performance on the new data.

Use dropout to drop the FC layer parameters and using Global average pooling instead of FC layer can reduce number of parameters and cure overfitting problems. Besides, we can also choose kernel regularization, like l2 or l1 regularization in convolution layers.

3.

The three ReLU functions have different values when inputs are negative.

ELU: $\alpha \times (e^z - 1)$ when z < 0

ReLU: 0 when z < 0

LeakyReLU: $\alpha \times z$, when z < 0

ReLU function is 0 when the input is negative, this attribute can make computation efficient, but sometimes it will lose the gradient and some parameters cannot be trained. Leaky ReLU can preserve the negative gradient and solve the gradient vanish. However, Leaky ReLU is similar as the linearly function. Especially when most of parameters are negative or positive. Hence, this function may degrade the model to normal linear model. ELU have some gradient when the input is negative but near 0. To some sense, if we choose proper parameters, ELU will be guidable on 0 point.

The three value also have the cons that the range of function is inf due to avoid gradient vanish. Besides, the three non-linear function can solve gradient vanish problem and calculation efficient.

4. L1Loss, MSELoss and BCELoss

L1Loss is the Mean Absolute Error,

$$loss(x, y) = |x - y|$$

The loss is used in regression problem, people cannot get which is higher for x and y. The gradient of loss is the same when x>y or x<y, y is considered as a constant and x is variable. Hence, they can just get the linear difference between x and y.

MSELoss is the Mean square Error,

$$loss(x, y) = (x - y)^2$$

The loss is used in regression problem. People can get the higher gradient when the difference of x and y is higher. This attribute can help accelerate the convergence speed.

Compared with L1Loss, MSELoss achieve higher when the difference is high and lower when difference is small.

BCELoss is Binary Cross Entropy Loss ,

$$loss = -[y_n \times log x_n + (1 - y_n) \times log(1 - x_n)]$$

x_n is the value after softmax, which represent the probability. In practice, The Pytorch function will calculate the sum or the average of a batch. Compared with the traditional Cross Entropy Loss, BCE add another part of entropy that the object is not the target. This loss function is widely used in binary or multiclass classification.

b) Classification performance on different dataset

1) Lenet-5

Convolution Layer 1: kernel size (5,5,6), activate function ReLU
MaxPooling Layer 1: stride=2, kernel size = (2,2)
Convolution Layer 1: kernel size (5,5,16), activate function ReLU
MaxPooling Layer 1: stride=2, kernel size = (2,2)
Flatten Layer
Dense Layer1: produce 120-dimension tensor per image
Dense Layer1: produce 84-dimension tensor per image
Dense Layer3: produce 10-dimension layer per image, activate function SoftMax

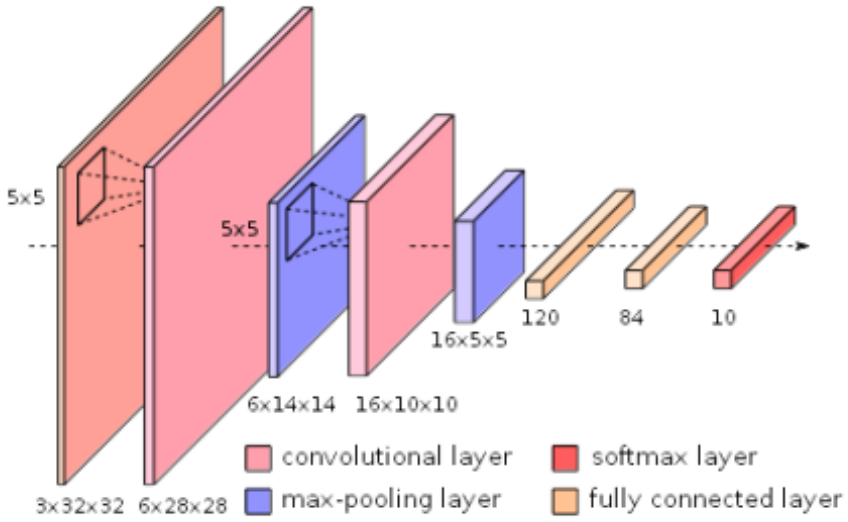


Figure1. The Lenet5 architecture adapted from HW5

2) Dataset

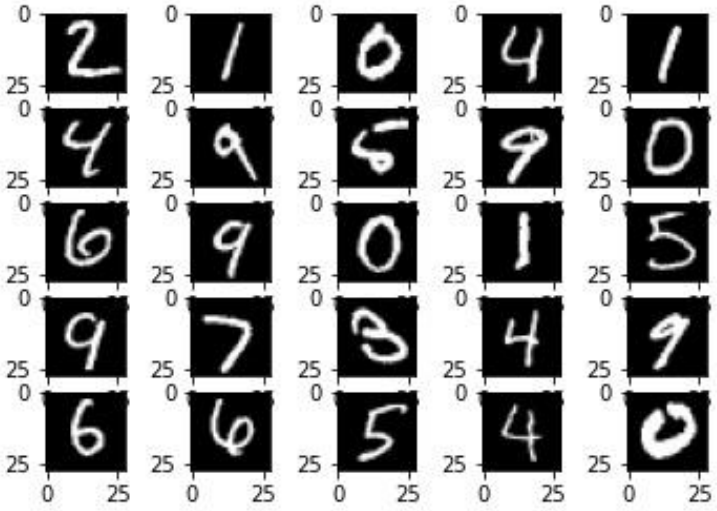


Figure2. MNIST dataset Visualization

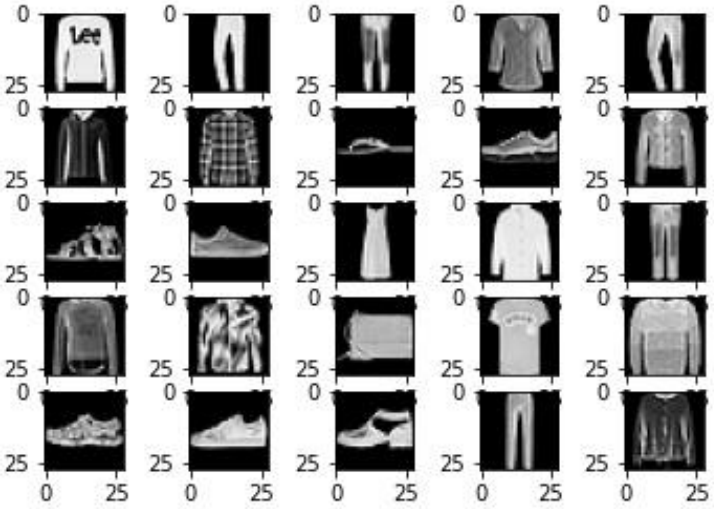


Figure3. Fashion MNIST dataset Visualization

Figure4. CIFAR10 dataset Visualization

3) Optimizer

Deep learning can be regarded as an optimization problem. From every epoch, using Loss function to calculate the difference between output and target. Then, using back propagation to calculate the gradient from every parameter. Finally, using optimizer to change the gradient to decrease loss.

Here I used Stochastic Gradient Descent (SGD) optimizer. Besides, I also introduced momentum which can avoid trapped into local minimum. Besides, I also used Nesterov, which let the momentum to change ahead of original gradient. This method can help accelerate the training process.

$$velocity = momentum \times velocity - learning\ rate \times gradient$$
$$w = w + momentum \times velocity - learning\ rate \times gradient$$

From all setting, I used SGD optimizers with momentum=0.9 and used Nesterov.

4) Loss Function

For loss function, it is common to use Cross Entropy as classification loss. Here I used Cross Entropy as our loss function. First, I need use one-hot encoding to change our label to 10-dimension vectors. The loss is computed from labels and output after softmax.

$$H(p,q) = -\sum p \times log(q)$$

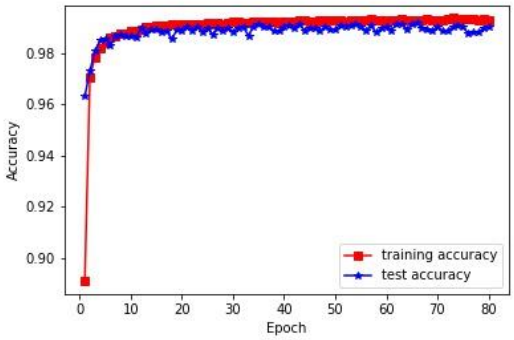
Experiments

1) MNIST Dataset

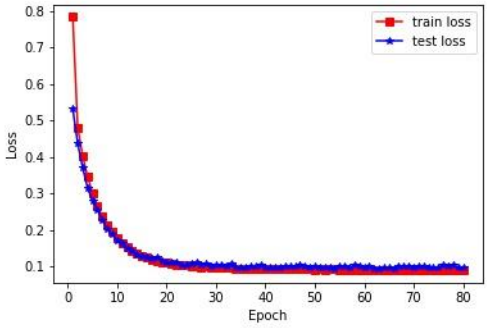
Here the default setting is the batch size 128, For MNIST dataset, I used 80 epochs. I used SGD optimizers with momentum=0.9 and used Nesterov, the only difference is learning rate.

Table1. Test Accuracy on MNIST Dataset

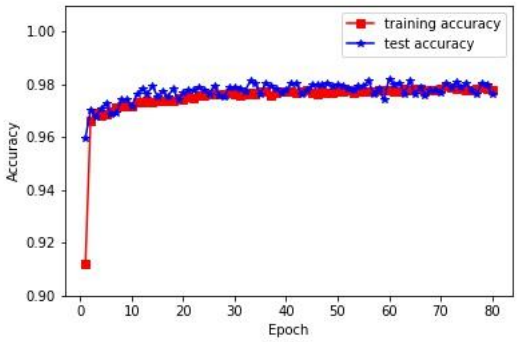
Experiments	Learning rate	Kernel Initialization	Kernel Regularization	Accuracy (mean,std,max)
Setting 1	0.01	Kaiming He Initialization	L2(0.001)	(0.9901,0.0008376,0.9919)
Setting 2	0.01	Kaiming He Initialization	L1(0.001)	(0.9786,0.001611,0.9805)
Setting 3	0.01	Kaiming He Initialization	None	(0.9896,0.00081,0.9908)
Setting 4	0.01	Random Initialization	L2(0.001)	(0.9897,0.0006196, 0.9905)
Setting 5	0.005	Kaiming He Initialization	L2(0.001)	(98.99, 0.0003208,0.9909)



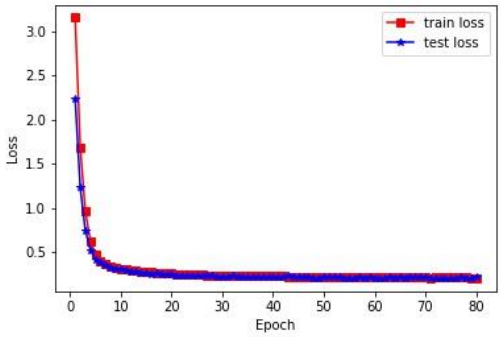
Setting 1



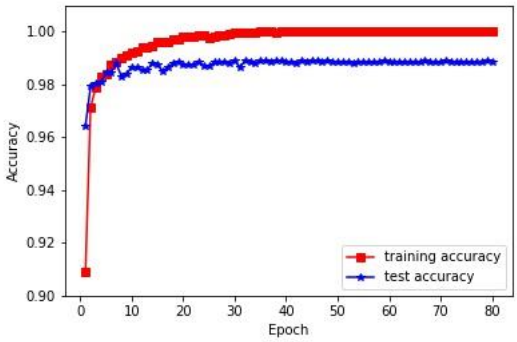
Setting 1



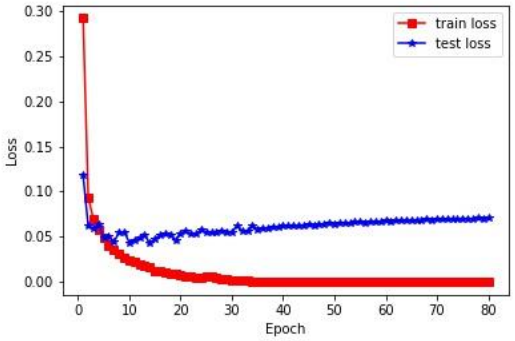
Setting 2



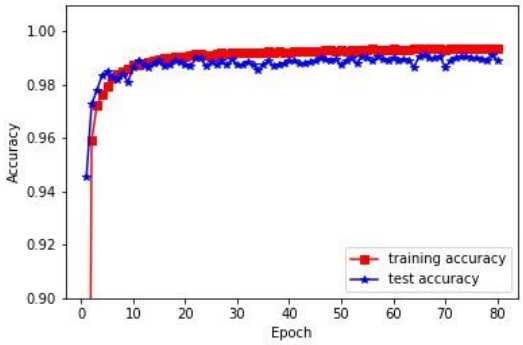
Setting 2



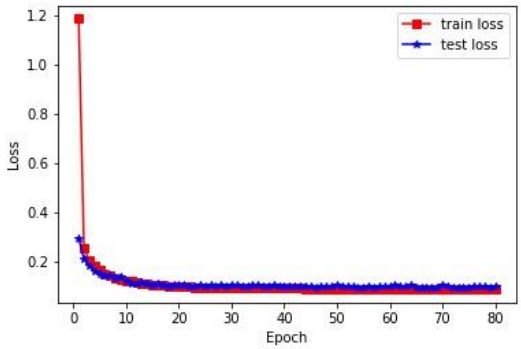
Setting 3



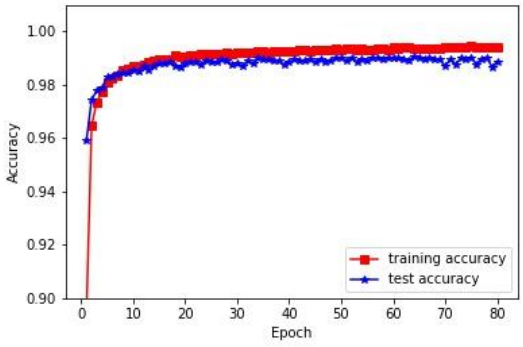
Setting 3



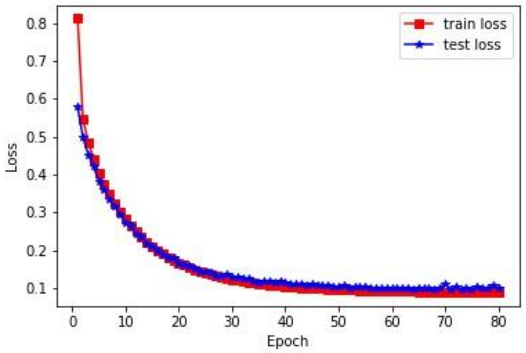
Setting 4



Setting 4



Setting 5



Setting 5

Figure5. Performance Curve on MNIST Dataset

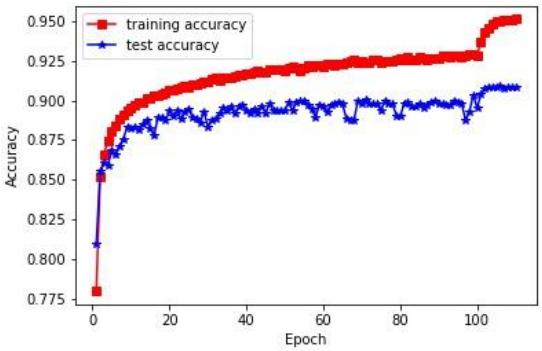
From setting 3, we know that if we didn't introduce the kernel regularization, the accuracy will degrade because of overfitting.

2) Fashion-MNIST Dataset

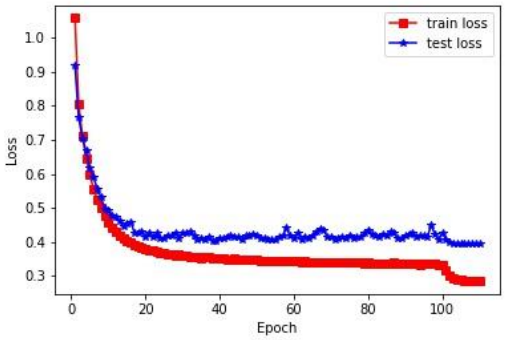
Here the default setting is the batch size 128, For Fashion-MNIST dataset, I used 80 epochs. To achieve a good test accuracy, here one setting (**setting 1**) I used 110 epochs and for last 10 epochs, using the learning rate of 0.5 * original learning rate. I used SGD optimizers with momentum=0.9 and used Nesterov, the only difference is learning rate.

Table2. Test Accuracy on Fashion MNIST Dataset

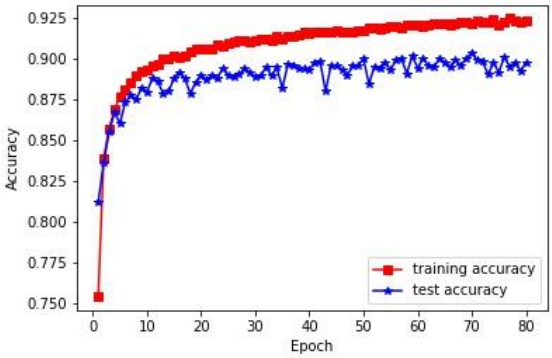
Experiments	Learning rate	Kernel Initialization	Kernel Regularization	Accuracy (mean,std,max)
Setting 1	schedule	Kaiming He Initialization	L2(0.001)	(0.9103,0.001422,0.9123)
Setting 2	0.01	Kaiming He Initialization	L2(0.001)	(0.8989,0.00187,0.9013)
Setting 3	0.01	Kaiming He Initialization	None	(0.8896,0.00320,0.8927)
Setting 4	0.01	Random Initialization	L2(0.001)	(0.8968,0.00406, 0.9021)
Setting 5	0.05	Kaiming He Initialization	L2(0.001)	(0.8875, 0.00620,0.8939)



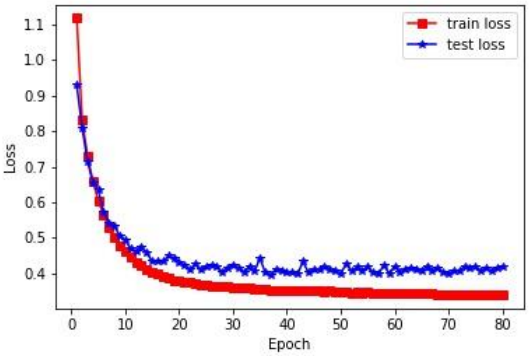
Setting 1



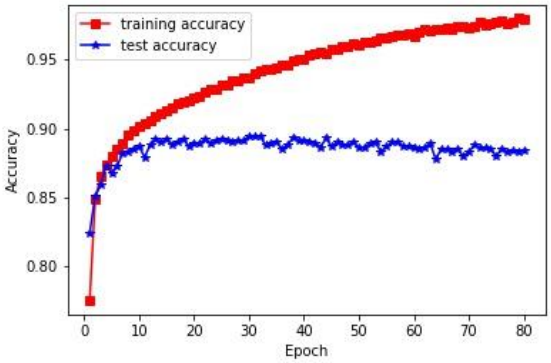
Setting 1



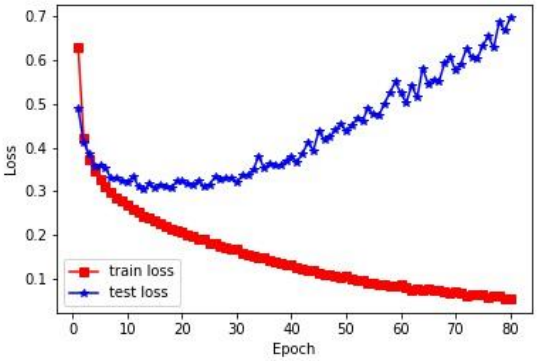
Setting 2



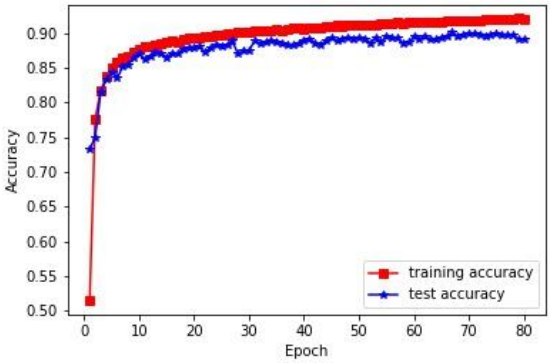
Setting 2

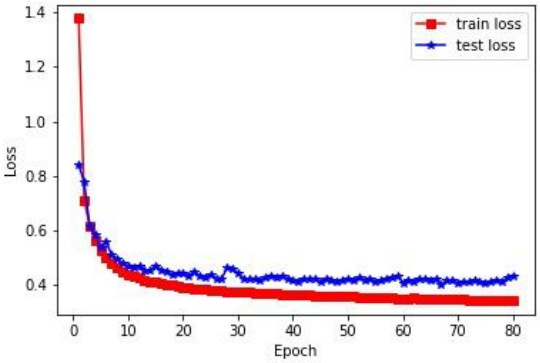


Setting 3

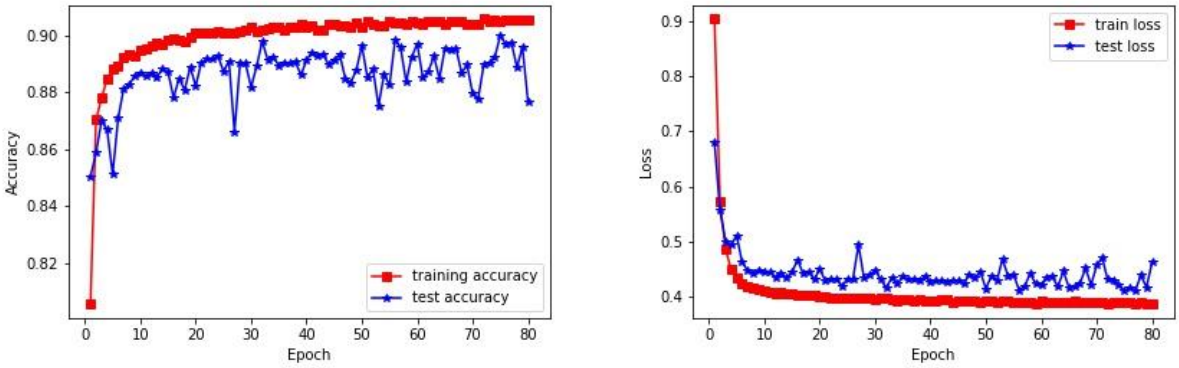


Setting 3





Setting 4 **Setting 4**



Setting 5 **Setting 5**

Figure 6. Performance Curve on Fashion-MNIST Dataset

Discussion

From the setting 3, we can find that it will contribute to overfitting if we didn't use kernel regularization here. From the setting 5, we can find that high learning rate will result in oscillation and highly zigzag and fluctuation. The best accuracy reaches by change the learning rate by multiplying 0.5, when the learning rate change, the accuracy as well as loss will jump to the high (low for loss).

3) CIFAR10 Dataset

CIFAR10 is a dataset with higher dimensions and channels images than MNIST or Fashion-MNIST. Here I used batch size 128 and learning rate 0.001. I use 150 epochs as the total epochs.

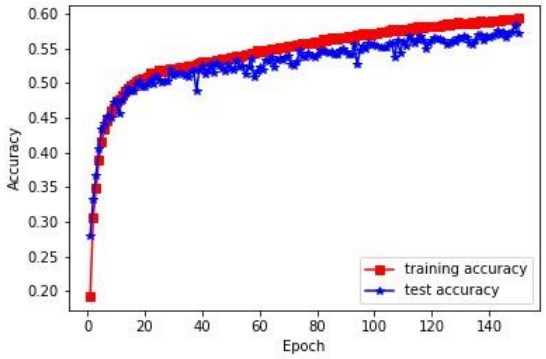
Here, I introduce two different set of learning rate change schedule. The first is schedule1 in setting 3, when the epochs belong to (120,140), use the 0.5 * original learning rate. When the epochs belong to (140,150), use 0.1 * original learning rate.

The second schedule2 in setting 2 is when epochs belong to (140,150), use 0.5 * original learning rate.

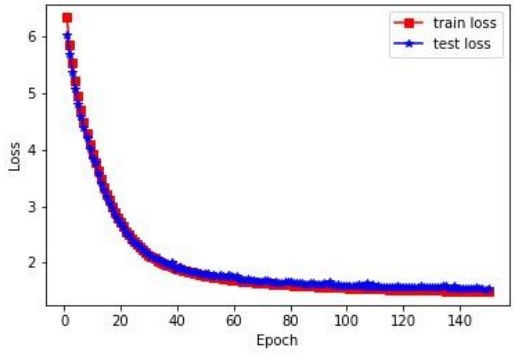
I used SGD optimizers with momentum=0.9 and used Nesterov, the only difference is learning rate.

Table3. Test Accuracy on CIFAR10 Dataset

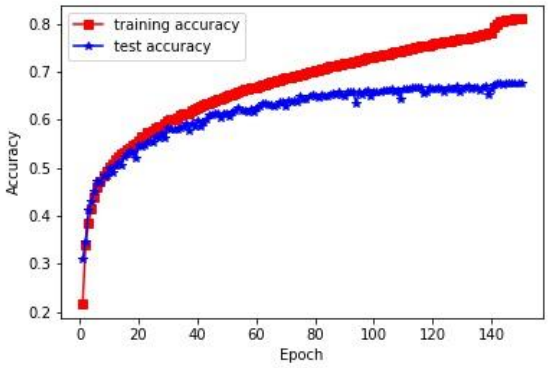
Experiments	Learning rate	Kernel Initialization	Kernel Regularization	Accuracy (mean,std,max)
Setting 1	0.001	Kaiming He Initialization	L1(0.001)	(0.5862,0.0111,0.6064)
Setting 2	Schedule 2	Kaiming He Initialization	L2(0.001)	(0.6536,0.0147,0.6759)
Setting 3	Schedule 1	Kaiming He Initialization	L2(0.001)	(0.6455,0.0131,0.6672)
Setting 4	0.001	Kaiming He Initialization	L2(0.001)	(0.6375,0.00663,0.6439)
Setting 5	0.001	Kaiming He Initialization	None	(0.6165,0.0141,0.6401)



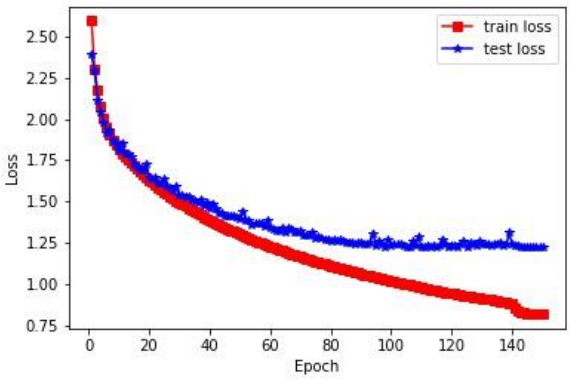
Setting 1



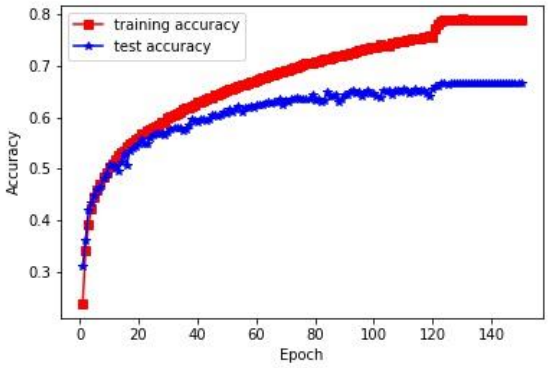
Setting 1



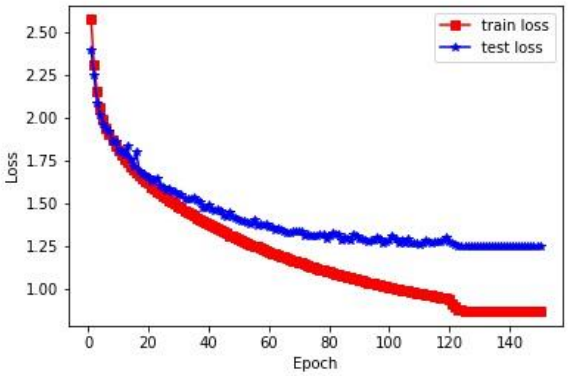
Setting 2



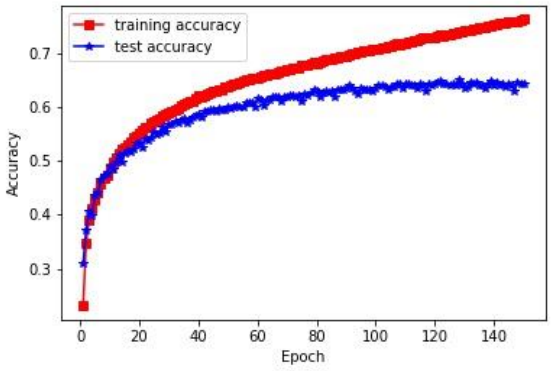
Setting 2



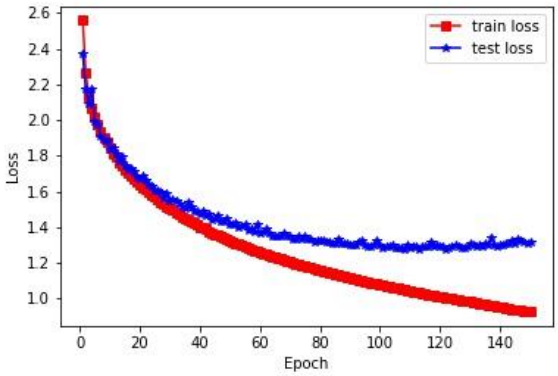
Setting 3



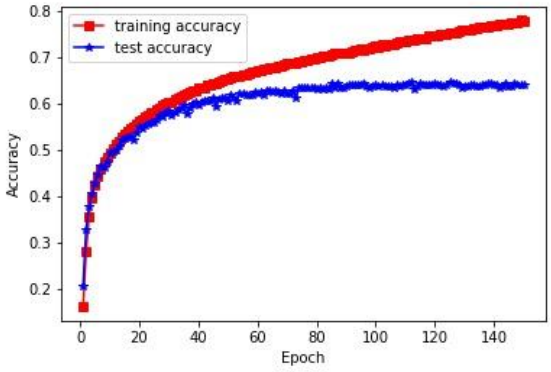
Setting 3



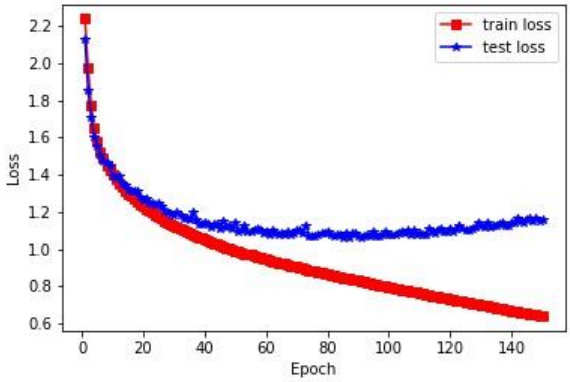
Setting 4



Setting 4



Setting 5



Setting 5

Figure 7. Performance Curve on CIFAR10 Dataset

From the setting 1, l1 kernel regularization will degrade the performance, the training accuracy is near the test accuracy. Reducing the learning rate during training properly can effectively improve the accuracy.

Discussion

The best performance on different datasets reaches different of accuracy. For MNIST Dataset, the best accuracy is around 0.9901, owing to the high accuracy, the standard deviation of the accuracy around five different trails are small. For the Fashion-MNIST dataset, the best performance 0.9103. For the CIFAR10, the best accuracy is lowest, 0.6536 and the standard deviation is highest among all experiment because the higher variance of accuracy.

The main dataset different is owing to the data variance. Owing to the dataset contains different knowledge in the high manifold, the test accuracies among different datasets are the different.

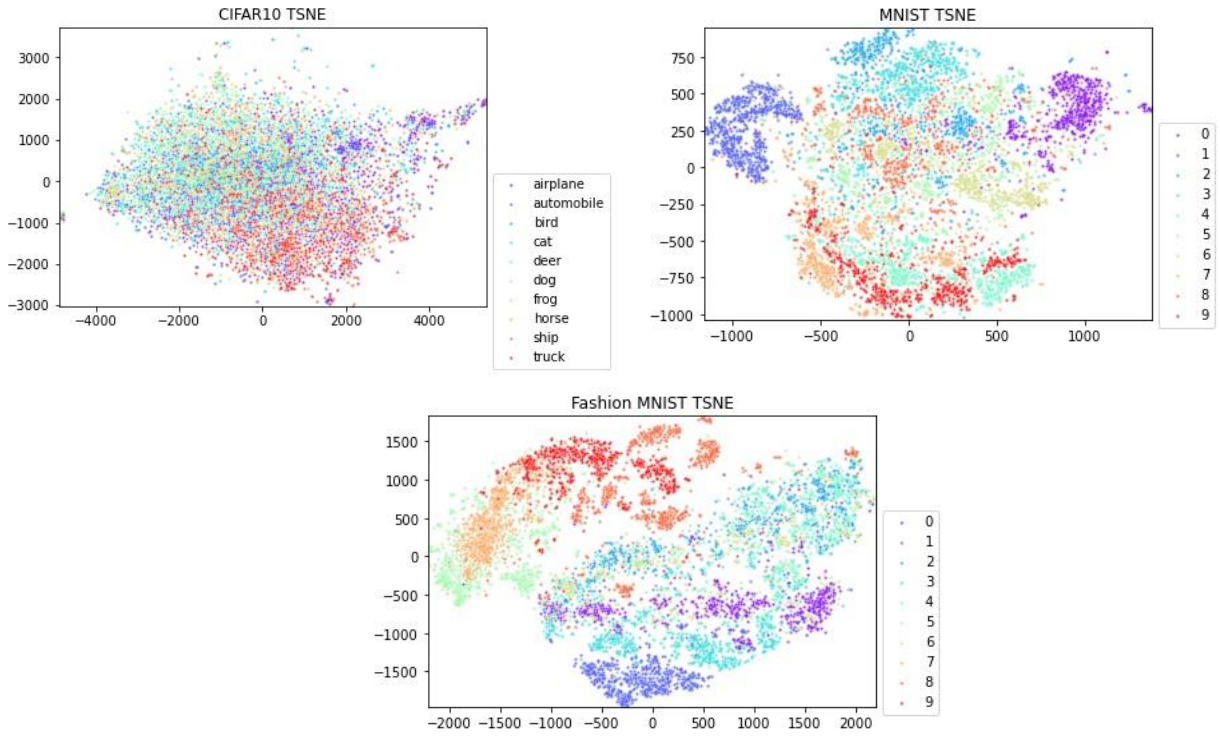


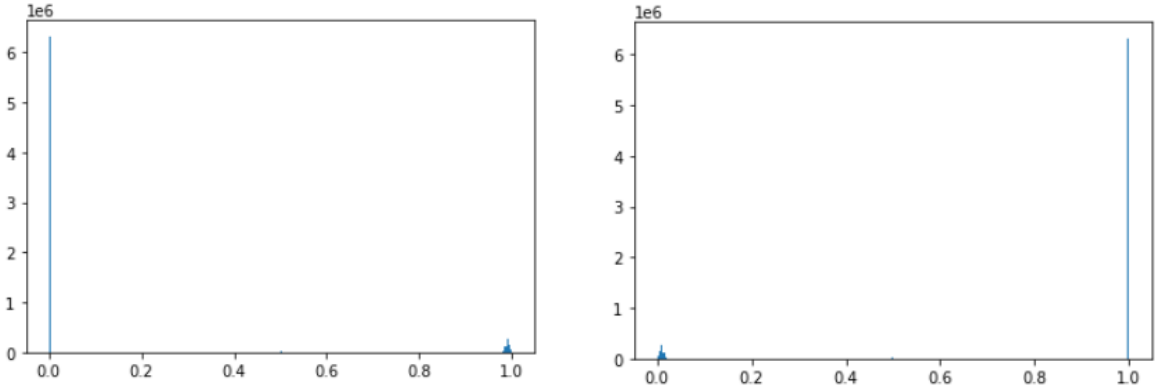
Figure 8. TSNE PLOT of three datasets

From the image above, we can find that CIFAR10 is the most undisguishable dataset. Hence, the performance of the dataset is lower is reasonable.

C) Apply network on negative dataset

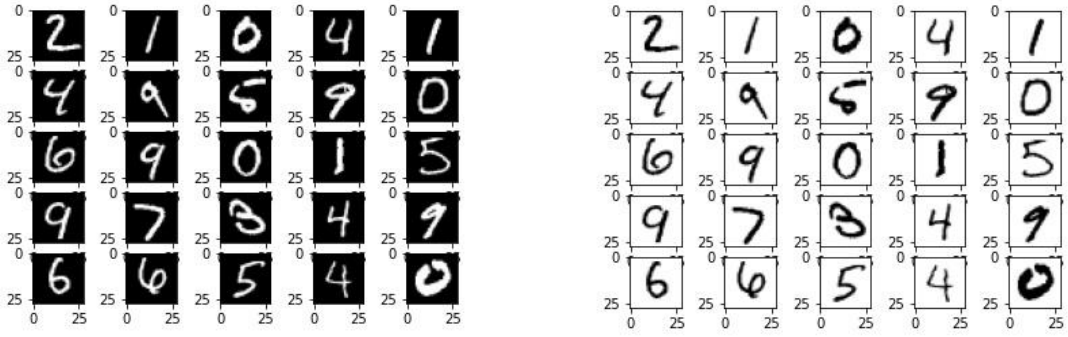
I firstly rescale the dataset from 0-255 to 0-1 by dividing it by 255 as the original training set and original test set. Then broadcast the dataset by 1-dataset to get the negative dataset.

Here I calculate the mean and std of the dataset as statistics. For the original train set, the mean is 0.1307 and the std is 0.3081. For the inverse train set, the mean is 0.8693 and the std is 0.3081. The mean of the inverse dataset is 1-mean of the dataset and the standard deviation doesn't change. For the test set, the original test set mean is 0.1325 and the standard deviation is 0.3104. For the inverse test set, the mean is 0.8675, which is 1-0.1325 and the standard deviation is also 0.3104.



Original test set Inverse test set

Figure 9. Histogram of MNIST test set



Original test set Inverse test set

Figure 10. Visualization of Sample MNIST Dataset

If we use the original training set and inverse test set for validation.

Here my hyperparameter setting is learning rate 0.01, l2 norm, batch size 128 and training epochs 100, we finally get the accuracy tuple (mean, std, max) is (0.44176, 0.063706905, 0.5674). The five experiment results are 39.29%, 40.60%, 42.11% 42.14 and 56.74%. The last run accuracy 56.74% is too high from others, which may be not convincing.

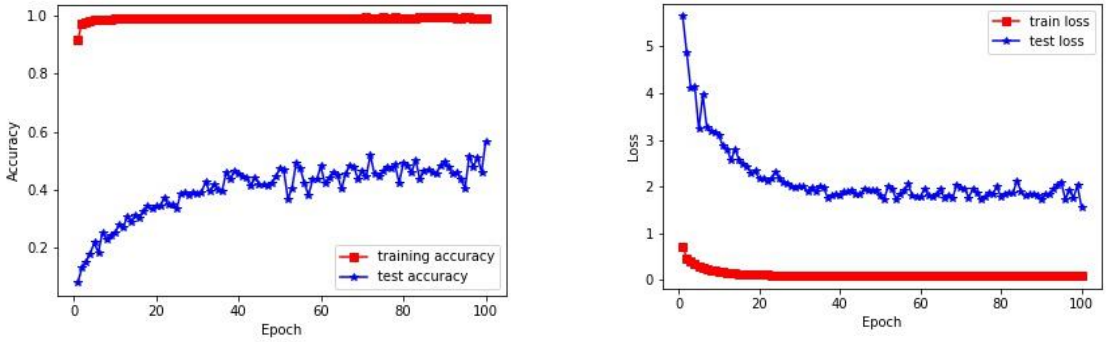


Figure 11. Performance Curve of Inverse MNIST test set

Discussion

We can only get around 40% accuracy for the test set. It is because that the data (pixel value distribution) is totally different according to Figure 8. The deep learning models don't have the high generalization ability. If we want to get the higher accuracy on the test accuracy, we should use the let the network see the same type of data during the training approach.

Design a new network

To Design a new network, we should use the method that let the network to see both training sets and test set during the training process. Here I would like to find the function that let the inverse dataset and original dataset get the same value,

$$F(x) = F(1-x)$$

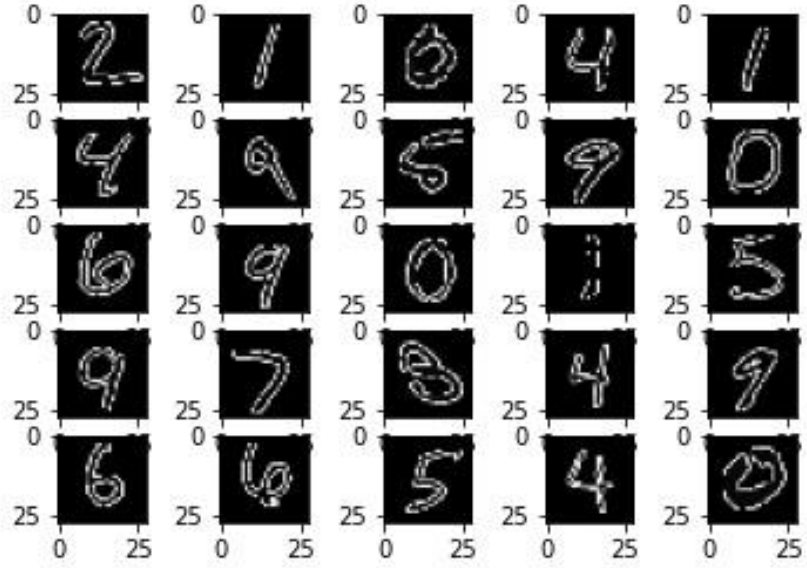


Figure 12 Visualization of Transformed MNIST Dataset

To be specified, I didn't preprocess any dataset. The only thing I do is add a transformation layer in the beginning of the layer. Figure 12 just shows the function of that layer.

Hence, here I used function, $F(x) = x \times (1-x)$ and added the function in the beginning of LENET5 and get an end-to-end pipeline. The method here can work as data augmentation. We can only use the original trainset and use inverse test set as test set. After training we use both original and inverse test set and get the same test accuracy. The test accuracy tuple (Mean,std,Max) is (0.979, 0.00163, 0.982) for five experiments.

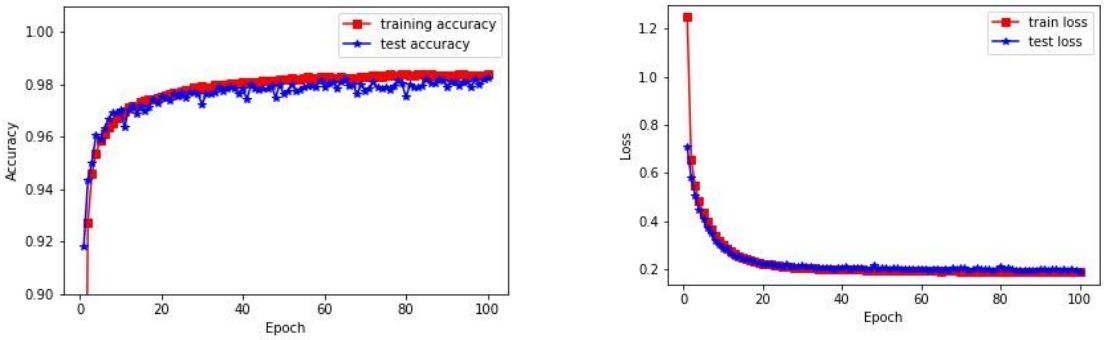


Figure 13. Performance Curve of Inverse MNIST test set

Discussion

Disadvantage:

Here I cascade a data transformation method to the deep learning network. However, this method only works when we have the original and inverse data set. If we use other transformed NMIST dataset we should find another transformation to connect with LENET5. However, it doesn't always make it.

Advantages:

Compared with other possible methods, this method didn't change the training set and original network architecture but only add a linear transfer function F(x). This method just like data augmentation but we don't need to do any other data augmentation procedure. For other methods, you need to consider what ratio of original and negative data in the training data set. However, here we don't have any limitations of what training data is. Even all training data is from inverse data set, we can also get the good performance on positive (original) dataset.