

## **Problem 1: CNN Training on LeNet-5**

### **I. Abstract and Motivation**

LeNet-5 is a convolutional neural network that can largely automate classification tasks like handwritten digit image classification. In the subject of computer vision, Convolutional neural network is one of the foundations of deep learning. Before Convolutional neural network, there is a series of steps for classification tasks such as feature extraction of predefined features, and machine learning based model for classification. Convolutional neural network enables feature extraction and classification automatically by training weight parameters. Usually, a convolutional neural network architecture consists of the hidden convolution layer, pooling layer, activation function, fully connected layer.

### **II. Approach and Procedures**

We construct a LeNet-5 neural network model to perform classification tasks given 10 class labels on MNIST, Fashion-MNIST, and CIFAR datasets by using Pytorch framework. Because Pytorch is one of deep learning frameworks that can efficiently construct convolutional neural network model.

LeNet-5 is a 7 layers neural network architecture including 2 hidden convolutional layers, 2 max pooling layers, 3 fully connected layers. Given the 32 by 32 input image, the first convolutional layer with 6 filter channels receives the input signals and apply a 5 by 5 input receptive field to obtain the output signal maps of 6 filter channels. Then, output signal maps are transformed to nonlinear values by ReLU activation function. A max pooling layer is applied to convolute the output signal maps by extracting max value within a 2 by 2 window. The output from max pooling layer contains the dominant features of the output signal maps from the previous convolutional layer. Another convolutional layer with 16 filter channels is applied to the output from max pooling layer with a 5 by 5 input receptive field. Then, a ReLU activation function is applied to the output signal maps of 16 filter channels to obtain nonlinear responses. A second max pooling layer with a 2 by 2 window is applied to the output signal maps to extract the max value within the window. The feature responses are extracted from two hidden layers with two max pooling layers. Then, the feature responses are classified to 120-D vector by the first

fully connected layer of 120 filters with ReLU activation function. The second fully connected layer with 84 filters takes the 120-D vector and transforms to an 84-D vector after ReLU activation. Finally, the third fully connected layer with 10 filters takes the 84-D vector and transforms to a 10-D vector. The 10-D vector is normalized to range of 0 to 1 and can represent the probability of 10 classes by applying SoftMax function.

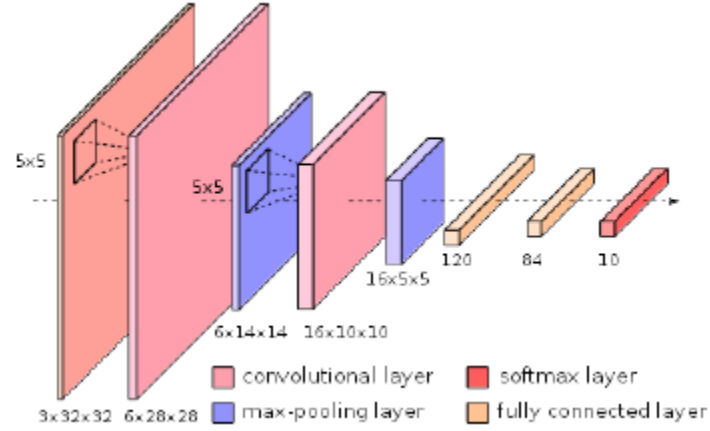


Figure 1: The structure of LeNet-5 model

To achieve better training and generalization performance, we can fine tune hyper parameters in the network, including filter weight initialization, learning rate, momentum, weight decay. In our case, we apply MNIST, Fashion-MNIST, and CIFAR-10 datasets to the LeNet-5 model separately to evaluate the classification performances on these three datasets. Due to the randomness of CNN performance, we apply the same set of hyperparameters for 5 runs and use the average to represent the model performance. By applying different parameter settings, we can obtain the best generalization performance for the given dataset.

The MNIST dataset contains 60000 grayscale training images with size of 28 by 28 and 10000 grayscale testing images with size of 28 by 28. Similarly, the Fashion-MNIST dataset contains 60000 grayscale training images with size of 28 by 28 and 10000 grayscale testing images with size of 28 by 28. The CIFAR-10 dataset contains 50000 colored training images with size of 32 by 32 and 10000 colored testing images with size of 32 by 32.

Before applying the dataset to LeNet-5 model, we normalize the whole dataset to zero mean distribution to have control of gradient descent and weight updates ranges for better performance improvement. In addition, since the LeNet-5 model takes the input image size of 32 by 32, we can apply padding to match the input image size of LeNet-5 model for the MNIST images and Fashion MNIST images.

### III. Experimental Results

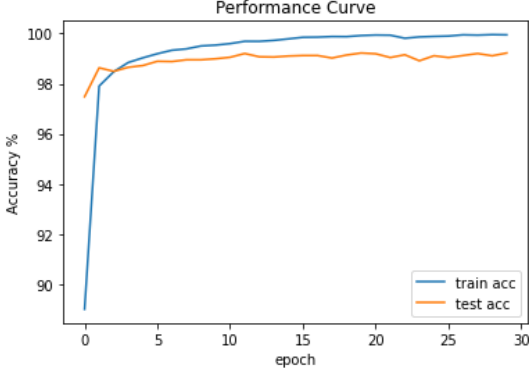
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.001 momentum=0.9 weight_decay=1e-5 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Accuracy %</p> <p>epoch</p> <p>train acc</p> <p>test acc</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 99.998% Mean train accuracy among 5 runs: 99.922% Std train accuracy among 5 runs: 0.06394 Best test accuracy among 5 runs: 99.22% Mean test accuracy among 5 runs: 99.094% Std test accuracy among 5 runs: 0.14263

Table 1: First parameter setting for LeNet-5 model on MNIST dataset.

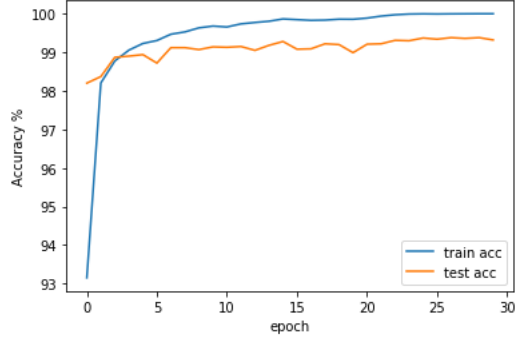
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.01 momentum=0.5 weight_decay=1e-4 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Accuracy %</p> <p>epoch</p> <p>train acc</p> <p>test acc</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 100.0% Mean train accuracy among 5 runs: 99.996% Std train accuracy among 5 runs: 0.002449 Best test accuracy among 5 runs: 99.350% Mean test accuracy among 5 runs: 99.31% Std test accuracy among 5 runs: 0.023664

Table 2: Second parameter setting for LeNet-5 model on MNIST dataset.

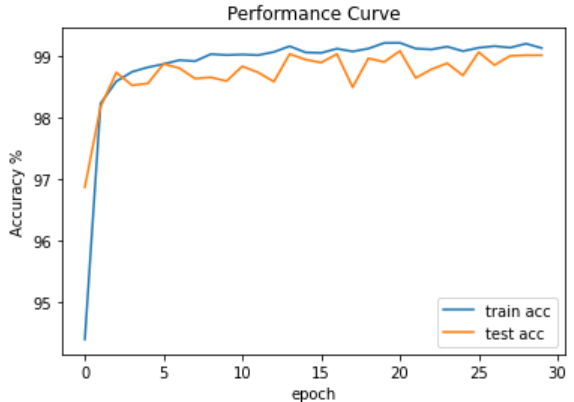
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.005 momentum=0.9 weight_decay=1e-3 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 99.146% Mean train accuracy among 5 runs: 99.121% Std train accuracy among 5 runs: 0.02284 Best test accuracy among 5 runs: 99.02% Mean test accuracy among 5 runs: 98.955% Std test accuracy among 5 runs: 0.08212

Table 3: Third parameter setting for LeNet-5 model on MNIST dataset.

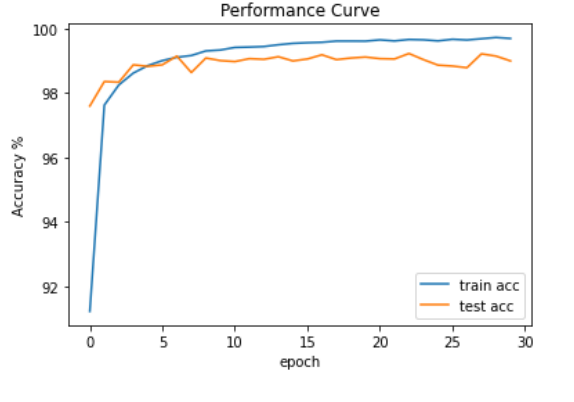
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.005 momentum=0.5 weight_decay=1e-3 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 99.703% Mean train accuracy among 5 runs: 99.682% Std train accuracy among 5 runs: 0.01820 Best test accuracy among 5 runs: 99.07005% Mean test accuracy among 5 runs: 99.012% Std test accuracy among 5 runs: 0.03487

Table4: Fourth parameter setting for LeNet-5 model on MNIST dataset.

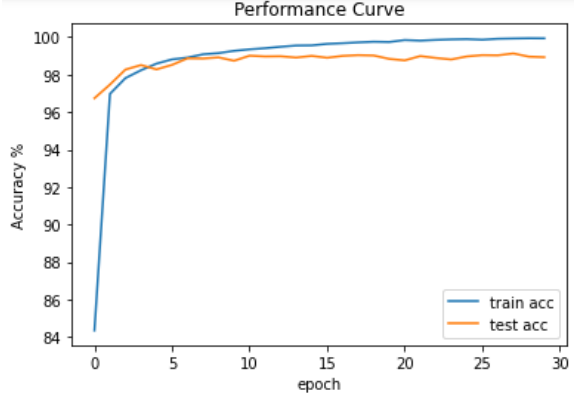
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.0005 momentum=0.9 weight_decay=1e-5 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Accuracy %</p> <p>epoch</p> <p>train acc</p> <p>test acc</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 99.978% Mean train accuracy among 5 runs: 99.955% Std train accuracy among 5 runs: 0.01711 Best test accuracy among 5 runs: 99.229% Mean test accuracy among 5 runs: 99.078% Std test accuracy among 5 runs: 0.09620

Table 5: Fifth parameter setting for LeNet-5 model on MNIST dataset.

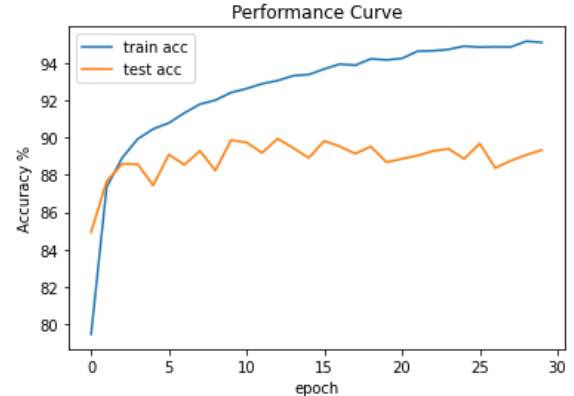
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=16 test_batch_size=1000 epoch=30 learning_rate=0.005 momentum=0.9 weight_decay=1e-5 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Accuracy %</p> <p>epoch</p> <p>train acc</p> <p>test acc</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 95.342% Mean train accuracy among 5 runs: 94.9833% Std train accuracy among 5 runs: 0.28924 Best test accuracy among 5 runs: 89.37% Mean test accuracy among 5 runs: 89.095% Std test accuracy among 5 runs: 0.47902

Table 6: First parameter setting for LeNet-5 model on Fashion-MNIST dataset.

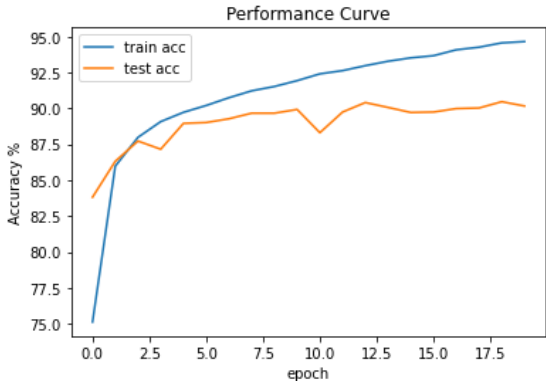
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=20 learning_rate=0.001 momentum=0.9 weight_decay=1e-5 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Accuracy %</p> <p>epoch</p> <p>train acc</p> <p>test acc</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 94.646% Mean train accuracy among 5 runs: 94.589% Std train accuracy among 5 runs: 0.049109 Best test accuracy among 5 runs: 90.16% Mean test accuracy among 5 runs: 90.011% Std test accuracy among 5 runs: 0.18605

Table 7: Second parameter setting for LeNet-5 model on Fashion-MNIST dataset.

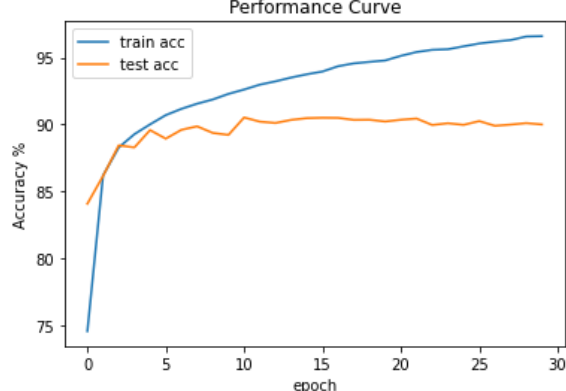
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.001 momentum=0.9 weight_decay=1e-5 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Accuracy %</p> <p>epoch</p> <p>train acc</p> <p>test acc</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 96.592% Mean train accuracy among 5 runs: 96.387% Std train accuracy among 5 runs: 0.20648 Best test accuracy among 5 runs: 90.8% Mean test accuracy among 5 runs: 90.021% Std test accuracy among 5 runs: 0.52028

Table 8: Third parameter setting for LeNet-5 model on Fashion-MNIST dataset.

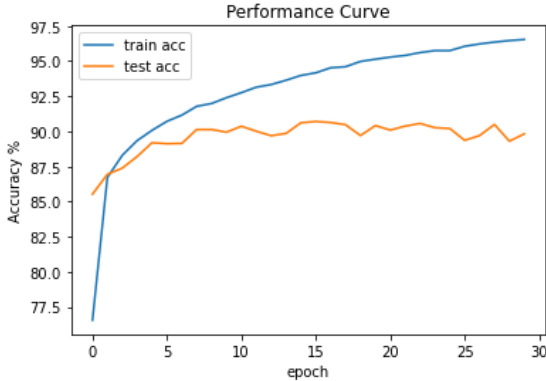
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=16 test_batch_size=1000 epoch=30 learning_rate=0.01 momentum=0.5 weight_decay=1e-6 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Y-axis: Accuracy % (77.5 to 97.5) X-axis: epoch (0 to 30)</p> <p>Legend: train acc (blue), test acc (orange)</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 96.556% Mean train accuracy among 5 runs: 96.362% Std train accuracy among 5 runs: 0.23722 Best test accuracy among 5 runs: 90.34% Mean test accuracy among 5 runs: 89.918% Std test accuracy among 5 runs: 0.35521

Table 9: Fourth parameter setting for LeNet-5 model on Fashion-MNIST dataset.

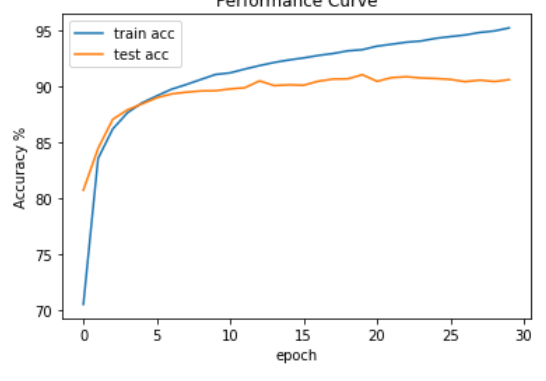
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.0005 momentum=0.9 weight_decay=1e-6 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Y-axis: Accuracy % (70 to 95) X-axis: epoch (0 to 30)</p> <p>Legend: train acc (blue), test acc (orange)</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 95.512% Mean train accuracy among 5 runs: 95.162% Std train accuracy among 5 runs: 0.27929 Best test accuracy among 5 runs: 91.06% Mean test accuracy among 5 runs: 90.436% Std test accuracy among 5 runs: 0.35869

Table 10: Fifth parameter setting for LeNet-5 model on Fashion-MNIST dataset.

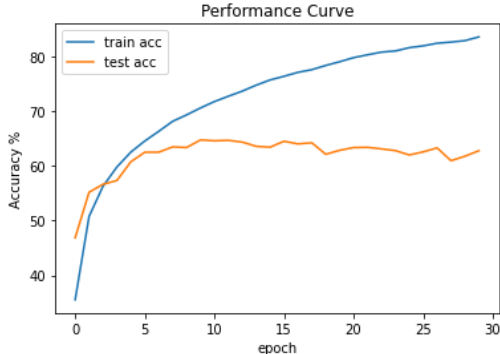
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.001 momentum=0.9 weight_decay=1e-5 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 83.668% Mean train accuracy among 5 runs: 83.235% Std train accuracy among 5 runs: 0.456018 Best test accuracy among 5 runs: 62.78% Mean test accuracy among 5 runs: 61.682% Std test accuracy among 5 runs: 0.71300

Table 11: First parameter setting for LeNet-5 model on CIFAR-10 dataset.

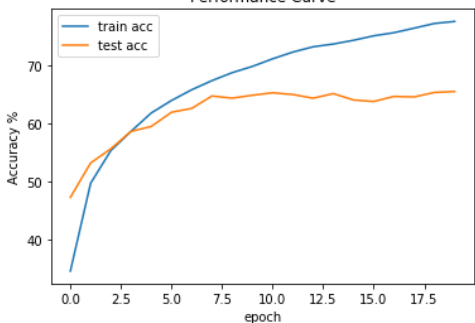
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=20 learning_rate=0.001 momentum=0.9 weight_decay=1e-3 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 77.794% Mean train accuracy among 5 runs: 77.407% Std train accuracy among 5 runs: 0.51768 Best test accuracy among 5 runs: 65.57% Mean test accuracy among 5 runs: 64.68% Std test accuracy among 5 runs: 0.86625

Table 12: Second parameter setting for LeNet-5 model on CIFAR-10 dataset.



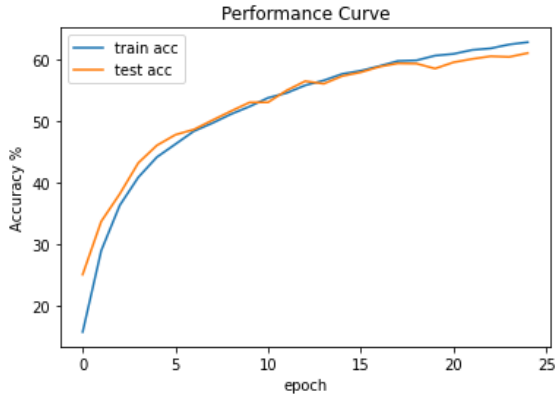
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=16 test_batch_size=1000 epoch=25 learning_rate=0.001 momentum=0.5 weight_decay=1e-2 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Y-axis: Accuracy % (20 to 60)</p> <p>X-axis: epoch (0 to 25)</p> <p>Legend: train acc (blue), test acc (orange)</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 62.822% Mean train accuracy among 5 runs: 61.073% Std train accuracy among 5 runs: 1.08470 Best test accuracy among 5 runs: 61.06% Mean test accuracy among 5 runs: 59.530% Std test accuracy among 5 runs: 0.95699

Table 13: Third parameter setting for LeNet-5 model on CIFAR-10 dataset.

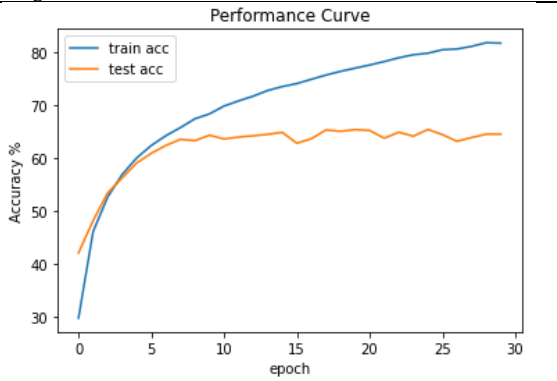
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.0005 momentum=0.9 weight_decay=1e-3 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Y-axis: Accuracy % (30 to 80)</p> <p>X-axis: epoch (0 to 30)</p> <p>Legend: train acc (blue), test acc (orange)</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 82.208% Mean train accuracy among 5 runs: 81.653% Std train accuracy among 5 runs: 0.360187 Best test accuracy among 5 runs: 65.18% Mean test accuracy among 5 runs: 64.486% Std test accuracy among 5 runs: 0.41209

Table 14: Fourth parameter setting for LeNet-5 model on CIFAR-10 dataset.

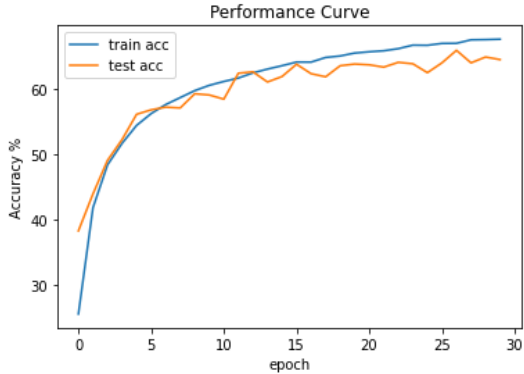
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.0005 momentum=0.9 weight_decay=1e-2 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Accuracy %</p> <p>epoch</p> <p>train acc</p> <p>test acc</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 68.62% Mean train accuracy among 5 runs: 67.980% Std train accuracy among 5 runs: 0.488375 Best test accuracy among 5 runs: 66.24% Mean test accuracy among 5 runs: 65.198% Std test accuracy among 5 runs: 0.75055

Table 15: Fifth parameter setting for LeNet-5 model on CIFAR-10 dataset.

Best parameter setting to produce the highest accuracy on MNIST test Dataset.

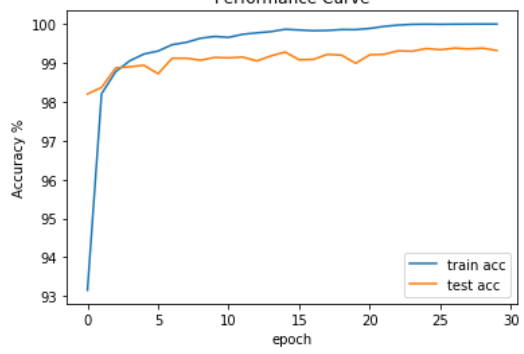
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.01 momentum=0.5 weight_decay=1e-4 weight initialization=default uniform distribution
Performance curve	 <p>Performance Curve</p> <p>Accuracy %</p> <p>epoch</p> <p>train acc</p> <p>test acc</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 100.0% Mean train accuracy among 5 runs: 99.996% Std train accuracy among 5 runs: 0.002449 Best test accuracy among 5 runs: 99.350% Mean test accuracy among 5 runs: 99.31% Std test accuracy among 5 runs: 0.023664

Table 16: Best parameter setting for LeNet-5 model on MNSIT dataset

Best parameter setting to produce the highest accuracy on Fashion-MNIST test Dataset.

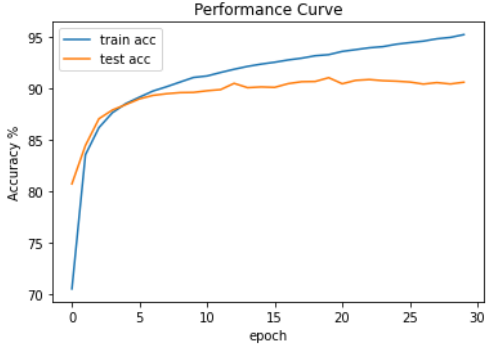
Parameter setting	<p>Optimizer: SGD  Loss function: CrossEntropyLoss  train_batch_size=10  test_batch_size=1000  epoch=30  learning_rate=0.0005  momentum=0.9  weight_decay=1e-6  weight initialization=default uniform distribution</p>
Performance curve	 <p>Performance Curve</p>
Performance results given 5 runs	<p>Best train accuracy among 5 runs: 95.512%  Mean train accuracy among 5 runs: 95.162%  Std train accuracy among 5 runs: 0.27929  Best test accuracy among 5 runs: 91.06%  Mean test accuracy among 5 runs: 90.436%  Std test accuracy among 5 runs: 0.35869</p>

Table 17: Best parameter setting for LeNet-5 model on Fashion-MNIST dataset

Best parameter setting to produce the highest accuracy on CIFAR-10 test Dataset.

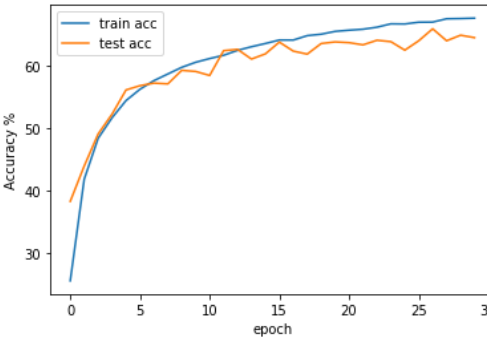
Parameter setting	<p>Optimizer: SGD  Loss function: CrossEntropyLoss  train_batch_size=10  test_batch_size=1000  epoch=30  learning_rate=0.0005  momentum=0.9  weight_decay=1e-2  weight initialization=default uniform distribution</p>
Performance curve	 <p>Performance Curve</p>
Performance results given 5 runs	<p>Best train accuracy among 5 runs: 68.62%  Mean train accuracy among 5 runs: 67.980%  Std train accuracy among 5 runs: 0.48837  Best test accuracy among 5 runs: 66.24%  Mean test accuracy among 5 runs: 65.198%  Std test accuracy among 5 runs: 0.75055</p>

Table 18: Best parameter setting for LeNet-5 model on CIFAR-10 dataset

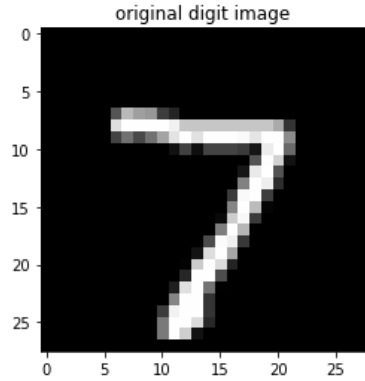
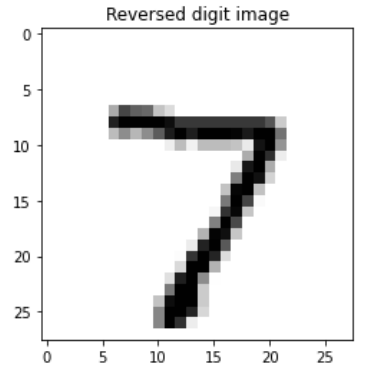
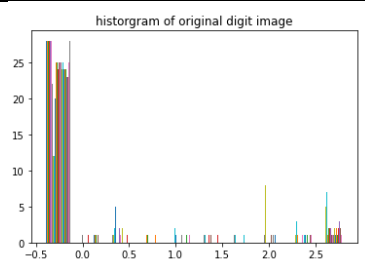
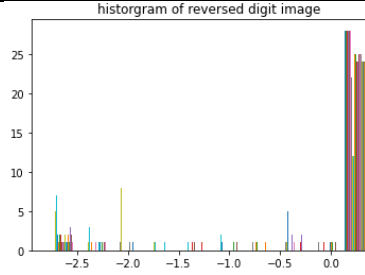
	Original digit image	Reversed digit image
Example		
Histogram of the given sample		
Statistics	mean of original test data: 0.005889658 std of original test data: 1.0077257 max of original test data: 2.8214867 min of original test data: -0.42421296	mean of reversed test data: -0.005889658 std of reversed test data: 1.0077257 max of reversed test data: 0.42421296 min of reversed test data: -2.8214867

Table 19: Statistics of test dataset

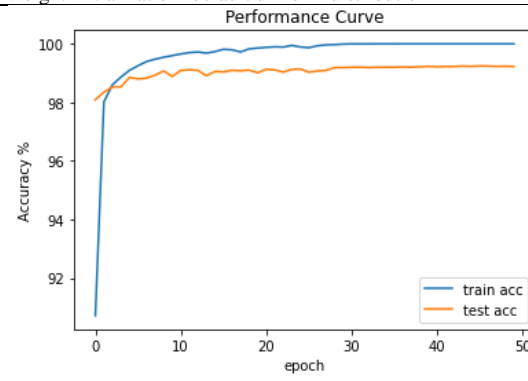
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=50 learning_rate=0.001 momentum=0.9 weight_decay=1e-4 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy: 100.0% Best test accuracy: 99.225%

Table 20: LeNet-5 model for both original and negative digit images

## IV. Discussion

Convolutional Neuron Network consists of multiple components. The main components include Convolutional layer, activation function, pooling layer, SoftMax function, and fully connected layer.

Fully connected layer is mainly to classify feature vector from feature maps after convolution through hidden layers. Fully connected layer projects all features into a flattened feature vector for classification of features from previous layers after feature extraction. It eliminates the location factor from feature maps. Since fully connected layer is linear transformation, it is also necessary to apply nonlinear activation function for classification.

Convolutional layer is a hidden layer in the neural network with number of filter channels as feature maps. In convolutional layer, we can obtain a feature map by sliding the input signal with a filter of certain size. Each filter comes with learnable weight parameters. A larger size of input receptive field indicates a more abstract feature can be detected. The convolutional layer is designed to parallelly compute numbers of features across filters for a given layer. Each filter channel extracts a very specific feature representation from the input data. The feature map is a representation of features detected from the input given all current weights. We apply activation function for each feature map to obtain nonlinear feature. The network is able to learn complex feature patterns due to the nonlinearity of the activation function.

Max pooling layer is to choose the max value from the output signal maps given a window of small size for each feature by raster parsing. Max pooling layer extracts dominant features of feature maps. Since the features in feature maps are sensitive to position in the image, max pooling layer reduces feature map size and keeps dominant features by down sampling the feature maps. The output from max pooling is more robust to feature position change. In the LeNet-5 model, a window with size of 2 by 2 slides through the feature maps and down sample the feature maps by take the max value within the window.

Activation function is to apply a non-linear, zero-centered, and differential function to feature maps from output signal to learn complex feature pattern in convolutional layer or Linear layer. Without activation function, the neural network is basically linear operations by sliding through the input with a filter to compute the feature maps in each convolutional layer. However, most feature pattern from input usually follows nonlinear pattern. Linear operation cannot detect feature with nonlinear patterns. The extracted feature is more accurate by adding non-linear activation function to each filter. In addition, activation function largely reduces computation cost and limits the gradient in a restricted range. In addition, activation function prevents the gradient from infinitely large or small values during gradient descent. Sigmoid is one of the popular activation functions. Because it has properties

like smooth gradient, limited value range. By applying Sigmoid function to output signal, the output value range is limited between 0 to 1. The gradient around 0 or 1 is very small. The gradient between 0 and 1 is larger and it works well for weight update to learn features. However, sigmoid function may have vanishing gradient problem as the input is near 0 or 1. There are also other popular activation functions for different tasks like Tanh function, ReLU function and its derivations like Leaky ReLU function. Here are some popular activation functions.

$$\text{Sigmoid function: } S(x) = \frac{1}{1 + e^{-x}}$$

$$\text{ReLU function: } Y(x) = \max(0, x)$$

Softmax function is to project values of a 1-D multi-class vector from last fully connected layer to a 1-D multi-class vector from a range 0 to 1. Larger value before applying Softmax function will be closer to 1 and vice versa. The value from Softmax function can be viewed as feature probability for classification task. And the sum of all feature probability is 1. Therefore, Softmax activation function fits well for classification tasks among a multi-class vector. Softmax Formula:

$$\text{Softmax function: } y(z)_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$$

$z$ : input vector

$K$ : number classes in the classifier

Overfitting issue is that the training data fits the learning model too well to negatively affect the generalization performance. The trained model is too complex and has high prediction accuracy on training dataset. But the trained model has low generalization performance by feeding test dataset, which is unknown data to the model. There are several techniques to avoid overfitting problem in CNN training. First, we can add dropout layers in the network to randomly deactivate some portions of features so that the model complexity can be reduced, and the training dataset will not fit too well in the learning model. Second, we can add a regularization term in cost function to penalize large value parameters to reduce model complexity. The penalty is larger if the parameter value is larger during backpropagation. In addition, we can apply K-fold cross-validation method to train the model to reduce model complexity. The training data is split into K subsets and we can train the model K times using K-1 number of subsets in different combinations and using 1 subset as validation set.

ReLU function is in a form of  $y = \max(0, x)$ , shown in Figure 1 below. ReLU function avoids vanishing gradient problem in Sigmoid function. Because the gradient of ReLU is 1 for input greater than 0 and 0 for input less than 0. However, ReLU function may suffer dead neuron problem. Because the neuron is deactivated if the output from hidden layer is less than 0. The gradient of these neuron will be 0 and the weight will not be updated during backpropagation. These neurons may never be activated again and cannot contribute to correction of weights for some features.

LeakyReLU function is a form of  $y = \max(0.01*x, x)$ , which is derived from ReLU function. When the output signal is greater than 0, LeakyReLU function is the same as ReLU function. However, when the output signal is less than 0, LeakyReLU allows a small, constant gradient change. Therefore, unlike ReLU deactivate neurons if the hidden layer output signal is less than 0, LeakyReLU avoids dead neurons problem by having a constant gradient 0.01. LeakyReLU activation function may result faster convergence of zero cost, because its mean is closer to zero than ReLU function.

ELU function is a form of  $y = (a*(e^x-1), x)$ .  $a$  is a positive number. ELU function is also the same as ReLU function when output signal is greater than 0. However, ELU has a small exponential term when hidden layer out is less than 0. ELU function can also avoid dead neuron problem since no hidden layer output signal is deactivated. In addition, ELU function alleviates gradient vanishing problem because the gradient is 1 for the positive part. And ELU has smooth gradient for the negative part and tends to converge faster because its mean is closer to zero and take negative hidden layer output signal.

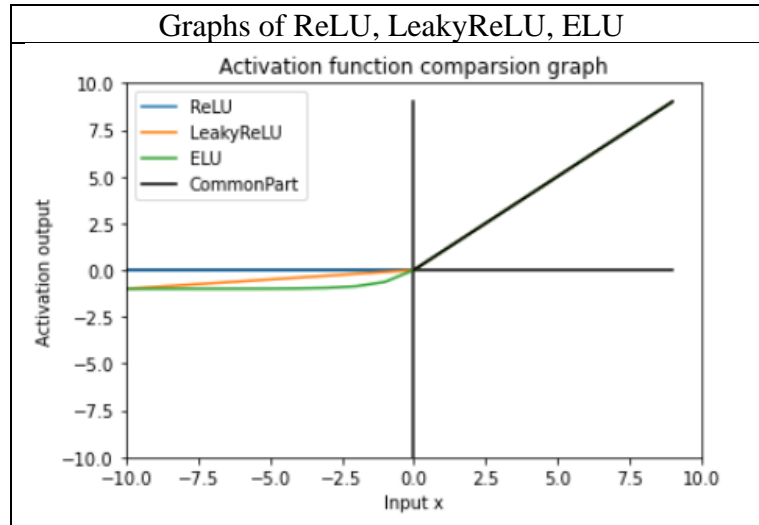


Figure 1: Graphs of ReLU, LeakyReLU, ELU functions

L1Loss function is to compute the mean absolute error, which is the average of the absolute difference between predicted labels and true labels. L1Loss function is robust to datasets with outliers. L1Loss can be used in regression model where the dataset mixes with outliers if outliers can be ignored. Because the loss computes the

absolute difference between the predicted value and target class, the gradient descent for weight updates is not affected by outliers too much. Therefore, the model will have better generalization performance.

MSELoss function is to compute the mean squared error, which is the average of the squared difference between predicted labels and true labels. MSELoss function punishes larger error for larger difference between predicted label and target class. Therefore, if there are outliers in the dataset, the MSELoss loss produce larger loss and tends to adjust weights in the model due to presence of outliers. MSELoss function is very sensitive to outliers in datasets. Then, MSELoss can be applied to regression problems if outliers need to be considered for more accurate convergence.

BCELoss function is to compute the binary cross entropy between predicted labels and true labels. it is usually used in classification problems. Because BCELoss measures the difference between two variable distributions. The input needs to be transformed to a range of 0 to 1 by sigmoid function because the input of BCELoss must be in a range of 0 to 1. In addition, BCELoss can be used in binary classification and multi-class classification because the different classes are independent from each other.

Given MNIST dataset, Fashion-MNIST dataset, and CIFAR-10 dataset, I apply each dataset to the LeNet-5 neural network to evaluate the classification performance. To preserve similar data distribution of the mini-batch and whole dataset, I set the train batch size to 10 to feed in the network. I set test batch size to 1000 because test dataset does not contribute to weight correction. Momentum is to consider the gradient direction and value from last weight update in addition to the current gradient. A larger momentum value indicates we take more consideration from the previous gradient direction and value and vice versa. Weight decay is a regularization term to penalize weight parameters and restrict the weights in a limited range in case of exploding weights. Therefore, weight decay work well to reduce model complexity and increate generalization performance on testing dataset. To compute the loss and gradient for weight updates during training, I apply SGD Optimizer and CrossEntropyLoss as the loss function for all parameter settings. In addition, I choose the default normal distribution for weight initialization. From the settings below, I adjust number of epochs, learning rate, momentum, weight decay to train the LeNet-5 model with different train dataset and evaluate the performance for train dataset and test dataset.

The MNIST dataset includes 60000 training samples and 10000 testing samples with grayscale 28 by 28 handwritten digit images. There are 10 target classes from 0 to 9. We apply this MNIST data to LeNet-5 neural network to recognize different classes of handwritten digits. Then, we apply different hyper parameter settings to determine the best performance of the network. I apply 5 different settings to compare the network performances on each setting and reach 99% test accuracy as our objective. Due to the randomness of network results, I apply the same parameter



setting for 5 runs and evaluate the mean accuracy of test accuracy as the final test accuracy for the given parameter setting.

For each parameter setting, I apply 30 epochs to better fit the training data to the LeNet-5 model and apply SGD Optimizer and use CrossEntropyLoss as the loss function. For the first parameter setting, in order to avoid overfitting problem to the model during the training process, I apply weight decay of  $1e-5$  in the optimizer. Under this parameter setting, the training data is able to fit the model perfect and stable. By observing the train accuracy curve in the performance graph, we notice the model learns training data very fast before 5 epochs and gradually converges after that. The training accuracy shows stable convergence curve as number of epochs increases. By applying the first parameter setting for 5 runs, I notice the mean train accuracy is near to 100% and the standard deviation is extremely small.

By observing the test accuracy curve in the graph, I notice the trained model shows fairly well classification performance after first epoch of training. After several epochs of weight updates from training, the test accuracy converges to above 99% steadily. By apply test dataset to this trained model for 5 runs, I obtain the best test accuracy at 99.22%, mean test accuracy at 99.09%, and standard deviation at 0.14263 among 5 runs. Given the small standard deviation value among 5 runs, the model shows stable test classification performance, and the mean test accuracy reaches to our objective target as well.

Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.001 momentum=0.9 weight_decay= $1e-5$ weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 99.998% Mean train accuracy among 5 runs: 99.922% Std train accuracy among 5 runs: 0.06394 Best test accuracy among 5 runs: 99.22% Mean test accuracy among 5 runs: 99.094% Std test accuracy among 5 runs: 0.14263

Table 1: First parameter setting for LeNet-5 model on MNIST dataset.

Under the second parameter setting, I increase the learning rate to 0.01, reduce the momentum to 0.5, and increase the weight decay to 1e-4 to evaluate the model with 30 epochs again. The model still performs very well on training dataset. By observing the train accuracy curve in the performance graph, we notice the model shows similar convergence pattern as the first parameter setting. By applying the second parameter setting for 5 runs, I notice the mean train accuracy is perfect, and the smaller standard deviation value shows more robust performance for the second parameter setting among 5 runs.

By observing the test accuracy curve in the graph, I notice the trained model shows very stable convergence as the number of epochs increase. The test accuracy still converges to above 99% after training. By apply test dataset to this trained model for 5 runs, I obtain the best test accuracy at 99.35%, and mean test accuracy at 99.31%, and standard deviation at 0.02366 among 5 runs. The parameter setting shows a stable convergence from the performance curve and a smaller standard deviation in comparison with the first parameter setting. The trained model with this parameter setting outperforms the model with first parameter setting.

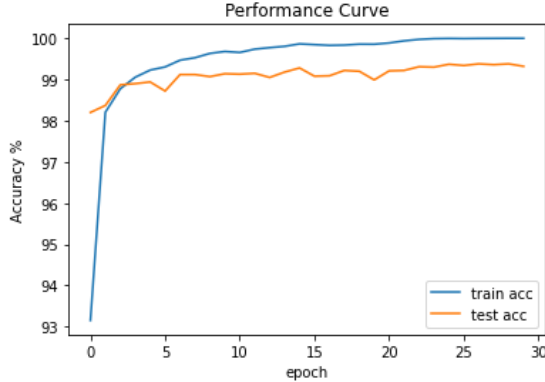
Parameter setting	<p>Optimizer: SGD  Loss function: CrossEntropyLoss  train_batch_size=10  test_batch_size=1000  epoch=30  learning_rate=0.01  momentum=0.5  weight_decay=1e-4  weight initialization=default uniform distribution</p>
Performance curve	 <p>The graph, titled 'Performance Curve', plots Accuracy % on the y-axis (ranging from 93 to 100) against epoch on the x-axis (ranging from 0 to 30). Two lines are shown: 'train acc' (blue) and 'test acc' (orange). The training accuracy starts at approximately 93.2% at epoch 0, rises sharply to about 99.9% by epoch 5, and then continues to rise slowly, reaching 100.0% by epoch 30. The test accuracy starts at approximately 98.2% at epoch 0, rises to about 99.3% by epoch 5, and then fluctuates slightly, ending at approximately 99.35% at epoch 30.</p>
Performance results given 5 runs	<p>Best train accuracy among 5 runs: 100.0%  Mean train accuracy among 5 runs: 99.996%  Std train accuracy among 5 runs: 0.002449  Best test accuracy among 5 runs: 99.350%  Mean test accuracy among 5 runs: 99.31%  Std test accuracy among 5 runs: 0.023664</p>

Table 2: Second parameter setting for LeNet-5 model on MNIST dataset.

In order to further increase the generalization performance, I apply the learning rate to 0.005, momentum to 0.9, and weight decay to  $1e-3$  to evaluate the model with 30 epochs again. By observing the train accuracy curve in the performance graph, we notice the model shows less stable convergence, compared with previous parameter settings. I notice the performances on best train accuracy and mean train accuracy are decreased to around 99% among 5 runs.

Due to a very unstable convergence on test accuracy curve in the graph, I notice the trained model shows some oscillations on test dataset. By applying test dataset to this trained model for 5 runs, I obtain the best test accuracy at 99.02%, mean test accuracy at 98.995%, and standard deviation at 0.08212 among 5 runs. This parameter setting exhibits a fluctuated convergence for test dataset and the model shows some degrees of randomness due to its large standard deviation among 5 runs. The trained model with this parameter setting fails to achieve 90% accuracy on testing dataset.

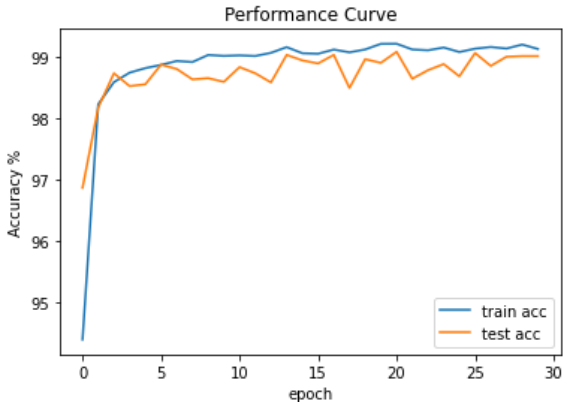
Parameter setting	<p>Optimizer: SGD  Loss function: CrossEntropyLoss  train_batch_size=10  test_batch_size=1000  epoch=30  learning_rate=0.005  momentum=0.9  weight_decay=<math>1e-3</math>  weight initialization=default uniform distribution</p>
Performance curve	
Performance results given 5 runs	<p>Best train accuracy among 5 runs: 99.146%  Mean train accuracy among 5 runs: 99.121%  Std train accuracy among 5 runs: 0.02284  Best test accuracy among 5 runs: 99.02%  Mean test accuracy among 5 runs: 98.955%  Std test accuracy among 5 runs: 0.08212</p>

Table 3: Third parameter setting for LeNet-5 model on MNIST dataset.

Based on the third parameter setting, I think the large momentum may be one of the important factors to contribute to the test accuracy oscillations. Because large momentum indicates more previous gradient contribution to current weight parameter updates. The gradient descent may not find global minimum given large momentum. Therefore, I reduce momentum to 0.5 and apply the same learning rate to 0.005, and weight decay to 1e-3 to evaluate the model with 30 epochs.

By observing the train accuracy curve in the performance graph, we notice the training shows more robust convergence, compared with third parameter settings. The performances on best train accuracy and mean train accuracy are increased to around 99.682% among 5 runs. The standard deviation for training among 5 runs is decreased to 0.01820.

We can also notice a more stable test accuracy convergence curve in the graph. By applying test dataset to this trained model for 5 runs, I obtain the best test accuracy at 99.07%, and mean test accuracy at 99.012%. The standard deviation is 0.03487 among 5 runs. This parameter setting exhibits a more stable convergence for test dataset. A small standard deviation indicates a better stability among 5 runs. However, the model may have overfitting issue because the test accuracy curve shows a decreasing trend at the end, while train accuracy remains the same.

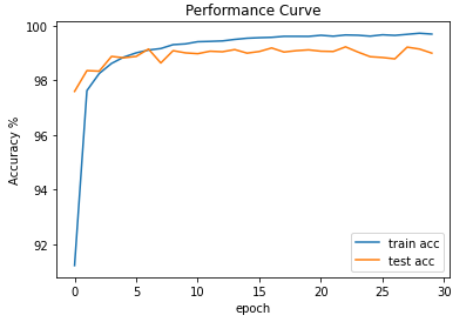
Parameter setting	<p>Optimizer: SGD  Loss function: CrossEntropyLoss  train_batch_size=10  test_batch_size=1000  epoch=30  learning_rate=0.005  momentum=0.5  weight_decay=1e-3  weight initialization=default uniform distribution</p>
Performance curve	
Performance results given 5 runs	<p>Best train accuracy among 5 runs: 99.703%  Mean train accuracy among 5 runs: 99.682%  Std train accuracy among 5 runs: 0.01820  Best test accuracy among 5 runs: 99.07005%  Mean test accuracy among 5 runs: 99.012%  Std test accuracy among 5 runs: 0.03487</p>

Table 4: Fourth parameter setting for LeNet-5 model on MNIST dataset.

From the third parameter setting, the test accuracy performance curve cannot maintain a stable increasing trend. Since the train accuracy is near perfect and test accuracy performs less stable, there might be an overfitting issue. Therefore, I reduce the weight decay to  $1e-5$  and keep other parameter settings the same. The train accuracy curve converges to above 99% after around 5 epochs and indicates the train data fits the model very well. The best train accuracy is 99.98% and the mean train accuracy is 99.95% among 5 runs. The test accuracy curve is more stable as number of epochs increases, compared with performance curve from the third parameter setting. The reduced standard deviation among 5 runs indicates better and stable generalization performance as the weight decay reduces to  $1e-5$ . The objective of 90% test accuracy is achieved as the best test accuracy is 99.23% and mean test accuracy is 99.08%. However, this parameter setting fails to outperform the second parameter setting.

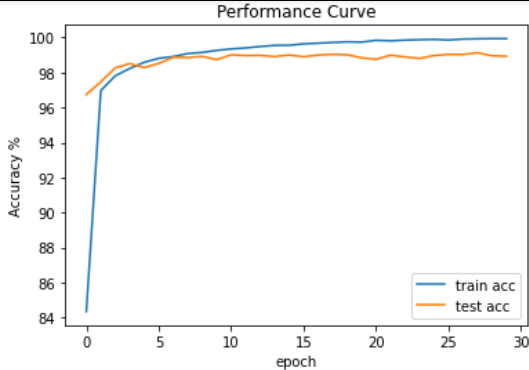
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.0005 momentum=0.9 weight_decay= $1e-5$ weight initialization=default uniform distribution
Performance curve	 <p>The graph titled 'Performance Curve' plots Accuracy % on the y-axis (ranging from 84 to 100 in increments of 2) against epoch on the x-axis (ranging from 0 to 30 in increments of 5). Two lines are shown: 'train acc' (blue) and 'test acc' (orange). The train accuracy starts at approximately 84% at epoch 0, rises sharply to about 97% by epoch 1, and then continues to rise more gradually, reaching approximately 99.98% by epoch 30. The test accuracy starts at approximately 97% at epoch 0, rises to about 98.5% by epoch 1, and then fluctuates slightly around 99% for the remainder of the 30 epochs, with a peak of approximately 99.23% around epoch 20.</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 99.978% Mean train accuracy among 5 runs: 99.955% Std train accuracy among 5 runs: 0.01711 Best test accuracy among 5 runs: 99.229% Mean test accuracy among 5 runs: 99.078% Std test accuracy among 5 runs: 0.09620

Table 5: Fifth parameter setting for LeNet-5 model on MNIST dataset.

The Fashion-MNIST dataset includes 60000 training samples and 10000 test samples with grayscale 28 by 28 clothing images. There are 10 target classes of different types of shirts and pants. We apply this Fashion-MNIST dataset to LeNet-5 neural network to recognize types of clothing items. For all parameter setting, I apply

SGD Optimizer and CrossEntropyLoss as the loss function for weight updates. Then, we apply different hyper parameter settings to determine the best performance of the network. I apply 5 different settings to compare the classification performances on each setting and reach 90% test accuracy as our objective. Due to the randomness of network results, I apply the same parameter setting for 5 runs and evaluate the mean accuracy of test accuracy as the final test accuracy for the given parameter setting.

As for the initial parameter setting, I choose learning rate to 0.0005, batch size to 16, momentum to 0.9, weight decay to 1e-5, and epoch to 30. From the training accuracy curve in the performance graph, the training accuracy curve starts from 80% accuracy and increase to 90% accuracy within 5 epochs. After that, it shows a slight uptrend strength. The best train accuracy is 95.342%, and the mean train accuracy is 94.98% among 5 runs. However, the test dataset does not perform well on this model setting. The test accuracy curve shows around 3% oscillations from epoch 5 to epoch 20. The best test accuracy is 89.37%, the mean test accuracy is 89.095% among 5 runs. The model with this parameter setting fails to achieve the objective accuracy.

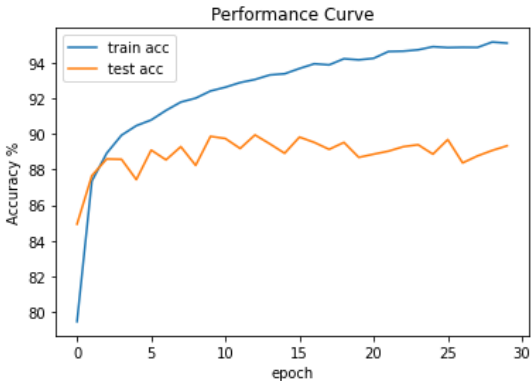
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=16 test_batch_size=1000 epoch=30 learning_rate=0.005 momentum=0.9 weight_decay=1e-5 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 95.342% Mean train accuracy among 5 runs: 94.9833% Std train accuracy among 5 runs: 0.28924 Best test accuracy among 5 runs: 89.37% Mean test accuracy among 5 runs: 89.095% Std test accuracy among 5 runs: 0.47902

Table 6: First parameter setting for LeNet-5 model on Fashion-MNIST dataset.

As for the second parameter setting, I apply 10 train batch size, learning rate to 0.001, momentum to 0.9, and weight decay to 1e-5 to evaluate the model for 20 epochs. By observing the train accuracy curve in the performance graph, we notice the model learns training dataset very fast from 75% to 90% within 5 epochs and shows a convergence trend after that. The best train accuracy is 94.646%, and mean train accuracy is 94.589% among 5 runs. The standard deviation is 0.049 for training among 5 runs.

The test accuracy curve converges steadily after 5 epochs in the graph. By applying test dataset to this trained model for 5 runs, I obtain the best test accuracy at 90.16%, mean test accuracy at 90.011%, and standard deviation at 0.18605 among 5 runs. There are around 4% accuracy difference between training data and unseen test data. The test accuracy curve tends to be flat curve, whereas the train accuracy shows an uptrend strength. We achieve our expectation Based on the 90.011 % mean test accuracy among 5 runs. But this parameter setting does not produce the best generalization performance.

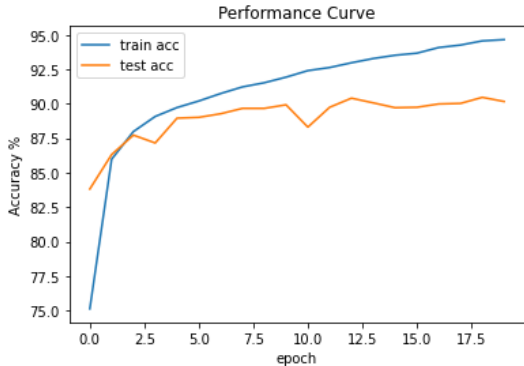
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=20 learning_rate=0.001 momentum=0.9 weight_decay=1e-5 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 94.646% Mean train accuracy among 5 runs: 94.589% Std train accuracy among 5 runs: 0.049109 Best test accuracy among 5 runs: 90.16% Mean test accuracy among 5 runs: 90.011% Std test accuracy among 5 runs: 0.18605

Table 7: Second parameter setting for LeNet-5 model on Fashion-MNIST dataset.

Since the model from second parameter setting achieves our expectation and the training accuracy curve still shows a slight uptrend strength, I increase the number of epochs to 30 and keep all other parameter values the same. From the performance graph, the training accuracy curve starts from 75% accuracy and increase to 90% accuracy within 5 epochs. It still shows a gradual uptrend convergence as epoch increases. The best train accuracy is 96.592%, the mean train accuracy is 96.387%. The test accuracy curve starts above 80% accuracy and converges to 90% accuracy around 5 epochs. Then, the test accuracy curve maintains the same test accuracy level as the number of epochs increases. In addition, the best test accuracy is 90.8%, and the mean test accuracy is 90.021%. The performance curve indicates a sign of overfitting problem because the model fits the training dataset too well but fails to improve generalization performance on test dataset.

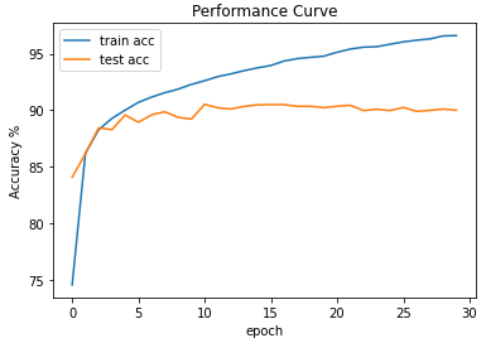
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.001 momentum=0.9 weight_decay=1e-5 weight initialization=default uniform distribution
Performance curve	 <p>The graph titled 'Performance Curve' plots Accuracy % on the y-axis (ranging from 75 to 95) against epoch on the x-axis (ranging from 0 to 30). Two lines are shown: 'train acc' (blue) and 'test acc' (orange). The training accuracy starts at 75% at epoch 0 and increases steadily, reaching about 90% by epoch 5 and continuing to rise to approximately 96.6% by epoch 30. The test accuracy starts at approximately 84% at epoch 0, rises to about 88% by epoch 2, and then fluctuates slightly around 90% from epoch 5 onwards, peaking at approximately 90.8% around epoch 10.</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 96.592% Mean train accuracy among 5 runs: 96.387% Std train accuracy among 5 runs: 0.20648 Best test accuracy among 5 runs: 90.8% Mean test accuracy among 5 runs: 90.021% Std test accuracy among 5 runs: 0.52028

Table 8: Third parameter setting for LeNet-5 model on Fashion-MNIST dataset.

As for the fourth parameter setting, I apply train batch size to 16, increase learning rate to 0.01, and choose momentum to 0.5, and weight decay to 1e-6. From the performance curve, the training accuracy increase from 75% to 90% within 5 epochs. Then, the training accuracy slowly climbs to 95% around 30 epochs. The best train accuracy is 96.556%, and the mean train accuracy is 96.362% among 5 runs.



The test accuracy curve starts from around 85% and converges to around 90% around 5 epochs. Then, the test accuracy curve experiences some degrees of oscillations as epoch increases. The best test accuracy is 90.34%, and the mean test accuracy is 89.918% among 5 runs. The mean test accuracy fails to reach the expectation. In addition, the standard deviation among 5 runs is relatively larger than the third parameter setting.

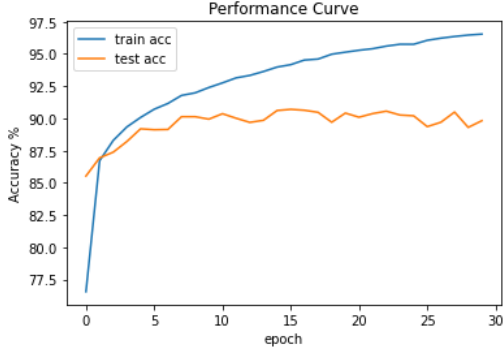
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=16 test_batch_size=1000 epoch=30 learning_rate=0.01 momentum=0.5 weight_decay=1e-6 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 96.556% Mean train accuracy among 5 runs: 96.362% Std train accuracy among 5 runs: 0.23722 Best test accuracy among 5 runs: 90.34% Mean test accuracy among 5 runs: 89.918% Std test accuracy among 5 runs: 0.35521

Table 9: Fourth parameter setting for LeNet-5 model on Fashion-MNIST dataset.

From the previous performance curve and statistics, the test accuracy indicates some degrees of overfitting issues. Because the gap between train accuracy curve and test accuracy curve is larger as the number of epochs increase. I reduced the learning rate and increase momentum, compared with the third parameter setting.

The train accuracy curve converges to above 90% after around 5 epochs from 70% and continue to improve accuracy to 95% as the number of epochs increase. The best train accuracy is 95.512% and the mean train accuracy is 95.162% among 5 runs. The test accuracy curve converges from 80% to 90% after around 5 epochs and shows stable convergence. The best test accuracy under this parameter setting is 91.06% and the mean test accuracy is 90.426% among 5 runs. The small gap between train accuracy curve and test accuracy curve indicates less overfitting issues, compared

with previous parameter settings. The model with this parameter setting achieves the best mean test accuracy among 5 different settings with acceptable standard deviation among 5 runs.

Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.0005 momentum=0.9 weight_decay=1e-6 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 95.512% Mean train accuracy among 5 runs: 95.162% Std train accuracy among 5 runs: 0.27929 Best test accuracy among 5 runs: 91.06% Mean test accuracy among 5 runs: 90.436% Std test accuracy among 5 runs: 0.35869

Table 10: Fifth parameter setting for LeNet-5 model on Fashion-MNIST dataset.

The CIFAR-10 dataset includes 50000 training samples and 10000 test samples with colored 32 by 32 object images of 10 classes. We apply this CIFAR-10 dataset to LeNet-5 neural network to recognize colored objects from 10 classes. For all parameter setting, I apply SGD Optimizer and CrossEntropyLoss as the loss function for weight updates. Then, we apply different hyper parameter settings to determine the best performance of the network. I apply 5 different settings to compare the network performances and reach 65% test accuracy as our objective. Due to the randomness of network results, I apply the same parameter setting for 5 runs and evaluate the mean accuracy of test accuracy as the final test accuracy for the given parameter setting.

As for the initial parameter setting, I choose train batch size to 10, learning rate to 0.001, momentum to 0.9, weight decay to 1e-5 for the model. From the performance curve, the train accuracy starts below 40% accuracy and increase to 65% accuracy around 5 epochs. Then, the train accuracy gradually increases above 80% around 30

epochs. The best train accuracy is 83.668%, and the mean train accuracy is 83.235% among 5 runs. The test accuracy curve improves from around 48% accuracy to 60% accuracy with 5 epochs. The test accuracy curve converges around 60% and show downward strength as number of epochs increases. The best test accuracy is 62.78%, and the mean test accuracy is 61.682% among 5 runs. The huge difference between train accuracy and test accuracy indicates the model overfits training dataset and fail to improve test accuracy.

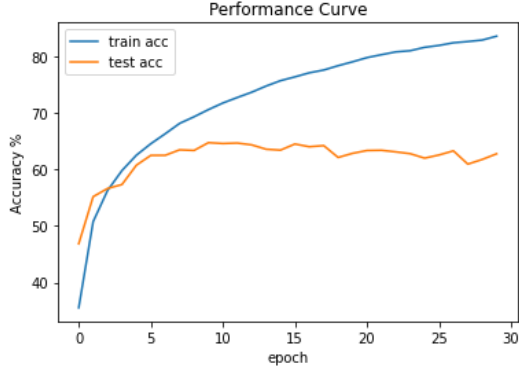
Parameter setting	<p>Optimizer: SGD  Loss function: CrossEntropyLoss  train_batch_size=10  test_batch_size=1000  epoch=30  learning_rate=0.001  momentum=0.9  weight_decay=1e-5  weight initialization=default uniform distribution</p>
Performance curve	 <p>The graph, titled 'Performance Curve', plots accuracy percentage against the number of epochs. The training accuracy (blue line) shows a consistent upward trend, starting from approximately 35% at epoch 0 and reaching about 84% by epoch 30. The test accuracy (orange line) starts at approximately 48%, rises to a peak of about 65% around epoch 10, and then exhibits a slight downward trend with some fluctuations, ending at approximately 63% by epoch 30. This divergence between training and testing accuracy is a classic sign of overfitting.</p>
Performance results given 5 runs	<p>Best train accuracy among 5 runs: 83.668%  Mean train accuracy among 5 runs: 83.235%  Std train accuracy among 5 runs: 0.456018  Best test accuracy among 5 runs: 62.78%  Mean test accuracy among 5 runs: 61.682%  Std test accuracy among 5 runs: 0.71300</p>

Table 11: First parameter setting for LeNet-5 model on CIFAR-10 dataset.

As for the second parameter setting, I increase the weight decay and reduce epoch size to 20. From the performance curve, the train accuracy increase from around 40% accuracy to 65% accuracy within 5 epochs. Then, the train accuracy gradually increases around 77% at the end. The best train accuracy is 77.794%, and the mean train accuracy is 77.407% among 5 runs. The test accuracy curve improves from around 45% to 60% with 5 epochs and shows an upward strength as the number of epochs increases. The best test accuracy among 5 runs reach to 65%. Due to the model randomness, the mean test accuracy among 5 runs is 64.68%. This parameter setting slightly reduces overfitting problem. But the standard deviation indicates this parameter setting produces unstable performance among 5 runs.

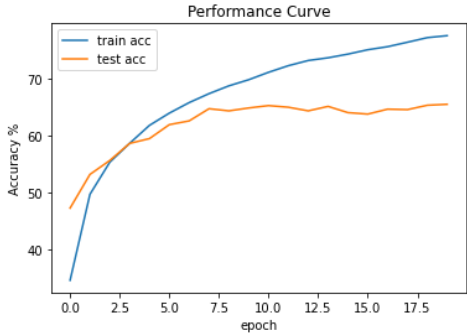
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=20 learning_rate=0.001 momentum=0.9 weight_decay=1e-3 weight initialization=default uniform distribution
Performance curve	 <p>The graph, titled 'Performance Curve', plots Accuracy % against epoch. The x-axis ranges from 0.0 to 17.5 with increments of 2.5. The y-axis ranges from 40 to 70 with increments of 10. Two lines are shown: a blue line for 'train acc' and an orange line for 'test acc'. The train accuracy starts at approximately 35% at epoch 0 and increases steadily, reaching about 78% by epoch 17.5. The test accuracy starts at approximately 45% at epoch 0, rises to about 60% by epoch 5, peaks at approximately 65% around epoch 10, and then slightly declines to about 64% by epoch 17.5.</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 77.794% Mean train accuracy among 5 runs: 77.407% Std train accuracy among 5 runs: 0.51768 Best test accuracy among 5 runs: 65.57% Mean test accuracy among 5 runs: 64.68% Std test accuracy among 5 runs: 0.86625

Table 12: Second parameter setting for LeNet-5 model on CIFAR-10 dataset.

As for the third parameter setting, I reduce the weight decay to  $1e-2$  and momentum to 0.5. From the performance curve, the train accuracy continuously increases from around 20% to 60% for 25 epochs. The best train accuracy is 62.822% and the mean train accuracy is 61.073% among 5 runs. The test accuracy curve shows consistent improvement with train accuracy curve. However, the test accuracy fails to reach to 65% accuracy objective. The best test accuracy among 5 runs is 61.06%. The mean test accuracy among 5 runs is 59.530%. Even though the parameter setting does not have overfitting problem, the mean test accuracy fails to reach our expectation.

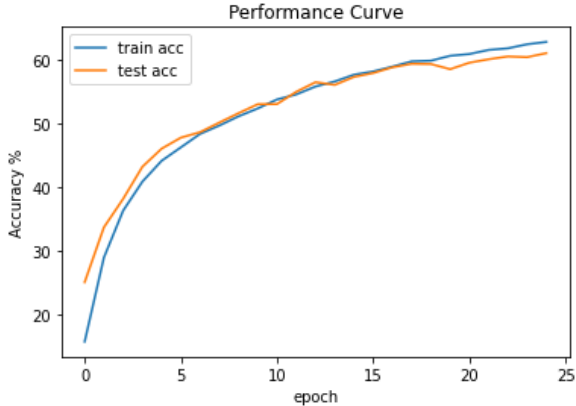
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=16 test_batch_size=1000 epoch=25 learning_rate=0.001 momentum=0.5 weight_decay=1e-2 weight initialization=default uniform distribution
Performance curve	 <p>The graph titled 'Performance Curve' plots Accuracy % against epoch. The x-axis ranges from 0 to 25 with increments of 5. The y-axis ranges from 20 to 60 with increments of 10. Two lines are shown: 'train acc' in blue and 'test acc' in orange. Both lines show a steady upward trend. The train accuracy starts at approximately 15% at epoch 0 and reaches about 63% by epoch 25. The test accuracy starts at approximately 25% at epoch 0 and reaches about 61% by epoch 25. The two lines are very close to each other throughout the training process.</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 62.822% Mean train accuracy among 5 runs: 61.073% Std train accuracy among 5 runs: 1.08470 Best test accuracy among 5 runs: 61.06% Mean test accuracy among 5 runs: 59.530% Std test accuracy among 5 runs: 0.95699

Table 13: Third parameter setting for LeNet-5 model on CIFAR-10 dataset.

As for the fourth parameter setting, I reduce the learning rate to 0.0005 and the weight decay to  $1e-3$  and apply momentum to 0.9. From the performance curve, the train accuracy shows a consistent improvement and increases from around 30% to 80% for 30 epochs. The best train accuracy is 82.208% and the mean train accuracy is 81.653% among 5 runs. The test accuracy curve starts from 40% and converges to around 64% after 5 epochs. Even though the best test accuracy among 5 runs reach to 65.18%, the mean test accuracy among 5 runs is 64.486%. The gap between train

accuracy and test accuracy indicates the model overfits the training data. The mean test accuracy fails to reach our expected objective.

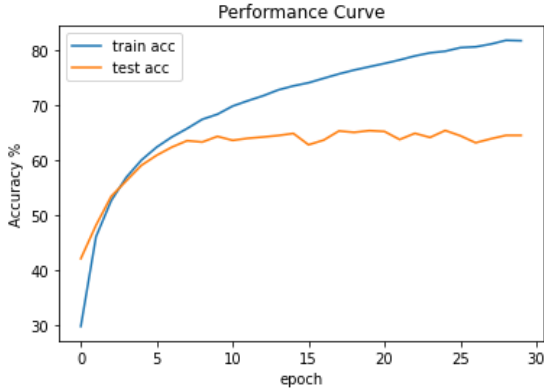
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.0005 momentum=0.9 weight_decay=1e-3 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 82.208% Mean train accuracy among 5 runs: 81.653% Std train accuracy among 5 runs: 0.360187 Best test accuracy among 5 runs: 65.18% Mean test accuracy among 5 runs: 64.486% Std test accuracy among 5 runs: 0.41209

Table 14: Fourth parameter setting for LeNet-5 model on CIFAR-10 dataset.

As for the fifth parameter setting, I apply the same parameter setting except that I increase weight decay to  $1e-2$  to mitigate the overfitting issue. From the performance curve, the train accuracy shows a steady improvement and increases from around 30% to 70% for 30 epochs. The test accuracy curve starts from 40% and converges above 65%. From the performance curve, the test accuracy improves consistently as the train accuracy curve increases. There is no overfitting issue under this parameter setting. The best train accuracy is 68.62% and the mean train accuracy is 67.98% among 5 runs. The best test accuracy among 5 runs reaches to 66.24% and the mean test accuracy among 5 runs is 65.198%. This parameter setting produces the best generalization performance among the 5 different parameter settings and reaches the objective goal.

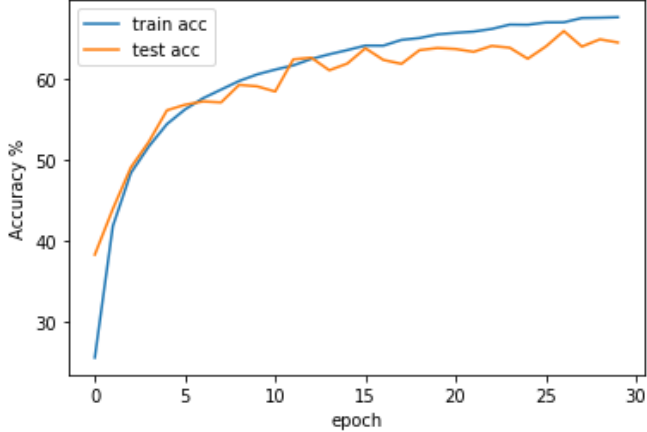
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.0005 momentum=0.9 weight_decay=1e-2 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy among 5 runs: 68.62% Mean train accuracy among 5 runs: 67.980% Std train accuracy among 5 runs: 0.488375 Best test accuracy among 5 runs: 66.24% Mean test accuracy among 5 runs: 65.198% Std test accuracy among 5 runs: 0.75055

Table 15: Fifth parameter setting for LeNet-5 model on CIFAR-10 dataset.

From the best performances of three dataset, I notice the LeNet-5 model produces the highest test accuracy on MNIST dataset. By attempting different parameter settings for these three datasets, the best test accuracy can reach to 99% after fine-tuning hyperparameters. The best classification performance of Fashion-MNIST dataset on LeNet-5 model reduces to 90% test accuracy. In addition, the best classification performance of CIFAR-10 dataset on LeNet-5 model only achieve 65% test accuracy. One of reason to explain the classification accuracy difference among the three datasets is LeNet-5 model fails to capture more complex features for classification from Fashion-MNIST and CIFAR datasets. Because LeNet-5 is composed of two hidden layers with 6 filter channels for the first layer and 16 filter channels for the second layer. Since MNIST and Fashion-MNIST dataset are grayscale images, both datasets have input channel of 1. Visually, images in Fashion-MNIST dataset are more complex than handwritten images. Filter channels in LeNet-5 model may not extract enough features for classification. In addition, the CIFAR dataset is RGB images with 3 channels of input images. Filter channels in both hidden layers of LeNet-5 model are not enough to extract all features to identify

classes accurately from images in CIFAR dataset. The CNN model may require more filter channels and hidden layers to improve accuracy. Therefore, the classification performance reduced to 65% test accuracy, in comparison with 99% test accuracy on MNIST dataset.

Best parameter setting to produce the highest accuracy on MNIST test Dataset.

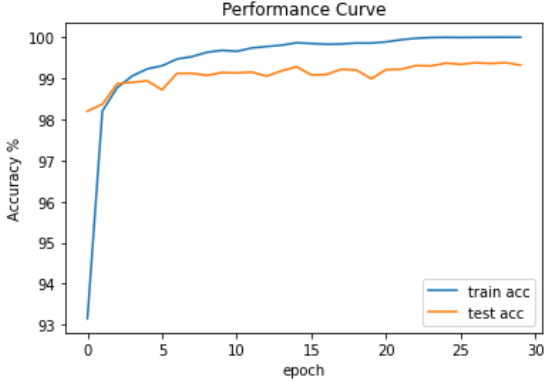
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.01 momentum=0.5 weight_decay=1e-4 weight initialization=default uniform distribution
Performance curve	 <p>The graph, titled 'Performance Curve', plots Accuracy % against epoch. The training accuracy (blue line) shows a rapid increase from approximately 93.2% at epoch 0 to nearly 100% by epoch 30. The test accuracy (orange line) starts higher, around 98.2%, and quickly reaches a plateau of approximately 99.3% after epoch 5, with minor fluctuations throughout the remaining epochs.</p>
Performance results given 5 runs	Best train accuracy among 5 runs: 100.0% Mean train accuracy among 5 runs: 99.996% Std train accuracy among 5 runs: 0.002449 Best test accuracy among 5 runs: 99.350% Mean test accuracy among 5 runs: 99.31% Std test accuracy among 5 runs: 0.023664

Table 16: Best parameter setting for MNIST dataset



Best parameter setting to produce the highest accuracy on Fashion-MNIST test Dataset.

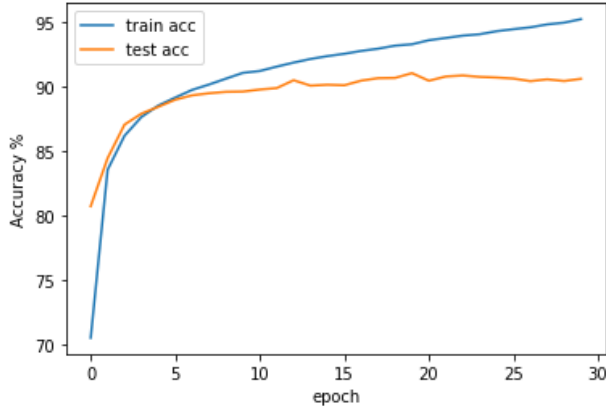
Parameter setting	<p>Optimizer: SGD</p> <p>Loss function: CrossEntropyLoss</p> <p>train_batch_size=10</p> <p>test_batch_size=1000</p> <p>epoch=30</p> <p>learning_rate=0.0005</p> <p>momentum=0.9</p> <p>weight_decay=1e-6</p> <p>weight initialization=default uniform distribution</p>
Performance curve	<p>Performance Curve</p> 
Performance results given 5 runs	<p>Best train accuracy among 5 runs: 95.512%</p> <p>Mean train accuracy among 5 runs: 95.162%</p> <p>Std train accuracy among 5 runs: 0.27929</p> <p>Best test accuracy among 5 runs: 91.06%</p> <p>Mean test accuracy among 5 runs: 90.436%</p> <p>Std test accuracy among 5 runs: 0.35869</p>

Table 17: Best parameter setting for Fashion-MNIST dataset

Best parameter setting to produce the highest accuracy on CIFAR-10 test Dataset.

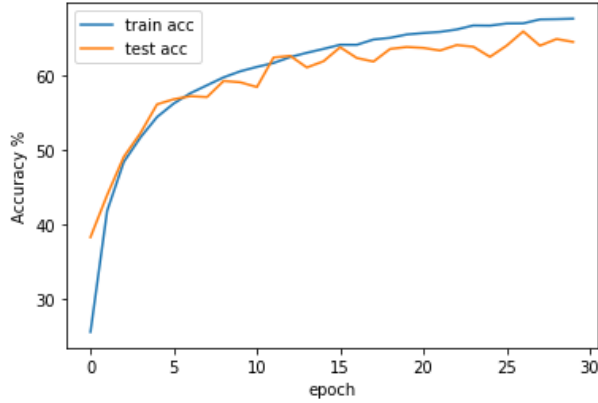
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=30 learning_rate=0.0005 momentum=0.9 weight_decay=1e-2 weight initialization=default uniform distribution																								
Performance curve	 <table><caption>Approximate data points from the Performance Curve graph</caption><thead><tr><th>Epoch</th><th>Train Accuracy (%)</th><th>Test Accuracy (%)</th></tr></thead><tbody><tr><td>0</td><td>25.0</td><td>38.0</td></tr><tr><td>5</td><td>55.0</td><td>55.0</td></tr><tr><td>10</td><td>62.0</td><td>60.0</td></tr><tr><td>15</td><td>65.0</td><td>63.0</td></tr><tr><td>20</td><td>67.0</td><td>65.0</td></tr><tr><td>25</td><td>68.0</td><td>66.0</td></tr><tr><td>30</td><td>68.62</td><td>66.24</td></tr></tbody></table>	Epoch	Train Accuracy (%)	Test Accuracy (%)	0	25.0	38.0	5	55.0	55.0	10	62.0	60.0	15	65.0	63.0	20	67.0	65.0	25	68.0	66.0	30	68.62	66.24
Epoch	Train Accuracy (%)	Test Accuracy (%)																							
0	25.0	38.0																							
5	55.0	55.0																							
10	62.0	60.0																							
15	65.0	63.0																							
20	67.0	65.0																							
25	68.0	66.0																							
30	68.62	66.24																							
Performance results given 5 runs	Best train accuracy among 5 runs: 68.62% Mean train accuracy among 5 runs: 67.980% Std train accuracy among 5 runs: 0.48837 Best test accuracy among 5 runs: 66.24% Mean test accuracy among 5 runs: 65.198% Std test accuracy among 5 runs: 0.75055																								

Table 18: Best parameter setting for CIFAR-10 dataset

LeNet-5 model shows above 99% test accuracy on original positive digit images by training the model with white digits and black background. However, unlike human beings, LeNet-5 model cannot understand the negative handwritten digits with pixel value reversed given the model trained with positive handwritten digits dataset.

The negative handwritten digit image can be represented as inverted original positive digit images. To convert positive handwritten digits images to negatives in testing dataset, I need to make sure the reversed dataset has similar data distribution as the original dataset. If applying 1 minus the normalized original images, the mean of reversed images shifts to near 1. Therefore, I first load the normalized original dataset based on the mean of 0.1307 and standard deviation of 0.3081. Then, I apply negative sign to the digit images to ensure the reversed dataset has zero mean data distribution as well. From the statistics in Table 19, the mean of original normalized test dataset is 0.0059 with 1 standard deviation. By applying negative function to the original test dataset, the mean of reversed test dataset is -0.0059 with 1 standard

deviation. The original test dataset is in a range of -0.4242 to 2.8215. And the reversed test dataset is in a range of -2.8215 to 0.4242. Given those statistics, we make sure both original test data and reversed test data are zero mean and have similar range. In Table 19, there is an example of the original sample digit image and the reversed sample digit image histogram with its histogram.

The statistics of normalized test dataset for both original digits and negative digits are shown in Table 19.

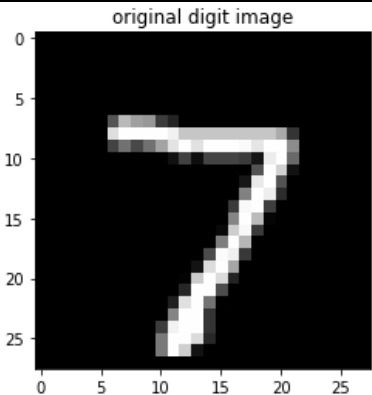
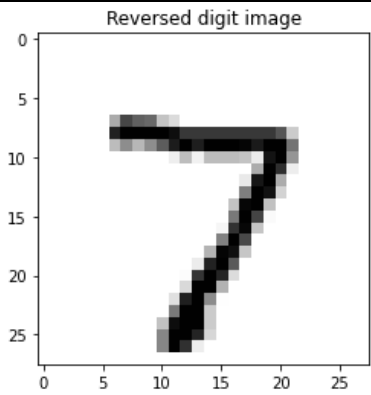
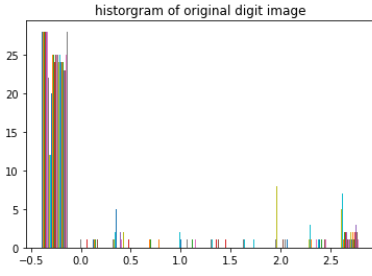
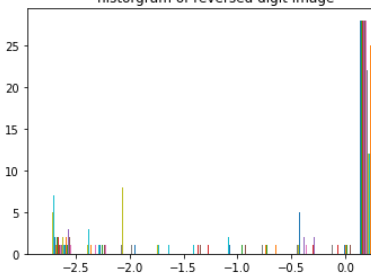
	Original digit image	Reversed digit image
Example		
Histogram of the given sample		
Statistics	mean of original test data: 0.005889658 std of original test data: 1.0077257 max of original test data: 2.8214867 min of original test data: -0.42421296	mean of reversed test data: -0.005889658 std of reversed test data: 1.0077257 max of reversed test data: 0.42421296 min of reversed test data: -2.8214867

Table 19: Statistics of test dataset

To determine the classification performance of reversed digit dataset, I apply the reversed handwritten digit images to the trained LeNet-5 model from part b. The accuracy of reversed test dataset is 37.2%. Given the train dataset of positive handwritten digit images for the LeNet-5 model, the test accuracy of positive digits images reaches to above 99%. However, the test accuracy of negative digit images drops to below 50%, which indicates the LeNet-5 model trained with positive digit images cannot recognize the reversed digit images well. From my perspective, the main reason for this result is that the LeNet-5 model only learns features from the

original handwritten digit images and the features in reversed dataset are different from those in original dataset for LeNet-5 model due to the reversed pixel values.

To be able to recognize both original and negative images from the MNIST test dataset, I modify train dataset for the LeNet-5 model and reach to 99% test accuracy for both positive and negative test dataset. To recognize negative digit images, LeNet-5 model should be trained to capture features from negative digit images as well as positive digit image. Therefore, I combine both original train dataset and reversed train dataset to train the LeNet-5 model with SGD optimizer and CrossEntropyLoss as the loss function. LeNet-5 model is able to capture features from both types of digit images. Since we obtain the reversed dataset by apply negative function to the original normalized dataset. The mean of both datasets is zero centered. By training the LeNet-5 model with combined original train dataset and reversed train dataset, I am able to achieve over 99% test accuracy for both original test dataset and reversed test dataset. The best accuracy on both positive and negative digit test dataset is 99.225%.

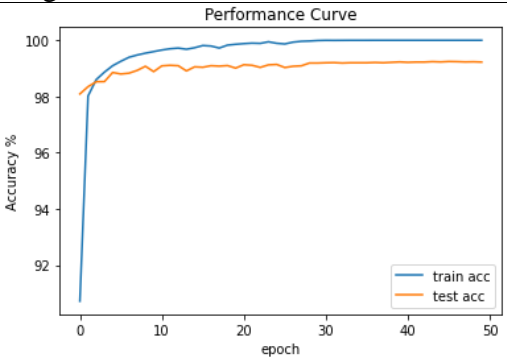
Parameter setting	Optimizer: SGD Loss function: CrossEntropyLoss train_batch_size=10 test_batch_size=1000 epoch=50 learning_rate=0.001 momentum=0.9 weight_decay=1e-4 weight initialization=default uniform distribution
Performance curve	
Performance results given 5 runs	Best train accuracy: 100.0% Best test accuracy: 99.225%

Table 20: LeNet-5 model for both original and negative digit images

Originally, the LeNet-5 model is trained with only positive handwritten digit dataset to recognize positive digits. LeNet-5 model update its weight parameter based on the features captured in the network. The features of reversed test dataset are not the same set of features of the original dataset due to the reversion. Since the LeNet-5 model is not trained with reversed digit images, it fails to capture these new features

to perform classification accurately given the fact that weight parameter is trained by the original dataset. Therefore, I reverse the train dataset and combine both types of digit images together after normalization to capture both features for weight updates. The LeNet-5 model will have the ability to recognize both positive and negative handwritten digits from the test dataset. From the performance curve, the train accuracy curve improves from 90% accuracy to over 98% accuracy within 10 epochs and gradually converges to 100% accuracy at the end. The test accuracy curve starts from 98% and converges to over 99% as train accuracy reaches to 100%. The accuracy result of train dataset and test dataset with performance curve is shown in Table 20.