Vishal Patil
USC ID: 7988-9741-79
vspatil@usc.edu

EE 569 Homework #5
April 16, 2023

Question 1: CNN Training on LeNet-5

Abstract and Motivation

The field of Computer Vision has made tremendous strides with the advent of neural networks. Neural network is a form of machine learning algorithm modeled after the neuron cells in the human brain. These can be thought of as an interconnected network of nodes where each node computes value based on inputs and depending on an activation function whether a node will transmit its output to the next layer of nodes or not is decided.

One key advancement in Computer Vision has been object recognition. Convolutional Neural Networks (CNNs) have been particularly successful in this area. These networks use convolutional layers to extract features from the image and then use fully connected layers to classify objects in the image.

Another area where CNNs have helped make progress is image segmentation based on various characteristics like similar texture regions, or similar colors etc. They have helped in creating highly accurate segmentation models which are being used in a range of applications from medical imaging to autonomous driving.

In this section we will be diving deeper to understand one such CNN called LeNet5. This was developed by Yann LeCun in the 1990s and has been largely successful in recognising handwritten digits. It's architecture is as shown below:

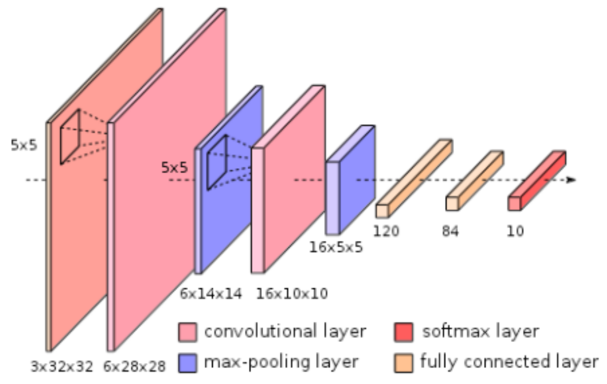


Figure 1: A CNN architecture derived from LeNet-5

The architecture consists of 2 convolutional layers for extracting features from the image, two max pooling layers to reduce the feature dimensions and 3 fully connected layers that perform classification based on the feature extracted by the previous layers.

LeNet5 was successful in achieving high levels of accuracy reaching up to 99% on MNIST dataset, a dataset which contains images of handwritten digits. Due to its success it serves as a guiding model for many new CNN architectures and it is still being used today for performance evaluation comparison studies for newer CNN architectures.

Approach and Procedure

b) Compare classification performance on different dataset

The following hyper-parameters were used for tuning the CNN:

- Train batch size
- Learning rate
- Momentum

Loading the data

The following are the steps to load the data:

Step 1: Download the required dataset (MNIST, Fashion MNIST, CIFAR-10) from torchvision datasets. We can specify the dataset to be used for training or testing by setting train = True or False for train and test set respectively. The dataset is downloaded to a local folder whose path is to be specified in the root parameter

Step 2: transform the data to normalize the inputs transform.compse() function to the transform parameter when downloading the dataset. We use empirical values for mean and standard deviation for each dataset to normalize them. These empirical values are as given in the table below:

Dataset	Mean	Standard Deviation
MNIST	0.1307	0.3081
Fashion MNIST	0.28604	0.35302
CIFAR-10	0.4914,0.4822,0.4465 For R,G,B channels	0.2470,0.2435,0.2616 For R,G,B channels

Step 3: Load the data into using Dataloaders from pytorch. Dataloader() function is used to load the train set and test sets. This function helps in feeding the data set in batches to the model.

We specify the train batch size and test batch size at this step. We can also specify for the batches to be selected randomly for the train set so that ordering in the data does not affect the model performance. This is done by setting the shuffle parameter to true for the train set.

Defining the CNN model

In pytorch to define a model we have to specify the model in the form of a class. Using this class we can then create several instances of the model as different objects. We define the CNN model for LeNet 5 as follows:

Step 1: Each model class should inherit from the nn.Module pytorch class. The 2 necessary functions to be present in the class are __init__ () and forward() functions. The __init__() function is a constructor which is used to initialize the class. We define the required layers in for the CNN in this layer

Step 2: Inside the _init__() function we define the following layers:

- 2 convolutional layers using the nn.Conv2d() function. This takes in the number of input channels, number of output channels, kernel size, stride, and padding as inputs.

  Convolution layer 1 :
  We specify input channels as 1, output channels as 6, kernel size used for convolving as 5, stride as 1, which specifies how much the convolving kernel has to be moved and padding as 2, which specifies how much padding to be applied to the input image. We specify 2 as padding because for MNIST and Fashion MNIST the image dimensions are 28x28 where LeNet5 expects an image of size 32x32. For CIFAR 10 we change the number of input channels from 1 to 3 because the image are RGB images

  Convolution layer 2:
  We specify the input channels as 6, output channels as 16, kernel size as 5, stride as 1 and padding as 0

- 3 fully connected layers using the nn.Linear() function. This takes the number of input features and number of output features as parameters

  Fully connected layer 1:
  We specify the input features as 16*5*5 = 400 and output features 120. This reduces the dimensions from 400 to 120.

  Fully connected layer 2:
  We specify input feature as 120, and output features as 84

Fully connected layer 3:
We specify the input features as 84 and finally output feature as 10 corresponding to 10 classes into which we are trying to classify the input images.

Step 3: Inside the forward() function we specify how the layers will be connected to each other and how the data will flow from one layer to the next.

- We apply ReLu activation function to the output of the first convolution layer and feed the ReLu output to max_poo2d layer with kernel size 2, which applies 2D max pooling operation. The max pooling operation is basically a down sampling operation in which we move a widow of kernel size over the output of the convolution layer and take the max value from the window
- We apply the maxpool 2d layer output to the second convolution layer, apply ReLu activation on the output of the second convolution layer and then pass the output of ReLu activation into the second maxpool 2d layer again with kernel size 2.
- The output of the second maxpool 2d layer is flattened and fed into the first fully connected layer. We take ReLu activation from the output of the first layer and pass it to the second fully connected layer. We perform the same step with output of the second fully connected layer and pass it to the final fully connected layer.
- Finally we apply the softmax function of the output of the last layer of CNN to convert the outputs into probabilities and assign the class with highest probability as the class of the input image.

Define Criterion and Optimizer

In order to train the CNN we need to specify what criterion function is to be used to compute the losses and what optimizer to be used to compute the gradients. For our model we are using Cross Entropy loss as the criterion and stochastic gradient descent as the optimizer.

Training the CNN

The following steps are performed to train the CNN:

Step 1: put the model in train mode using model.train() function

Step 2: iterate over the batches using the train loader

Step 3: For each input image and label in the batch

- Clear the gradients using optimizer.zro_grad()
- Get the output by feeding the input image to the model
- Calculate the loss by passing the model output and labels to the criterion function
- Update the weights and optimizer parameter of the neural network using backpropagation of the loss by calling loss.backward() and optimizer.step() functions

Step 4: Repeat step 3 for all the train batches.

Testing the CNN

The following steps are performed to test the CNN:

Step 1: put the model into testing states by calling torch.no_grad() function. This specifies that now when the data is fed into the model do not calculate the gradients as we are testing the model.

Step 2: iterate over the batches from test set

Step 3: for each image and label in the batch
- Get the output of the from the model by passing the image to it.
- Compute the predicted class by taking the argmax of the model output
- Calculate the accuracy using the number of correctly classified inputs

Step 4: Repeat step 3 for all the batches

Finding the best parameters

We use 3 different combinations of training batch size, learning rate and momentum. The combinations used for each dataset are as given below:

Dataset	Training batch size	Learning rate	Momentum
MNIST	32	0.001	0.6
	32	0.01	0.8
	64	0.01	0.8
Fashion MNIST	32	0.001	0.6
	32	0.01	0.8
	64	0.01	0.8
CIFAR-10	32	0.001	0.6
	32	0.01	0.8
	64	0.008	0.9

For each of the combinations we run 5 epochs. To evaluate the performance we first calculate the mean testing accuracy over all the epochs and call it mean epoch testing accuracy then using this we calculate the mean testing accuracy, standard deviation of testing accuracy and the max testing accuracy over the 5 runs.

The parameters which gave the highest max testing accuracy over the 5 runs are selected as best parameters. In case the max testing accuracy over 5 runs is close we then consider the mean and standard deviation of the average epoch testing accuracy and select the parameters which gives the lowest standard deviation of the average epoch testing accuracy.

We take the epoch with highest test accuracy achieved for plotting the epoch vs accuracy plot.

c) Evaluation and Ablation Study

1)
Using the best parameters obtained for each dataset from part 1b) above we train the model and obtain predictions as specified in the above section.

To generate the confusion matrix, confusion_matrix() function from scikit-learn metrics module is used. This function takes actual labels and predicted labels and generates a confusion matrix where the rows indicate the true labels and columns indicate the predicted labels. We can normalize this matrix by setting normalize='true' in the confusion matrix.

The matrix is generated by calculating the number of true positives, true negatives, false positives and false negatives. An ideal confusion matrix would have diagonal elements as 1 and off diagonal elements as 0.

Using the confusion matrix we find the top 3 confused classes and choose those instances where the model gave those predictions and display one sample image in each case.

2)
The Receiver Operating Characteristic curve (ROC) is a curve that is plotted by taking the true positivity rate (TPR) along the y axis and false positive rate (FPR) along the x axis. It is one of the metrics to evaluate multiclass performance. The TPR and FPR are give as below:

TPR = true positives / (true positives + false negatives)
FPR = false positives / (false positives + true negatives)

The FPR and TPR are generally used in binary classification but can be used for multiclass classification by binarizing the output. One method which we have used is the one vs rest scheme where we consider one of the classes as a positive class and compare it with all the other classes considering them to be negative classes.

To plot the ROC curve for CIFAR 10 dataset use the following steps:

Step 1: Obtain the softmax outputs from the model on testing data and not the label predictions. This is just the output probabilities.

Step 2: Transform the class labels into one hot encoded form so as to apply the one vs rest scheme. This is done by using the LabelBinarizer() function from scikit-learn library.

Step 3: Using these transformed label and output probabilities we plot the ROC curve for all the 10 classes. We use the function RocCurveDisplay.from_predictions() which takes the true labels and output probabilities as input and plot the ROC curve.

3)
To summarize the ROC into a single metric we use another metric called Area Under the Curve (AUC). This is calculated using the auc() function which takes in tpr and fpr as input parameters.

We calculate the AUC for all the 10 classes from the CIFAR10 dataset. Then we check if each class has an equal number of labels. If each class has equal number of labels then we can simply take the average of all 10 AUCs to get the mean AUC for the dataset or else we take a weighted average of the 10 AUCs depending on the number labels in each class.

d)
Adding symmetric label noise to data

Since most of the time real world datasets have noise we try to evaluate the CNN performance by adding certain levels of noise and see how noise affects the testing accuracy.

The following steps are performed to add symmetric label noise in the CIFAR 10 dataset:

Step 1: Depending on the desired level of noise calculate how many labels have to changed

Step 2: Depending of the number of labels to be changed generate that many random indexes

Step 3: For each of the random index generated change the true label to any other label randomly.

In this section we add 20%, 40%, 60% and 80% noise in the CIFAR-10 dataset and train the CNN using this noisy data and evaluate its testing accuracy, for each noise level we train the model for 5 epochs for 5 times and take the average testing accuracy over 5 runs to plot the noise vs test accuracy plot.

Experimental Results:

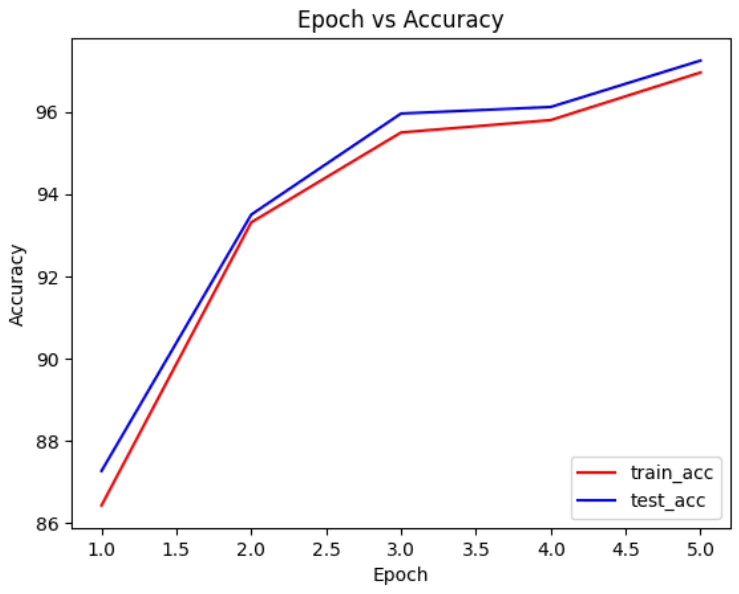
1b) Compare classification performance of different datasets

MNIST dataset

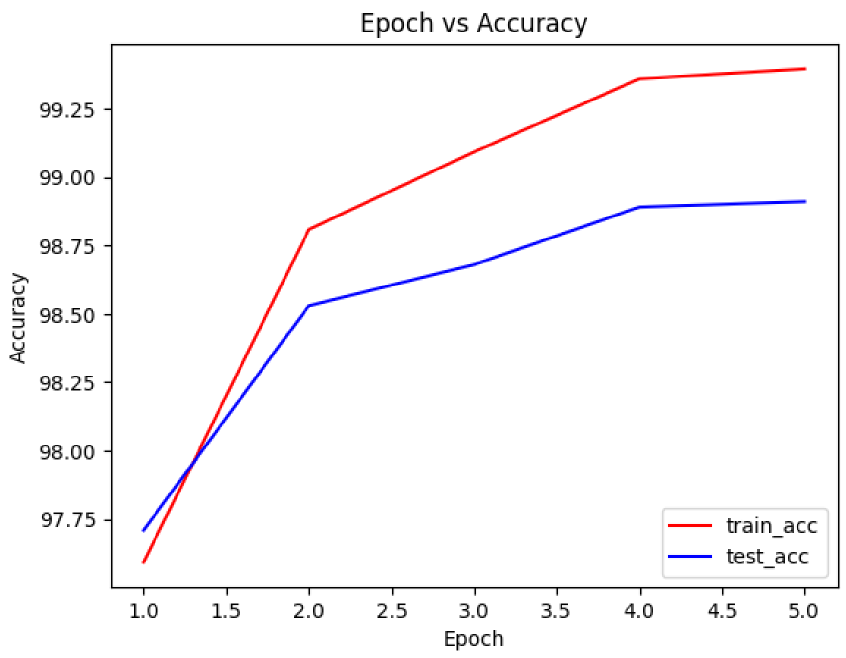
The table below shows the best accuracy, mean accuracy and standard deviation of accuracy for 5 runs with each run going up to 5 epochs for 3 different settings of the hyperparameters namely train batch size, learning rate, and momentum. The statistics are reported on testing accuracy.

Hyper parameter setting	Best accuracy (5 runs)	Mean accuracy (5 runs)	STD accuracy (5 runs)
Train batch size = 32 Learning rate = 0.001 Momentum = 0.6	97.25%	93.110%	4.096
Train batch size = 32 Learning rate = 0.01 Momentum = 0.8	99.04%	98.479%	0.435
Train batch size = 64 Learning rate = 0.01 Momentum = 0.8	98.79%	98.228%	0.480

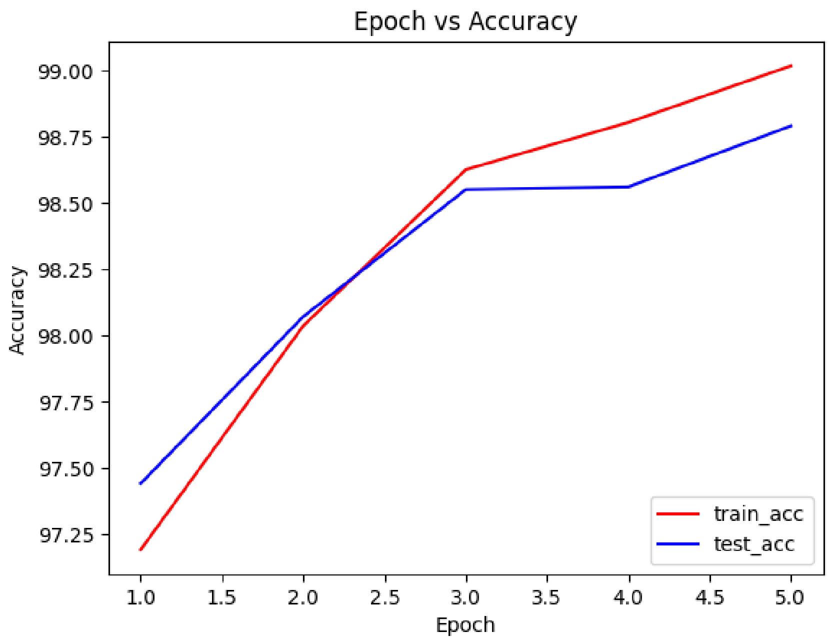
The plots below are the accuracy vs epoch plots for each setting and for one of the 5 runs



Setting 1) train batch size = 32, lr = 0.001, momentum = 0.6



Setting 2) train batch size = 32, lr = 0.01, momentum = 0.8



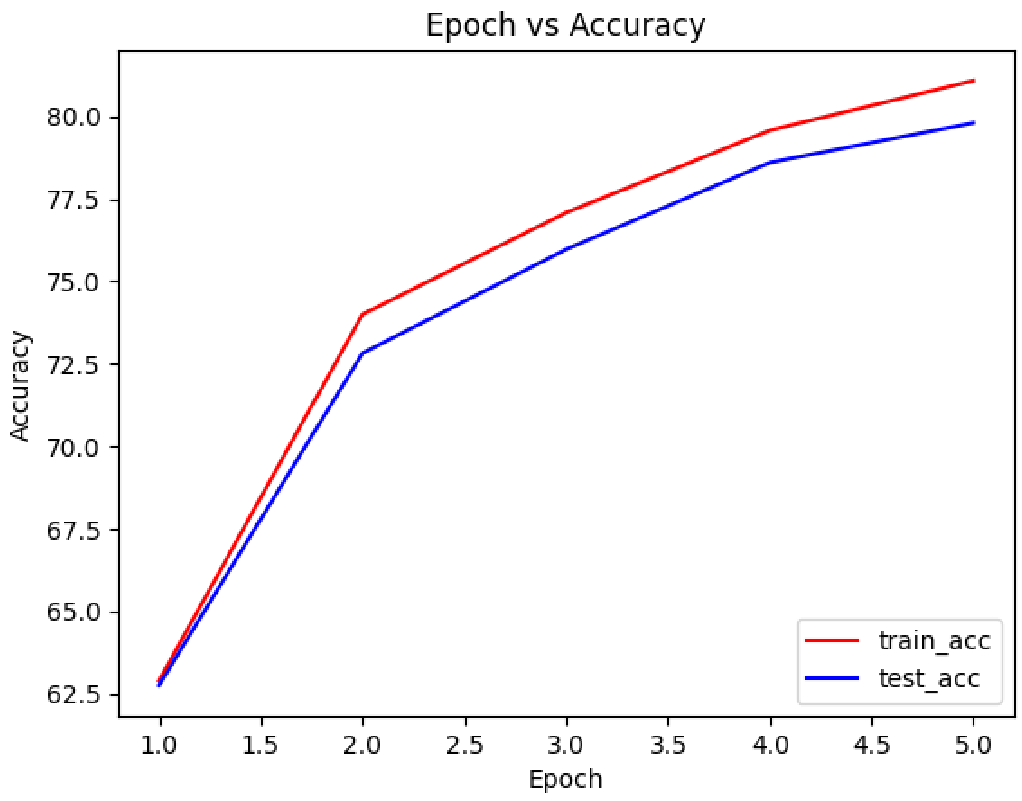
Setting 3) train batch size = 64, lr = 0.01, momentum = 0.8

Fashion MNIST dataset

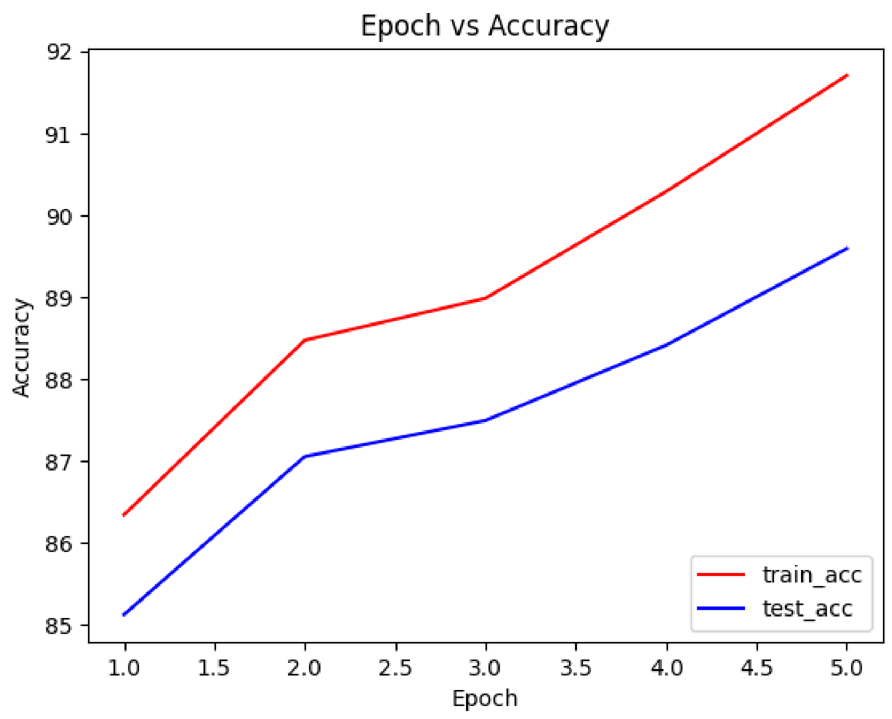
The table below shows the best accuracy, mean accuracy and standard deviation of accuracy for 5 runs with each run going up to 5 epochs for 3 different settings of the hyperparameters namely train batch size, learning rate, and momentum. The statistics are reported on testing accuracy.

Hyper parameter setting	Best accuracy (5 runs)	Mean accuracy (5 runs)	STD accuracy (5 runs)
Train batch size = 32 Learning rate = 0.001 Momentum = 0.6	80.64%	73.256%	5.666
Train batch size = 32 Learning rate = 0.01 Momentum = 0.8	89.66%	87.430%	1.672
Train batch size = 64 Learning rate = 0.01 Momentum = 0.8	88.71%	85.874%	2.554

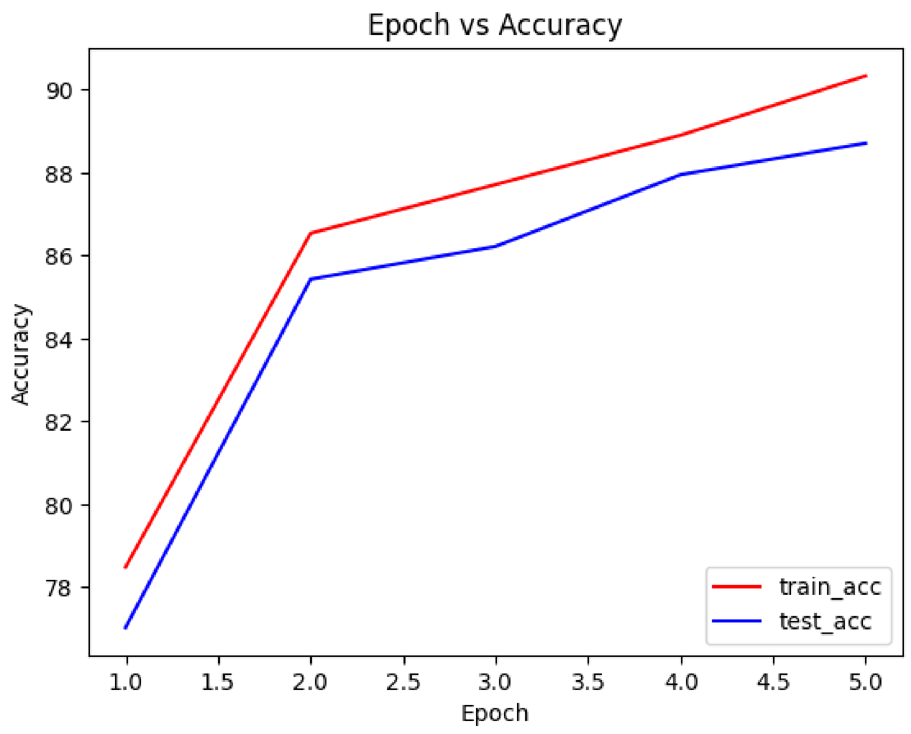
The plots below are the accuracy vs epoch plots for each setting and for one of the 5 runs



Setting 1) train batch size = 32, lr = 0.001, momentum = 0.6



Setting 2) train batch size = 32, lr = 0.01, momentum = 0.8



Setting 3) train batch size = 64, lr = 0.01, momentum = 0.8

CIFAR-10 dataset

The table below shows the best accuracy, mean accuracy and standard deviation of accuracy for 5 runs for 3 different settings of the hyperparameters namely train batch size, learning rate, epochs and momentum. The statistics are reported on testing accuracy.

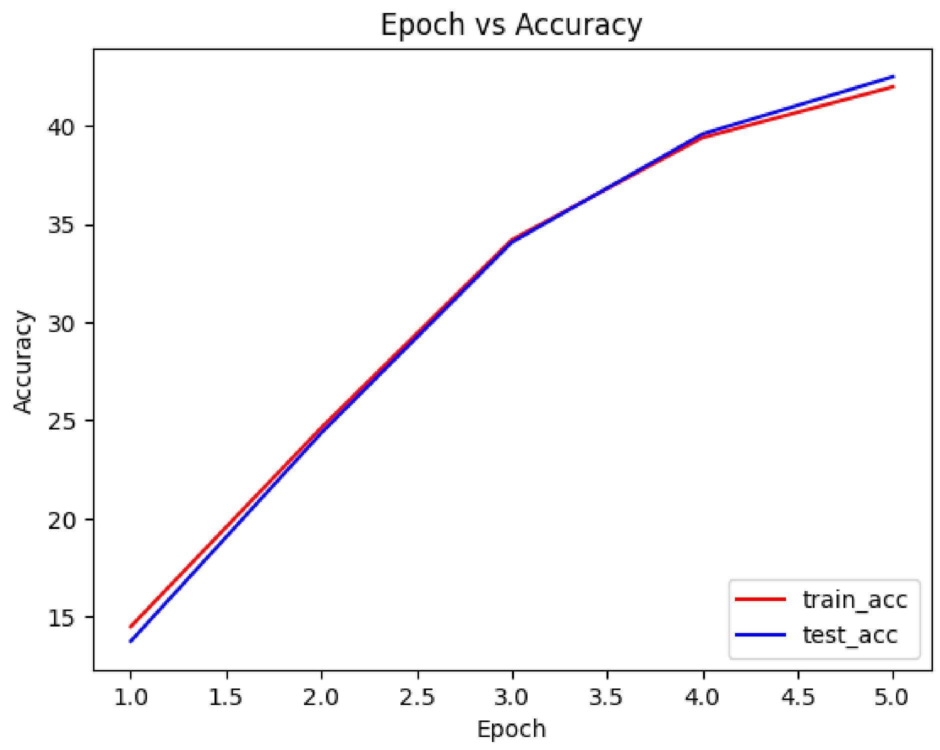
Hyper parameter setting	Best accuracy (5 runs)	Mean accuracy (5 runs)	STD accuracy (5 runs)

Train batch size = 32
Learning rate = 0.001
Momentum = 0.6
Epochs = 5
	42.52%	30.482%	9.485

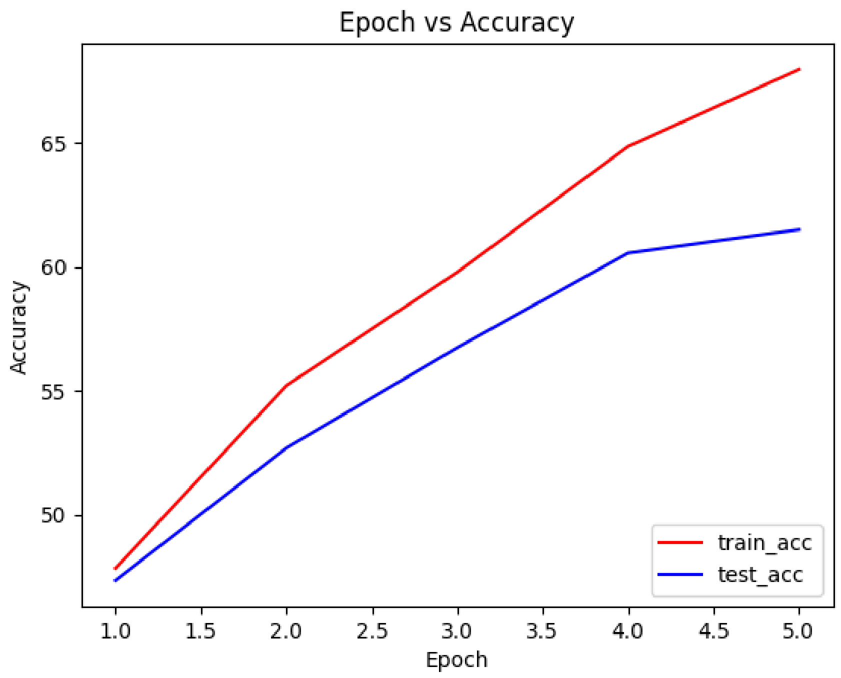
Train batch size = 32
Learning rate = 0.01
Momentum = 0.8
Epochs = 5
	61.5%	55.692%	4.352

Train batch size = 64
Learning rate = 0.008
Momentum = 0.9
Epochs = 10
	65.6%	58.953%	5.214

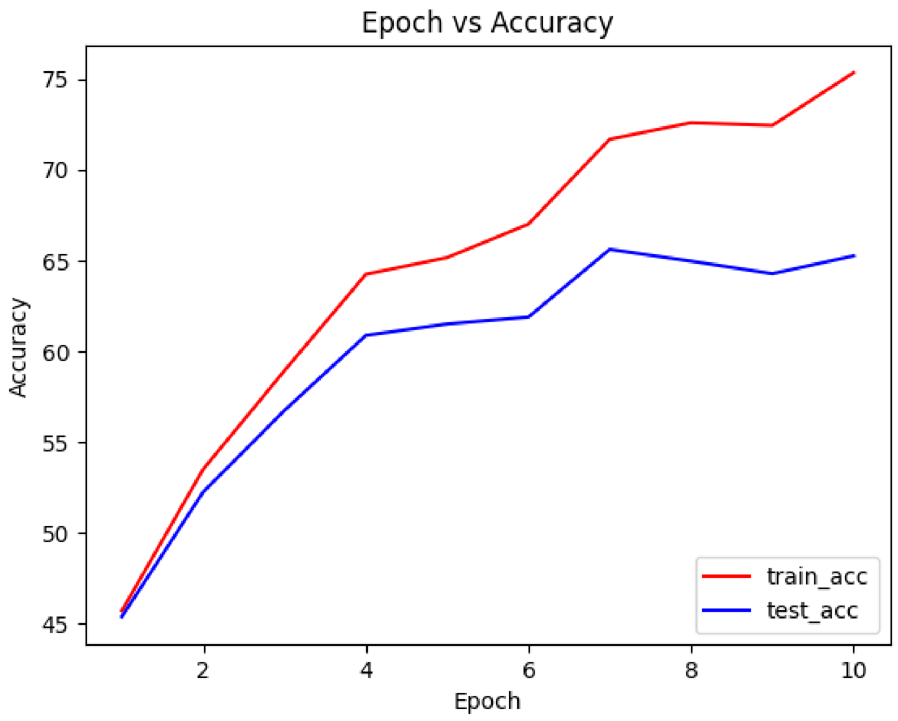
The plots below are the accuracy vs epoch plots for each setting and for one of the 5 runs



Setting 1) train batch size = 32, lr = 0.001, momentum = 0.6, epochs = 5



Setting 2) train batch size = 32, lr = 0.01, momentum = 0.8, epochs = 5

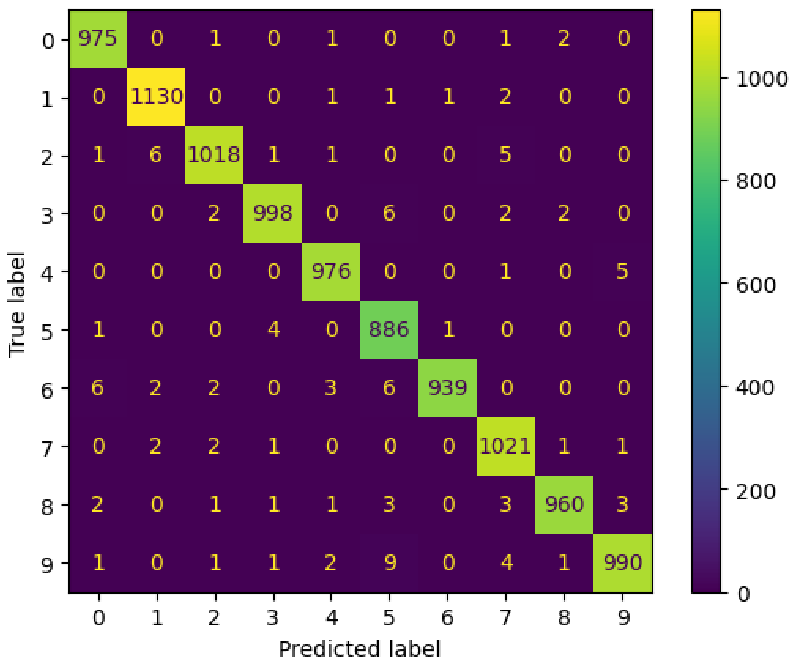


Setting 3) train batch size = 64, lr = 0.008, momentum = 0.9, epochs = 10

1c) Evaluation and Ablation Study

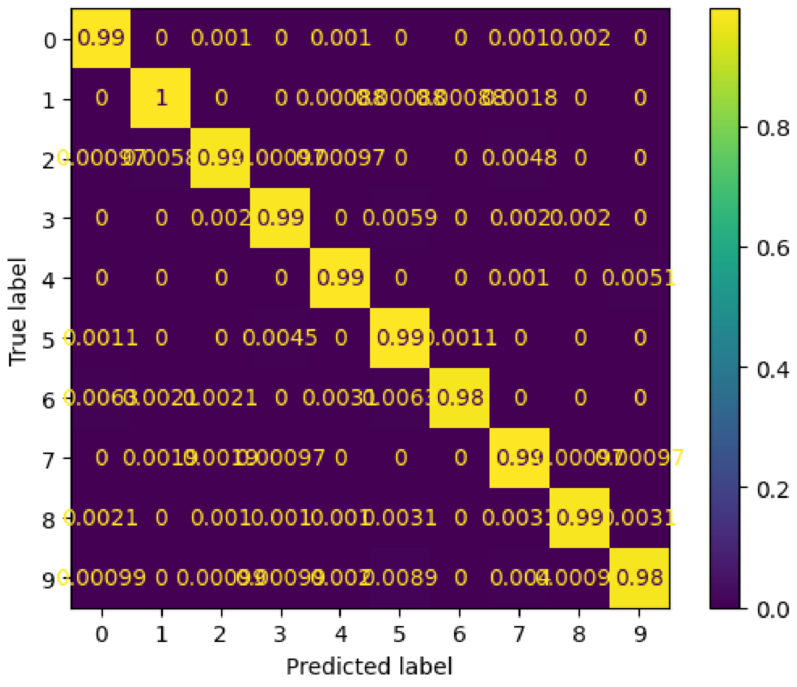
MNIST dataset

Confusion matrix:



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	975	0	1	0	1	0	0	1	2	0
1	0	1130	0	0	1	1	1	2	0	0
2	1	6	1018	1	1	0	0	5	0	0
3	0	0	2	998	0	6	0	2	2	0
4	0	0	0	0	976	0	0	1	0	5
5	1	0	0	4	0	886	1	0	0	0
6	6	2	2	0	3	6	939	0	0	0
7	0	2	2	1	0	0	0	1021	1	1
8	2	0	1	1	1	3	0	3	960	3
9	1	0	1	1	2	9	0	4	1	990

Un-normalized confusion matrix



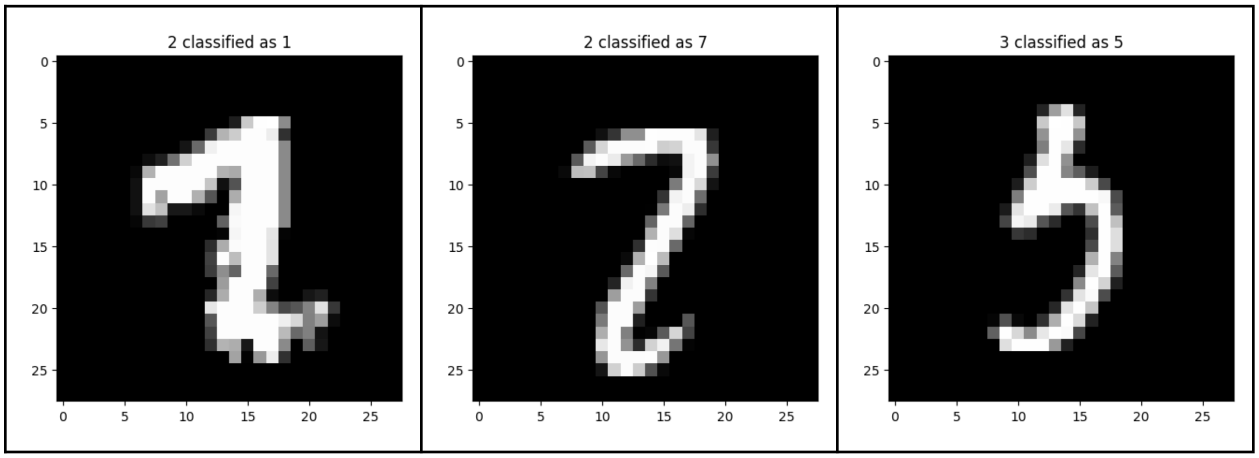
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	0.99	0	0.001	0	0.001	0	0	0.001	0.002	0
1	0	1	0	0	0.00088	0.00088	0.00088	0.0018	0	0
2	0.00097	0.0058	0.99	0.00097	0.00097	0	0	0.0048	0	0
3	0	0	0.002	0.99	0	0.0059	0	0.002	0.002	0
4	0	0	0	0	0.99	0	0	0.001	0	0.0051
5	0.0011	0	0	0.0045	0	0.99	0.0011	0	0	0
6	0.0063	0.0021	0.0021	0	0.0031	0.0063	0.98	0	0	0
7	0	0.0019	0.0019	0.00097	0	0	0	0.99	0.00097	0.00097
8	0.0021	0	0.001	0.001	0.001	0.0031	0	0.0031	0.99	0.0031
9	0.00099	0	0.00099	0.00099	0.002	0.0089	0	0.004	0.00099	0.98

Normalized confusion matrix

Top 3 confused pairs as observed from the confusion matrix above are as given below:

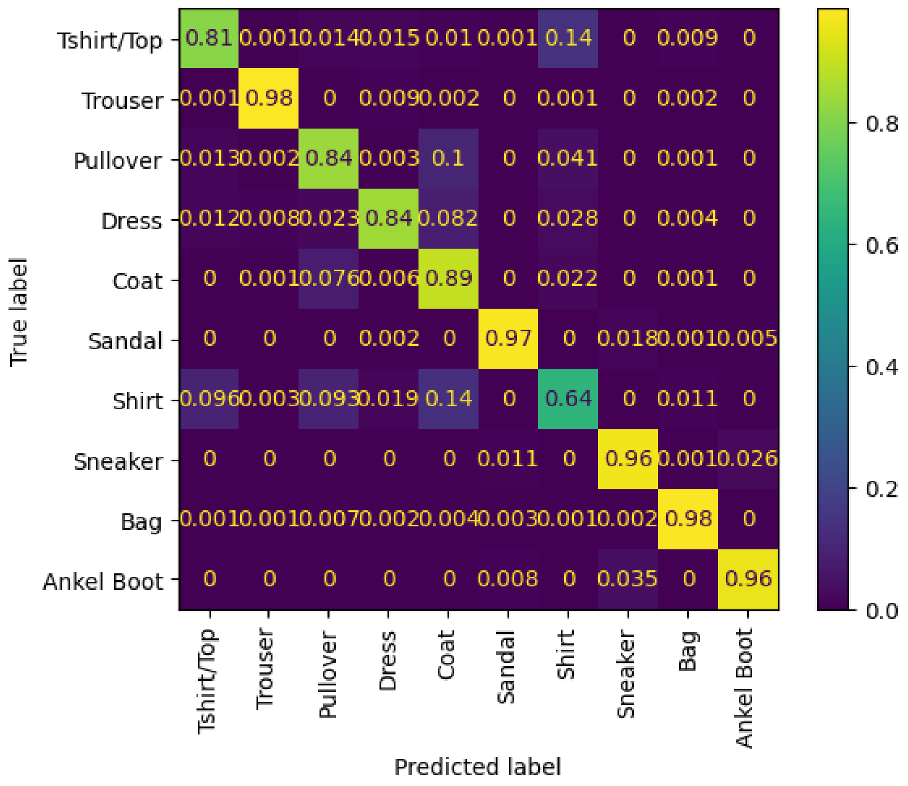
- 2 classified as 1
- 2 classified as 7
- 3 classified as 5

One sample image for each of the above are shown below:



Fashion MNIST dataset

Confusion matrix:

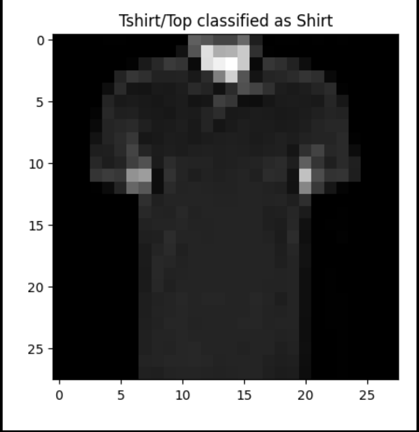


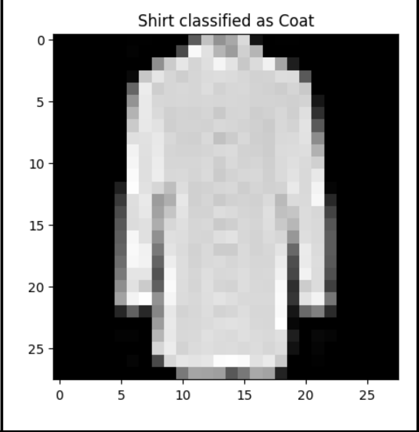
	Tshirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankel Boot
Tshirt/Top	0.81	0.001	0.014	0.015	0.01	0.001	0.14	0	0.009	0
Trouser	0.001	0.98	0	0.009	0.002	0	0.001	0	0.002	0
Pullover	0.013	0.002	0.84	0.003	0.1	0	0.041	0	0.001	0
Dress	0.012	0.008	0.023	0.84	0.082	0	0.028	0	0.004	0
Coat	0	0.001	0.076	0.006	0.89	0	0.022	0	0.001	0
Sandal	0	0	0	0.002	0	0.97	0	0.018	0.001	0.005
Shirt	0.096	0.003	0.093	0.019	0.14	0	0.64	0	0.011	0
Sneaker	0	0	0	0	0	0.011	0	0.96	0.001	0.026
Bag	0.001	0.001	0.007	0.002	0.004	0.003	0.001	0.002	0.98	0
Ankel Boot	0	0	0	0	0	0.008	0	0.035	0	0.96

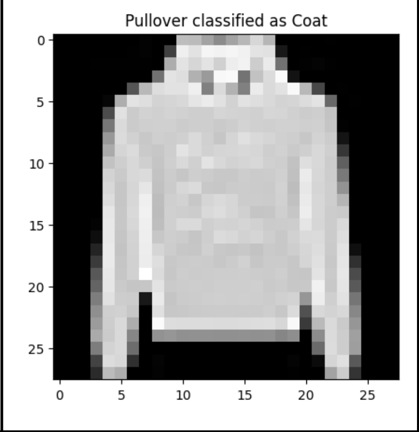
Top 3 confused pairs as observed from the confusion matrix above are as given below:

1) Tshirt/Top classified as Shirt
2) Shirt classified as Coat
3) Pullover classified as Coat

One sample image for each of the above are shown below:

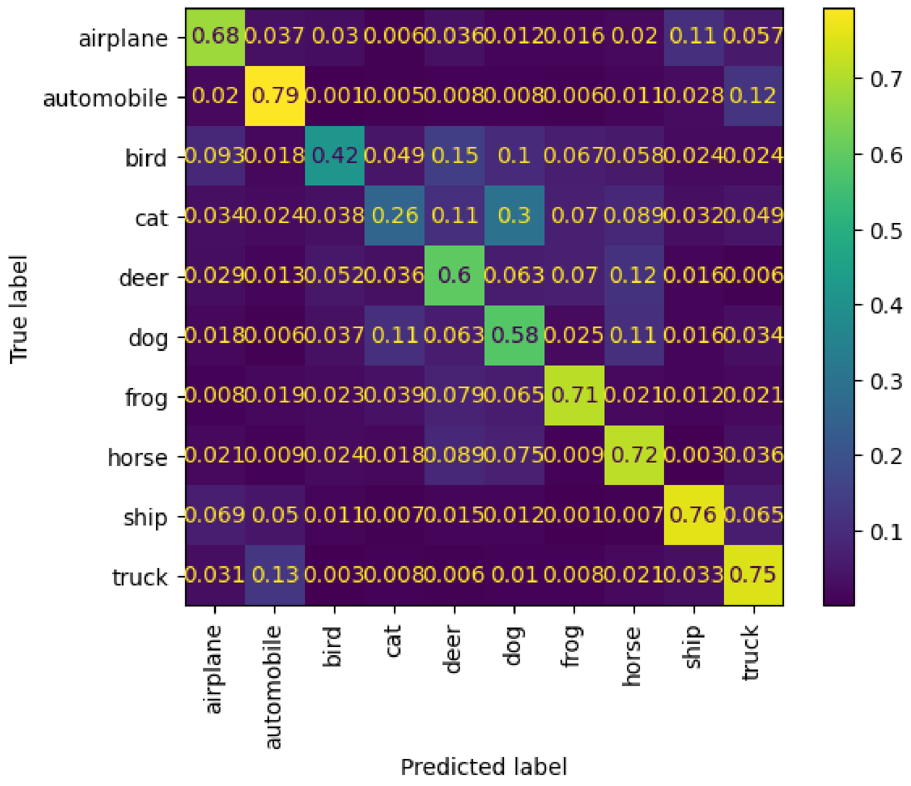






CIFAR-10 dataset

Confusion matrix:

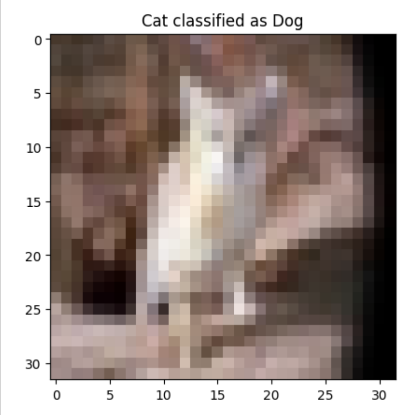


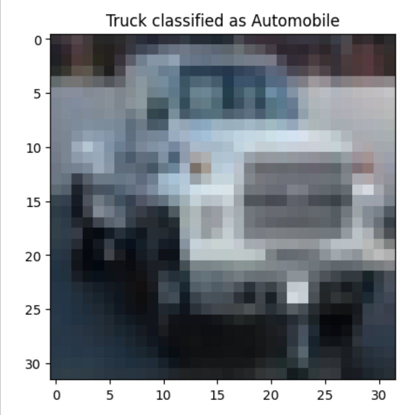
	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0.68	0.037	0.03	0.006	0.036	0.012	0.016	0.02	0.11	0.057
automobile	0.02	0.79	0.001	0.005	0.008	0.008	0.006	0.011	0.028	0.12
bird	0.093	0.018	0.42	0.049	0.15	0.1	0.067	0.058	0.024	0.024
cat	0.034	0.024	0.038	0.26	0.11	0.3	0.07	0.089	0.032	0.049
deer	0.029	0.013	0.052	0.036	0.6	0.063	0.07	0.12	0.016	0.006
dog	0.018	0.006	0.037	0.11	0.063	0.58	0.025	0.11	0.016	0.034
frog	0.008	0.019	0.023	0.039	0.079	0.065	0.71	0.021	0.012	0.021
horse	0.021	0.009	0.024	0.018	0.089	0.075	0.009	0.72	0.003	0.036
ship	0.069	0.05	0.011	0.007	0.015	0.012	0.001	0.007	0.76	0.065
truck	0.031	0.13	0.003	0.008	0.006	0.01	0.008	0.021	0.033	0.75

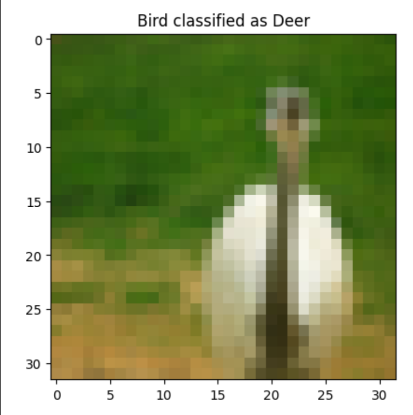
Top 3 confused pairs as observed from the confusion matrix above are as given below:

1) Cat classified as Dog
2) Truck classified as Automobile
3) Bird classified as Deer

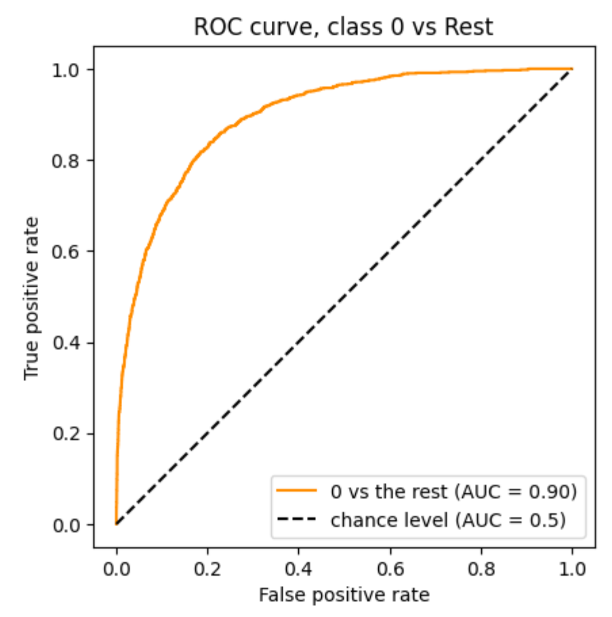
One sample image for each of the above are shown below:

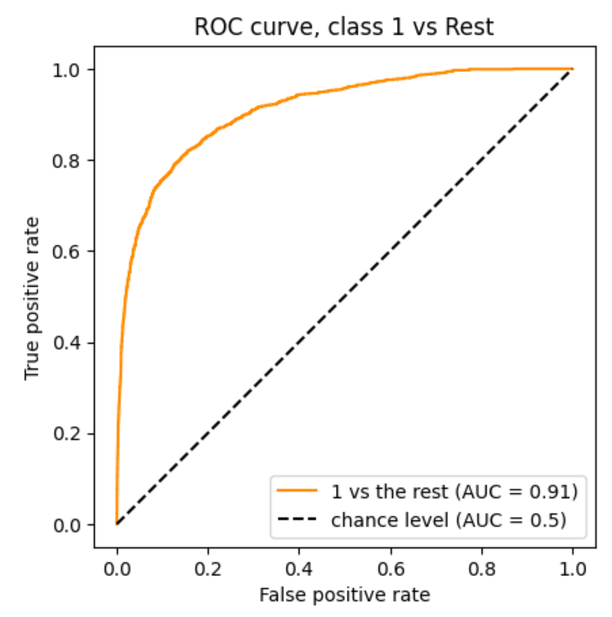


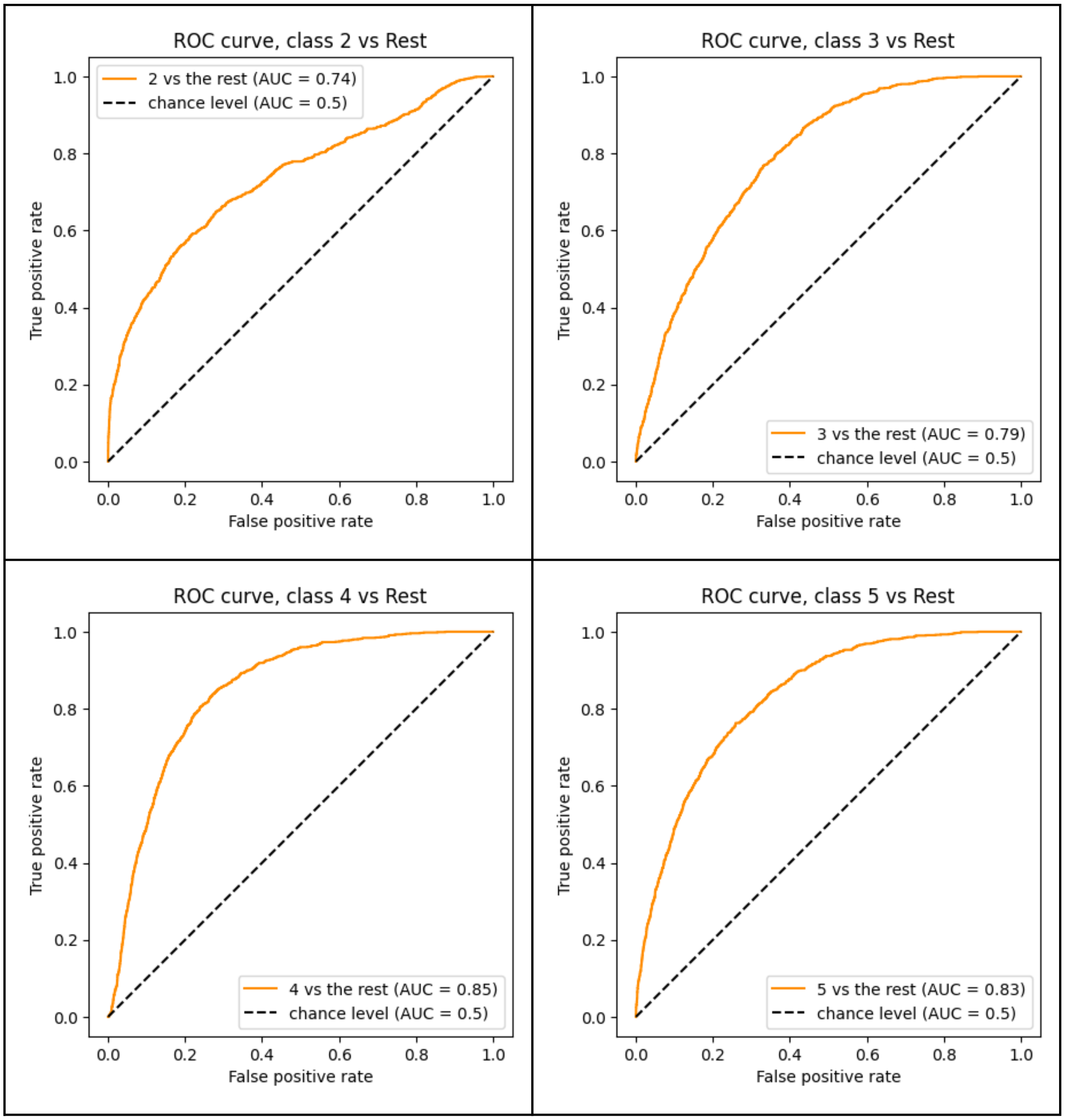


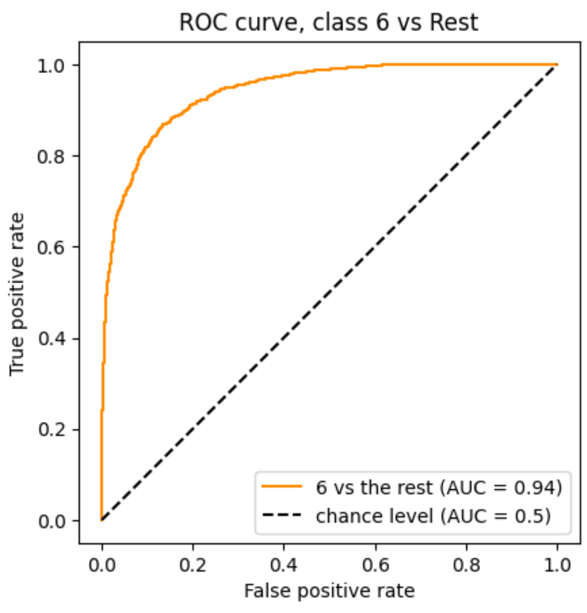


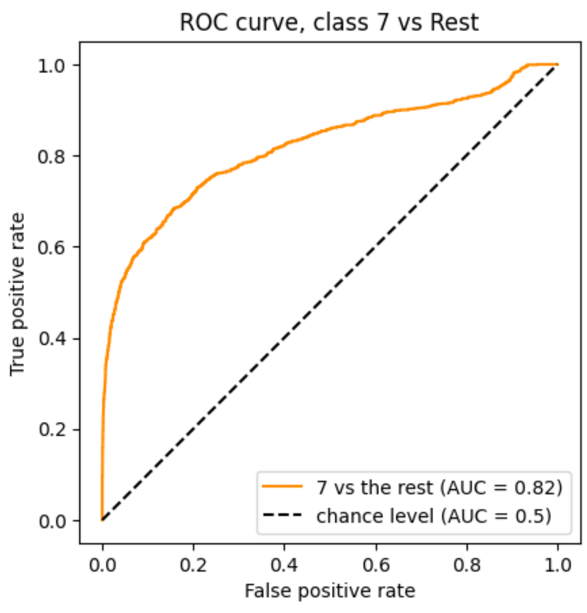
ROC curves for all classes of CIFAR-10 datasets

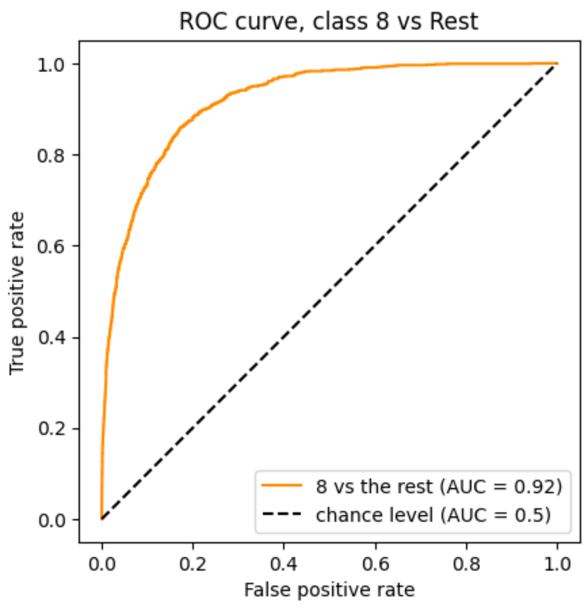


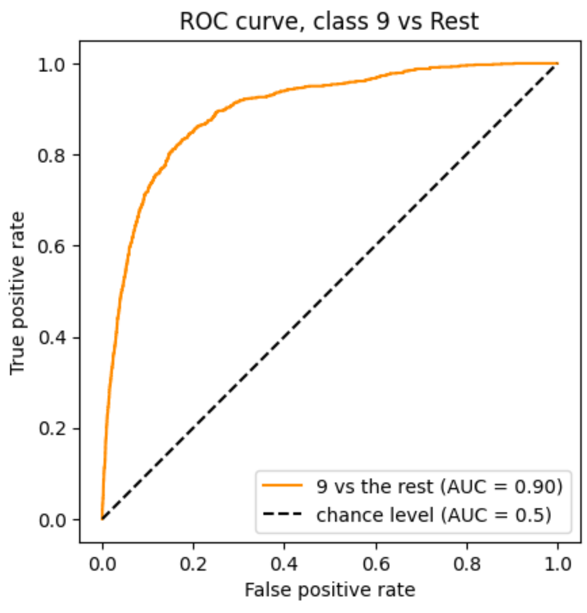



ROC curve, class 2 vs Rest
2 vs the rest (AUC = 0.74)
chance level (AUC = 0.5)
ROC curve, class 3 vs Rest
3 vs the rest (AUC = 0.79)
chance level (AUC = 0.5)
ROC curve, class 4 vs Rest
4 vs the rest (AUC = 0.85)
chance level (AUC = 0.5)
ROC curve, class 5 vs Rest
5 vs the rest (AUC = 0.83)
chance level (AUC = 0.5)
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0









AUC values for all classes of CIFAR-10 Dataset

Class	AUC value
0 vs rest	0.8965671111111112
1 vs rest	0.9109940000000001
2 vs rest	0.7362815555555555

3 vs rest	0.7897602222222222
4 vs rest	0.8476486666666667
5 vs rest	0.8304432222222221
6 vs rest	0.9419877777777778
7 vs rest	0.8184359999999999
8 vs rest	0.9199746666666666
9 vs rest	0.8975497777777778

Count of class labels:

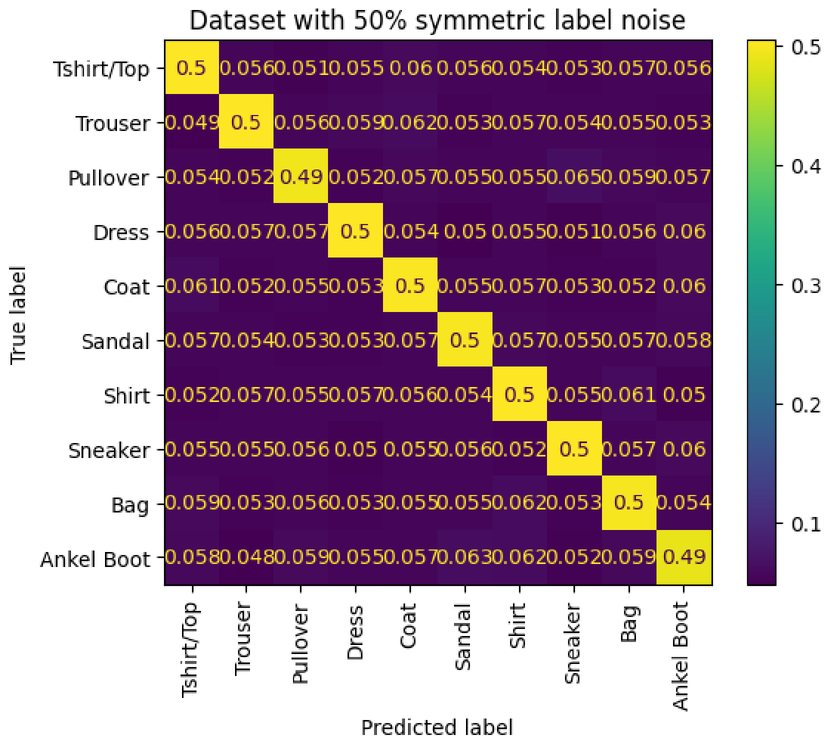
count of class 0: 1000
count of class 1: 1000
count of class 2: 1000
count of class 3: 1000
count of class 4: 1000
count of class 5: 1000
count of class 6: 1000
count of class 7: 1000
count of class 8: 1000
count of class 9: 1000

Since all the classes have equal number of labels we can simply take the mean to calculate the average AUC for the CIFAR-10 dataset

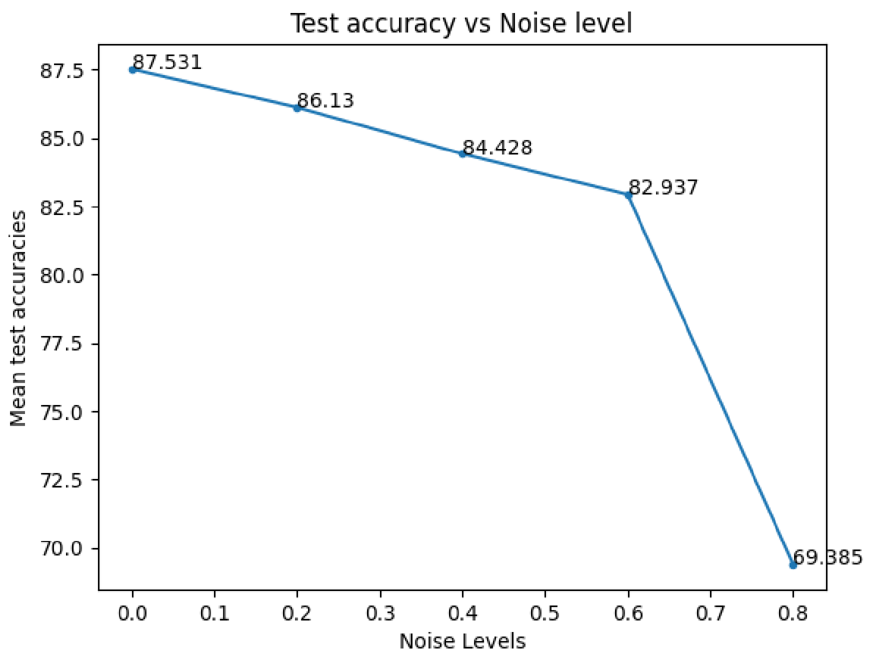
Average AUC for CIFAR-10 dataset : 0.859

1d) Classification with Noisy data

Confusion matrix with ϵ = 50%



Testing accuracy vs ϵ plot:



Discussion

1a) CNN architecture

1) **Fully connected layers**: In the LeNet-5 there are 3 fully connected layers in the end. We say a layer is fully connected if all the nodes are connected with all the nodes of the next layer as well as all the nodes of the previous layer, the first fully connected layer brings down the feature dimension from 400 to 120, the second fully connected layer brings down the dimensions further from 120 to 84 and finally the third layer brings down the dimensions from 84 to 10 corresponding to 10 classes.

   These layers perform the classification or regression tasks based on the features extracted from the previous max pooling and convolutional layers. In LeNet5 the fully connected layer performs the classification task.

   Convolutional layers: In LeNet-5 there are 2 convolutional layers. A convolutional layer can be thought of as a set of learnable filters where each filter is convolved with the input image to extract information from it. In LeNet 5 the first convolutional layer uses 6 filters of size 5x5 and the second convolutional layer uses 16 filters of size 5x5.

   This is in a way similar to Laws' filters that we used during texture classification problems. The main difference is that the filters used in convolutional layers are adaptive. The filter coefficients can be thought of as weights which are optimized by backpropagation and optimization algorithms like stochastic gradient descent. In case of Laws' filters the filter weight or coefficient are fixed. These make the convolutional layers efficient in extracting good features from the input images. Also the learning is hierarchical which means that the early layers learn simpler features and deeper layers learn more complex features.

   Max Pooling layer: There are 2 max pooling layers in LeNet5. The main purpose of max pooling layers is to reduce the spatial resolution of input features while retaining the most important features. The input feature map is divided into non overlapping regions from which the maximum of each region is retained. In LeNet 5 the size of the regions is 2x2. It also helps enhance translational invariance of learnt features which means it helps in making the learnt features invariant to small translations to an extent.

   Activation function: In LeNet 5, the convolutional layers and the fully connected layer use ReLu (rectified linear unit) activation function. Activation functions are used in neural networks to introduce non linearity in the network. This is necessary because the non linearity gives the neural networks the ability to approximate any function. This property is called the universal approximation property of neural networks which makes them powerful function approximators of complex functions and relationships and enables them to learn from data.

Softmax function: The softmax function is a mathematical function that maps a real valued input vector to a probability distribution that sums to one, which can be interpreted as the likelihood of each class in a classification problem. It is given by

softmax(x_i) = exp(x_i) / sum(exp(x_j))

where x_i is the i-th element of the input vector, exp is the exponential function, and the denominator is the sum of the exponential functions of all elements in the input vector.

2) Overfitting is model learning is a problem when the model performs extremely well on training data but performs poorly on testing data. Hence the model cannot generalize well. It occurs when the model fits too closely with the training data and learns the noise and outliers in the training data rather than learning the underlying patterns and relationships that are applicable to unseen data.

   One commonly used method of avoiding overfitting that has been used in CNN is drop out regularization. In this method some randomly selected neurons are dropped out (output set to 0) during training so that the model does not rely heavily on only a subset of neurons and learns more robust features from the data

3) Difference between ReLU, Leaky ReLU and ELU

	ReLU	Leaky ReLU	ELU
Function formula	Given by: f(x) = max(0, x)	Given by: f(x) = max(0.01*x, x)	Given by: f(x) = x if x > 0 $\alpha (e^x - 1)$ if x <=0
Range of output values	0 for all input < 0 and linear output for positive values	Small non zero output for input < 0	Smooth graph approach to 0 for inputs < 0
Continuity	Piecewise continuous, it is discontinuous at 0	This is continuous function	This is continuous function
Computational complexity	Simpler, easy to compute	Moderately easy to compute	Expensive to compute in large model

4) **L1 Loss:** This is also called mean absolute error (MAE) loss and is computed by taking the sum of absolute differences between the predicted and actual values and dividing by the total number of values.

Application: L1 loss is commonly used in regression problems when there are outliers present in the data as it is less sensitive to outliers than L2 loss. It is also used in regularization. A linear regression using L1 regularization is called Lasso regression.

MSE Loss: The mean squared error loss is computed by taking the sum of squared differences between the actual and predicted values and dividing by the total number of values.

Application: MSE loss is also used in regression problems but is sensitive to data with outliers because we are taking the squared differences. It could be used over L1 loss when there is a small amount of noise in the data and there are no outliers because MSE loss will penalize the loss function more and is more sensitive to small amounts of noise.

Binary Cross Entropy Loss: This is given be the following expression:

$$L(y, p) = -[y * \log(p) + (1 - y) * \log(1 - p)]$$

where y is the true label (either 0 or 1) and p is the predicted probability of the positive class.
BCE loss measures how close the predicted probability is to the actual value in terms of information content. Closer the predicted probability to the actual label lower is the BCE loss.

Application: This is commonly used in binary classification problems because for two class problems BCE is a simple loss that is easy to compute and interpretable.

1b) Compare classification performance of different datasets

1) MNIST and Fashion MNIST dataset: The hyperparameters that were considered for tuning were training batch size, learning rate and momentum. The number of epochs was set to a constant of 5.
   For both the datasets the best values for these parameters were observed to be the following:
   Train batch size: 32
   Learning Rate: 0.01
   Momentum: 0.8

   For the CIFAR-10 dataset all the parameters stated above with the addition of number of epochs are used for tuning the model. The best values of these parameters are as give below:
   Train batch size: 64
   Learning Rate: 0.008
   Momentum: 0.9
   Epochs: 10

Effects of these parameters on model performance:

- Training batch size: There is a tradeoff between larger and smaller batch sizes, larger batch size can help speed up the training process but the weight updates are less frequent whereas with smaller training batches the weight updates are more frequent and hence could lead to higher model accuracy but the training time is increased.
- Learning rate: The convergence of the model largely depends on the learning rate. With a small learning rate the model may not converge to a minimum but we might reach the max number of iterations which is also used inorder to stop the iterations in the training process. With a large learning rate the model may initially make good improvements towards the minimum but then bounce back and forth not reaching the global minimum at all.
- Momentum: this is often used to accelerate the optimization process during the training process of the neural network. It also helps to come out from getting stuck in local minima by pushing the gradient allowing to jump off from a valley and reaching a more optimal minimum point

Observations:
MNIST and Fashion MNIST- it was observed that by increasing the learning rate from 0.001 to 0.01 there was a good increase in accuracy. Further increase the batch size leads to slight decrease in accuracy.
CIFAR-10 dataset - for the same parameters it was observed that the model needed more epochs to achieve the target accuracy of 63% - 65%.

5) The best accuracy achieved for each dataset are as given below:
MNIST: 99.04%
Fashion MNIST: 89.66%
CIFAR10: 65.6%

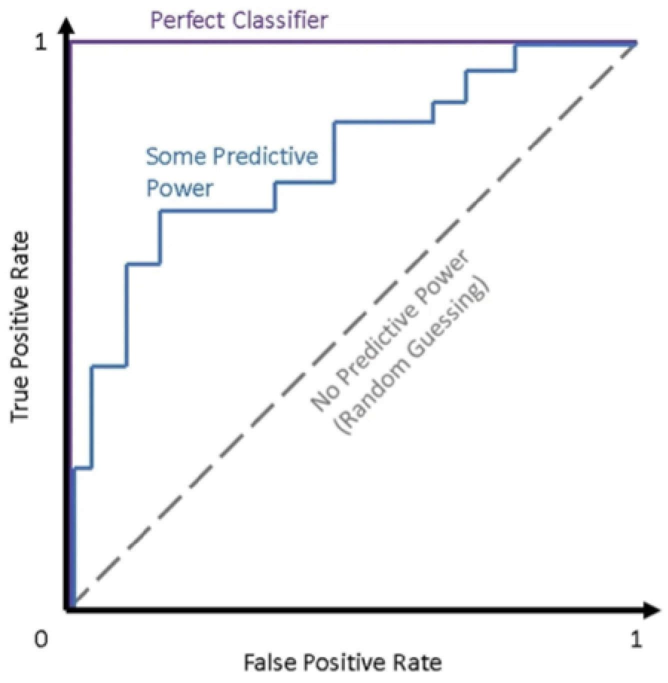
The fundamental reason for this could be that the CNN architecture was specifically designed for the purpose of recognising handwritten digits, hence it performs extremely well on the MNIST dataset. It performs fairly well on the fashion MNIST because the dataset is similar in terms of image properties to MNIST, Fashion MNIST also has only gray scale images. The model performs poorly on CIFAR10 because the CIFAR 10 dataset has an RGB image unlike the MINST and Fashion MNIST and also the objects that we are trying to classify are fairly varied even when there are only 10 classes. For MNIST and Fashion MNIST we are only focusing on either digits or clothing.

1c) Evaluation and Ablation study

1) The confusion matrix is as shown in the result section above. It is observed that the top 3 confused classes were cats and dogs, automobiles and trucks and birds and deers. Perhaps since Dogs and Cats have a kind of similar outline feature like two ears, fur, eyes location the model might be getting confused between the 2 classes. For trucks and automobiles it is very evident because the model did badly on truck images which

were taken from the front face of the truck. It could be said that the front face of trucks and automobiles are very similar when it comes to SUVs and trucks. Where as for deers being confused with birds was an interesting point to be observed, but when we see the images where the model is classifying wrongly we can see that for most of the emu bird images the model is classifying them as deers again this is could be due to the fact that on a very highly level the body structures like a long neck with thin body are similar to both emus and deers.

2) The ROC curves for all the 10 classes of the Fashion MNIST dataset are shown in the Results section above. Unlike accuracy, we can judge the discrimination power of the model for each class using the ROC curves. A straight line in the ROC plot would mean that the model has no discriminant power. An ideal curve would be 1 at 0 and then constant 1 for all values there on, this would be a perfect classifier. In most cases the curve appears to rapidly increase and then taper off to a value of 1 which shows that the model has some discriminant power and is indeed learning something. An illustration is show below:



ROC curve

From the ROC curves we can see that for classes 0, 1, 6, 8, 9 which correspond to airplane, automobile, frog, ship and truck, the model has good discriminant power and could classify these objects more easily than others. For classes 4, 5, 7 which correspond to deer, dog, horse classes the model has moderate discriminant power perhaps the model might be getting confused between cats and dogs or between deers and horses or between deers and emu birds. For classes 2 and 3 corresponding to bird and cat the model is not doing so well when it comes to discriminant power.

3) AUC (area under the curve) is another metric which helps in summarizing the ROC by a single scalar quantity. For AUC = 0.5 we can say the model has no discriminant power. For AUC = 1, it is the ideal case of a perfect classifier and for AUC between 0.5 and 1 we can say that the model has some discriminant power.

1d) Classification with noisy data:

The confusion matrix with 50% symmetric label noise is as shown in the results section above. It can be seen that it follows the given confusion matrix form shown below:

$$\begin{bmatrix} 1-\epsilon & \frac{\epsilon}{2} & \frac{\epsilon}{2} \\ \frac{\epsilon}{2} & 1-\epsilon & \frac{\epsilon}{2} \\ \frac{\epsilon}{2} & \frac{\epsilon}{2} & 1-\epsilon \end{bmatrix}$$

The process used for adding symmetric label noise is as given below:
Step 1: Get the train labels
Step 2: Calculate the proportion of labels whose values are to be changed depending on epsilon value
Step 3: Depending on the proportion of labels to be changed generate random indexes equal to the proportion of labels to be changed. For example if there are 100 labels and we are adding 40% noise then generate 40 random indexes
Step 4: Using these random indexes, change the value of the true labels. Set the value of the true label to another class label randomly.

We obtain the testing accuracy vs noise level epsilon by adding noise levels of 20%, 40%, 60% and 80%. We can expect that the accuracy of the model decreases as the noise in the data increases, and this in fact is the case as seen from the graph. We can see that the testing accuracy plummets as we increase the noise in the training data.