CS6220 HW2: Problem 2 Option 1

Introduction

The code, screenshots, and artifacts are all contained in this Github repo. Please refer to the README.md file in the repo for setting up the conda environment and any other configuration.

Input Analysis

I decided to use the MNIST dataset for this problem. The dataset consists of 70000 images of handwritten digits from 0-9 (equally distributed for each digit.)

Parameter	**Value**
Dataset Storage Size	54.9 MB
Storage Size Per Image	~785B
Input Resolution	28 x 28 Grayscale Image

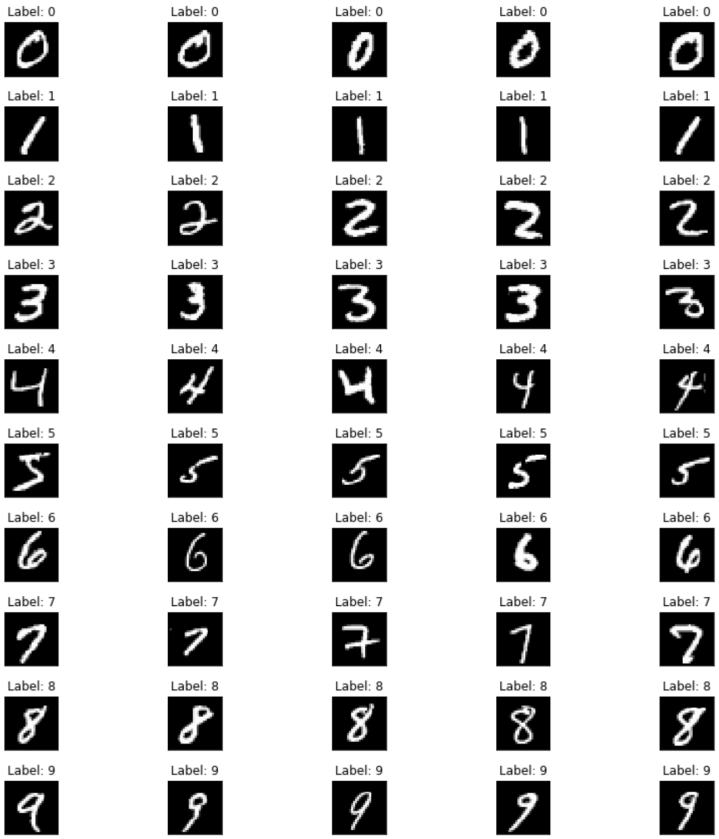


Figure: A visualization of 5 examples from each of the 10 classes in the MNIST dataset.

Train-Test Split
I used a 6:1 train-test ratio as it is the recommended division for the MNIST dataset.

Number of Training Examples	60000
Number of Test Examples	10000

Default Hyperparameter Settings
The following table is the default set of hyperparameters that the base CNN model is trained on.

Hyperparameter	**Value**
Epochs	10
Learning Rate	0.001
Minibatch size	64
Optimizer	Stochastic Gradient Descent
Loss Criterion	Cross Entropy Loss

The Base CNN Model
The Base CNN model consists of a Convolution layer, ReLU layer, another Convolution layer, a Max Pooling layer, and a Fully-Connected layer. The image below shows the parameters of each layer in the model:

```
BaseCNN(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (relu): ReLU()
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=12544, out_features=10, bias=True)
  (softmax): Softmax(dim=None)
)
```

The first convolution layer takes in a single channel (since MNIST images are grayscale) and produces 32 feature maps as its output (32 x 28 x 28.) The output is then passed through the ReLU non-linearity to the next convolution layer. This layer produces 64 feature maps as its output (64 x 28 x 28). The output is then downsampled via a max-pooling layer with a 2x2 kernel that has a stride and zero-padding of 1 each (to maintain an integer output dimension.) The final output of 64 x 14 x 14 is flattened and passed to the fully connected layer which outputs 10 logits (equal to the number of classes.) These logits are passed into a softmax layer to be interpreted as probabilities.

Hyperparameter Configurations

We are asked to choose 2 different hyperparameters and choose 2 settings for each hyperparameter, thereby giving us 4 different configurations. The table below outlines the 2 hyperparameters and their 2 values.

Hyperparameter	**Value**
Learning rate	0.1, 0.01
# Epochs	5, 8

Experiments

Experiment 1: Training the Base CNN on the MNIST Dataset
The first experiment we run will be to train the base CNN architecture on the MNIST dataset under the given parameters.

Statistic	**Value**
Training time	1125.44 seconds
Training Accuracy	0.932
Test time	6.48 seconds
Test Accuracy	0.935
Model Size	565KB

Figure: Training loss curve for the Base CNN Model.

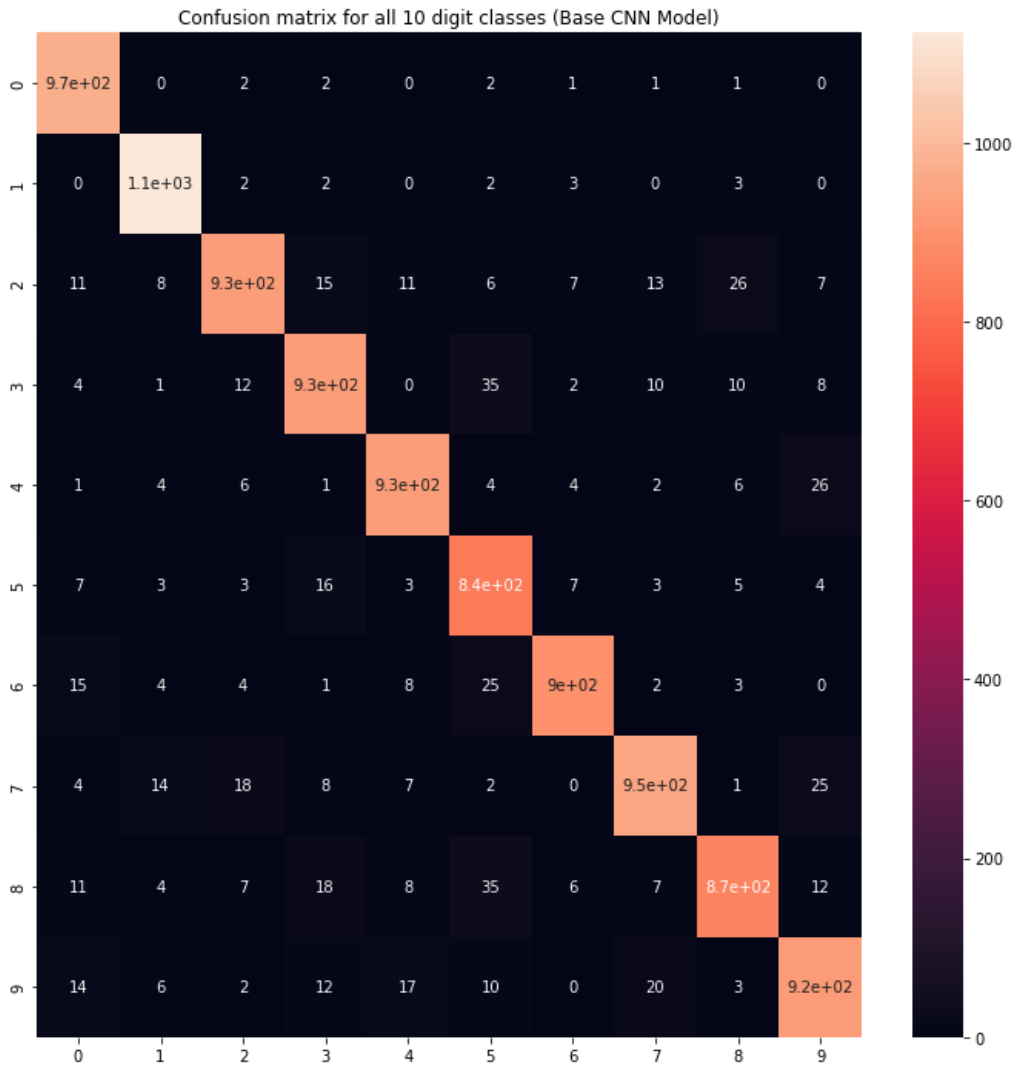


Figure: Confusion matrix for all 10 digit classes predicted by the Base CNN Model.

Experiment 2: Outlier Test on the Base CNN Model

For the 10 outlier examples, I decided to obtain a single batch of data from the Fashion MNIST dataset. The dataset consists of 70000 images of fashion apparel divided into 10 classes. Shown below is a visualization of the examples along with their target labels:

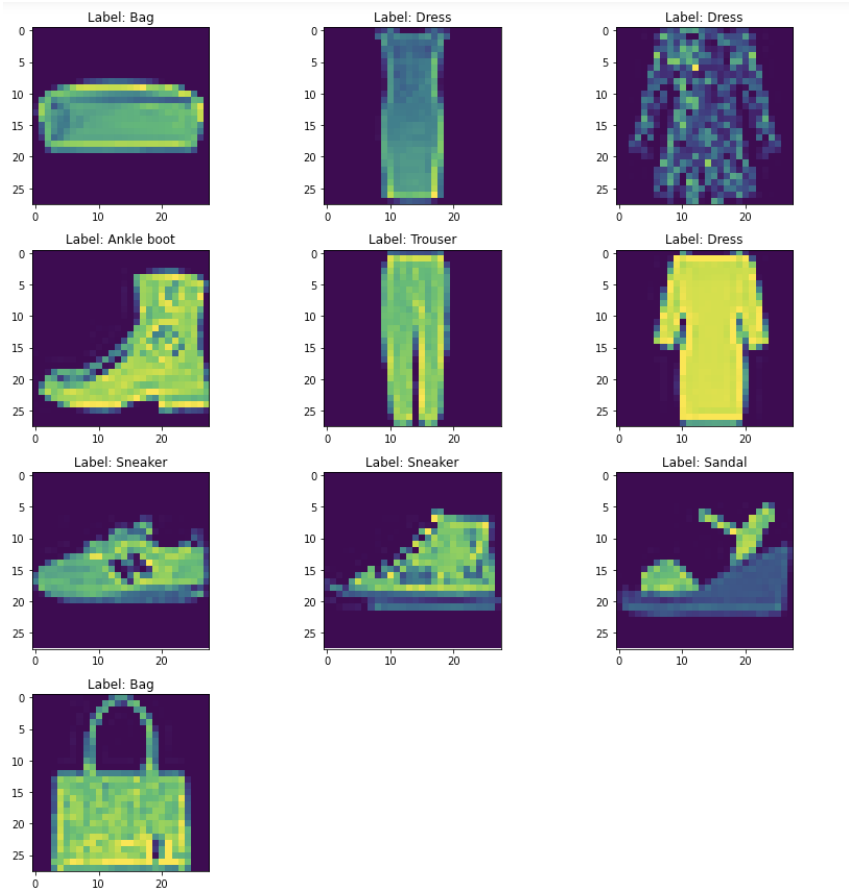
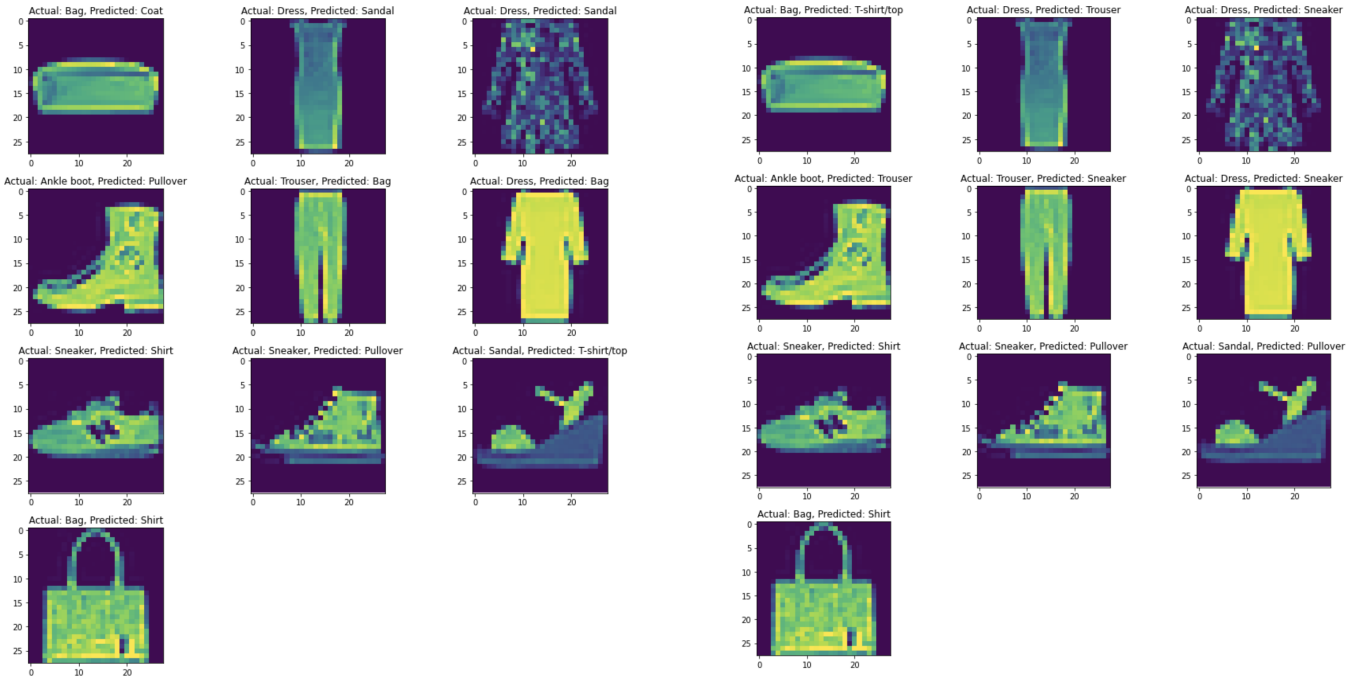
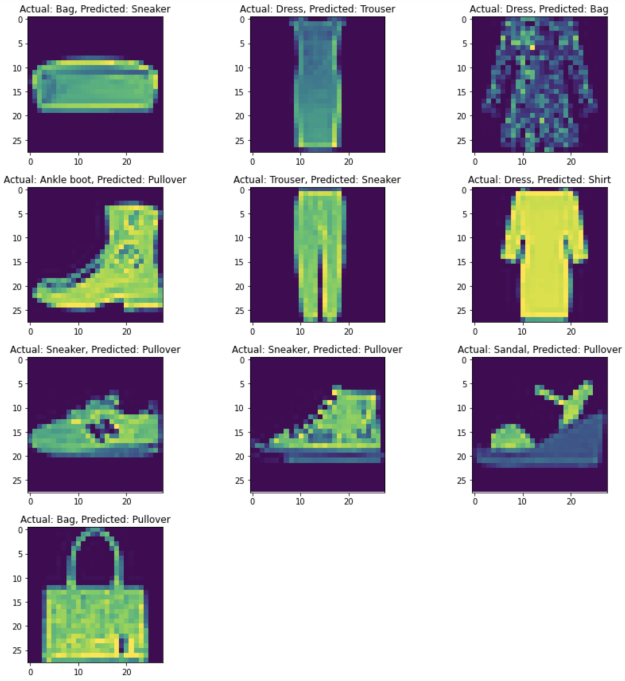


Figure: Examples from the outlier test set that was obtained from the Fashion MNIST dataset.

For the outlier test, I chose to go with option (i). The 3 pretrained models that I chose were: 1. The base CNN trained under the default hyperparameter settings, 2. The base CNN trained with a learning rate of 0.1 for 5 epochs, and 3. The base CNN trained with a learning rate of 0.1 for 8 epochs. The following 3 figures show the labels vs predictions for the 3 models:





(NOTE: The predictions and labels are in the range [0, 9]. What is shown is the class equivalent of what the model predicts. For instance, 1 is mapped to Trouser.)

From the images above, the models are predicting *different* labels for the same image even though they were trained on the same dataset. For instance, notice the top-right prediction in each of the images. The actual label is **Dress:** the first model thinks it's a **Sandal,** the second model thinks it's a **Sneaker**, and the third model thinks it's a **Bag.** Intuitively, this makes sense: the models used were trained to predict handwritten digits in the range [0, 9] while our outlier images are of various kinds of fashion apparel. Naturally, the model will struggle to classify these outliers into their correct classes. In fact, of the 30 prediction queries (3 x 10), *none* of them were correct! The models learnt representations for a dataset from an unrelated domain, and thus do not possess any of the information on the outlier's representations. As for the models predicting different probability vectors, each of the models were trained under different learning rate and epoch combinations. A different learning rate implies that the gradient updates in each iteration will be different. And since the second and third model have a different number of epochs, their parameters will go through less/more updates. Therefore, the probability vectors, which are a function of the weights of each network, will be different in each network for the same input.

Experiment 3: The Base CNN Model with an LR of 0.1 and # Epochs = 5

Statistic	**Value**
Training time	570.09 seconds
Training Accuracy	0.988

Test time	6.67 seconds
Test Accuracy	0.983
Model Size	565KB

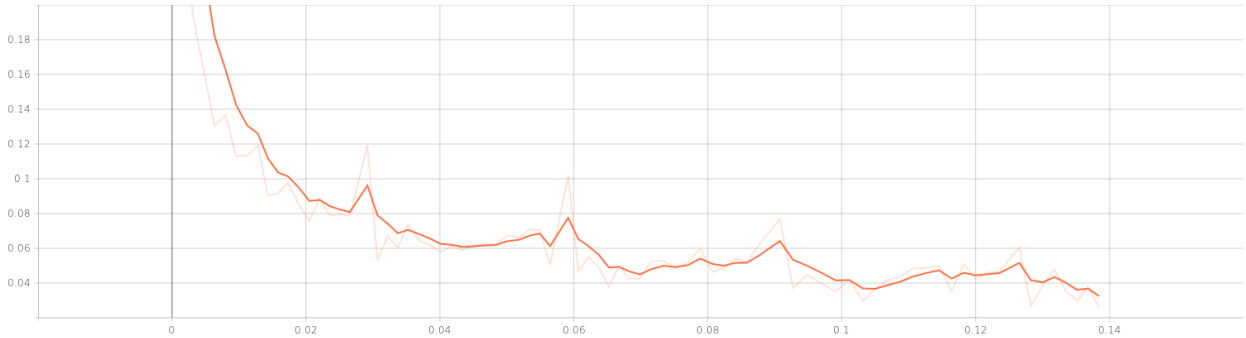


Figure: Training loss for the base CNN model with a learning rate of 0.1 trained for 5 epochs.

Experiment 4: The Base CNN Model with an LR of 0.1 and # Epochs = 8

Statistic	**Value**
Training time	850.39 seconds
Training Accuracy	0.993
Test time	7.01 seconds
Test Accuracy	0.986
Model Size	565KB

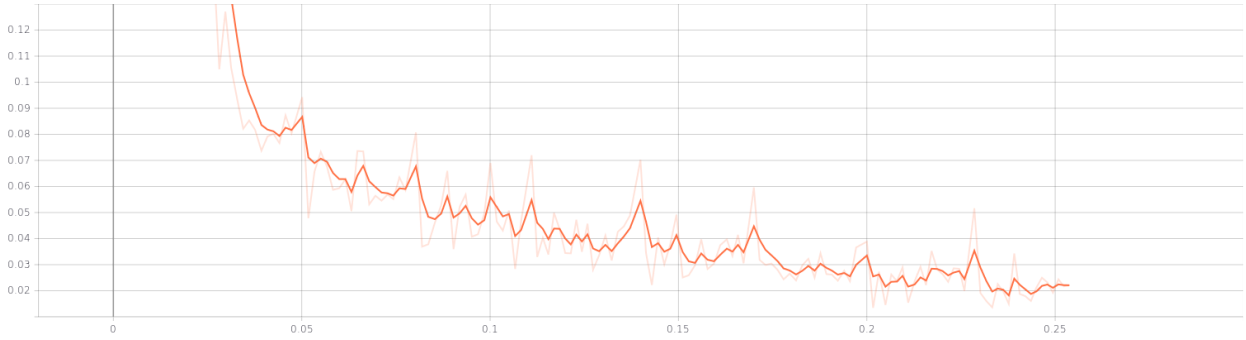


Figure: Training loss for the base CNN model with a learning rate of 0.1 trained for 8 epochs.

Experiment 5: The Base CNN Model with an LR of 0.01 and # Epochs = 5

Statistic	**Value**

Training time	573.23 seconds
Training Accuracy	0.978
Test time	6.51 seconds
Test Accuracy	0.979
Model Size	565KB

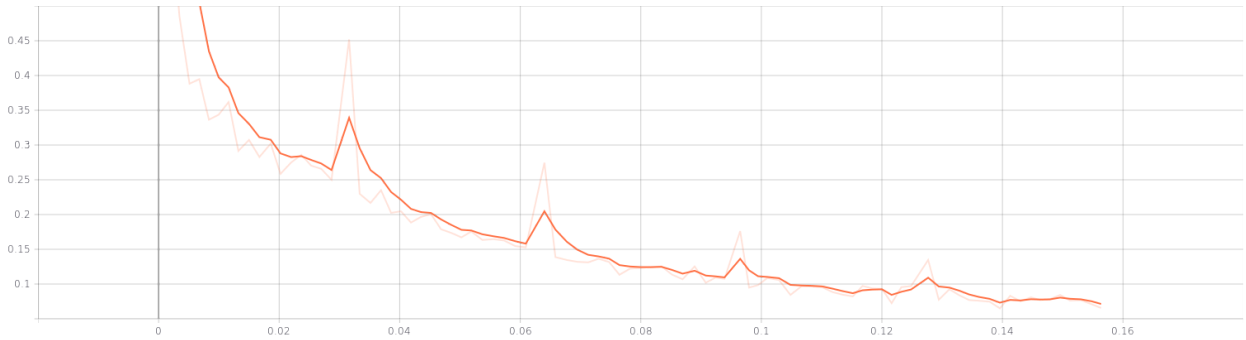


Figure: Training loss for the base CNN model with a learning rate of 0.01 trained for 5 epochs.

Experiment 6: The Base CNN Model with an LR of 0.01 and # Epochs = 8

Statistic	**Value**
Training time	858.88 seconds
Training Accuracy	0.983
Test time	6.57 seconds
Test Accuracy	0.982
Model Size	565KB

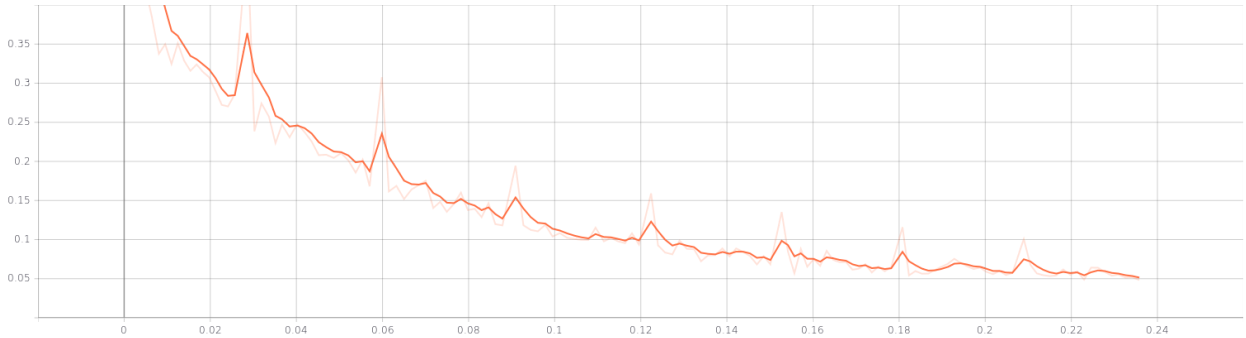
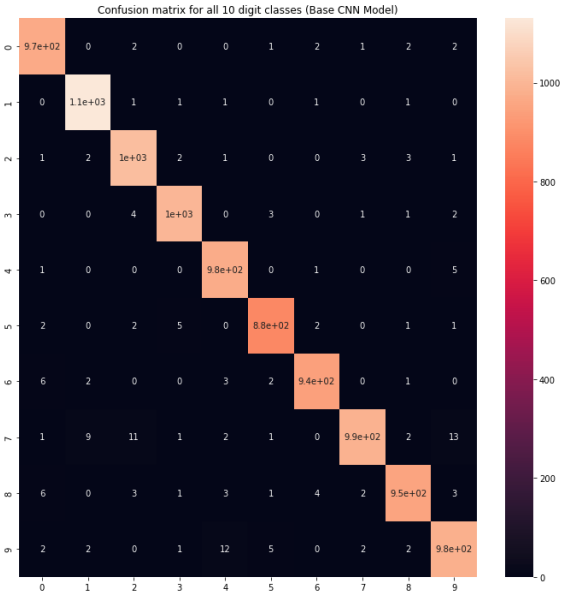
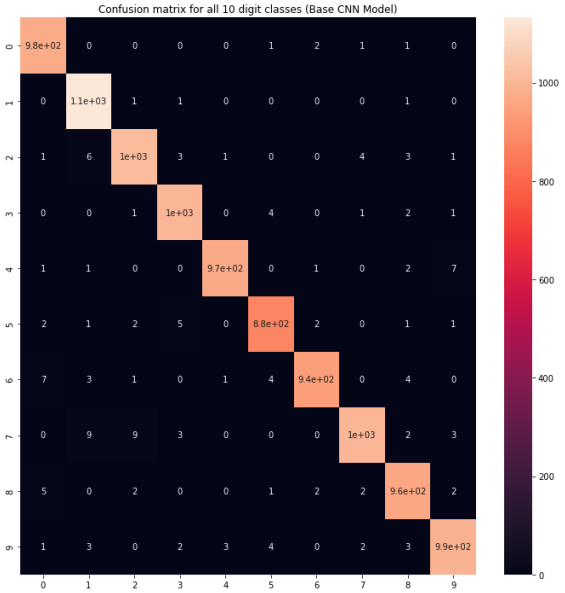


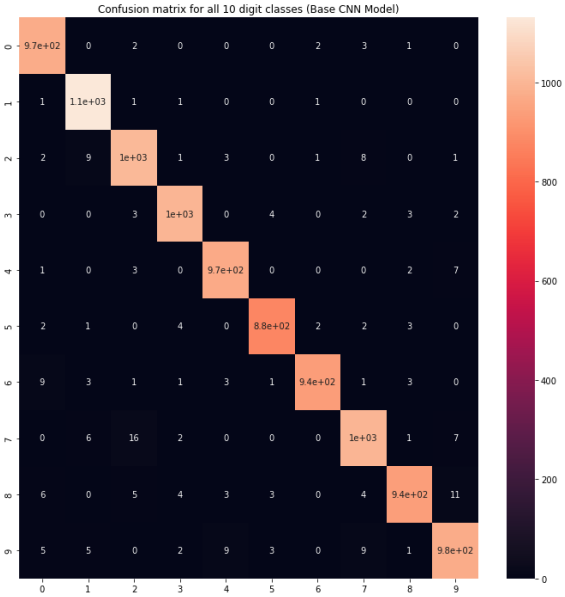
Figure: Training loss for the base CNN model with a learning rate of 0.01 trained for 8 epochs.

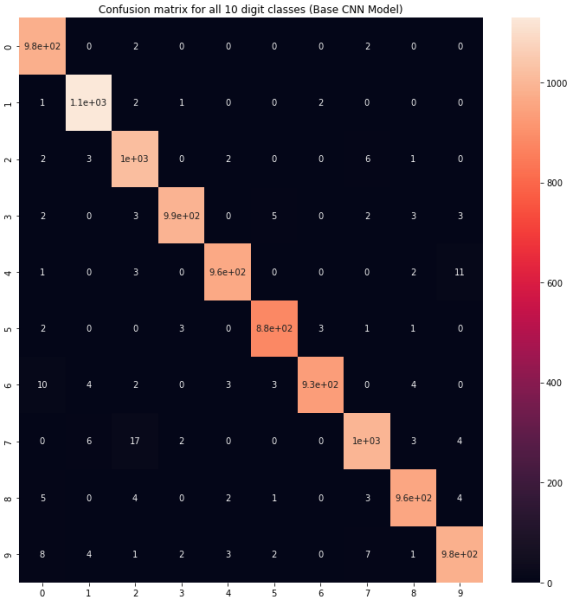
Output Analysis

Model	# Epochs	Learning Rate	Train Time (s)	Train Accuracy	Test Time (s)	Test Accuracy	Model Size
Model 1	10	0.001	1125.44	0.932	6.48	0.935	565KB
Model 2	5	0.1	**570.09**	0.988	6.67	0.983	565KB
Model 3	8	0.1	850.39	**0.993**	7.01	**0.986**	**565KB**
Model 4	5	0.01	573.23	0.978	6.51	0.979	565KB
Model 5	8	0.01	858.87	0.983	6.57	0.982	565KB









The table above compares the 5 models trained using the two hyperparameters that were varied (learning rate and #epochs) and also using their train time and accuracy and test time and accuracy. The models were trained under the following configurations: a batch-size of 64, weights optimized using Stochastic Gradient Descent (SGD), and error quantified using categorical cross-entropy loss. The confusion matrices shown above are for the following configurations: in the top row, the learning rate is 0.1 and the number of epochs are 5 and 8 while the bottom row has a learning rate of 0.01 for 5 and 8 epochs as well. The lighter the color of the cell, the higher the numeric value.

Observation and Analysis

1. Generally, a learning rate of 0.1 is considered too high as the gradient updates would be much steeper and the loss function may begin shooting around erratically and miss the minima. However, we see an exception with the models in this case as the configuration that had a learning rate of 0.1 and was trained for 8 epochs had the highest training and test accuracy. There are some reasons for this: 1. The MNIST dataset is well curated and contains plenty of data for a classification task that is relatively less complex than most. It may not contain a lot of noise in the data, thus making any kind of overfitting less likely. 2. Models are generally trained for tens of epochs at a lower learning rate (like 0.001). But, in this case the higher learning rate trained for a smaller number of epochs may have resulted in the model's loss hitting the global minima without bouncing around. It is less likely to see this in practice.
2. An interesting observation arises from comparing model 2 and model 5. The former is trained for 5 epochs with an LR of 0.1 while the latter is trained for 8 epochs with an LR of 0.01. The premise here is to understand whether it is more effective to set learn *aggressively for a short period of time* (model 2) or learn *passively for a longer period of time* (model 5.) In this case, model 2 outperforms model 5 on the test accuracy by 0.01. Also notice that model 2 took 5 minutes less to train than model 5. Therefore, for MNIST, it is reasonable to conclude that an aggressive learning schedule that aims to complete training quickly produces better results than a more patient learning schedule. Once again, it may be due to the well-curated, noiseless nature of MNIST (it is a benchmark dataset for a reason!) Also, notice that when we do set an aggressive LR of 0.1, training it for 8 epochs instead of 5 only increases the test accuracy by 0.3%. It is important to note that the models are already highly performant and any gains in test accuracy will begin to diminish, but it's interesting to note that the CNN model we use here requires a relatively few number of epochs to begin performing extremely well. This is certainly not the case with CNN-related tasks in the real world. SoTA architectures such as Alexnet and InceptionV4 are trained for tens of hundreds of epochs on ImageNet before they could achieve comparable performance. Therefore, it is imperative that these results are taken with a grain of salt: the more complex the dataset and architectures become, the higher the price to obtain high performance.
3. From all 5 confusion matrices, it is clear that all the models that were trained are high-performing. The numbers on the diagonal are very high and quite similar across the matrices, which means most of the predictions are correct. The number of correct predictions for the digit 1 are especially high for the models. The confusion matrices for

models 3 and 5 (top and bottom right in the image above) show slightly lighter colors on some of the diagonal values than others. This is attributed to the fact that these models perform slightly better on the dataset than the rest (the test accuracy is higher by 1-1.5%.) Nevertheless, the off-diagonal values for all the confusion matrices show extremely dark colors which means that all models rarely make incorrect predictions.