Homework 5: EE569

Ishaan Vasant

6989-5065-37

ivasant@usc.edu

Problem: CNN Training and Its Application to the MNIST Dataset

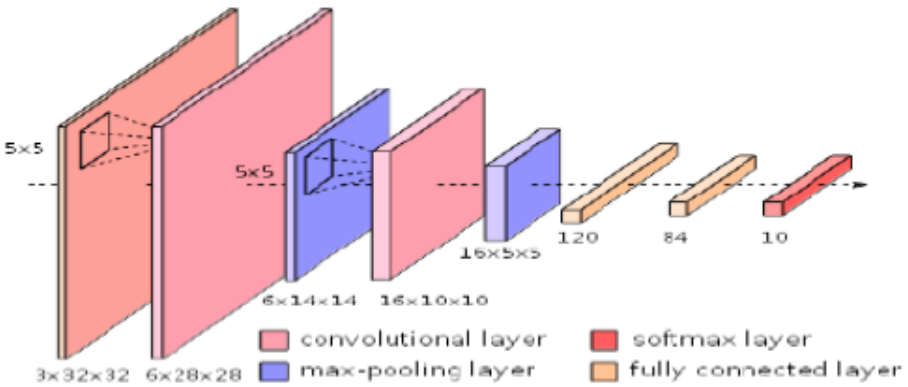
Abstract

Convolutional Neural Networks are very similar to ordinary Neural Networks architectures make the explicit assumption that the inputs are images, which allows to encode certain properties into the architecture. The LeNet-5 network was trained on the MNIST dataset in this problem.

Approach

For the first part, each of the questions about CNNs were answered. For the second part, the LeNet-5 CNN was trained on the MNIST dataset in Keras using Google Colab. Then the process was repeated 5 times but with changing a CNN parameter every time. The accuracy and loss curves for the training and test data were plotted and reported. The mean and variance of the train and test accuracies were recorded. The model with the best accuracy was chosen and it was used for the third part. In the third part, the model chosen was used to train on the negative of the MNIST dataset. The accuracy and loss curves for this system were plotted too. Finally, the original and negative MNIST datasets were merged and the CNN was trained on the new dataset, plotting the accuracy and loss curves once more.

Results and Discussion (merged for ease of readability)

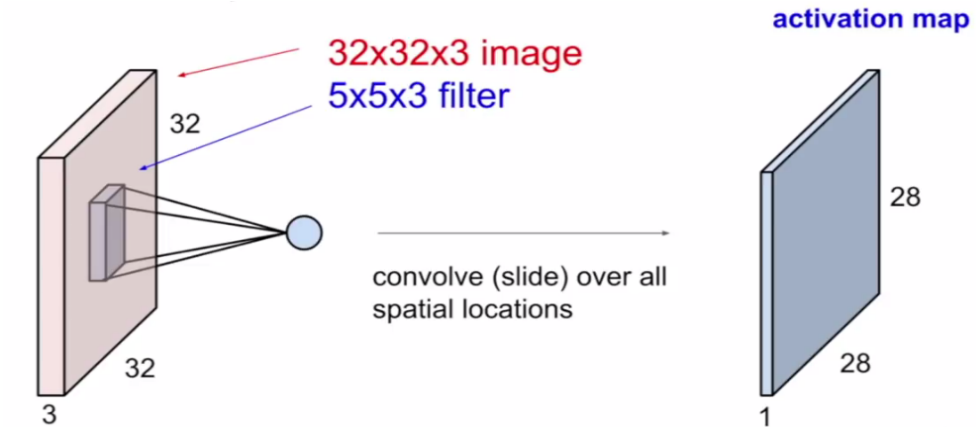


a)

1. Description of Convolutional Neural Network components: -

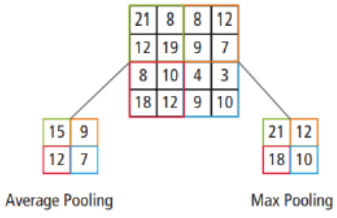
- Convolutional Layer

The convolutional layer is the first layer of the CNN and it works on the input to the network. It consists of a filter which is typically 5x5x3 for a color image which slide over the receptive field. This is the convolution process which can have a stride of one where it does not skip any value, or greater than one where it skips values in the horizontal and vertical direction. The result of the convolution is an activation map which has the responses of the operation at every position. Each convolutional layer will have multiple filters convolving over the input and therefore the response will be multidimensional.



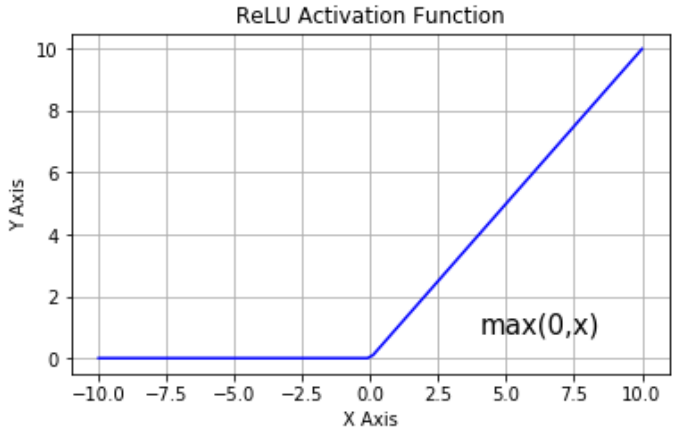
- Max Pooling Layer

It is common practice to include a pooling layer after a convolutional layer in the network architecture. The pooling layer reduces the size of the representation in order to speed up computation. The pooling layer works on each of the dimensions of the representation coming in to it. It does do by identifying small 2x2 windows and depending on the type of pooling layer, it reduces the window to a single value, therefore reducing the size by half. In max pooling, the maximum value out of the 4 in the 2x2 window is chosen. In average pooling, it is the average of the 4 values that is chosen.



- Activation Function

  The activation function is a node that is placed in between or at the end of a neural network. Its main function is to decide whether the neuron will fire or not. It is basically a transformation on the input signal that is non linear in nature. The non linear transformed output is sent to the subsequent layer is an input. There are different types of activation functions – sigmoid, tanh, maxout, ELU, leaky ReLU – but the most commonly used activation function nowadays is the ReLU (Rectified Linear Unit), depicted below. Its greatest advantage is that it does not activate all neurons at the same time.



- Fully Connected Layer

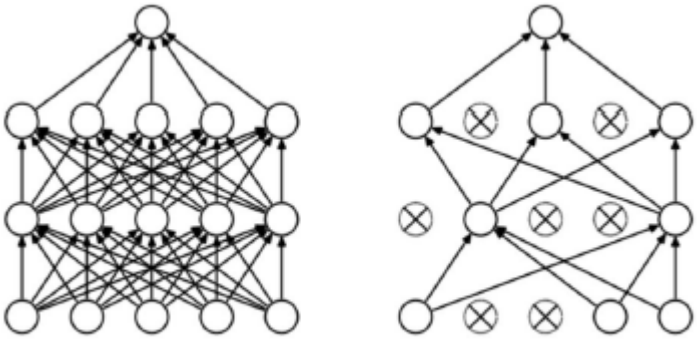
  All the layers up to this point have helped us in identifying and detecting the features of the input. A fully connected layer is usually attached towards the end of the neural network. This layer takes the input volume and gives result to an N dimensional vector where N is the number of classes that we have to classify an input into. In this example of digits, N would be 10. How the fully connected layer works is that it identifies the output of the preceding layer which are coming from the activation function in most cases and it decides which features correlate the most to a particular class.

- Softmax Function

  The output from the fully connected layer is an N dimensional vector. The softmax function takes this N dimensional vector as its input and transforms it into a real number vector with all values between 0 and 1, and their sum equal to 1. Rather than choosing one greatest component, it breaks the entire with maximal component getting the biggest bit of the distribution, yet other littler components getting some of it too. This distribution of probabilities helps in classification of the input.

2. Overfitting happens during data analysis where the predicted model is too close to the training data and thus probably won't most likely fit the future data into trained framework. What this means is that the model has almost memorized the training data and its curve fits the samples too closely. It can then not generalize and adapt to any new information. An overfitted model contains more parameters than that can be supported by the data. Underfitting happens when the system can't catch the structure of information data satisfactorily. Overfitting will in general have more variance in the forecasts however it will have less bias towards wrong results. This is what is known as the Bias-Variance Tradeoff.

   A technique that has been used in CNN training to avoid the overfitting of the model is regularization. Regularization refers to a type of strategy whose main function is to make model less complex. The most famous variation of regularization is dropout. Dropout is one of the most well-known systems since it is practically utilized by all deep learning models. It removes arbitrary number of initiations by making them zero. This reduces the number of associations with the following layers which may cause loss of data. However, it diminishes the odds of overfitting. Dropout keeps the model from getting to be reliant on any one neuron. In this way, a great practice is to begin with low dropout in first layer and after that slowly expanding it and consequently less data will be lost. Two other well-known forms of regularization are the L1 and L2 forms of regularization.



(a) Standard Neural Net (b) After applying dropout.

3. Traditional methods in computer vision problems can be split into two main parts. The first is feature extraction and the second is classification. In traditional methods, feature extraction is a very difficult and computationally expensive process. It is also different for every case and cannot be generalized. Therefore, a lot of domain knowledge is also necessary in carrying out feature extraction. The second part, classification, depends heavily on feature extraction and therefore it also become extremely difficult. In CNNs however, the model learns the useful features and extracts them on its own from the input. Even in the example of image classification, the CNN architecture can be applied to the input data and the various layers of the network learn the useful features on its own without

requiring any domain knowledge. Therefore, the system is generalized. Other advantages of CNNs are its invariance to shifts and distortions in an image, its memory usage being lesser, its high accuracy and its ease in training.

4.

- Loss Function

  The loss function is also called the cost function. The main purpose of the loss function is to find the difference between the actual output and the desired output. For the model to perform better, the loss function needs to be lower. The type of loss function being used in this problem is Cross Entropy and it is defined by the following expression: -

  $$Cross\ Entropy = -\sum_{i} y_i' \log(y_i)$$

  Here, y' is the label of the ground truth and y is the prediction of the classifier.

  Another common loss function is the Mean Square Error loss function: -

  $$MSE := \frac{1}{n}\sum_{t=1}^{n} e_t^2$$

  Here, e is the error between the ground truth label and the predicted label.

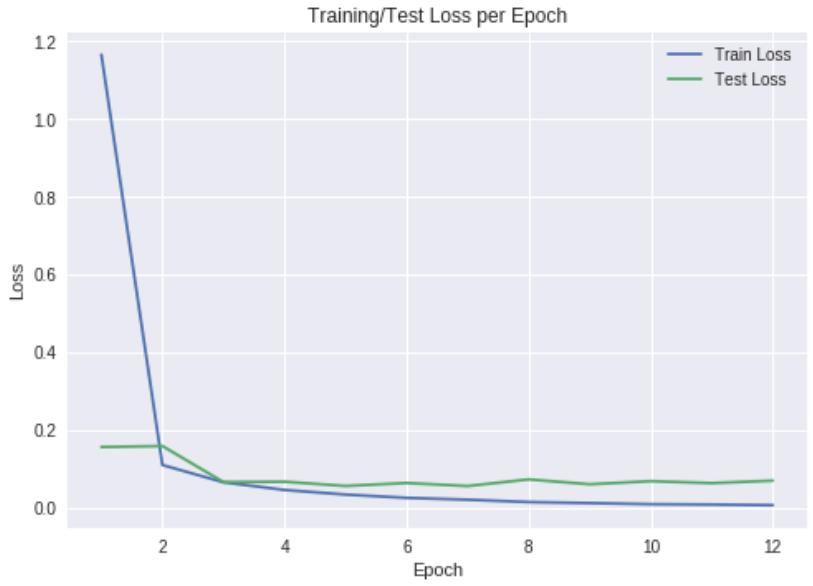
- Back Propagation

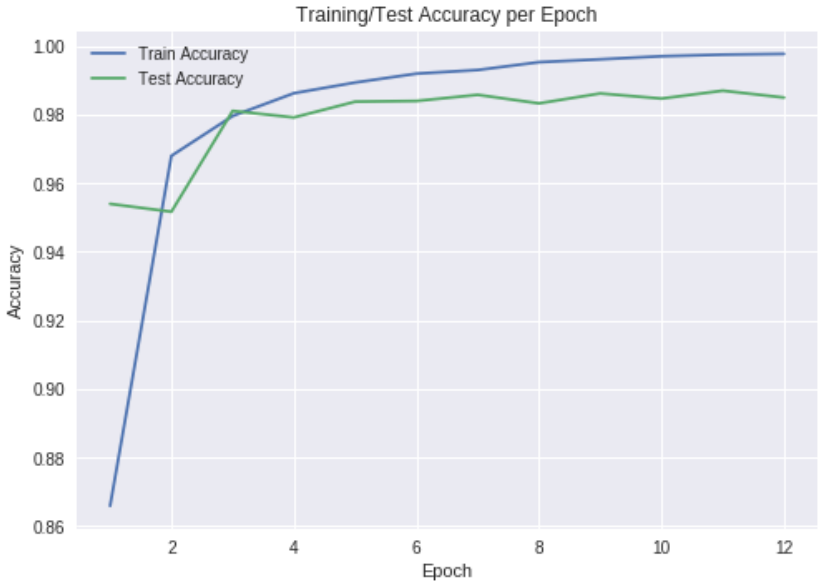
  Back propagation is a concept used in CNNs which are heavily reliant on the loss function. When an input image is given to the network, each layer gives an output which is passed on to the next until the final error in estimation of classification is carried out. The error is then propagated backwards through the very same layers and same neurons until every neuron in each layer is associated with the error value and therefore depicting its weight in the actual output. The goal is that through the propagation of these error backwards, the system can learn what features to map to the corresponding output without any external information from the user or the input.

b)

- Base Model – Standard LeNet-5 Architecture with 12 epochs, 3x3 Kernel, 128 batch size, MaxPooling, ReLU Activation and Adadelta Optimizer

LeNet 5 Training Error: 0.2267%
LeNet 5 Training Loss: 0.6396%
LeNet 5 Training Accuracy: 99.7733%
LeNet 5 Test Error: 1.5000%
LeNet 5 Test Loss: 6.9518%
LeNet 5 Test Accuracy: 98.5000%



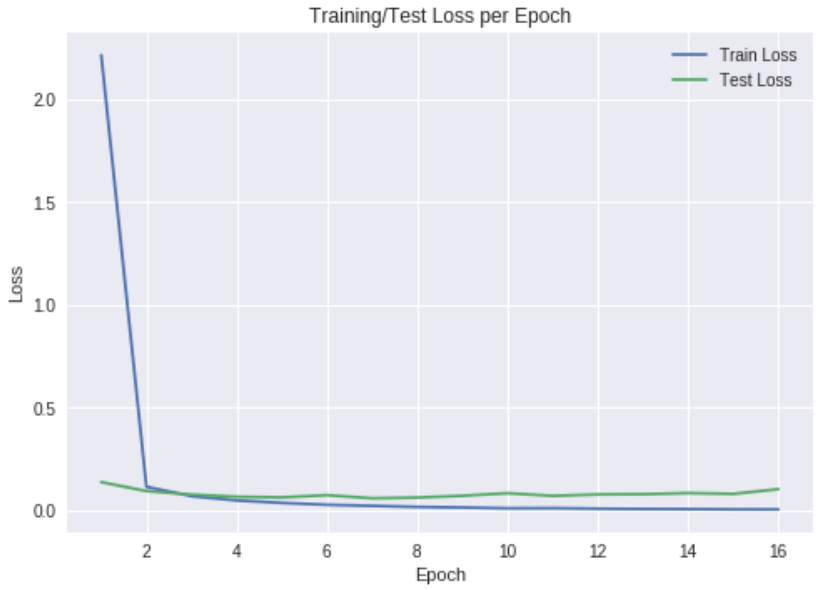


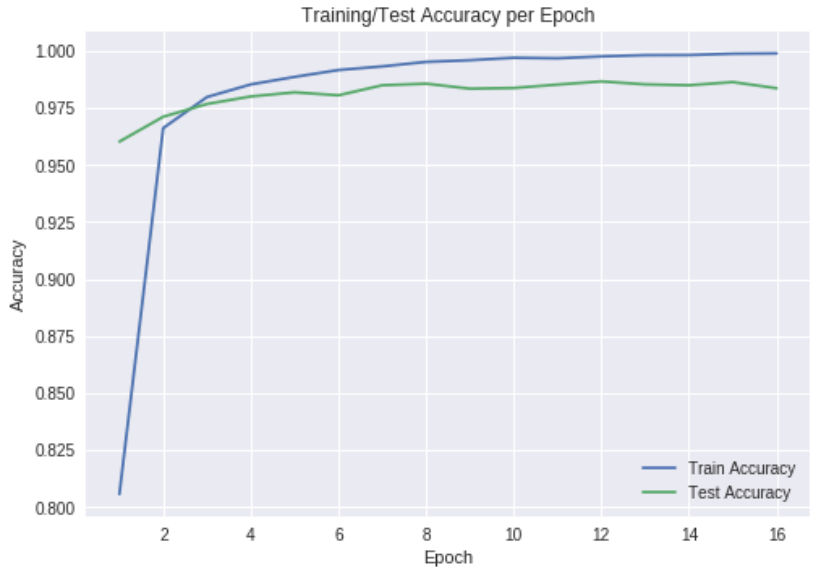
The base model training and testing results in a good test accuracy of over 98%.

- Setting 1: Changing epochs from 12 to 16

  LeNet 5 Training Error: 0.1133%
  LeNet 5 Training Loss: 0.3595%
  LeNet 5 Training Accuracy: 99.8867%
  LeNet 5 Test Error: 1.6400%
  LeNet 5 Test Loss: 10.2494%
  LeNet 5 Test Accuracy: 98.3600%

  One would expect the test accuracy to increase with an increase in epochs but that is not seen in this particular case. A healthy accuracy of over 98% is still achieved.

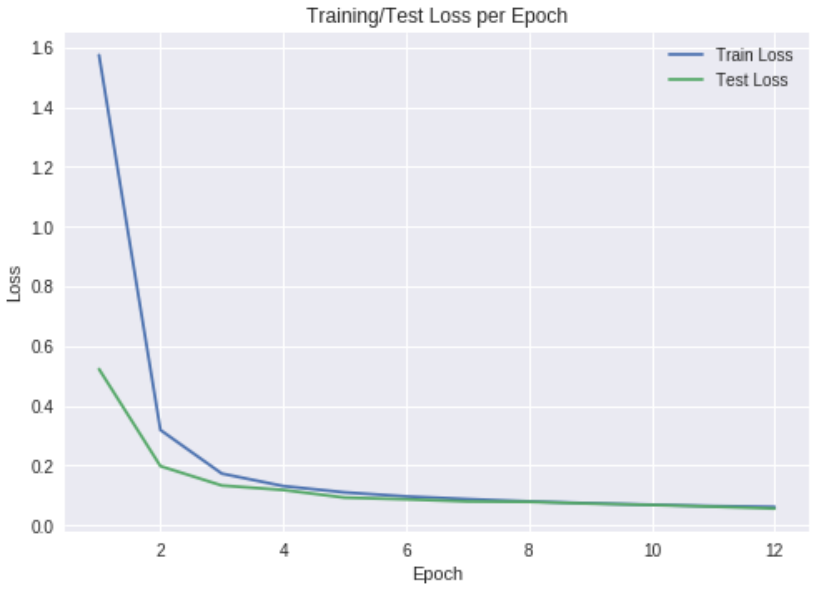


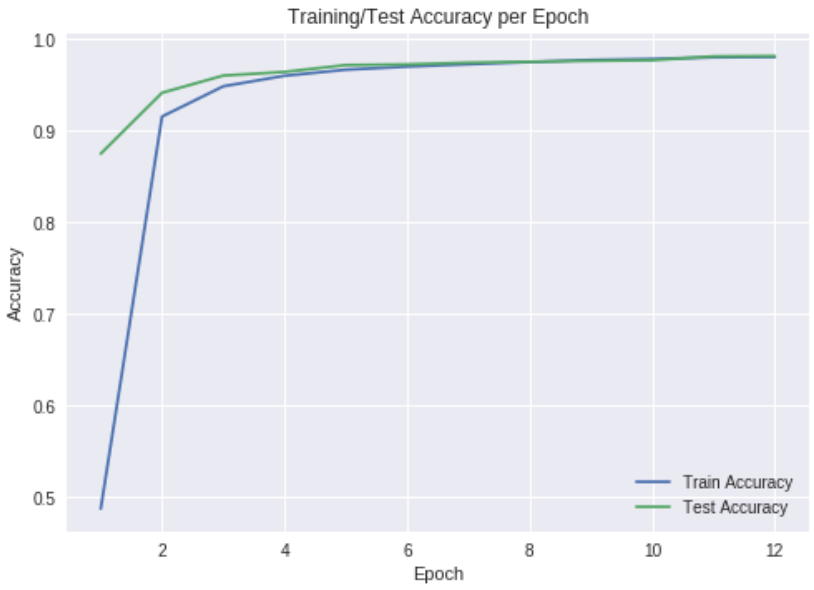


- Setting 2: Changing the activation function from ReLU to sigmoid

  LeNet 5 Training Error: 1.9167%
  LeNet 5 Training Loss: 6.1822%
  LeNet 5 Training Accuracy: 98.0833%
  LeNet 5 Test Error: 1.8100%
  LeNet 5 Test Loss: 5.6043%
  LeNet 5 Test Accuracy: 98.1900%

  The performance using a sigmoid activation function has definitely dropped even though it is above 98%.

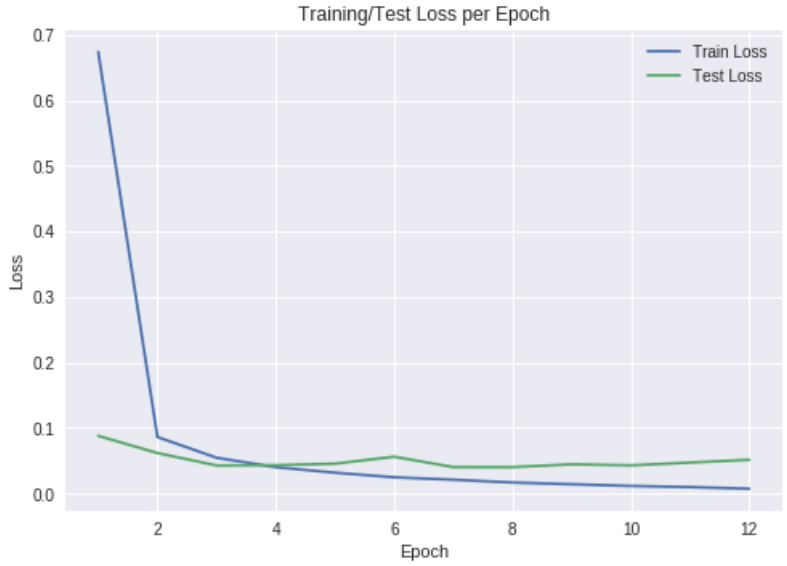


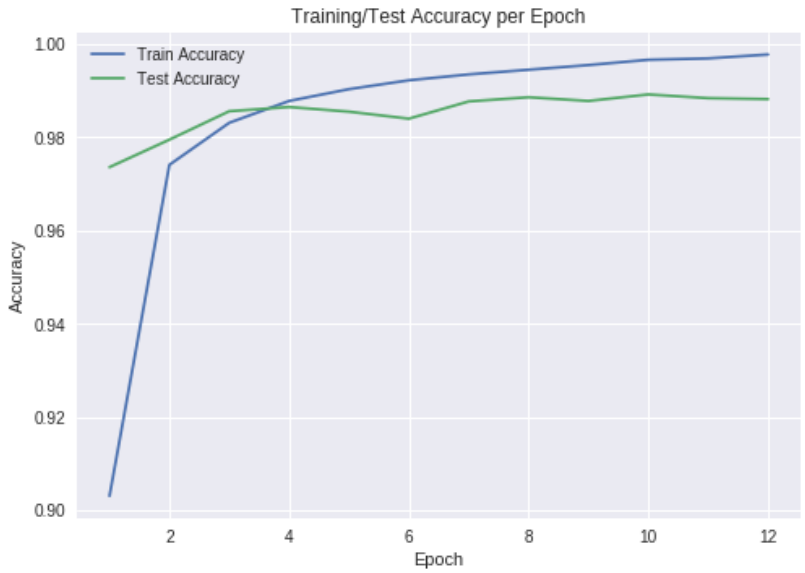


- Setting 3: Changing from MaxPooling to AveragePooling

  LeNet 5 Training Error: 0.2233%
  LeNet 5 Training Loss: 0.7283%
  LeNet 5 Training Accuracy: 99.7767%
  LeNet 5 Test Error: 1.1800%
  LeNet 5 Test Loss: 5.1451%
  LeNet 5 Test Accuracy: 98.8200%

  In this case, when using Average Pooling, the performance on the test set improves. The accuracy increases to 98.82%.

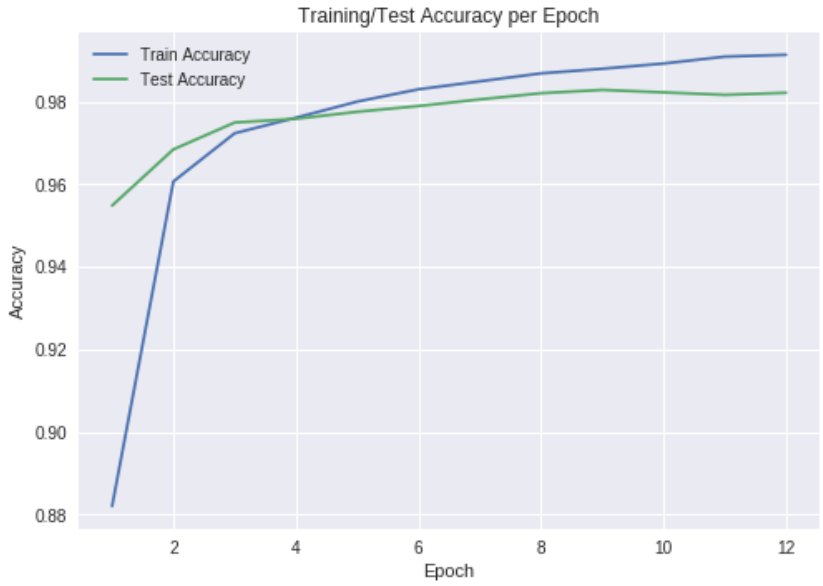




- Setting 4: Changing optimizer from Adadelta to Stochastic Gradient Descent (SGD with Learning Rate 0.01)

  LeNet 5 Training Error: 0.8700%
  LeNet 5 Training Loss: 2.8135%
  LeNet 5 Training Accuracy: 99.1300%
  LeNet 5 Test Error: 1.7900%
  LeNet 5 Test Loss: 6.8347%
  LeNet 5 Test Accuracy: 98.2100%

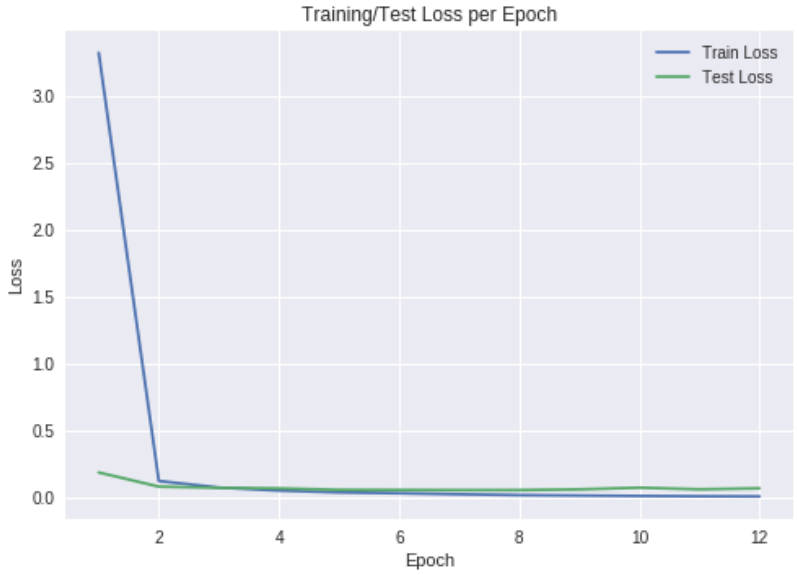
  In this setting again, the performance drops as we see the test accuracy go down to 98.21%.

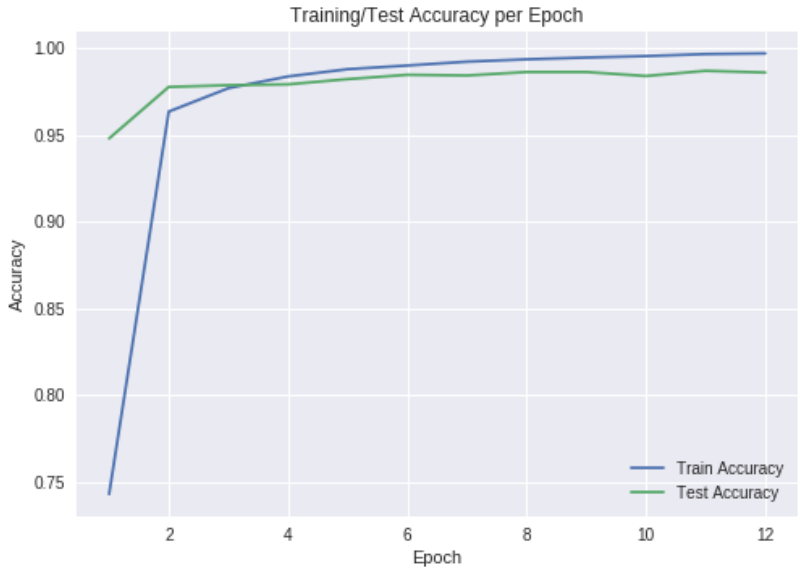


- Setting 5: Changing Kernel Size from 3x3 to 5x5

  LeNet 5 Training Error: 0.3167%
  LeNet 5 Training Loss: 1.0265%
  LeNet 5 Training Accuracy: 99.6833%
  LeNet 5 Test Error: 1.4200%
  LeNet 5 Test Loss: 7.1062%
  LeNet 5 Test Accuracy: 98.5800%

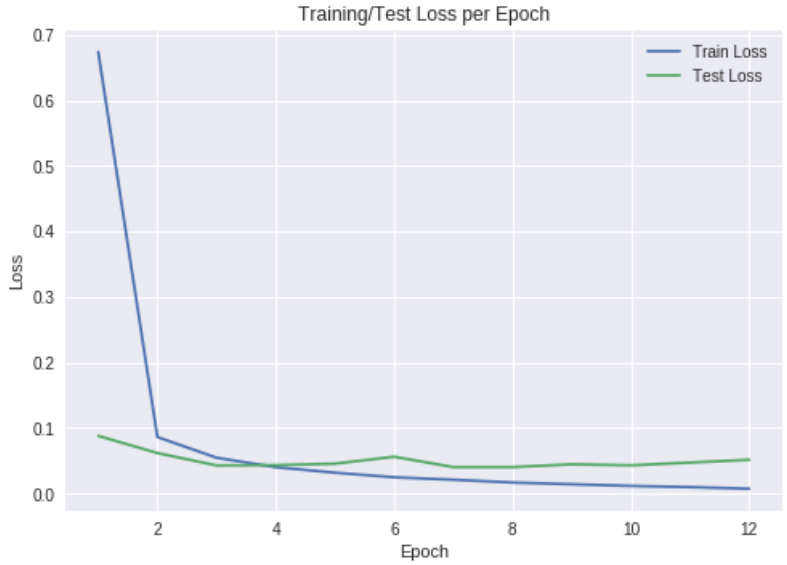
  Generally, a smaller kernel size would lead to higher accuracy but in this case the 5x5 kernel size seems to perform just a little bit better than the 3x3 kernel size. The test accuracy increases from 98.5% to 98.58%.

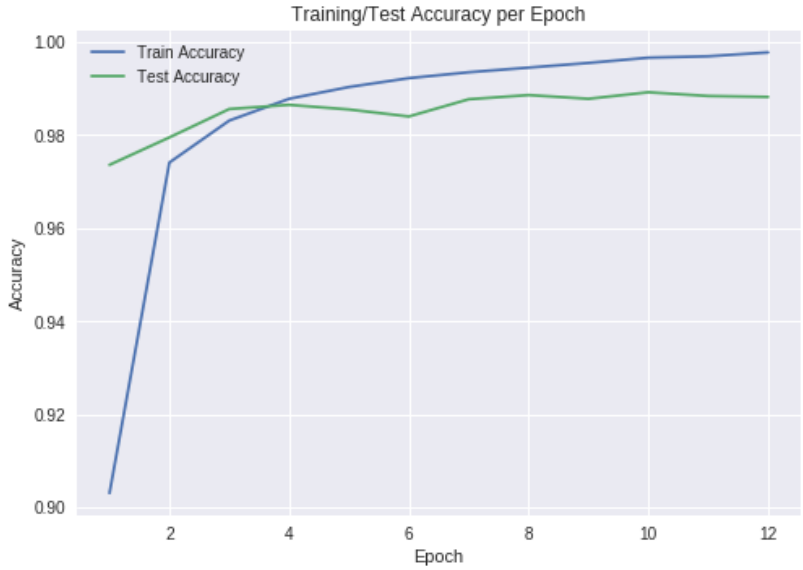




- After considering the base model and the models with the 5 settings changed, the best performing model is Setting 3 which uses AveragePooling. It's stats and plots are reported below again: -

  LeNet 5 Training Error: 0.2233%
  LeNet 5 Training Loss: 0.7283%
  LeNet 5 Training Accuracy: 99.7767%
  LeNet 5 Test Error: 1.1800%
  LeNet 5 Test Loss: 5.1451%
  LeNet 5 Test Accuracy: 98.8200%





- The mean and variance of the train and test accuracies were calculated as follows: -

  Train Accuracy Mean: 99.38891666666667%
  Train Accuracy Variance: 0.4007340213888933

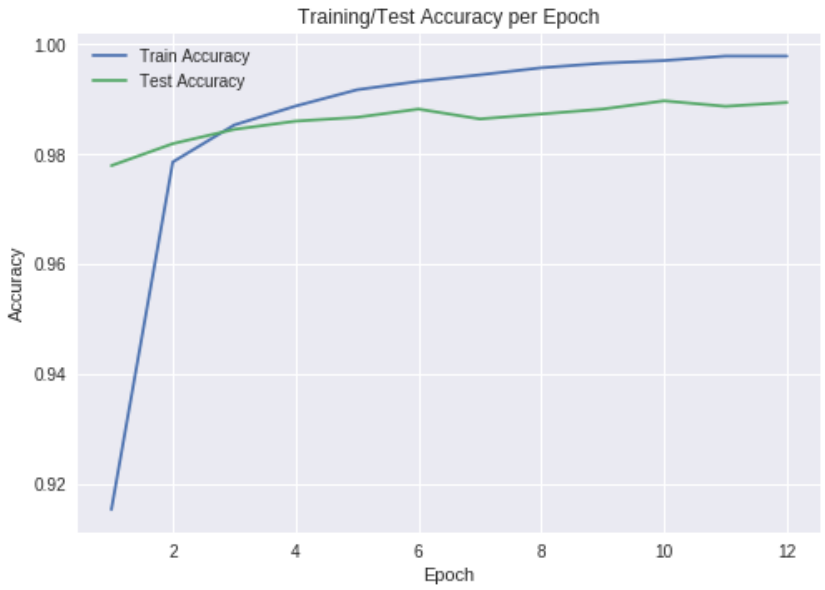
  Test Accuracy Mean: 98.44333333333333%
  Test Accuracy Variance: 0.04822222222222198

c)

1. The best performing model in part (b) was Setting 3 which used the Average Pooling. This model was applied to the negative MNIST dataset after subtracting the original MNIST dataset from 255. In this case, the original dataset has black background with white lettering and the negative dataset has white background with black lettering. The performance stats and curves are below: -

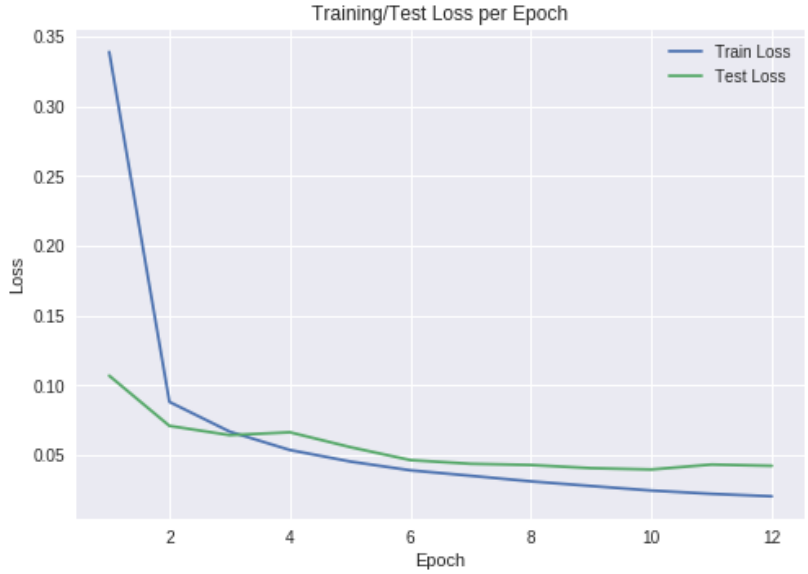
   LeNet 5 Training Error: 0.2150%
   LeNet 5 Training Loss: 0.6255%
   LeNet 5 Training Accuracy: 99.7850%
   LeNet 5 Test Error: 1.0600%
   LeNet 5 Test Loss: 4.0916%
   LeNet 5 Test Accuracy: 98.9400%

   It is seen that the model performs slightly better on the negative dataset than on the original dataset as the test accuracy increases from 98.82% to 98.94%.



2. A new network was constructed which included the original MNIST dataset as well as the negative MNIST dataset. The two training and testing datasets were stacked and then shuffled to produce 120,000 samples instead of 60,000. The X_Train and X_Test datasets were normalized as well. Again, average pooling was used. The model was trained and tested with the results shown below: -

   LeNet 5 Training Error: 0.6308%
   LeNet 5 Training Loss: 2.0431%
   LeNet 5 Training Accuracy: 99.3692%
   LeNet 5 Test Error: 1.2050%
   LeNet 5 Test Loss: 4.2283%
   LeNet 5 Test Accuracy: 98.7950%



It is seen that the test accuracy comes to 98.79% which is very good for the merging of the original and negative MNIST datasets.