Homework 4 Report
// Date : 16 April 2018
// Name : Tamoghna Chattopadhyay
// ID : 8541324935
// email : tchattop@usc.edu

Problem 1.(a)

I. Abstract and Motivation

Artificial Neural Networks is an important part of Artificial Intelligence and tries to replicate the work of neurons present in human brains. Convolutional Neural Networks is a type of feed forward Artificial Neural Network. In CNN, the connectivity pattern between the neurons is designed on the animal visual cortex.

The MNIST dataset has a set of 60000 examples as training set and 10000 examples as test set. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. The CNN architecture to be implemented was based on LeNet-5 which was introduced by LeCun et al. for handwritten and machine printed character recognition. It's as shown:

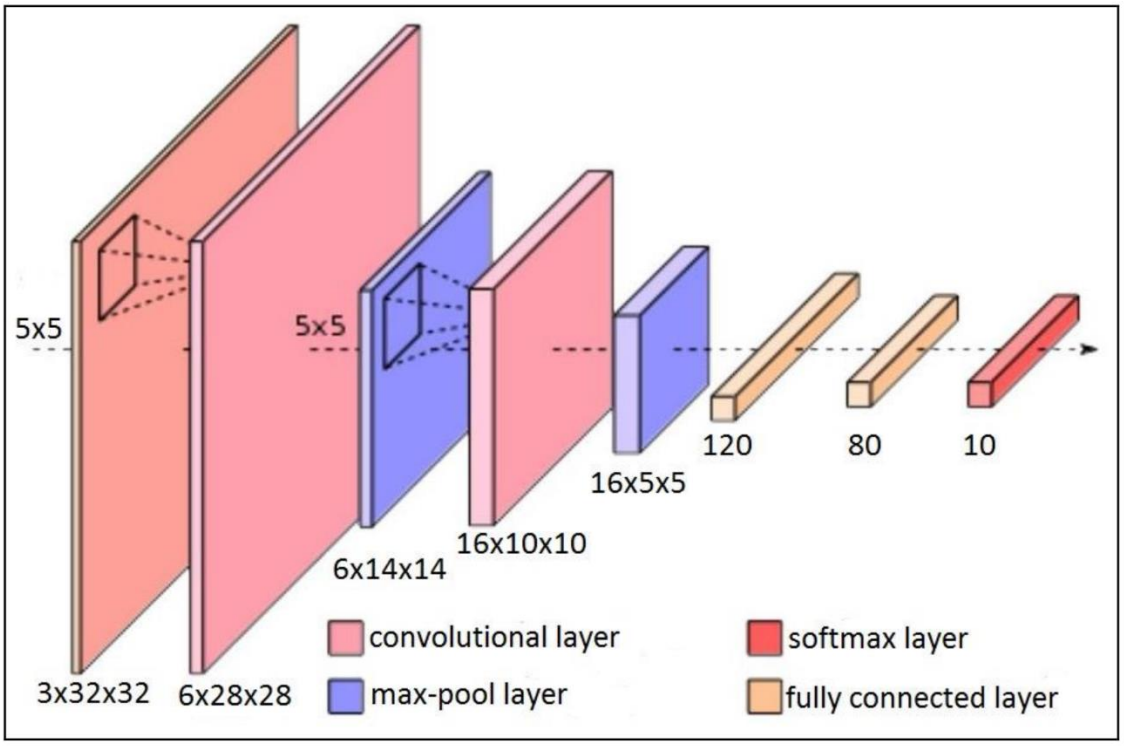


Figure-1.1: A CNN architecture derived from LeNet-5

This network has two *conv* layers, and three *fc* layers. Each *conv* layer is followed by a *max pooling* layer. Both *conv* layers accept an input receptive field of spatial size *5x5*. The filter numbers of the first and the second *conv* layers are 6 and 16 respectively. The stride parameter is 1 and no padding is used. The two *max pooling* layers take an input window size of *2x2*, reduce the window size to *1x1*

by choosing the maximum value of the four responses. The first two *fc* layers have 120 and 80 filters, respectively. The last fc layer, the output layer, has size of 10 to match the number of object classes in the MNIST dataset. The popular ReLU activation function is used for all *conv* and all *fc* layers except for the output layer, which uses softmax to compute the probabilities.

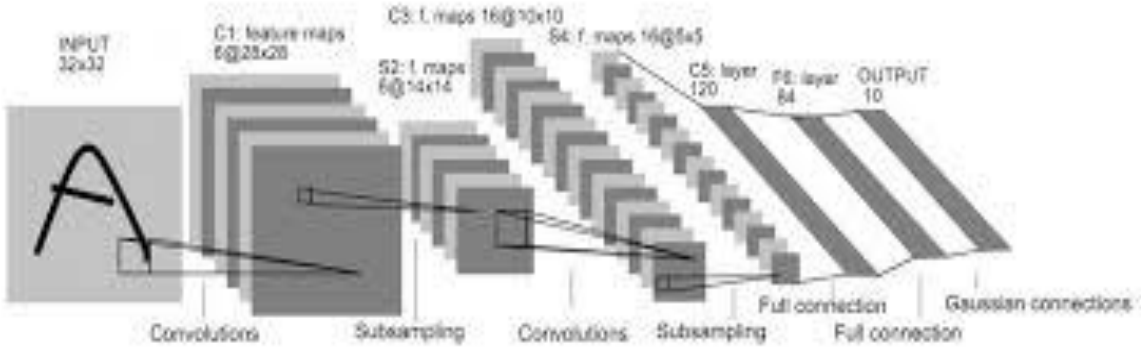


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

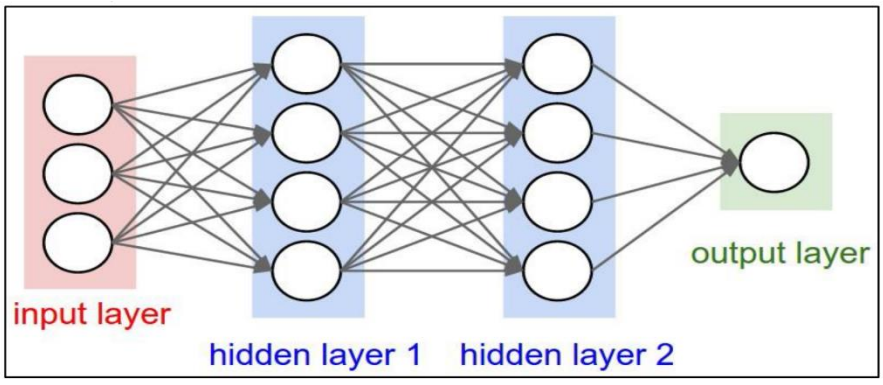
The LeNet 5 architecture as an exemplary CNN

II. Discussion

a. Following are the descriptions of each layer of the LeNet-5 CNN:

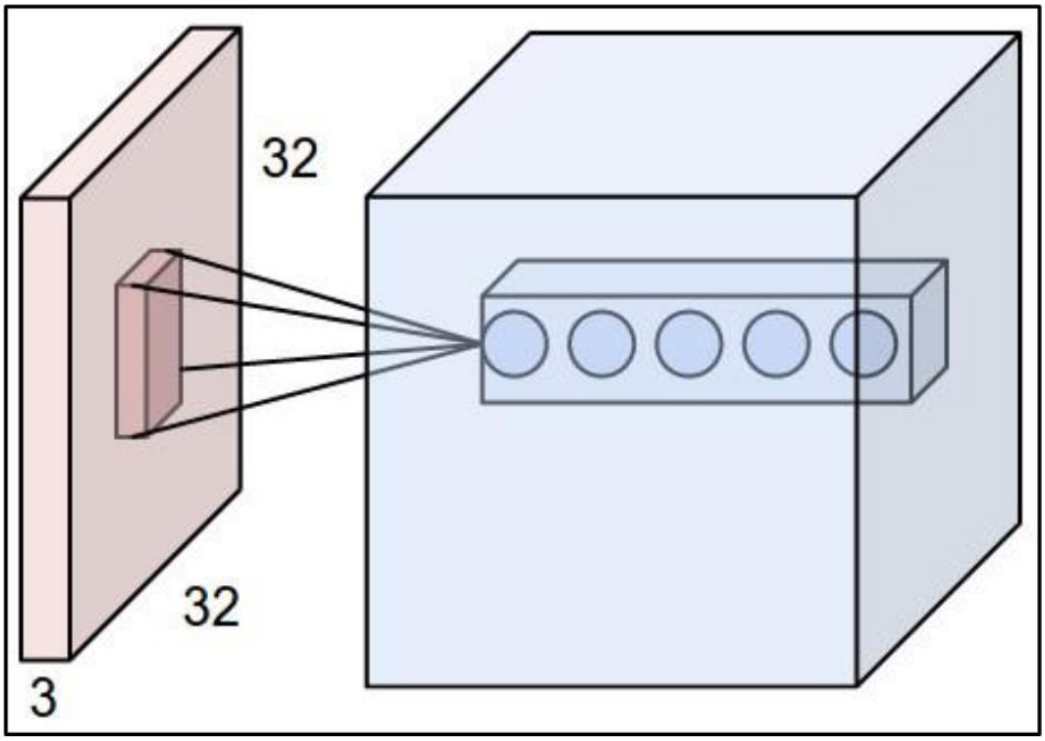
1. Fully Connected Layer :

This layer has full connections to all the activations in the previous layer. The activations for this layer can be calculated using matrix multiplication and then providing necessary bias. This layer basically takes as input an input volume and outputs an N dimensional vector where N is the number of classes that the classifier must choose from. It looks at the outputs from the previous layer and determines which features most correlate to a class and have weights so that when the product between the weights and previous layer is calculated, we get the correct probabilities for the different classes. The previous layer, i.e., the input to this layer should represent the the activation maps of high level features. An example is shown in the image:



2. Convolutional Layer :

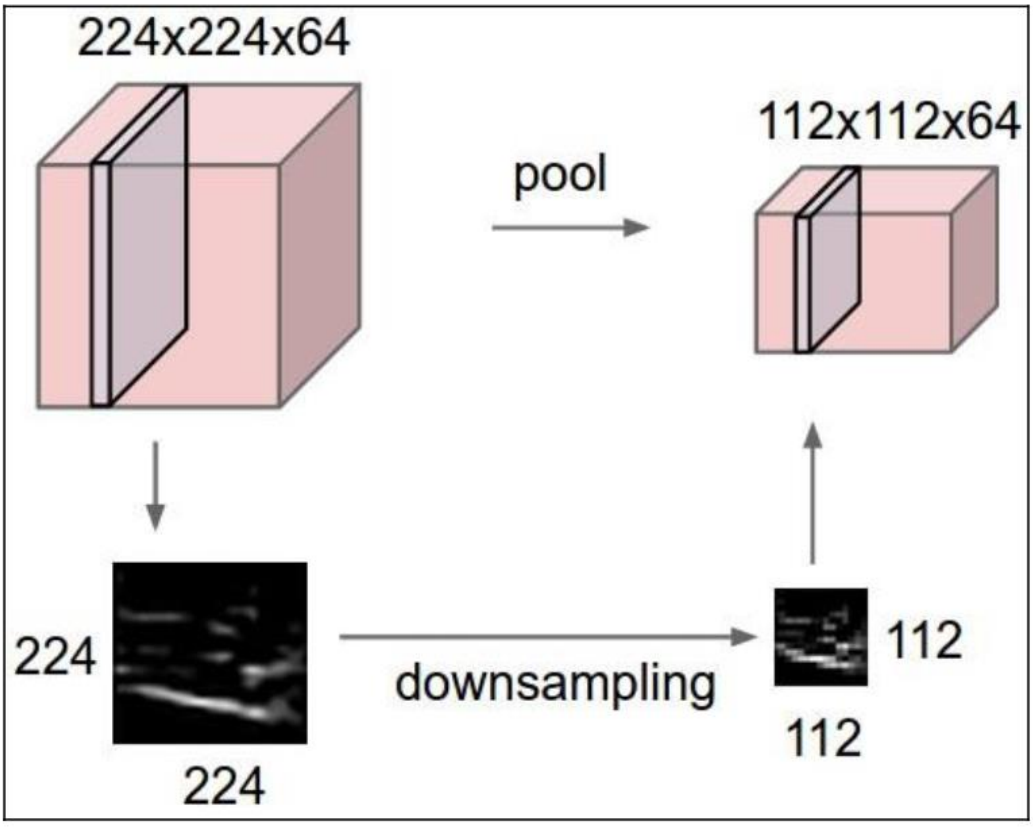
The convolutional layer is the basic building block of any CNN network. It does most of the computationally heavy tasks. The layer has a set of learnable filters. The filters are normally small spatially along the height and width, but they cover the entire depth of the image. The input image is forward passed by convolving the filters across the entire image height and width. This step includes the calculation of the dot product of the filter and image elements at every position along the width and height of image. Thus, a 2 dimensional activation map is generated based on the filter response in every spatial position. Based on the number of convolutional layers specified, each layer has their own 2D activation maps which are stacked along the depth to generate an output image volume. Some of the important parameters to consider while designing this layer are number of filters, filter size, stride and zero padding. Here, stride refers to the movement of the convolutional window for the filter shift for doing the next convolution. Zero padding is done by expanding the boundary so that the pixels at the edges are also taken into consideration during convolution.



3. Max-Pooling Layer :

This layer is present between successive Convolution Layer. The major function of this layer is to progressively reduce the spatial size of the convolution layer output image to reduce the number of parameters and computation in the in the network so that overfitting gets controlled. This layer has no learning parameters. It works independently on each depth slice of the input and resizes it spatially using the MAX operation. . The commonly used pooling layer example is

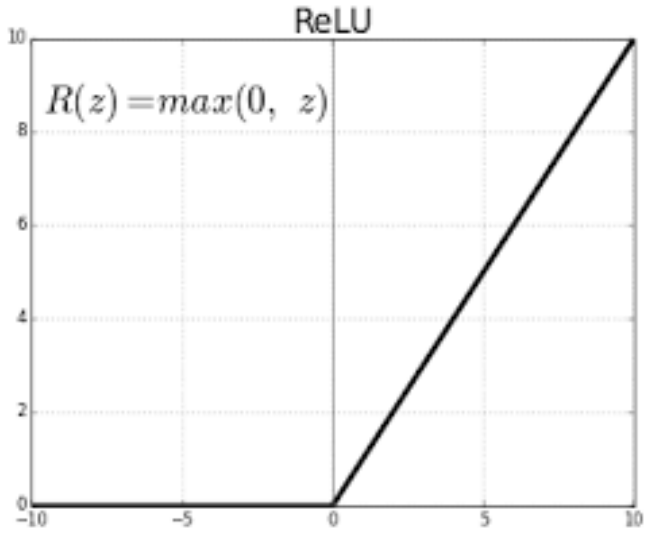
of filter size 2x2. This is applied with a stride of 2 down samples, which removes approximately 75% of the activations in the network. Every MAX operation takes a max over 4 numbers in this case. The depth dimension remains unchanged. Some of the important parameters to consider while designing this layer are window size and stride.



The image shows the input volume size being pooled with filter size 2 and stride 2 to give output volume size.

4. Activation Function :

Activation Functions are present in the nodes of the different layers. They define the output for a node, given the different sets of inputs possible. It is a non linear function which decides whether a neuron is activated. Thus, it helps to introduce non-linearity into the network. They contain zero learnable parameters. Activation functions are nonlinear in nature, as summing up layers with linear activation function would just give another linear function. Some of the popular activation functions are Sigmoid function, Tanh function, ReLU function and leaky ReLU function. We use ReLU function in the layers here.



The ReLU function computes the function f(x) = max(0,x), i.e. the activation is threshold at 0. It doesn't saturate in the positive region and converges faster than the Sigmoid and Tanh functions. The output values are not zero centered. But ReLU functions are fragile during training and can die out if the learning rate is improperly set. If the function dies out, then it gives the same output irrespective of the input and hence can't differentiate between inputs. Once a ReLU dies, it's not possible to recover back.

5. Softmax Function :

Mathematically, the softmax function is a generalization of the logistic function. It reduces a K-dimensional vector of arbitrary real values into a K-dimensional vector of real values in the range of [0,1] so that the sum becomes 1. The function serves as the final layer of the neural network classifier. These networks are commonly trained under a cross-entropy regime, giving a nonlinear variant of multinomial logistic regression. Thus, this function is basically a generalization of the logistic regression.

b. One of the major issues with artificial neural networks is that the models are quite complicated. By building a very complex model, it's very easy to perfectly fit the given dataset. But when the same model is implemented on new data, the performance is very poor. The model doesn't generalize well. This phenomenon is called overfitting. This issue is significant as neural networks have large number of layers with millions of connections. As the number of connection increases, so does the chance of overfitting the data.

   One method of combating overfitting is regularization. Regularization changes the objective function which we minimize. It adds additional terms that penalize large weights. In other words, we change the objective function so that it becomes Error + Regularization term which is a function of a variable multiplied by a parameter. The value of the parameter chosen sets the amount of overfitting to be allowed. The setting of the parameter takes trial and error method. L2 regularization is the most popular type of regularization used.

Dropout is another method to remove overfitting. It is often used in place of other techniques. While training, in the process of dropout, a neuron stays active only with some probability p, otherwise it is set to zero. This makes the network to be accurate even in the absence of some information. It prevents the network from becoming too dependent on any one (or any small combination) of neurons.

c. CNNs work better than other traditional methods in many computer vision problems. Using the image classification problem as an example, all images have a hierarchical structure. The images have pixels at the lowest levels, which combine to make up lines and curves which further make up shapes. These shapes constitute the final image. CNN I better algorithm in such cases as it uses this hierarchical structure in it's algorithm to classify problems and give solutions.

   The integral part of a CNN are convolutional layers and max pooling layers which directly work with the pixels of the image. The outputs obtained from this layer is the lines and curves contained in the image. This output is again fed to the next layers to get shapes. This repetitive process is what constitutes image. This ability of CNN to interact with deeper layers to establish solutions make it better for computer vision problems. Also CNN architecture is easily adaptable to new problems. This ensures high classification accuracy for the most part.

d. Loss Function is a function that maps an event onto one or more values which represent some sort of cost associated with the event. It is used in optimization procedures to reduce the cost to obtain desired results. It is used in the back-propagation mechanism to reduce the cost involved in training the images. Backward Propagation is a method of training neural networks used alongside optimization methods such as gradient descent.

   There are two steps in this algorithm: propagation and weight update. When image input volume is given to the network, it is propagated forward into the network, through each layer until it reaches the final layer. Then this output is compared to the desired output using a loss function. This comparison evaluates the error value for each neuron in the output layer. These error layers are propagated backwards from the output layer so that each neuron has some error value which represents it's contribution to the output layer. The back propagation uses this error values to calculate the gradient of the loss function with respect to weights in the network. In the next step, the gradient obtained is fed to the optimization process which updates the weights to minimize the loss function. This method requires known outputs for each input value to find the loss function gradient, therefore it's a supervised learning method.

Problem 1.(b)

I. Abstract and Motivation

In this problem we have to train the LeNet-5 CNN architecture on the 60000 training images present in the MNIST dataset. LeNet-5 CNN architecture is a good method developed for performing text classification and recognition.

II. Approach and Procedures

The code for the problem was written in python. The model for the CNN was developed as follows:

1. The input layer which provides the input image pixel values to the CNN network. The MNIST dataset consists of images of dimensions 28x28 in both the training and testing datasets. Thus the size of the input layer was set to 28x28 with depth equal to 1 as the images are grayscale.

2. The local regions of input image data is taken and connected to each neuron in the first convolutional layer. The LeNet-5 architecture says that the receptive field is set to be 5x5 with no zero padding and stride 1. The input image is convolved over 6 filter banks. The number of weights that each neuron in this layer will have is (5x5x1(weights) + 1(bias))*6 = 156 that correspond to a 5x5x1 region in the input volume. The convolution layer has ReLu activation.

3. The next layer is a max-pooling layer that takes the output of the previous layer as input and subsamples the output. A 2x2 window is considered in this case. This is done to improve the input layer and reduce the number of weights required in next layer. The dimension of the output thus becomes 14x14. There are no weights in this layer.

4. The next layer is another convolution layer similar to the previous one. The receptive field is set to be 5x5 with no zero padding and stride 1. The input image is convolved over 16 filter banks. The dimension of the image output here is 10x10. . The number of weights that each neuron in this layer will have is (5x5x6(weights) + 1(bias))*16 = 2416. The convolution layer has ReLu activation.

5. The next layer is a max-pooling layer that takes the output of the previous layer as input and subsamples the output. A 2x2 window is considered in this case. This is done to improve the input layer and reduce the number of weights required in next layer. The dimension of the output thus becomes 10x10x16. There are no weights in this layer.

6. The next layer is the first fully connected hidden layer consisting of 120 neurons connected to the previous layer. This layer has the maximum receptive field. ReLu activation is used here again.

7. The next layer is the first fully connected hidden layer consisting of 84 neurons connected to the previous layer. The neurons are connected to the previous 120 neurons from the previous layer. ReLu activation is used here again.

8. The final output layer consists of 10 neurons representing the output labels connected to the 84 neurons in the previous layer. This is cascaded with a log softmax layer that provides the log probabilities of the output labels. It is a method to normalize the outputs to easily determine the output label for each input.

In addition to the above architecture, some preprocessing steps were also done. The input image was normalized by dividing with 255 to get the pixel values in range from 0 to 1. Another important criteria which decides the speed and accuracy of the entire training and testing process is batch size. The batch size contributes towards convergence and performance of the network. By using batches, the variance of the update is reduced. Larger batch size contribute towards unstable update vectors which converges slowly. I set the batch size as 128.

Another important parameter is the learning rate which determines the time for convergence of the network. Smaller learning rates converge very slowly and larger learning rates may not converge to global minima. I set the learning rate as 0.1.

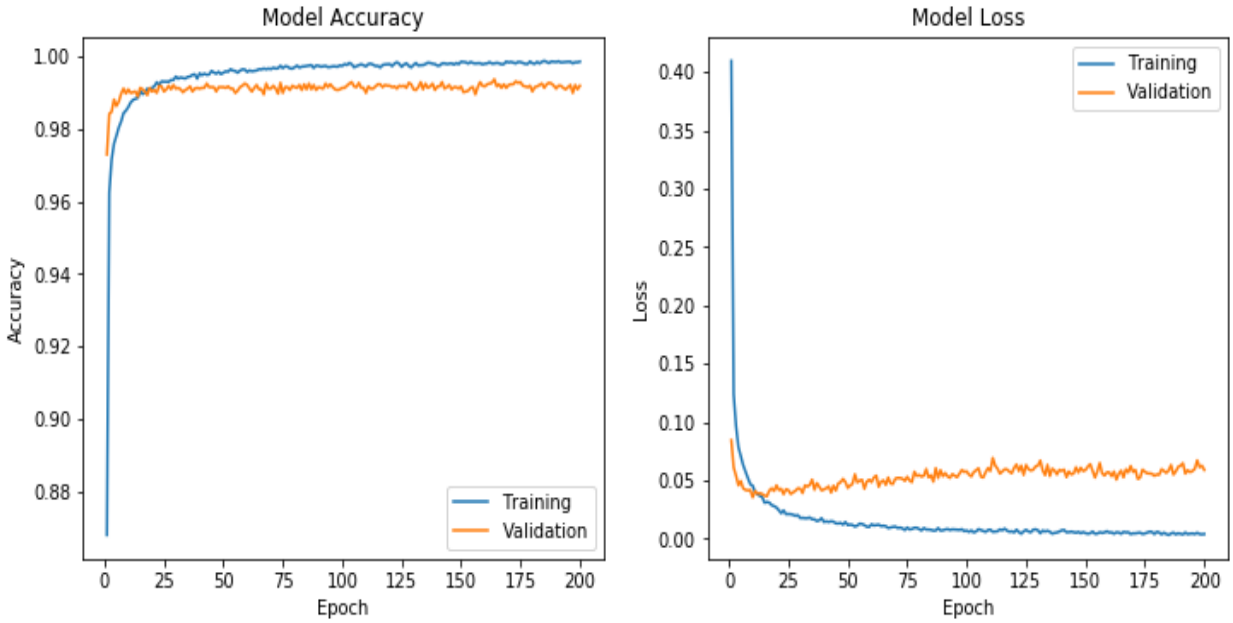
I used the SGD (Stochastic Gradient Descent) as the optimizer and also to prevent overfitting, the training set was randomly shuffled after every epoch and trained in batches. The experiment was conducted for 200 epochs.

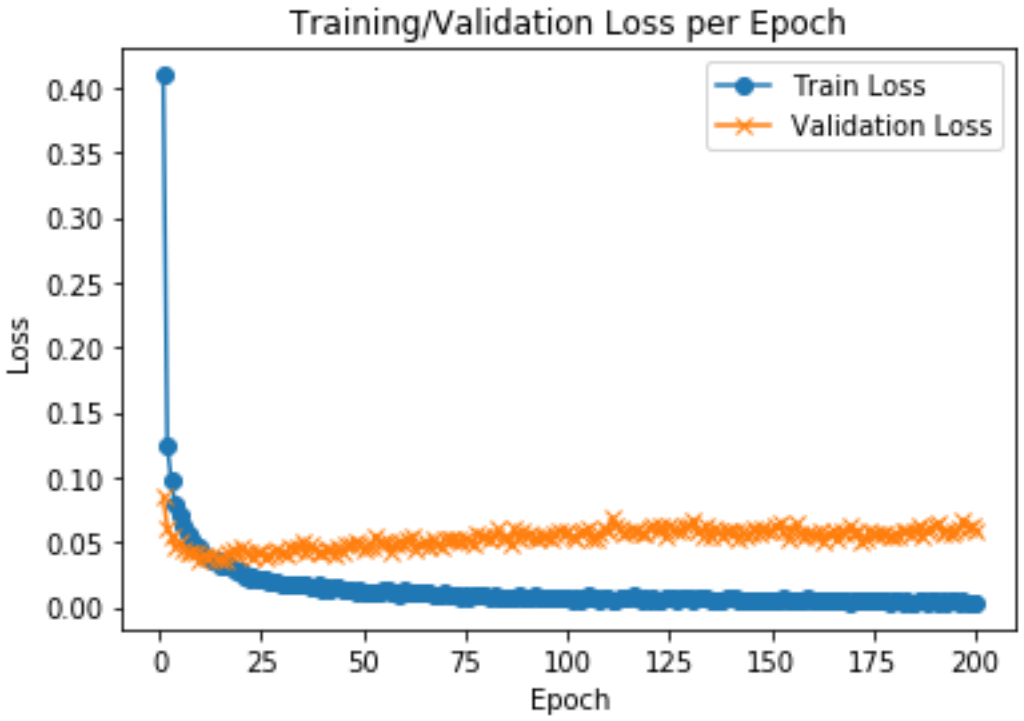
Another important factor to consider is dropout which indicates what portion of the neurons are to be dropped in layer during a batch. This is done to reduce overfitting. I applied a dropout rate of 0.5.

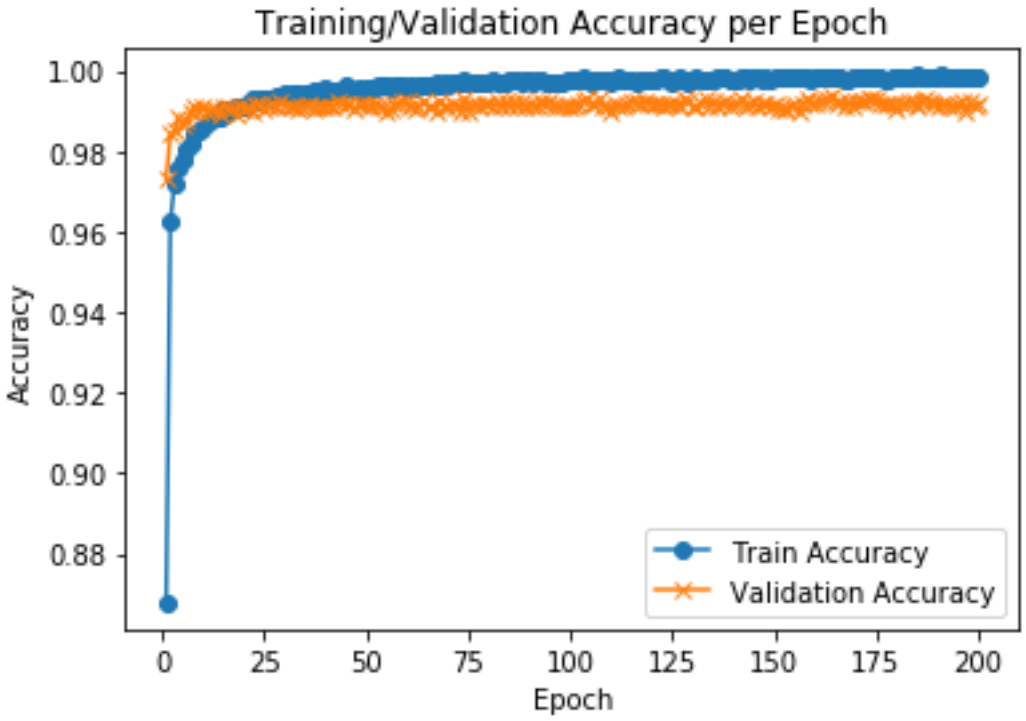
III. Discussion

For the LeNet-5 architecture,

LeNet 5 Test Error: 0.8200%
LeNet 5 Test Loss: 0.0500%
LeNet 5 Test Accuracy: 99.1800%

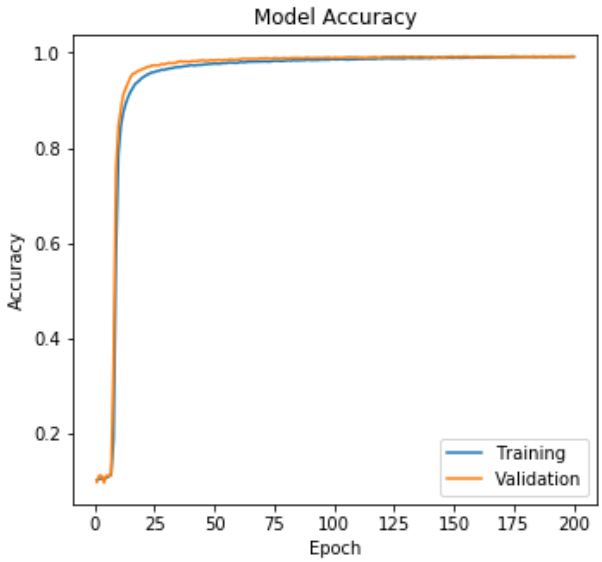


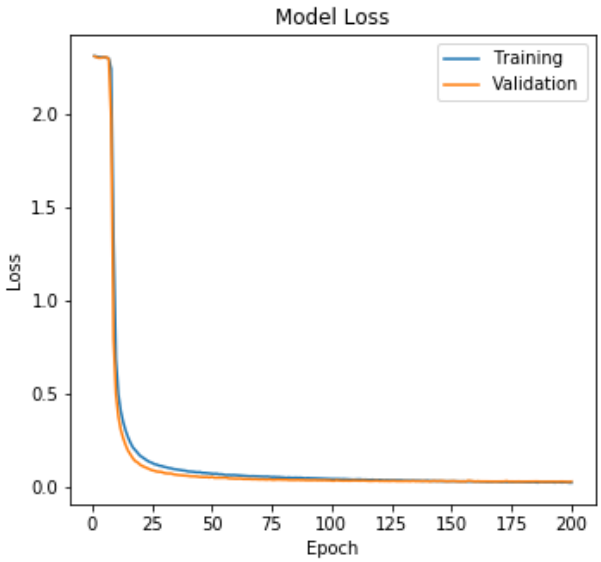

Training/Validation Loss per Epoch
Train Loss
Validation Loss
Loss
Epoch
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
25
50
75
100
125
150
175
200

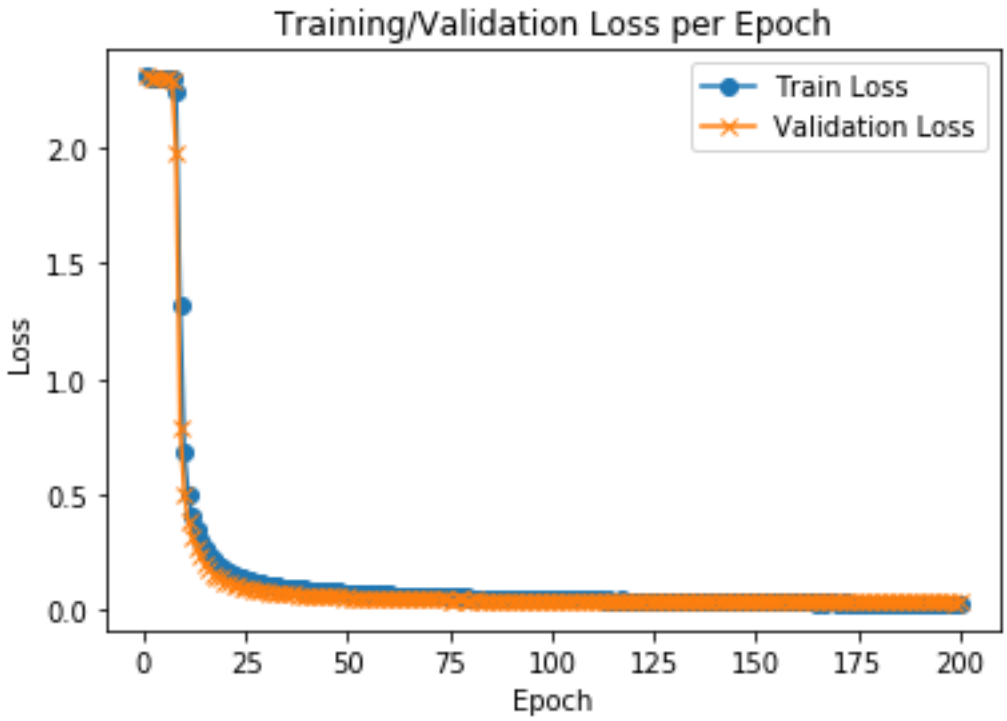

Training/Validation Accuracy per Epoch
Train Accuracy
Validation Accuracy
Accuracy
Epoch
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0
25
50
75
100
125
150
175
200

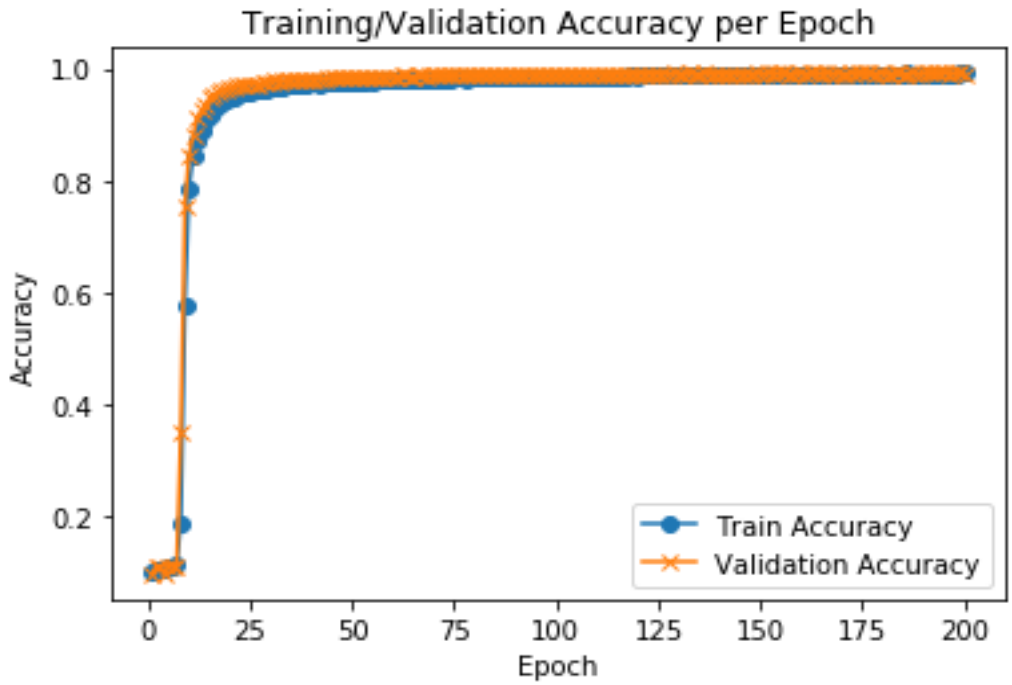
If I change the activation function to sigmoid,

LeNet 5 Test Error: 0.9200%
LeNet 5 Test Loss: 0.0270%
LeNet 5 Test Accuracy: 99.0800%



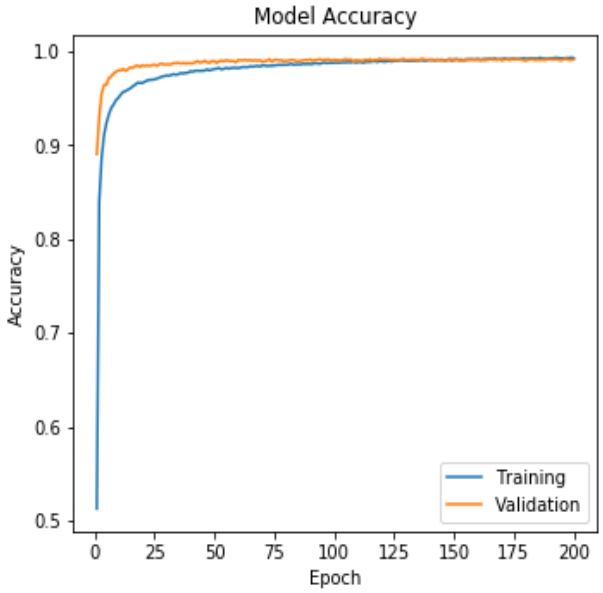


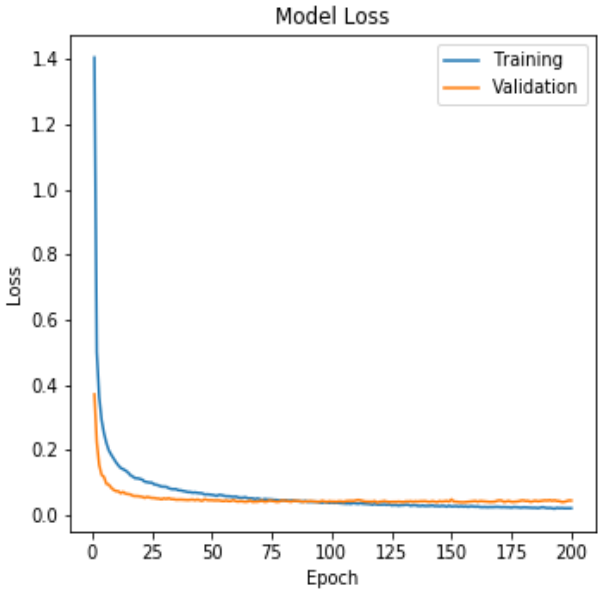


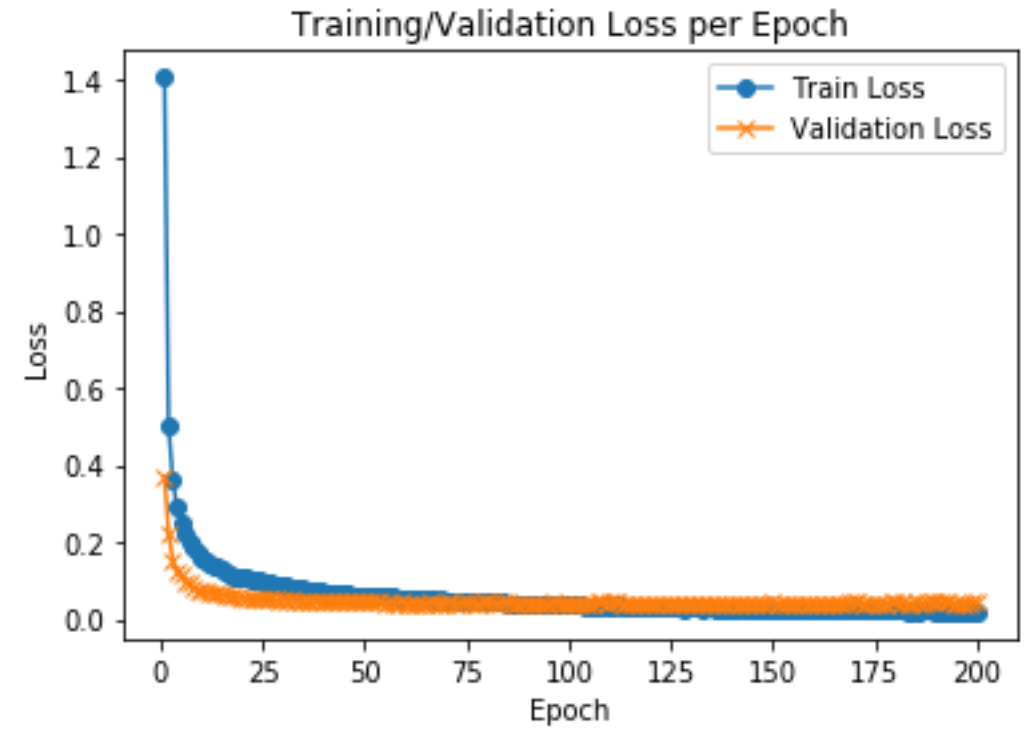


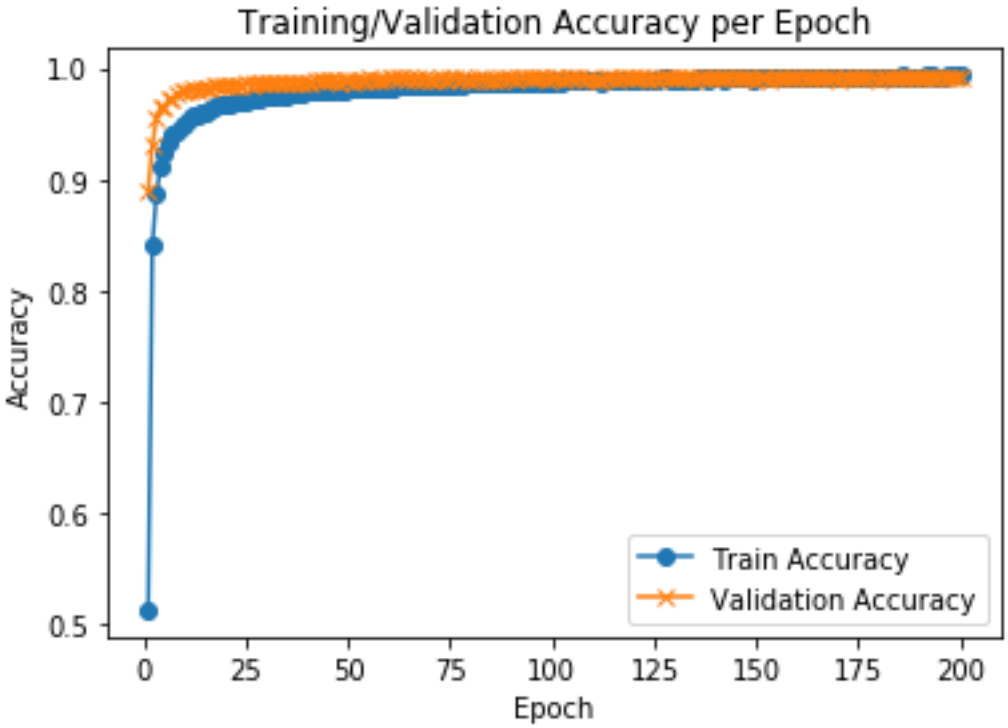
Next I kept the values of Dropout as 0.6 and learning rate as 0.01. The results were:

Test Error: 0.8100%
Test Loss: 0.0305%
Test Accuracy: 99.1900%







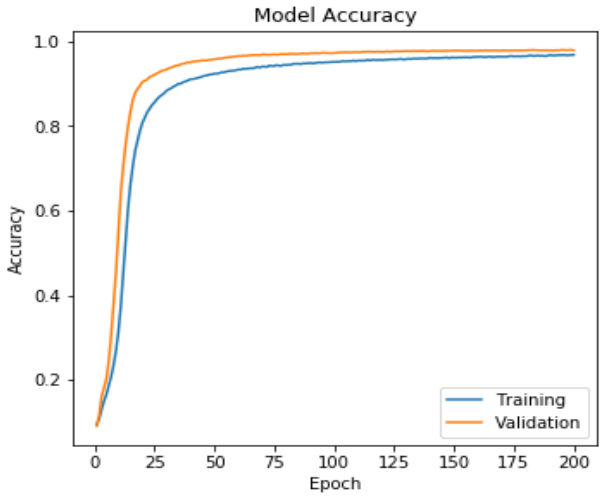


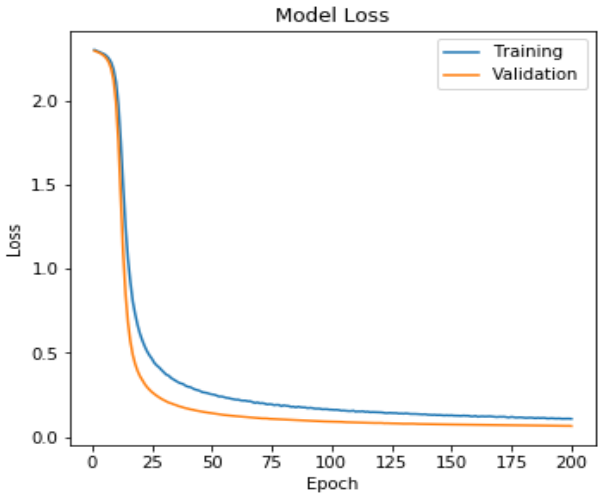
This gave lesser test error.

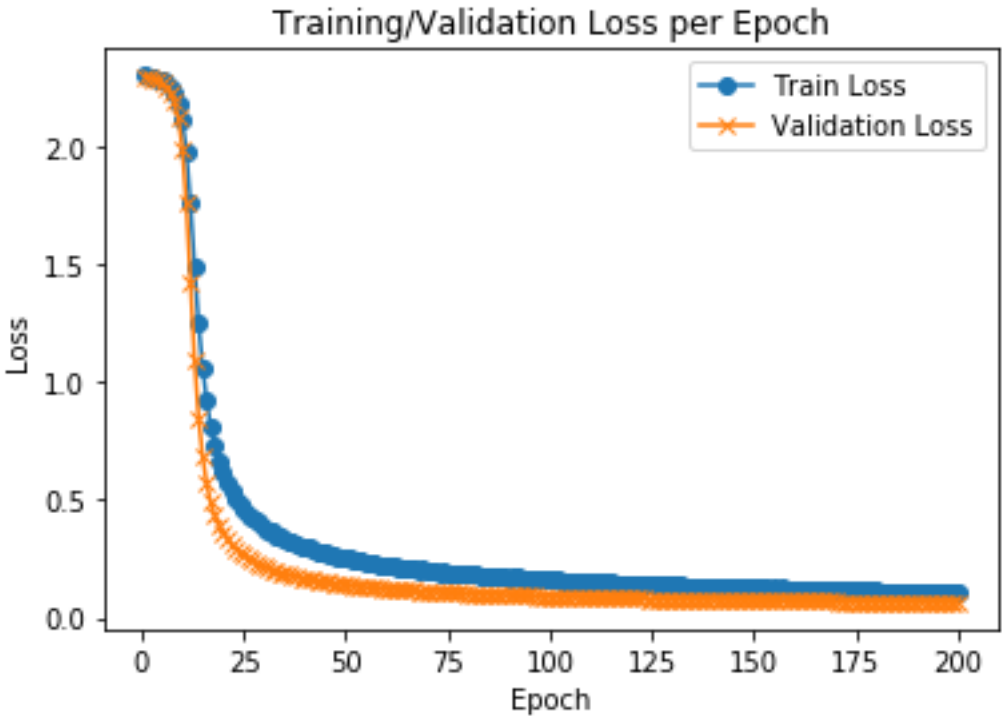
For better result, I tried to change the parameters and got as follows:
Batch Size = 256 Dropout = 0.5 Learning Rate = 0.001

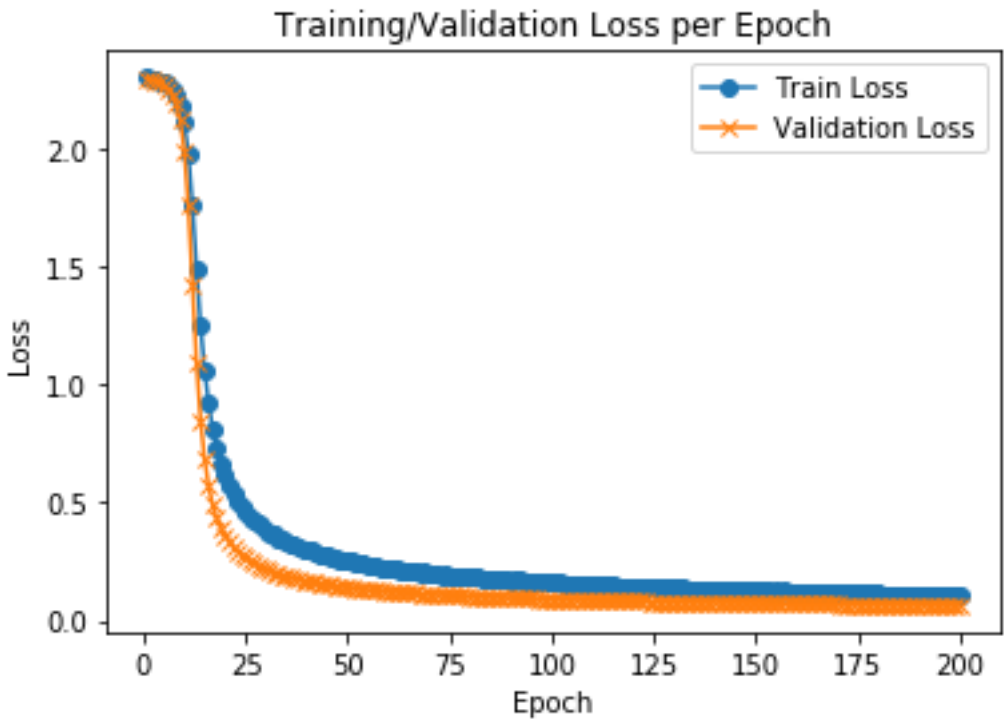
Test Error: 1.7700%
Test Loss: 0.0549%
Test Accuracy: 98.2300%

In this case, the test accuracy decreases from previous results.



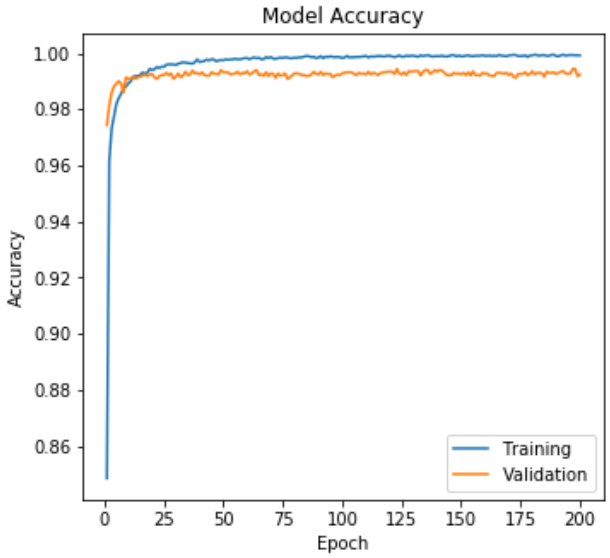


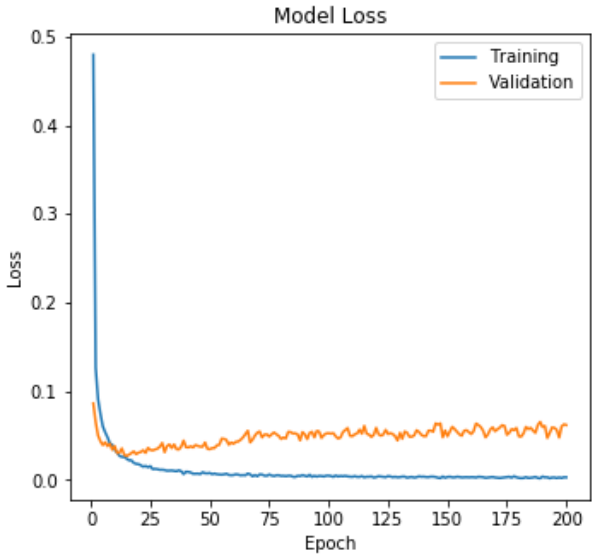


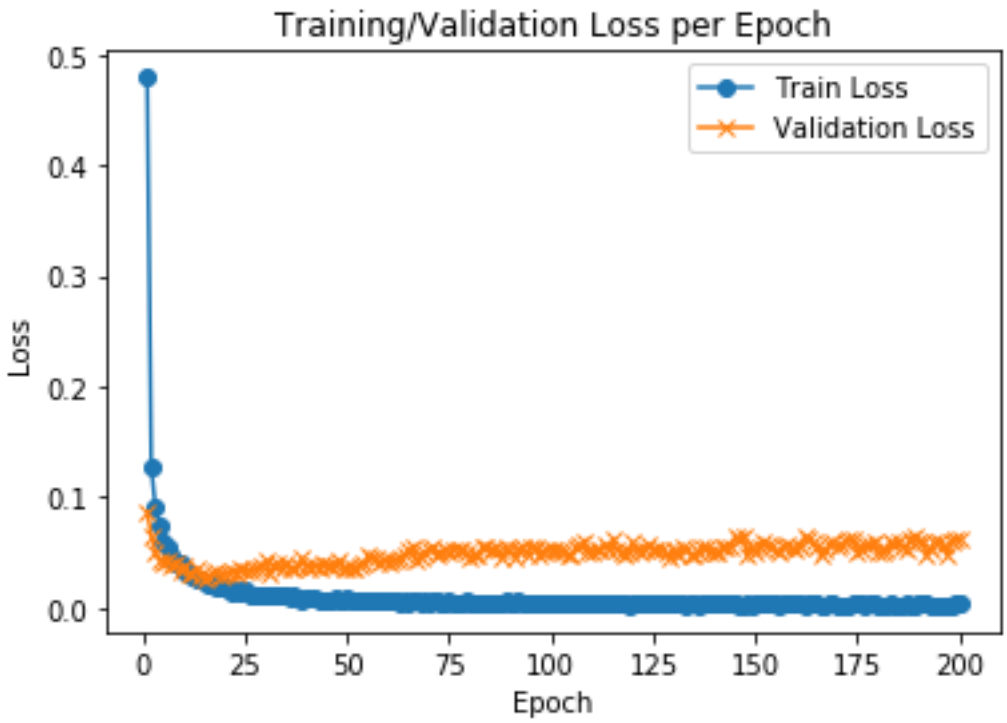


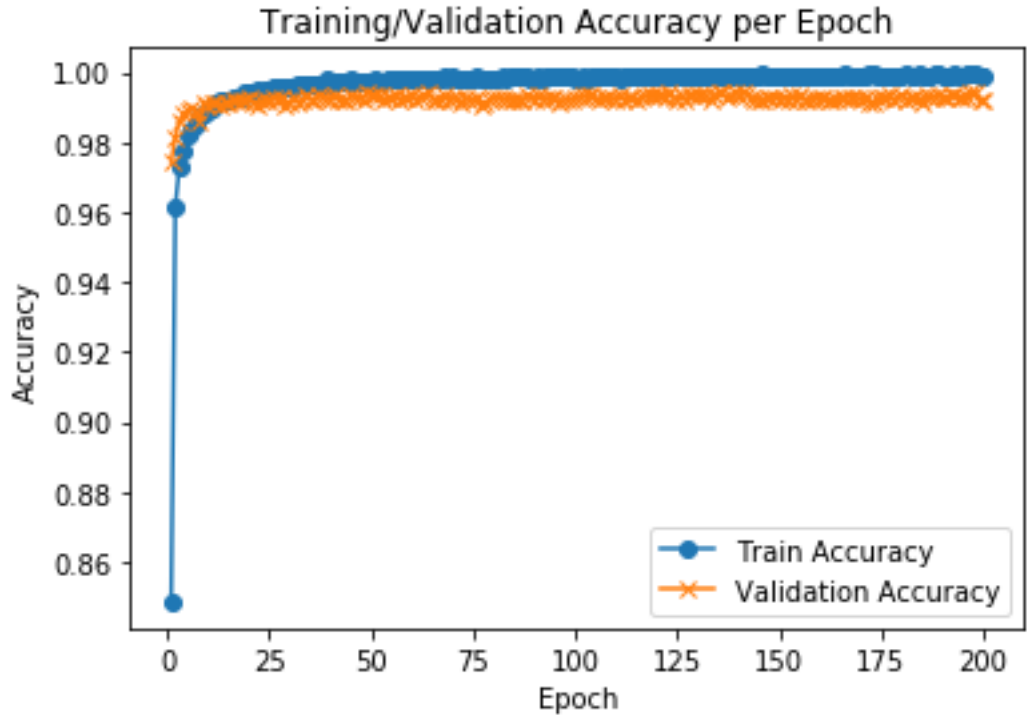
Then, I kept the above parameters same but changed the optimizer to an Adam optimizer. The results were better than before:

Test Error: 0.6800%
Test Loss: 0.0415%
Test Accuracy: 99.3200%









As we can see, the test accuracy increased in this case.

Thus, Adam optimizer is better than sgd. Also learning rate of 0.001 is better for better test accuracy.

Problem 1.(c)

I. Abstract and Motivation

In this problem we have to improve over the LeNet-5 CNN architecture on the 60000 training images present in the MNIST dataset.

II. Approach and Procedures

The code for the problem was written in python. The model for the CNN was developed as follows:

1. The input layer which provides the input image pixel values to the CNN network. The MNIST dataset consists of images of dimensions 28x28 in both the training and testing datasets. Thus the size of the input layer was set to 28x28 with depth equal to 1 as the images are grayscale.

2. The local regions of input image data is taken and connected to each neuron in the first convolutional layer. In the architecture I used, the receptive field is set to be 5x5 with no zero padding and stride 1. The input image is convolved over 30 filter banks. The number of weights that each neuron in this layer will have is (5x5x1(weights) + 1(bias))*30 = 780 that correspond to a 5x5x1 region in the input volume. The convolution layer has ReLu activation.

3. The next layer is a max-pooling layer that takes the output of the previous layer as input and subsamples the output. A 2x2 window is considered in this case. This is done to improve the input layer and reduce the number of weights required in next layer. The dimension of the output thus becomes 14x14. There are no weights in this layer.

4. The next layer is another convolution layer similar to the previous one. The receptive field is set to be 3x3 with no zero padding and stride 1. The input image is convolved over 15 filter banks. The dimension of the image output here is 10x10. . The number of weights that each neuron in this layer will have is (5x5x30(weights) + 1(bias))*15 = 4065. The convolution layer has ReLu activation.

5. The next layer is a max-pooling layer that takes the output of the previous layer as input and subsamples the output. A 2x2 window is considered in this case. This is done to improve the input layer and reduce the number of weights required in next layer. The dimension of the output thus becomes 10x10x16. There are no weights in this layer.

6. The next layer is the first fully connected hidden layer consisting of 128 neurons connected to the previous layer. This layer has the maximum receptive field. ReLu activation is used here again.

7. The next layer is the first fully connected hidden layer consisting of 50 neurons connected to the previous layer. The neurons are connected to the previous 128 neurons from the previous layer. ReLu activation is used here again.

8. The final output layer consists of 10 neurons representing the output labels connected to the 50 neurons in the previous layer. This is cascaded with a log softmax layer that provides the log probabilities of the output labels. It is a method to normalize the outputs to easily determine the output label for each input.

In addition to the above architecture, some preprocessing steps were also done. The input image was normalized by dividing with 255 to get the pixel values in range from 0 to 1. Another important criteria which decides the speed and accuracy of the entire training and testing process is batch size. The batch size contributes towards convergence and performance of the network. By using batches, the variance of the update is reduced. Larger batch size contribute towards unstable update vectors which converges slowly. I set the batch size as 256.

Another important parameter is the learning rate which determines the time for convergence of the network. Smaller learning rates converge very slowly and larger learning rates may not converge to global minima. I set the learning rate as 0.001.

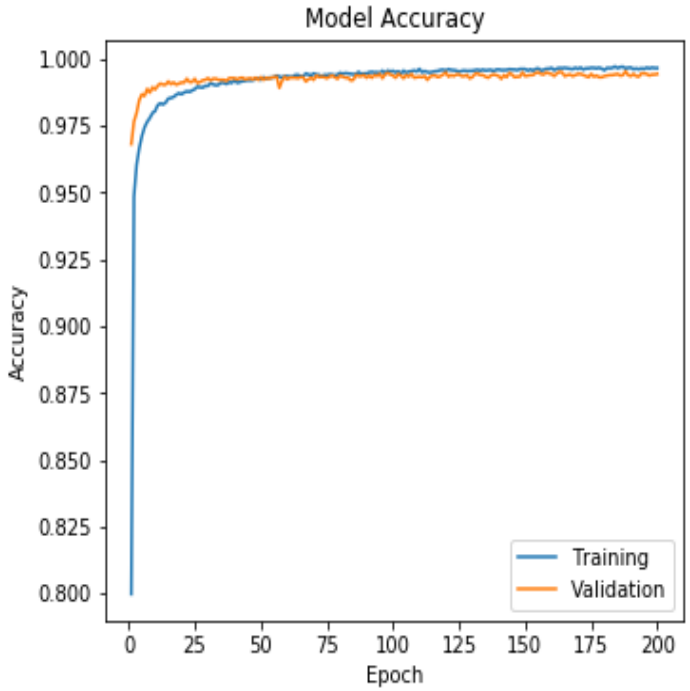
I used the Adam optimizer as the optimizer and also to prevent overfitting, the training set was randomly shuffled after every epoch and trained in batches. The experiment was conducted for 200 epochs.

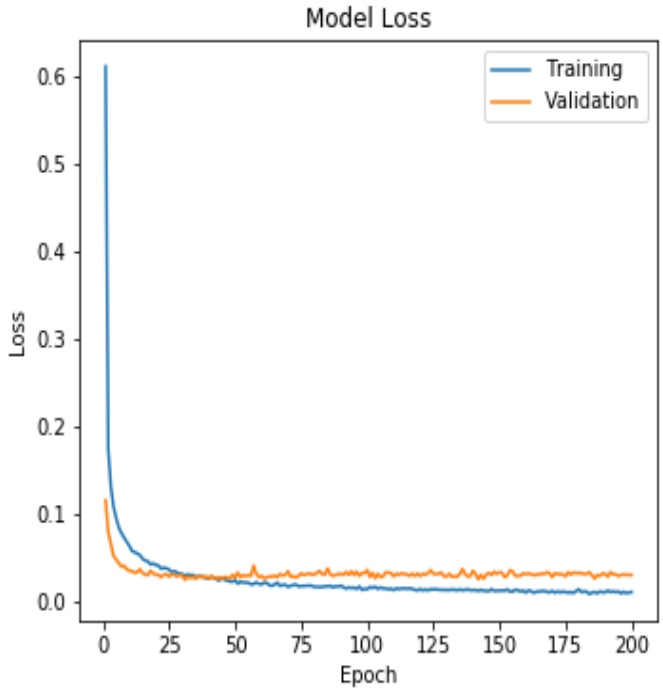
Another important factor to consider is dropout which indicates what portion of the neurons are to be dropped in layer during a batch. This is done to reduce overfitting. I applied a dropout rate of 0.2 after the second convolution layer and a dropout of 0.5 after the first fully connected layer.

III. Discussion

The total number of trainable parameters in the network : 81,053
Model took 8639.67 seconds to train

Test Error: 0.5000%
Test Loss: 0.0292%
Test Accuracy: 99.5000%

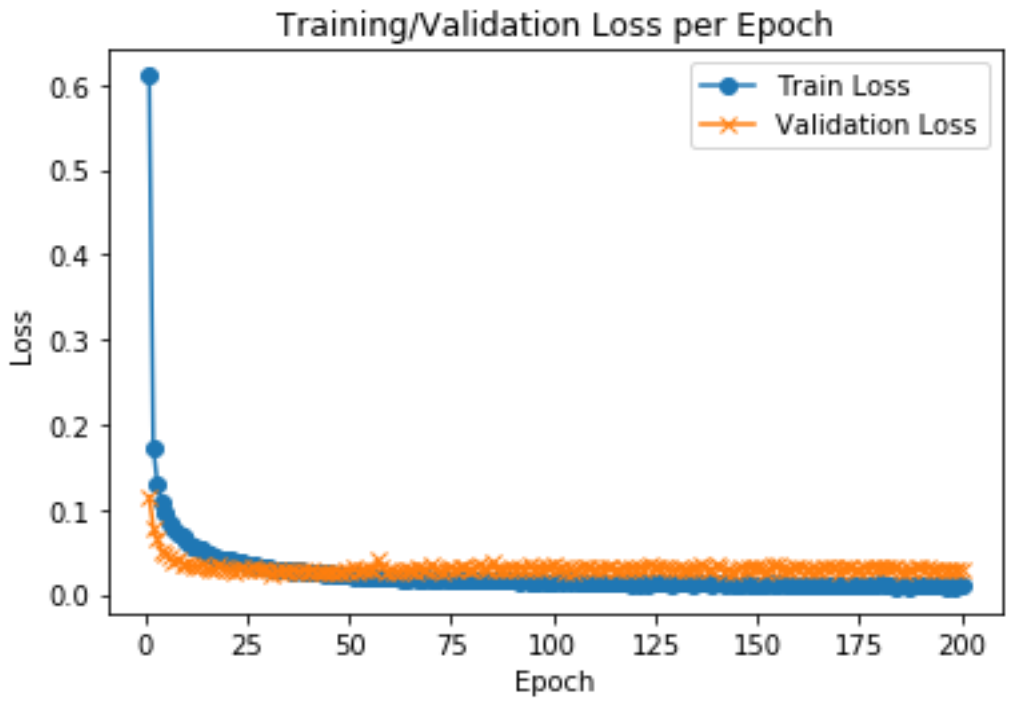


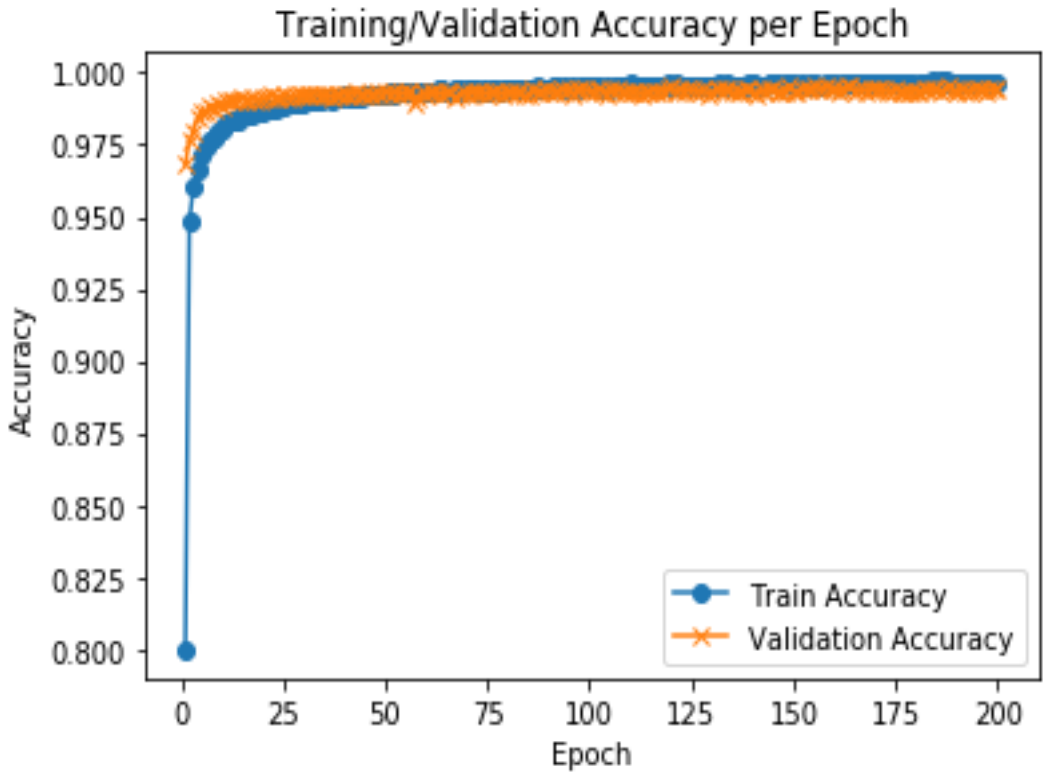


The test accuracy I could reach with this model was 99.5% which was more than the LeNet-5 Architecture which had a test accuracy of 99.18% as seen from the results in the previous question.

This is not an optimized network topology. Nor is a reproduction of a network topology from a recent paper. There is a lot of opportunity for us to still tune and improve upon this model. But this was the best error rate I could achieve over this time period.

One of the important factors which proved a difference was the selection of Adam optimizer over SGD and setting a batch size of 256. Also changing the kernel sizes from (5,5) to (3,3) in the second convolution layer helped to increase the test accuracy.





Problem 2.(a)

I. Abstract and Motivation

In this problem we are asked to do a comparison between SAAK Transform and CNNs based on our understanding of the paper "On Data-Driven Saak Transform" introduced by Professor Kuo.

II. Discussion

The Saak transform is a abbreviation for Subspace Approximation with Augmented Kernels. It consists of three basic steps:

1. The first step constructs the optimal linear subspace approximation with orthonormal bases. This step uses the PCA to do this along with the second-order statistics of second order.
2. The transform kernels in the first step are augmented with their negatives.
3. The ReLu activation is applied to the transformed outputs from the previous step.

One of the most significant factors which led to the development of the Saak transform was the elimination of approximation loss and rectification loss during the RECOS transform. CNN faced a couple of problems: the RECOS transform showed that it faced both the above mentioned losses. The use of limited number of transform kernels led to approximation loss. If we use more number of filters to reduce this error, we will increase the computational complexity and will require higher memory, thus leading to further problems. The nonlinear activations in the architecture caused the rectification loss.

In SAAK transform, the first step is to use the second order statistics and select orthonormal eigenvectors of the covariance matrix as transform kernels using PCA. This is done to get an optimal subspace approximation. IN cases of high input dimensions, rather than using a single stage PCA, we decompose the higher dimension vector into multiple lower dimension vectors. Then this process can be done repeatedly to form a hierarchical structure. To resolve any 'sign confusion' problems in such cases, ReLu activation function is used in between. Also kernel augmentation is done during this step to eliminate rectification loss. When an input vector is projected on the positive and negative kernel pair, ReLu activation function allows only one of the kernels to pass. Thus, cascaded multiple Saak transform can be used for large image sizes.

Although CNN and multi stage Saak transform both use the ReLu activation problems, there is a difference in their filter weight selection method. In CNN, the training data and the associated class labels determine the filter weights. After initialization, CNNs form a cost function which is optimized iteratively via back propagation. This back propagation is thus used to update the filter weights till an optimal value is reached. The number of iterations which takes place is very large. These iteratively optimized features are called the deep or the learned features.

The Saak transform on the other hand uses the second order statistics of the input vector at each stage to find the filter weights. This process is forward oriented and no backpropagation is required for this step. Thus computation complexity decreases. Even data labels are not required to find the filter weights. The CNN solution has multiple problems such as robustness against perturbation, scalability against the class number and portability among different datasets. These problems are dealt with by the Saak transform.

The SAAK coefficients are generated in 5 stages, reducing the feature set in each stage. For each input image, we select the non-overlapping block region with spatial size 2x2. Then calculate the variance of each block and remove the small variance blocks. Next, subtract its own mean of each block. Then perform the Principle Component Analysis (PCA) to the zero-mean block data and get the PCA transform matrix. Transform all the input data patches by selecting the important spectral components in the PCA matrix. Now we take this representative set of samples and determine the KLT basis functions. The DC coefficient is same as that in the RECOS transform, such that uniform harmonic mean distance is maintained from the unit circle circumference. The remaining AC coefficients are obtained by augmentation. Then we project the samples onto the augmented kernel set and apply ReLU.

Another difference is that inverse of CNN solution has not been studied and utilized. Whereas inverse of SAAK Transform can help in image generation. This has already been studied and documented. GAN and VAE are two methods which are based on the CNN architecture that helps in image generation, i.e., similar to the work of inverse SAAK Transform.

This is the way Saak transform works and is differnent from CNNs.

Problem 2.(b)

I. Abstract and Motivation

In this problem we are asked to apply the SAAK Transform to the MNIST dataset. We have to change the feature dimension to 32, 64 and 128 and then report on the accuracies.

II. Approach and Procedures

The code was available on GitHub. It was written in Python. Some changes needed to be done with it for RF and SVM classifiers. The basic methodology was as follows:

For every input image, we select non-overlapping patches of size 2x2. The variance of each patch is calculated and patches with small variance are removed. Then PCA is applied to zero mean patch data. Important spectral components are selected by PCA and a transform matrix is formed. Then, sign to position conversion is used to augment these transform matrices. This process is repeated for the five Saak Transform stages. The number of important components in each stage are : 3,4,7,6,8.

After getting the response from all the stages, we get a feature vector of dimension 2048. We perform PCA and F-test on it to reduce the dimension to 32, 64 and 128. Then we perform Classification using SVM and RF.

III. Results and Discussion

For the case of feature vector of 32 dimensions,

Extracting ./data/train-images-idx3-ubyte.gz
Extracting ./data/train-labels-idx1-ubyte.gz

Extracting ./data/t10k-images-idx3-ubyte.gz
Extracting ./data/t10k-labels-idx1-ubyte.gz
Input MNIST image shape: (1000, 28, 28, 1)
Resized MNIST images: (1000, 32, 32, 1)
Start to extract Saak anchors:

Stage 1 start:
Extracted image batch shape: (256, 1000, 2, 2, 1)
Processed image batch shape: (256000, 4)
Number of anchors to keep: 4
Anchor vector shape: (4, 4)
Augmented anchor vector shape: (8, 4)
Reshaped anchor shape: (8, 2, 2, 1)
Tensorflow formated anchor shape: (2, 2, 1, 8)
Saak coefficients shape: (1000, 16, 16, 8)
Stage 1 end

Stage 2 start:
Extracted image batch shape: (64, 1000, 2, 2, 8)
Processed image batch shape: (64000, 32)
Number of anchors to keep: 32
Anchor vector shape: (32, 32)
Augmented anchor vector shape: (64, 32)
Reshaped anchor shape: (64, 2, 2, 8)
Tensorflow formated anchor shape: (2, 2, 8, 64)
Saak coefficients shape: (1000, 8, 8, 64)
Stage 2 end

Stage 3 start:
Extracted image batch shape: (16, 1000, 2, 2, 64)
Processed image batch shape: (16000, 256)
Number of anchors to keep: 256
Anchor vector shape: (256, 256)
Augmented anchor vector shape: (512, 256)
Reshaped anchor shape: (512, 2, 2, 64)
Tensorflow formated anchor shape: (2, 2, 64, 512)
Saak coefficients shape: (1000, 4, 4, 512)
Stage 3 end

Stage 4 start:
Extracted image batch shape: (4, 1000, 2, 2, 512)
Processed image batch shape: (4000, 2048)
Number of anchors to keep: 2048
Anchor vector shape: (2048, 2048)
Augmented anchor vector shape: (4096, 2048)
Reshaped anchor shape: (4096, 2, 2, 512)

Tensorflow formated anchor shape: (2, 2, 512, 4096)
Saak coefficients shape: (1000, 2, 2, 4096)
Stage 4 end

Stage 5 start:
Extracted image batch shape: (1, 1000, 2, 2, 4096)
Processed image batch shape: (1000, 16384)
Number of anchors to keep: 1000
Anchor vector shape: (1000, 16384)
Augmented anchor vector shape: (2000, 16384)
Reshaped anchor shape: (2000, 2, 2, 4096)
Tensorflow formated anchor shape: (2, 2, 4096, 2000)
Saak coefficients shape: (1000, 1, 1, 2000)
Stage 5 end

Build up Saak model
Prepare testing images
Compute saak coefficients
Save saak coefficients
Saak feature dimension: 32

Do classification using SVM
Accuracy: 0.880

For the case of feature vector of 64 dimensions,

Extracting ./data/train-images-idx3-ubyte.gz
Extracting ./data/train-labels-idx1-ubyte.gz
Extracting ./data/t10k-images-idx3-ubyte.gz
Extracting ./data/t10k-labels-idx1-ubyte.gz
Input MNIST image shape: (1000, 28, 28, 1)
Resized MNIST images: (1000, 32, 32, 1)
Start to extract Saak anchors:

Stage 1 start:
Extracted image batch shape: (256, 1000, 2, 2, 1)
Processed image batch shape: (256000, 4)
Number of anchors to keep: 4
Anchor vector shape: (4, 4)
Augmented anchor vector shape: (8, 4)
Reshaped anchor shape: (8, 2, 2, 1)
Tensorflow formated anchor shape: (2, 2, 1, 8)
Saak coefficients shape: (1000, 16, 16, 8)
Stage 1 end

Stage 2 start:

Extracted image batch shape: (64, 1000, 2, 2, 8)
Processed image batch shape: (64000, 32)
Number of anchors to keep: 32
Anchor vector shape: (32, 32)
Augmented anchor vector shape: (64, 32)
Reshaped anchor shape: (64, 2, 2, 8)
Tensorflow formated anchor shape: (2, 2, 8, 64)
Saak coefficients shape: (1000, 8, 8, 64)
Stage 2 end

Stage 3 start:
Extracted image batch shape: (16, 1000, 2, 2, 64)
Processed image batch shape: (16000, 256)
Number of anchors to keep: 256
Anchor vector shape: (256, 256)
Augmented anchor vector shape: (512, 256)
Reshaped anchor shape: (512, 2, 2, 64)
Tensorflow formated anchor shape: (2, 2, 64, 512)
Saak coefficients shape: (1000, 4, 4, 512)
Stage 3 end

Stage 4 start:
Extracted image batch shape: (4, 1000, 2, 2, 512)
Processed image batch shape: (4000, 2048)
Number of anchors to keep: 2048
Anchor vector shape: (2048, 2048)
Augmented anchor vector shape: (4096, 2048)
Reshaped anchor shape: (4096, 2, 2, 512)
Tensorflow formated anchor shape: (2, 2, 512, 4096)
Saak coefficients shape: (1000, 2, 2, 4096)
Stage 4 end

Stage 5 start:
Extracted image batch shape: (1, 1000, 2, 2, 4096)
Processed image batch shape: (1000, 16384)
Number of anchors to keep: 1000
Anchor vector shape: (1000, 16384)
Augmented anchor vector shape: (2000, 16384)
Reshaped anchor shape: (2000, 2, 2, 4096)
Tensorflow formated anchor shape: (2, 2, 4096, 2000)
Saak coefficients shape: (1000, 1, 1, 2000)
Stage 5 end

Build up Saak model
Prepare testing images
Compute saak coefficients

Save saak coefficients
Saak feature dimension: 64

Do classification using SVM
Accuracy: 0.878

For the case of feature vector of 128 dimensions,

Extracting ./data/train-images-idx3-ubyte.gz
Extracting ./data/train-labels-idx1-ubyte.gz
Extracting ./data/t10k-images-idx3-ubyte.gz
Extracting ./data/t10k-labels-idx1-ubyte.gz
Input MNIST image shape: (1000, 28, 28, 1)
Resized MNIST images: (1000, 32, 32, 1)
Start to extract Saak anchors:

Stage 1 start:
Extracted image batch shape: (256, 1000, 2, 2, 1)
Processed image batch shape: (256000, 4)
Number of anchors to keep: 4
Anchor vector shape: (4, 4)
Augmented anchor vector shape: (8, 4)
Reshaped anchor shape: (8, 2, 2, 1)
Tensorflow formated anchor shape: (2, 2, 1, 8)
Saak coefficients shape: (1000, 16, 16, 8)
Stage 1 end

Stage 2 start:
Extracted image batch shape: (64, 1000, 2, 2, 8)
Processed image batch shape: (64000, 32)
Number of anchors to keep: 32
Anchor vector shape: (32, 32)
Augmented anchor vector shape: (64, 32)
Reshaped anchor shape: (64, 2, 2, 8)
Tensorflow formated anchor shape: (2, 2, 8, 64)
Saak coefficients shape: (1000, 8, 8, 64)
Stage 2 end

Stage 3 start:
Extracted image batch shape: (16, 1000, 2, 2, 64)
Processed image batch shape: (16000, 256)
Number of anchors to keep: 256
Anchor vector shape: (256, 256)
Augmented anchor vector shape: (512, 256)
Reshaped anchor shape: (512, 2, 2, 64)
Tensorflow formated anchor shape: (2, 2, 64, 512)

Saak coefficients shape: (1000, 4, 4, 512)
Stage 3 end

Stage 4 start:
Extracted image batch shape: (4, 1000, 2, 2, 512)
Processed image batch shape: (4000, 2048)
Number of anchors to keep: 2048
Anchor vector shape: (2048, 2048)
Augmented anchor vector shape: (4096, 2048)
Reshaped anchor shape: (4096, 2, 2, 512)
Tensorflow formated anchor shape: (2, 2, 512, 4096)
Saak coefficients shape: (1000, 2, 2, 4096)
Stage 4 end

Stage 5 start:
Extracted image batch shape: (1, 1000, 2, 2, 4096)
Processed image batch shape: (1000, 16384)
Number of anchors to keep: 1000
Anchor vector shape: (1000, 16384)
Augmented anchor vector shape: (2000, 16384)
Reshaped anchor shape: (2000, 2, 2, 4096)
Tensorflow formated anchor shape: (2, 2, 4096, 2000)
Saak coefficients shape: (1000, 1, 1, 2000)
Stage 5 end

Build up Saak model
Prepare testing images
Compute saak coefficients
Save saak coefficients
Saak feature dimension: 128

Do classification using SVM
Accuracy: 0.867

If we divide the datasets into different portions of training and testing data, we get the following results:

Applying both RF and SVM classifiers and reducing the number of features we get,

Size of dataset : Training data – 59000 Testing data – 10000

Classifier used – SVM
No. of features: 128 Training Accuracy – 0.9801 Testing Accuracy – 0.9760

No. of features: 64 Training Accuracy – 0.9887 Testing Accuracy – 0.9815
No. of features: 32 Training Accuracy – 0.9923 Testing Accuracy – 0.9831

Classifier used – RF
No. of features: 128 Training Accuracy – 0.9992 Testing Accuracy – 0.9057
No. of features: 64 Training Accuracy – 0.9989 Testing Accuracy – 0.9280
No. of features: 32 Training Accuracy – 0.9986 Testing Accuracy – 0.9325

Size of dataset : Training data – 50000 Testing data – 19000

Classifier used – SVM
No. of features: 128 Training Accuracy – 0.9789 Testing Accuracy – 0.9755
No. of features: 64 Training Accuracy – 0.9882 Testing Accuracy – 0.9809
No. of features: 32 Training Accuracy – 0.9921 Testing Accuracy – 0.9833

Classifier used – RF
No. of features: 128 Training Accuracy – 0.9989 Testing Accuracy – 0.9035
No. of features: 64 Training Accuracy – 0.9991 Testing Accuracy – 0.9234
No. of features: 32 Training Accuracy – 0.9990 Testing Accuracy – 0.9351

Size of dataset : Training data – 40000 Testing data – 29000

Classifier used – SVM
No. of features: 128 Training Accuracy – 0.9790 Testing Accuracy – 0.9703
No. of features: 64 Training Accuracy – 0.9879 Testing Accuracy – 0.9767
No. of features: 32 Training Accuracy – 0.9923 Testing Accuracy – 0.9804

Classifier used – RF
No. of features: 128 Training Accuracy – 0.9991 Testing Accuracy – 0.8972
No. of features: 64 Training Accuracy – 0.9986 Testing Accuracy – 0.9134
No. of features: 32 Training Accuracy – 0.9992 Testing Accuracy – 0.9247

From the results, we can see that SVM works better as a classifier than RF for this dataset as the test accuracies are more for SVM. Another result which comes to the fore is that as we reduce the dimension of the feature vector and select only the best features, we find that the test accuracies increase. This is true for both the SVM and RF classifiers. Thus test accuracy is the most for 32 feature vector dimension and least for 128 feature dimension. Also the best results are got in the first case when the dataset was divided as 59000 training data and 10000 testing data. These were my observations from the results which we got.

We also see, comparing the test accuracies for CNN and SAAK Transform, that SAAK Transform has better accuracies on testing data. The CNN architecture gave me a maximum accuracy of only 0.95 whereas SAAK has given accuracies of 0.98 too.

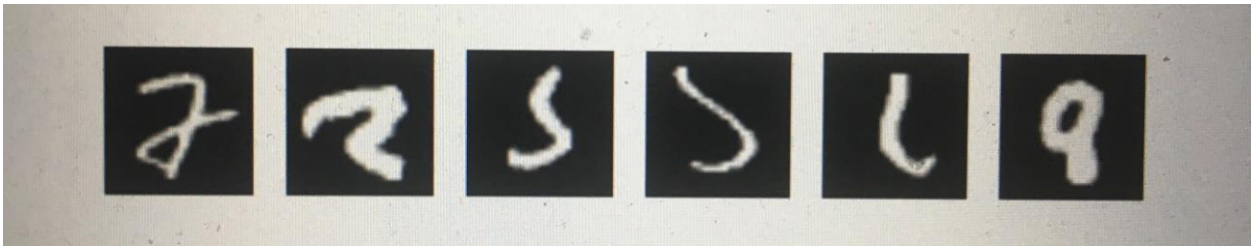
Problem 2.(c)

I. Abstract and Motivation

In this problem we are asked to do a comparison between SAAK Transform and CNNs based on classification errors. We are also asked to propose ideas to improve both the CNN solution and the SAAK solution.

II. Discussion

As can be seen from the test accuracies we got from the CNN solution and SAAK Transform solutions, the better accuracies were got in the latter method. This means there were more classification error cases in CNN solution. If we look at some of the errors in classification, by showing the digits using matplotlib functions, we see,



These are examples of some of the misclassified samples. The correct labels are 2,2,5,5,6,8 for the above images. But the CNN solution marked them as 7,7,4,4,1,9. These errors are sometimes even difficult for the human mind to interpret.

The SAAK Transform had less problems in classifying some of these errors. For example, it classified the digits 5 properly. It also correctly classified the second digit 2. But it had the same wrong classification errors on the other digits as CNN.

Thus, though SAAK has a better test accuracy, it still shows some common errors as CNN. But these errors are mostly those which even the human mind will have difficulty to interpret.

To improve upon the CNN architecture, we can try convolutional deep belief networks on the dataset. They have architecture like CNNs and use pre training like deep believe networks. Another problem with CNN are the large number of parameters to learn. If somehow, there can be a method to incorporate PCA or Fischer Discriminant Formula with the CNN architecture to reduce the feature dimensions to be learnt, it can be very useful.

For SAAK transform, a method for extracting discriminant features called DCFA which combines Linear Discriminant Ananlysis (LDA) and 1-D class dependence feature analysis. This method was proposed in an IEEE paper in 2010 and maybe a more effective method to extract discriminative features rather than using F-test scores.

For SAAK, improvement in discriminant feature extraction is the way to improve upon the accuracy. I think trying to implement the method proposed may help in increasing the test accuracy.

References:

- [LeNet-5]http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
- [MNIST] http://yann.lecun.com/exdb/mnist/h
- Saak Transform https://arxiv.org/abs/1710.04176
- Effective discriminant feature extraction framework for face recognition. **INSPEC Accession Number:** 11306064
- https://github.com/morningsyj/Saak-tensorflow
- https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/
- http://cs231n.github.io/convolutional-networks/
- https://www.tensorflow.org