EE 569: Homework #4

Table of Contents:

Problem 1- CNN Training and its application to MNIST dataset

A) CNN architecture and Training

I. Abstract and Motivation

Convolutional Neural Networks are considered an extension to Artificial Neural Networks wherein there is an input layer, an output layer and multiple hidden layers. Images have a hierarchical structure and CNN takes into account this hierarchal structure to predict the output classes. Convolutional Neural Networks are very similar to ordinary neural networks as even they are made up of neurons having learnable weights and biases. In Convolutional Neural Networks, the pixels in the image behave as features and each neuron takes an input and computes the dot product of the input values and the weight of the neuron. It then applies a bias and passes it through nonlinear activation function to predict the output scores. These output scores have a loss function attached to it to detect the difference between the predicted scores and the ground truths. The network then optimizes over this loss function to give the best possible scores.

As the number of dimensions in an image is increased, the number of parameters to be tuned increases which increases the computational capacity in Artificial Neural Networks. The only difference between ordinary neural networks and Convolutional Neural Network is that in CNN parameter sharing takes place which decreases the computational complexity of the entire network and avoids overfitting. CNN has a lot of applications in object detection, tracking, classification etc.

CNN have different layers with different functionalities namely Convolutional Layer, Max pooling layer, Fully connected layer. In CNN, the neurons are stacked up in 3 dimensions namely width, height and depth. Hence, weights are stacked up in the 3D volume and each 2D slice has learnable parameters which extracts important characteristics of the image. Hence, in the learning process, the weights of the number of filters are learnt so as to identify the characteristics of the image. The outputs from these filters is then stacked to generate the output volume. This is similar to the Laws filters used to extract segments in the texture, except that in the case of CNN, the weight of the filters is not predefined but is learnt to give the best possible output.

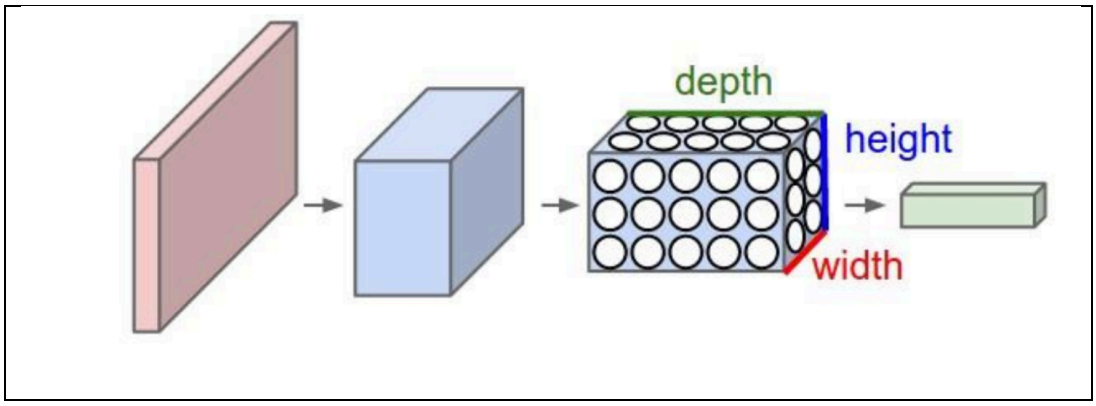


Figure 1: Architecture of Convolutional Neural Network
(Source: http://cs231n.github.io/convolutional-networks/)

The CNN architecture used in this question is LeNet 5 which was developed by LeCun et all to detect hand written digits. The CNN architecture using LeNet 5 is as given below:

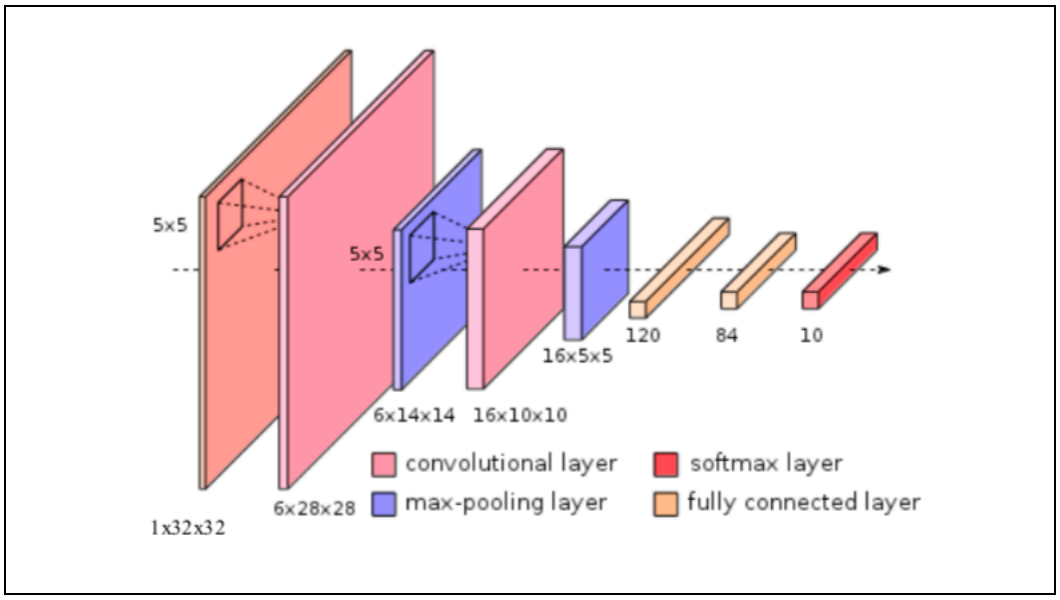


Figure 2: CNN architecture using LeNet 5

II. Discussion

Answer a) The CNN architecture can be explained using the following components:

i.) Convolutional Layer
ii.) Max Pooling layer
iii.) Fully Connected Layer
iv.) Activation Function
v.) Softmax Function

i.) Convolutional Layer

The convolutional layer is computationally the most effective layer as maximum computations are done in this layer. This layer is used to detect the spatial similarities in the image. In this layer, instead of connecting all the neurons to all the pixels in the input image, the neurons are connected to small portions of the input image. Parameter sharing takes place in this layer as each 2D filter has the same weight associated with it. Each filter in this layer generates a 2D activation map by sliding over the input regions of the image. Each filter has learnable weights which are learnt to detect special characteristics of the image. After sliding the filters over the image, the 2D activation map generated gives the features that generated the highest response for that image. Each filter takes a dot product of the weights and the input portion of the image and slides it over the entire image to generate a 2D activation map.

The filter is designed in a such a way that it covers the entire depth of the input image and generates an output volume proportional to the number of filters used in the layer. An activation function is applied to the neurons of the convolutional layer to determine which neurons are getting fired. Normally, a ReLU function is used as an activation function.

Following components are required to be specified to construct this layer:

- Size (receptive field) of the filters
- Number of filters used- This decides the depth of the output volume
- Stride- which decides the steps taken while sliding through the image
- Padding- which decides if the boundaries of the input image have to be extended

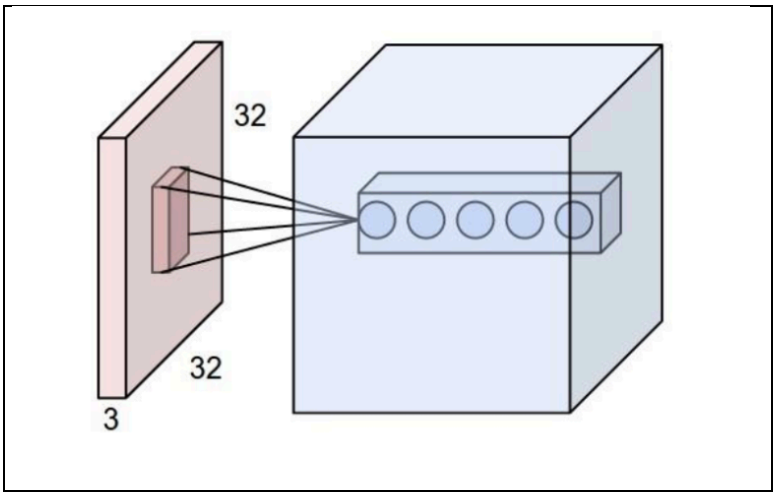


Figure 3: Convolutional Layer Architecture (Source: http://cs231n.github.io/convolutional-networks/)

From the figure given above, it can be understood that in the input volume of 32x32x3 which defines the height, width and the depth of the image respectively is transformed into an output volume of 32x32x5 indicating that 5 filters were used in this layer whose outputs are stacked along the depth of the output volume.

The size of the output volume can be calculated as follows:

$$output\ volume = \frac{inout\ volume - F + 2P}{S} + 1$$

Where F denotes the receptive field or the size of the filters used

P denotes the padding used for the input image

S denotes the stride with which the filter moves over the image

In the LeNet-5 architecture,

$$output\ volume = \frac{32 - 5 + 0}{1} + 1 = 28$$

Hence after applying the first convolutional layer, an output volume of 28x28x6 is generated.

ii.) Max Pooling layer

The max pooling layer is mainly used to reduce the spatial size of the image i.e. to decrease the computations in the image. This helps in controlling the overfitting problem prevalent in machine learning methods. In the maximum pooling layer there are zero learnable parameters. This layer is placed in between the convolutional layer and it helps to decrease the dimensions considerably. In this layer, the spatial height and width of the filters from the previous layer is reduced, however, the depth is maintained. Generally, a 2x2 max pooling layer is used which has a stride of 2 is used on every depth slice, which reduces the size by 2 by eliminating 75% of the activations. In a 2x2 filter with stride 2, the maximum of the 4 elements is taken and replaced in the output.

Following components are required to be specified to construct this layer:

- Size of filter
- Stride

The maximum operation used in this layer is represented as below.

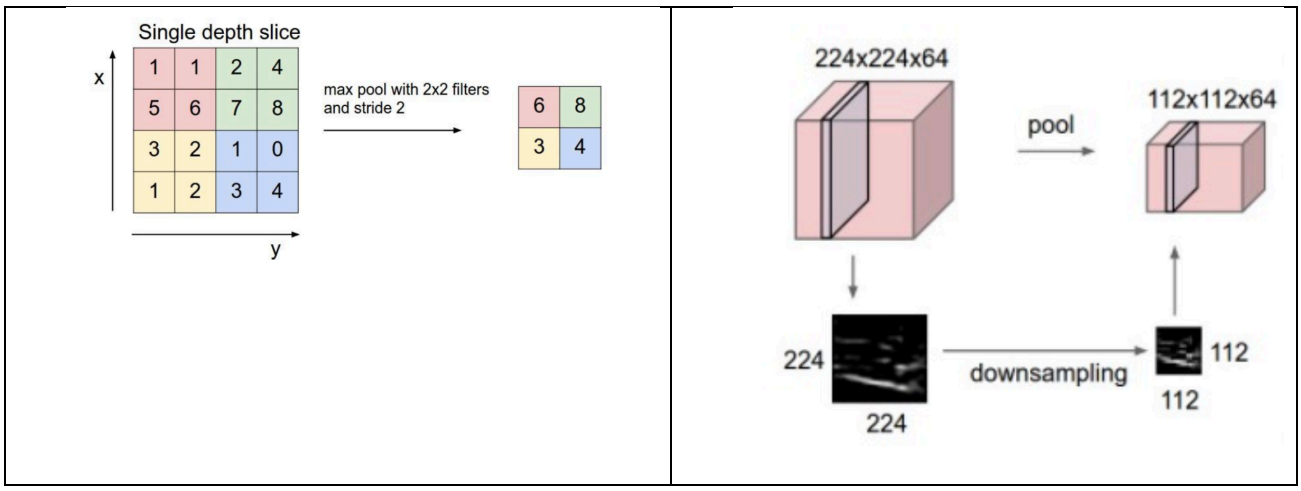


Figure 4: (Source: http://cs231n.github.io/convolutional-networks/)

This is how the dimensions of the input image are reduced by using a filter of size 2x2 and stride 2.

iii.) Fully connected Layer

A fully connected layer is similar to the ordinary neural network as in this layer all the neurons in the layer are connected to the neurons in the previous layer. However, there is no connection between the neurons of the particular layer. This layer has maximum receptivity as it has the capability of forward and backward propagation. As there are full connections to previous layer, a matrix can be used to compute the dot products. Hence, though the computations are huge, they are simple to implement. The fully connected layer basically takes input from the convolution or the max pooling layer and generates an output vector dependent on the number of classes. Dropouts can be used in this layer to avoid overfitting and reduce the number of computations. The number of neurons and the activation function to be used have to be specified in this layer.

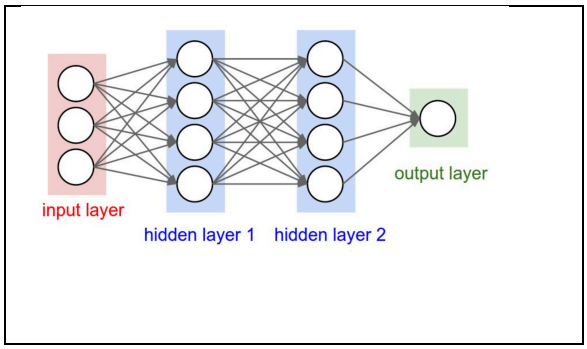
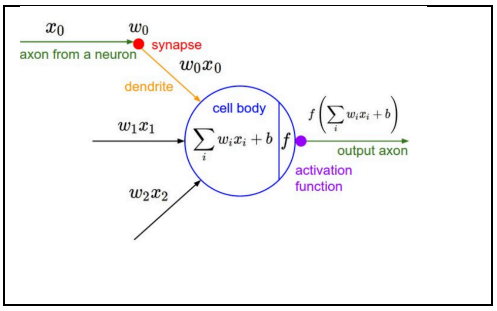


Figure 5: A fully connected layer (Source: Discussion slides)

iv.) Activation Function

An activation function basically takes a layer and performs some mathematical operation on it. This is required to bring in some non-linearity in the system. Suppose an artificial neuron takes a weighted sum of all its inputs, adds a bias to it and generates an output. This output value is then given to an activation function which decides if the respective neuron would be fired or not. This mechanism can be summarized by showing its correspondence in case of neurons in the brain.



The above figure shows the firing of neuron in the brain depending upon the output generating by summing the biases. The same concept is used in CNN. The activations functions used are nonlinear as summing up with a linear activation function would give a linear output which won't be useful. These activation functions have zero learnable parameters.

The following activation functions are used in practice. Every activation function has its pros and cons.

a) Sigmoid activation function

The sigmoid function can be represented using the mathematical expression:

$$\sigma(x)=1/(1+e-x)$$

The sigmoid function takes real values and compresses them between 0 and 1. Therefore, large negative numbers are converted to zero and large positive values are converted to 1. The sigmoid function is represented as follows:

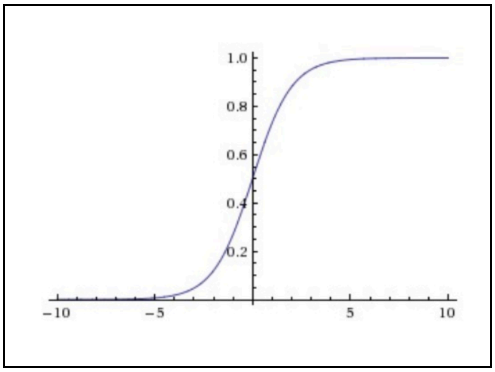


Figure 6: Sigmoid function (Source: Discussion slides)

However, the sigmoid function has certain cons. It saturates and kills the gradients during back propagations. Also, as the sigmoid function is not centered around zero, it leads to zigzagging patterns during gradient update, leading to slower convergence.

b) Tanh activation function

The tanh function is a scaled value of the sigmoid function. It is given as:

$$\tanh(x)=2\sigma(2x)-1$$

The tanh function takes real values numbers are squishes them in the range -1 to 1. This function also has the drawback of saturating and killing the gradients. But as the function is zero centered, it converges much faster than sigmoid function.

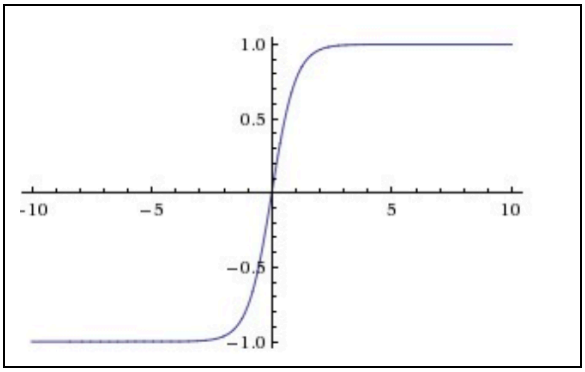


Figure 7: Tanh activation function (Source: Discussion slides)

c) ReLU activation unction

The ReLU function is used very widely as it uses the mathematical formula f(x)=max(0,x) to threshold its values to be greater than zero. The ReLU function has a linear, non-saturated form in the positive region. Hence, it converges quickly compared to tanh and sigmoid functions. The output values obtained after applying the ReLU function are not zero-centered.

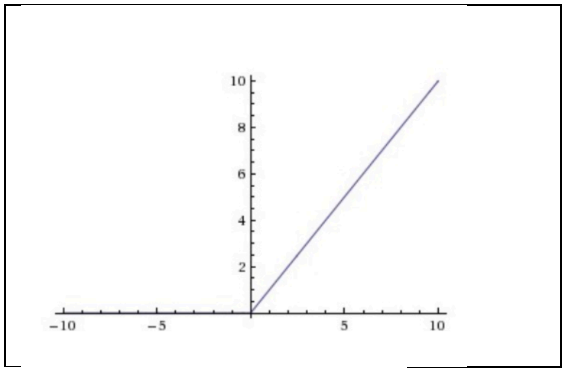


Figure 8: ReLU activation function (Source: Discussion slides)

The ReLU function has some drawbacks. The ReLUs can die out during training if proper learning rate parameter is not selected. In such a case, the output remains unchanged i.e. the output remains same as that of the input. Hence, the discriminating power of the ReLU diminishes and it's very hard to recover once the ReLU is stuck in this stage.

d) Leaky ReLU activation function

The Leaky ReLU function is similar to the ReLU function as it is also not zero centered and it converges faster than sigmoid or tanh functions. However, the Leaky ReLU activation function performs better than the ReLU activation function as it has a small positive slope in the negative direction unlike ReLU function which has a zero slope in the negative region. Hence, due to this change the output does not saturate even if the learning parameter is not properly set.

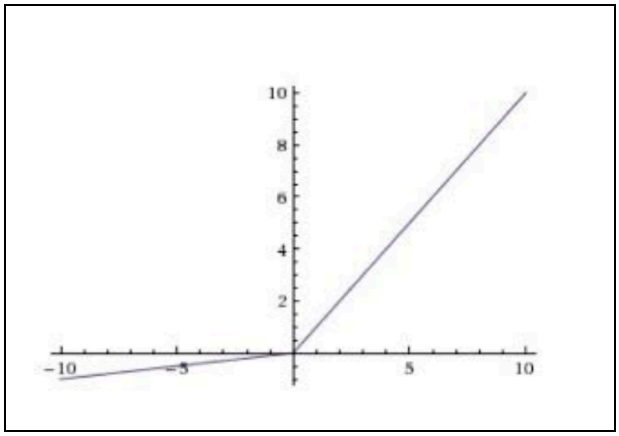


Figure 9: Leaky ReLU function (Source: Discussion slides)

v.) Softmax function

The softmax function is a generalization of the logistic regression function wherein it takes K dimensional arbitrary real values and transforms them into K-dimensional vectors of real values in the range (0,1) which add up to 1. The softmax function is an extension of the binary logistic regression classifier and it used to generalize to the multi class case. The softmax function calculates the losses between the predicted labels and the ground truths and then uses back propagation to minimize the loss. The softmax function is used in the last stage of the neural network. The softmax function is as given below:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

The cross entropy between a true distribution p and an estimated function q is given as:

$$H(p,q) = -\sum x p(x) \log q(x)$$

The softmax function hence tries to reduce the loss between the estimated probabilities and the actual probabilities.

To summarize the architecture of LeNet 5 in this problem:

- Input – Gray scale images of size 32x32x1.
- Convolutional Layer 1: It has input size of 32x32x1 which is transformed into size 28x28x6 using 6 filters of size 5x5 and using no padding and stride 1. The total number of learning parameters in this stage is (5x5x3+1(bias)) =76 weights.
- Max pooling layer 1: This is a 2x2 layer with stride 2 which reduces the output volume of 28x28x6 after the application of convolutional filter 1 to 14x14x6 after the pooling layer.

- Convolutional Layer 2: It has input size of 14x14x6 which is transformed into size 10x10x16 using 16 filters of size 5x5 and using no padding and stride 1. The total number of learning parameters in this stage is (5x5x3+1(bias)) =76 weights.
- Max pooling layer 2: This is a 2x2 layer with stride 2 which reduces the output volume of 10x10x16 after the application of convolutional filter 2 to 5x5x16 after the pooling layer.
- Fully connected layer 1: This layer has 120 neurons which are connected to all the neurons in the previous layer.
- Fully connected layer 2: This layer has 80 neurons which are connected to all the neurons in the previous layer.
- Fully connected layer 2: This layer has 10 neurons which are connected to 80 neurons in the previous layer. The 10 neurons in the last stage correspond to the 10 label categories for the input image.

Answer b)

Overfitting is a phenomenon in machine learning wherein the model works well for the given data but fails to generalize well over unseen data. The model generated after training the classifier should generalize well on the unseen data offered to it. While training a classifier, sometimes a lot of details are extracted from the training data which is particular to only the data which is provided. Hence, the accuracy obtained for that data is really high. But when unseen data is provided to the model, these details used for training are insignificant and hence, the accuracy obtained becomes too low. To avoid overfitting, cross validation is used in machine learning. The training data is split into validation data and training data and using the validation data, the hyper parameters are tuned. Due to this, the model generalizes well. Also, more data is used for training so that the model becomes well trained.

In CNN, the problem of overfitting is quite common. Regularization loss is monitored in CNN so as to avoid the problem of overfitting. The regularization function is a function of weights wherein the weights are distributed, and the clustering of large weights is avoided. When the weights are distributed properly, the problem of overfitting is reduced. To implement this in practice, max pooling layers are used to reduce the size of the layers and hence decrease the number of computations. The max pooling layers are included in between the convolutional layers and they reduce the height and width of the image but keep the depth constant. Also, dropouts are used in the fully connected layer wherein a neuron is kept active with some probability p, else it is set to zero. This is how random neurons are deactivated to deal with the problem of overfitting.

Answer c) Advantages of CNN over other traditional methods

Image Classification is a process in which we have images that are to be labelled as categories. Our primary job is to classify unknown images into these categories and measure their accuracy. Some traditional methods such as KNN, ANN, SVM were used earlier to classify images. However, now-a-days CNN is used instead of these traditional methods due to the following advantages:

- Though traditional methods like KNN take less time to train, the time taken to classify unknown samples is very high. On the contrary, in case of CNN, the time taken to train is high, but it is very easy to classify unknown images. Hence, this proves to be beneficial in image classification problems.
- CNN takes into account hierarchy of the image. Ever layer in the CNN extracts some important characteristics of the image. For instance, lines or curves in the image are detected by neurons in each layer which proves to be effective while classifying unknown samples. On the other hand, traditional methods like KNN don't take into consideration hierarchy of the image and hence are not as effective as CNN.
- The parameters in CNN can be tuned using Back Propagation. This is not used in other methods and hence the accuracy of classifying test images is not as high as compared to CNN.
- In CNN, the features are extracted automatically depending on how the layers are trained and what are the weights in the underlying filters. However, in traditional methods for image classification, hand crafted features have to be provided which may lead to erroneous labelling of images. This is undesirable and hence, CNN are preferred.
- In CNN, in the convolutional layer parameter sharing takes place. Hence, the memory requirement is reduced considerably which isn't the case in other methods used in computer vision applications.

Answer d) Loss Function and Back Propagation

The main purpose of a classifier is to predict scores for the images supplied to it as inputs. The score function does the job of transforming pixel values into class scores using class weights W. We don't have control over the training data which is supplied to us or on the test data which we have to predict scores on. However, we do have a control on the weights for the neurons in each layer. The weights can be decided in such a way that they predict labels close to the ground truth labels. But sometimes, there is a deviation in the predicted outputs and the ground truths. Hence, a loss function is required which measures this difference between the observed output and the expected output. The loss function is used in optimization procedures to reduce the cost to obtain the desired results.

For instance, while predicting the image of a dog on a trained classifier, if the weights obtained are optimal, the probability of dog output in the final one hot encoded output will be high. However, if the training of weights is not proper, the probability of getting dog score at the output is less. In this case the loss function will be more. Now, the classifier will try to obtain weights in such a way that the loss function is less, and it will use back propagation for this purpose.

Back propagation is used to minimize the loss function at the output. Given an error function of the obtained output with respect to the desired output, the back propagation method calculates the gradient of errors with respect to the weights in the network. In backward propagation, the gradient of the weight in the final layer is calculated first and propagated backward throughout the network. The backward propagation of error leads to effective calculation of weights in the network such that the predicted output is close to the desired output.

Optimizers play a very important role in the back propagation mechanism as they update the weights depending on the error and hence the learning rate in them should be set high. The optimizers normally used are stochastic gradient descent and Adam optimizer. This mechanism of back propagation generally requires the output labels so as to calculate the error and hence is a supervised learning method.

B) Train LeNet-5 on MNIST Dataset

I. Abstract and Motivation

Le-Net-5 architecture is a very good architecture used in Convolutional Neural Networks which gives a decent accuracy. The main purpose of this part was to make some changes to the LeNet-5 network to obtain the best accuracies.

II. Approach and Procedure

While specifying the architecture, all the parameters have to be provided so that the model can train and test on unknown data. The following parameters were kept constant in the LeNet-5 architecture:

- **Convolutional Layer 1:** It has input size of 32x32x1 which is transformed into size 28x28x6 using 6 filters of size 5x5 and using no padding and stride 1.
- **Max pooling layer 1:** This is a 2x2 layer with stride 2 which reduces the output volume of 28x28x6 after the application of convolutional filter 1 to 14x14x6 after the pooling layer.
- **Convolutional Layer 2:** It has input size of 14x14x6 which is transformed into size 10x10x16 using 16 filters of size 5x5 and using no padding and stride 1.
- **Max pooling layer 2:** This is a 2x2 layer with stride 2 which reduces the output volume of 10x10x16 after the application of convolutional filter 2 to 5x5x16 after the pooling layer.
- **Fully connected layer 1:** This layer has 120 neurons which are connected to all the neurons in the previous layer.
- **Fully connected layer 2:** This layer has 80 neurons which are connected to all the neurons in the previous layer.
- **Fully connected layer 2:** This layer has 10 neurons which are connected to 80 neurons in the previous layer.

However, I tried changing the other setting in the network. The significance of each of these settings can be explained as follows:

- **Weight Initializations:** After the network is built, weights have to be initialized for each layer. These weights are then optimized by using the back propagation mechanism. I tried changing the weight initializations to be Xaviers weight initialization method and

Random uniform weight initialization method. Both of them gave different accuracies according to other parameter settings.

- **Learning Rate:** The Learning rate parameter in the network decides how fast the optimization process converges. Choosing a very high value for the learning rate may not give us a local minimum whereas choosing a low value for learning rate may take a lot of time for convergence. Hence, different values are tried for this setting which are 0.01,0.1 and 0.5. However, according to the observations it can be inferred that the maximum value is obtained for learning rate value of 0.01.
- **Activation functions:** As there are a lot of options on deciding the activation function for the network, I tried different setting for this as well. Different parameter settings have different effects on the accuracy. I used the ReLU, leaky ReLU, tanh, sigmoid and linear activation functions.
- **Optimizers:** The optimizers decide how the back propagation mechanism will work and hence prove to be extremely important while designing an neural network. I tried the Stochastic Gradient Descent, Adam, Adadelta, Adagrad and RMS prop. The Stochastic Gradient descent (SGD) took a lot of time to converge and hence proved to be a bad optimizer.
- **Dropouts:** Dropouts are added after the fully connected layer so as to avoid the overfitting of data. Adding the dropouts decrease the computation time as well. I tried two values for dropouts namely 0.2 and 0.5.
- **Epochs:** The process of forward pass, estimating the loss function, back propagating the weights and updating the weights is termed to be one epoch. This parameter was set to 100 in my observations.
- **Batch size:** This is a very important parameter in neural networks. The batch size specifies how many training samples are provided to the network at any given time. In my observations, the batch size was kept constant which was 512.

The parameter settings can be summarized as follows:

Activation functions = Linear, Leaky ReLU, ReLU, tanh, sigmoid
Dropouts = 0.2, 0.5
Optimizers = SGD, Adadelta, Adam, Adagrad, RMSprop'
Learning Rate = 0.01, 0.1, 0.5
Weight Initializations = glorot_uniform, RandomUniform
Maximum Number of epochs = 100
Batch size = 512

Different parameters mentioned above were changed and the total number of 600 combinations were generated using the above-mentioned values. The best values have been summarized below. Also, the plot of Accuracy vs epochs is generated and the plots with best value of accuracy have been displayed.

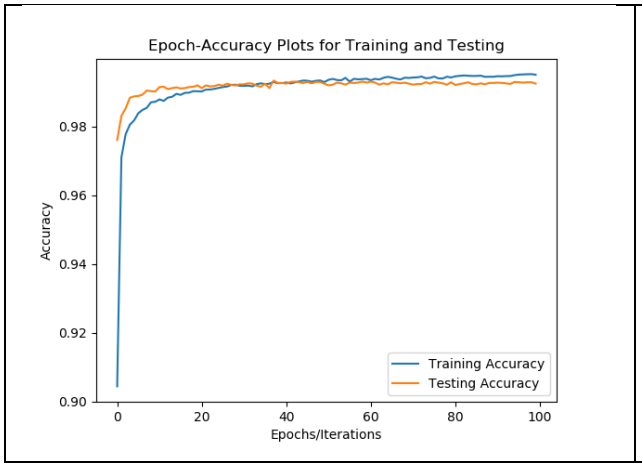
III. Experimental Results

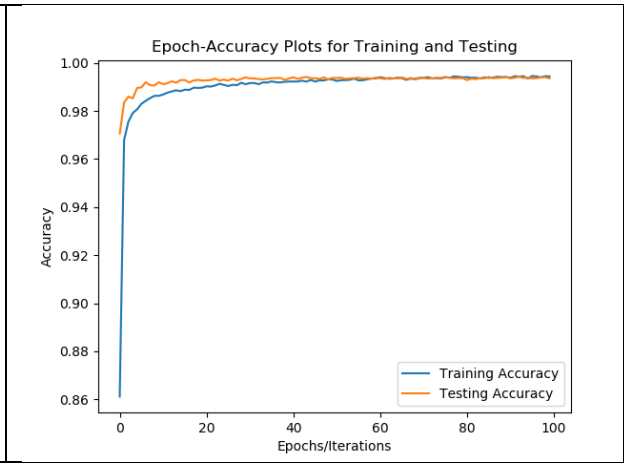
The best accuracies of the different parameter setting have been included. The best accuracies were those having values greater than 99%.

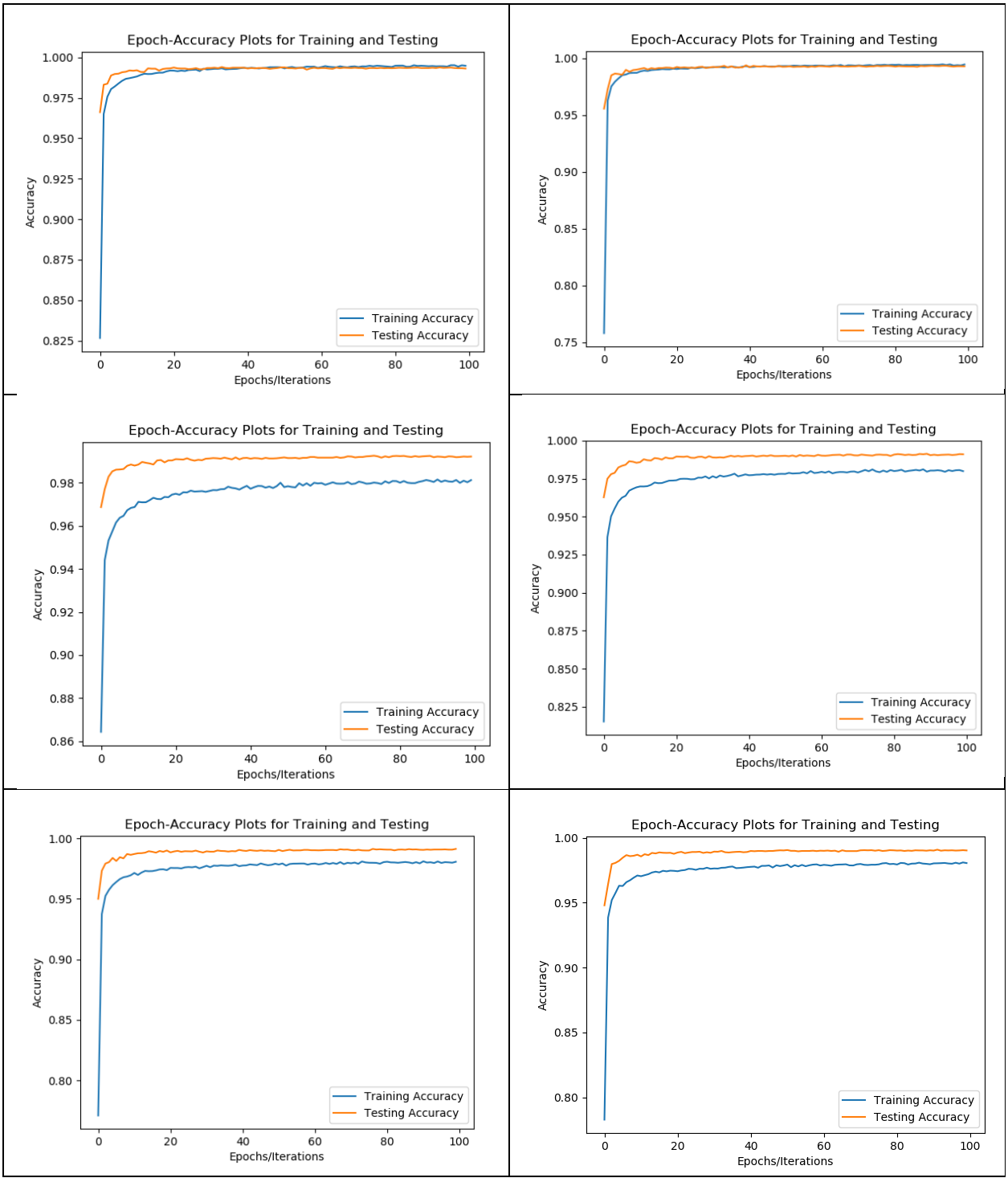
Iterations	Activation	Dropout	Optimizer	lr_decay	Weight_initilizations	Tran_loss	Train_Accuracy	Test_loss	Test_Accuracy
12	LeakyReLU	0.2	Adam	0.01	glorot_uniform	0.00407333	0.999066878	0.0242039	0.992512
13	LeakyReLU	0.2	Adam	0.01	RandomUniform	0.00491764	0.998916565	0.0197192	0.993634
24	LeakyReLU	0.2	RMSprop	0.01	glorot_uniform	0.00404187	0.999122	0.021815	0.993154
25	LeakyReLU	0.2	RMSprop	0.01	RandomUniform	0.00521776	0.99544	0.021575	0.993655
42	LeakyReLU	0.5	Adam	0.01	glorot_uniform	0.01900584	0.9943323	0.023402	0.992286
43	LeakyReLU	0.5	Adam	0.01	RandomUniform	0.02104578	0.993516386	0.0263521	0.9914
54	LeakyReLU	0.5	RMSprop	0.01	glorot_uniform	0.02089176	0.99384	0.0269971	0.991345
55	LeakyReLU	0.5	RMSprop	0.01	RandomUniform	0.02267754	0.9929454	0.0285856	0.990245
312	ReLU	0.2	Adam	0.01	glorot_uniform	0.02267776	0.999183444	0.0214574	0.993279
313	ReLU	0.2	Adam	0.01	RandomUniform	0.02267759	0.9984544	0.0196877	0.993164
324	ReLU	0.2	RMSprop	0.01	glorot_uniform	0.02267758	0.99905767	0.0220943	0.992269
325	ReLU	0.2	RMSprop	0.01	RandomUniform	0.02267774	0.997665675	0.0207447	0.993447
342	ReLU	0.5	Adam	0.01	glorot_uniform	0.02267729	0.992833667	0.029814	0.9906555
354	ReLU	0.5	RMSprop	0.01	glorot_uniform	0.02267796	0.993445	0.0276844	0.9919565
412	tanh	0.2	Adam	0.01	glorot_uniform	0.02267784	0.999716868	0.0288736	0.99136
413	tanh	0.2	Adam	0.01	RandomUniform	0.02267784	0.999766756	0.027259	0.990653
424	tanh	0.2	RMSprop	0.01	glorot_uniform	0.02267784	0.99975565	0.0299947	0.9909
454	tanh	0.5	RMSprop	0.01	glorot_uniform	0.02267795	0.9946343	0.0318673	0.9902

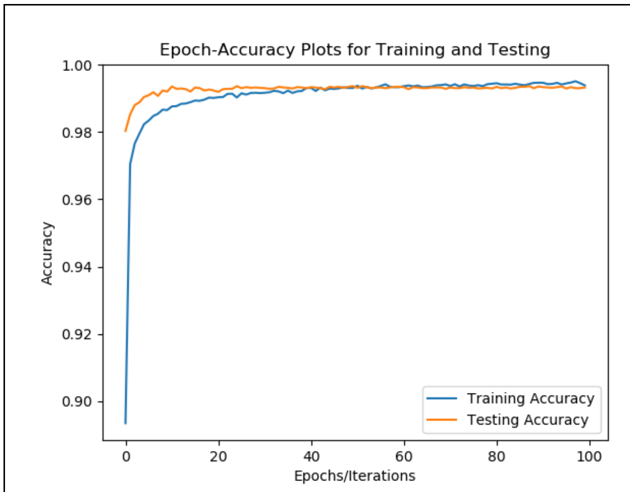
Table 10: Best accuracies on different parameter settings

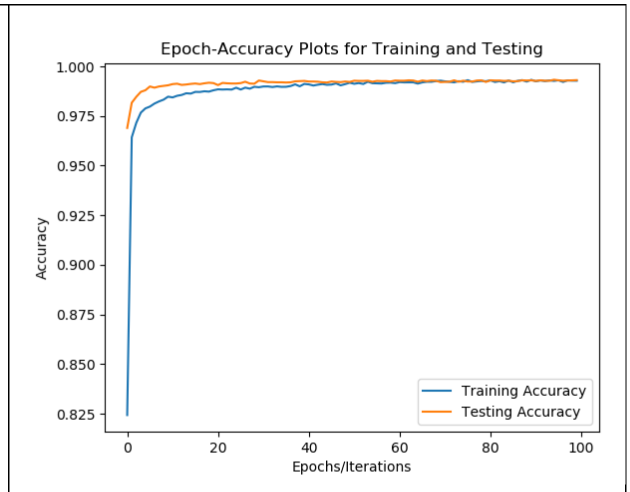
The plots for all the above-mentioned combinations are as shown below:

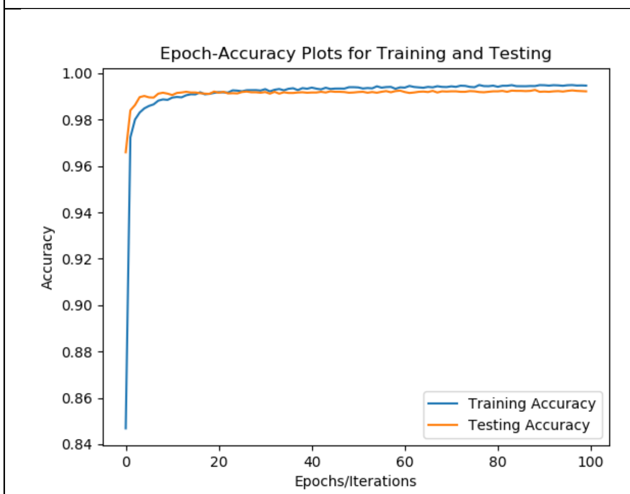


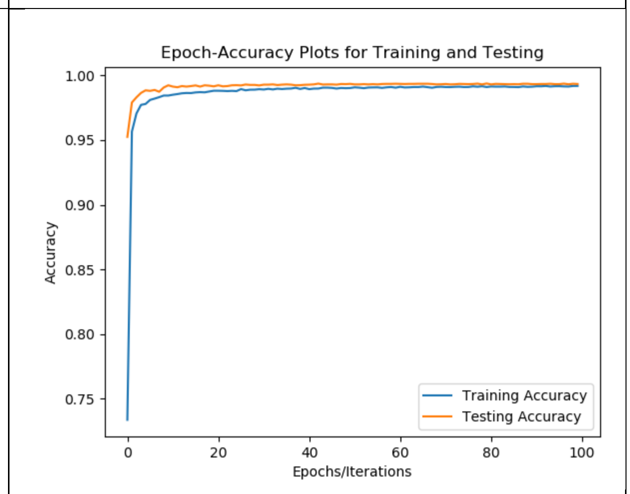


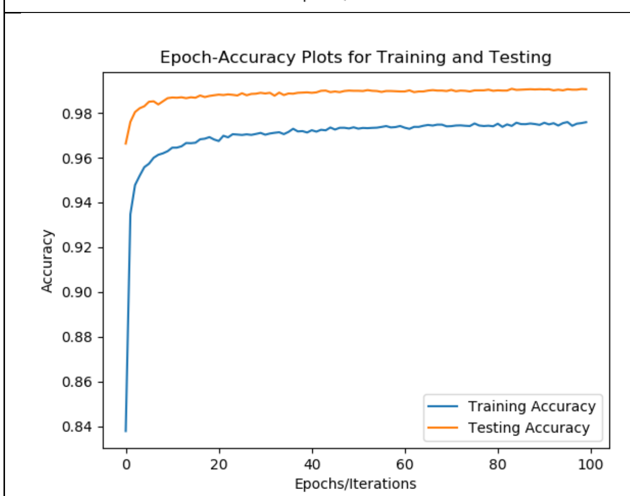

Epoch-Accuracy Plots for Training and Testing
Accuracy
Epochs/Iterations
Training Accuracy
Testing Accuracy
Epoch-Accuracy Plots for Training and Testing
Accuracy
Epochs/Iterations
Training Accuracy
Testing Accuracy
Epoch-Accuracy Plots for Training and Testing
Accuracy
Epochs/Iterations
Training Accuracy
Testing Accuracy
Epoch-Accuracy Plots for Training and Testing
Accuracy
Epochs/Iterations
Training Accuracy
Testing Accuracy
Epoch-Accuracy Plots for Training and Testing
Accuracy
Epochs/Iterations
Training Accuracy
Testing Accuracy
Epoch-Accuracy Plots for Training and Testing
Accuracy
Epochs/Iterations
Training Accuracy
Testing Accuracy

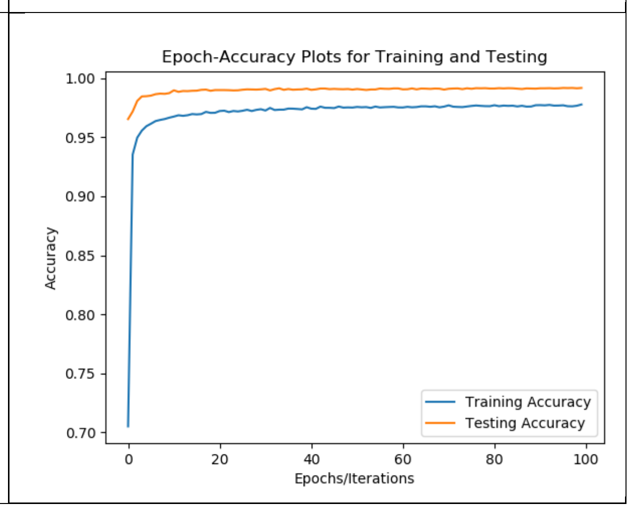

Epoch-Accuracy Plots for Training and Testing
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0
20
40
60
80
100
Epochs/Iterations
Training Accuracy
Testing Accuracy


Epoch-Accuracy Plots for Training and Testing
Accuracy
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0
20
40
60
80
100
Epochs/Iterations
Training Accuracy
Testing Accuracy


Epoch-Accuracy Plots for Training and Testing
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0
20
40
60
80
100
Epochs/Iterations
Training Accuracy
Testing Accuracy


Epoch-Accuracy Plots for Training and Testing
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0
20
40
60
80
100
Epochs/Iterations
Training Accuracy
Testing Accuracy


Epoch-Accuracy Plots for Training and Testing
Accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0
20
40
60
80
100
Epochs/Iterations
Training Accuracy
Testing Accuracy


Epoch-Accuracy Plots for Training and Testing
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
20
40
60
80
100
Epochs/Iterations
Training Accuracy
Testing Accuracy

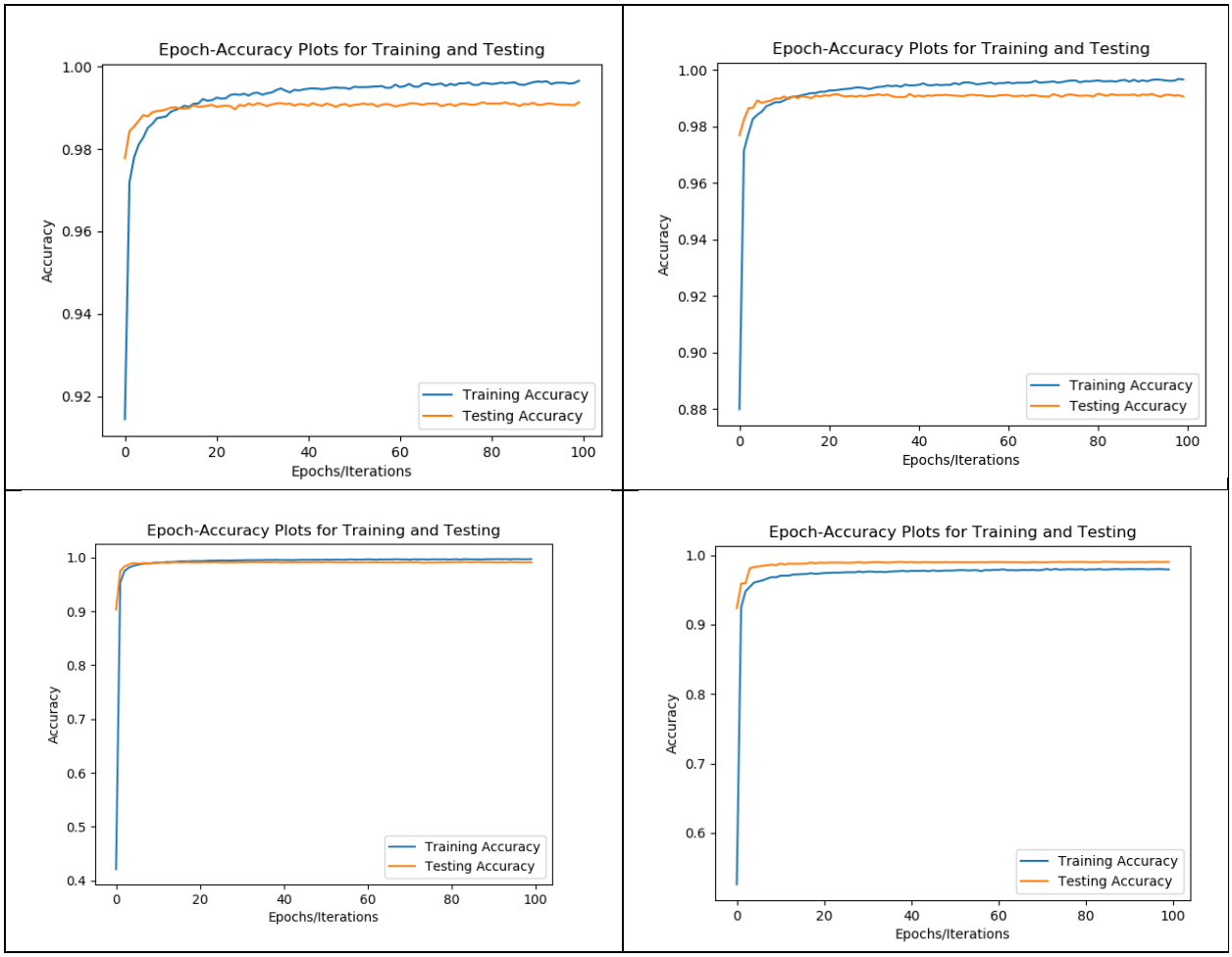


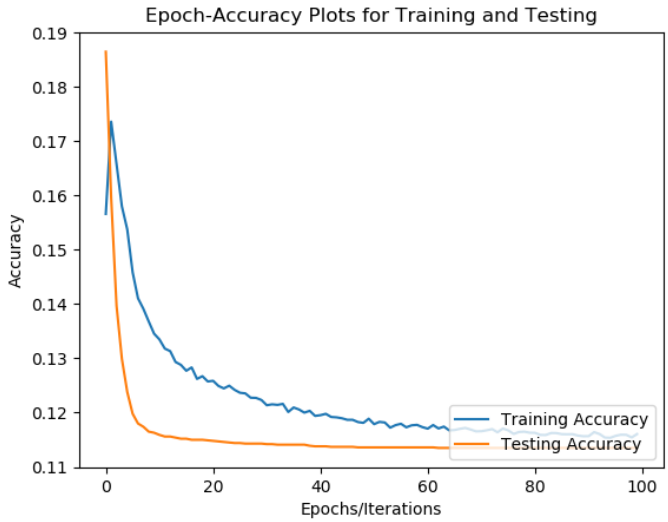
Figure 11: Plot of accuracy vs epoch for different parameter settings

IV. Discussion

As mentioned earlier, I changed the five parameters such as Activation functions, Dropouts, Optimizers, Learning Rate, Weight Initializations and kept the epochs and Batch size constant. I computed for 300 combinations of the above-mentioned features and the following observations were made:

- By changing the parameter, the plot of accuracy vs epochs changes considerably. When the parameters are designed to be perfect, the test accuracy score is high, approximately in the range of 90%. After observing the plots closely, it can be seen that for these good set of parameters the train and test accuracies increase rapidly i.e. in a few epochs and stays steady after reaching certain number of epochs. This indicates that the network designed will work best on the unknown testing data as it converges to the higher training accuracy very fast.

- After comparing with the following plot which gives a very bad accuracy, it is very evident that the initialization of parameters has to be proper.

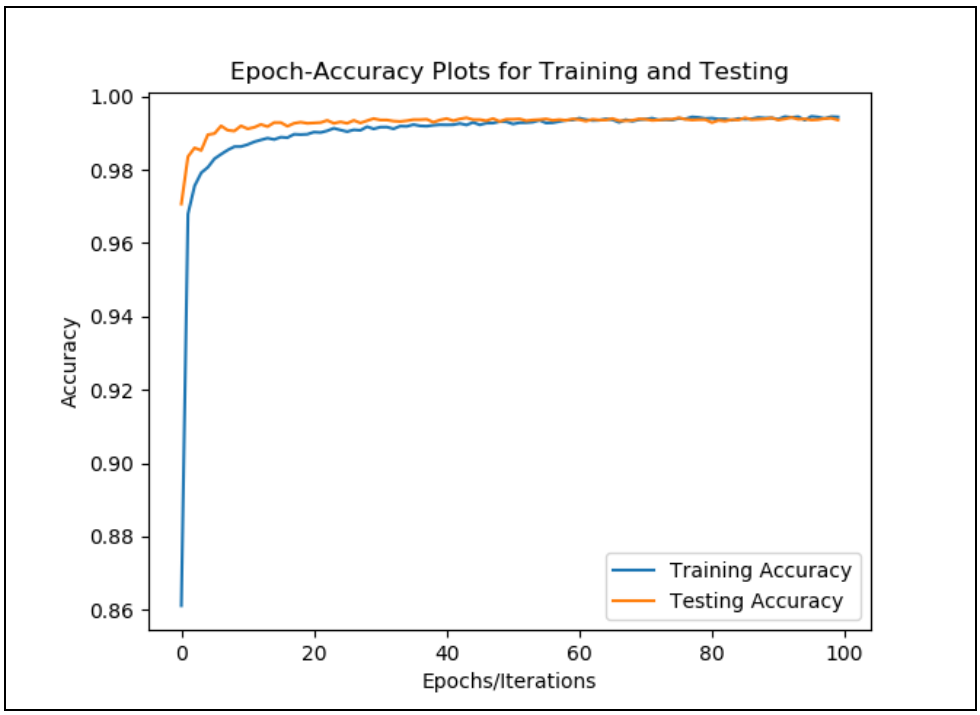


This plot id for the following initializations:
Epoch Number: 7
Activation function: Leaky ReLU
Dropout: 0.2
Optimizer: Adadelta
Learning rate: 0.01
Weight Initialization: Random Uniform
Test Accuracy: 11.35%
This Accuracy is not acceptable and hence the parameters should not be initialized to these values.

- From the above-mentioned table 10, the best test accuracy obtained is **99.36%** and the following parameter settings give this accuracy:

Epoch number: 13
Activation function: Leaky ReLU
Dropout: 0.2
Optimizer: Adam
Learning rate: 0.01
Weight Initialization: Random Uniform
Train loss: 0.00491711
Train Accuracy: 99.89%
Test Loss: 0.0197193
Test Accuracy: 99.36%

The plot of accuracy vs epoch for this high accuracy is:



As this is the best parameter setting, as compared to the other plots it can be observed that the training and testing accuracies reach a very high value in less than 10 epochs. Also, this value is maintained constant after reaching a certain number of epochs. We can also observe that the testing accuracy does not drop down at any point indicating that there is no overfitting while training. Also, the testing accuracy is greater than the training accuracy which is also a good sign. So, after going through all these observations I could conclude about the initialization of the parameters to give the best accuracy.

C) Improve the LeNet-5 for MINST dataset

I. Abstract and Motivation

I tried improving my accuracy score from the previous part by changing the architecture of LeNet 5 by adding more layers in the network. I also experimented with some optimizers and number of epochs. My observations are described in the subsequent sections.

II. Approach and Procedure

To improve the accuracy of my model from the previous section, I tried preprocessing techniques which helped in improving the score considerably. I also tried some data augmentation techniques which increased the score as well.

- Data preprocessing:

  This step is performed to first scale the images in the range 0,1 by dividing by 255. The pixel values are in the range 0-255 which are normalized to eliminate illumination effects. Also, the mean removal is also performed wherein the mean of the image is found out and subtracted from all the pixels in the image. This helps in preventing any high frequencies in the image to dominate the training data.

- Data Augmentation:

  Data augmentation is basically increasing the size of the dataset so that the accuracy is improved. This is needed for the MNIST dataset as while writing any digit, all the possible combinations of rotation, scaling, cropping, vertical and horizontal flips have to be considered and the model has to be trained to take all these combinations into consideration and give an output. The augmentation of data is done by the ImageDataGenerator function in Keras wherein the possible combinations can be given and then the model can be evaluated.

- Architecture change:

  Changing the architecture helps in improving the accuracy. If the number of filters and the receptive field of the filters is changed, the accuracy is affected greatly. As the number of filters is increased, the neural network is trained deeper and hence the accuracy increases at the cost of increase in computation time. Also, a combination of different activation functions and optimizers can be tried which is already done in part b.

III. Experimental Results

The different structure used in this part along with the parameters used and the accuracy obtained is listed below. The Accuracy vs Epoch plots are also plotted.

- Data Augmentation:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Data Augmentation	Yes	Yes	Yes	Yes	Yes

- Convolutional Layer 1:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	32	32	32	16	12

Number of filters	3x3	5x5	3x3	3x3	5x5
Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU
Weight Initialization	he_normal	he_normal	RandomUniform	RandomUniform	he_normal
Padding	same	same	same	same	same

- Convolutional Layer 2:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	32	32	32	16	N/A
Number of filters	3x3	5x5	3x3	3x3	N/A
Activation Function	ReLU	ReLU	ReLU	ReLU	N/A
Weight Initialization	he_normal	he_normal	RandomUniform	RandomUniform	N/A
Padding	same	same	same	same	N/A

- Convolutional Layer 3:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	N/A	N/A	N/A	16	N/A
Number of filters	N/A	N/A	N/A	3x3	N/A
Activation Function	N/A	N/A	N/A	ReLU	N/A
Weight Initialization	N/A	N/A	N/A	RandomUniform	N/A
Padding	N/A	N/A	N/A	same	N/A

- Max pooling Layer 1:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	2x2	2x2	2x2	2x2	2x2
Stride	2	2	2	2	2

- Convolutional Layer 4:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	64	64	64	32	25
Number of filters	3x3	3x3	3x3	3x3	5x5
Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU
Weight Initialization	he_normal	he_normal	RandomUniform	RandomUniform	he_normal
Padding	same	same	same	same	same

- Convolutional Layer 5:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	64	64	64	32	N/A
Number of filters	3x3	3x3	3x3	3x3	N/A
Activation Function	ReLU	ReLU	ReLU	ReLU	N/A
Weight Initialization	he_normal	he_normal	RandomUniform	RandomUniform	N/A
Padding	same	same	same	same	N/A

- Convolutional Layer 6:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	N/A	N/A	N/A	32	N/A
Number of filters	N/A	N/A	N/A	3x3	N/A
Activation Function	N/A	N/A	N/A	ReLU	N/A
Weight Initialization	N/A	N/A	N/A	RandomUniform	N/A
Padding	N/A	N/A	N/A	same	N/A

- Max pooling Layer 2:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	2x2	2x2	2x2	2x2	2x2
Stride	2	2	2	2	2

- Convolutional Layer 7:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	128	N/A	N/A	N/A	N/A
Number of filters	3x3	N/A	N/A	N/A	N/A
Activation Function	ReLU	N/A	N/A	N/A	N/A
Weight Initialization	he_normal	N/A	N/A	N/A	N/A
Padding	same	N/A	N/A	N/A	N/A

- Fully Connected Layer 1:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	128	256	512	512	180
Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU

- Dropout

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Dropout	0.25	0.5	0.2	0.25	0.5

- Fully Connected Layer 2:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	N/A	N/A	N/A	1024	100
Activation Function	N/A	N/A	N/A	ReLU	ReLU

- Dropout

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Dropout	N/A	N/A	N/A	0.5	0.5

- Fully Connected Layer 3:

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Size of filter	10	10	10	10	10
Activation Function	softmax	softmax	softmax	softmax	softmax

- Other parameters

Parameters	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Learning Rate	0.0001	0.0001	0.01	0.00001	0.01
Epoch	20	30	20	20	20
Batch size	64	86	128	16	16
Optimizer	RMSprop	RMSprop	Adam	Adam	Adam

- Accuracy

Parameter	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Accuracy	**99.69%**	**99.61%**	**99.47%**	**99.32%**	**98.68%**

The accuracy vs Epoch plots for each structure is as given below:

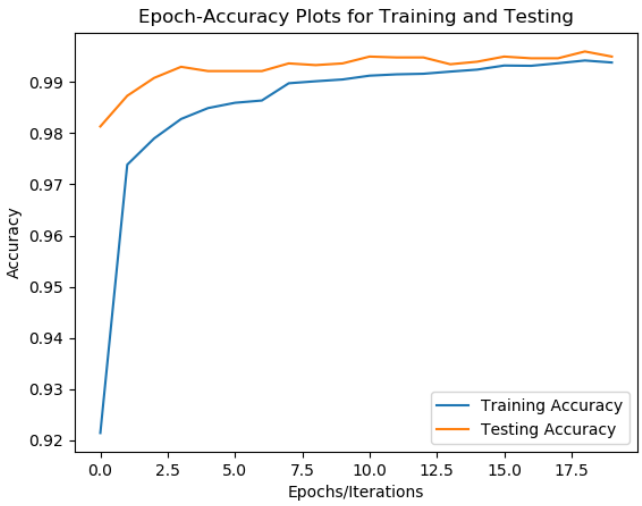


Figure 12: Accuracy vs Epoch plot for Structure 1

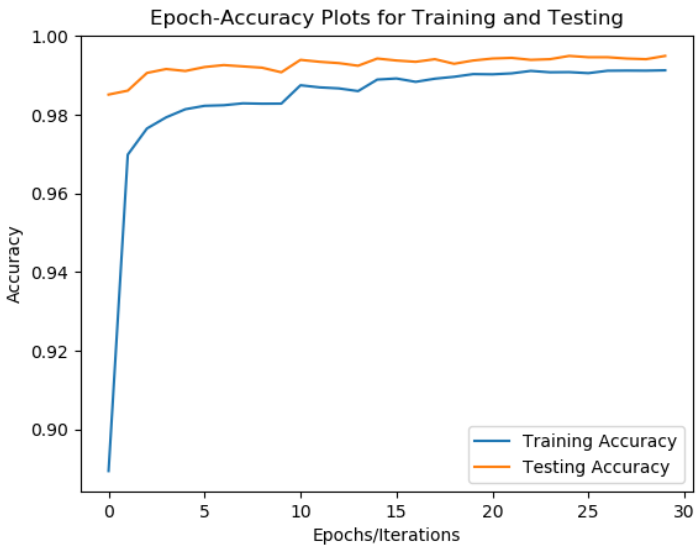


Figure 13: Accuracy vs Epoch plot for Structure 2

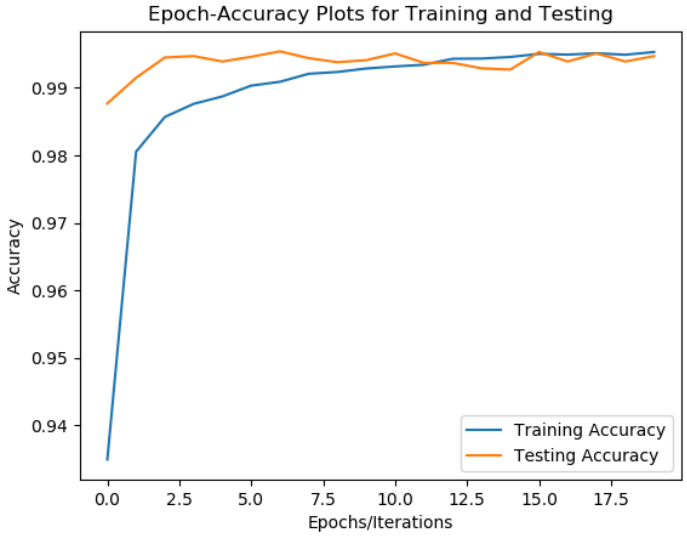


Figure 14: Accuracy vs Epoch plot for Structure 3

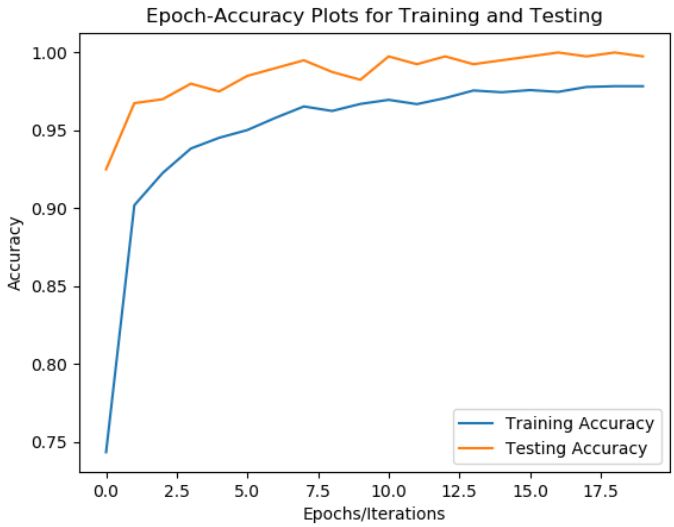


Figure 15: Accuracy vs Epoch plot for Structure 4

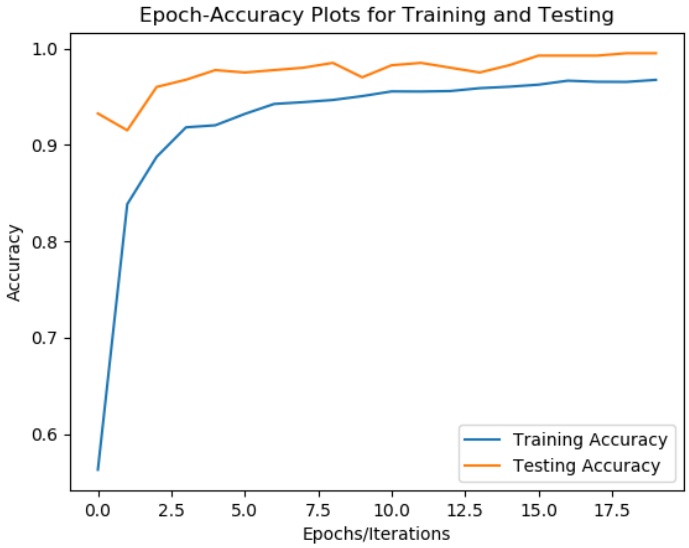


Figure 16: Accuracy vs Epoch plot for Structure 5

IV. Discussion

- As mentioned earlier, the accuracy of CNN model can be improved by performing some preprocessing techniques and performing data augmentation. Therefore, in all the structures these two operations are performed wherein mean is subtracted from each pixel and a function for image augmentation is used. Also, to further improve the accuracy different architectures are tried. I changed the number of layers and the number of filters in each layer to study the resultant accuracy. The accuracy for all the sets except structure 5 is high compared to part b due to the mentioned improvements.
- It can be observed that the accuracy vs epoch plot obtained is also better as compared to the plots in part b, as the testing accuracy in each of these plots is greater than the training accuracy and the testing accuracy increases in fewer epochs. That means that the convergence to maximum accuracy takes place faster.
- After observing the 5 structures, the maximum accuracy is obtained in structure 1 whose parameters can be summarized as follows:

  2 convolution layers of 32 filters of size 3x3, 1 max pooling layer of size 2x2, 2 convolution layers of 64 filters of size 3x3, 1 max pooling layer of size 2x2, 1 convolution layer of 128 filters of size 3x3, 2 fully connected layers of sizes 128 and 10 respectively. The activation function used is ReLU. The optimizer used is Adam and the accuracy obtained is **99.69%.**

Problem 2- Saak Transform and Its Application to the MNIST Dataset

A) Comparison between the Saak Transform and the CNN architecture

Saak transform has a lot of advantages over Convolutional Neural Networks and hence is widely used now-a-days in the fields of handwritten digit recognition, image processing, object detection and classification. The Saak architecture has some similarities with the CNN architecture in the sense that both of them perform some kind of transform i.e. convolution on the input image and then apply the ReLU activation function before passing it to the next layer. However, Subspace Approximation with Augmented Kernels(Saak) has the concept of subspace and kernel approximation above the CNN architecture to help in the applications of image analysis and image synthesis. The following are the notable differences in CNN and Saak transforms:

- Architecture of CNN and Saak:

CNN has an end-to-end architecture which has optimization functions associated with it so as to update the filter weights to get the obtained output close to the desired output. For this, CNN uses the concept of back propagation using Stochastic Gradient Descent in which the filter weights are updated from the loss function obtained at the output. However, in the case of Saak, it uses the concept of feature extraction and feature reduction so as to use handcrafted features and then uses traditional classification methods used in pattern recognition like SVM. In Saak, no back propagation is needed as the transform kernels are generated using the feed forward path unlike CNN.

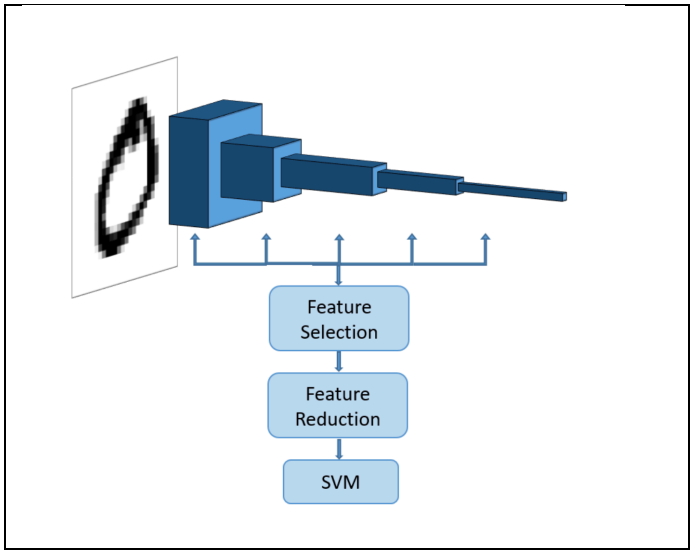


Figure 17: Architecture of Saak

- Scalability:

The filter weights obtained in CNN are a function of the data labels and the training data. Hence, when a small change is made in the class labels or the training data, the output of the CNN changes and hence training process has to be done again. This decreases the scalability of the CNN classifier. However, in the case of Saak transform, the output is not dependent on the class labels and the training data. Hence, a small change in them does not affect the accuracy that much and the training process need not be done again.

- Robustness:

CNN is not as robust as compared to the Saak transform. Saak transform is unaffected by noise and this can be checked by comparing the efficiency of CNN and Saak in presence of Salt and pepper noise. Hence, Saak is more robust than CNN in presence of noise.

- Usage of PCA in Saak:

Saak uses the concept of lossy components that is the feature selection process is done using PCA which makes it very hard to recover the input back from the output. But this also has an

advantage as change in the object classes or the data makes no effect on the output as the coefficients are anyways scaled down by using PCA. This is not the case in CNN as the change on object classes affects the final accuracy of CNN.

- Difference in layers in Saak and CNN:

CNN uses overlapping blocks in the convolution layer which is decided by the stride parameter and the pooling layer is used to reduce the spatial dimension of the input volume. However, in case of Saak non-overlapping blocks are used and no pooling layer

is required to reduce the spatial dimension. This reduction in spatial dimension is done by using PCA which reduces the Saak coefficients. The spatial dimension while using the Saak transform can be odd or even but the spatial dimensions in CNN is always odd of the size 3x3,5x5,7x7 etc.

- Mathematical Computations:

The main difference between the Saak transform and CNN architecture is that in Saak, the architecture is based on linear algebra. Hence, it is very easy to comprehend. But the architecture of Saak is very complex having a lot of mathematical computations and hence it is very difficult to monitor and understand.

- Computation Time:

As CNN uses the concept of back propagation and keeps updating its filter weights, the number of parameters to be hyper tuned increases considerably for CNN. On the other hand, Saak does not use back propagation to change its weights and the Saak coefficients are also reduced by using PCA. Hence, the computation time for Saak is low compared to CNN.

- Difference in Architecture:

CNN is a multilayer Rectified-Correlations on a Sphere (RECOS) which uses different layers such as convolution layer, fully connected layer, pooling layer. However, Saak uses the Karhunen-Lo´eve transform (KLT) to build the subspace approximation using the orthonormal basis which are later transformed using the ReLU activation function.

B) Application of Saak transform to MNIST dataset

I. Abstract and Motivation

Subspace Approximation with augmented kernels (Saak) uses the concepts of CNN with additional things such as Subspace approximation and augmented kernels. The main characteristic of Saak is that it uses forward and inverse transforms which are useful for image synthesis and analysis. The Kernels in multistage Saak transform are generated using feed-forward process. Hence, the data labels are not needed for computing the kernels.

The kernels derived in Saak are derived from the second order statistics of the input data vectors. Saak uses orthonormal transform kernels to compute the inverse and forward transforms. The process of calculating forward transform in Saak is comprised of the following steps:

- Building subspace approximation having orthonormal basis using Karhunen-Lo´eve transform (KLT). This step is performed by taking into consideration the largest Eigen values.
- Kernel Augmentation with non-positive values. This step takes uses the F-test to determine the features having strongest discriminant power.
- Application of ReLU (Rectified Linear Unit) activation function to the output.

The second and third steps combined are the signal to position format conversion of kernels. In multistage Saak transform, the input image is broken down into four quadrants such that the root is the original structure. And the leaf is the decomposed structure. This process continues till the root of the tree decomposes to 1x1 size. The coefficients in each stage of this decomposition are called the Saak coefficients. These Saak coefficients are the discriminating features of the input image.

The entire process of Saak transform is based on feature selection, feature reduction and training. The features are selected using the largest Eigen values, reduced using lossy compression techniques like PCA and given to classifiers such as SVM and Random Forest.

II. Approach and Procedure

I used the online code for Saak transform using tensorflow.[7]

The procedure followed to implement the Saak transform is as follows:

1) There are 5 stages to be used having values of components to be 3,4,7,6,8 respectively. The Saak coefficients are generated by using non-overlapping 2x2 blocks of the image such that the variance between these blocks is very low. Saak transform generates the anchor vectors and choses the vectors in maximum variance direction.
2) The best feature selection is done using f_test inbuilt function in sklearn library. The selection of components is done on the training set and the MNIST train labels. The number of feature dimensions is 1000 generated using a threshold.
3) Principal Component Analysis is used to transform the 1000 selected features to 32,64,128 dimensions.
4) The reduced features are then passed to the SVM and Random Forest Classifier to train the model. The training accuracy can be found out by passing the train features and train labels and the test accuracy can be found out by passing test features and test labels respectively.
5) As there are three reduction sizes of 32,64,128 and two classifiers to be used, there are in all 6 test cases which have to monitored.

III. Experimental Results

Validation set is kept constant as 5000 and the training set is 55000.

PCA components	Training SVM Accuracy	Testing SVM Accuracy	Training Random Forest Accuracy	Testing Random Forest Accuracy
32	99.236%	**98.31%**	**99.870%**	93.04%
64	98.889%	98.30%	99.892%	92.31%
128	98.092%	97.70%	99.898%	89.90%

Table 18: Saak transform outputs for SVM and Random forest classifier.

IV. Discussion

- As seen from the above table, the best accuracies for all the 6 cases has been shown. The best training accuracy obtained is for 32 as the principal component and using SVM as the classifier. The accuracy obtained for this combination is 99.87%.
- The best testing accuracy obtained is 98.31% using 32 as the principal components and SVM as the classifier.
- The best testing accuracy obtained in part 1 for CNN is 99.69% whereas the best testing accuracy obtained in Saak is 98.31%.
- If we compare the accuracy of the Saak transform and CNN, it can be observed that the accuracy of Saak is almost at par with CNN. But the best thing about Saak is that it uses less amount of training data to predict unknown labels, unlike CNN. Also, the classification of Saak is invariant to change of data which is not the case in CNN. Hence, the scalability and robustness of Saak is definitely better than CNN that too with comparable accuracy.
- Also, as Saak does not need back propagation and labels for training and trains with only fewer data and coefficients, it is comparatively faster than CNN and hence computationally effective.

C) Error Analysis

To do the error analysis between Saak and CNN generated outputs, I compared the predicted labels for both CNN and Saak to the expected labels. By doing so, I could deduce how many testing data points were getting misclassified by Saak or CNN or both. The total number of testing samples available was 10,000.
The following is the statistics obtained from studying the errors.

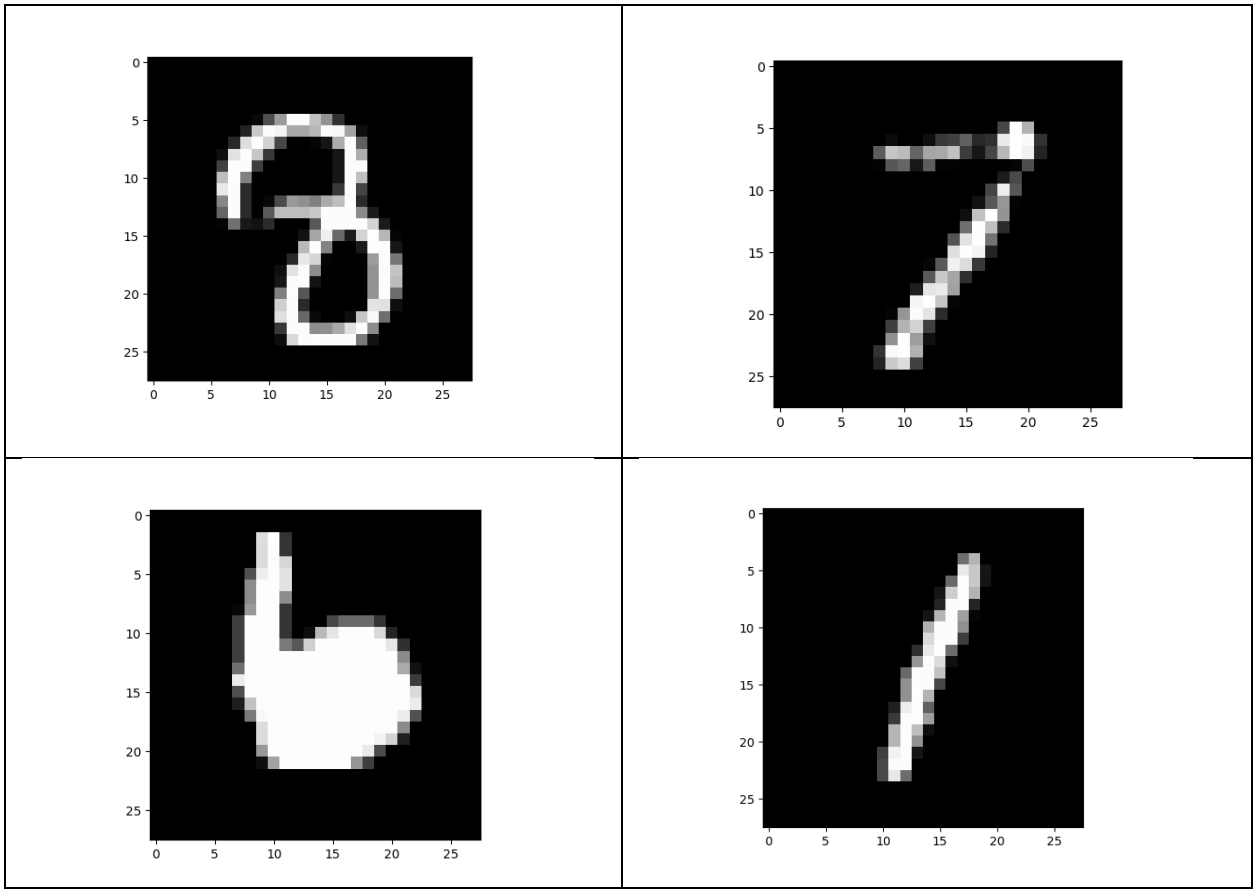
Count samples missed by Saak	169
Count of samples that Saak misclassified but CNN classified correctly	151
Percentage of samples that Saak misclassified but CNN predicted correctly	89.2856%

Count samples missed by CNN	39
Count of samples that CNN misclassified but Saak classified correctly	21
Percentage of samples that CNN misclassified but Saak predicted correctly	52.631%

Percentage of testing dataset that was misclassified by both CNN and Saak	19%
Accuracy of Saak	98.3256%
Accuracy of CNN	99.6251%

As seen from the above table,
The percentage of testing data that was predicted properly by CNN and Saak is 82% which is a very good result. Only 19% of the data is misclassified by both CNN and Saak. Therefore, the same percentage of error in misclassification of Saak and CNN is 19%.
The misclassifications in Saak are more than that in CNN which is visible in the data provided above. Therefore, there is a tradeoff between the number of misclassified data points and the time taken for running as in the case of Saak the computation time is low, but the accuracy is a bit less than CNN.
Also, the accuracy for CNN is 99.6251% whereas for Saak it is 98.3256% which is quite impressive. Other methods can be used to increase this accuracy which are described below.

From the previous part, I could find the samples misclassified by both CNN and Saak. I observed these samples closely to come up with methods to improve the performance of CNN and Saak i.e. to prevent misclassification by both these methods.
The testing samples misclassified by both CNN and Saak are as follows:



After observing these misclassified samples closely, it can observed that due to noisy contents in the image, the classifier was not able to predict the true labels. Hence, better pre-processing techniques can be employed before passing the images to the classifier so that the classifier predicts the score better.
Also, lack of training data might be one of the problems while training. If we increase more training data, the classifier will understand all the possible variants in the image and will predict the test data in a better way. By increasing the data, the accuracy will definitely increase. While increasing that data, we can make sure that we consider rotations, scaling and other such transformations in the image. If we see the digit one in the image shown above, according to me the reason that Saak and CNN methods could not classify it properly was because it is rotated with respect to the ideal image of digit 1. Hence, sampling in such a way will definitely help. Also, additional color manipulations can be done on the training images while increase the database by performing image enhancement, brightness contrast etc. This will definitely create an impact on the accuracy.

As we have mentioned earlier, Saak transform works on limited data i.e. less amount data is needed for training. But this might also be the reason why Saak might misclassify more samples which is quite evident in the statistics mentioned above. Hence, including more data and training on more data might prove beneficial for Saak. Also, in case of Saak transform used currently, F-test is used to select the best discriminating features among the available features.

Instead, other methods can be used which will pick better features that can be used for training like Dunn's test, Bonferroni's test, Scheffe's test.

An ensemble of CNN and Saak can be used which can increase the accuracy. By creating such an ensemble, the pros of both the methods can be incorporated.

References

[1] LeNet-5 architecture- http://yann.lecun.com/exdb/publis/pdf/ lecun-01a.pdf

[2] MNIST dataset- http://yann.lecun.com/exdb/mnist/h

[3] Stanford Lecture on Convolutional Neural Networks- http://cs231n.github.io/

[4] Saak Transform https://arxiv.org/abs/1710.04176

[5] Lecture and Discussion slides

[6] www.wikipedia.com

[7] https://github.com/morningsyj/Saak-tensorflow