EE 569 Homework #4 Report

Akash Mukesh Joshi
USC ID : 4703642421

Submission Date : 04^{h} December 2016

Homework #4

Problem 1: LeNet-5 Training and Its Application to the MNIST Dataset

(a) CNN Architecture and Training

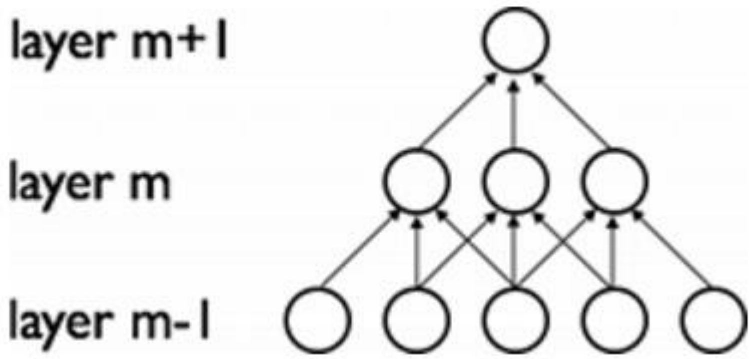
1. Describe the architecture and building components of the LeNet-5 in your own words.

To understand the concept of LeNet-5 neural network we need start from the basis. Firstly, we need to understand the concept of an artificial neural network (ANN). Let’s consider a space containing N neurons which are interconnected in such a way that a branch from one node traveling to another node is considered as an output from the branch leaving a particular node and the is given as a input to the other node.

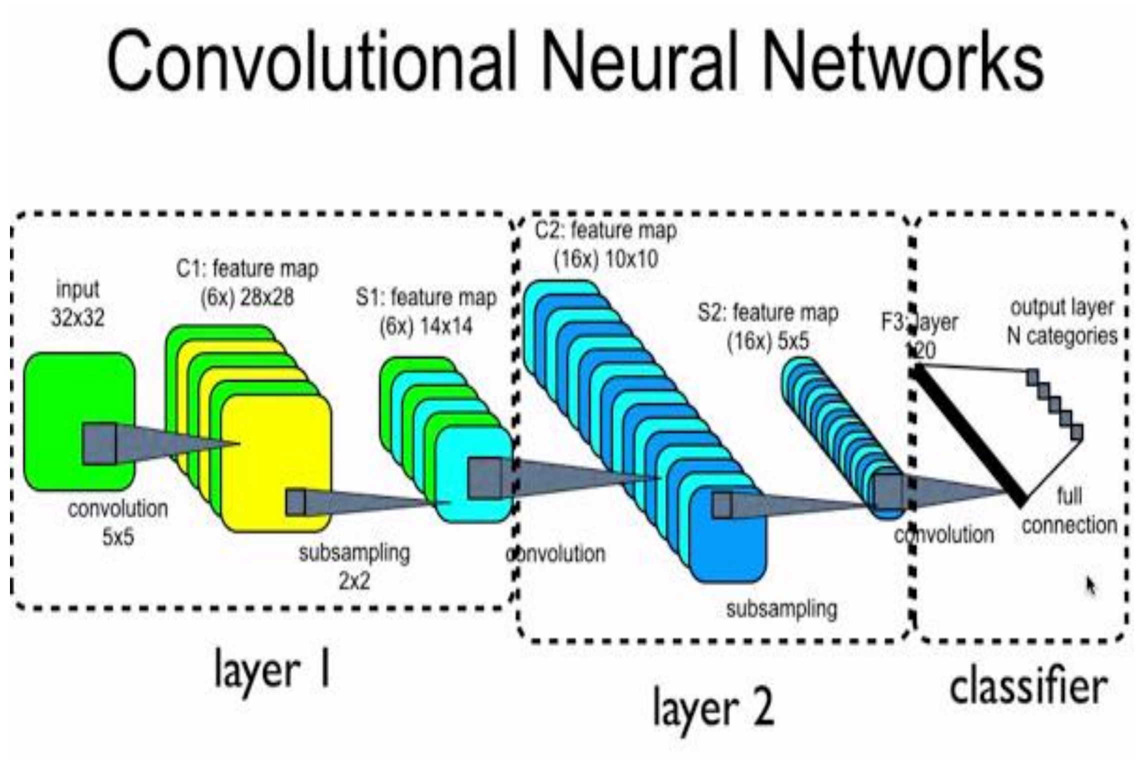
Now lets consider a such layers of such nodes spread in a 3 dimensional space. Here each layer has different nodes present in them the layers itself are connected to each other by branches. This kind of an artificial neural network is called a multilayer perceptron neural network (MLPs). MPLs are basically feed-forward ANN that maps a set of input data to an output dataset. MPLs work on a supervised learning classification technique called back-propagation for training of the neural network.

Now let’s talk about how a convolution neural network (CNN) works. CNN is a biological inspired variants of a multilayer perceptron neural network. Now we know that the visual cortex contains a complex arrangement of cells. The animal visual cortex is the most advanced form of image processing system in the animal kingdom. The complex arrangement of cells act like a neural network. This complex arrangement of cells/nodes of the neural network for a convolution neural network. Hence we are interested in analyzing the most powerful neural network available out there.

The convolution neural network works on local connectivity of nodes/neurons of adjacent layers. It means that the inputs to a node in a particular layer are provided from a node/neuron just adjacent to it as shown in the image below.



From the above image we can see that a node in a particular neural net layer has multiple connectivity from more than one nodes/neurons from its adjacent. Hence, we can image that all such neurons in a particular layer have a connectivity with other neurons from its adjacent layer and form a complex neural network called the convolution neural network. Additionally a CNN also have replicative filtering across the entire visual field. These replicated i allow the features of the image captured to be detected regardless of their position in the visual field. This concept is discussed in the problems solved below. This replication of filters increases the weights of the filtering system and hence improves the performance of the neural network. Bellow we can see a convolution layer which shows filter replication and improvement in weights.



2. What is the main difference between the LeNet-5 and the classic three-layer artificial neural network (ANN)?

The main difference between a ANN and a LeNet-5 network is that a LeNet-5 network works on convolution based neural network. In an artificial neural network we have neurons connected to each other within the same network layer and the there is only a one is to one connection between layers. Whereas, in a LeNet-5 network we have neurons connected to multiple neurons between layers hence creating a complex network between layers which is absent in a artificial neural network. A LeNet-5 networks works on back-propagation and have replicated filters which is applied on a patch of a image which is not the case for a ANN.

3. Explain why LeNet-5 works better for the handwritten digit classification problem as compared with ANN?

The handwritten digit classification problem is classification of images that have to be analyzed using a LeNet-5 and not a ANN because the LeNet-5 is based on a CNN which is a much more complex network. The CNN is based on a visual cortex which will give us high accuracy on the handwritten dataset as compared to ANN.

(b) Application of LeNet-5 to MNIST Dataset

I. Abstract and Motivation

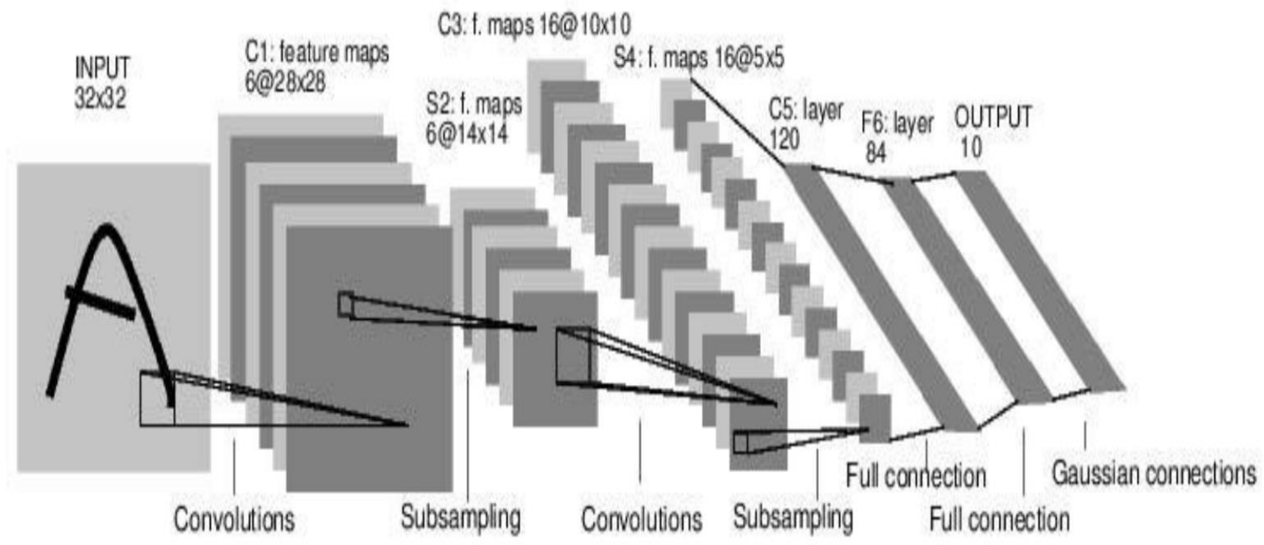
The MNIST database is a database of 60000 training and 10000 testing samples of handwritten digits from 0 to 9 and are used for classification. The dataset of handwritten digits is image dataset that can be easily classified by any human. The main purpose of this assignment is to design a classification system which is able to do the same thing. Hence we will be using LeNet-5 neural network to perform classification on the dataset due to the reasons mentioned above. The MNIST dataset will be run on a neural network proposed by Dr. Yann LeCun who was interdependently responsible for forming the back-propagation algorit.hm. The MNIST dataset was also developed by him.

One of the most important applications of performing such a classification is called object code recognition (OCR). OCR has immense applications in many fields like converting handwritten notes into a word or a printable document file (.pdf). OCR is also used in postal service applications where image processing and computer vision work hand in hand to determine the address of a package and this helps in packaging. One of the latest applications we see this being used is in Amazon Packaging Facilities for packaging of items being ordered and sorting them according to delivery postal location.

II. Approach and Procedures

The main aim is to set up a convolution neural network and run the network over the MNIST training dataset of 60000 samples and then testing it over the 10000 test dataset samples.

- The first step is to include all the packages that are needed for running the neural network. This is done using the ***require*** function to include packages in the .lua code file.
- A main function is defined where the neural network, training of the dataset and also the testing is performed.
- We first seed the random number generator such that it becomes easy for debugging such that it doesn’t randomized any parameters during initialization
- Now is load the train and test dataset provided and store it in two torch tensor data variable called as ***dataset_train*** and ***dataset_test***.
- The next step is to divide this dataset into data and label and also converting the data into double data-type using the ***:double()***.
- The next step is to normalize the dataset with its mean and standard deviation. The training and the testing dataset are both normalized with their respective means and standard deviations.
- Now we create a network and the criterion function for the back-propagation. A local ***network*** is initialized using a sequential neural network using ***nn.Sequential()*** and a local ***criterion*** using the class negative log likelihood criterion using n***n.ClassNLLCriterion().***
- Now we start defining the LeNet-5 neural network using the ***network*** that was initialized in the above step based on the image shown below.

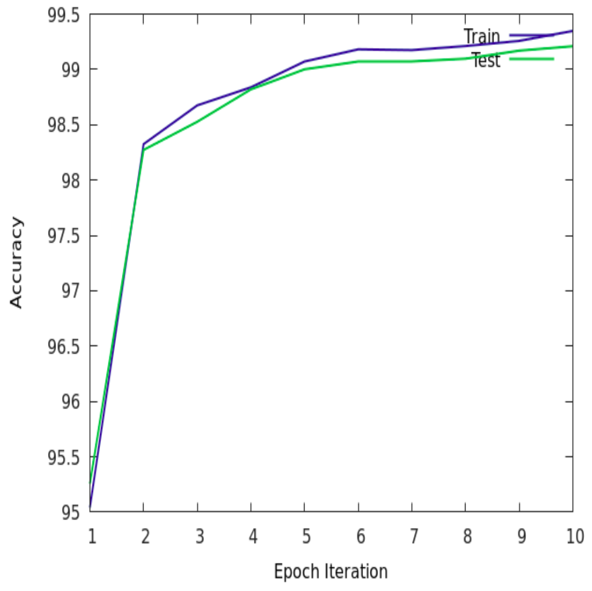


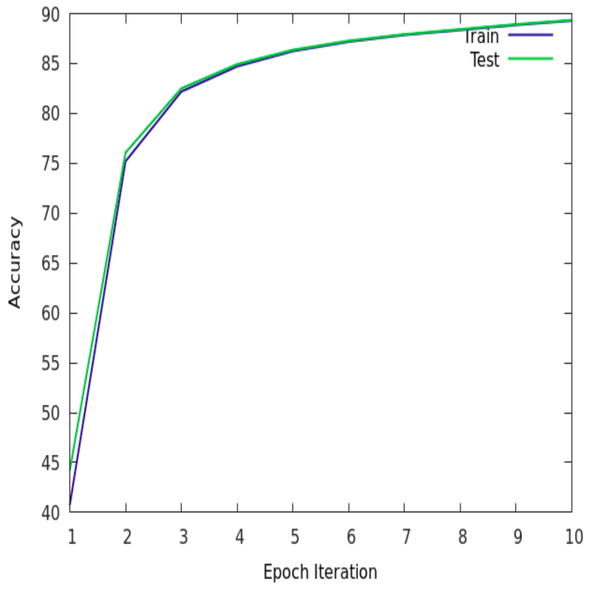
- From the above image we can see that the first step to perform is convolution for a 1 channel image dataset with 6 outputs channels using a 5 x 5 convolution kernel. We achieve this using ***network:add(nn.SpatialConvolution(1, 6, 5, 5))***
- Now we need to sample a small part of the layer i.e. a few neurons/nodes to node in the next layer. To perform this step we need to use ***network:add(nn.ReLU())*** which performs a non-linear connection between the above layer and connect it to the next layer.
- The step is not to perform a maximum pooling operation that looks at a 2 x 2 window on an image patch and find the maximum out of it and then a spatial convolution is performed. This is done using ***network:add(nn.SpatialMaxPooling(2,2,2,2))*** for maximum pooling and ***network:add(nn.SpatialConvolution(6, 16, 5, 5))*** for spatial pooling.
- After performing convolution we now again perform subsampling and maximum pooling using a 2 x 2 window on an image patch and find the maximum.
- Now that spatial convolution is performed twice we now need to convert the 3-d tensor of 16 x 5 x 5 into a 1-d tensor of 16*5*5. This is performed using ***network:add(nn.View(16*5*5))*** and this is done by connecting the entire layer to the next layer node thus forming a full connection between the two layers using ***network:add(nn.Linear(16*5*5, 120))*** function.
- Now moving on to the last layer we again perform a full connection but now we connect the 120 values to 84 values in that layer using ***network:add(nn.Linear(120, 84))*** and then this layer is converted into 10 number of outputs of the network for the 10 classes that we need for the dataset to be classified into.
- Once the network is formed and we have 10 output layer we apply a log-probability on the output of the dataset classification for getting probability values of each image being classified into to a particular class. This is done using the ***network:add(nn.LogSoftMax())*** function to get a log-probability value of the output of the dataset.
- Now that the network is initialized we need to extract the network parameters and arrange them linearly in memory so that we have a large vector containing all the information and parameter values of the neural network. This is performed using p***arameters,gradParameters = network:getParameters()*** function.
- The next step that is taken to initialize 4 arrays with zeros in them to calculate the loss and accuracy on the training set and testing dataset once the network has trained on the training dataset.

- To achieve this we run several epochs on small batches of the training set. An epoch in a neural network is an iterative training of the neural network as an epoch is a single pass through the entire training dataset which is usually followed by testing the network on the test dataset. For this algorithm we have found that running the epoch 10 times stabilizes the network accuracy,
- Now we create a batch size of 10 samples to train the network with. This is done to smoothen the gradients of the output of the network.
- Now we define a set of local classes as an array of characters ranging from 0 to 9 and also define the confusion matrix using ***optim.ConfusionMatrix(classes)*** where classes is defined as ***classes = {'0','1','2','3','4','5','6','7','8','9'}.***
- The training parameters need to be set now and the first thing we do is set the learning rate of the neural network to a value of 0.06. This is a pre-determined value and hasn't been changed as it does not give on the file reported code.
- We can seen that increasing the batch size increases the accuracy of the neural network but also increases the complexity of network and makes the algorithm run much slower. We also noted that decreasing the learning rate improves the accuracy of the network and we the network was tested with learning rates of 0.06 to 0.01 with 0.01 giving a really good accuracy.
- Now we define two variable ***input*** and ***target*** which take small batches of the training dataset with its labels and perform training on that small batch.
- The training on this small batch of dataset is done till all the training samples have been used and this is done on function called ***feval***.
- The first step is to reset the gradients parameters to zero and then start the training on the batch dataset.
- We now forward the data to the network using the function ***network:forward(input)*** and this function returns a 10 class classification output with is then compared with the target labels initialized in the above steps to calculate the average loss and accuracy on the dataset.
- We now send the feval function, the network parameters and the configuration parameters to different cost functions for training the neural network using ***optim.sgd(feval, parameters, config).*** This done using sgd, asgd, adadelta and lbfsg. It was found that sgd gave the best accuracy on the neural network.
- Every iteration of the neural network we test the dataset to check the accuracy of the network on the test dataset. This is done on the entire test dataset instead of a batch test. To do testing on the dataset we forward the test dataset to the network and then send the output with the test labels to the criterion function to calculate the testing accuracy on the dataset.
- Next we clean the data to reduce the size of the network file and save the network as an output file.
- Now we need to plot the accuracy of the training and testing dataset on a single graph. We use a function called gnuplot to print accuracy every epoch iteration. The last thing to do is to calculate accuracy that we get in the last epoch is the mean average precision on the testing dataset and is given as an output on the terminal.

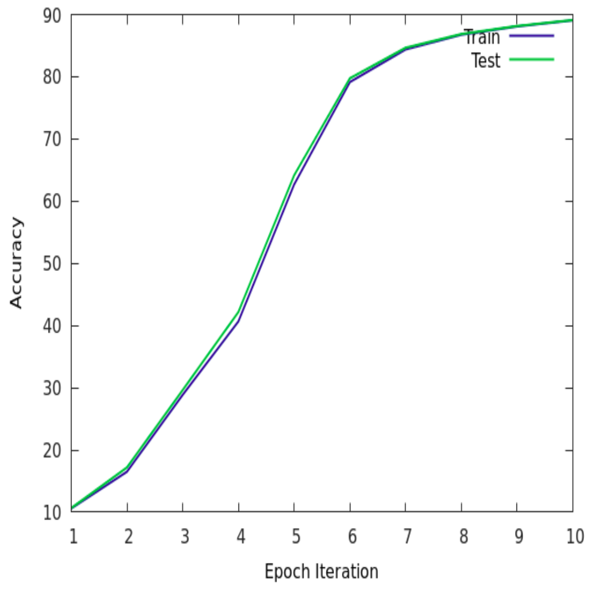
III. Results

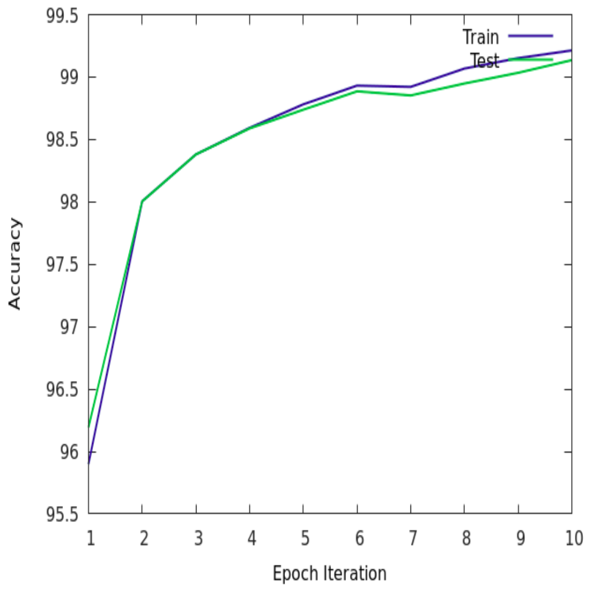
Applying the above algorithm yields the following outputs on different cost functions as shown in the images below.





SGD on the Training and Testing Dataset ***LBFSG on the Training and Testing Dataset***





ASGD on the Training and Testing Dataset ***ADADELTA on the Training and Testing Dataset***

IV. Discussion

Looking at the output graphs we can say that we have achieved greater than 90 % accuracy using all different network settings. We have seen that increasing the batch size and reducing the learning rate value we get better results. We have set a learning rate of 0.06 and batch size of 10 for the final code. From the above results we can see that using Stochastic gradient descent (SGD) we have achieved maximum accuracy. The LBFSG method gives the most closest accuracy with comparison to the train dataset.

Problem 2: Capability and Limitation of Convolution Neural Networks

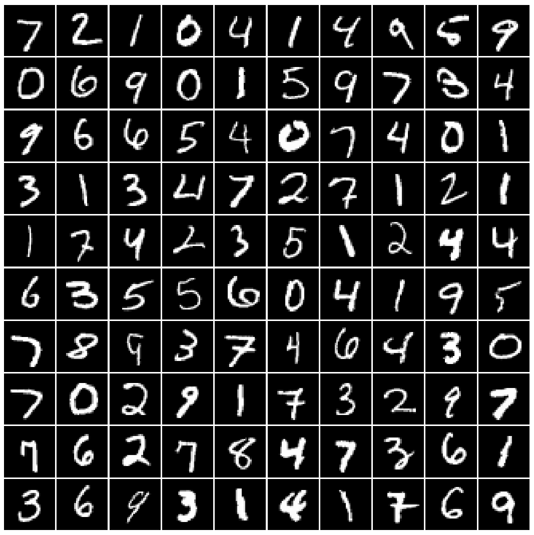
(a) Application of LeNet-5 to Negative MNIST Images

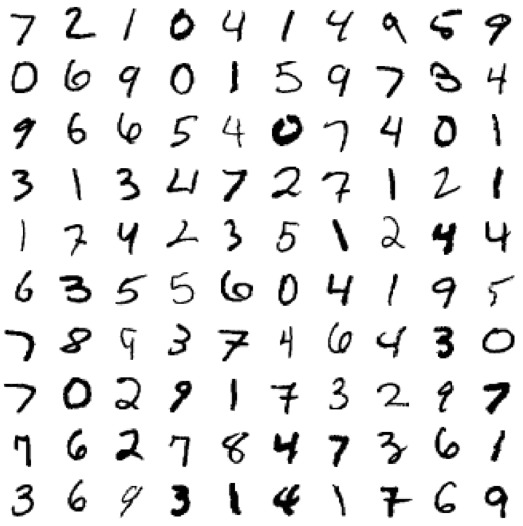
I. Abstract and Motivation

The LeNet-5 neural network works with positive bacground iamges and now we need to check if the neural network will work with negative of the image dataset. The main aim of this method is to check if an image dataset of a different color space. In humans digits handwritten in different colors also can be recognized by the visual cortex and we want to check if the neural network developed in the above problem is able to recognize the digits or not. To perform this kind of test we need to train the neural network on the positive training dataset and test the neural network on the negative testing dataset. We can see that this does not result in a good classification and hence we make need to train a new network which works on both positive and negative images from the dataset. This procedure is explained in detail below.

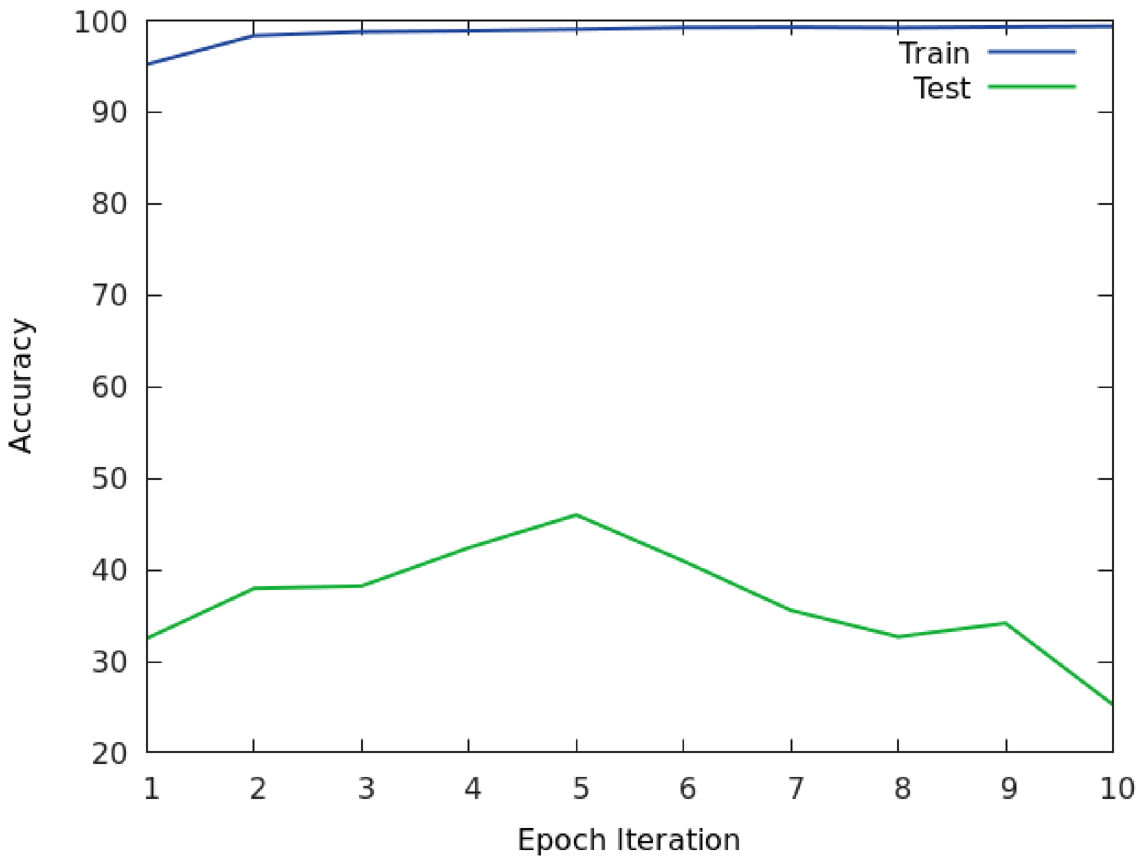
II. Approach and Procedures

- The first that we need to do is make the test data negative. To do this we need edit the first problem itself and test the dataset onto the neural network. This done by subtracting test dataset from 255 to invert the image as shown. ***testset.data = 255-dataset_test.data:double()*** and then normalize the data set as done in the previous method. The negative dataset in shown below in comparison to the original dataset.

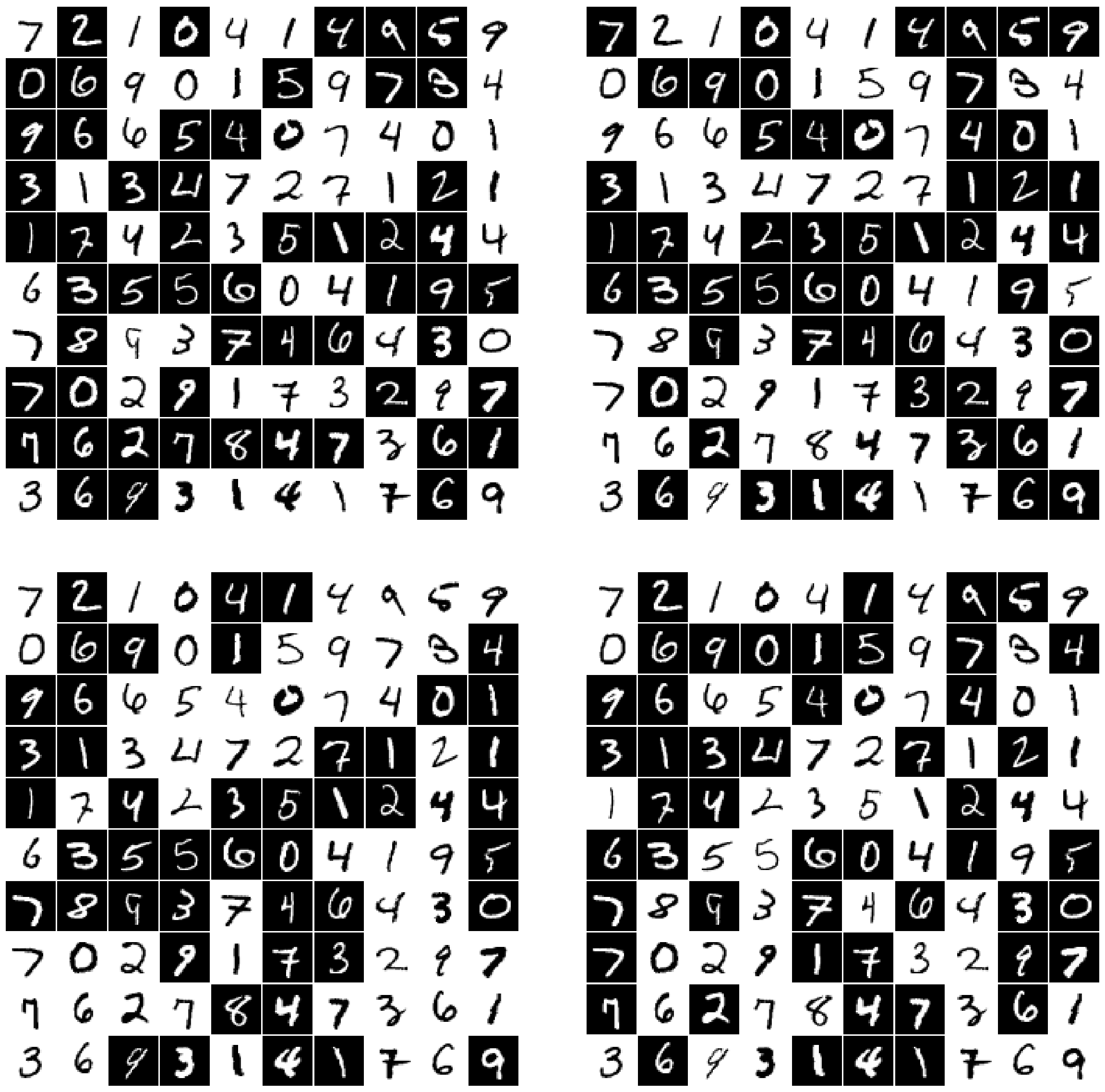




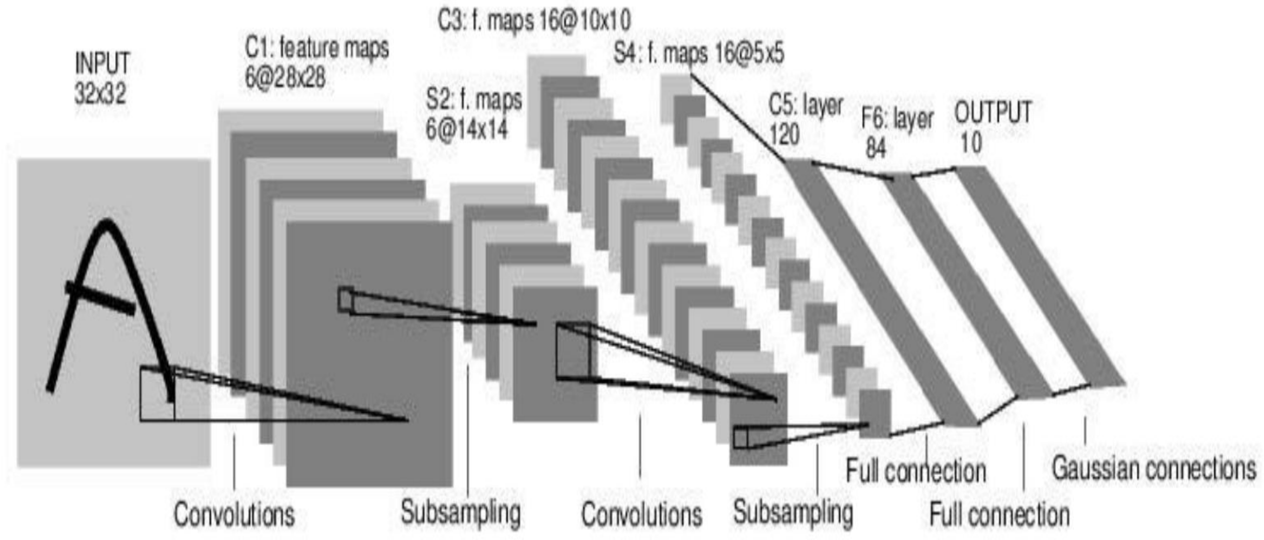
- The above image is a small portion of the test dataset and we can see that the test dataset is inverted and then we test dataset on the training set. The output of the neural network in shown below:

- 

  ies that the neural network is not good for the negative test dataset. The mAP in the above testing was around 26 %. Hence we need train a new neural network which works on both the positive and negative dataset.
- The first step is to include all the packages that are needed for running the neural network. This is done using the ***require*** function to include packages in the .lua code file.
- A main function is defined where the neural network, training of the dataset and also the testing is performed.
- We first seed the random number generator such that it becomes easy for debugging such that it doesn't randomized any parameters during initialization
- Now is load the train and test dataset provided and store it in two torch tensor data variable called as ***dataset_train*** and ***dataset_test***.
- The next step is to divide this dataset into data and label and also converting the data into double data-type using the ***:double()***.
- Now we save the train dataset into two variables ***image_p*** and ***image_n*** as the positive and negative images respectively. These two variables are merged using concatenation using the following function: ***trainset.data = image_p:cat(image_n,1):double().*** The same thing is done for the labels of the training dataset.
- The next step is to randomly convert images in the test dataset into negative images using a ***torch.random*** function and then randomly assigning images in the test dataset its negative image values as shown below:



- The next step is to normalize the dataset with its mean and standard deviation. The training and the testing dataset are both normalized with their respective means and standard deviations.
- Now we create a network and the criterion function for the back-propagation. A local ***network*** is initialized using a sequential neural network using ***nn.Sequential()*** and a local ***criterion*** using the class negative log likelihood criterion using n***n.ClassNLLCriterion().***
- Now we start defining the LeNet-5 neural network using the ***network*** that was initialized in the above step based on the image shown below.

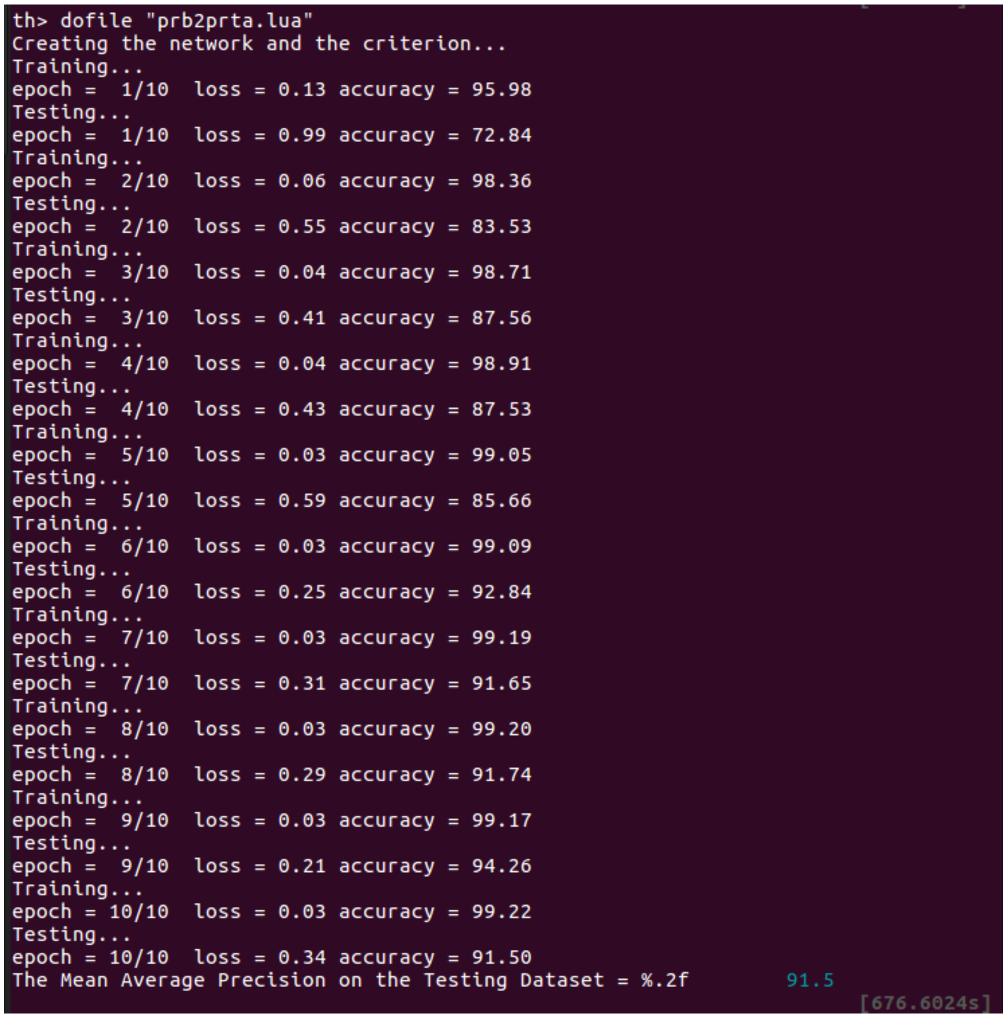


- From the above image we can see that the first step to perform is convolution for a 1 channel image dataset with 6 outputs channels using a 5 x 5 convolution kernel. We achieve this using ***network:add(nn.SpatialConvolution(1, 6, 5, 5))***
- Now we need to sample a small part of the layer i.e. a few neurons/nodes to node in the next layer. To perform this step we need to use ***network:add(nn.ReLU())*** which performs a non-linear connection between the above layer and connect it to the next layer.
- The step is not to perform a maximum pooling operation that looks at a 2 x 2 window on an image patch and find the maximum out of it and then a spatial convolution is performed. This is done using ***network:add(nn.SpatialMaxPooling(2,2,2,2))*** for maximum pooling and ***network:add(nn.SpatialConvolution(6, 16, 5, 5))*** for spatial pooling.
- After performing convolution we now again perform subsampling and maximum pooling using a 2 x 2 window on an image patch and find the maximum.
- Now that spatial convolution is performed twice we now need to convert the 3-d tensor of 16 x 5 x 5 into a 1-d tensor of 16*5*5. This is performed using ***network:add(nn.View(16*5*5))*** and this is done by connecting the entire layer to the next layer node thus forming a full connection between the two layers using ***network:add(nn.Linear(16*5*5, 120))*** function.
- Now moving on to the last layer we again perform a full connection but now we connect the 120 values to 84 values in that layer using ***network:add(nn.Linear(120, 84))*** and then this layer is converted into 10 number of outputs of the network for the 10 classes that we need for the dataset to be classified into.
- Once the network is formed and we have 10 output layer we apply a log-probability on the output of the dataset classification for getting probability values of each image being classified into to a particular class. This is done using the ***network:add(nn.LogSoftMax())*** function to get a log-probability value of the output of the dataset.
- Now that the network is initialized we need to extract the network parameters and arrange them linearly in memory so that we have a large vector containing all the information and parameter values of the neural network. This is performed using p***arameters,gradParameters = network:getParameters()*** function.
- The next step that is taken to initialize 4 arrays with zeros in them to calculate the loss and accuracy on the training set and testing dataset once the network has trained on the training dataset.

- To achieve this we run several epochs on small batches of the training set. An epoch in a neural network is an iterative training of the neural network as an epoch is a single pass through the entire training dataset which is usually followed by testing the network on the test dataset. For this algorithm we have found that running the epoch 10 times stabilizes the network accuracy,
- Now we create a batch size of 10 samples to train the network with. This is done to smoothen the gradients of the output of the network.
- Now we define a set of local classes as an array of characters ranging from 0 to 9 and also define the confusion matrix using ***optim.ConfusionMatrix(classes)*** where classes is defined as ***classes = {'0','1','2','3','4','5','6','7','8','9'}.***
- The training parameters need to be set now and the first thing we do is set the learning rate of the neural network to a value of 0.06. This is a pre-determined value and hasn't been changed as it does not give on the file reported code.
- We can seen that increasing the batch size increases the accuracy of the neural network but also increases the complexity of network and makes the algorithm run much slower. We also noted that decreasing the learning rate improves the accuracy of the network and we the network was tested with learning rates of 0.06 to 0.01 with 0.01 giving a really good accuracy.
- Now we define two variable ***input*** and ***target*** which take small batches of the training dataset with its labels and perform training on that small batch.
- The training on this small batch of dataset is done till all the training samples have been used and this is done on function called ***feval***.
- The first step is to reset the gradients parameters to zero and then start the training on the batch dataset.
- We now forward the data to the network using the function ***network:forward(input)*** and this function returns a 10 class classification output with is then compared with the target labels initialized in the above steps to calculate the average loss and accuracy on the dataset.
- We now send the feval function, the network parameters and the configuration parameters to different cost functions for training the neural network using ***optim.sgd(feval, parameters, config).*** This done using sgd, asgd, adadelta and lbfsg. It was found that sgd gave the best accuracy on the neural network.
- Every iteration of the neural network we test the dataset to check the accuracy of the network on the test dataset. This is done on the entire test dataset instead of a batch test. To do testing on the dataset we forward the test dataset to the network and then send the output with the test labels to the criterion function to calculate the testing accuracy on the dataset.
- Next we clean the data to reduce the size of the network file and save the network as an output file.
- The last thing to do is to calculate accuracy that we get in the last epoch is the mean average precision on the testing dataset and is given as an output on the terminal.

III. Results

Applying the above algorithm yields the following outputs on different cost functions as shown in the images below.



```
th> dofile "prb2prta.lua"
Creating the network and the criterion...
Training...
epoch =  1/10  loss = 0.13 accuracy = 95.98
Testing...
epoch =  1/10  loss = 0.99 accuracy = 72.84
Training...
epoch =  2/10  loss = 0.06 accuracy = 98.36
Testing...
epoch =  2/10  loss = 0.55 accuracy = 83.53
Training...
epoch =  3/10  loss = 0.04 accuracy = 98.71
Testing...
epoch =  3/10  loss = 0.41 accuracy = 87.56
Training...
epoch =  4/10  loss = 0.04 accuracy = 98.91
Testing...
epoch =  4/10  loss = 0.43 accuracy = 87.53
Training...
epoch =  5/10  loss = 0.03 accuracy = 99.05
Testing...
epoch =  5/10  loss = 0.59 accuracy = 85.66
Training...
epoch =  6/10  loss = 0.03 accuracy = 99.09
Testing...
epoch =  6/10  loss = 0.25 accuracy = 92.84
Training...
epoch =  7/10  loss = 0.03 accuracy = 99.19
Testing...
epoch =  7/10  loss = 0.31 accuracy = 91.65
Training...
epoch =  8/10  loss = 0.03 accuracy = 99.20
Testing...
epoch =  8/10  loss = 0.29 accuracy = 91.74
Training...
epoch =  9/10  loss = 0.03 accuracy = 99.17
Testing...
epoch =  9/10  loss = 0.21 accuracy = 94.26
Training...
epoch = 10/10  loss = 0.03 accuracy = 99.22
Testing...
epoch = 10/10  loss = 0.34 accuracy = 91.50
The Mean Average Precision on the Testing Dataset = %.2f        91.5
                                                                      [676.6024s]
```

IV. Discussion

- From the above results we can conclude that the neural network is not designed to work with images of different color space.
- This is due to the fact that the network works on spatial arrangement of data points within a 2-d space and inverting the color of the image complements the locations of such data points.
- Since the data points the network is looking for are complemented it is unable to differentiate between the digits and the background
- Hence we made a training dataset which has both such images in them such that the neural network can train on both the positive and the negative dataset.
- Once the network was formed we could see that the randomly negated test dataset worked with really high accuracy and the neural network was able to classify the dataset.
- Hence we can conclude that a color spaced dataset should be first converted into grayscale and then a threshold needs to be applied to the dataset to make it black and white and then we will be able to classify any kind of dataset of any color.

(b) Application of LeNet-5 to MNIST Images with Background

I. Abstract and Motivation

The MNIST dataset is a set of handwritten digits written on black background or a white background. This just means that the digits are written on a background which has the same grayscale value. But in a real world scenario we have digits which will be written on a piece of paper with guide lines on them or digits written on a colored background. Hence the need to create a neural network we can work on digits written on random backgrounds.

II. Approach and Procedure

- The first step is to include all the packages that are needed for running the neural network. This is done using the ***require*** function to include packages in the .lua code file.
- A main function is defined where the neural network, training of the dataset and also the testing is performed.
- We first seed the random number generator such that it becomes easy for debugging such that it doesn't randomized any parameters during initialization
- Now is load the train and test dataset provided and store it in two torch tensor data variable called as ***dataset_train*** and ***dataset_test***.
- The next step is to divide this dataset into data and label and also converting the data into double data-type using the ***:double()***.
- The next step is to normalize the dataset with its mean and standard deviation. The training and the testing dataset are both normalized with their respective means and standard deviations.
- The data is normalized over 3 channels and this done in a for loop by indexing the dataset as follows : ***trainset.data[{ {},{i},{},{} }].*** Here the index location using variable ***"i"*** we will be able to normalize each channel of the image.
- Now we create a network and the criterion function for the back-propagation. A local ***network*** is initialized using a sequential neural network using ***nn.Sequential()*** and a local ***criterion*** using the class negative log likelihood criterion using n***n.ClassNLLCriterion().***
- Now we start defining the LeNet-5 neural network using the ***network*** that was initialized in the above step.
- We need to perform convolution for a 3 channel image dataset with 6 outputs channels using a 5 x 5 convolution kernel. We achieve this using ***network:add(nn.SpatialConvolution(3, 6, 5, 5))***
- Now we need to sample a small part of the layer i.e. a few neurons/nodes to node in the next layer. To perform this step we need to use ***network:add(nn.ReLU())*** which performs a non-linear connection between the above layer and connect it to the next layer.
- The step is not to perform a maximum pooling operation that looks at a 2 x 2 window on an image patch and find the maximum out of it and then a spatial convolution is performed. This is done using ***network:add(nn.SpatialMaxPooling(2,2,2,2))*** for maximum pooling and ***network:add(nn.SpatialConvolution(6, 16, 5, 5))*** for spatial pooling.
- After performing convolution we now again perform subsampling and maximum pooling using a 2 x 2 window on an image patch and find the maximum.
- Now that spatial convolution is performed twice we now need to convert the 3-d tensor of 16 x 5 x 5 into a 1-d tensor of 16*5*5. This is performed using ***network:add(nn.View(16*5*5))*** and this is done by connecting the entire layer to the next layer node thus forming a full connection between the two layers using ***network:add(nn.Linear(16*5*5, 120))*** function.
- Now moving on to the last layer we again perform a full connection but now we connect the 120 values to 84 values in that layer using ***network:add(nn.Linear(120, 84))*** and then this layer is converted into 11 number of outputs of the network for the 11 classes that we need for the

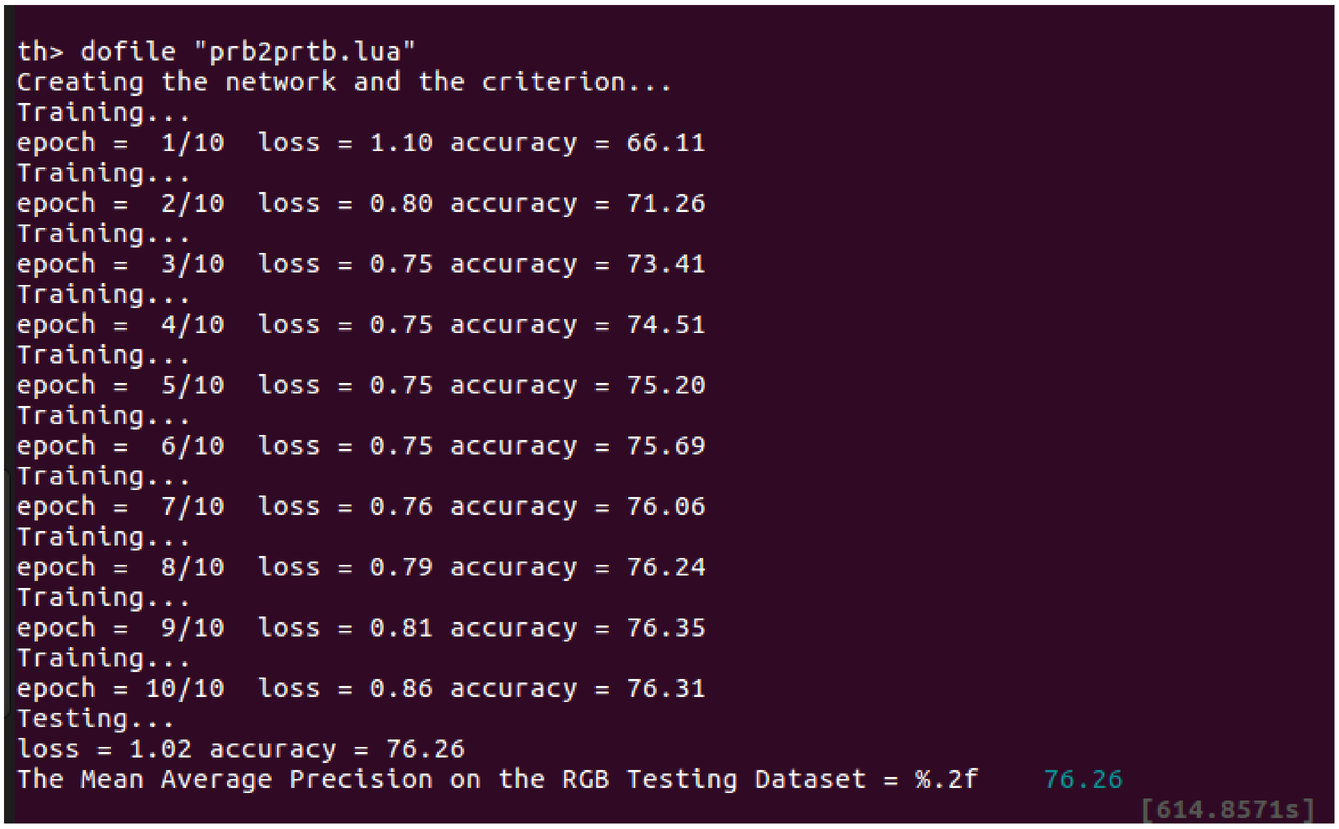
dataset to be classified into which includes an image which doesn't have any digit written on it and its done using ***network:add(nn.Linear(84, 11))***

- Once the network is formed and we have 11 output layer we apply a log-probability on the output of the dataset classification for getting probability values of each image being classified into to a particular class. This is done using the ***network:add(nn.LogSoftMax())*** function to get a log-probability value of the output of the dataset.
- Now that the network is initialized we need to extract the network parameters and arrange them linearly in memory so that we have a large vector containing all the information and parameter values of the neural network. This is performed using p***arameters,gradParameters = network:getParameters()*** function.
- The next step that is taken to initialize 4 arrays with zeros in them to calculate the loss and accuracy on the training set and testing dataset once the network has trained on the training dataset.
- To achieve this we run several epochs on small batches of the training set. An epoch in a neural network is an iterative training of the neural network as an epoch is a single pass through the entire training dataset which is usually followed by testing the network on the test dataset. For this algorithm we have found that running the epoch 10 times stabilizes the network accuracy,
- Now we create a batch size of 10 samples to train the network with. This is done to smoothen the gradients of the output of the network.
- Now we define a set of local classes as an array of characters ranging from 0 to 9 and also define the confusion matrix using ***optim.ConfusionMatrix(classes)*** where classes is defined as ***classes = {'0','1','2','3','4','5','6','7','8','9'}.***
- The training parameters need to be set now and the first thing we do is set the learning rate of the neural network to a value of 0.06. This is a pre-determined value and hasn't been changed as it does not give on the file reported code.
- We can seen that increasing the batch size increases the accuracy of the neural network but also increases the complexity of network and makes the algorithm run much slower. We also noted that decreasing the learning rate improves the accuracy of the network and we the network was tested with learning rates of 0.06 to 0.01 with 0.01 giving a really good accuracy.
- Now we define two variable ***input*** and ***target*** which take small batches of the training dataset with its labels and perform training on that small batch.
- The training on this small batch of dataset is done till all the training samples have been used and this is done on function called ***feval***.
- The first step is to reset the gradients parameters to zero and then start the training on the batch dataset.
- We now forward the data to the network using the function ***network:forward(input)*** and this function returns a 10 class classification output with is then compared with the target labels initialized in the above steps to calculate the average loss and accuracy on the dataset.
- We now send the feval function, the network parameters and the configuration parameters to different cost functions for training the neural network using ***optim.sgd(feval, parameters, config).*** This done using sgd, asgd, adadelta and lbfsg. It was found that sgd gave the best accuracy on the neural network.
- Every iteration of the neural network we test the dataset to check the accuracy of the network on the test dataset. This is done on the entire test dataset instead of a batch test. To do testing on the dataset we forward the test dataset to the network and then send the output with the test labels to the criterion function to calculate the testing accuracy on the dataset.
- Next we clean the data to reduce the size of the network file and save the network as an output file.

- Now we need to plot the accuracy of the training and testing dataset on a single graph. We use a function called gnuplot to print accuracy every epoch iteration. The last thing to do is to calculate accuracy that we get in the last epoch is the mean average precision on the testing dataset and is given as an output on the terminal.

III. Results

The results of the neural network run on the digits with background images is shown below and we can see that average precision of the neural network has reduced due to the presence of background noise. The below images contain a small portion of the test dataset and the output of the neural network is also shown in the images below.



```
th> dofile "prb2prtb.lua"
Creating the network and the criterion...
Training...
epoch =  1/10  loss = 1.10 accuracy = 66.11
Training...
epoch =  2/10  loss = 0.80 accuracy = 71.26
Training...
epoch =  3/10  loss = 0.75 accuracy = 73.41
Training...
epoch =  4/10  loss = 0.75 accuracy = 74.51
Training...
epoch =  5/10  loss = 0.75 accuracy = 75.20
Training...
epoch =  6/10  loss = 0.75 accuracy = 75.69
Training...
epoch =  7/10  loss = 0.76 accuracy = 76.06
Training...
epoch =  8/10  loss = 0.79 accuracy = 76.24
Training...
epoch =  9/10  loss = 0.81 accuracy = 76.35
Training...
epoch = 10/10  loss = 0.86 accuracy = 76.31
Testing...
loss = 1.02 accuracy = 76.26
The Mean Average Precision on the RGB Testing Dataset = %.2f    76.26
                                                                    [614.8571s]
```

IV. Discussion

- The above images show that the precision of the classification has reduced due to fact we have background noise.
- The neural network when performed on a black and white dataset becomes very easy to work because the network is able to define boundaries because we only have two gray-level values.
- Now considering the background noise dataset we have pixel values ranging from 0 to 255 and hence we have large number of variations in the pixel values which makes its difficult for the neural network to detect boundaries and hence the neural network fails to achieve an accuracy more that 90 % and we get a accuracy around 76 %.

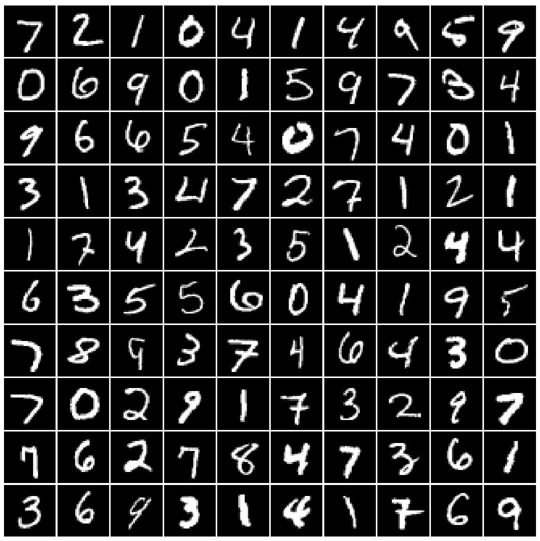
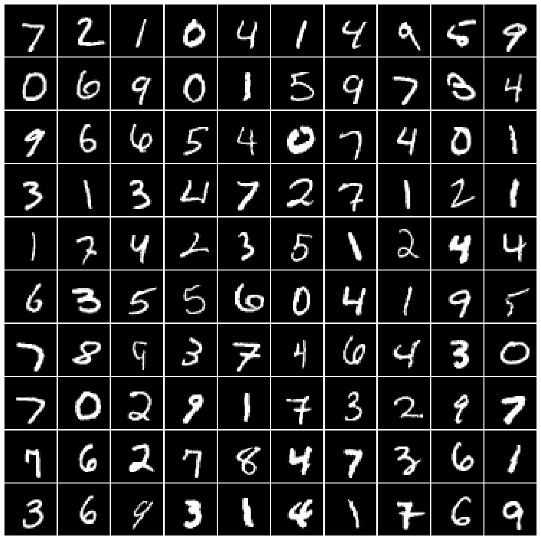
(c) Is LeNet-5 Translationally Invariant?

I. Abstract and Motivation

In this section of the code we are going to check if the digits written in spatial form gets detected by the neural network or not. Basically, we need to check the transitional invariance of neural network. This is done to check the accuracy of the neural network on the test dataset if the handwritten digits are not written in the center or if they are rotated and written. According to the theory of the neural network that we have discussed in the above questions in problem 1 we can that neural network should work with translated digits if the size of the images in the train and test dataset remain the same.

II. Approach and Procedure

- The first step is to include all the packages that are needed for running the neural network. This is done using the ***require*** function to include packages in the .lua code file.
- A main function is defined where the neural network, training of the dataset and also the testing is performed.
- We first seed the random number generator such that it becomes easy for debugging such that it doesn't randomized any parameters during initialization
- Now is load the train and test dataset provided and store it in two torch tensor data variable called as ***dataset_train*** and ***dataset_test***.
- The next step is to divide this dataset into data and label and also converting the data into double data-type using the ***:double()***.
- The next step over here is to pad the train and testing dataset using the ***pad=nn.SpatialZeroPadding(2,2,2,2)*** function and we need to forward the 2 datasets to this function and get an padded dataset.
- Now we randomly generate two numbers between 0 and 2 and use the translate function ***image.translate(testset.data[i],2,2)*** to randomly translate the digit by 0 to 2 pixel location within the same padded image size.
- The next step is to normalize the dataset with its mean and standard deviation. The training and the testing dataset are both normalized with their respective means and standard deviations.
- The translated test data set is shown below in comparison to non translated dataset.

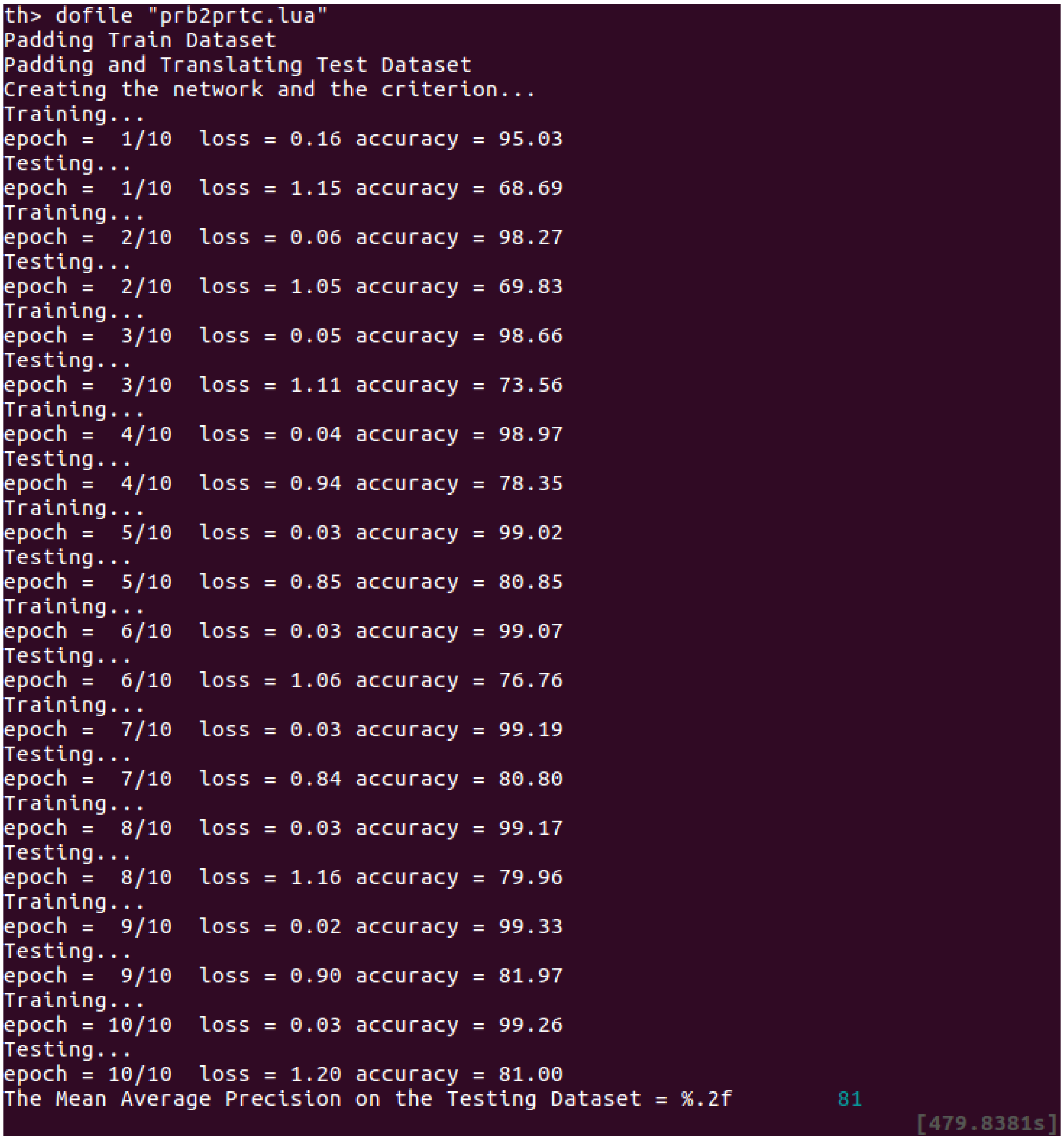



- Now we create a network and the criterion function for the back-propagation. A local ***network*** is initialized using a sequential neural network using ***nn.Sequential()*** and a local ***criterion*** using the class negative log likelihood criterion using n***n.ClassNLLCriterion().***
- Now we start defining the LeNet-5 neural network using the ***network*** that was initialized in the above step.
- We need to perform convolution for a 3 channel image dataset with 6 outputs channels using a 5 x 5 convolution kernel. We achieve this using ***network:add(nn.SpatialConvolution(3, 6, 5, 5))***
- Now we need to sample a small part of the layer i.e. a few neurons/nodes to node in the next layer. To perform this step we need to use ***network:add(nn.ReLU())*** which performs a non-linear connection between the above layer and connect it to the next layer.
- The step is not to perform a maximum pooling operation that looks at a 2 x 2 window on an image patch and find the maximum out of it and then a spatial convolution is performed. This is done using ***network:add(nn.SpatialMaxPooling(2,2,2,2))*** for maximum pooling and ***network:add(nn.SpatialConvolution(6, 16, 5, 5))*** for spatial pooling.
- After performing convolution we now again perform subsampling and maximum pooling using a 2 x 2 window on an image patch and find the maximum.
- Now that spatial convolution is performed twice we now need to convert the 3-d tensor of 16 x 5 x 5 into a 1-d tensor of 16*5*5. This is performed using ***network:add(nn.View(16*6*6))*** and this is done by connecting the entire layer to the next layer node thus forming a full connection between the two layers using ***network:add(nn.Linear(16*6*6, 120))*** function.
- Now moving on to the last layer we again perform a full connection but now we connect the 120 values to 84 values in that layer using ***network:add(nn.Linear(120, 84))*** and then this layer is converted into 10 number of outputs of the network for the 10 classes that we need for the dataset to be classified into which includes an image which doesn’t have any digit written on it and its done using ***network:add(nn.Linear(84, 10))***
- Once the network is formed and we have 10 output layer we apply a log-probability on the output of the dataset classification for getting probability values of each image being classified into to a particular class. This is done using the ***network:add(nn.LogSoftMax())*** function to get a log-probability value of the output of the dataset.
- Now that the network is initialized we need to extract the network parameters and arrange them linearly in memory so that we have a large vector containing all the information and parameter values of the neural network. This is performed using p***arameters,gradParameters = network:getParameters()*** function.
- The next step that is taken to initialize 4 arrays with zeros in them to calculate the loss and accuracy on the training set and testing dataset once the network has trained on the training dataset.
- To achieve this we run several epochs on small batches of the training set. An epoch in a neural network is an iterative training of the neural network as an epoch is a single pass through the entire training dataset which is usually followed by testing the network on the test dataset. For this algorithm we have found that running the epoch 10 times stabilizes the network accuracy,
- Now we create a batch size of 10 samples to train the network with. This is done to smoothen the gradients of the output of the network.
- Now we define a set of local classes as an array of characters ranging from 0 to 9 and also define the confusion matrix using ***optim.ConfusionMatrix(classes)*** where classes is defined as ***classes = {'0','1','2','3','4','5','6','7','8','9'}.***
- The training parameters need to be set now and the first thing we do is set the learning rate of the neural network to a value of 0.06. This is a pre-determined value and hasn’t been changed as it does not give on the file reported code.

- We can seen that increasing the batch size increases the accuracy of the neural network but also increases the complexity of network and makes the algorithm run much slower. We also noted that decreasing the learning rate improves the accuracy of the network and we the network was tested with learning rates of 0.06 to 0.01 with 0.01 giving a really good accuracy.
- Now we define two variable ***input*** and ***target*** which take small batches of the training dataset with its labels and perform training on that small batch.
- The training on this small batch of dataset is done till all the training samples have been used and this is done on function called ***feval***.
- The first step is to reset the gradients parameters to zero and then start the training on the batch dataset.
- We now forward the data to the network using the function ***network:forward(input)*** and this function returns a 10 class classification output with is then compared with the target labels initialized in the above steps to calculate the average loss and accuracy on the dataset.
- We now send the feval function, the network parameters and the configuration parameters to different cost functions for training the neural network using ***optim.sgd(feval, parameters, config).*** This done using sgd, asgd, adadelta and lbfsg. It was found that sgd gave the best accuracy on the neural network.
- Every iteration of the neural network we test the dataset to check the accuracy of the network on the test dataset. This is done on the entire test dataset instead of a batch test. To do testing on the dataset we forward the test dataset to the network and then send the output with the test labels to the criterion function to calculate the testing accuracy on the dataset.
- Next we clean the data to reduce the size of the network file and save the network as an output file.
- Now we need to plot the accuracy of the training and testing dataset on a single graph. We use a function called gnuplot to print accuracy every epoch iteration. The last thing to do is to calculate accuracy that we get in the last epoch is the mean average precision on the testing dataset and is given as an output on the terminal.

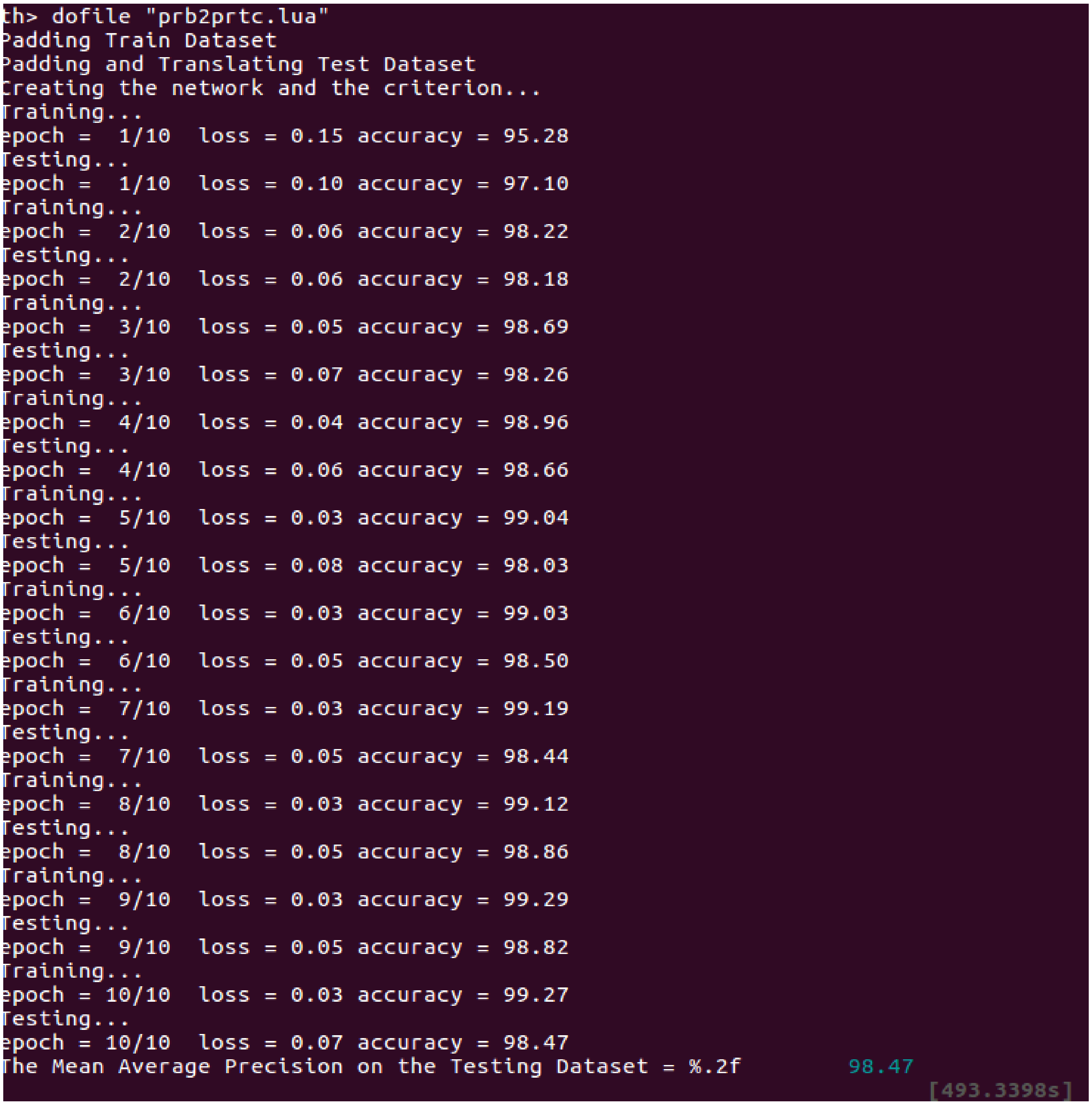
III. Results

The results of the neural network to check for transitionally invariance is shown below and we can that the networks fine with translated digit in images.



```
th> dofile "prb2prtc.lua"
Padding Train Dataset
Padding and Translating Test Dataset
Creating the network and the criterion...
Training...
epoch =  1/10  loss = 0.16 accuracy = 95.03
Testing...
epoch =  1/10  loss = 1.15 accuracy = 68.69
Training...
epoch =  2/10  loss = 0.06 accuracy = 98.27
Testing...
epoch =  2/10  loss = 1.05 accuracy = 69.83
Training...
epoch =  3/10  loss = 0.05 accuracy = 98.66
Testing...
epoch =  3/10  loss = 1.11 accuracy = 73.56
Training...
epoch =  4/10  loss = 0.04 accuracy = 98.97
Testing...
epoch =  4/10  loss = 0.94 accuracy = 78.35
Training...
epoch =  5/10  loss = 0.03 accuracy = 99.02
Testing...
epoch =  5/10  loss = 0.85 accuracy = 80.85
Training...
epoch =  6/10  loss = 0.03 accuracy = 99.07
Testing...
epoch =  6/10  loss = 1.06 accuracy = 76.76
Training...
epoch =  7/10  loss = 0.03 accuracy = 99.19
Testing...
epoch =  7/10  loss = 0.84 accuracy = 80.80
Training...
epoch =  8/10  loss = 0.03 accuracy = 99.17
Testing...
epoch =  8/10  loss = 1.16 accuracy = 79.96
Training...
epoch =  9/10  loss = 0.02 accuracy = 99.33
Testing...
epoch =  9/10  loss = 0.90 accuracy = 81.97
Training...
epoch = 10/10  loss = 0.03 accuracy = 99.26
Testing...
epoch = 10/10  loss = 1.20 accuracy = 81.00
The Mean Average Precision on the Testing Dataset = %.2f        81
                                                                            [479.8381s]
```

The above result is based on padded and translated test dataset which are tested on a neural network trained on a padded training dataset.



```
th> dofile "prb2prtc.lua"
Padding Train Dataset
Padding and Translating Test Dataset
Creating the network and the criterion...
Training...
epoch =  1/10  loss = 0.15 accuracy = 95.28
Testing...
epoch =  1/10  loss = 0.10 accuracy = 97.10
Training...
epoch =  2/10  loss = 0.06 accuracy = 98.22
Testing...
epoch =  2/10  loss = 0.06 accuracy = 98.18
Training...
epoch =  3/10  loss = 0.05 accuracy = 98.69
Testing...
epoch =  3/10  loss = 0.07 accuracy = 98.26
Training...
epoch =  4/10  loss = 0.04 accuracy = 98.96
Testing...
epoch =  4/10  loss = 0.06 accuracy = 98.66
Training...
epoch =  5/10  loss = 0.03 accuracy = 99.04
Testing...
epoch =  5/10  loss = 0.08 accuracy = 98.03
Training...
epoch =  6/10  loss = 0.03 accuracy = 99.03
Testing...
epoch =  6/10  loss = 0.05 accuracy = 98.50
Training...
epoch =  7/10  loss = 0.03 accuracy = 99.19
Testing...
epoch =  7/10  loss = 0.05 accuracy = 98.44
Training...
epoch =  8/10  loss = 0.03 accuracy = 99.12
Testing...
epoch =  8/10  loss = 0.05 accuracy = 98.86
Training...
epoch =  9/10  loss = 0.03 accuracy = 99.29
Testing...
epoch =  9/10  loss = 0.05 accuracy = 98.82
Training...
epoch = 10/10  loss = 0.03 accuracy = 99.27
Testing...
epoch = 10/10  loss = 0.07 accuracy = 98.47
The Mean Average Precision on the Testing Dataset = %.2f        98.47
                                                                        [493.3398s]
```

The above result is based on padded and translated test dataset which are tested on a neural network trained on a padded and translated training dataset.

IV. Discussion

- In the above discussions we have defined a neural network to work with any orientation of digits in image provided. For a neural network it does not matter if the digits are zoomed in or zoomed out or rotated or translated.
- If pad and translate the training dataset too then the accuracy of the neural network improves by 17 % and give use an accuracy of 98 % instead of 81 %.
- We can now conclude that if the training data is padded and translated the network is able to train much better and hence better accuracy.

References

[1] [Online] http://yann.lecun.com/exdb/mnist/
[2] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324
[3][Online]https://en.wikipedia.org/wiki/Rectifier_(neural_networks).
[4][Online]https://en.wikipedia.org/wiki/Softmax_function
[5] C.-C. Jay Kuo, "Understanding CNN with a mathematical model" 2016, arXiv 1609.04112.
[6][Online]https://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi
[7][Online]http://deeplearning.net/tutorial/lenet.html
[8][Online]https://github.com/soumith/cvpr2015/blob/master/Deep%20Learning%20with%20Torch.ipynb
[9][Online]https://github.com/torch/image/blob/master/doc/simpletransform.md
[10][Online]http://stats.stackexchange.com/questions/114385/what-is-the-difference-between-convolutional-neural-networks-restricted-boltzma
[11][Online]http://www.ersatz1.com/new/demos/