

EE 569 Homework #4

Maroof Mohammed Farooq

maroofmf@usc.edu

7126869945

Table of Contents

1. Problem 1	3
1.1 Abstract and Motivation	3
1.2 CNN Architecture	3
1.2.1 Difference between LeNet – 5 and ANN	5
1.2.2 Why LeNet – 5 performs better than ANN?.....	5
1.3 Approach and Procedures	5
1.4 Experiment Results	6
1.5 Discussion	7
2. Problem 2	8
2.1 Abstract and Motivation	8
2.2 Approach and Procedures	8
2.2.1 Negative Images	8
2.2.2 Color Images.....	8
2.2.3 Translation Invariance.....	8
2.3 Experiment Results.....	9
2.4 Discussion.....	10
4. References	11

1. Problem 1

1.1 Abstract and Motivation:

With the availability of big data and enormous computing resources, it is now possible to realize the idea of neural networks. Neural network is an approach to model a data set for incorporating a form of intelligence in machines. Neural network is based on a large collection of neurons. These neurons are inspired from the working of the human brain. These neurons connect in a certain fashion which depends on the architecture of the network. Providing enormous data will help the network to learn from its intricacies and memorize the underlying pattern of the provided data.

Back in 1998, the director of Facebook AI research Yann LeCun wrote a paper on handwritten digit recognition that became famous as it had proven to be an accurate model. Since then, the architectures of the neural networks were tweaked and applied to big datasets like ImageNet and CIFAR. The huge improvement in results over the traditional methods played in favor of neural networks. Massive efforts have been put forwards by researchers to optimize and generalize these networks. In this problem, we will make an attempt to solve and investigate handwritten character recognition using convolutional neural networks.

1.2 CNN Architecture:

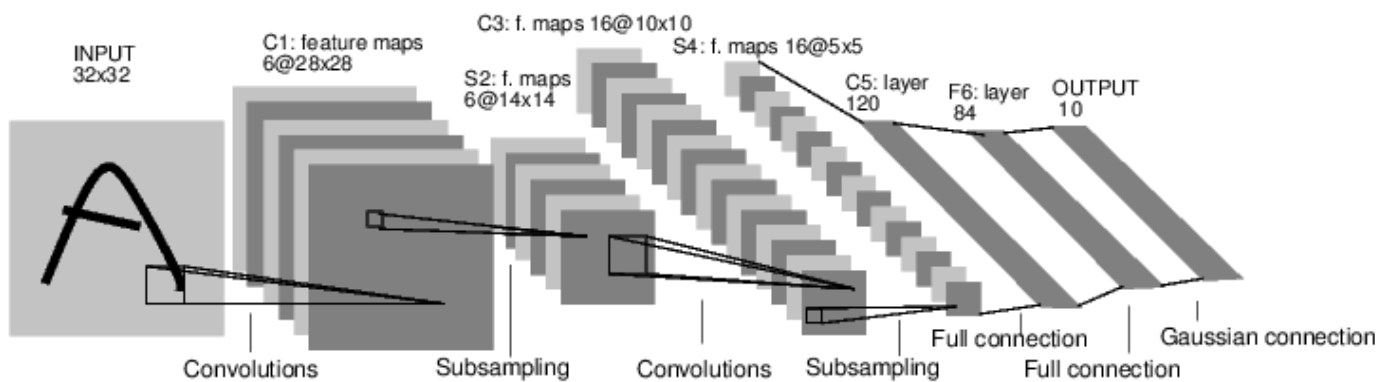


Figure 1.1: LeNet – 5 Architecture

Figure 1.1 shows the LeNet – 5 architecture. It comprises of two spatial convolution and max pooling layers with two fully connected hidden layer. It contains a total of 61,706 parameters that are optimized over the training period. The following are the modules placed in LeNet – 5 with their respective orders:

- **Input Layer:** This layer provides the input pixel values to the network. The size of this layer typically depends on the size of images in the training and testing databases. For the MNIST dataset, the dimensions of the input images were 32x32. Hence the input layer consists of 32x32 neurons arranged spatially.
- **Spatial Convolution 1:** When dealing with high dimensional input data such as images, it gets impractical to connect each neuron to the neuron in the following layer. Let me convince you why. For the sake of argument, let us consider the connectivity between the input and the following layer. If we go with the traditional approach of connecting all the input neuron with the next layer, we end up having 4,821,600 weights to be learnt in the first layer itself. This may cause the performance of the neural network to degrade if we don't have sufficient training samples. A smarter approach is to connect each neuron in the output to only a local region of the input. This reduces the receptive field of each neuron which is compensated for in the next layers of this architecture. In the LeNet-5, the receptive field was selected to be 5x5. So each output neuron would see a 5x5 patch from the input. Additionally, we reduce the number of weights by introducing

parameter sharing. This allows the neurons in one output layer to share the same set of weights over the entire input. This is done by assuming that if one feature is useful for a patch in the input space, then it might be useful for other patches in the input space as well. This assumption has proven to be reasonable. In LeNet 5 we have set the receptive field as 5x5 with no zero padding and stride = 1. This operation is similar to convolution. The input image is convolved over 6 filter banks and thus an image of size 32x32 gets transformed to 28x28 with 6 layers. The number of weights in this layer is 156.

- **Spatial Max-pooling 1:** This layer subsamples the output of the preceding layer. It is done to improve the receptive field with respect to the input layer and reduce the number of weight in the subsequent layers. In LeNet-5, a 2x2 window is considered with stride =2 and an L_∞ norm in the selected window. The dimension of the input after max pooling with the specified hyper parameters will become 14x14. There are no weights involved in this layer.
- **ReLU Non-Linear Activation 1:** This layer will apply an element-wise activation function that is threshold at zero. The input size remains unchanged and there are no weights to be learned in this layer. The ReLu function is given as follows:

$$\text{output} = \max(0, \text{input}) \text{ -----(Equation 1.1)}$$

The purpose of using a non linear activation layer is to introduce this non-linearity in the network. Real world data and processes can be non linear in nature. If we train the system without this, then the performance will degrade if the data to be modelled is non linear in nature. To effectively utilize the power of neural networks, we introduce a non-linear activation function. Additionally, we use ReLu as it has been proven to converge in less time.

- **Spatial Convolution 2:** Similar to spatial convolution 1, it takes an input which is 14x14x6 in size and convolves using a receptive field of 5x5. This layer outputs 16 layered images of size 10x10. The number of weights in this layer are 2416.
- **Spatial Max-pooling 2:** In this layer, we subsample the input of size 10x10x16 to 5x5x16. We scan the input by a window of size 2x2 and stride = 2.
- **ReLU Non-Linear Activation 2:** Again, the function of this unit is similar to the one in layer 1. Up until this point, we reduced the dimension of the input image from 1024 neurons to 400 neurons. In essence, the previous layers were utilized for automated dimension reduction and feature extraction. Now, we pass these 400 features to the following fully connected layers.
- **Fully Connected Hidden Layer 1:** This layer consists of 120 neurons. It is connected from 400 neurons in the previous layer. The number of weights with biases that needs to be learnt are 48,120. It is also important to note that the fully connected layers have the maximum receptive field. It simply means that they each neuron has all the information about the input image. These layers have ReLu activations in their outputs to model non-linearity.
- **Fully Connected Hidden Layer 2:** This layer has 84 neurons. It is connected from 120 neurons from the previous layer. The number of weights with biases that needs to be learnt are 10,164. This layer is also equipped with ReLu activations to provide non linearity.
- **Output Layer:** This layer consists of 10 neurons which represent the output labels. It is connected from 84 neurons in the previous layer. The number of weights with biases that needs to be learnt are 850. This is cascaded with a log soft max layer that provides the log probabilities of the output labels. It is a method to

normalize the outputs to easily determine the output label for each input. The log softmax function is applied to each neuron and is given by:

$$s(x) = \log \left(\frac{e^x}{\sum_{i=1}^N e^i} \right) \text{ -----(Equation 1.2)}$$

Here, the output of each neuron x is passed to this function. All the N neurons are computed with the same formula. During the training process, these values are passed to the negative log likelihood criterion to reduce cross entropy.

In total the network has **61,706 weights** to be trained.

1.2.1 Difference between LeNet-5 and ANN:

Prior to the inception of LeNet-5, classifiers such as SVM, ANN and K-NN were used. The main challenge with using these classifiers was feature extraction. Feature engineering was a big field of study. There was a positive correlation between quality of features and performance of these classifiers. To boost the performance, an automated feature extraction phase was added to the traditional ANN. This phase effectively reduces the number of weights to be learnt directly from input image and memorizes important features automatically. These powerful features set the main differences between LeNet-5 and ANN. To summarize:

- **LeNet-5 can take correctly sized images as its inputs where as the traditional ANN cannot.** The number of learning parameters will exponentially rise if we happen to modify the ANN to accept images as its input. This may cause improper training as the system is undetermined.
- LeNet-5 has spatial convolution and max pooling layers that extract **meaningful features** and reduce the number of weights to be learnt whereas ANN requires handcrafted features.

1.2.2 Why LeNet-5 works better than ANN?:

One of the main differences between LeNet-5 and ANN is the feature extraction stage. In LeNet-5, we extract features that the machine thinks are useful whereas in ANN, the handcrafted features may not fit the data well. Due to these differences, the LeNet-5 demonstrates better robustness and performance than ANN.

1.3 Approach and Procedures:

The architecture discussed above was assembled together in Lua using Torch. In addition to this, we perform data normalization by dividing each pixel by 255. This is done to get the input pixels ranging from 0 to 1. Later the parameters of the neural networks were tweaked as follows:

Weight Initialization: After the network is built, we initialize the weights to a suitable value. Various methods were tried on using the open source code provided on Github.^[1] The weight initialization method in [2], [3], [4] and [5] were utilized and the convergence performance on each was tested. The best way to initialize the network by using the following formula:

$$init_{weight} = \sqrt{\frac{1}{receptive\ field\ from\ the\ previous\ layer}} \text{ -----(Equation 1.3)}$$

Cost Function Selection: Cost function determines the magnitude of error in classifying the output. The cost function selected was the negative log likelihood criterion. This is because it reduces the cross entropy. Cross entropy can also be related as a measure of surprise. Let us consider the binary classification problem's output in table 1.1. NLL seems to be the perfect choice for our loss function as it produces the desirable loss output.

Case	Label	Predicted	Desired Loss	MSE	NLL
1	1	1	0	0	0
2	1	0	+ Inf	0.5	+ Inf
3	0	1	+ Inf	0.5	+ Inf
4	0	0	0	0	0

Table 1.1: Performance of loss functions

Batch size: Our approach to training is called mini-batch gradient descent. Here the weights are updated after a batch of input is processed. The size of batch contributes heavily towards convergence and performance of the neural network. This is a hyper-parameter of the network that needs to be optimized using brute force. We have used a batch size of 10 samples per update. By using batches, we reduce the variance of the updates. Larger batch sizes contribute towards an unstable update vectors which will converge slowly. Challenges involving selection of a suitable learning rate for a particular batch size needs to be studied carefully.

Learning Rate: Learning rate determines the time for convergence of the network. Smaller learning rates converge painfully slow whereas larger learning rates may not converge to the global minima. These challenges are addressed by implementing a learning rate decay value or brute force search. For the given data and batch size, the network converged well for 0.06.

Momentum: The optimization technique used for updating weights is mini-batch stochastic gradient descent method. SGD has problems navigating ravines. It oscillates in these regions that can be detrimental for converging. To prevent these oscillations, we set the momentum parameters. In this network, we have set the momentum as 0.9.

Data Shuffling: Neural networks are a strong tool for leaning. Its strengths may sometimes cause overfitting. Overfitting is the phenomenon in which the data gets modelled to the noise in the training set. This may result in poor performance in the test set. To prevent overfitting or learning the data in series, the training set was randomly shuffled and trained in batches. At every epoch the training data is shuffled.

Drop Out: The underlying concept of drop out is to neglect a certain proportion of neurons in layer during a batch. This reduces overfitting as it restricts the strong features to dominate over the weaker ones. In this experiment, the concept of drop out was tried, but the probability of dropout couldn't be tuned to work in favor for the given data set.

1.4 Experiment Results:

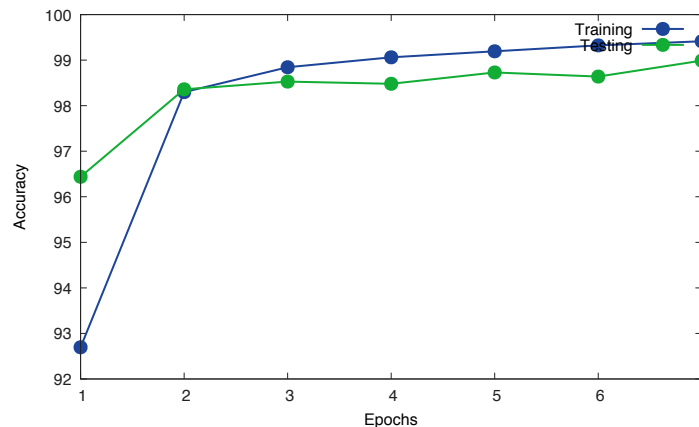


Figure 1.2: Epoch – Accuracy curve for training and testing

1.5 Discussion:

From figure 1.2, we can notice the learning curve of our network. In the initial epochs, the model doesn't perform well on the training and test set. Early stopping strategy was utilized to decide the number of epochs. If we train the system for more than 7 epochs, it starts to learn the noise present in the training set and cause overfitting. For better starting accuracy and faster convergence, we can initialize the network using equation 1.3.

The mean accuracy precision on train set was found to be 99.41 and the test set was 98.98. The mean accuracy precision heavily depends on the model parameters and architecture. Figure 1.3 shows the training and testing curves when the batch size was changed from 10 to 200. **The mean accuracy precision on the train set converged to 97.86 and the test set converged to 97.95.** This is considered as poor training. We can observe the drop in performance because for the selected batch size, the learning rate and momentum were not tuned correctly. It is thus very important to tune all the hyper parameters before concluding the performance.

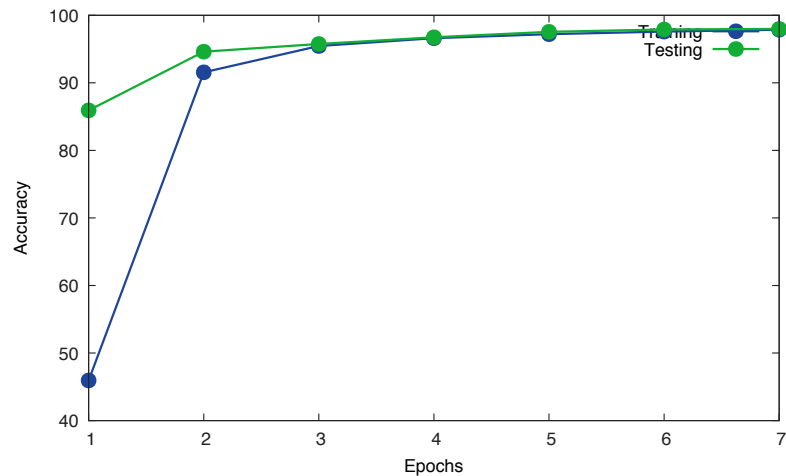


Figure 1.3: Epoch – Accuracy curve for training and testing with batch size = 200

2. Problem 2

2.1 Abstract and Motivation

Convolutional Neural Networks perform very well on the training and testing data that were provided to them by optimizing the hyper-parameters. The actual test for these networks are with the real time – real world data. There are many cases where the CNN may fail due to deficiencies in the training or testing dataset. It is important for us to look into these deficiencies and try to overcome it.

Few deficiencies may include switching between background and foreground values, noisy addition, rotation variance, translation variance, changing of stroke width, squeezing etc. In this problem we will discuss the limitations of convolutional neural networks. We will also make an attempt to overcome these limitations.

2.2 Approach and Procedures

In this experiment, we will :

- Test our trained model from problem 1 on negative images
- Train a new model to perform better on both positive and negative images
- Train and test a new model on MNIST with background images.
- Test for translation invariance
- Fix the translation invariance problem.

2.2.1 Negative Images:

It is important for us to understand the limitations of CNN and attempt to overcome these limitations. One such limitation is that our CNN performs poorly when the image are negated i.e. the background and foreground values are interchanged. This is caused due to the dependence of our CNN on the pixel values. To effectively remove this dependence, we change the training data while retaining the network architecture and training parameters.

In our new dataset, we will include the negative image samples along with the original samples. The training data size is doubled and shuffled randomly so that positive and negative data is presented randomly to the network. By doing this, we can improve the output of the network by a large value.

2.2.2 Color Images:

The only difference between color and grayscale networks is the input dimensionality. The convolutional neural network developed in problem 1 is used but the input spatial convolution layer is changes from 1->6 to 3->6 layer. This was done to input color information. Also, we now have 3d spatial filters in the first convolution rather than 1d spatial filter. Data normalization and training parameters/procedures are followed as done before.

2.2.3 Translation Invariance:

To test for translation invariance, the model developed in problem 1 is selected and the input image is translated along the horizontal and vertical axis. The accuracy is recorded and plotted. Further discussion on this is made in the discussion section.

To achieve translation invariance, we randomly translate the training samples between -5 to 5 pixels from the center. This is achieved by generating a 2D uniformly random number between -5 and 5. Later the dataset is shuffled to provide more randomness and then it is given for training.

2.3 Experiment Result

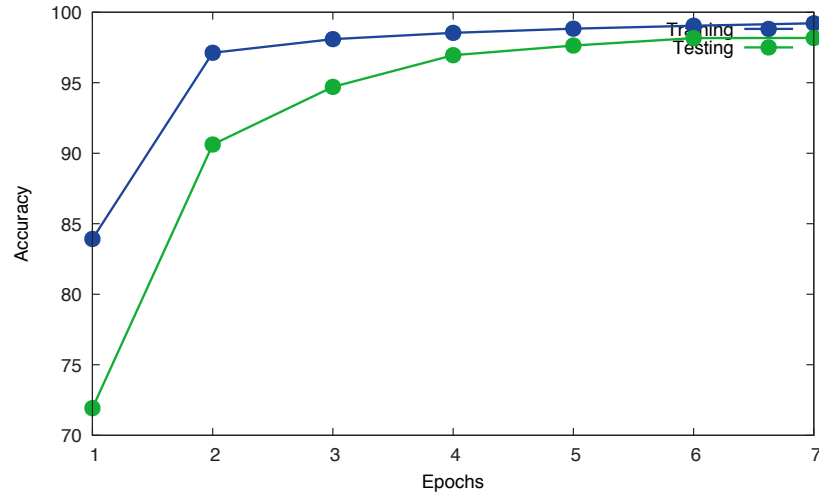


Figure 2.1: Epoch – Accuracy curve for training and testing of negative and positive images

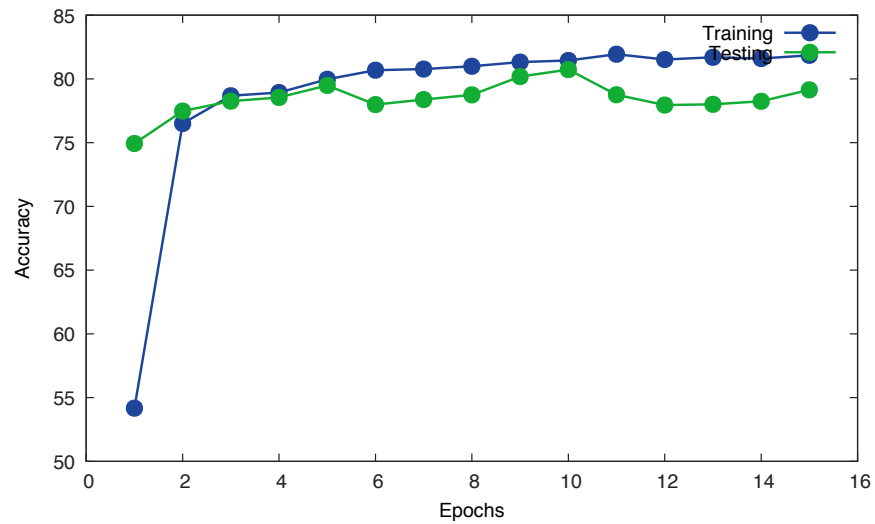


Figure 2.2: Epoch – Accuracy curve for training and testing of color images

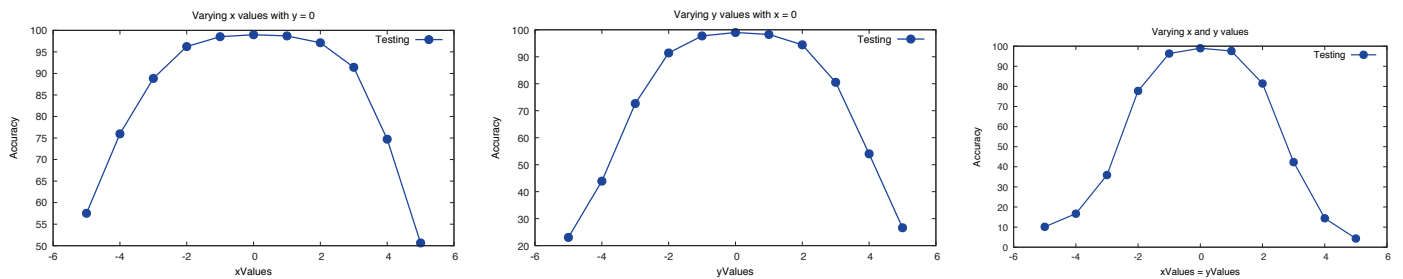


Figure 2.3: Translation – Accuracy curve to test translation invariance

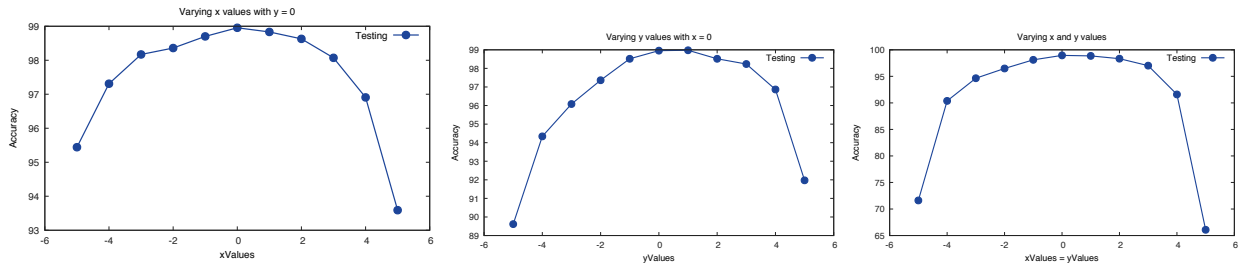


Figure 2.4: Translation – Accuracy curve after training for translational invariance

2.4 Discussion

The problem with our network trained in problem 1 was that it was heavily dependent on the pixel values rather than the underlying structure of the number. This can be attributed to the quality of training data. If our training data contains black background and white foreground, then it models these features. This is why we get a bad accuracy when a test set with negative data is passed. **The mean accuracy precision on negative test images were computed to be 33.61.**

Figure 2.1 shows the learning curve of the network on positive and negative data. **The final recorded mean accuracy precision of the newly trained model on the train set was 99.20 and on the test set was 98.13.** The mixture of negative and positive samples removed the dependence of the background and foreground color and improved the robustness of the network against changing background colors.

From figure 2.2 we can see the learning curve of the network on color images. **The mean accuracy precision was found to be 81.76 on the training set and 79.10 on the testing set.** From figure 2.5, we can see that the accuracy drops because the network is unable to model for “No Class”. The no class data samples are very random and it has effectively hindered the training process. As seen from the confusion matrix in figure 2.5, we can see a maximum mismatch with the “No Class” category.

The image net is not robust against big translations. A translation of +/- 3 pixels have shown to attain an accuracy above 95%. This can be attributed to the max-pooling layers that increase the receptive field of the network to learn high level features. Translating the image by more than 3 pixels drops the accuracy greatly. This can be shown by studying figure 2.3. It can be seen that accuracies can drop down to 10 percent. This happens when the digits are closer to the boundary. To improve its performance we have trained a new network with randomly translated images and plotted its performance. It can be seen in figure 2.4. Here the minimum accuracy is boosted from 10 percent to 70 percent and for small translations, it performs satisfactory as well.

```
ConfusionMatrix:
[[ 756   3   3   7   8  21  41   6   7   8  32] 84.753% [class: 0]
 [   2 866  29   6  14   6  17   8  11   4  58] 84.819% [class: 1]
 [  52  27 680  30   9  11  14  27  35   2  45] 72.961% [class: 2]
 [   9   6  12 737  10  57  15   9  17  11  30] 80.723% [class: 3]
 [   8  10   6  10 675  10  28  22  12  46  57] 76.357% [class: 4]
 [  12   7   2  31  13 619  23   6  42  10  36] 77.278% [class: 5]
 [  18  11   3  11  15  20 713   3  26   8  34] 82.715% [class: 6]
 [  11  19  15  15  22   4   8 753  13  33  47] 80.106% [class: 7]
 [   8   7  14  51  12  20  18   8 687  14  26] 79.422% [class: 8]
 [  10  12   4  16  51  25   8  45  28 670  34] 74.197% [class: 9]
 [  23  43  15  17  31  22  28  18  15  17 758] 76.798% [class: NC]
+ average row correct: 79.102796858007%
+ average rowCol correct (VOC measure): 65.681600570679%
+ global correct: 79.14%
```

Figure 2.5: Confusion matrix from testing color images

References :

- [1] Online Resource used in code: <https://github.com/e-lab/torch-toolbox/blob/master/Weight-init/weight-init.lua>
- [2] Yan LeCun, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Muller, “Efficient BackProp”.
- [3] Glorot Xavier, “Understanding the difficulty of training deep feedforward neural networks ”, 2010.
- [4] Kaiming He, “Delving deep into rectifiers: Surpassing human level performance on ImageNet Classification”, 2010.