

College of Engineering and Technology
Mody University of Science and Technology, Lakshmangarh

CERTIFICATE

This is to certify that the project work entitled “**Digit Recognition using Neural Networks: A case study approach**” submitted by **Gargee Sanyal** and **Charul Rathore** in fulfilment for the requirements of the award of Bachelor of Technology Degree in Computer Science and Engineering at College of Engineering and Technology, Mody University of Science and Technology, Lakshmangarh is an authentic work carried out by them under my supervision and guidance. To the best of my knowledge, the matter embodied in the project has not been submitted to any other University/Institution for the award of any degree in India or abroad.

Dr. A.K. Singh
Assistant Professor, CSE

CANDIDATE'S DECLARATION

We hereby declare that the work presented in this report entitled “**Digit Recognition using Neural Networks: A case study approach**” in fulfilment for the requirements of the award of Bachelor of Technology Degree in Computer Science and Engineering at College of Engineering and Technology, Mody University of Science and Technology, Lakshmangarh is an authentic record of our own work carried out during our degree under the guidance of Dr.A.K.Singh.

The work reported in this has not been submitted by us for the award of any degree or diploma to any other University/Institution in India or abroad.

Date: 19/12/2017

Charul Rathore (140135)

Place: Lakshmangarh

Gargee Sanyal (140209)

ACKNOWLEDGEMENT

First and foremost, we would like to thank God Almighty for giving us the strength, knowledge, ability and opportunity to undertake this research study and to persevere and complete it satisfactorily. Without his blessings, this achievement would not have been possible.

We have great pleasure in acknowledging our gratitude to **Mody University**, the President, Dean and HOD. They have been providing their heartfelt support and guidance at all times and gave us invaluable guidance, inspiration and suggestions in our quest for knowledge. They gave us all the freedom to pursue our research, while silently and non-obtrusively ensuring that we stay on course and do not deviate from the core of our research. Without their able guidance, this project would not have been possible and we shall eternally be grateful to them for their assistance.

We take pride in acknowledging the insightful guidance of **Dr. A.K. Singh** for sparing his valuable time whenever we approached him and showing me the way ahead. This project would not have been possible without his kind support and help.

ABSTRACT

Neural networks are a robust technology for pattern recognition and stratification of visual inputs. This paper delineates a comparative study of various implementations of neural networks for digit recognition on two platforms that can be availed to get good results. The feasibility on both the platforms is compared extensively; including but not limited to performance, efficiency and accuracy amongst others. Our claims are illustrated on the MNIST database of handwritten digits. The open source software library Tensorflow along with Python2 environment for scripting is used to carry out simple Feed Forward Neural network. MATLAB is used to develop a model using Deep Neural Networks to recognise digits when it is in the form of an image. The model uses Convolutional Neural Networks(CNN) architecture. The batch variable in this case is 100 common to both. Weights and bias are updated in each epoch to produce accurate results. The performance of how they are updated and whether or not and how they conform to the desired results are compared in this study.

TABLE OF CONTENTS

S.NO.	TITLE	PAGE NO.
	<i>List of Figures</i>	<i>vii</i>
Chapter 1	Introduction	8
1.1	Aim	8
1.2	Intended Audience	8
1.3	Scope	8
1.4	Motivation	9
1.5	Approach	9
1.6	Assumptions	11
1.7	Expected Outcomes	11
Chapter 2	Literature Review	12
Chapter 3	Research Gap	14
Chapter 4	Objective	15
Chapter 5	Design and Implementation	16
5.1	Defining and training CNN	16
5.2	Model Development	17
5.3	Data Collection	18
5.4	Using TensorFlow	18
5.4.1	What is TensorFlow	18
5.4.2	Methodology	19
5.5	Using MATLAB	20
5.5.1	What is MATLAB	21
5.5.2	Testing CNN	21
Chapter 6	Testing and Results	22
6.1	Test Results on Tensorflow	22
6.2	Test Results on MATLAB	24

6.3	Training and Testing with one hundred epochs	25
6.4	Running CNN on our image	26
	Conclusion	28
	References	29
	Self Assessment Report	30

LIST OF FIGURES

S.NO	TITLE	PAGE NO.
1.1	Convolutional neural network with eight layers	10
5.1	An example of Fully Connected MLP	16
5.2	Architecture of CNN	17
5.3	Project Methodology	17
5.4	MNIST data Representation	18
5.5	Learning Proficiency Plot	19
5.6	Operations on placeholders	19
5.7	Softmax function visualization	20
5.8	Gradient Descent Algorithm	20
5.9	MNIST Dataset in MATLAB Workspace	21
6.1	Predictions on Test Case	22
6.2	TensorFlow with epochs=30	23
6.3	TensorFlow with epochs=100	23
6.4	TensorFlow CNN with epochs=20,000	24
6.5	Testing CNN for 1 Epoch	25
6.6	Plot of Mean Squared Errors and Batches	25
6.7	MATLAB Command View after 100 Epochs	26
6.8	Mean Squared Error for 100 Epochs	26
6.9	Digits to be recognized	26
6.10	CNN correctly classifying handwritten digit 'seven'	27

INTRODUCTION

1.1 Goal

Image classification, detection and segmentation are some of the problems at which now Deep Neural Networks shine. The most powerful hardware can also be brought down to its knees when there is a high computational complexity used to scan images by means of sliding window with most appropriate masks. Thus, our project shows the acceleration of the process by classifying digits in orders of magnitude using dynamic programming and Deep Neural Networks even in the presence of Max-Pooling layers.

1.2 Intended Audience

This project is intended for anyone who wants to learn about computer vision and the various efficient ways to do it. Clients or developers who need perception problems to be solved in an efficient manner, without making the solution too complex, which reduces the chances of errors, making their detection and removal much easier. Despite the sophistication of the calculations involved, some very complicated problems can easily be handled by frameworks created for deep learning.

1.3 Scope

The advancement in technologies that has happened over the last decade is exponential. In every part of the world, top most technologies are being used to improve existing products while also investing immense research into discovering products that make the world not only better, but the best place to live in. Deep learning algorithms have played a very important part, whether it be autopilot cars, amazon walk-in technology, amongst others. This project solves a very important part of these recent developments, i.e perception. We've trained a model to identify digits, primarily using classification in both Matlab and TensorFlow. Perception forms the very important problems related to computer vision. Recognising what is in an image is what this classifier is trained to do, which in this particular scenario are digits written in various styles and formatting.

1.4 Motivation

The motivation to learn deep learning, hence this project came from the amazing technologies that have been developed by Amazon and Google such as Walk-In technology, Alexa, Alpha-Go, etc to name a few. Also, Elon Musk has made headlines for how Tesla cars drive themselves. These are but a few examples. More less cerebrated examples also include their involvement in curing and developing cancer medications, predicting stock prices, etc. To learn more about this amazing technology is by far yet the single most important factor of motivation.

1.5 Approach

Deep Max-Pooling Convolutional Neural Networks are Deep Neural Networks (DNN) with convolutional and max-pooling layers. They were first successfully applied to relatively small tasks such as digit recognition, image interpretation [3] and object recognition. Back then their size was greatly limited by the low computational power of available hardware. A decade later, however, DNN have greatly profited from Graphics Processing Units (GPU). Simple GPU-based multilayer perceptrons established new state of the art results on the MNIST handwritten digit dataset when made both deep and large. A year later, we saw the first implementation of GPU-based DNN on the CUDA parallel computing platform. This yielded new benchmark records on multiple object detection tasks. The field of Deep Learning with Neural Networks exploded. Multi-Column DNN [1] improved previous results by over 30% on many benchmarks including: handwritten digits (MNIST) [2]. Another flavor of DNN greatly improved the accuracy on a subset of ImageNet. Recently, Google parallelized a large DNN on a cluster with over 10000 CPU cores [4]. For image classification, the DNN returns a vector of class posterior probabilities when provided with an input patch whose fixed width and height usually does not exceed a few hundreds of pixels and depends on the network architecture. But DNN also excel at image segmentation and object detection. For segmentation, image data within a square patch of odd size is used to determine the class of its central pixel. The network is trained on patches extracted from a set of images with ground truth segmentations (i.e. the class of each pixel is known). To segment an unseen image, the trained net is used to classify all of its pixels. Object detection within an image is trivially cast as a segmentation problem: pixels close to the centroid of each object are classified differently from background pixels. Once an unseen image is segmented, the centroid of each detected object is determined using simple image processing techniques. Solving segmentation and detection tasks requires applying the network to every patch

contained in the image, which is prohibitively expensive when implemented in the naive, straightforward way.

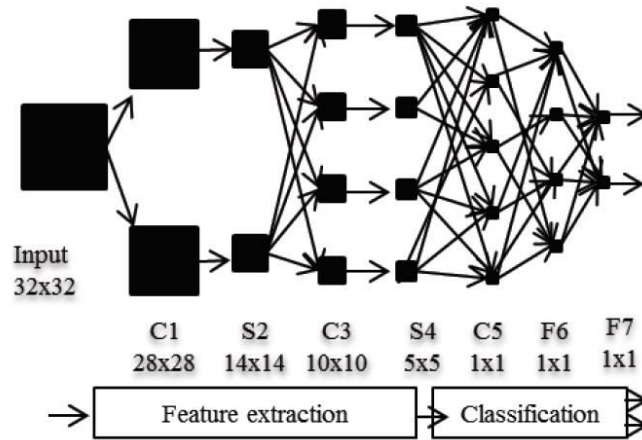


Fig.1.1 Convolutional neural network with eight layers

As shown in Fig1.1, the prefix “C”, “S” and “F” stands for convolution, sub sampling and classifier layers respectively. Layers C1 to C5 and C5 to F7 constitute the frontal feature extraction and classifier parts respectively. The proposed work represents the depicted architecture with a string of 1-2-2-4-4-5-4-2 where each number denotes the count of feature maps in that layer. Consider a net with a convolutional layer immediately above the input layer: when evaluating the first patch contained in the input image, the patch is convolved with a large number of kernels to compute the output maps; when evaluating the next patch, such convolutions are re-evaluated – a huge amount of redundant computation. It is better to compute each convolution only once for the whole input image: the resulting set of images contains the maps for each patch contained in the input image. In the particular case of a CNN without max-pooling layers, this optimization is trivially implemented by computing all convolutions in the first layer on the entire input image, then computing all convolutions in subsequent layers on the resulting extended maps. This approach yielded real time detection performance when combined with dedicated FPGA or even ASIC integrated circuits. However, present DNN owe much of their power to max-pooling layers interleaved with convolutional layers. Max-pooling cannot be handled using the straightforward approach outlined above. For example, when we perform a 2×2 max-pooling operation on an extended map, we obtain a smaller extended map which does not contain information from all the patches contained in the input image; instead, only patches whose upper left corner lies at even coordinates of the original image are represented. Any subsequent max-pooling layer

would further aggravate the problem. Our contribution consists in an optimized forward-propagation approach which avoids such problems by fragmenting the extended maps resulting from each max-pooling layer, such that each fragment contains information independent of other fragments, and the union of all fragments contains information about all the patches in the input image. A similar approach was previously used for handling a single sub sampling layer in a simple CNN for digit detection. Our mechanism, however, is completely general. It handles arbitrary architectures mixing convolutional and max-pooling layers in any order, and ensures that no redundant computation is performed at any stage.

1.6 Assumptions

1. Internet dependency is taken care of since MNIST needs to be loaded.
2. Client should have TensorFlow and Matlab installed.
3. Python environment should be there at the host PC.

1.7 Important Outcomes

1. Knowledge about Deep Neural Network
2. Knowledge about TensorFlow
3. Knowledge about Matlab
4. Knowledge about Python Programming
5. Critical comparative knowledge about both TensorFlow and Matlab

CHAPTER 2

LITERATURE REVIEW

People effortlessly recognize digits when they see handwritten images of digits. In each hemisphere of our brain, humans have a primary visual cortex, also known as V1, containing 140 million neurons, with tens of billions of connections between them. And yet human vision involves not just V1, but an entire series of visual cortices - V2, V3, V4, and V5 - doing progressively more complex image processing. CNN has been proved to be powerful tool for image classification and object detection.

Great strides have been achieved in pattern recognition in recent years. Particularly striking results have been attained in the area of handwritten digit recognition [5]. This rapid progress has resulted from a combination of a number of developments including the proliferation of powerful, inexpensive computers, the invention of new algorithms that take advantage of these computers, and the availability of large databases of characters that can be used for training and testing. At AT&T Bell Laboratories we have developed a suite of classifier algorithms. In this project we contrast the relative merits of each of the algorithms on two different platforms. In addition to accuracy, we look at measures that affect implementation, such as training time, run time, and memory requirements.

Good old on-line back-propagation for plain multi-layer perceptrons yields a very low 0.35% error rate on the famous MNIST handwritten digits benchmark. All we need to achieve this best result so far are many hidden layers, many neurons per layer, numerous deformed training images, and graphics cards to greatly speed up learning.

Handwriting recognition is of great academic and commercial interest. Current algorithms are already pretty good at learning to recognize handwritten digits. Post offices use them to sort letters; banks use them to read personal checks. MNIST [6] is the most widely used benchmark for isolated handwritten digit recognition. More than a decade ago, artificial neural networks called Multilayer Perceptrons or MLPs were among the first classifiers tested on MNIST. Most had few layers or few artificial neurons (units) per layer, but apparently back then they were the biggest feasible MLPs, trained when CPU cores were at least 20 times slower than today. A more recent MLP with a single hidden layer of 800 units achieved 0.70% error. However, more complex methods listed on the MNIST web page always seemed

to outperform MLPs, and the general trend went towards more and more complex variants of Support Vector Machines or SVMs and combinations of NNs and SVMs etc. Convolutional neural networks (CNNs) achieved a record-breaking 0.40% error rate, using novel elastic training image deformations. Recent methods pre-train each hidden CNN layer one by one in an unsupervised fashion, then use supervised learning to achieve 0.39% error rate. The biggest MLP so far also was pre-trained without supervision then piped its output into another classifier to achieve an error of 1% without domain-specific knowledge. Are all these complexifications of plain MLPs really necessary? Can't one simply train really big plain MLPs on MNIST? Why is there no literature on this? One reason is that at first glance deep MLPs do not seem to work better than shallow networks. Training them is hard as back-propagated gradients quickly vanish exponentially in the number of layers, just like in the first recurrent neural networks. Indeed, previous deep networks successfully trained with back-propagation either had few free parameters due to weight-sharing or used unsupervised, layer-wise pre-training. But is it really true that deep BP-MLPs do not work at all, or do they just need more training time? How to test this? Unfortunately, on-line BP for hundreds/thousands of epochs on large MLPs may take weeks or months on standard serial computers. But can't one parallelize it? Well, on computer clusters this is hard due to communication latencies between individual computers. Multi-threading on a multi-core processor is not easy either. We may speed up BP using SSE (Streaming Single Instruction, Multiple Data Extensions), either manually, or by setting appropriate compiler flags. The maximum theoretical speedup under single precision floating-point, however, is four, which is not enough. And MNIST is large - its 60,000 images take almost 50MB, too much to fit in the L2/L3 cache of any current processor. This requires continually accessing data in considerably slower RAM. To summarize, currently it is next to impossible to train big MLPs on CPUs. We will show how to overcome all these problems by training large, deep MLPs on MATLAB and Tensorflow.

RESEARCH GAP

Many modern statistical questions are plagued with asymptotic regimes that separate our current theoretical understanding with what is possible given finite computational and sample resources. Two important examples of such gaps appear in sparse inference and high-dimensional non-convex optimisation. In the former, proximal splitting algorithms efficiently solve the ℓ_1 -relaxed sparse coding problem, but their performance is typically evaluated in terms of asymptotic convergence rates. In the latter, a major challenge is to explain the excellent empirical performance of stochastic gradient descent when training large neural networks.

Here we illustrate how deep architectures can be used in order to attack such gaps. We will first see how a neural network sparse coding model can be analyzed in terms of a particular matrix factorization of the dictionary, which leverages diagonalisation with invariance of the ℓ_1 ball, revealing a phase transition that is consistent with numerical experiments. We will then discuss the loss surface of half-rectified neural networks. Despite defining a non-convex objective, we will show that by increasing the size of the model, one can again trade-off complexity with computation, in the sense that the landscape becomes asymptotically free of poor local minima. The DNN aims to compress information in one of its layers, known as bottleneck (BN) layer, which is used to obtain a new frame representation of the audio signal. This representation has been proven to be useful for the task of language identification (LID). Thus, bottleneck features are used as input to the language recognition system, instead of a classical parameterization of the signal based on cepstral feature vectors such as MFCCs. Despite the success of this approach in language recognition, there is a lack of studies analyzing in a systematic way how the topology of the DNN influences the performance of bottleneck feature-based language recognition systems. In this work, we try to fill-in this gap, analyzing language recognition results with different topologies for the DNN used to extract the bottleneck features, comparing them and against a reference system based on a more classical cepstral representation of the input signal with a total variability model. This way, we obtain useful knowledge about how the DNN configuration influences bottleneck feature-based language recognition systems performance.

OBJECTIVE

The project will be done to obtain following objective:

1. To develop system that can recognize handwritten digits using CNN.
2. Train the developed model in both TensorFlow and MATLAB platforms.
3. Compare accuracy and performance of the model at the platforms implemented.

DESIGN AND IMPLEMENTATION

Fully Connected network for even 28 x 28 images is computationally intensive task. One solution may be creating neural network where each hidden unit is connected to small subset of input units. Images also have property of being stationary meaning patches of image repeat. Using Convolutional Neural Network, fewer features can be used for classification and digit recognition.

5.1 Defining and Training CNN

Convolutional Neural Networks (CNN) are family of Deep Neural Networks and they consists of one or more convolution layers. CNN consists of many layers. Convolutional layers consist of a rectangular grid of neurons. It requires that the previous layer also be a rectangular grid of neurons. Each neuron takes inputs from a rectangular section of the previous layer; the weights for this rectangular section are the same for each neuron in the convolutional layer. Thus, the convolutional layer is just an image convolution of the previous layer, where the weights specify the convolution filter.

The pooling operation decreases dimensions of convolved image. It may be min-pooling, max-pooling or averaging. The pooling regions are very small. If pooling region is just 2x2 then this will have effect of sub sampling the output maps by a factor of 2 in both dimensions.

After reducing image to suitable feature-maps, fully connected MLP is used. This MLP consists of neurons that are connected to every neurons of both forward and backward direction as seen in Fig.5.3.

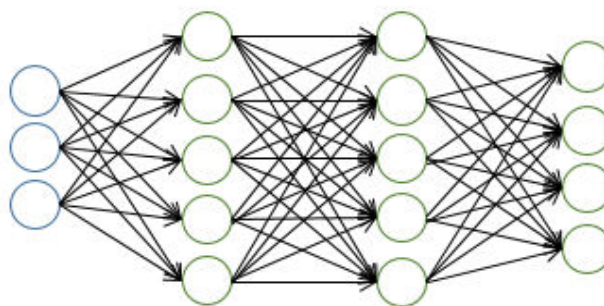


Fig.5.1 An example of Fully Connected MLP

CNN that we are going to build consists of input layer, convolutional layers, and subsampling layers as in figure below. Input layer is input image. After this layer there is hidden convolution layer. Input image is convolved with 6 different convolution filter and we get six feature maps of the same image. These 6 feature maps are then average-pooled and down-sampled at next layer. The down-sampled feature map is again convolved with convolution filter to get 12 other feature maps. These are again sub-sampled. At final hidden layer we get 12 feature maps of image. These feature maps are representation of images with very low dimensions. This final feature map is then fully connected with output classes. After defining structure of CNN and initializing different parameters, CNN can be trained with dataset. It is supervised case as there is large example and we are teaching our model to classify.

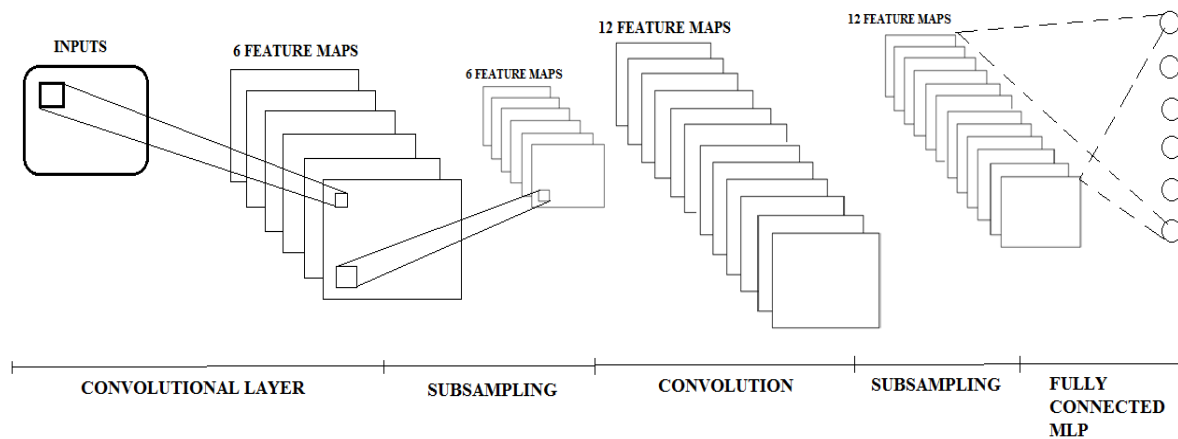


Fig.5.2 Architecture of CNN

5.2 Model Development

To develop the system, we have to follow steps as depicted in following flow diagram of project methodology. Each step is further described in separate headings.

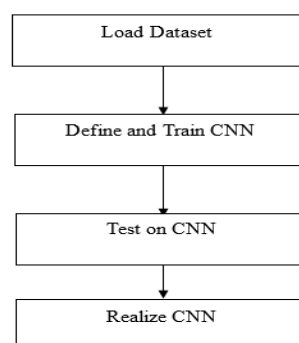
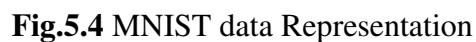


Fig.5.3 Project Methodology

Vast amount of data is already available. We have used MNIST Dataset. It is collection of images of handwritten digits. It has a training set of 55,000 examples, and a test set of 10,000 examples, and 5000 sets of validations. It is a subset of a larger set available from NIST. The digits have been size-normalized and centred in a fixed-size image. Each image is 28 pixels by 28 pixels. We can interpret this as a big array of numbers:



5.4 Using TensorFlow

Tensorflow is an open source library created by the Google Brain Trust for heavy computational work, geared towards machine learning and deep learning tasks. It is built on C, C++ making its computations very fast while it is available for use via a Python, C++, Haskell, Java and Go API. It created data graph flows for each model, where a graph consists of two units – a tensor and a node.

- 18

A data graph flow essentially maps the flow of information via the interchange between these two components. Once this graph is complete, the model is executed and the output is computed.

5.4.2 Methodology

1. We have built feedforward neural network using softmax to predict the number in each image. We begin by calling in a Python environment.
2. Initializing parameters for the model. In machine learning, an epoch is a full iteration over samples. Here, we are restricting the model to 30 complete epochs or cycles of the algorithm running through the dataset.
3. The batch variable determines the amount of data being fed to the algorithm at any given time, in this case, 100 images.

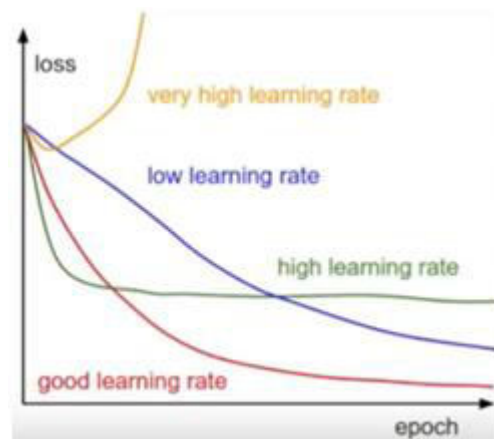


Fig.5.5 Learning Efficiency plot

4. The learning rate controls the size of the parameters and rates, thereby affecting the rate at which the model “learns” which is taken as 0.01, a number well known for this problem.
5. Here, W is the weight and b is the bias of the model. They are initialized with tf. variable as they are components of the computational graph that need to change values with the input of each different neuron.

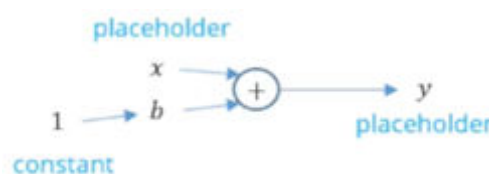


Fig.5.6 Operations on placeholders

6. Initializing the model. We will be using a simple softmax model to implement our network.
Softmax is a generalization of logistic regression, usually used in the final layer of a network. It is useful because it helps in multi-classification models where a given output can be a list of many different things. It provides values between 0 to 1, that in addition give you the probability of the output belonging to a particular class.
7. Defining Cost Function
8. Determining the accuracy of parameters.
9. Implementing Gradient Descent Algorithm. The gradient descent algorithm starts with an initial value and keeps updating the value till the cost function reaches the global minimum i.e. the highest level of accuracy. This is obviously dependent upon the number of iterations being permitted for the model.

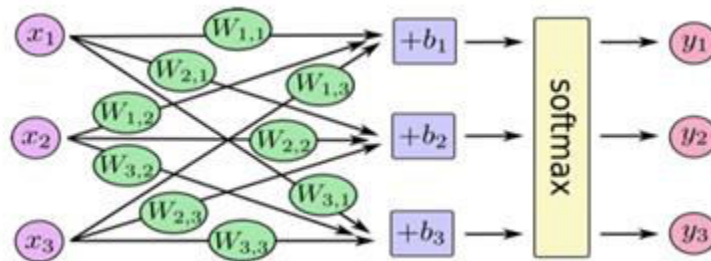


Fig.5.7 Softmax function visualization

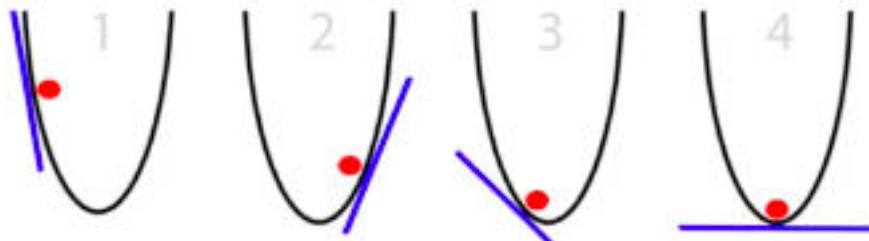


Fig.5.8 Gradient Descent Algorithm

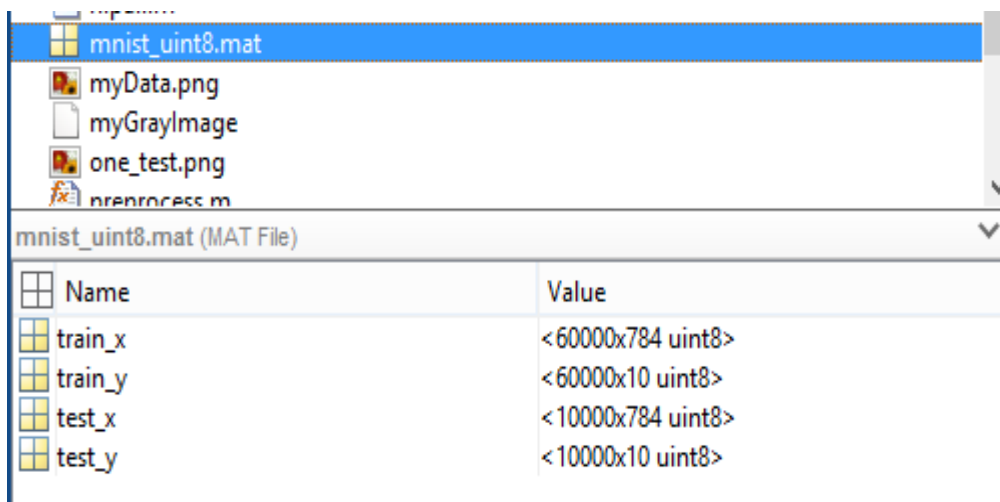
10. Initializing the session.
11. Creating batches of data for epochs.
12. Executing the model.
13. Print accuracy of the model

5.5 Using MATLAB

To speed up training on large data sets, you can distribute computations and data across multi-core processors and GPUs on the desktop, or scale up to clusters and clouds, including with MATLAB Distributed Computing Server.

5.5.1 Loading Dataset

Initial step to precede the project is to load data to current context. Neural network or any machine learning tool cannot give desirable output unless it is provided with huge set of examples. For our project we have MNIST Dataset.



The image shows the MATLAB Workspace window. At the top, a list of files is displayed: `mnist_uint8.mat` (selected), `myData.png`, `myGrayImage`, `one_test.png`, and `nnprocess.m`. Below this, the `mnist_uint8.mat (MAT File)` is expanded, showing a table of variables loaded from the file.

Name	Value
<code>train_x</code>	<60000x784 uint8>
<code>train_y</code>	<60000x10 uint8>
<code>test_x</code>	<10000x784 uint8>
<code>test_y</code>	<10000x10 uint8>

Fig.5.9 MNIST Dataset in MATLAB Workspace

5.5.2 Testing CNN

Our dataset consists of 10,000 samples for testing CNN. After training, testing is done to see whether the model works fine or not. After training and testing images, CNN will be ready for use. Once we get this final CNN, we don't need training data any more. This CNN holds different weights and biases. Any digit can be fed to this CNN with suitable pre-processing.

TESTING AND RESULTS

6.1 Test Results on TensorFlow

The model is trained with set of total 60,000 images from MNIST. The MNIST data is split into three parts: 55,000 data points of training data, 10,000 points of test data, and 5,000 points of validation data.

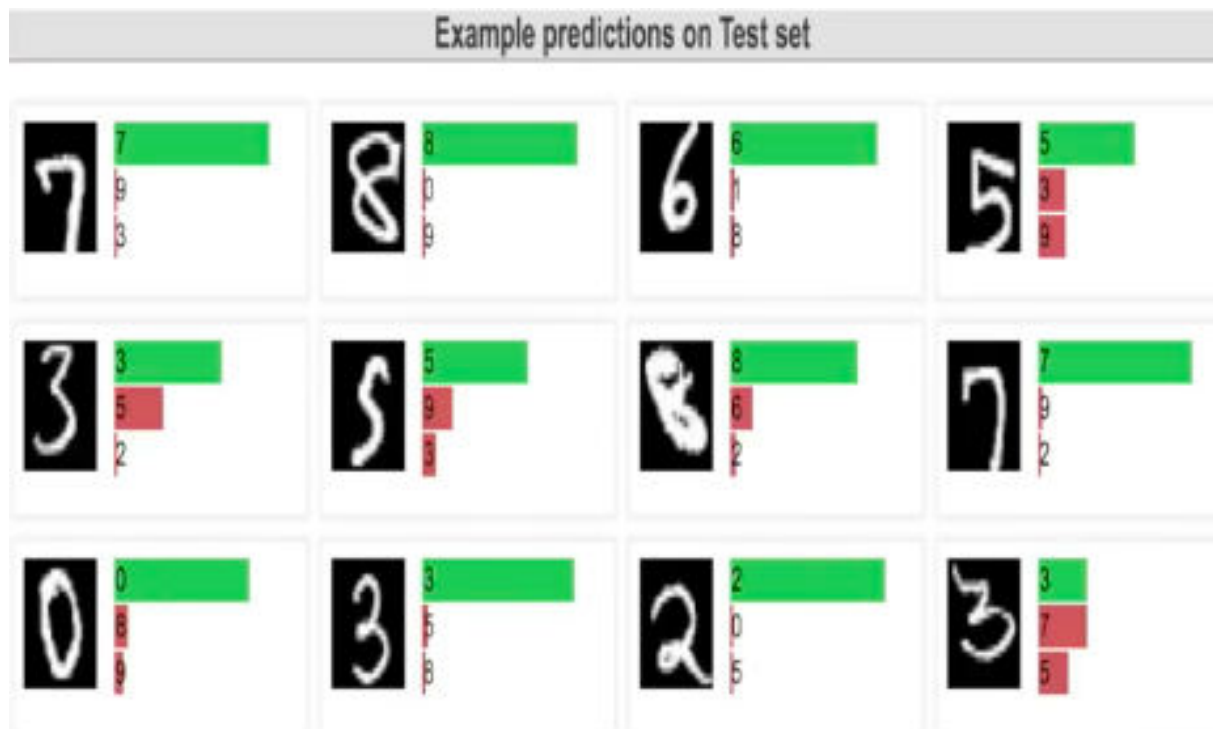


Fig.6.1 Predictions on Test Case

6.1.1 Training with Feed Forward

Model is trained with batch=100, learning rate=0.01 and epochs=30, time: ~1 min 35 sec

```
tensorflow_demo — python board2.py — 80x24
Instructions for updating:
Use `tf.global_variables_initializer` instead.
2017-12-19 00:36:58.048944: I tensorflow/core/platform/cpu_feature_guard.cc:137]
Your CPU supports instructions that this TensorFlow binary was not compiled to
use: SSE4.1 SSE4.2 AVX AVX2 FMA
WARNING:tensorflow:Passing a `GraphDef` to the SummaryWriter is deprecated. Pass
a `Graph` object instead, such as `sess.graph`.
Iteration: 0001 cost= 29.860465115
Iteration: 0003 cost= 21.120191600
Iteration: 0005 cost= 20.177400271
Iteration: 0007 cost= 19.676049789
Iteration: 0009 cost= 19.294817591
Iteration: 0011 cost= 19.055197762
Iteration: 0013 cost= 18.924092803
Iteration: 0015 cost= 18.721716655
Iteration: 0017 cost= 18.677906119
Iteration: 0019 cost= 18.517147991
Iteration: 0021 cost= 18.370799550
Iteration: 0023 cost= 18.317961532
Iteration: 0025 cost= 18.354472974
Iteration: 0027 cost= 18.212409182
Iteration: 0029 cost= 18.081464451
Tuning completed!
Accuracy: 0.9198
```

Fig.6.2 TensorFlow with epochs=30

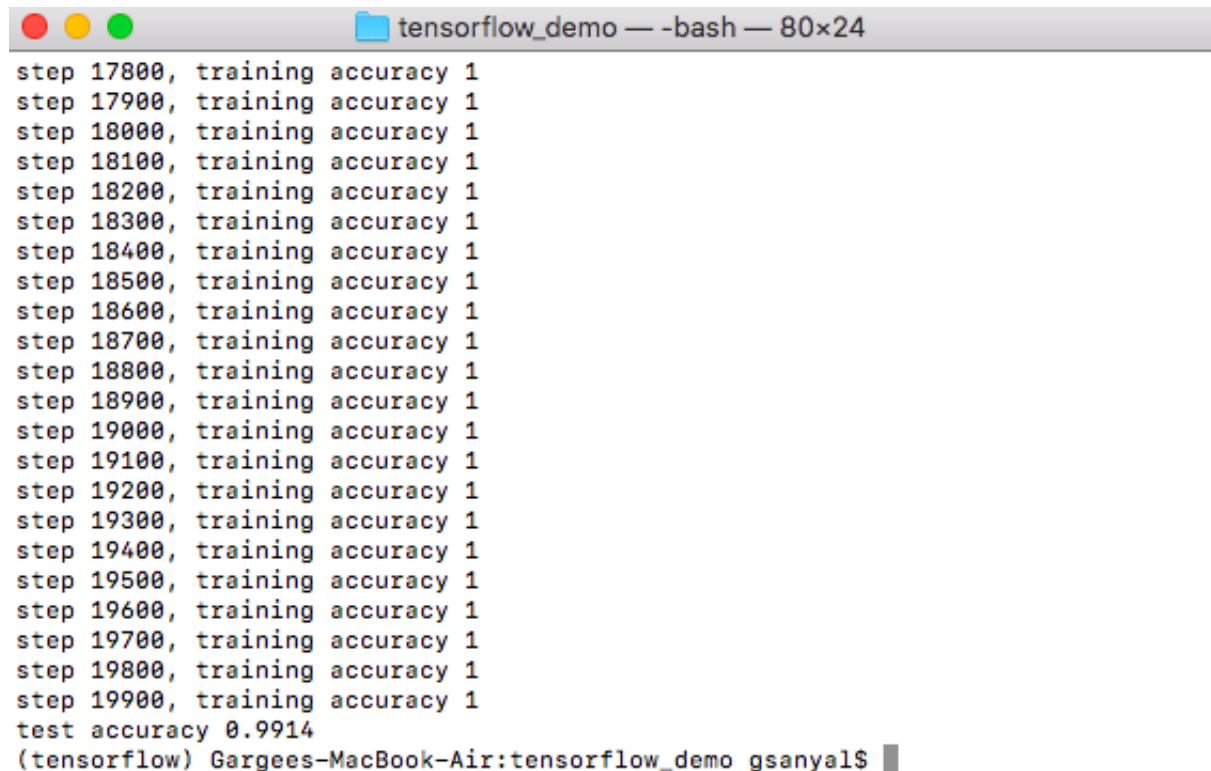
Model is trained with batch=100, learning rate=0.01 and epochs=100, time: ~4 min 31 sec

```
tensorflow_demo — python board2.py — 80x24
Iteration: 0057 cost= 17.561776311
Iteration: 0059 cost= 17.580189955
Iteration: 0061 cost= 17.589507287
Iteration: 0063 cost= 17.426853893
Iteration: 0065 cost= 17.478294438
Iteration: 0067 cost= 17.414584765
Iteration: 0069 cost= 17.357794401
Iteration: 0071 cost= 17.432868291
Iteration: 0073 cost= 17.340066389
Iteration: 0075 cost= 17.425906884
Iteration: 0077 cost= 17.341326496
Iteration: 0079 cost= 17.419234221
Iteration: 0081 cost= 17.336455854
Iteration: 0083 cost= 17.312537982
Iteration: 0085 cost= 17.284317357
Iteration: 0087 cost= 17.303003259
Iteration: 0089 cost= 17.308407476
Iteration: 0091 cost= 17.176572169
Iteration: 0093 cost= 17.291698253
Iteration: 0095 cost= 17.260598222
Iteration: 0097 cost= 17.151030630
Iteration: 0099 cost= 17.202918346
Tuning completed!
Accuracy: 0.9232
```

Fig.6.3 TensorFlow with epochs=100

6.1.2 Training with CNN

Model is trained with batch=50, (ADM) learning rate=1e-4 and epochs=20,000, time: ~1 hour 2 min 49 sec.

A screenshot of a macOS terminal window titled 'tensorflow_demo --bash-- 80x24'. The window displays the output of a TensorFlow CNN training process. It shows a series of lines for steps 17800 through 19900, each reporting 'training accuracy 1'. After step 19900, it reports 'test accuracy 0.9914'. The prompt '(tensorflow) Gargees-MacBook-Air:tensorflow_demo gsanyal\$' is visible at the bottom.

```
step 17800, training accuracy 1
step 17900, training accuracy 1
step 18000, training accuracy 1
step 18100, training accuracy 1
step 18200, training accuracy 1
step 18300, training accuracy 1
step 18400, training accuracy 1
step 18500, training accuracy 1
step 18600, training accuracy 1
step 18700, training accuracy 1
step 18800, training accuracy 1
step 18900, training accuracy 1
step 19000, training accuracy 1
step 19100, training accuracy 1
step 19200, training accuracy 1
step 19300, training accuracy 1
step 19400, training accuracy 1
step 19500, training accuracy 1
step 19600, training accuracy 1
step 19700, training accuracy 1
step 19800, training accuracy 1
step 19900, training accuracy 1
test accuracy 0.9914
(tensorflow) Gargees-MacBook-Air:tensorflow_demo gsanyal$
```

Fig.6.4 TensorFlow CNN with epochs=20,000

With the simple feed forward in although it took less time, but the desired accuracy wasn't obtained even after increasing the epochs. But in CNN, the accuracy reached ~99.2 per cent which is a pretty descent outcome.

NOTE: The system used has Processor: 5th Gen Intel Core i5 1.8 GHz, Memory: 8 GB 1600 MHz DDR3, Disk: Macintosh SSD

6.2 Test Results on Matlab

6.2.1 Training and Testing CNN with one epoch

CNN is trained with batch size 50, learning parameter 1 and number of epochs 1. The output obtained is as follows


```
Command Window

>> cnn
Undefined function or variable 'cnn'.

>> cnn
Undefined function or variable 'cnn'.

>> test_example_CNN
Training the CNN...
epoch 1/1
Elapsed time is 106.404200 seconds.
...Done. Training took 106.47 seconds
Evaluating test set...
Accuracy: 88.87%
fx >>
```

Fig.6.5 Training CNN for 1 epoch

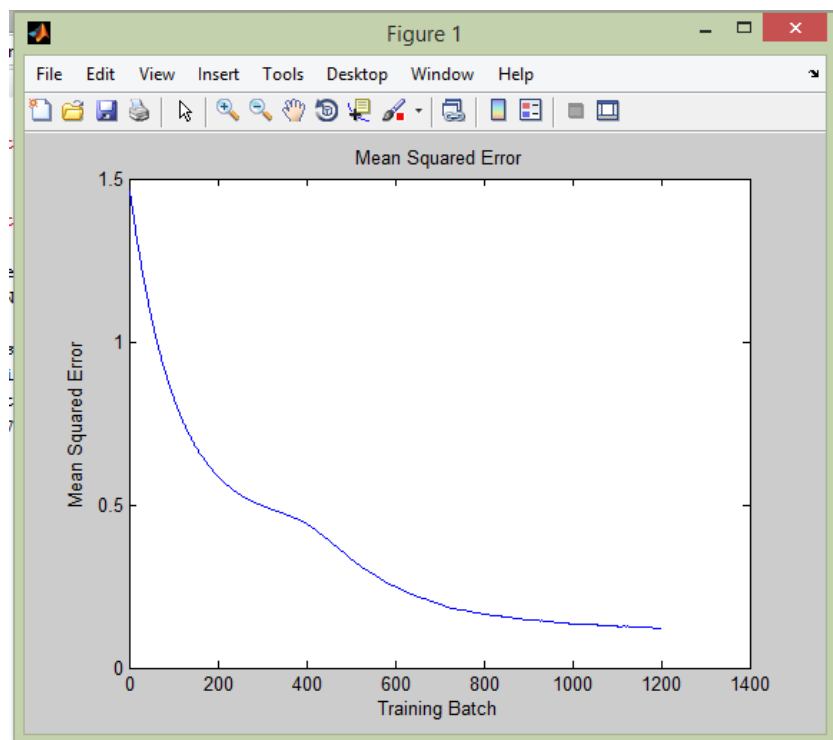


Fig.6.6 Plot of Mean Squared Errors and Batches

Time taken for one epoch is 106.47 seconds. CNN is then tested over test dataset. CNN gave accuracy of 88.87 %. It is quite high accuracy though for the first epoch.

6.3 Training and Testing with one hundred epochs

Under similar conditions, but the number of epochs is changed to 100 to train the model. It took 3.065 hours to train on our computer [Intel® Core™ i5-4210U CPU @1.70GHz(4 CPUs), ~2.4 GHz and 4 GB DDR3 RAM]. The accuracy on test data set became 98.92 %.

This is very impressive. It becomes very computationally intensive if we increase further epochs without getting more accuracy.

```
epoch 98/100
Elapsed time is 110.238234 seconds.
epoch 99/100
Elapsed time is 108.683504 seconds.
epoch 100/100
Elapsed time is 108.586642 seconds.
...Done. Training took 11104.81 seconds
Evaluating test set...
Accuracy: 98.92%
```

Fig.6.7 MATLAB Command View after 100 Epochs

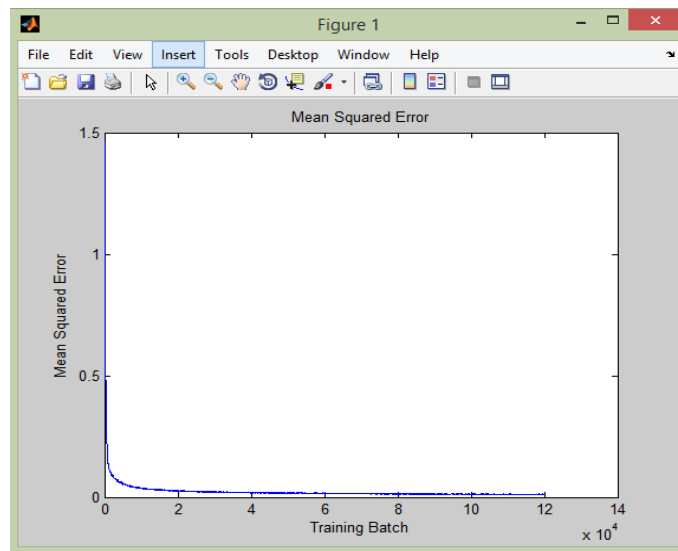


Fig.6.8 Mean Squared Error for 100 Epochs

6.4 Running CNN for our Image

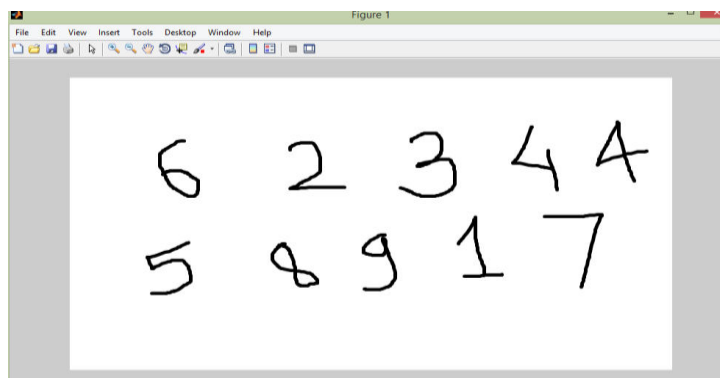


Fig.6.9 Digits to be recognized

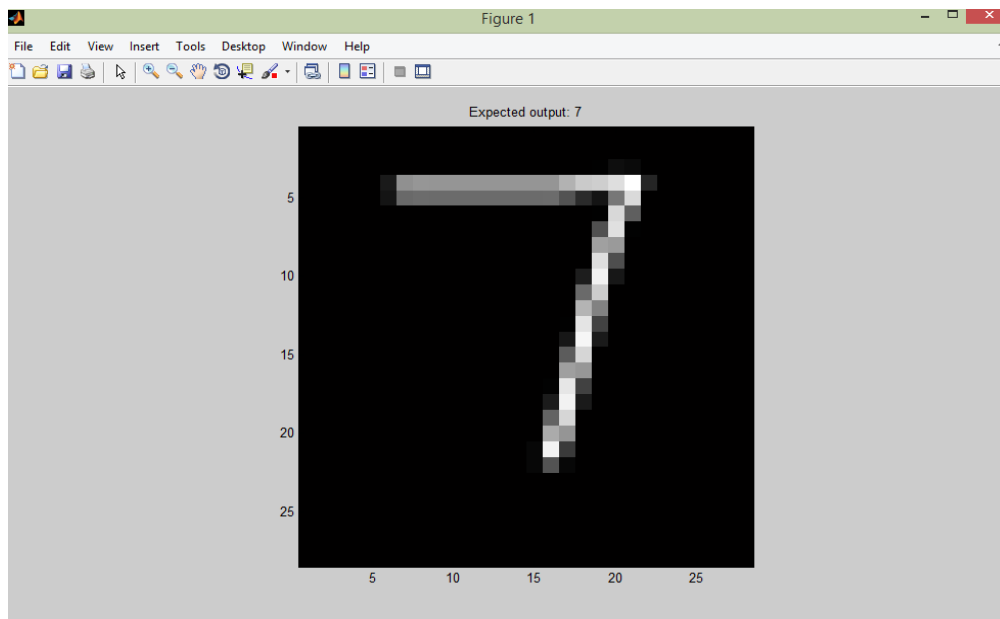


Fig.6.10 CNN correctly classifying handwritten digit ‘seven’

CONCLUSION

Our human mind can identify a number within fractions of nano-second. But for a computer to recognize and write code for it becomes the difficult part. Deep neural networks mimic the way human nervous system works, thus making the problems which seem very difficult to code, easier. The need for such deep neural networks is ever increasing as these architectures learn themselves, saving time and effort. This project is a small step towards that. We have critically measured the performances of feed forward and convolutional neural networks in two different platforms, one being TensorFlow, and the other being MATLAB. We concluded that while TensorFlow has significant performance advantages, MATLAB beats TensorFlow in terms of showing results in a GUI.

REFERENCES

- [1] Dan Claudiu Ciresan, Ueli Meier, and Jurgen Schmidhuber, "Multi-column deep neural networks for image classification," in Computer Vision and Pattern Recognition, 2012, pp. 3642– 3649.
- [2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradientbased learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, November 1998.
- [3] Sven Behnke, Hierarchical Neural Networks for Image Interpretation, vol. 2766 of Lecture Notes in Computer Science, Springer, 2003.
- [4] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc Le, Mark W. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng, "Large scale distributed deep networks," in Neural Information Processing Systems, 2012.
- [5] M. D. Garriss et al: NIST Form-Based Handprint Recognition System. Internal Report National Institute of Standards and Technology NISTIR 5469, 1994.
- [6] K. Turner and J. Ghosh: Theoretical Foundations of Linear and Order Statistics Combiners for Neural Pattern Classifiers. TR-95-02-98, The Computer and Vision Research Center, The University of Texas at Austin, 1995.
- [7] Netzer, Y., et al. Reading Digits in Natural Images with Unsupervised Feature Learning. NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.
- [8] Goodfellow, I., et al. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. ICLR, 2014.
- [9] Goodfellow, I., Bengio, Y. and Courville, A. Deep Learning. MIT Press, 2016.

SELF ASSESSMENT REPORT

Feedback on Student Outcomes (EAC-Computer Engineering):

Student Outcomes	Response*
a) An ability to apply knowledge of Mathematics, Science, and Engineering	4
b) An ability to design and conduct experiments, as well as to analyze and interpret data	4
c) An ability to design a system, component, or process to meet desired needs within realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability	4
d) An ability to function on multidisciplinary teams	5
e) An ability to identify, formulate, and solve engineering problems	3
f) An understanding of professional and ethical responsibility	4
g) An ability to communicate effectively	5
h) The broad education necessary to understand the impact of engineering solutions in a global, economic, environmental, and societal context	4
i) A recognition of the need for, and an ability to engage in life-long learning	4
j) A knowledge of contemporary issues	4
k) An ability to use the techniques, skills, and modern engineering tools necessary for engineering practice	4

* Please respond with 1, 2, 3, 4 or 5, where 1 – not met and 5 – fully met.

Feedback on Student Outcomes (CAC-Computer Science):

Student Outcomes	Response*
a) An ability to apply knowledge of Computing and Mathematics appropriate to the program's student outcomes and to the discipline	4
b) An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution	4
c) An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs	3
d) An ability to function effectively on teams to accomplish a common goal	5
e) An understanding of professional, ethical, legal, security and social issues and responsibilities	4
f) An ability to communicate effectively with a range of audiences	4
g) An ability to analyze the local and global impact of computing on individuals, organizations, and society	5
h) Recognition of the need for and an ability to engage in continuing professional development	4
i) An ability to use current techniques, skills, and tools necessary for computing practice	4
j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices	4
k) An ability to apply design and development principles in the construction of software systems of varying complexity	4

* Please respond with 1, 2, 3, 4 or 5, where 1 – not met and 5 – fully met.