Handwritten Digit Recognition

Ananda Kishore Sirivella, UFID: 9951-5080
Department of Computer Science, University of Florida,
Gainesville, USA, Email: asirivella@ufl.edu

ABSTRACT

Human brain is one of the wonders in the world. The human visual system is one such complex system. Problems like handwritten digit recognition is a piece of cake to the human eye. But when it comes to developing a visual system to recognize the same, we would recognize high complexity of the problem. The problem of handwritten digits went hidden for ages, until the new automation processes arrived and better system were developed. For example, the general forms had postal code which were always manually read and updated to the records, but with the automation of the system, the computer need to read the image of the form and recognize the digits on the postal code. We would try to develop one such system for handwritten digit recognition using machine learning algorithms. In this project, we shall choose Deep learning and its convolutional neural network concepts to build an effective system. We would utilize the MNIST dataset of digits to train and test the system. In the project, we be going through the motivation of using these algorithms or the basic science behind them, working of the aforementioned algorithm, tune the system for better accuracy. Towards the end of the project, I aim to develop a Visual system for handwritten digits which is 95% or more accurate.

INTRODUCTION

Human brain is one of the wonders in the world. Every day activities like reading, playing, thinking or moving a simple limb is a simple task to the brain. Over time, numerous computer scientist has tried to replicate the some of the activities of the human brain and failed miserably. Till date, the biggest question in the computer world is building of an Artificial intelligence that could substitute at least some of the simple functionalities of the human brain.

Every day, we come across a new development in the field of AI, could be amazon Alexa or google echo. The world is thriving for developing a functional AI. Building a AI similar to the human brain is a sophisticated task involving multiple sub-tasks which are equally tough, like speech recognition, pattern recognition, image recognition, text recognition, etc.

One such complex task is recognition of an image. Every day phenomenon of looking at an image and recognizing the traits and characteristics of a marvelous wonder. The human image processing is a complex problem. The Below image is one of the example to recognize the image recognition complexity.

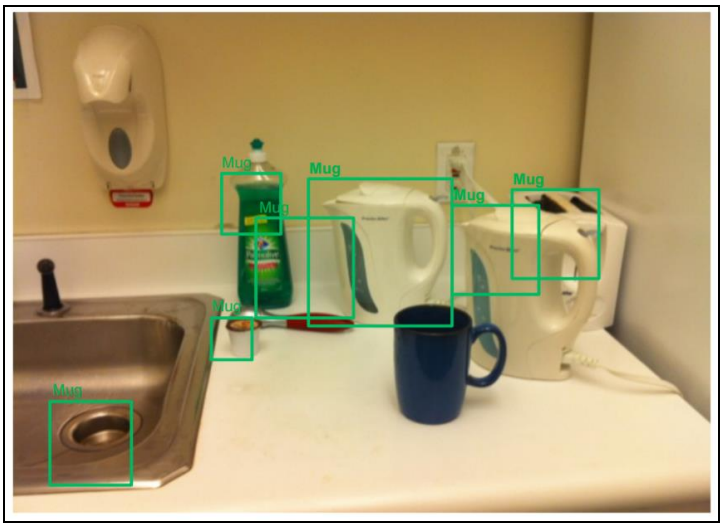


Figure 1: Identifying a coffee mug

Figure 1, where an Artificial intelligence is used to recognize the mug. As you can see, the computer is not able to recognize the mug as a mug. That is because, all that machine sees is simple number 0 to 255 numbers. Figure 2 indicates, all that the machine can see. A human eye would recognize the image with blue coffee mug. We will explain more on how a computer vision would recognize such problems.

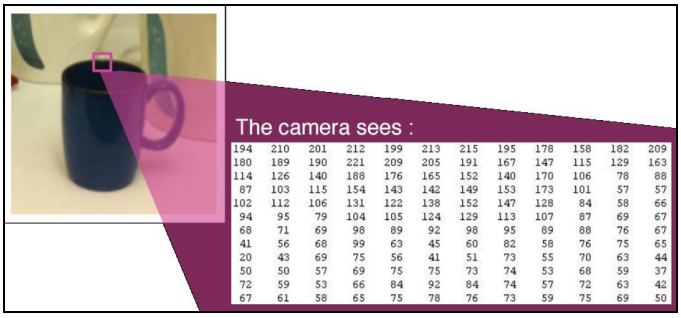


Figure 2: What a machine sees!

In order to solve the image recognition problem, the algorithm need to identify and understand each of the featurea and learn about the traits of each object that had to be recognised. Figure 3, represents the simple flowchart to improvise a machine learning

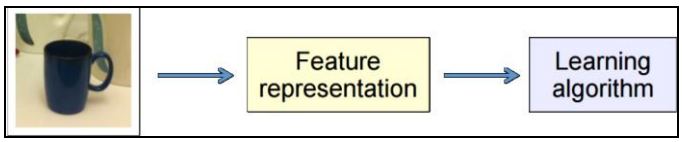


Figure 3: Image recognition

This image recognition is recognized as a field of computer vision. Computer vision is a field that deals with computers gaining high level understanding from digital images or videos. The computer vision involves tasks of processing, acquiring, understanding, analyzing the digital images and extraction of high dimensional data for numerical information. Computer vision is a pioneer in Artificial intelligence. We would be digging more into some of the algorithms to implements machine learning models to create an efficient recognition model and compare its results.

PROPOSED SOLUTION

Machine learning is a well proposed solution to such problem. Machine learning being latest science to such algorithms, where algorithms are developed to learn from the data and re-construct is model to provide better accuracy. Machine learning mainly consists of supervised learning and unsupervised learning. Supervise learning, where the algorithm is trained & constructed using a train set and verified with a test for its calibration. Unsupervised learning, where the algorithm is trained by itself & constructs itself with recursive trial of its constructed model and its calibrations. These being two classes of learning algorithms. We have more specific algorithms under each class of learning to help us. Nave Bayes classifier & K-Nearest neighbor are a good example of supervised learning, clustering being a good example of unsupervised learning. We would utilize the supervise learning version of Deep learning to implement the image classification problem.

Unlike most learning algorithm whose learning get saturated over time, Deep learning is a constant learning algorithm, where the performance is enhanced with more data processed. Deep learning is an algorithm which utilizes the neural network concept of human brain in creating multiple layers of learning to classify and categorize the data. Each one of the layers act as individual feature extraction and work on the transformation of the features. Output of each layer serves as input to the next layer to work on. Some layers are even more complexly related by sharing weights, feature mapping across the layers to enhance the learning of the deep learning network. In our case, we shall be using Deep learning networks of three types:- basic-model using Multi-Layer Perceptron, a simpler version of convolutional Nueral network (CNN) & bigger version of implementation CNN. We shall Look deeper into each kind of networks for better understanding.

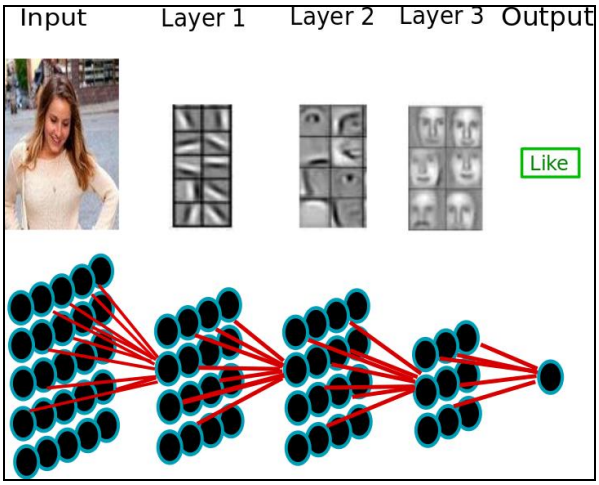


Figure 4: A Deep learning example

MNIST DATASET

The MNIST dataset is a modified and smaller version of humongous set available from NIST. The MNIST is specially devised to help with comparison of the machine learning models. The MNIST was developed by Yann LeCun, Corinna Cortes and Christopher Burges. The source of MNIST, NIST is basically a huge collection of image constructed from numerous number of scanned documents. MNIST dataset is a subset of images picked from MNIST, the images are normalized in size and centered before being populated to the dataset. MNIST is designed so that developer could spend less time in evaluating machine learning model rather than spending time on data cleaning or preparation of data.

Each of the MNIST contains images of size 28 by 28 pixels (784 pixels in total). A standard split of dataset would be 60,000 samples into training set and 10,000 samples into test set to test. The dataset is designed for digit recognition task, to predict digits from 0 to 9. We would be focusing on the Deep learning concepts in building a better solution to the MNIST classification with an accuracy of more than 98%.

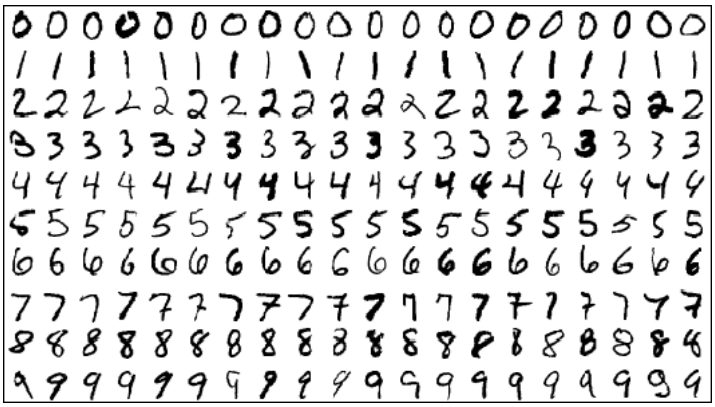


Figure 5: Some of the image from the MNIST dataset

MULTI-LAYER PERCEPTRON

Multi-Layer Perceptron (MLP) is one of the simplest form of Deep learning networks. Multi-Layer Perceptron generally contains an input layer & output layer with one or more hidden layers for learning. Multi-Layer Perceptron uses a supervised learning model named backpropagation, to pass on the learnt data at each layer to the next. Here, each layer is connected ot the next & consists of multiple node, with each node working as a neuron preceptor and processes each data packet to enhance the result much more. MLP utilizes the modified version of linear perceptron and is able to separate the non-linear perception of the data too. We will further dig deep into the concept of MLP.

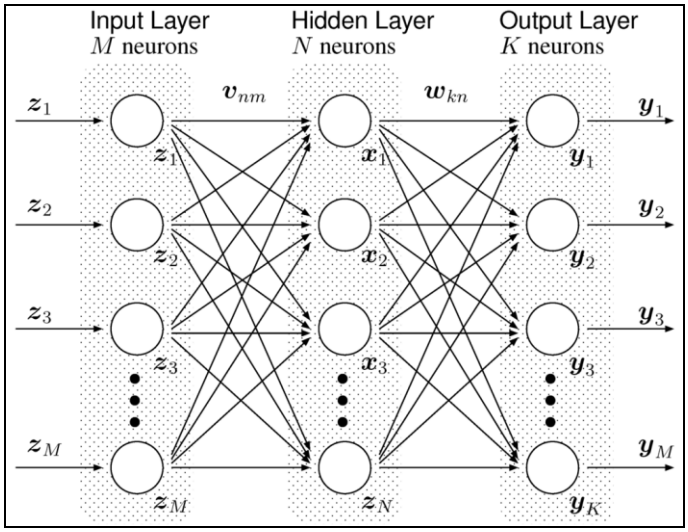


Figure 6: A Multi-Layer Perceptron

In general, MLP contains multiple layers of abstractions, but for our understanding we shall consider a one layer MLP to deduce the math behind it. A simple one layer MLP is mathematically defined as a function with

$$f : R^D \rightarrow R^L$$

Where,
D corresponds to the input vector (x) size
L corresponds to the output vector (f(x)) size.

The above function being a very generic form, we shall deduce a matrix verison to proceeed further with out discussion. The matrix verison of the function would be:

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x))),$$

Where,
$b^{(1)}$ & $b^{(2)}$ denotes the bias vectors
$W^{(1)}$ & $W^{(2)}$ denotes the weight matrices
With G & s as activation functions.

We define a vector h(x) to for each of the hidden layers. $h(x) = \Phi(x) = s(b^{(1)} + W^{(1)}x)$. $W^{(1)} \in R^{D \times D_h}$ a weight matrix for connecting to the hidden layer from the input vector. Each i-th node on hidden layer respresnets the mapping weights to the input layer $W_i^{(1)}$. The activation function ***s*** has than ***tanh***, where $tanh(a) = (e^a - e^{-a})/(e^a + e^{-a})$, or a sigmoid logistic funtions as $sigmoid(a) = 1/(1 + e^{-a})$. We can use ***tanh*** in deep learn network for faster training (for better local minima too) than ***sigmoid***. The output vector finally becomes:

$$o(x) = G(b^{(2)} + W^{(2)}h(x)).$$

Likely, the MLP has more parameters to be tunede like gradient descent. Some of them are:-

a) Non-Linearity:
Of the ***tanh*** and ***sigmoidal*** functions which are most common function to implement non-linearity. These functions would reproduce a zero-mean inputs for the further layers on the multi-layer perceptron. In our literature research, we found tanh functions does converge faster

b) Initialization of weights:-
With Initial weights chosen to be near zero and small, So that it helps activation function to work similar to a linear function in the initial stages. The network regularly conserves the variance in back propagation as well as variance in activation to the another layer from one another. With this allowed information learnt with data to flow back and forth between layers to minimize the discrepancies across the layers. We recognize a better way to initialize weight is by using:-

$uniform[-\frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}, \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}]$ for tanh and $uniform[-4 * \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}, 4 * \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}]$ for sigmoid.

Where,
$f^{an}{}_{in}$ refers to the number of inputs
$f^{an}{}_{out}$ refers to the number of hidden nodes in that layer.

c) Number of nodes in a Hidden Layers:-
The number of nodes is a parameter which is regulated over the size of the dataset on which the deep network operates on. With a highly complex dataset, we would have to spawn more number of nodes in hidden layer.
We would find an optimal number of hidden layer for each dataset using 100-1000 hidden layer nodes with a step over of 100 nodes.

d) Learning Rate:
Generally, Learning rate of a regular deep network is kept to be a contact value, irrelevant to the usage of gradient descent or any other improvement techniques. The learning rate could be tuned by moderating the learning rate over time and parallelly checking validation error, and reaching a constant value to maximize the learning rate. In our implementation, we are using a constant value for learning rate.

With the end of a layer, we had tried optimizing with a negative log likelihood to reduce the misclassification error. We would loop through the calibration to iterate until the misclassification error converges on to a value.

The simple implementation of one-layer multilayer perceptron model takes a few seconds to train and formulate the output on out MNIST dataset. With a 1.92% error rate, the one-layer deep network is quite time efficient and accurate when a huge regular dataset is to be processed in a short time. Below are the results with following configuration details: -

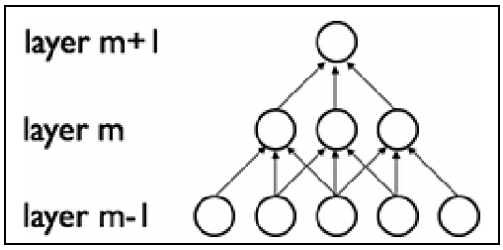
- ✓ Training Set :- 60,000
- ✓ Test Set :- 10,000
- ✓ Execution time:- 81 seconds
- ✓ Accuracy :- 98.02 %
- ✓ Error rate :- 1.92 %

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
9s - loss: 0.2761 - acc: 0.9217 - val_loss: 0.1294 - val_acc: 0.9610
Epoch 2/10
8s - loss: 0.1089 - acc: 0.9685 - val_loss: 0.0908 - val_acc: 0.9727
Epoch 3/10
8s - loss: 0.0700 - acc: 0.9787 - val_loss: 0.0784 - val_acc: 0.9764
Epoch 4/10
8s - loss: 0.0484 - acc: 0.9858 - val_loss: 0.0708 - val_acc: 0.9792
Epoch 5/10
8s - loss: 0.0364 - acc: 0.9897 - val_loss: 0.0667 - val_acc: 0.9797
Epoch 6/10
8s - loss: 0.0257 - acc: 0.9933 - val_loss: 0.0588 - val_acc: 0.9820
Epoch 7/10
8s - loss: 0.0191 - acc: 0.9952 - val_loss: 0.0689 - val_acc: 0.9781
Epoch 8/10
8s - loss: 0.0140 - acc: 0.9970 - val_loss: 0.0634 - val_acc: 0.9814
Epoch 9/10
8s - loss: 0.0103 - acc: 0.9978 - val_loss: 0.0640 - val_acc: 0.9822
Epoch 10/10
8s - loss: 0.0076 - acc: 0.9985 - val_loss: 0.0627 - val_acc: 0.9808
 Error: 1.92>> >%
```

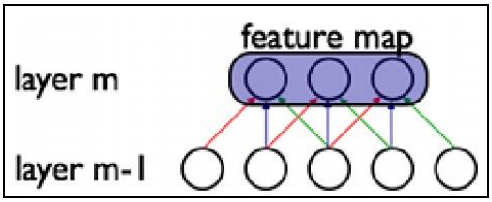
Figure 7: Results with Multi-Layer Perceptron

CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Network is a version of feed-forward artificial neural network, which is inspired on the biological networks (similar to that of animal neural network). Convolutional Neural Network is a popular implementation of deep learning network when handling with images. The CNN contains multiple layers of small neural network which represents the portion of train/test image pattern, these layers can be called as receptive fields. The output of these layers is feed forward to next layers which is then continued for every layer, this process is called tiling. This Tiling of CNNs helps in translation of images. The CNN uses spatial-local correlation to enforce a local connectivity across neurons in the adjacent layers. The input to hidden nodes in layer (m) in CNN is form a subset of hidden node of layer (m-1) so that spatially contiguous receptive fields. Below is figure that explains it better.



That how the CNN maintains a feature map across the adjacent layers. The weight across layer (m) and layer (m-1) is normalized to be same value, which is termed as weight sharing that helps us in improving the learning efficiency and reducing the count of free parameters that has to be learnt.



We would construct the feature map using CNN by applying a function recursively on sub regions of the whole image input pattern.i.e. With convolution of input image pattern by a linear filter in addition of a bias term and then applying a non-linearity function. Thus, the k-th feature map on the layer h^k, where filter are recognized by weights W^k and bias is b_k, the feature map thus obtained is as follows:

$$h_{ij}^k = \tanh((W^k * x)_{ij} + b_k)$$

Thus, the CNN is recognized to be efficient with the image pattern recognition. Below is the result of CNN over UCI Optical Handwritten Numbers on the flattened image features.

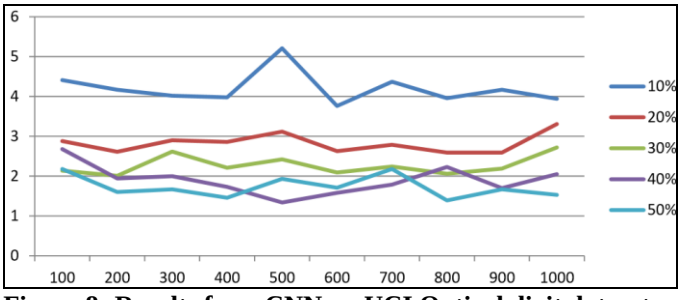


Figure 8: Results for a CNN on UCI Optical digit dataset.

A Simple Convolutional Neural Network

Convolutional neural networks are highly sophisticated than the regular multi-layer perceptron. We had start with implementation of a simple CNN. Below is a summary of the network design.

1. The initial hidden convolutional layer is known as Covolution2D. This layer utilized 32 feature maps of size 5 x 5 and an activation function for verification. The input layer awaits the image to be in the structure of pixels, width & height.

2. The next layer would be a pooling layer, which can be called as MaxPooling2D. The pooling size is configured to be 2 x 2.

3. Next would be a regularization layer with dropout called dropout layer. This layer is configured to randomly exclude 20% of neurons from the layer to counter the overfitting problem.

4. The further layer helps in processing the 2D matrix data on a vector called Flatten. This helps processing of output with the help of the fully connected layers.

5. The next layer is a fully connected version of the layer with 128 nuerons and activation function for validation.

6. At the end, the output layer is to output 10 classes. Hence, we utilize 10 nueron for 10 classess and a softmax activation function for probabilistic prediction for individual classes.

As always, the model is trained with ADAM gradient descent algorithm and logarithmic loss.

The simple implementation of Convolutional nueral network model takes a few seconds to train and formulate the output on out MNIST dataset. With a 1.00 % error rate, the simple deep network is efficient and accurate when a huge regular dataset is to be processed in a short time for a little compromise on accuracy. Below is the results with following configuration details:-

- ✓ Training Set :- 60,000
- ✓ Test Set :- 10,000
- ✓ Execution time:- 1,094 seconds
- ✓ Accuracy :- 99.00 %
- ✓ Error rate :- 1.00 %

```
Using Theano backend.
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
71s - loss: 0.2508 - acc: 0.9272 - val_loss: 0.0786 - val_acc: 0.9753
Epoch 2/10
69s - loss: 0.0759 - acc: 0.9774 - val_loss: 0.0543 - val_acc: 0.9827
Epoch 3/10
86s - loss: 0.0546 - acc: 0.9831 - val_loss: 0.0468 - val_acc: 0.9840
Epoch 4/10
89s - loss: 0.0423 - acc: 0.9871 - val_loss: 0.0366 - val_acc: 0.9885
Epoch 5/10
96s - loss: 0.0335 - acc: 0.9897 - val_loss: 0.0359 - val_acc: 0.9877
Epoch 6/10
93s - loss: 0.0285 - acc: 0.9910 - val_loss: 0.0353 - val_acc: 0.9888
Epoch 7/10
88s - loss: 0.0235 - acc: 0.9929 - val_loss: 0.0331 - val_acc: 0.9888
Epoch 8/10
92s - loss: 0.0216 - acc: 0.9929 - val_loss: 0.0328 - val_acc: 0.9889
Epoch 9/10
89s - loss: 0.0170 - acc: 0.9947 - val_loss: 0.0312 - val_acc: 0.9895
Epoch 10/10
92s - loss: 0.0148 - acc: 0.9952 - val_loss: 0.0320 - val_acc: 0.9900
Error: 1.00%
```

Figure 9: Results with Convolutional Neural Network

More Complex Convolutional Neural Networks

Now we shall construct a larger Convolution neural network with extra hidden layers, pooling layers and well connected layers. With the help of adding more hidden layers and more nodes to process the input, we proceed to achieve better accuracy for the classification. We would implement such models when we are in great need of high precision. We construct the network design as follows: -

1. Initial layer would be a Convolutional layer. This layer utilized 30 feature maps with size of 5x5.

2. The secondary layer being Pooling layer (like MaxPooling2D), is configured to a pooling size of over 2*2.

3. The next layer would be a Convolutional layer. This layer utilizes 15 feature maps with size of 3×3.

4. Further the next layer being Pooling layer (like layer 2), is configured to pooling size of over 2 * 2.

5. Next would be a regularization layer with dropout called dropout layer. This layer is configured to randomly exclude 20% of neurons from the layer to counter the overfitting problem.

6. The further layer helps in processing the 2D matrix data on a vector called Flatten. This helps processing of output with the help of the fully connected layers.

7. The next layer is a fully connected version of the layer with 128 nuerons and activation function for validation.

8. The next layer is a fully connected version of the layer with 50 nuerons and activation function for validation.

9. At the end, the output layer is to output 10 classes. Hence, we utilize 10 nueron for 10 classess and a softmax activation function for probabilistic prediction for individual classes.

The complex version of Convolutional nueral network model takes some time to train and formulate the output on out MNIST dataset. This model contains 2 sets of convolutions & pooling layers. With a 0.79 % error rate, this complex deep network is highly efficient and accurate when a huge regular dataset is to be processed. With systems that require a high precision of the image classification, we would suggest going forward with a complex convolution neural network model.

Below is the results with following configuration details: -

- ✓ Training Set :- 60,000
- ✓ Test Set :- 10,000
- ✓ Execution time:- 2,098 seconds
- ✓ Accuracy :- 99.21 %
- ✓ Error rate :- 0.79 %

```
Using Theano backend.
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
75s - loss: 0.3875 - acc: 0.8800 - val_loss: 0.0790 - val_acc: 0.9746
Epoch 2/10
201s - loss: 0.1036 - acc: 0.9683 - val_loss: 0.0512 - val_acc: 0.9833
Epoch 3/10
434s - loss: 0.0735 - acc: 0.9774 - val_loss: 0.0455 - val_acc: 0.9851
Epoch 4/10
395s - loss: 0.0599 - acc: 0.9819 - val_loss: 0.0385 - val_acc: 0.9872
Epoch 5/10
205s - loss: 0.0498 - acc: 0.9848 - val_loss: 0.0310 - val_acc: 0.9904
Epoch 6/10
168s - loss: 0.0452 - acc: 0.9855 - val_loss: 0.0274 - val_acc: 0.9911
Epoch 7/10
199s - loss: 0.0415 - acc: 0.9867 - val_loss: 0.0274 - val_acc: 0.9909
Epoch 8/10
95s - loss: 0.0373 - acc: 0.9877 - val_loss: 0.0297 - val_acc: 0.9904
Epoch 9/10
253s - loss: 0.0325 - acc: 0.9896 - val_loss: 0.0261 - val_acc: 0.9909
Epoch 10/10
73s - loss: 0.0311 - acc: 0.9901 - val_loss: 0.0240 - val_acc: 0.9921
 Error: 0.79%
```

Figure 10: Results with a Complex Convolutional Neural Network

IMPLEMENTATION

For the implementation of each Deep learning model, we had chosen Python. There are many libraries that are helpful in neural network building using python. We would be utilizing the following libraries to

- Theano:-

  Theano is an efficient framework that works well in visualization of multi-dimensional array. As an image is an 2D array, it obviously seems helpful to utilize Theano.

  - ✓ Theano internally utilized a computers GPU processor to improve the computational speed.
  - ✓ Theano is equipped with internal self testing and unit verification capabilities.
  - ✓ Theano integrates with Numpy and Keras for better array implementation.

- Numpy:-

  Numpy helps in mathematical computations easier in python. Numpy support C++ and FORTRAN. Some of the simple operations that Numpy operates are fourier transform, linear algebra, trigonometry function and random functions.

- Keras:-

  Keras is one such open source library in Python for neural network implementation. Keras runs on the top of Theano or TensorFlow. We choose Keras, as it is one such capable of the fast implementation of neural networks. It was developed during research of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System).

  The library has number of implementation of regularly utilized neural network with building parts like layers, objectives, optimizers, activation function & a host for tools which make working along image and text data easier. The code is available on Github. As of September 2016, Keras is second fastest growing deep learning framework following Google's Tensorflow.

CONCLUSION

We had successfully implemented and executed all the three algorithms over MNIST data set, namely Mult-Layer Perceptron, Simple Convolutional nueral network and a Complex Convolutional neural network. With close observation of the result, we conclude the complex CNN works with highest efficiency of 99.21% over 2,000 seconds . But if we would have a small trade off of error we can use multilayer perceptron with near to 1.8% error rate and running at a marvelous speed of 81 seconds, which we can quote a remarkable considering training of 60,000 samples and testing 10,000 samples.

- Basic version of Multi-Layer Perceptron
  - ✓ Training Set :- 60,000
  - ✓ Test Set :- 10,000
  - ✓ Execution time:- 81 seconds
  - ✓ Accuracy :- 98.02 %
  - ✓ Error rate :- 1.92 %

- Simple Convolutional Neural Network
  - ✓ Training Set :- 60,000
  - ✓ Test Set :- 10,000
  - ✓ Execution time:- 1,094 seconds
  - ✓ Accuracy :- 99.00 %
  - ✓ Error rate :- 1.00 %

- Complex Convolutional Neural Network
  - ✓ Training Set :- 60,000
  - ✓ Test Set :- 10,000
  - ✓ Execution time:- 2,098 seconds
  - ✓ Accuracy :- 99.21 %
  - ✓ Error rate :- 0.79 %

REFERENCES

[1] Sergios Theodoridis, *Machine Learning: A Bayesian and Optimization Perspective*, 1st edition, Academic Press, 2015.
[2] Richard O. Duda, Peter E. Hart, David G. Stork, *Pattern Classification*, 2nd Edition, Wiley-Interscience, October 2000.
[3] Lazy Programmer, *Deep Learning in Python: Master Data Science and Machine Learning with Modern Neural Networks written in Python, Theano, and TensorFlow* (Machine Learning in Python)
[4] Li Deng and Dong Yu (2014), "Deep Learning: Methods and Applications", Foundations and Trends® in Signal Processing: Vol.7:No.3–4,pp197-387.http://dx.doi.org/10.1561/2000000039
[5] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *the Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Second Edition, Springer, February 9, 2009.
[6] Bickel, P.J., Ritov, Y., Zakai, A., *Some theory for generalized boosting algorithms.* Journal of Machine Learning Research 7, 705 –732 (2006)
[7] keras.io
[8] python.org
[9] google.com
[10] en.wikipedia .org
[11] yann.lecun.com/exdb/mnist/
[12] machinelearningmastery.com
[13] deeplearning.net/software/theano/
[14] neuralnetworksanddeeplearning.com