Handwritten Digit Recognition Using Machine Learning and Deep Learning

***Abstract*— Handwritten digit recognition has recently been of very interest among the researchers because of the evolution of various Machine Learning, Deep Learning and Computer Vision algorithms. In this report, I compare the results of some of the most widely used Machine Learning Algorithms like SVM, KNN & RFC andwith Deep Learning algorithm like multilayer CNN using Keras with Theano and Tensorflow. Using these, I was able to get the accuracy of 98.70% using CNN (Keras+Theano) as compared to 97.91% using SVM, 96.67% using KNN, 96.89% using RFC**

***Index Terms*—SVM, KNN, RFC, CNN.**

I. Introduction

HANDWRITTEN digit recognition is the ability of a computer system to recognize the handwritten inputs like digits, characters etc. from a wide variety of sources like emails, papers, images, letters etc. This has been a topic of research for decades. Some of the research areas include signature verification, bank check processing, postal address interpretation from envelopes etc.

A lot of classification techniques using Machine Learning have been developed and used for this like K-Nearest Neighbors, SVM Classifier, Random Forest Classifier etc. but these methods although having the accuracy of 97% are not enough for the real world applications.

One example of this is, if you send a letter with addressee name as “Anuj” and the system detects and recognizes it as “Tanuj” then it will not be delivered to “Anuj” but “Tanuj”. Although eventually it may come to the right address but if the mail is important, this delay can cost a lot. In short, the accuracy in these applications is very critical but these techniques do not provide the required accuracy due to very little knowledge about the topology of a task.

Here comes the use of Deep Learning. In the past decade, deep learning has become the hot tool for Image Processing, object detection, handwritten digit and character recognition etc. A lot of machine learning tools have been developed like scikit-learn, scipy-image etc. and pybrains, Keras, Theano, Tensorflow by Google, TFLearn etc. for Deep Learning. These tools make the applications robust and therefore more accurate.

The Artificial Neural Networks can almost mimic the human brain and are a key ingredient in image processing field. For example, Convolutional Neural Networks with Back Propagation for Image Processing, Deep Mind by Google for creating Art by learning from existing artist styles etc.

.

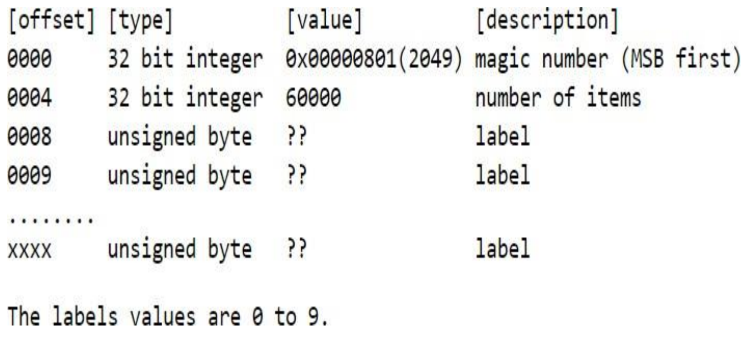
II. Database Modelling & Preprocessing

A. MNIST Dataset

The MNIST dataset, a subset of a larger set NIST, is a database of 70,000 handwritten digits, divided into 60,000 training examples and 10,000[2] testing samples. The images in the MNIST dataset are present in form of an array consisting of 28x28 values representing an image along with their labels. This is also the same in case of the testing images. The data is stored in four files[2][12]:

1. train-images-idx3-ubyte: training set images[2][12]
2. train-labels-idx1-ubyte:training set labels[2][12]
3. t10k-images-idx3-ubyte: test set images[2][12]
4. t10k-labels-idx1-ubyte: test set labels[2][12]

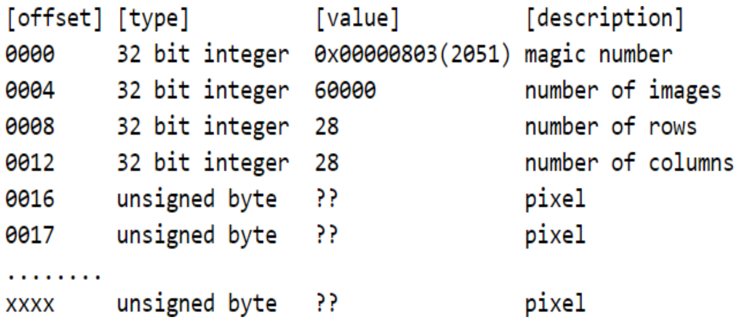
The ***Training Set Label*** file has the data in the following representation[2]:



```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000801(2049) magic number (MSB first)
0004     32 bit integer  60000            number of items
0008     unsigned byte   ??               label
0009     unsigned byte   ??               label
........
xxxx     unsigned byte   ??               label

The labels values are 0 to 9.
```

The ***Training Set Image*** file has the data in following representation[2]:



```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

The pixels are provided as an array of 784-d pixels and the values range from 0 to 255 i.e. 0 means background Black and 255 means it is White.

The ***Test Set Label*** file has the data represented as follows[2]:

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000801(2049) magic number (MSB first)
0004     32 bit integer  10000            number of items
0008     unsigned byte   ??               label
0009     unsigned byte   ??               label
........
xxxx     unsigned byte   ??               label
```

The label values are labelled 0 to 9.
The ***Test Set Image*** file has data represented in format as follows[2]:

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  10000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

B. *MNIST Dataset Format Analysis*

As you can see from above, the MNIST data is provided in a specific format. So, to be able to read the dataset it is first important to know that in what format the data is available to us. Both the Training and Testing images and labels have the first two columns consisting of the "Magic Number" and the number of items in the file.

The magic number has its first two bytes equal to zero.This magic number is read as MSB first and its format is as shown below:

2 Bytes	**1 Byte**	**1 Byte**
00	Data Type	Dimensions

For example, the train set images file contains the image information as shown below:

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            number of imag
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of colu
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

The magic number value for that data is 0x00000803(2051).
So the magic number value can be seen as[1]:

2 Bytes	**Data Type**	**Dimension**
00 00	08	03

This gives us the following information[1]:

1. The 0000 tells that it is the beginning of the file.
2. 08 tells us that it is unsigned byte type.
3. 03 tells us that the matrix has three dimensions.

The third byte represents whether the data is an integer, float, short, long or unsigned type. The list of all the cases used in place of third byte are as follows[1]:

Byte Value	**Data Type**
0x08	Unsigned Byte
0x09	Signed Byte
0x0B	Short (2 Bytes)
0x0C	int (4 Bytes)
0x0D	float (4 Bytes)
0x0E	double (8 Bytes)

The fourth byte tells thedimension of the vector or matrixi.e. the number of rows and columns. If it is equal to 1, then it‟s a vector else it is a matrix. The number of items variable is also read as MSB first.

C. *Reading the MNIST Dataset*

To read in the MNIST dataset, I have written a python code with class "MNIST" and defined the functions inside this class. The basic outline of the code can be summarized as follows:

1. Read the MNIST dataset files using python as „rb‟.
2. For each file, there is a specific magic number. Take the data files one by one and read them if the condition for the magic number is satisfied i.e. if the magic number matches the type of the file. For example, to read the ***Training Set Image***data file, you need to check first if the magic number is equal to 2051 else don‟t read the file for training labels[11].
3. Read the number of rows and columns provided in the data file in next row to the magic number.
4. The using this information, read the 28x28 data corresponding to the respective label provided in the row-wise format.
5. Follow the above steps for rest of the files and put the respective data in the variables.

The function that reads the image data returns the image information and the labels. This data is used further in every program for making predictions.
To check that the data has been read correctly, we print the data for some of the labels. The output is as follows:

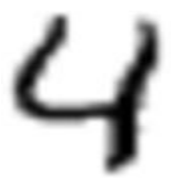


Fig. 1: SampleMNIST Data

III. Classification Using Machine Learning

To show the working accuracy of Machine Learning algorithms, I am using three classifiers as follows:

1. Random Forest Classifier [RFC]
2. K-Nearest Neighbors [KNN]
3. Supervised Vector Machine [SVM]

A. *Classification using Random Forest Classifier (RFC)*

Random Forest Classifier is an ensemble method used for classification or regression. Random Forest Classifier works using a huge collection of de-correlated decision trees. In this, the training data forms a matrix as input.Using this matrix, a large number of new matrix with random elements are created.Using each of these matrix, a corresponding decision tree is formed for classification of the testing data.
When the testing data is input, all these decision trees classify the input test data and predict the class to which the input belongs.The result is found based on the prediction result which has the maximum count as the result of the classifiers.
To make predictions, once the training is done, the average of predictions from all individual regression treesis taken using the following formula:

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x')$$

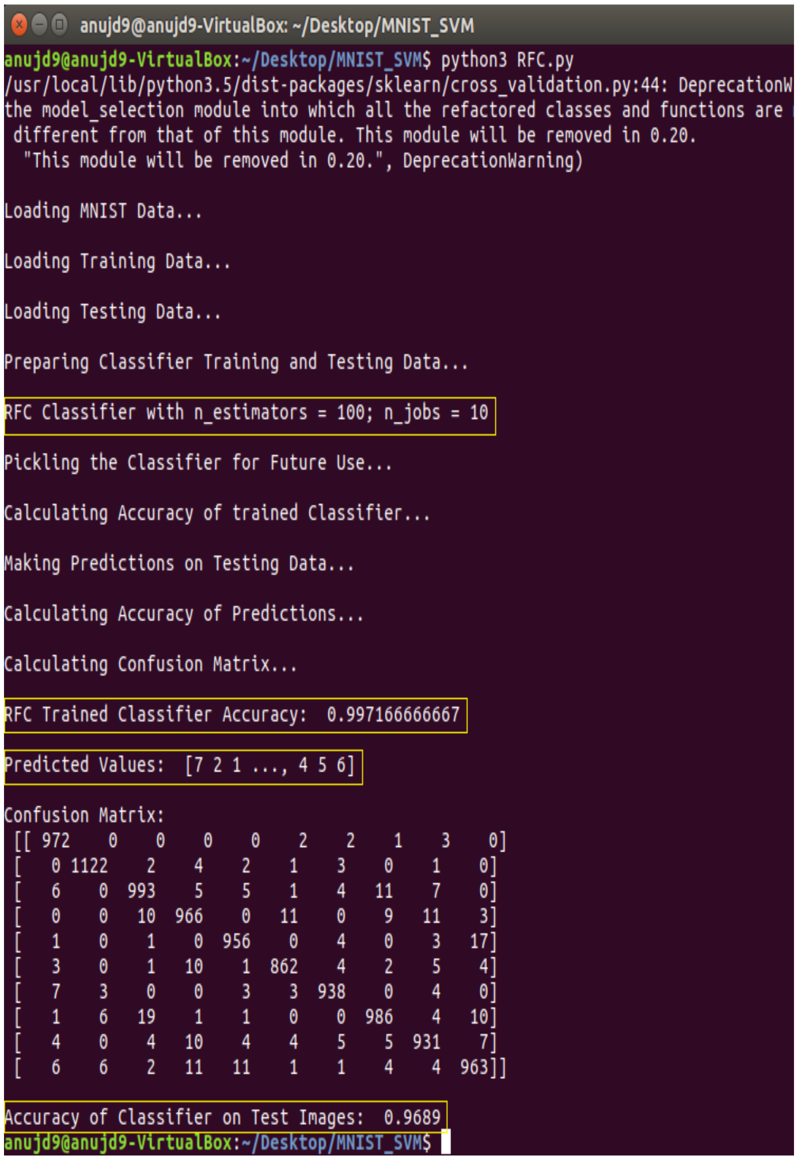
Doing this improves accuracy of the algorithm as well as prevents overfitting.
For classification of the MNIST data, the RFC works as follows:

1. Load the MNIST data.
2. Divide and label the data as Training and Testing Image and labels.
3. Use *cross validation* to divide the training data into training and testing data to train the classifier.
4. Train the Classifier using *RFC* algorithm. Provide training data and labels as input to train the classifier. RFC requires the number of trees in forest, number of features to look for best split, maximum depth of the tree etc. as the input.
5. The digit recognized using *RFC* is then matched with the provided *Training Labels* to get the score/accuracy of the trained classifier.
6. This trained classifier is pickled to be used again on the testing data.
7. The *Test Image* data is used to predict the labels of digits and is compared with the provided test labels to see for the accuracy of the algorithm.
8. The *Confusion Matrix* is printed that provides the percentage of accuracy with which each digit has been recognized.

Using Scikit-Learn, the RFC is applied to data in the following format[4]:

"class sklearn.ensemble.**RandomForestClassifier**(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_split=1e07, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None[4])
"[4]

The following image shows step by step output of the RFC classifier:



```
anujd9@anujd9-VirtualBox: ~/Desktop/MNIST_SVM
anujd9@anujd9-VirtualBox:~/Desktop/MNIST_SVM$ python3 RFC.py
/usr/local/lib/python3.5/dist-packages/sklearn/cross_validation.py:44: DeprecationW
the model_selection module into which all the refactored classes and functions are
 different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

Loading MNIST Data...

Loading Training Data...

Loading Testing Data...

Preparing Classifier Training and Testing Data...

RFC Classifier with n_estimators = 100; n_jobs = 10

Pickling the Classifier for Future Use...

Calculating Accuracy of trained Classifier...

Making Predictions on Testing Data...

Calculating Accuracy of Predictions...

Calculating Confusion Matrix...

RFC Trained Classifier Accuracy:  0.997166666667

Predicted Values:  [7 2 1 ..., 4 5 6]

Confusion Matrix:
 [[ 972    0    0    0    0    2    2    1    3    0]
 [    0 1122    2    4    2    1    3    0    1    0]
 [    6    0  993    5    5    1    4   11    7    0]
 [    0    0   10  966    0   11    0    9   11    3]
 [    1    0    1    0  956    0    4    0    3   17]
 [    3    0    1   10    1  862    4    2    5    4]
 [    7    3    0    0    3    3  938    0    4    0]
 [    1    6   19    1    1    0    0  986    4   10]
 [    4    0    4   10    4    4    5    5  931    7]
 [    6    6    2   11   11    1    1    4    4  963]]

Accuracy of Classifier on Test Images:  0.9689
anujd9@anujd9-VirtualBox:~/Desktop/MNIST_SVM$
```

Fig. 2: RFC MNST Digit Recognition Accuracy

From the image above, we can see that the accuracy of the trained classifier is 99.7% and for handwritten digit prediction, the accuracy comes down to 96.89%. This means that this algorithm lacks the accuracy by 3.11%. The error of 3.11% is more than enough to predict a character wrong and send a mail to a wrong address.
Let"s see some more algorithms and compare their accuracy for this prediction.

B. *Classification using K-Nearest Neighbors (KNN)*

K-Nearest Neighbors is an algorithm in which the best estimate among all the values is the value that has maximum number of neighbors with smallest Euclidian or Hamming distance.

KNN is an instance based learning[7]. To work well, this algorithm requires a training dataset which is a set of well labelled data points.
This algorithm takes as input a new data point and makes the classification for this by calculating the Euclidian or Hamming distance between the new data point and the labelled data point[5]. The Euclidian distance is calculated using the following formula:

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

For classification, the KNN algorithm works as follows:

1. Load the MNIST data.
2. Divide and label the data as Training andTesting Image and labels.
3. Use *cross validation* to divide the training data into training and testing data to train the classifier.
4. Train the Classifier using *KNN* algorithm. Provide training data and labels as input to train the classifier. KNN uses the difference in distance between the actual points and the points provided to classify the digit in the image.
5. The digit recognized using *KNN* is then matched with the provided *Training Labels* to get the score/accuracy of the trained classifier.
6. This trained classifier is pickled to be used again on the testing data.
7. The *Test Image* data is used to predict the labels of digits and is compared with the provided test labels to see for the accuracy of the algorithm.
8. The *Confusion Matrix* is printed that provides the percentage of accuracy with which each digit has been recognized.

Using Scikit-Learn, the KNN is applied to data in the following format[5]:

class sklearn.neighbors.**KNeighborsClassifier**(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=10, **kwargs)[5]

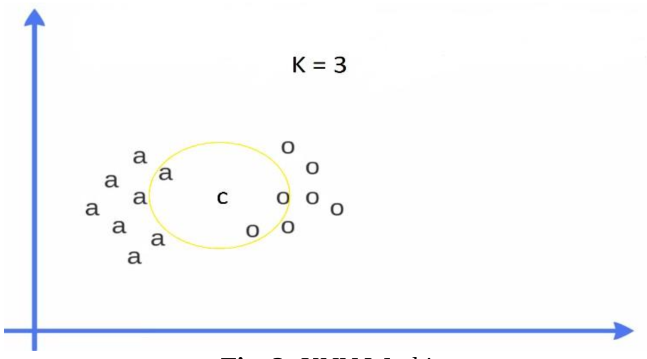
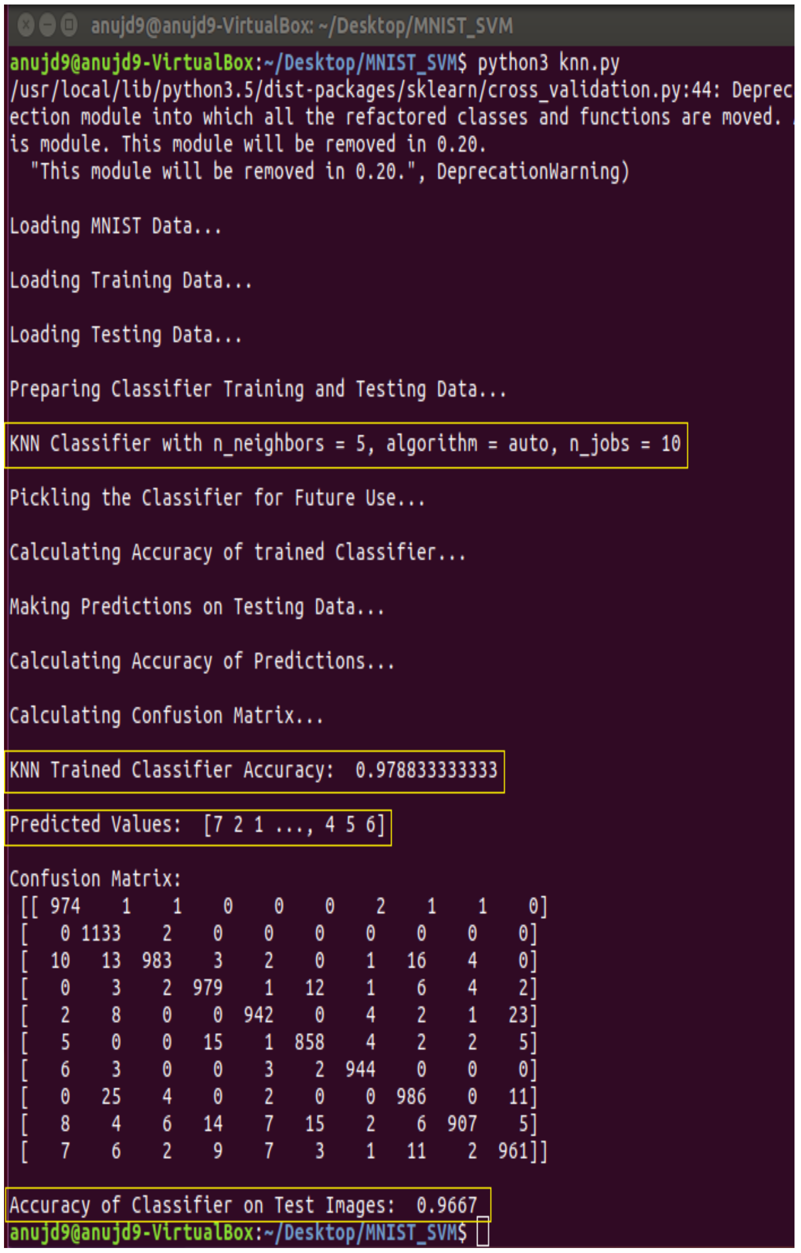


Fig. 3: KNN Working

The following image shows step by step output of the KNN classifier:


```
anujd9@anujd9-VirtualBox: ~/Desktop/MNIST_SVM
anujd9@anujd9-VirtualBox:~/Desktop/MNIST_SVM$ python3 knn.py
/usr/local/lib/python3.5/dist-packages/sklearn/cross_validation.py:44: Deprec
ection module into which all the refactored classes and functions are moved.
is module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

Loading MNIST Data...

Loading Training Data...

Loading Testing Data...

Preparing Classifier Training and Testing Data...

KNN Classifier with n_neighbors = 5, algorithm = auto, n_jobs = 10

Pickling the Classifier for Future Use...

Calculating Accuracy of trained Classifier...

Making Predictions on Testing Data...

Calculating Accuracy of Predictions...

Calculating Confusion Matrix...

KNN Trained Classifier Accuracy:  0.978833333333

Predicted Values:  [7 2 1 ..., 4 5 6]

Confusion Matrix:
 [[ 974    1    1    0    0    0    2    1    1    0]
 [    0 1133    2    0    0    0    0    0    0    0]
 [   10   13  983    3    2    0    1   16    4    0]
 [    0    3    2  979    1   12    1    6    4    2]
 [    2    8    0    0  942    0    4    2    1   23]
 [    5    0    0   15    1  858    4    2    2    5]
 [    6    3    0    0    3    2  944    0    0    0]
 [    0   25    4    0    2    0    0  986    0   11]
 [    8    4    6   14    7   15    2    6  907    5]
 [    7    6    2    9    7    3    1   11    2  961]]

Accuracy of Classifier on Test Images:  0.9667
anujd9@anujd9-VirtualBox:~/Desktop/MNIST_SVM$
```

Fig. 4: KNN MNIST Digit Recognition Accuracy

From the image above, we can see that the accuracy of the trained classifier is 97.88% and for handwritten digit prediction, the accuracy comes down to 96.67%. This means that this algorithm lacks the accuracy by 3.33%. The error of 3.33% is large when considered for this case.

C. *Classification using SVM Classifier*

In machine learning, support vector machine is an important model. It is a supervised learning model used for classification and regression[8]. In this model, we are given a set of training examplesin which each one of them is marked to be belonging to one of the two categories[8]. A Support Vector Machine model represents a point in space which is mapped such that the two different classes with their elements are separated by as much distance as possible.The image pixels or the input under test is then mapped into this space and predictions are made based on the class or category to which the test input belongs [2]. The graphical representation of working of an SVM classifier is as shown below:

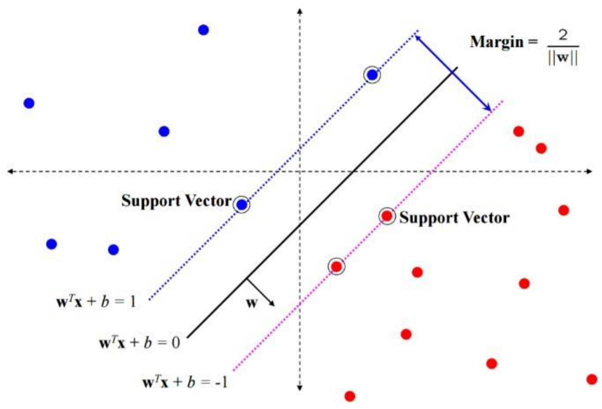

Margin = $\frac{2}{||\mathbf{w}||}$
Support Vector
Support Vector
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}$
$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b = -1$

Fig. 5: Support Vector Machine

Here,

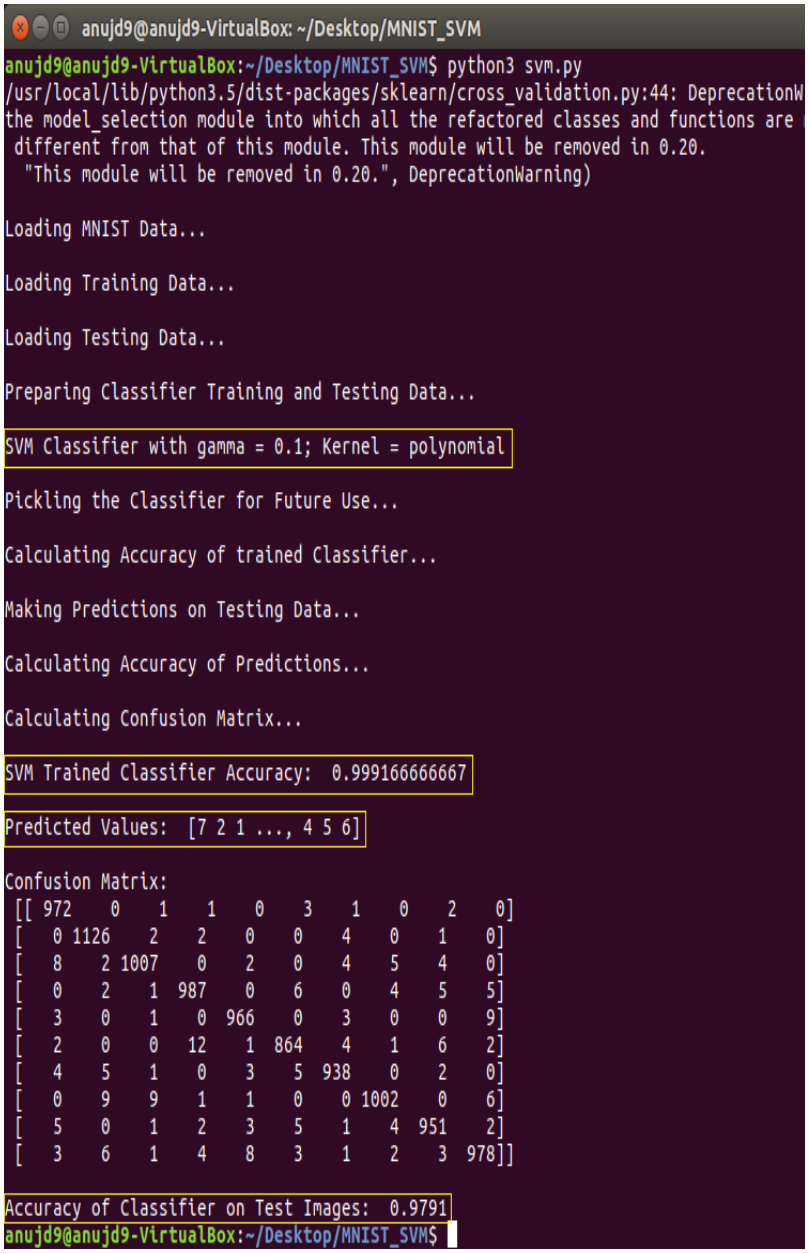
For classification, the SVM algorithm works as follows:

1. Load the MNIST data.
2. Divide and label the data as Training and Testing Image and labels.
3. Use *cross validation* to divide the training data into training and testing data to train the classifier.
4. Train the *SVM Classifier*. Provide training data and labels as input to train the classifier. SVM uses the existing labelled data to learn and then classifies the unlabeled data based on that learning.
5. The digit recognized using *SVM* is then matched with the provided *Training Labels* to get the score/accuracy of the trained classifier.
6. This trained classifier is pickled to be used again on the testing data.
7. The *Test Image* data is used to predict the labels of digits and is compared with the provided test labels to see for the accuracy of the algorithm.
8. The *Confusion Matrix* is printed that provides the percentage of accuracy with which each digit has been recognized.

Using Scikit-Learn, the KNN is applied to data in the following format[6]:

class sklearn.svm.**SVC**(C=1.0, kernel='rbf', degree=3, gamma ='auto', coef0=0.0, shrinking=True, probability=False, tol=0.0 01, cache_size=200, class_weight=None, verbose=False, max _iter=1, decision_function_shape=None, random_state=None) [6]

The following image shows step by step output of the SVM classifier:



```
anujd9@anujd9-VirtualBox:~/Desktop/MNIST_SVM$ python3 svm.py
/usr/local/lib/python3.5/dist-packages/sklearn/cross_validation.py:44: DeprecationW
the model_selection module into which all the refactored classes and functions are
 different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

Loading MNIST Data...

Loading Training Data...

Loading Testing Data...

Preparing Classifier Training and Testing Data...

SVM Classifier with gamma = 0.1; Kernel = polynomial

Pickling the Classifier for Future Use...

Calculating Accuracy of trained Classifier...

Making Predictions on Testing Data...

Calculating Accuracy of Predictions...

Calculating Confusion Matrix...

SVM Trained Classifier Accuracy:  0.999166666667

Predicted Values:  [7 2 1 ..., 4 5 6]

Confusion Matrix:
 [[ 972    0    1    1    0    3    1    0    2    0]
 [    0 1126    2    2    0    0    4    0    1    0]
 [    8    2 1007    0    2    0    4    5    4    0]
 [    0    2    1  987    0    6    0    4    5    5]
 [    3    0    1    0  966    0    3    0    0    9]
 [    2    0    0   12    1  864    4    1    6    2]
 [    4    5    1    0    3    5  938    0    2    0]
 [    0    9    9    1    1    0    0 1002    0    6]
 [    5    0    1    2    3    5    1    4  951    2]
 [    3    6    1    4    8    3    1    2    3  978]]

Accuracy of Classifier on Test Images:  0.9791
anujd9@anujd9-VirtualBox:~/Desktop/MNIST_SVM$
```

Fig. 6: SVM MNIST Digit Recognition Accuracy

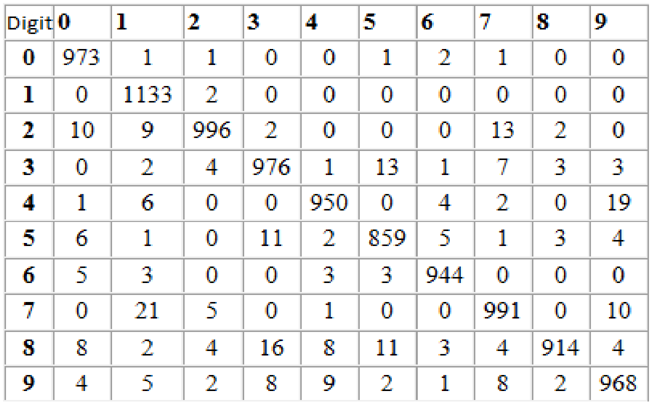
From the image above, we can see that the accuracy of the trained classifier is 99.91% and for handwritten digit prediction, the accuracy comes down to 97.91%.

D. Confusion Matrix

In all the outputs of the classifiers, we see a confusion matrix. A confusion matrix defines a specific table that allows the visualization of the performance of an algorithm by providing the accuracy corresponding to each of the input and output classes[14]. In this the instance in a predicted class is shown as the columns of data whereas the rows represent the instances in actual classes.

In short a confusion matrix C is such that C [x, y] is equal to the number of observations known to be in group x but predicted to be in group y[9].

Thus in binary classification, the count of true negatives is C_[0,0], false negatives is C_[1,0], true positives is C_[1,1] and false positives is C_[0,1].
So according to this definition, the confusion matrix in the images above shows the accuracy with which the particular digit is recognized using that algorithm respectively.
For example:



Digit	0	1	2	3	4	5	6	7	8	9
0	973	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	10	9	996	2	0	0	0	13	2	0
3	0	2	4	976	1	13	1	7	3	3
4	1	6	0	0	950	0	4	2	0	19
5	6	1	0	11	2	859	5	1	3	4
6	5	3	0	0	3	3	944	0	0	0
7	0	21	5	0	1	0	0	991	0	10
8	8	2	4	16	8	11	3	4	914	4
9	4	5	2	8	9	2	1	8	2	968

Fig. 7: Sample Confusion Matrix representing the Accuracy of each Digit

E. *Machine Learning Accuracy Comparison*

We have seen the working of three of the most commonly used Machine Learning algorithms used for Handwritten Digit recognition.
The method of Random Forest Classifier is able to recognize the digits 96.89% correctly.
Using the K Nearest Neighbors, we are able to get an accuracy of 96.67% for the digit recognition.
Similarly, for the SVM classifier, we get an accuracy of 97.91% for the digit recognition.

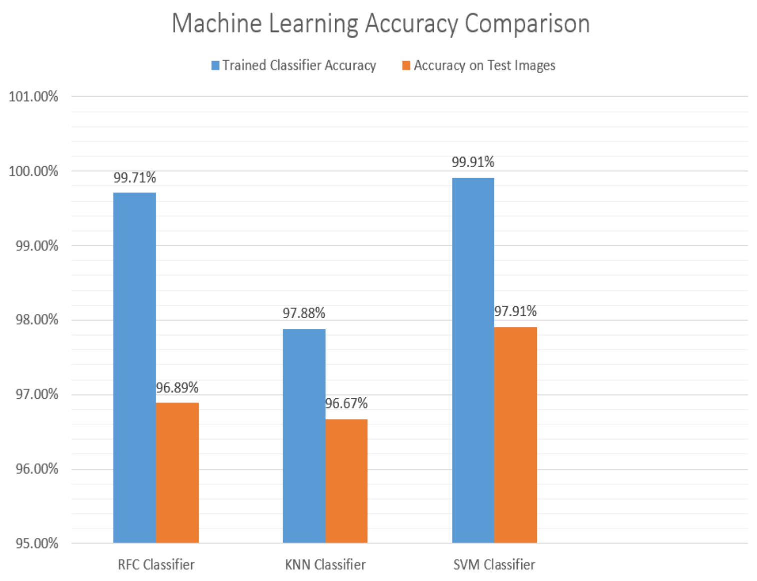


Fig. 8: Comparison of Accuracy using Machine Learning Algorithms for MNIST Handwritten Digit Recognition

IV. Classification Using Deep Learning

For digit recognition using Deep Neural Networks, I am using multi-layer Deep Convolutional Neural Network [CNN]. I have also used two different tools, Keras + Theano and Tensorflow by Google to show the working accuracy of Deep Neural Network.

A. *Convolutional Neural Network*

A Convolutional Neural Network (CNN) is a type of feed-forward Artificial Neural Network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex[10].
Convolutional Neural Networksconsist of neurons that have learnable weights and biases. Each neuron receives some input, performs a dot product and optionally follows it with a non-linearity[15][10].

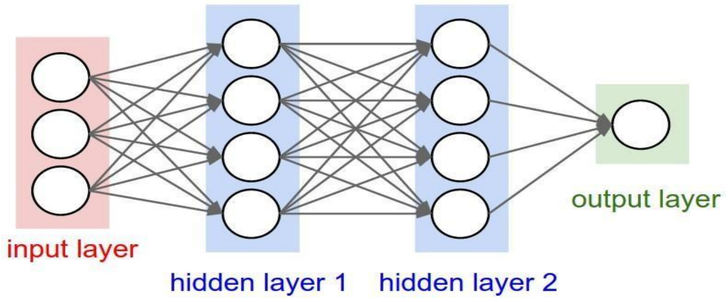


Fig. 9: Convolutional Neural Network Basic Layout

The whole Convolutional Neural Network expresses a differentiable scorefunction that is further followed by a Softmax function.
The data input into the Convolutional Neural Network is arranges in the form of its width, height and depth as shown in figure below.

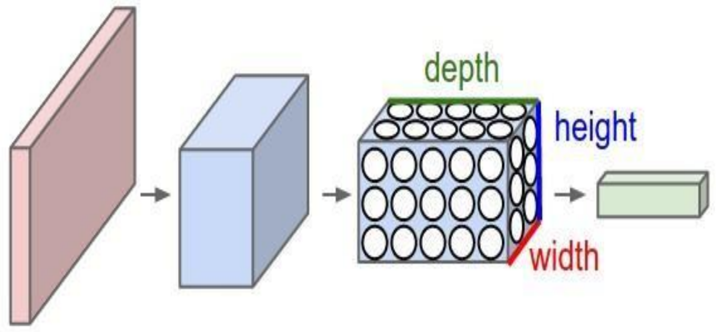


Fig. 10: Arrangement of Neurons in CNN

B. *Layers of Convolutional Neural Network*

A CNN consists of a lot of layers. These layers when used repeatedly, lead to a formation of a Deep Neural Network.
Three main types of layers used to build a CNN are:

1. **Input:** This layer holds the raw pixel values of image.
2. **Convolutional Layer:** This layer gets the results of the neuron layer that is connected to the input regions.We define the number of filters to be used in this layer. Each filter may be a 5x5 window that slider over the input data and gets the pixel with the maximum intensity as the output[10].
3. **Rectified Linear Unit [ReLU] Layer**: This layer applies an element wise activation function on the

image data[10]. We know that a CNN uses back propagation. So in order to retain the same values of the pixels and not being changed by the back propagation, we apply the ReLU function.

4. **Pooling Layer:** This layer perform a down-sampling operation along the spatial dimensions (width, height), resulting in volume[10].
5. **Fully Connected Layer:** This layers is used to compute the score classes i.e which class has the maximum score corresponding to the input digits[10].

C. *CNN for Handwritten Digit Recognition*

The CNN for Handwritten Digit Recognition works in three main phases.

1. **<u>Phase1 - Input MNIST Data1</u>:** The first phase is to input the MNIST data. The MNIST data is provided as 784-d array of pixels. So firstly we convert it to grayscale images using 28x28 matrix of pixels.

2. **<u>Phase2 – Building Network Architecture:</u>** In the second phase, we define the models to be used to build a convolutional neural network. Here, we use the *Sequential* class from *Keras* to build the network. In this network, we have three layer sets of layers "*CONV =>ReLU=> POOL".*

a) **<u>First Convolution Layer:</u>** In the first layer, we take 20 convolutional filters that go as a sliding window of size 5x5 over all the images of 28x28 matrix size and try to get the pixels with most intensity value.

b) **<u>ReLU Function:</u>** We know that convolution is a method that uses *Back Propagation.* So using the ReLU function as the activation function just after the convolutional layer reduces the likelihood of the vanishing gradient and avoids sparsity. This way we don‟t lose the important data and even get rid of redundant data like a lot of 0‟s in the pixels.

c) **<u>Pooling Layer:</u>** The pooling layer gets the data from the ReLU function and down-samples the steps in the 3D tensor. In short it pools all the pixels obtained from previous layers and again forms a new image matrix of a smaller size. These images are again input into the second set of layers i.e. "*CONV =>ReLU=> POOL"* and this process goes on till we get to a smallest set of pixels from which we can classify the digit.

3. **<u>Phase 3 –Fully Connected Layer:</u>** The fully connected layer is used to connect each of the previous layers to the next layers. This layer consists of 500 neurons. Finally, we apply a Softmax Classifierthat returns a list of probabilities for each of the 10 class labels. The class label with the largest probability is chosen as the final classification from the network and shown in the output.

This output received is used to make the confusion matrix for the model. In this we can add more number of layers but adding more layers might affect the accuracy of the system. ince, it uses multiple layers, so it‟s called a Deep Learning system.

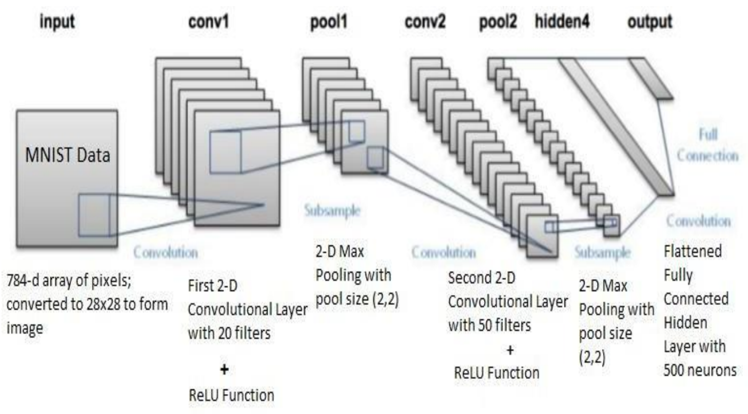

input
conv1
pool1
conv2
pool2
hidden4
output
MNIST Data
Convolution
Subsample
Convolution
Subsample
Full Connection
Convolution
784-d array of pixels; converted to 28x28 to form image
First 2-D Convolutional Layer with 20 filters + ReLU Function
2-D Max Pooling with pool size (2,2)
Second 2-D Convolutional Layer with 50 filters + ReLU Function
2-D Max Pooling with pool size (2,2)
Flattened Fully Connected Hidden Layer with 500 neurons

Fig. 11: CNN Layers for Handwritten Digit Recognition

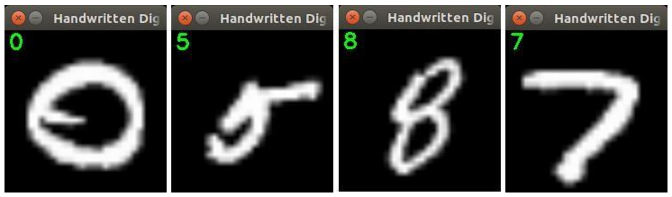
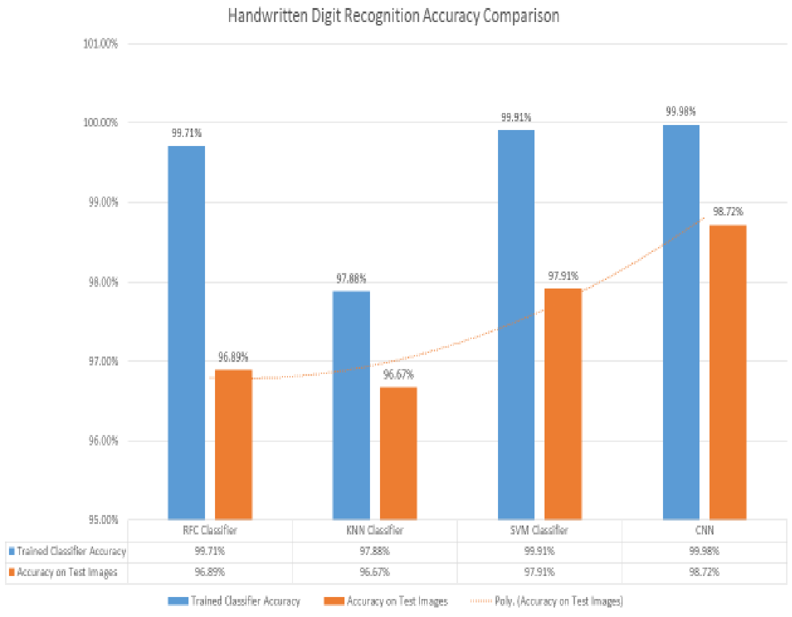


Fig. 12: CNN Prediction Results on MNIST

V. Results and Analysis

For commercial applications, the accurate recognition of the digits, characters etc. along with the speed of recognition is of great interest.

The image below shows the accuracy comparison of the various techniques used by me for handwritten digit recognition.



	RFC Classifier	KNN Classifier	SVM Classifier	CNN
Trained Classifier Accuracy	99.71%	97.88%	99.91%	99.98%
Accuracy on Test Images	96.89%	96.67%	97.91%	98.72%

Fig. 13: Accuracy Comparison of all Techniques

We see that the CNN with 3 hidden layers gives the most amount of accuracy of 98.72%. Although, this accuracy is not optimal as more accuracy can also be achieved. Using Google‟s Tensorflow the accuracy of 99.70% is achieved.

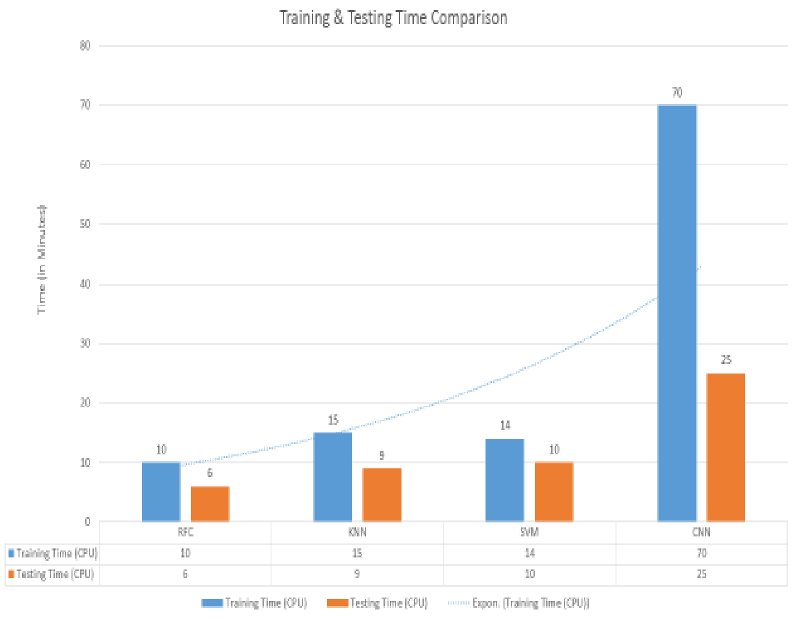
Table 2: Percent Accuracy of Each Classification Technique

	RFC	KNN	SVM	CNN
Trained Classifier Accuracy	99.71%	97.88%	99.91%	99.98%
Accuracy on Test Images	96.89%	96.67%	97.91%	98.72%

Table 3: Classifier Error Rate Comparison

Model	Test Error Rate
Random Forest Classifier	3.11%
K Nearest Neighbors	3.33%
Supervised Vector Machine	2.09%
Convolutional Neural Network	1.28%

Next comes the speed. The recognition system must be able to recognize the images as quickly as possible. For this large dataset, the training and testing time of all the classifiers is listed below. Note that these timings are for training and testing on the CPU only. Using GPU for this purpose can greatly reduce the training and testing time.



	RFC	KNN	SVM	CNN
Training Time (CPU)	10	15	14	70
Testing Time (CPU)	6	9	10	25

Fig. 14: Classifiers Training & Testing Time Comparison

The table below lists the training& testing time required by each classifier defined above.

Table 4: Training & Testing Time Comparison

Model	Training Time	Testing Time
Random Forest Classifier	10 min	6 min
K Nearest Neighbors	15 min	9 min
Supervised Vector Machine	14 min	10 min
Convolutional Neural Network	70 min	20 min

VI. Conclusion

An implementation of Handwritten Digit Recognition using Deep Learning has been implemented in this paper. Additionally, some of the most widely used Machine Learning algorithms i.e. RFC, KNN and SVM have been trained and tested on the same data to draw a comparison as to why we require deep learning methods in critical applications like Handwritten Digit Recognition. In this paper, I have shown that that using Deep Learning techniques, a very high amount of accuracy can be achieved. Using the Convolutional Neural Network with Keras and Theano as backend, I am able to get an accuracy of 98.72%.

In addition to this, implementation of CNN using Tensorflow gives an even better result of 99.70%. Every tool has its own complexity and accuracy. Although, we see that the complexity of the code and the process is bit more as compared to normal Machine Learning algorithms but looking at the accuracy achieved, it can be said that it is worth it. Also, the current implementation is done only using the CPU.

I have also implemented the same using CPU on EC2 instance using Amazon Web Service and got similar results. For additional accuracy, reduced training and testing time, the use of GPU‟s is required. Using GPU‟s I can get much more parallelism and attain much better results.

References

[1] Fatahi, M., 2014. MNIST handwritten digits.

[2] http://cvisioncentral.com/resources-wall/?resource=135

[3] https://en.wikipedia.org/wiki/Support_vector_machine

[4] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[5] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[6] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[7] Kumar, R., Goyal, M.K., Ahmed, P. and Kumar, A., 2012, December. Unconstrained handwritten numeral recognition using majority voting classifier. In *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on* (pp. 284-289). IEEE.

[8] https://en.wikipedia.org/wiki/Support_vector_machine

[9] Kabir, F., Siddique, S., Kotwal, M.R.A. and Huda, M.N., 2015, March. Bangla text document categorization using Stochastic Gradient Descent (SGD) classifier. In *Cognitive Computing and Information Processing (CCIP), 2015 International Conference on* (pp. 1-4). IEEE.

[10] http://cs231n.github.io/convolutional-networks/

[11] Simard, P.Y., Steinkraus, D. and Platt, J.C., 2003, August. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*(Vol. 3, pp. 958-962).

[12] http://yann.lecun.com/exdb/mnist/

[13] Koprinkova-Hristova, V.M.P., Villa, G.P.A.E. and Kasabov, B.A.N., Artificial Neural Networks and Machine Learning–ICANN 2013.

[14] Alsaad, A., 2016. *Enhanced root extraction and document classification algorithm for Arabic text* (Doctoral dissertation, Brunel University London).

[15] Meng, W., 2015. A sentence-based image search engine.

Anuj Dutt is a Computer Engineering graduate from University of Florida, FL, USA. His interests lie in the field of AI, Machine Learning, Deep Learning, Natural Language Processing, Computer Vision and IOT.

AashiDutt is a student at Computer Science Department at Kurukshetra University, Kurukshetra, India. Her interests lie in the field of Machine Learning, Cognitive Computing, Deep Learning and Embedded Systems.