Digit Recognizer using Hand Written Images

Pratiksha Rakesh Singh, Suchet Vinod Pawar, Parth Bharatkumar Shah, Melvin Alex Mathew Andoor

December 12, 2019

1. Abstract

The rapid growth of new documents and multimedia news has created new challenges in pattern recognition and machine learning. Handwriting character recognition has become a standard research area due to advances in technologies such as the handwriting capture devices and powerful mobile computers. However, since handwriting very much depends on the writer, building a high-reliability recognition system that recognizes any handwritten character input to an application, is challenging. Before the invention of computers all the data, the information's, etc. where written on paper and this was a complicated form of storage because if any one paper is lost or gets damaged then all the information depended on that paper will be void. Therefore, when the Computers where invented all the data and the information where able to be stored safely without any fear of it getting lost or damaged. But still there existed one issue and that was to store all the data and information that was written on paper before the invention of computers into the computers, and for this money was wasted on manual labor. If there was Handwritten Recognition system that could identify and convert this paper information into digital information, then the expenditure on the manual labor could have been avoided.

2. Introduction

Handwritten recognition system is the working of a machine to train itself or to recognize the digits from different sources like emails, bank cheque, papers, images, etc. The studies on this system is being done since 1998 with various types of Algorithms. The error rate has decreased since 1998 from 12% using linear classifier to 0.23% in 2012 using Convolutional Nets. Unsupervised learning methods like Auto Encoder ad Deep learning are the methods using which the Data scientists are planning to develop this Handwritten digit recognition system. The goal of this project is to create a model that will be able to recognize & determine the handwritten digits from its image by using the concepts of Support Vector Machine. The reason for choosing SVM classifier is because, SVM falls into the category of supervised learning, and with the bonus of classification as well as regression problems. SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. Digit recognizers should be built such that they achieve accuracy and reliable performance. SVMs are effective in high dimensional spaces, therefore we use SVMs given the high dimensionality of our input space, i.e. 784 features. Also, the accuracy rate of SVM is much higher than the other classifiers like KNN, Neural Network, etc.

3. Related Work

Patrice Y. Simard, Dave Steinkraus, John C. Platt [1] narrated that the convolutional neural networks are better for the visual document analysis like handwriting recognition task. The main essential practices for this task are the expansion of the data sets using elastic distortions and the use of convolutional neural networks. T.Siva Ajay [2] also proposed that the higher rate of accuracy in handwritten digit recognition task can be achieved by the use of convolutional neural networks. The implementation of CNN is made easy and simple by the use of LeNet engineering. As a result of this accuracy greater than 98% is obtained in this paper.

Ming Wu and Zhen Zhang [3] in 2010 made a comparison between different classifiers to conclude which gives better performance in the recognition task. The comparison is done between the six classifiers namely LDA (Linear Discriminant Analysis), GMM (Gaussian Mixture Models), QDA(Quadratic Discriminant Analysis), SVML (SVM with linear kernel function), SVMR (SVM with radial basis kernel function) and k-NN. Out of all the classifiers,k-NN (k=3) gives the lowest error rate. Islam [4] started a challenging task of recognition task of Arabic handwritten digits. For this, they decided to carry on the research using the Deep Convolutional Neural Networks. The accuracy of 95.7% is achieved as a result of this work. A Comprehensive Data Analysis on Handwritten Digit Recognition using Machine Learning Approach 1450 Published By: Blue Eyes Intelligence Engineering Retrieval Number: F3888048619/19©BEIESP & Sciences Publication

In 2015, Saeed AL Mansoori [5] applied the Multi-Layer Perceptron model to identify handwritten digits. The samples from the data set are trained by employing gradient descent backpropagation algorithm and later feedforward algorithm. From the obtained results it can be observed that the digit 5 has the highest accuracy of 99.8% whereas digit 2 has the lowest accuracy of 99.04%. And the proposed system achieved an overall accuracy of 99.32%. Xuan Yang and Jing Up [6] focused on the handwritten multi-digit recognition on mobile using Convolutional Neural Network. In this paper, a mobile application MDig is explained which is used to identify multiple digits using CNN. A narrow CNN is designed on mobile which gives a 99.07% accuracy using MNIST data set. Shobhit Srivastava, Sanjana Kalani, Umme Hani and Sayak Chakraborty [7] illustrated the handwritten recognition using Template Matching, Support Vector Machine and Artificial Neural Networks. Among the methods used, Artificial Neural Networks turned out to give more accurate results.

Shashank Mishra, D.Malathi and K.Senthil Kumar [8] attempted the handwritten recognition using Deep Learning. They used Convolutional Neural Network as a result of which they concluded that accuracy is increased and there is a reduction in the computation time. The accuracy of 99.2% is obtained. One of the works by Deric Pang and Saidutt Nimmagadda on “An Investigation of Classification Techniques for Handwritten Digit Recognition”. The goal for this project was quite simple—To take pixel data from images of hand-drawn digits and classify them as a number from 0 to 9. They experienced the advantages of CNN on large datasets and also have drawn conclusions on kernelization and various classifier techniques.

4. Problem Definition

The problem definition is given a dataset of labeled handwritten images, we build a classifier that would assign correct labels to the new images. Our basic motto is classification of handwritten digits via extracting the features and then recognizing them. Our basic Work-Flow for classifying using SVM is to get data from MNIST, separate labels and features then recheck correctness of feature/label combination in training and then continue to build the model and calculate accuracy. Our Work-Flow for Prediction will be taking the input, threshold the image i.e. resizing, add pixels into data array then transform the image and output the prediction that the system has predicted.

5. Methodology

The basic approach for this problem is commencing with accepting the image and then pre-processing the image. Pre-processing of the image includes various other processes into it, but the most prominent ones are the scaling of the image which means re-shaping the entire image in a required and defined size; centering of the image which is to get the digit in the center of the defined size frame; to remove pixels with constant intensity and along with it the re-scaling of picture intensity which is changing values from 0 i.e. black to 255 i.e. white. After the image is pre-processed, we segmented the image in equal size segments. This pre-processed segmented image is then passed on for feature extraction.

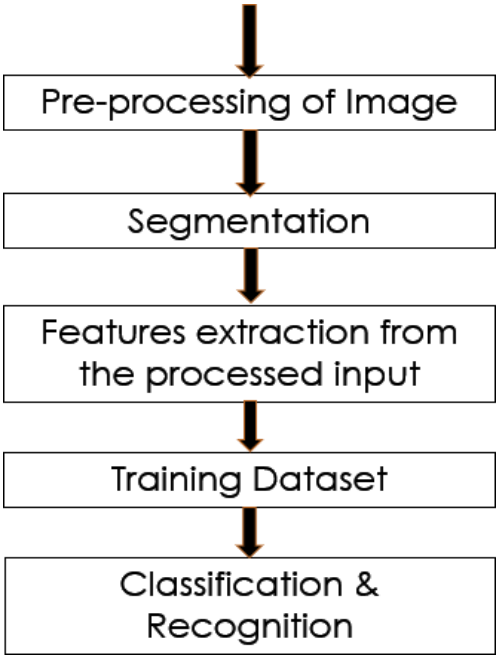


Figure 1: Framework Overview

Feature extraction is basically to recognize the input. In order to do this, variant number of steps are required to be performed which will ultimately at end give us an almost clear recognizing factor for the image. The feature extraction process is performed from left to right and top to bottom of the image. Initially the original image is converted to a sharpened one, then the gray scale is converted to black and white. We then made sure that the lines are thin and detected the end points. These end points are then

removed and new end points are detected on the thin image. At the last, we detected the junction points on the new thin image. For understanding the end points and junction points on a hand-written image refer Figure 2.

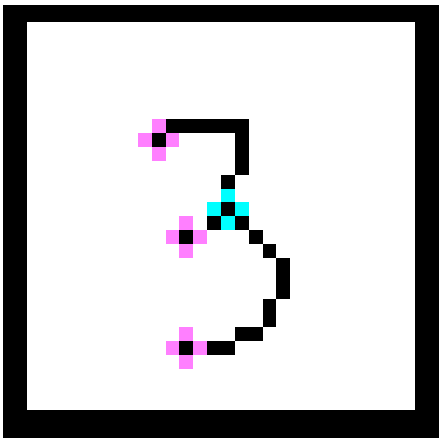


Figure 2: Pink mark – End points & Blue mark – Junction point

After extracting features from the processed input, it is considered with the training dataset and at the last Classification and Recognition is performed. Therefore, the entire methodology is commencing with configuring the project, importing the dataset, defining the network architecture, building a TensorFlow graph wherein TensorFlow is the tensor, a data structure similar to an array or list and lastly Training & Testing.

6. Datasets and Calculations

We at first implemented our project by using MNIST dataset. MNIST is an abbreviation for Modified National Institute of Standards and Technology Dataset. MNIST is a simple computer vision dataset. It consists of 28x28 pixel images of handwritten digits. Since each image has 28x28 pixels, we get 28x28 array. Every MNIST data point, every image, can be thought of as an array of numbers describing how dark each pixel is. It has decreasing error rate with different classifiers and parameters and error rate with pre-processing techniques from 12% error rate with linear classifier to achieving 0.23% error rate.

- Overview of Support Vector Machine(SVM) –

One of the set to solve binary data classification problem is Support Vector Machine. SVM's are supervised learning models that can be trained on labeled data sets. By meaning labeled, the data sets contain data which belongs to either category of two given categories.

- Linear Classification –

  Aim of SVM – The main aim of SVM is to find the exact classifier that best splits the data. What it means by finding the exact split or the exact classifier is that, in the given data, the data points are scattered over the plane, which are not exactly on either side of the plane, hence there exists multiple ways by which

the data can be split into two different parts. SVM tries to find the best possible split between the 2 data points which divides the plane or data points into their specific category perfectly. Based on this particular split, SVM then classifies the new data into either of the data sets.

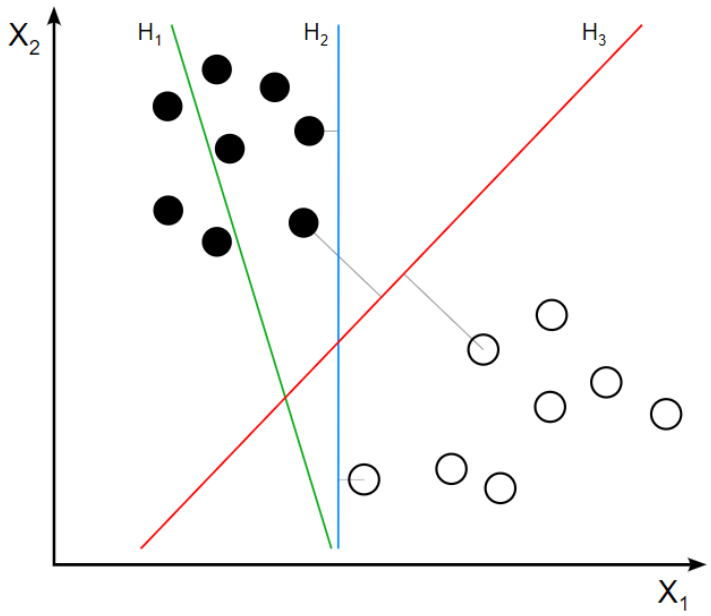


Figure 3: SVM classifier with multiple splits

As it can be seen in the figure above, there are two data sets let's say empty and filled circles. To divide these two data sets, there are multiple possible ways, which would be H1, H2 and H3. It is clear that H1 does not divide the plane in two data sets, H2 divides the plane into exactly 2 different sets and so does H3. Here, the major task of SVM is to select the one which gives a perfect split. Even though, H2 divides the data into two splits, H3 have the perfect distance from the first occurring points from both datasets. Hence, H3 is the prefect split here. The question here is, how does SVM decides H3 is the best split out of all 3.

Concept of Point, Line and Hyperplane – Since the beginning, mathematical formulations have been represented on the graphs so as to visualize the concepts and make them understand. When the formulation is in 1 dimension, the representation on graph is in the form of a Point. When the dimension is changed and increased to 2, i.e. 2 dimensions the representation is in the form of a Line. As the dimension's increase further to 3 or more, it is not possible to represent the formulation using points or lines. The term Hyperplane is introduced to represent such *n* dimensional formulations.

A hyperplane is defined as –

$$f(x) = \vec{w} \cdot \vec{x} + b = 0$$

Where, $\bar{w}$ is the vector normal to hyperplane and *b* is the bias. This representation is in terms of higher dimensions, to explain or to understand the

concept easily; the visualization can be done in a 2 dimensional plane. Let's assume $\bar{w}(-a,1)$ *and* $\bar{x}(x,y)$ –

$$\vec{w} \cdot \vec{x} = y - ax$$

$$\vec{w} \cdot \vec{x} - b = y - ax - b$$

Optimal Hyperplane –
For the points on one side of the hyperplane, let's assume the labelled with '+' would be –

$$f(x_+) = \vec{w} \cdot \vec{x_+} + b > 0$$

Here, what the optimal hyperplane is expected to give for the plus labelled points is –

$$f(x_+) = \vec{w} \cdot \vec{x_+} + b \geq \delta$$

By convention σ = 1. Here, the consideration is only for '+' labels. We can derive the same equation for two labels, let's say one is '+' and other is '-'.

$$\vec{w} \cdot \vec{x_+} + b \geq +1$$

$$\vec{w} \cdot \vec{x_-} + b \leq -1$$

Now, we know from the above equations that the hyperplane is divided into 2 parts, one is positive and other is negative. Coming back to the point where SVM had to select the perfect split.

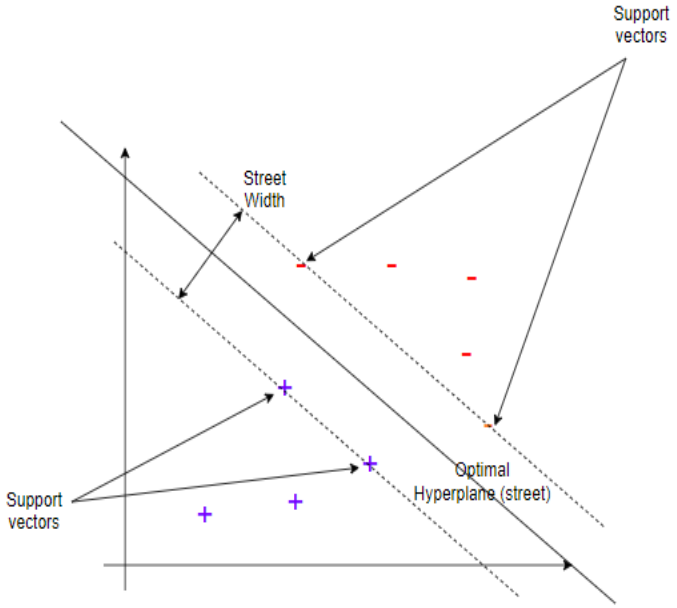


Figure 4: Hyperplane with two datasets and multiple splits

Here, '+' and '-' are the two support vectors and the 2 dotted lines are where the first points from the datasets are found and the solid line is the optimal hyper plane. The distance between the two dotted lines is called margin between the two datasets.
The aim here is to maximize the distance between the two dotted lines, i.e. to maximize the margin and still have the optimal hyperplane. By means of optimal, it means the hyperplane should have the same distance from both the starting points of the datasets. To do so mathematically, introduce a new variable y_i associated with all x_i taking the value +1 when x_i is of one label and −1 when it is of another then we can reduce the two equations above to –

$$y_i(\vec{x}_i \cdot \vec{w} + b) - 1 \geq 0$$

For two different support vectors, we will have two different equations –

$$W^t * x_1 + b = 1 \quad ....(1)$$
$$W^t * x_2 + b = -1 ..... (2)$$

Subtracting the above two equations, we get the following result –

$$W^t (x_1 - x_2) = 2$$

To simplify further, we would like to normalize the weight vectors. The reason for normalization is that one cannot directly divide the weight vectors, they have to be converted to scalars and then can be normalized.

$$\frac{wt}{||w||}(x1 - x2) = \frac{2}{||w||}$$

Such that,

$$Yi = \begin{cases} 1 & wtxi + b \geq 1 \\ -1 & wtxi + b \leq -1 \end{cases}$$

Although, the equation above fits for a particular situation, the real world scenario is pretty different, which is represented by the equation below –

$$Min\left(\frac{||w||}{2}\right) + c * \sum \sigma i$$

Where, *c* is the number of how many errors and σi is how much error

7. Execution and Output

```
In [*]: import classification_compelete
```

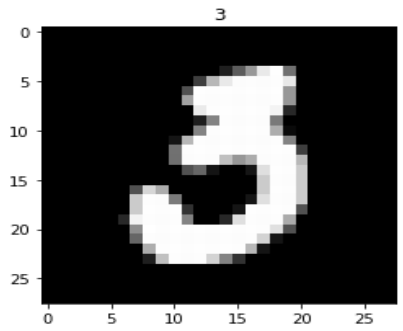
```
       label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0          3       0       0       0       0       0       0       0       0
1          0       0       0       0       0       0       0       0       0
2          2       0       0       0       0       0       0       0       0
3          4       0       0       0       0       0       0       0       0
4          9       0       0       0       0       0       0       0       0
...      ...     ...     ...     ...     ...     ...     ...     ...     ...
41995      9       0       0       0       0       0       0       0       0
41996      6       0       0       0       0       0       0       0       0
41997      9       0       0       0       0       0       0       0       0
41998      5       0       0       0       0       0       0       0       0
41999      4       0       0       0       0       0       0       0       0

       pixel8  ...  pixel774  pixel775  pixel776  pixel777  pixel778  \
0           0  ...         0         0         0         0         0
1           0  ...         0         0         0         0         0
2           0  ...         0         0         0         0         0
3           0  ...         0         0         0         0         0
4           0  ...         0         0         0         0         0
...       ...  ...       ...       ...       ...       ...       ...
41995       0  ...         0         0         0         0         0
41996       0  ...         0         0         0         0         0
41997       0  ...         0         0         0         0         0
41998       0  ...         0         0         0         0         0
41999       0  ...         0         0         0         0         0
```

Figure 5: Classification with MNIST Dataset 1

```
       pixel779  pixel780  pixel781  pixel782  pixel783
0             0         0         0         0         0
1             0         0         0         0         0
2             0         0         0         0         0
3             0         0         0         0         0
4             0         0         0         0         0
...         ...       ...       ...       ...       ...
41995         0         0         0         0         0
41996         0         0         0         0         0
41997         0         0         0         0         0
41998         0         0         0         0         0
41999         0         0         0         0         0

[42000 rows x 785 columns]
```




```
Fitting this might take some time .....
```

Figure 6: Classification with MNIST Dataset 2

```
In [2]: import svm_starter
```

```
       label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0          8       0       0       0       0       0       0       0       0
1          4       0       0       0       0       0       0       0       0
2          2       0       0       0       0       0       0       0       0
3          8       0       0       0       0       0       0       0       0
4          8       0       0       0       0       0       0       0       0
...      ...     ...     ...     ...     ...     ...     ...     ...     ...
41995      0       0       0       0       0       0       0       0       0
41996      6       0       0       0       0       0       0       0       0
41997      5       0       0       0       0       0       0       0       0
41998      7       0       0       0       0       0       0       0       0
41999      6       0       0       0       0       0       0       0       0

       pixel8  ...  pixel774  pixel775  pixel776  pixel777  pixel778  \
0           0  ...         0         0         0         0         0
1           0  ...         0         0         0         0         0
2           0  ...         0         0         0         0         0
3           0  ...         0         0         0         0         0
4           0  ...         0         0         0         0         0
...       ...  ...       ...       ...       ...       ...       ...
41995       0  ...         0         0         0         0         0
41996       0  ...         0         0         0         0         0
41997       0  ...         0         0         0         0         0
41998       0  ...         0         0         0         0         0
41999       0  ...         0         0         0         0         0
```

Figure 7: SVM with MNIST Dataset 1

```
       pixel779  pixel780  pixel781  pixel782  pixel783
0             0         0         0         0         0
1             0         0         0         0         0
2             0         0         0         0         0
3             0         0         0         0         0
4             0         0         0         0         0
...         ...       ...       ...       ...       ...
41995         0         0         0         0         0
41996         0         0         0         0         0
41997         0         0         0         0         0
41998         0         0         0         0         0
41999         0         0         0         0         0

[42000 rows x 785 columns]
Model score/accuracy is 1.0
```

Figure 8: SVM with MNIST Dataset 2

```
In [4]: import classification_compelete
```

```
     label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0        0       0       0       0       0       0       0       0       0
1        0       0       0       0       0       0       0       0       0
2        1       0       0       0       0       0       0       0       0
3        1       0       0       0       0       0       0       0       0
4        2       0       0       0       0       0       0       0       0
..     ...     ...     ...     ...     ...     ...     ...     ...     ...
158      2       0       0       0       0       0       0       0       0
159      3       0       0       0       0       0       0       0       0
160      1       0       0       0       0       0       0       0       0
161      1       0       0       0       0       0       0       0       0
162      0       0       0       0       0       0       0       0       0

     pixel8  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0         0  ...         0         0         0         0         0         0
1         0  ...         0         0         0         0         0         0
2         0  ...         0         0         0         0         0         0
3         0  ...         0         0         0         0         0         0
4         0  ...         0         0         0         0         0         0
..      ...  ...       ...       ...       ...       ...       ...       ...
158       0  ...         0         0         0         0         0         0
159       0  ...         0         0         0         0         0         0
160       0  ...         0         0         0         0         0         0
161       0  ...         0         0         0         0         0         0
162       0  ...         0         0         0         0         0         0
```

Figure 9: Classification with Handmade Dataset 1

```
     pixel780  pixel781  pixel782  pixel783
0           0         0         0         0
1           0         0         0         0
2           0         0         0         0
3           0         0         0         0
4           0         0         0         0
..        ...       ...       ...       ...
158         0         0         0         0
159         0         0         0         0
160         0         0         0         0
161         0         0         0         0
162         0         0         0         0

[163 rows x 785 columns]
```

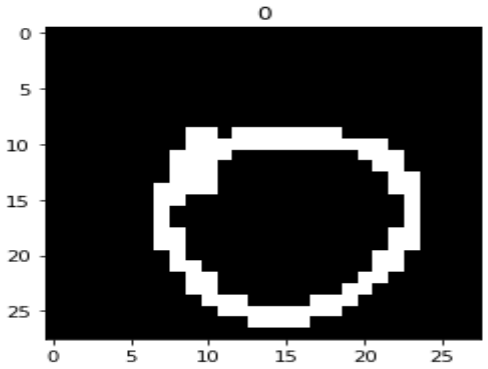
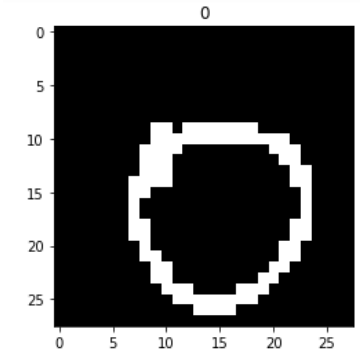


Figure 10: Classification with Handmade Dataset 2




```
Fitting this might take some time .....
predicting .....
Getting Accuracy .....
Score 0.9877300613496932
```

Figure 11: Classification with Handmade Dataset 3

Figure 12: Output of Paint Input Digit

Figure 13: Output of Predictor Screen Capture Input

```
In [2]: import predictor_live

        Prediction:  3
        Prediction:  3
```

Figure 14: Output of the Prediction of Digit

8. Future Work

The future steps that to go for would be having a closer look at the results of all the versions in order to find new rules. By extracting and implementing them, we will be able to enhance the performance of these versions. Moreover, it would be good if we could make some modifications to both the reference set and the rules in order to make our program more general and able to identify both typed and handwritten digits. Furthermore, in the future, we could make great use of the matrices that indicate the first maximum overlap of each test image with the reference images, along with the number of pixels left out from both. These matrices could be used with some clustering algorithms to build a program able to recognize handwritten digits with very high efficiency. Last but not least, we thought about using linear or high-level regression in the versions we have developed in order to create more rules. As regression could be used for binary classification and is not very suitable to classify a digit out of ten, this technique could be used in order to tell which digit is the most suitable, the first maximum or second maximum, which will enable us to generate more rules; thus, reach higher efficiency. We can also extend the model to work on NIST dataset. The accuracy can be increased further by implementing more number of hidden layers and/or epochs.

9. Conclusion

The hand written digit recognizer using SVM classifier has proved to be of a fairly good efficiency. It works almost similar to other classifiers. Thus, by understanding the working and most appropriately the neural networks, we have more control over the execution of the system. With the help of such labelled data, the model is trained and then tested by giving an unlabeled input, which is labelled by the model based on its training experience. The representation of objects is usually in the form of numerical vectors, but other representations are also possible. A clever use of this algorithm is to solve multiple class problems by breaking down the multiple class problems into two class problems. In other words, the problem is solved in a divide and rule fashion.

10.References

1. P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings., Edinburgh, UK, 2003, pp. 958-963. T Siva Ajay, “Handwritten Digit Recognition Using Convolutional Neural Networks”
2. International Research Journal of Engineering and Technology (IRJET) Volume: 04 Issue: 07 | July -2017
3. Wu, Ming & Zhang, Zhen. (2019). Handwritten Digit Classification using the MNIST Data Set
4. Al-Wzwazy, Haider& M Albehadili, Hayder & Alwan, Younes& Islam, Naz & E Student, M &, Usa. (2016). Handwritten Digit Recognition Using Convolutional Neural Networks. International Journal of Innovative Research in Computer and Communication Engineering. 4. 1101-1106
5. AL-Mansoori, Saeed. (2015). Intelligent Handwritten Digit Recognition using Artificial Neural Network. 10.13140/RG.2.1.2466.0649
6. Yang, Xuan Jiang. “MDig : Multi-digit Recognition using Convolutional Neural Network on Mobile.” (2015)
7. Srivastava, Shobhit & Kalani, Sanjana& Hani, Umme& Chakraborty, Sayak. (2017). Recognition of Handwritten Digits using Machine Learning Techniques. International Journal of Engineering Research and. V6. 10.17577/IJERTV6IS050456
8. Parveen Kumar, Nitin Sharma, and Arun Rana. Article: Handwritten Character Recognition using Different Kernel-based SVM Classifier and MLP Neural Network (A COMPARISON). International Journal of Computer Applications 53(11):25-31, September 2012
9. https://www.researchgate.net/publication/220030157_AN_EFFICIENT_FEATURE_EXTRACTION_AND_CLASSIFICATION_OF_HANDWRITTEN_DIGITS_USING_NEURAL_NETWORKS