Optical Character Recognition using Machine Learning

AbhilashRejanair,Anand C. Mathew and Dr. Sarojadevi H

Department of Computer Science and Engineering
NitteMeenakshi Institute of Technology
Bangalore – 560064

Abstract : The aim of this paper is to use image processing and machine learning concepts to detect patterns in the input image data, from which we try to identify and classify the pattern and thereby transform the input image into its machine understandable counterpart. The input is a real world unfilteredimage which is then processed to extract features using image processing techniques. The extracted features are classified using machine learning algorithms, which are previously trained using a standard dataset. Out of the several algorithms experimented, Neural networks and SVM are found to perform the best.

1. INTRODUCTION

Machine learning, as the name implies, is a system which unlike conventionally programmed systems, has the ability to “learn” dynamically without being programmed explicitly. Machine learning systems are similar to data mining systems in the fact that they search through data to look for, obtain, extract and categorize data by their differing patterns. Applying machine learning algorithms for image processing applications can produce clear advantages.

This paper presentsimage processing capabilities, combined with machine learning algorithms to detect patterns in the data from which we successfully identify and classify the said pattern. The work presentedis an application of the classification problem of machine learning, which is done by a supervised learning process, as well as the application of image processing algorithms like segmentation, thresholding and contour detection. Data recognition and classification is done by usingdifferent machine learning algorithmsand a comparative analysis is carried out.

2. DESIGN

The designinvolves major stages as shown in figure 1.

Image Acquisition
This is the first and the basic step of digital image processing. It is as simple as being given an image, which is already in digital form. The image taken is acquired by the computer as the input device. In our approach we have also taken images from MNIST data for systematic analysis.

Image Enhancement
The next phase is the image enhancement,which highlights certain features of interest in an image, like brightness, contrast, etc.

Image Restoration
The next phase is the image restoration. This phase deals with improving the appearance of the image. The restoration is done based in mathematical probabilistic models.

Compression
This phase deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. It is very much necessary to compress data.

Segmentation
This phase partitions an image into its constituent parts or objects. This phase is the most difficult phase of all. To extract information from the image, it is required it identify the objects individually.

Representation
This phase follows the segmentation phase. The output of the segmentation phase is usually a raw pixel data that includes either the boundary of a region or all the points in the region. This raw data should be translated into a form that can be processed by the computer. This is handled by the representation phase.

Object Recognition
This phase is the process of assigning a label to an object based on its descriptors. We use a machine algorithm to recognize the objects.

The first step to image processing is determining the intensity of the pixel**.**
Pixel Intensity
Pixel intensity is the brightness of the pixels present in the image.

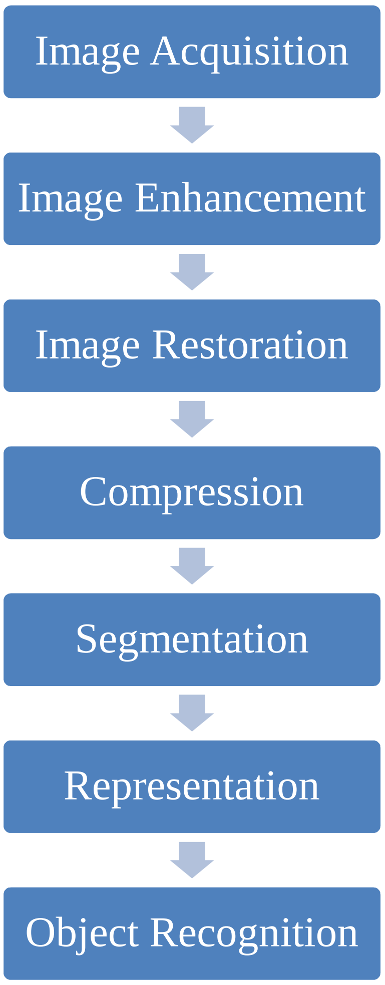


Fig 1. Design Flow

We need to determine the pixel intensity in order to pre-process it. Pixel intensity for optical character recognition is determined usingthe below mentioned formula using the R,G & B color component values.

Pixel intensity = 0.30R + 0.59G + 0.11B

Once we have calculated the pixel intensity, it becomes easier to determine the edges. To determine the edges we use Canny edge detection algorithm. On applying Canny edge detection on image A we get the image B.

3. EDGE BASED CONNECTED COMPONENT

For analyzing given image, we have to binarize it so as to extract related information, detect data patterns statistically, and perform operations on it. This is done by an edge-based connected components approach. This approach is used for detecting the threshold for each component, which can then be used to differentiate the character (foreground) from the background in the case of OCR (Optical Character Recognition) application. Edge detection is performed on each channel (R, G and B) and is combined using logical OR to determine the actual edge from which an edge box is drawn. The edge box is drawn in such a way as to eliminate the boxes with minimal or small area to avoid false detection.

To create an edge based connected component, the first step is to detect the edges. This is done by the Canny edge detection algorithm.

3.1. Canny Edge Detection Algorithm

3.1.1. Development of Canny Algorithm

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny.

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Therefore an edge detection solution can be implemented in a wide range of situations.

3.1.2. Process of Canny Edge Detection Algorithm

The process of Canny edge detection involves the following 5 steps:

1. Apply Gaussian filter to smoothen the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply non-maximum suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

3.1.2.1. Gaussian Filter

All the resultant images from the edge detection are affected by noise. Thus, it is very essential to remove or eliminate the noise. This can be done by applying the Gaussian filter.

The Gaussian filter smoothens the image to get rid of the noise from the digital image. The formula for applying the Gaussian filter is:

$$G_\sigma = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The equation is applied in the vertical as well as in the horizontal direction, which is indicated by$(x^2 + y^2)$ in the above formula. Once applied, the formula produces a surface whose contours are concentric circles. These values are then used for generating a convolution matrix, which is applied to the original image.

3.1.2.2. Intensity Gradient

In an image, the edge may point to a variety of directions. So the Canny edge detection algorithm basically uses four filters to detect the horizontal, vertical and the diagonal edges in the blurred image.

$$G = \sqrt{({G_x}^2 + {G_y}^2)}$$

The edge detection operator, (in this case Sobel) returns a value for the first derivative in the horizontal direction (G_x) and vertical direction (G_y).

3.1.2.3. Non Maximum Suppression

This is an edge thinning technique. Non-Maximum suppression is applied to "thin" the edge. After applying gradient intensity calculation, the edge extracted from the gradient value is still quite blurred. Thus non-maximum suppression can help to suppress all the gradient values to 0 except the local maxima, which indicates location with the sharpest change of intensity value. Continuing the process with thick edges becomes a problem, as it may result in a faulty edge. Thus, it is essential to suppress those thick edges by finding out the brightest pixel among the edges in the same direction. The algorithm for each pixel in the gradient image is as below.

1. Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient directions.
2. If the edge strength of the current pixel is the largest compared to the other pixels in the mask with the same direction (i.e., the pixel that is pointing in the y direction, it will be compared to the pixel above and below it in the vertical axis), the value will be preserved. Otherwise, the value will be suppressed.

3.1.2.4. Double Thresholding

After the application of non-maximum suppression, the edges that remain provide an accurate representation of real edges. However few edges remain, that are the outcome of noise and variation in colour. As a result these spurious edges are eliminated. This is done by double thresholding. It is important to retain the edge pixel with high gradient value and remove the ones with the low gradient value. This is accomplished by selecting high threshold and low threshold values. If the gradient value of an edge pixel is higher than the high threshold, it is marked as a strong edge pixel. If the gradient value of an edge pixel is smaller than the high threshold and larger than the low threshold, it is marked as a weak edge pixel. If an edge pixel's value is smaller than the low threshold value, it will be suppressed. The two threshold values are analytically determined and their definition will depend on the content of a given input image.

3.1.2.5. Edge Tracking by Hysteresis

So far, the strong edge pixels should certainly be involved in the final edge image, as they are extracted from the true edges in the image. However, there will be some debate on the weak edge pixels, as these pixels can either be extracted from the true edge, or the noise/color variations. To achieve an accurate result, the weak edges caused by the latter reasons should be removed. Usually a weak edge pixel caused from true edges will be connected to a strong edge pixel while noise responses are unconnected. To track the edge connection, blob analysis is applied by looking at a weak edge pixel and its 8 connected neighborhood pixels. As long as there is one strong edge pixel that is involved in the blob, that weak edge point can be identified as one that should be preserved.

The approach used in this work is the edge based connected component analysis. The reason for choosing this approach is as follows. This approach is used to detect the threshold for each component, which can then be used to differentiate the character (foreground) from the background. Edge detection is performed on each channel (R, G and B) and is combined to using logical OR to determine the actual edge from which, an edge box is drawn. Canny edge detection is performed individually on each channel of the colour image and the edge map E is obtained by combining the three edge images as follows.

$$E = E\mathit{R} \vee E\mathit{G} \vee E\mathit{B}$$

Here, E*R*, E*G* and E*B* are the edge images corresponding to the three colour channels and ∨denotes the logical OR operation.

An 8-connected component labelling, follows the edge detection step and the associated bounding box information is computed. We call each component, thus obtained, an edge-box (EB). We make some sensible assumptions about the document and use the area and the aspect ratios of the EBs to filter out the obvious non-text regions. The aspect ratio is constrained to lie in between 0.1 and 10 in order to eliminate elongated regions. The size of the EB should be greater than 15 pixels but smaller than 1/5th of the image dimension to be considered for further processing.

Since the edge detection captures both the inner and outer boundaries of the characters, it is possible that an EB may completely enclose one or more EBs as illustrated in Fig. 2.

We have computed edge boxes forEnglish alphabets and numerals. Some characters have two edge boxes. There are not more than two edge boxes on any character.

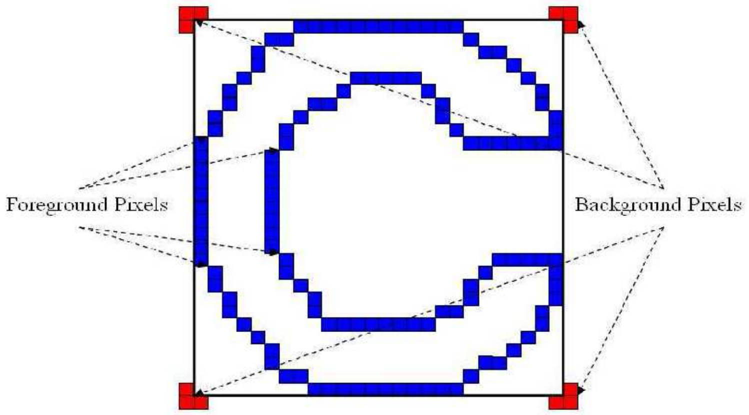
Edge boundary is determined as shown in Fig.2 which gives rise to two components. One is due to the inner boundary. Let's name it as EB(in). The other is due to outer boundary. Let's name this as EB(out). If an EB has exactly one or two EBs inside it, then those inside EBs can be neglected or ignored. On the other hand, if an EB encloses more than three EBs then the external EB(out) is ignored.
Thus, the unwanted components are eliminated by subjecting each edge component to the following constraints:
if (Nint<3)
{Reject EB(in) Accept EB(out)}
else
{Reject EB(out), Accept EB(in)}

where EB(*in)* denotes the EBs that lie completely inside the current EB under consideration and N*(in)* is the number of EB(*in)*. These constraints on the edge components effectively remove the obvious non-text elements while retaining all the text-like elements. Only the filtered set of EBs are considered for binarization.

The choice of window size in local methods can severely affect the result of binarization and may give rise to broken characters and voids, if the characters are thicker than the size of the window considered. Moreover, we often encounter text of different colours in a document image. Conventional methods assume that the polarity of the foreground-background intensity is known a priori. Images have multicolour textual content and varying background shades. A conventional binarization technique, using a fixed foreground background polarity, treats some characters as background, leading to the loss of some textual information generally assumed to be either bright on a dark background or vice versa. If the polarity of the foreground background intensity is not known, the binary decision logic could treat some text as background and no further processing can be done on those text characters.

A simple decision logic is to invert the result of binarization based on the assumption that the background pixels far outnumber the text pixels. Within each window, the number of pixels having intensity values higher or lower than the threshold are counted and the one which is less in number is treated as the foreground text. This simple inversion logic cannot handle the case where the characters are thick and occupy a significant area of the window under consideration.

Binarization using a single threshold on such images, without a priori information of the polarity of foreground-background intensities, will lead to loss of textual information as some of the text may be assigned as background. The characters once lost cannot be retrieved back and



are not available for further processing. Possible solutions need to be sought to overcome this drawback so that any type of document could be properly binarized without the loss of textual information.

Fig. 2. Edge Boundary

After pre-processing we get an image, with foreground as black and background as white. In other words, the text or any characters will be in black colour, the remaining regions are in white colour.

4. IMPLEMENTATION

The flow chart in fig.3 shows the design flow of the process or technique used in the work. Starting from giving the handwritten image as the input, all the way up to resizing the image to 28x28 size dimension is part of image processing of this work. The important algorithm that the above process utilizes is the Canny edge detection algorithm which is already explained.The whole of image processing used in this work sums up to 5 phases listed below. These process steps are shown in fig.3 and detailed below.

I. Input the image
II. Binarize the image
III. Invert the image
IV. Create bounding box on the characters
V. Extract the bounding box to a 28X28 sized

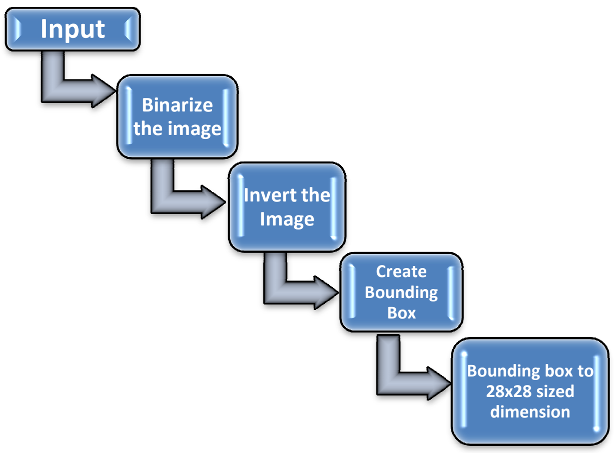


Figure 3. Design Flow

I. Input the image

An user can write anything that he wishes the computer to recognize on a paper. Then a snapshot of the hand written text is taken from a phone and given as input to the computer to process the image in a format that is understood by the computer.

The figure 4 below is a sample input image written in hand. The text reads "WELCOME".

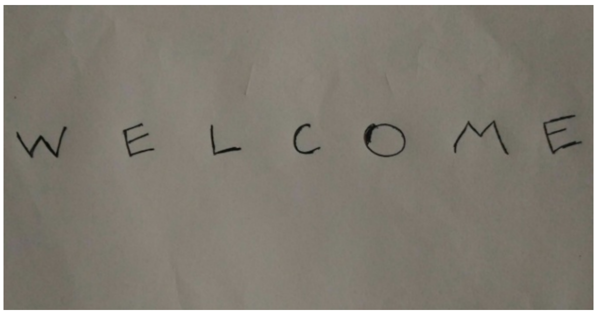


Fig.4. Input Image

II. Binarize the image

Once the input image is read, the process of edge based connected component approach is applied. The edge based connected component as discussed, starts with the Canny edge detection. It starts looking for the edges, so that it can draw edge boxes around the characters. The background is made white in colour and the foreground is made black in colour.The Canny edge detection algorithm uses the Gaussian filter to smoothen the image, as a result of which the noise is removed. Figure 5 shows a binarized image.

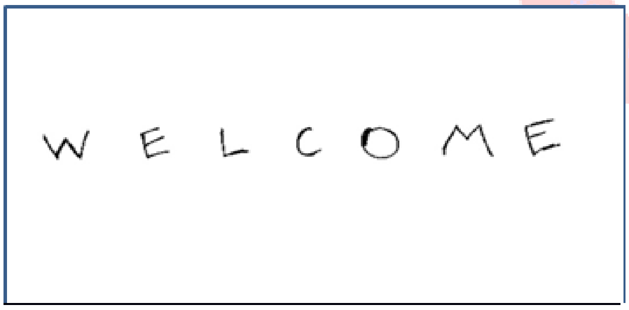


Fig.5. Binarized Image

III. Inverting the image

The next step that happens after binarizing the image is inverting the image. We invert the image to recognize the characters, we compare the input image to the data present in the Mnist data set. Mnist data set is a database that contains handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. The digits in the mnist database have been size-normalized and centered in a fixed-size format. The images are resized to a 28X28 sized images. The images are stored in terms of their respective pixel values. Therefore each image is represented by a total of 784 pixels. To match the binarized image with that of the MNIST database, we invert the image.

The figure 6 below shows the inverted image:

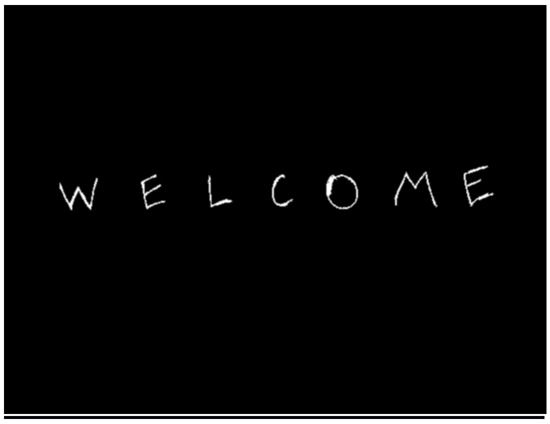


Fig.6. Inverted Image

IV. Creating Bounding Box

Once the image is inverted, each white character is bounded into a box. This is done so that each character is separated, from which it is later extracted. There is an openCV function that is used to find the contours. This function normally uses watershed algorithm. The Watershed algorithm is basically used to create the bounding box on characters that are very close to each other. This algorithm is time consuming. However there is also another algorithm that does not consume much time. This algorithm is called Suzuki algorithm and is used to create bounding boxes on characters that are far apart from each other.

The Suzuki algorithm initially has the start pixel. It moves along all the directions to see if there is a pixel next to the start pixel. If it exists, then the new pixel becomes the start pixel and the process continues.

The algorithm comes to an end, if it does not find any pixel next to the start pixel.

The figure 7 below shows the bounding box created on the characters.

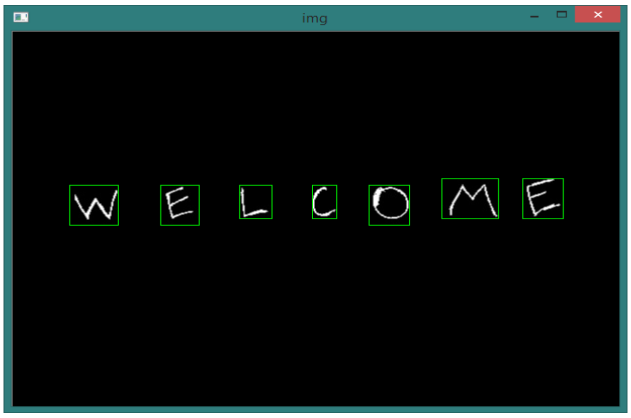


Fig. 7. Bounded Boxes

V. Extracting the bounding box to a 28X28 sized image

The bounding box then extracted to separate images of size-28x28. The images are resized to a 28x28 dimension because the MNIST database contains images of dimension 28x28. If not resized then the comparison will be incompatible. For example consider the letter "W" and "E" from the word "WELCOME". The two letters are resized to a 28x28 dimension.

The figure below shows the images resized to a 28x28 dimension.

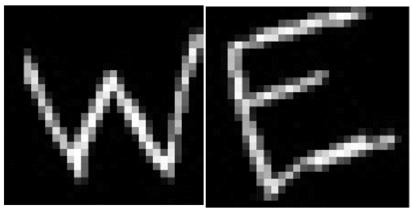


Fig.8. 28x28 Sized-Image

The above figure shows how the two letters look when they are resized to a 28x28 dimension. The step is not complete, without representing the 28x28 sized images to their respective pixel arrays. Each pixel is represented by 3 values. But we need to convert the pixel array to a list, such that each pixel is represented only by one value. This is done by taking the mean of the three numbers that represent a pixel. This process is computed for all the other pixels, thereby deriving a list of all the pixels, that are represented by one value. Hence the total number of value that will be present the list are 784 elements i.e. 28x28=784.

5. MACHINE LEARNING

Machine Learning is used extensively to predict the character by the algorithm. The machine learning algorithm is first trained by the MNIST dataset. It is then trained and then tested against a test dataset from which we can determine the confidence value or the accuracy of the machine learning algorithm for the data.

5.1. Dataset

To train the machine, we needed a dataset which has lots of variation between its data. The dataset also needs to be large enough to train the machine for it to work with data that is not from the dataset itself,also with the features extracted from the manually created images.

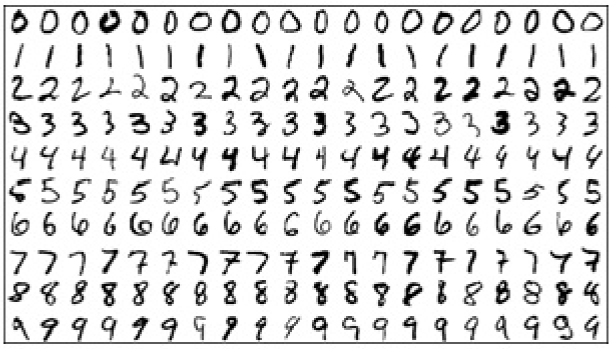


Fig.9. MNIST Dataset

We use the standard MNIST database of handwritten digits to train our machine learning algorithms. The MNIST database is a subset of the larger NIST database, but in MNIST, the sizes of the digits have been corrected and have been centered into the center of the image. This dataset is widely used for in many optical character recognition implementations to identify digits.

The dataset consists of a 28x28 size image which is available in a binary C-structure format. Each element of the dataset is a linear array of size 784 pixels, which are in greyscale.

MNIST provides 60,000 images in its training dataset and 10,000 different images in the testing dataset.

We do not require the full dataset of MNIST because of the large amount of time processing takes for different machine learning algorithms, so therefore, we take around 4000 elements from the training dataset and 400 elements from the testing dataset to determine the accuracy and various other parameters.

Since MNIST provides only digits, this project only deals with the recognition of digits and not, say, alphabet characters, special characters, punctuation, etc.

5.2. Machine Learning Algorithms

The following machine learning algorithms are used in the project

5.2.1.Linear Regression

Simple Linear Regression is the approach of determining relationship between a dependent variable and an independent variable using a linear function of the independent variable. On training the machine with linear regression, it was noticed that it is impossible for linear regression algorithm to fit the data and therefore, cannot be used in this use-case of optical character recognition.

5.2.2.Logistic Regression

Logistic Regression is the analysis of relationship between a binary dependent variable and independent variable. It is used mainly in classification machine learning problems. The response of the data is binary and it is used to estimate the probability of a certain state, such as 0 or 1, pass or fail, etc. Compared to the more complex machine learning algorithms available, logistic regression is simple to implement.

The following equation denotes the logistic function that denotes the probability of that an element belongs to a particular class.

We can also determine the probability of it not belonging to a class by subtracting the above sigmoid function by 1.

5.2.3. K Nearest Neighbor

K Nearest Neighbor is a simple machine learning algorithm used for machine learning classification problems. This algorithm involves identifying the neighbors of the given element so as to identify its position and then determine or predict the classification of that given element. The distance metric used to find the neighbors is the Euclidean distance formula (or sometimes also the hamming distance).

In this implementation, Euclidean distance is used given by the following formula.

$$\boldsymbol{d} = \sqrt{(\boldsymbol{x}-\boldsymbol{a})^2 + (\boldsymbol{x}-\boldsymbol{b})^2}$$

K Nearest Neighbor classifier works extremely well for this particular problem of optical character recognition, because since there is no co-relation or any connection among the various pixels in the same image, it makes it easy for k nearest algorithm to simply classify the data based on the position of each pixels during the training set.

To predict the classification of an image, we first take the new data and we scatter plot it on the graph. We compare all the nearest neighbors of the give new point, as denoted by the value of k provided. We have tried various values of k and have found that the value of k at 5 performs optimally considering the time taken to train the machine as well as the size in memory.

5.2.4. Support Vector Machines

Support Vector Machines is a machine learning model that is used for classification problems. This model assigns data to different categories depending on its values and then determines or predicts new examples non-probabilistically. Different categories of plane exist in their own dimension as determined by the algorithm. A plane called the hyper-plane is used to separate between these dimensions linearly, which is why SVM is called a linear classifier.

SVMs are not based on probability, therefore, they gather a better result in the problem of optical character recognition. Given the 10 different classes of the problem (that is, from 0 to 9 of the numeric digits), the hyperplane is constructed in an infinite dimension space, which is used as a multi-classifier to distinguish the different digits.

The SVM implemented is based on the well-known *libsvm* library implementation. Universally, SVMs are widely used for machine learning applications for images such as optical character recognition, image segmentation, etc.

5.2.5. Decision Trees

A decision tree is similar to the tree data structure. It consists of the root, branches and many hierarchical nodes. Each node depicts a decision or an attribute and each branch denotes the outcome. This concept is used in machine learning to obtain a process called decision tree learning.

Decision trees in general are easy to interpret and visualize when compared to other algorithms. The data for decision trees also require less preparing or pre-processing such as normalization or scaling, etc.

However, decision trees are also prone to be over-fitted, which causes it to reduce its accuracy score. There are also chances that depending on certain data, a bias can be created, which causes further inputs to not be trained successfully. To solve this issue, other machine learning algorithms based on the decision trees such as Random Forest and Extra Trees are used.

5.2.6. Random Forest

Random Forest is a machine learning method based on decision trees. It works by constructing multiple decision trees and then by selectively providing training data to the decision trees, arrives at the output by finding the mean or the average of the individually created smaller decision trees.

In Random Forest, the entire data is not taken immediately, but subsets of the data are selected and then a decision tree is constructed based on the subset data. Many such trees are created and the final output is based on the aggregate of such subset data created decision trees.

Using subsets instead of the whole data helps in removing the bias created by a complete decision tree.

It also helps to prevent over fitting, where the machine tries to fit the data more than it is needed, thereby reducing its performance and value.

5.2.7. Extra Trees

Extra trees is a variantof Random Forest machine learning algorithm method. In this method, the decision boundaries are picked randomly instead of the best boundary available.

The changes in Extra Trees compared to Random Forest are minimal, expect for one important difference. In Extra Trees, the samples are drawn randomly similar to Random

Forest, but the sample is from the entire training set rather than from a subset, as in Random Forests.

The accuracy and performance of Extra Trees is very similar to that of Random Forest, though it performs slightly better.

5.2.8. Gaussian Naïve Bayes

Naïve Bayes machine learning algorithm is based on Bayes probability theorem. The decision boundaries are made based on probability. Gaussian is a Naïve Bayes machine learning model which uses the Gaussian model for the classification part of the algorithm. Bayes Probability theorem, on which this algorithm is based on, is given by the equation

$$P(y \mid x_1 \dots x_n) = \frac{P(y)\, P(x_1 \dots x_n \mid y)}{P(x_1 \dots x_n)}$$

We obtain the Naïve Bayes classifier by applying the Naïve or independence assumption to the Bayes theorem. This assumption denotes that there is anstatistical independence or no co-dependency between the different features of the dataset. To obtain the Gaussian Naïve Bayes, we operate under the assumption that the data is continuous or non-discreet in nature.

Gaussian Naïve Bayes is a probabilistic model, which is why the performance of this machine learning algorithm with the dataset is sub-optimal when compared to the other machine learning algorithms.

5.2.9. Voting Classifier

Voting classifier is a weighted majority algorithm classifier. It is an algorithm which is created from the pre-existing machine learning algorithms. The output is determined by the weight given to each of them and the higher voted output is selected. If a wrong prediction is made, then the vote of that algorithm is reduced.

While Voting Classifier is not a standalone machine learning classification algorithm on its own, it runs various other machine learning classifiers given to it and then votes based on the output obtained. In this implementation of Voting classifier, “hard” voting is used, where the final output class label is obtained from the most frequently produced class labels.This is different compared to “soft” voting where probability is used to obtain the class label.

5.2.10. Artificial Neural Network

An artificial neural network is made up of multiple layers with connections in between them similar to a biological neural network. The first layer is the input layer and the last is the output layer.The layers in between are called the hidden layers. Based on the features sent to the input layer, the layers process the features to identify and classify the data to the output layer.

The signal path of a neural network is from the input layer, to the hidden layers and then finally to the output layers. The path cannot be traced back to the previous layer. Each layer consolidates features from the input and processes it, serving as an input to the next layer of nodes.In our implementation of neural networks, we created three layers of hidden nodes with 100, 50 and 25 nodes for each layer. We found this the most optimal in regards to performance, time taken and accuracy obtained.

6. EVALUATION OF MACHINE LEARNING ALGORITHMS

Out of the many machine learning algorithms implemented, to know which one performs the best under different evaluation criteria, an evaluation is done. The values obtained are dependent on the implementation as well as the hardware and software of the platform which executes the algorithms.

6.1. Size in Memory

Different algorithms consume different sizes in the memory of the computer. Algorithms which take more space usually have good performance, whereas those which occupy small space have relatively less performance. This space-time tradeoff has certain exceptions.

To measure the space taken by the program in memory, we first serialize the trained machine into a file and then measure the size. This is done in order to avoid inaccurate readings of memory during the runtime execution of an algorithm as they are highly implementation and platform specific.Table 1listsmemory consumed by each algorithm.

Algorithm	Size in kB
Logistic Regression	62.1
K Nearest Neighbor	27000
Support Vector Machines	8900
Decision Trees	124.1
Random Forest	1400
Extra Trees	2800
Gaussian Naïve Bayes	123.2
Voting Classifier	63100
Neural Network	2000

[Table 1] Size Comparison

It is observed that K Nearest neighbor algorithm takes the most space in memory since it contains the position of all elements on a graph so as to find the nearest neighbor. As the value of k increases, the size also increases. Logistic

Regression takes the least memory, next to which isGaussian Naïve Bayes, but the former offers much better scores than the latter.

6.2. Time Taken to Train Machine

Various algorithms take different times to train their machines. Although this does not determine the time taken while predicting, it is useful to note this during development purposes with a large amount of data since a very long time taken to train machines may prove counter-intuitive.

Algorithm	Time in seconds
Logistic Regression	34.2
K Nearest Neighbor	0.164
Support Vector Machines	3.78
Decision Trees	0.707
Random Forest	0.42
Extra Trees	0.382
Gaussian Naïve Bayes	0.062
Voting Classifier	0.6101
Neural Network	26.13

[Table 2] Timing Comparison

From the table 2, we can infer an indirect proportionality between the space consumed and the time taken for execution. Logistic Regression takes the largest amount of time for training its machine and Gaussian Naïve Bayes takes the least time, though K nearest neighbor which comes next has a better prediction rate.

6.3. Accuracy or Prediction Scores

The accuracy or prediction scores of machine learning algorithms are the true measures of how successful a machine learning algorithm is. The higher the score, better is the algorithm.

We observed an average of accuracy in the high 80's to early 90's. However, a few algorithms felt short of such an accuracy, which indicates that these algorithms are not suitable for use for this particular application. Table 3 shows the accuracy of each machine learning algorithm for a dataset of 4000 trained elements by using a testing dataset of 400 elements.

Algorithm	Accuracy (%)
Logistic Regression	86
K Nearest Neighbor	90
Support Vector Machines	90.5
Decision Trees	72.5
Random Forest	87.25
Extra Trees	90.25
Gaussian Naïve Bayes	54.75
Voting Classifier	91.25
Neural Network	94.5

[Table 3]Accuracy comparison

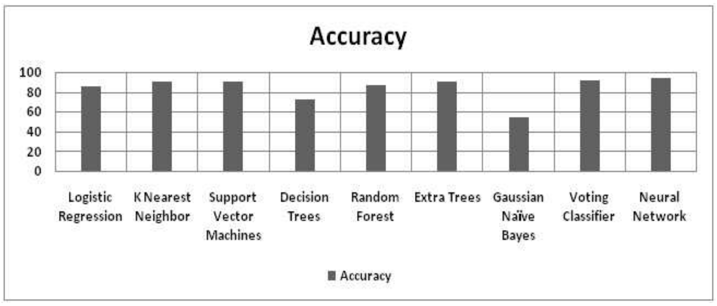


Fig.10. Accuracy of Machine learning algorithms

Graphs of comparative analysis of various algorithms are as in Fig.10. It can be inferred that neural network suits optical character recognition the most, while voting classifier is next, following which isthe support vector machines.

6.4. Change in dataset size

It is observed that with the change in the size of the training dataset, the accuracy values also change. The change is not consistent, that is, accuracy cannot be said to increase or decrease but depends on the dataset which is used.

7. CONCLUSION

The work presented in this paper involves preparing an image for the process of optical character recognition as well as training various classifier machines to process data and predict information and thereby combining them both to produce the optically recognized character.

Image processing accomplishes the need of isolating the data and optimizing the pixels of the image to make it ready to be used by the machine learning algorithms. The machine learning algorithms are trained and they are further checked to ensure that they can be reliably used for predictions.

The basic premise of this project was to successfully implement the optical character recognition system. However, since the dataset contained only digits, we arelimited to 10 classes. If a suitable dataset of all English characters is present, it could be implemented as well to obtain a real world OCR system with good accuracy.

The accuracy of certain algorithms can also be tweaked and improved by subtle variations in the kernels used, or gamma values, etc. A very impressive 94.5% accuracy is reached but we believe it can be further improved.

Future scope for this work is working on real time optical character recognition.

REFERENCES

1. Ivan Dervisevic, Machine Learning Methods for Optical Character Recognition. 2006

2. Machine Learning Final Project: Handwritten Sanskrit Recognition using a Multi-class SVM with K-NN Guidance [http://people.csail.mit.edu/yichangshih/mywebsite /sanskrit.pdf]
3. Number Plate recognition System using Matlab [http://www.slideshare.net/ NamraAfzal /number-plate-recognition-system-using-matlab]
4. Vanita Jain et. al., Comparative analysis of machine learning algorithms in OCR .IEEE 2016
5. J. Canny. A computational approach to edge detection. IEEE trans. PAMI, 8(6):679–698, 1986.
6. T Kasar, J Kumar and A G Ramakrishnan, Font and Background Color Independent Text Binarization.2007
7. Y.LeCun et.al., Gradient based Learning Applied to Document Recognition. IEEE 1998
8. Pierre Geurts et.al., Extremely Randomized Trees. Springer 2006.
9. Gareth James, Majority Vote Classifiers: Theory and Applications. Stanford Univ.Thesis 1998
10. Wernick, Yang, Brankov, Yourganov and Strother, Machine Learning in Medical Imaging, IEEE Signal Processing Magazine, vol. 27, no. 4, July 2010, pp. 25–38
11. Michie D., Spiegelhalter D. J., and Taylor C. C., Machine Learning, Neural and Statistical Classification. Ellis Horwood, 1994
12. Tinku Acharya and Ajoy K. Ray, Image Processing - Principles and Applications. Wiley InterScience, 2006
13. Wilhelm Burger and Mark J. Burge, Digital Image Processing: An Algorithmic Approach Using Java. 2008
14. Milan Sonka, Vaclav Hlavac and Roger Boyle, Image Processing, Analysis, and Machine Vision. PWS Publishing ,1999