A Project Report

on

HANDWRITTEN CHARACTER RECOGNITION

Submitted in partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

by

A. Tejaswini (16WH1A1201)

A. Niharika (16WH1A1204)

N. Pooja Sathvika (16WH1A1250)

Y. Lakshmi Pratyusha (16WH1A1254)

Under the esteemed guidance of

Mr. A. Rajashekar Reddy

Assistant Professor

Department of Information Technology

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Accredited by NBA & NAAC '**A**' Grade, Approved by AICTE, and

Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

April 2020

DECLARATION

We hereby declare that the work presented in this project entitled **"HANDWRITTEN CHARACTER RECOGNITION"** submitted towards completion of the major project in IV year of B.Tech IT at "BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN", Hyderabad is an authentic record of our original work carried out under the esteem guidance of Mr. A. Rajashekar Reddy, Assistant Professor, IT department.

A. Tejaswini
(16WH1A1201)

A. Niharika
(16WH1A1204)

N. Pooja Sathvika
(16WH1A1250)

Y. Lakshmi Pratyusha
(16WH1A1254)

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Accredited by NBA & NAAC 'A' Grade, Approved by AICTE, and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Department of Information Technology

Certificate

This is to certify that the Project report on "**Handwritten Character Recognition**" is a bonafide work carried out by **A. Tejaswini (16WH1A1201), A. Niharika (16WH1A1204), N. Pooja sathvika (16WH1A1250), Y. Lakshmi Pratyusha (16WH1A1254)** in the partial fulfillment for the award of B.Tech degree in **Information Technology, BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide
Mr. A. Rajashekar Reddy
Assistant Professor
Department of IT

Head of the Department
Dr. Aruna Rao S L
Professor and HOD
Department of IT

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K. V. N. Sunitha, Principal**, BVRIT Hyderabad, for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr**. **Aruna Rao S L, Head**, Dept. of IT, BVRIT Hyderabad for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Mr. A. Rajashekar Reddy, Assistant Professor, Department of IT,** BVRIT Hyderabad for his constant guidance, encouragement and moral support throughout the project.

Finally, We would also like to thank our Project Coordinator, all the faculty and staff of IT Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

A.Tejaswini
(16WH1A1201)
A. Niharika
(16WH1A1204)
N. Pooja Sathvika
(16WH1A1250)
Y. Lakshmi Pratyusha
(16WH1A1254)

ABSTRACT

Character recognition is one of the most important research fields of image processing and pattern recognition. Character recognition is generally known as Handwritten Character Recognition (HCR) or Optical Character Recognition (OCR). HCR is the process of electronic translation of handwritten images or typewritten text into machine editable text. It becomes very difficult if there are lots of paper based information on companies and offices. Because they want to manage huge volume of documents and records. Computers can work much faster and more efficiently than human. It is used to perform many of the tasks required for efficient document and Content management. But computer knows only alphanumeric characters as ASCII code. So computer cannot distinguish character or word from a scanned image. In order to use the computer for document management, it is required to retrieve alphanumeric information from a scanned image. There are so many methods which are currently used for OCR and are based on different languages. The existing method like Artificial Neural Network (ANN based on English Handwritten character recognition needs the features to be extracted and also the performance level is low. So Convolutional Neural Network (CNN) based English handwritten character recognition method is used as a deep machine learning method for which it doesn't want to extract the features and also a fast method for character recognition.

LIST OF FIGURES

LIST OF ABBREVATIONS

Term/Abbreviation	Definition
CNN	Convolutional Neural Network
OCR	Optical Character Recognition
EMNIST	Extended Modified National Institute of Standards and Technology
HCR	Handwritten Character Recognition
TPU	Tensor Processing Unit
GCP	Google Cloud Platform
ICA	Independent Component Analysis
SIFT	Scale Invariant Feature Extraction
RELU	Rectified Linear Unit

TABLE OF CONTENTS

1. INTRODUCTION

This project, 'Handwritten Character Recognition' is a software algorithm project to recognize any hand written character efficiently on computer with input is either an optical image or currently provided through touch input, mouse or pen. Character recognition, usually abbreviated to optical character recognition or shortened OCR, is the mechanical or electronic translation of images of handwritten, typewritten or printed text (usually captured by a scanner) into machine-editable text.

Optical Character Recognition uses the image processing technique to identify any character computer/typewriter printed or hand written. A lot of work has been done in this field. The time used in entering the data and also the storage space required by the documents can be highly reduced by the use of OCR or in other words it can be retrieved fast. OCR in advance can be inferred in two ways based on type of the text and document acquisition.

Many a times the writing style of same individual is different at times. Further OCR is characterized into two forms as Offline and Online recognition systems based on acquiring of the documents. Offline System deals with recognizing the pre written document acquired through various input methods. But in Online recognizing system, the writing is recognized the moment it is written. The device used for the online system is Electric pen where it is used for writing the letters or words on the device called as digitizer and on the basis of the pen movement the input is recorded.

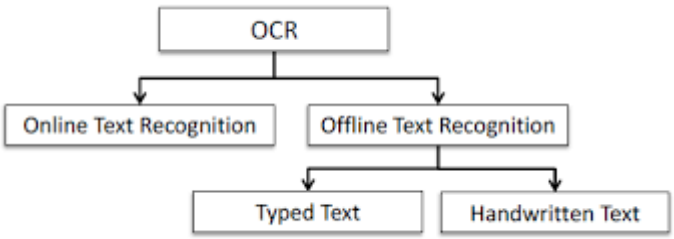


Figure: 1.0 OCR Types

1.1 OBJECTIVE

The objective of this project is to identify handwritten characters with the use of Convolution Neural Networks. We have to construct suitable neural network and train it properly. The program should be able to extract the characters one by one and map the target output for

training purpose. After processing of the image, the training dataset has to be used to train "classification engine" for recognition purpose.

1.2 DATASET DESCRIPTION

The dataset is taken from the EMNIST (Extended Modified National Institute of Standards and Technology) is a large database of handwritten digits and characters that is commonly used for training various image processing systems. It was created by "re-mixing" the samples from NIST's original datasets.

1.2.1 Dataset Summary

There are six different splits provided in this dataset. A short summary of the dataset is provided below:

- EMNIST ByClass: 814,255 characters. 62 unbalanced classes.
- EMNIST ByMerge: 814,255 characters. 47 unbalanced classes.
- EMNIST Balanced: 131,600 characters. 47 balanced classes.
- EMNIST Letters: 145,600 characters. 26 balanced classes.
- EMNIST Digits: 280,000 characters. 10 balanced classes.
- EMNIST MNIST: 70,000 characters. 10 balanced classes.

The full complement of the NIST Special Database 19 is available in the ByClass and ByMerge splits. The EMNIST Balanced dataset contains a set of characters with an equal number of samples per class. The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class task. The EMNIST Digits and EMNIST MNIST dataset provide balanced handwritten digit datasets directly compatible with the original MNIST dataset.

We used ByClass split of EMNIST dataset which contains 697, 932 training images and 116, 323 testing images. See figure: 1.2

1.3 PROBLEM IN EXISTING SYSTEM

Earlier character recognition techniques were not sophisticated enough for practical recognition of data handwritten on forms or documents.

Figure: 1.2 EMNIST Dataset.

1.4 FEATURES

We can change the performance type, performance goal, momentum, and learning rate values of the selected neural network. After altering the values, we have to retrain our network. The performance type can be Sum Squared Error and Mean Squared Error. The goal describes the minimal error value, at which the training will stop. Momentum and learning rate helps us get through local minimums easily, so we have to choose them wisely. Bigger values make training faster, but we can slip past the best minimum. Clicking the train button, we start the training of the network with the data described for the selected network and with the selected parameter values. If we have an image loaded, and something selected on it, clicking the recognize button, the recognition will start. The selected image part is cropped, it is transformed to binary image, then it gets cut to its edges, and finally it will be projected on the grid. This process is called feature extraction. The last step produces a vector. The elements of it show us how many features of the recognized character are at the appropriate grid part in an interval of 0 and 1. After this, the vector is put on the input of the selected net. In the output vector, we will have to check which element has the greatest value. With this we can conclude which number was recognized.

2. LITERATURE SURVEY

Character recognition originated as early as 1870 when Carey invented the retina scanner, which is an image transmission system using photocells. It is used as an aid to the visually handicapped by the Russian scientist Tyurin in 1900. The first generation machines are characterized by the "constrained" letter shapes which the OCRs can read. Next generation machines were able to recognize regular machine-printed and hand printed characters. The character set was limited to numerals and a few letters and symbols. Such machines appeared in the middle of 1960s to early 1970s. The methods were based on the structural analysis approach. Significant efforts for standardization were also made in this period. For the third generation of OCR systems, the challenges were documents of poor quality and large printed and hand-written character sets. Low cost and high performance were also important objectives. The fourth generation can be characterized by the OCR of complex documents intermixing with text, graphics, tables and mathematical symbols, unconstrained handwritten characters, color documents, low-quality noisy documents, etc. Among the commercial products, postal address readers, and reading aids for the blind are available in the market.
Nowadays, there is much motivation to provide computerized document analysis systems. OCR contributes to this progress by providing techniques to convert large volumes of data automatically. Various methods have been proposed to increase the accuracy of optical character recognizers. Thus, current active research areas in OCR include handwriting recognition, and also the printed typewritten version of non-Roman scripts (especially those with a very large number of characters).

2.1 METHODOLOGY

Our goal is to detect the character which is given as input, in whatever style the input text might be. In this project, we develop a model for handwritten character recognition. We also present the algorithms for recognizing the characters which is given as the input and give the correct output for the user. In this recognition process there will very less wrong recognition of characters. Instead it recognizes the input and gives the correct output for the user, with accuracy of 86.31% and in a less time while compared to existing once.

Off-line handwritten character recognition refers to the process of recognizing characters in a document that have been scanned from a surface such as a sheet of paper and are stored digitally in gray scale format. The storage of scanned documents have to be bulky in size and

many processing applications as searching for a content, editing, maintenance are either hard or impossible. Such documents require human beings to process them manually, for example, postman's manual processing for recognition and sorting of postal addresses and zip code. Character recognition systems translate such scanned images of printed, typewritten or handwritten documents into machine encoded text. This translated machine encoded text can be easily edited, searched and can be processed in many other ways according to requirements.

These features are used to train the system. The features extracted are given as input to the classification stage and the output of this stage is a recognized character. The selection of the combination of feature-classifier contributes to the performance of the system. Several research works has been focusing toward evolving such methods to reduce the processing time and providing higher recognition.

Character recognition follows the existing modules to develop the stages of the character recognition. They are

1. Drawing Text
2. Pre-processing
3. Segmentation
4. Feature extraction
5. Classification
6. Recognition

Let's discuss each and every module briefly; the above modules can be represented by the block diagram – figure: 2.1

1. Draw Text

In this module, the input is given through drawing a text, i.e., from the dataset particular character image is selected and we can check those text to particular character. The character is processed by the following modules.

2. Preprocessing

The uploaded image may have some noise. This noise may be due to some unnecessary information available in the image. Various steps involved in the preprocessing are Shown in figure: 2.1.1

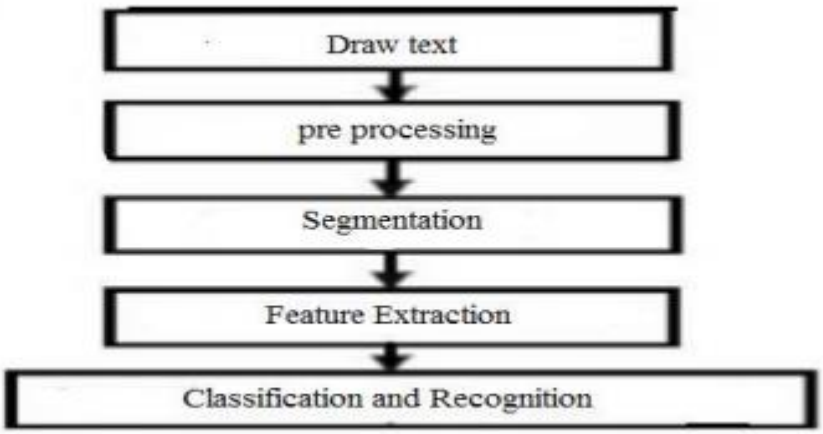


Figure: 2.1.0 Block Diagram of HCR

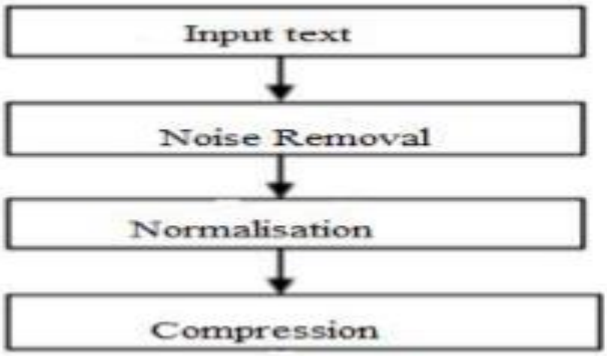


Figure: 2.1.1 Preprocessing Steps

The main objective of pre-processing is to de noise and enhance the quality of image.

2.1 Input Text

Here the input is given through the characters, each character can be given by the user and the character can be processed future steps as shown in the above figure.

2.2 Noise Removal

Optical scanning devices introduces some noises like, disconnected line segments, bumps and gaps in lines, filled loops etc. It is necessary to remove all these noise elements prior to the character recognition.

2.3 Normalization

The main component of the pre-processing stage is normalization, which attempts to remove some of the variations in the images, which do not affect the identity of the word. Handwritten image normalization from a scanned image includes several steps, which usually

begin with image cleaning, skew correction, line detection, slant and slope removal and character size normalization.

2.4 Compression

Space domain techniques are required for compression. Two important techniques are thresholding and thinning. Thresholding reduces the storage requirements and increases the speed of processing by converting the gray-scale or color images to binary image by taking a threshold value. Thinning extracts the shape information of the characters.

3. Segmentation

In the segmentation stage, an image consisting of a sequence of characters is decomposed into sub-images of individual characters. The main goal is to divide an image into parts that have a strong correlation with objects or areas of the real world contained in the image. Segmentation is very important for recognition system. Segmentation is an important stage because the extent one can reach in separation of words, lines, or characters directly affects the recognition rate of the script. Image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics. In Character Recognition techniques, the Segmentation is the most important process. Segmentation is done to make the separation between the individual characters. The segmentation stage takes in a page image and separates the different logical parts, like text from graphics, lines of a paragraph, and characters (or parts thereof) of a word. After the preprocessing stage, most HCR systems isolate the individual characters or strokes before recognizing them. Segmenting a page of text can be broken down into two levels: page decomposition and word segmentation, When working with pages that contain different object types like graphics, headings, mathematical formulas, and text blocks, page decomposition separates the different page elements, producing text blocks, lines, and sub-words. While page decomposition might identify sets of logical components of a page, word segmentation separates the characters of a sub-word. The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries in images. Script segmentation is done by executing the following operations: Line segmentation, Word segmentation and character segmentation.

4. Feature Extraction

Feature extraction is the method to retrieve the most significant data from the raw data. The main objective of feature extraction is to extract a set of features, which maximizes the recognition rate with the least amount of elements. Feature extraction is the heart of any pattern recognition application. Feature extraction techniques like Principle Component Analysis (PCA), Linear Discriminant Analysis (LDA), Independent Component Analysis (ICA), Chain Code (CC), Scale Invariant Feature Extraction (SIFT), zoning, Gradient based features, Histogram might be applied to extract the features of individual characters.

5. Classification and recognition

The classification phase is the decision making part of the recognition system. The performance of a classifier based on the quality of the features. This stage uses the features extracted in the previous stage to identify the character. When input is presented to HCR system, its features are extracted and given as an input to the trained classifier like artificial neural network or support vector machine. Classifiers compare the input feature with stored pattern and find out the best matching class for input

3. REQUIREMEMNT SPECIFICATION

Software requirements deal with software and hardware deals with resources that need to be installed on a serve which provides optimal functioning for the application. These software and hardware requirements need to be installed before the packages are installed. These are the most common set of requirements defined by any operation system. These software and hardware requirements provide a compatible support to the operation system in developing an application

3.1.1 SOFTWARE REQUIREMENTS

The software requirements specify the use of all required software products like data management system. The required software product specifies the numbers and version. Each interface specifies the purpose of the interfacing software as related to this software product.

To implement this project we will be using Keras which is a popular deep learning framework for Python and some additional libraries like TensorFlow, NumPy, openCV, Matplotlib, and Google Colab. Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser. Google Colab comes with collaboration backed in the product. In fact, it is a Jupyter notebook that leverages Google Docs collaboration features. It also runs on Google servers and you don't need to install anything. Moreover, the notebooks are saved to your Google Drive account. Google Colab runs on Google Cloud Platform (GCP). Hence it's robust, flexible, Google Colab recently added support for Tensor Processing Unit (TPU) apart from its existing GPU and CPU instances. Despite being so good at hardware, the services provided by Google Colab are completely free. This makes it even more awesome.

Operating System : Windows10 (64 bit)

IDE : Google's Collaboratory

Technologies : Python

3.1.2 HARDWARE REQUIREMENT

The hardware requirement specifies each interface of the software elements and the hardware elements of the system. These hardware requirements include configuration characteristics.

Operating System - Windows 8 or higher version is required

Hard Driver - Minimum 32Gb , Recommended 64Gb or more

Processor - Minimum intel core i5 CPU and 2.50 GHz processor is needed

Physical Memory - Minimum 8Gb RAM is required

3.2 COMPONENTS

- **TensorFlow –** It is an open-source toolkit that can be used to build machine learning pipelines so that you can build scalable systems to process data. It is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google

- **Keras –** It is an open-source neural network library that provides support for Python. It is popular for its modularity, speed and ease of use. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network mod

- **OpenCv (Open source computer vision) –** It is library of programming functions mainly aimed at real-time computer vision. It is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel. The library is cross-platform and free for use under the open-source BSD license.

    - OpenCV supports some models from deep learning frameworks like TensorFlow, Torch, PyTorch (after converting to an ONNX model) and Caffe according to a defined list of supported layers. It promotes

OpenVisionCapsules. , which is a portable format, compatible with all other formats.

- **NumPy -** NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with the arrays. It is the fundamental package for scientific computing with Python. It contains among other things:

    - a powerful N-dimensional array object
    - sophisticated (broadcasting) functions
    - tools for integrating C/C++ and Fortran code
    - useful linear algebra, Fourier transform, and random number capabilities

  Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

- **Matplotlib** The matplotlib.pyplo**t** is a plotting library used for two dimensional graphics in python programming language. It can be used in python scripts, shell, web application servers and other graphical user interface toolkits.

    - There are several toolkits which are available that extend python matplotlib functionality. Some of them are separate downloads, others can be shipped with the matplotlib source code but have external dependencies.

    - There are various plots which can be created using python matplotlib. Some of them are bar graph, histogram, scatter plot, area plot, pie plot

    - Using matplotlib the visualization of data can be done in a better way. Using this library publication quality plots can be developed with just a few lines of code and also createinteractive figures that can zoom, pan, update.

    - Customization by taking full control of line styles, font properties, axes properties is possible and also export and embed to a number of file formats and interactive environments. Exploinge tailored functionality provided bythird party packages is possible.

4. DESIGN OF THE SYSTEM

Using a fully connected neural network to make an image classification requires a large number of layers and neurons in the network, which increases the number of parameters leading the network to over-fitting (memorizing the training data only). The input image may also lose its pixels correlation properties since all neurons (carrying pixels values) are connected to each other. Convolutional neural networks have emerged to solve these problems through their kernel filters to extract main features of the input image and then inject them into a fully connected network to define the class.

The chosen architecture in our application is convolutional neural network. It contains 6 layers of convolution and simplification functions made by 5x5 kernel filters, Batch Normalization and a max pooling filter of 5x5 to reduce at last the input image of 28x28. The feature images carry most important features to define a specified age and gender classes by processing them into fully connected network.

In deep learning, a CNN is a class of deep neural network most commonly applied to analyzing visual imagery. The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product.

The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

A CNN typically has three layers:

1. Convolutional Layer,
2. Pooling Layer
3. Fully Connected Layer.

4.1 CONVOLUTIONAL LAYER

The convolution layer is the core building block of CNN. It carries the main portion of the network’s computational load. The main objective of convolution is to extract features such as edges, colors, corners from the input. As we go deeper inside the network, the network

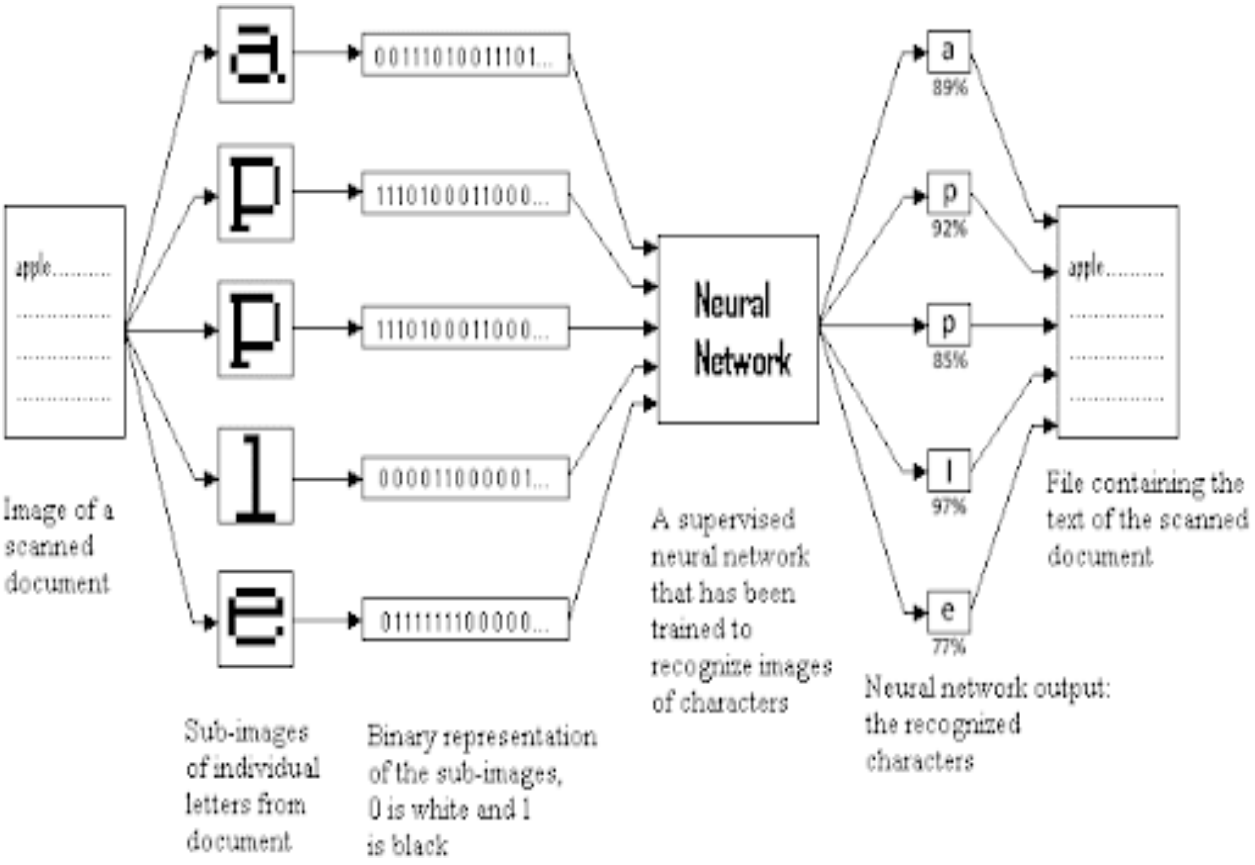


Figure 4.0: CNN Architecture

starts identifying more complex features such as shapes, digits, as well. This layer performs a dot product between two matrices, where one matrix (known as filter/kernel) is the set of learnable parameters, and the other matrix is the restricted portion of the image. If the image is RGB then the filter will have smaller height and width compared to the image but it will have the same depth (height x width x 3) as of the image. At the end of the convolution process, we have a featured matrix which has lesser parameters (dimensions) than the actual image as well as more clear features than the actual one.

4.2 POOLING LAYER

This layer is solely to decrease the computational power required to process the data. It is done by decreasing the dimensions of the featured matrix even more. In this layer, we try to extract the dominant features from a restricted amount of neighborhood. The of pooling technique is MAX-pooling. In MAX-pooling, we just take the maximum amongst all the

values lying inside the pooling region. So, after pooling layer, we have a matrix containing main features of the image and this matrix has even lesser dimensions.

4.3 FULLY CONNECTED LAYER

Now that we have converted our input image into a suitable form for our Multi-Level fully connected architecture, we shall flatten the image into one column vector. The flattened output is fed to a feed-forward neural network and back propagation applied to every iteration of training. Over a series of epochs, the model can distinguish between dominating and certain low-level features in images and classify them.

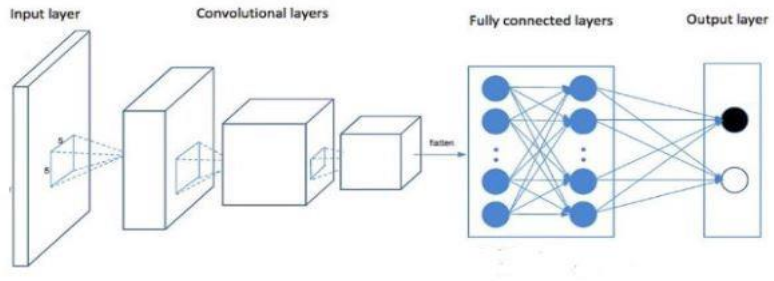


Figure 4.3: Fully connected layer inside CNN

A part from these layers we also have additional layers that we will using for full implement of our project and for getting better accuracy.

Dropout layer is added. This is a method for regularizing our model in order to prevent over fitting.

MaxPooling is a way to reduce the number of parameters in our model by sliding a 2x2 pooling filter across the previous layer and taking the max of the 4 values in the 2x2 filter.

A **dense layer** is just a regular **layer** of neurons in a neural network. Each neuron recieves input from all the neurons in the previous **layer**, thus densely connected. The **layer** has a weight matrix W, a bias vector b, and the activations of previous **layer** a.

5. MODULES

5.1 PREPROCESSING THE DATA

The raw data is subjected to a number of preliminary processing steps to make it usable in the descriptive stages of character analysis. Pre-processing aims to produce data that are easy for the handwritten character recognition systems to operate accurately.

Here we applied normalization technique which is shown in the figure: 5.0. In image processing, **normalization** is a process that changes the range of pixel intensity values. Applications include photographs with poor contrast due to glare, for example. Normalization is sometimes called contrast stretching or histogram stretching. In more general fields of data processing, such as digital signal processing, it is referred to as dynamic range expansion.

```
X_train = np.array(X_train) / 255.0
y_train = np.array(y_train)
X_test = np.array(X_test) / 255.0
y_test = np.array(y_test)
```

we need to **reshape** our dataset inputs (X_train and X_test) to the shape that our model expects when we train the model. The first number is the number of images (60,000 for X_train and 10,000 for X_test). Then comes the shape of each image (28x28). The last number is 1, which signifies that the images are greyscale.

```
#reshape data to fit model
X_train = X_train.reshape(X_train.shape[0], 784,1)
X_test = X_test.reshape(X_test.shape[0], 784,1)
```

5.2 HANDWRITTEN CHARACTER RECOGNITION Using CNN

In this section, we discuss the proposed method to recognize handwritten character. The framework involves several stages to accurately identify character. Figure: 5.2.0 describes the proposed schematic overview of our framework. It has several phases:

Step i: Collected English Handwritten characters from EMNICT Dataset.

Step ii: Collected samples are pre-processed to scale the images by 28X28 in gray scale format and normalize them between 0 and 1.

```
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.01568627 0.01568627 0.01568627 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.00392157 0.07843137 0.1254902
  0.14509804 0.2        0.44705882 0.49019608 0.44705882 0.17647059
  0.01960784 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.00784314 0.03529412 0.18431373 0.62352941 0.79215686
  0.85098039 0.87058824 0.96078431 0.97647059 0.96078431 0.76078431
```

Figure: 5.1 Normalized data

Step iii: The dataset is divided into two groups as train and test phase. In training phase, we have considered 814, 255 individual samples i.e 697, 932 characters for training and for testing phase 116, 323 samples.

Step iv: We implemented a Convolutional Neural Network (CNN) model in Python using Colab. The training samples are input to CNN and that adjusted the weights to accurately recognize characters.

The proposed CNN method is evaluated on test dataset to measure the overall accuracy and individual digit and character recognition rate. We ran different iterations by varying the neural network parameters that can observed in lot of variation in accuracy.

The overall accuracy is captured of 15 epochs is shown in Fig. 5.1.1. At the 15th epoch, we measured heights accuracy of 86.96% is captured. In our experiments, we also captured the each digit and character recognition rate that gives the insight of how each digits can be accurately identified. Most of the digits are recognized more the 84% except letter e, digit 2, digit 5 and letter r are very low in accuracy due to ambiguity in patterns. These digits and letter are closely resemble each other that can successfully recognize individual patterns.

Next, a diagnostic plot is shown, giving insight into the learning behavior of the model across each fold.

In this case, we can see that the model generally achieves a good fit, with train and test learning curves converging. There is no obvious sign of over- or under fitting.

The accuracy and loss graphs for the CNN model are shown in figure: 5.2.1 and figure 5.2.2 respectively.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_3 (Reshape)          (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 64)        51264
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               1606144
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 62)                31806
=================================================================
Total params: 1,690,046
Trainable params: 1,690,046
Non-trainable params: 0
_________________________________________________________________
None
Train on 697932 samples, validate on 116323 samples
```

Figure: 5.2.0 Model Summary of CNN

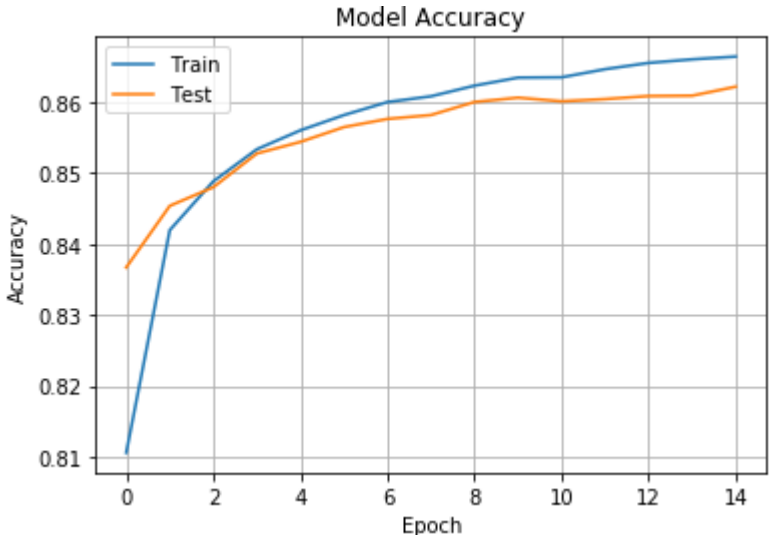


Figure: 5.2.1 Model Accuracy

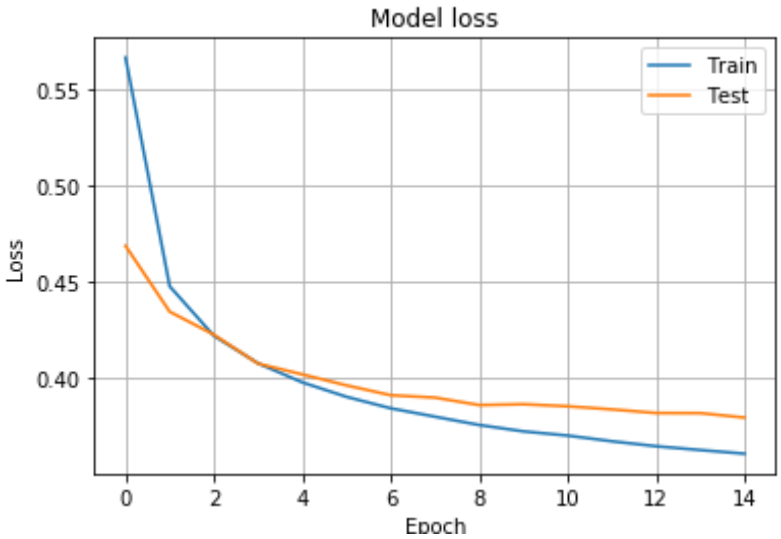


Figure: 5.2.2 Model Loss

After we found out accuracy and loss for our model, we created a graph to see which optimizer works well for model and it is shown in figure: 5.2.3.

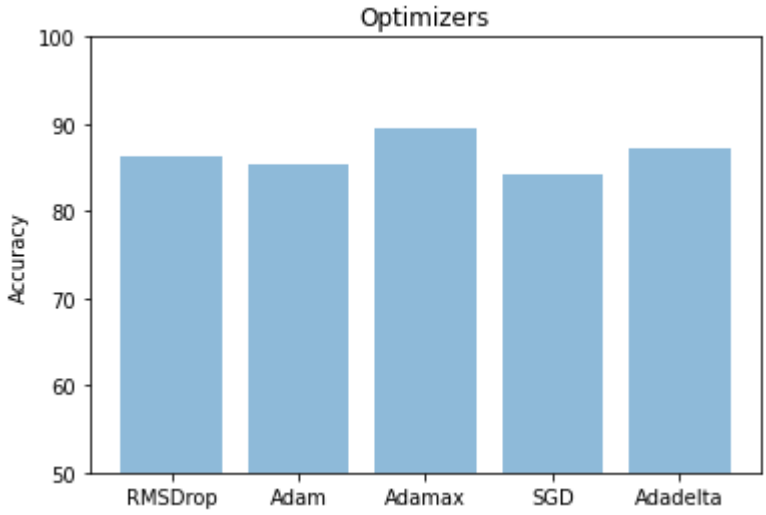


Figure: 5.2.3 Optimizers

5.3 SAVE THE MODEL

A final model is typically fit on all available data, such as the combination of all train and test dataset.

In this tutorial, we are intentionally holding back a test dataset so that we can estimate the performance of the final model, which can be a good idea in practice. As such, we will fit our model on the training dataset only.

```
#fit model
history = model.fit(train_images,train_labels,validation_data=
(test_images, test_labels), batch_size=128, epochs=15)
```

Once fit, we can save the final model to an H5 file by calling the *save()* function on the model and pass in the choosen filename.

```
 # save model
 model.save('final_model.h5')
```

Note, saving and loading a Keras model requires that the h5py library is installed on your workstation.

6. IMPLEMENTATION

Convolutional neural networks are more complex than standard multi-layer perceptrons, so we will start by using the structure used in the given code to begin with that uses all of the elements for state of the art results. Below summarizes the network architecture.

1. The first two hidden layers are convolutional layesr called a Convolution2D. These layers have 32 feature maps, which with the size of 5×5 and a rectifier activation function. This is the input layer, expecting images with the structure outline above [pixels][width][height].
2. Next we define two pooling layers that takes the max called MaxPooling2D. It is configured with a pool size of 2×2.
3. The next layer is a regularization layer using dropout called Dropout. It is configured to randomly exclude 40% of neurons in the layer in order to reduce overfitting.
4. Next is a layer that converts the 2D matrix data to a vector called Flatten. It allows the output to be processed by standard fully connected layers.
5. Next a fully connected layer with 512 neurons and rectifier activation function.
6. Again we have a dropout layer to randomly exclude 50% of neurons
7. Finally, the output layer has 62 neurons for the 62 classes and a softmax activation function to output probability-like predictions for each class.

.

Source Code

ModelTrain.ipnb - This file is used train the CNN model and to save the model for later use.

```
!pip install python-mnist

from google.colab import drive
drive.mount('/content/drive')

import tensorflow as tf
```

```
import numpy as np
from mnist import MNIST
from tensorflow.python.framework.graph_util import convert_variables_to_constants


def freeze_graph(session, keep_var_names = None, output_names = None,
clear_devices=True):
    graph = session.graph
    with graph.as_default():
        fvr = list(set(v.op.name for v in tf.global_variables()).difference(keep_var_names or []))
        output_names = output_names or []
        output_names += [v.op.name for v in tf.global_variables()]
        igf = graph.as_graph_def()
        if clear_devices:
            for node in igf.node:
                node.device=''
        frozen_graph= convert_variables_to_constants(session, igf, output_names, fvr)
        return frozen_graph


mndata = MNIST('data')
#This will load the train and test data
X_train, y_train = mndata.load('drive/My Drive/data/emnist-byclass-train-images-idx3-
ubyte',
                               'drive/My Drive/data/emnist-byclass-train-labels-idx1-ubyte')
X_test, y_test = mndata.load('drive/My Drive/data/emnist-byclass-test-images-idx3-
ubyte',
                             'drive/My Drive/data/emnist-byclass-test-labels-idx1-ubyte')

# Convert data to numpy arrays and normalize images to the interval [0, 1]
X_train = np.array(X_train) / 255.0
y_train = np.array(y_train)
X_test = np.array(X_test) / 255.0
y_test = np.array(y_test)
```

```
X_train = X_train.reshape(X_train.shape[0], 28, 28)
X_test = X_test.reshape(X_test.shape[0], 28, 28)
print( X_train.shape)
print (X_test.shape)
from matplotlib import pyplot as plt
#Display a random image
plt.imshow(X_train[0])
plt.show

#we can see how an image array looks like. all float values b/w 0 and 1
m = X_train[2]
print(m)

#Now we perform Image preprocessing. We reverse and rotate all train and test images

#for train data
for t in range(697932):
    X_train[t]=np.transpose(X_train[t])

#checking
plt.imshow(X_train[0])
plt.show

#for test data
for t in range(116323):
    X_test[t]=np.transpose(X_test[t])

#checking
plt.imshow(X_test[1])
plt.show
print('Process Complete: Rotated and reversed test and train images!')

#Checking the last train image, just to be sure!
m = X_train[697931]
```

```
plt.imshow(m)
plt.show

#Reshaping train and test data again for input into model
#The last number is 1, which signifies that the images are grayscale

X_train = X_train.reshape(X_train.shape[0], 784,1)
X_test = X_test.reshape(X_test.shape[0], 784,1)
print (X_train.reshape)
print (X_test.reshape)

# Creation of model

from keras.models import Sequential
from keras import optimizers
from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense,
Reshape, LSTM
from keras import backend as K
from keras.constraints import maxnorm
def resh(ipar):
   opar = []
 for image in ipar:
      opar.append(image.reshape(-1))
   return np.asarray(opar)

from keras.utils import np_utils

train_images = X_train.astype('float32')
test_images = X_test.astype('float32')

train_images = resh(train_images)
test_images = resh(test_images)

#one-hot encode target column
train_labels = np_utils.to_categorical(y_train, 62)
```

```
test_labels = np_utils.to_categorical(y_test, 62)
K.set_learning_phase(1)

model = Sequential()
model.add(Reshape((28,28,1), input_shape=(784,)))
model.add(Convolution2D(32, (5,5),activation='relu', padding='same'))
model.add(Convolution2D(64, (5,5),activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(MaxPooling2D(pool_size=(2,2))


model.add(Dropout(0.4model.add(Flatten())

model.add(Dense(512, activation='relu',kernel_constraint=maxnorm(3)))

model.add(Dropout(0.5))

model.add(Dense(62, activation='softmax'))
opt = optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0)


#compile model using accuracy to measure model performance
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

# Training of model and evaluation
print(model.summary())

#parameters: training data (train_X), target data (train_y), validation data, and the number of
epochs.
history = model.fit(train_images,train_labels,validation_data=(test_images, test_labels),
batch_size=128, epochs=15)

#evaluating model on test data. will take time
scores = model.evaluate(test_images,test_labels, verbose = 0)
print("Accuracy: %.2f%%"%(scores[1]*100))
```

```
# Creating model history graphs
print(history.history.keys())

# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.grid()
plt.show()


# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.grid()
plt.show()

objects = ('RMSDrop', 'Adam', 'Adamax', 'SGD', 'Adadelta')
y_pos = np.arange(len(objects))
performance = [86.2,85.39,89.53,84.29,87.11]

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('Accuracy')
plt.title('Optimizers')
plt.ylim(50,100)
```

```
plt.show()

frozen_graph = freeze_graph(K.get_session(), output_names=[model.output.op.name])
tf.train.write_graph(frozen_graph,'.','drive/My Drive/Colab Notebook/HCR-N/PBfile89531.pb',as_text=False)
print(model.input.op.name)
print(model.output.op.name)

#Predicting a single image using the model

m = X_test[258].reshape(28,28)
plt.imshow(m)
plt.show
print('prediction: '+str(model.predict_classes(X_test[258].reshape(1,784))))

from keras.models import load_model
from keras.models import model_from_json

model_json = model.to_json()
with open("drive/My Drive/Colab Notebook/HCR-N/model1.json", "w") as json_file:
    json_file.write(model_json)
#saves the model info as json file

model.save_weights("drive/My Drive/Colab Notebook/HCR-N/model1.h5")
# Creates a HDF5 file 'model.h5'
```

For usage of this model to predict words, open segment.ipynb where we will load the saved model and use it for character recognition.

```
# Loading the model

from keras.models import load_model
from keras.models import model_from_json
from IPython import get_ipython
```

```
from google.colab import drive
drive.mount('/content/drive')

language = 'en'

json_file = open('drive/My Drive/Colab Notebook/HCR-N/model1.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

loaded_model.load_weights('drive/My Drive/Colab Notebook/HCR-N/model1.h5')

model = loaded_model

print('Model successfully loaded')

import cv2
import numpy as np
from matplotlib import pyplot as plt

characters =
['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']

#enter input image here
image = cv2.imread('drive/My Drive/Colab Notebook/HCR-N2/g.png')
height, width, depth = image.shape

#resizing the image to find spaces better

image = cv2.resize(image, dsize=(width*5,height*4), interpolation=cv2.INTER_CUBIC)

#grayscale
```

```
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
#binary
ret,thresh = cv2.threshold(gray,127,255,cv2.THRESH_BINARY_INV)
#dilation
kernel = np.ones((5,5), np.uint8)
img_dilation = cv2.dilate(thresh, kernel, iterations=1)
#adding GaussianBlur
gsblur=cv2.GaussianBlur(img_dilation,(5,5),0)

#find contours
ctrs, hier = cv2.findContours(gsblur.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
m = list()

#sort contours
sorted_ctrs = sorted(ctrs, key=lambda ctr: cv2.boundingRect(ctr)[0])
pchl = list()
dp = image.copy()
for i, ctr in enumerate(sorted_ctrs):
 # Get bounding box
 x, y, w, h = cv2.boundingRect(ctr)
 cv2.rectangle(dp,(x-10,y-10),( x + w + 10, y + h + 10 ),(90,0,255),9)
 plt.imshow(dp)
for i, ctr in enumerate(sorted_ctrs):
# Get bounding box
 x, y, w, h = cv2.boundingRect(ctr)
```

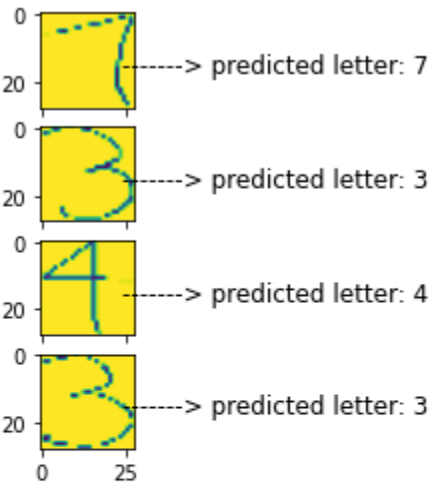
```
# Getting ROI
roi = image[y-10:y+h+10, x-10:x+w+10]
roi = cv2.resize(roi, dsize=(28,28), interpolation=cv2.INTER_CUBIC)
roi = cv2.cvtColor(roi,cv2.COLOR_BGR2GRAY)
roi = np.array(roi)
t = np.copy(roi)
t = t / 255.0
t = 1-t
t = t.reshape(1,784)
m.append(roi)
pred = model.predict_classes(t)
pchl.append(pred)
pcw = list()
interp = 'bilinear'
fig, axs = plt.subplots(nrows=len(sorted_ctrs), sharex=True, figsize=(1,len(sorted_ctrs)))
for i in range(len(pchl)):
#print (pchl[i][0])
pcw.append(characters[pchl[i][0]])
axs[i].set_title('-------> predicted letter: '+characters[pchl[i][0]], x=2.5,y=0.24)
axs[i].imshow(m[i], interpolation=interp)
plt.show()
predstring = ''.join(pcw)
print('Predicted String: '+predstring)
```

7 . TESTING

Outputs for Handwritten Character Recognition

Input 1:

Output 1:

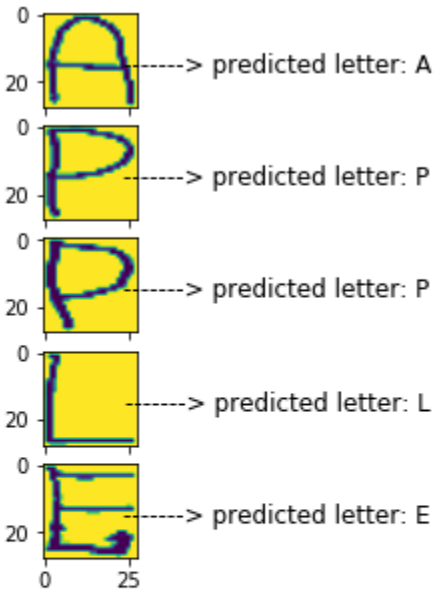



```
] predstring = ''.join(pcw)
  print('Predicted String: '+predstring)
```

```
Predicted String: 7343
```

Input 2:

Output 2:

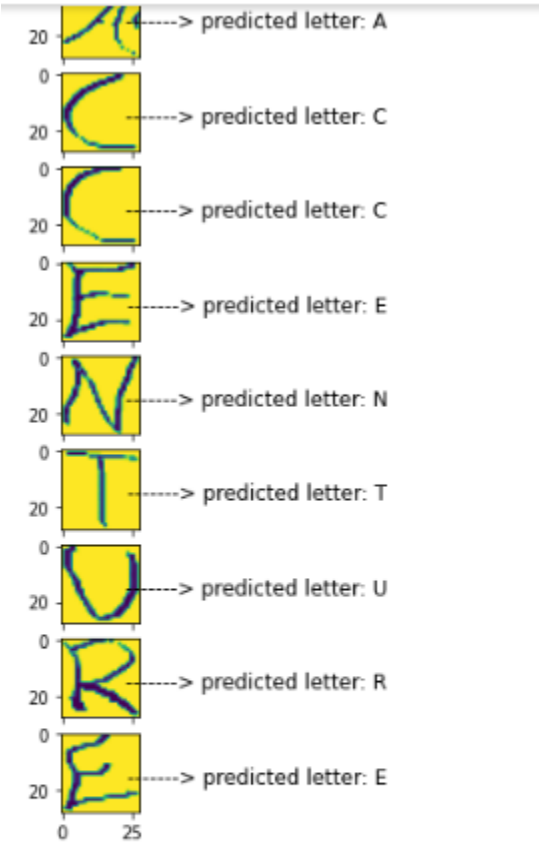



```
predstring = ''.join(pcw)
print('Predicted String: '+predstring)
```

```
Predicted String: APPLE
```

Input 3:

Output 3:




```
predstring = ''.join(pcw)
print('Predicted String: '+predstring)
```

```
Predicted String: ACCENTURE
```

8. CONCLUSION AND FUTURE SCOPE

Handwritten Character Recognition plays an important role in optical character recognition and pattern recognition. It greatly contributes to automation of many things like medical prescriptions, tax returns etc. Earlier handcrafted feature methods were used for character recognition, which are inefficient and requires much effort and time. One of the best alternative for conventional handcrafted features was to use deep learning techniques for character recognition. All the features used here are machine generated features and produces much efficiency and accuracy to the result. The various deep learning techniques used are convolutional neural network, etc.

As a part of future scope, we are trying to increase the complexity from character and word recognition level to sentence and paragraph recognition level. Apart from this we will implement front-end for this project using HTML, Bootstrap so that our project can be user friendly.

9. REFERENCES

- Handwritten Character Recognition using Deep Learning
  https://ieeexplore.ieee.org/document/8473291
- HCR using CNN and Tensor Flow
  https://www.ijitee.org/wp-content/uploads/papers/v8i6s4/F12940486S419.pdf
- https://ieeexplore.ieee.org/document/8741412
- https://www.youtube.com/watch?v=i_XJa165_9I
- https://elitedatascience.com/keras-tutorial-deep-learning-in-python
- CNN detailed video
  https://www.youtube.com/watch?v=umGJ30-15_A