HANDWRITTEN CHARACTER RECONITION

A PROJECT REPORT

Submitted by

ANANTHA NARAYANAN G.S.	**111617104003**
ASHOK KUMAR N.	**111617104006**
BHARATH KUMAR K.	**111617104012**

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

R.M.K COLLEGE OF ENGINEERING AND TECHNOLOGY

PUDUVOYAL

ANNA UNIVERSITY : CHENNAI 600 025

APRIL 2021

BONAFIDE CERTIFICATE

Certified that this project report "**HANDWRITTEN CHARACTER RECONITION**" is the bonafide work of "ANANTHA NARAYANAN G.S. 111617104003, ASHOK KUMAR N. 111617104006, BHARATH KUMAR K. 111617104012" who carried out the project work under my supervision.

SIGNATURE

Dr. D. PAULRAJ
HEAD OF THE DEPARTMENT,
Computer Science and Engineering,
R.M.K. College of Engineering and Technology, RSM Nagar, Puduvoyal,
Gummidipoondi Taluk, Thiruvallur District, Tamil Nadu - 601 206.

SIGNATURE

Ms. Pradheeba U.,
Assistant Professor,
Computer Science and Engineering,
R.M.K. College of Engineering and Technology, RSM Nagar, Puduvoyal,
Gummidipoondi Taluk, Thiruvallur District, Tamil Nadu - 601 206.

Submitted to project Viva Voce Examination held on 27 – 03 – 2021.

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We take this opportunity to thank **LORD ALMIGHTY** for all the blessings and grace showered upon us and for giving us patience and wisdom to complete the research work and project successfully. We would like to express our sincere gratitude to our college Chairman **Mr. R. S. MUNIRATHINAM** and Vice - Chairman **Mr. R. M. KISHORE, R.M.K COLLEGE OF ENGINEERING AND TECHNOLOGY** for the all the facilities endowed in order to make education provided is the best. Our humble indebtedness to **Dr. T. RENGARAJA, PRINCIPAL** of R.M.K College of Engineering and Technology, for giving us this opportunity to do this project and our most supportive HEAD OF THE DEPARTMENT and PROFESSOR, **Dr. D. PAULRAJ**, M.E., PhD and **Dr. VIGILSON PREM** for his valuable and innovative suggestions, constructive interaction, constant encouragement and zeal in seeing us succeed have enabled us to culminate the project triumphantly. Special gratitude and thankfulness to our **PROJECT COORDINATOR and SUPERVISOR, Ms. S. INDRA PRIYADARSHINI** and **Ms. PRADHEEBA U.,** Department of Computer Science and Engineering for their involvement, continuous support, motivation and guidance for the completion of this project. We also convey our humble and heartfelt thanks to all the **FACULTY MEMBERS** who had provided their contribution to steer the project into the right path. Above all, we would like to express our deepest obligation and reverence to our beloved PARENTS, for their motivation, moral support and encouragement throughout our lifetime.

- ANANTHA NARAYANAN G.S.

- ASHOK KUMAR N.

- BHARATH KUMAR K.

ABSTRACT

Optical Character Recognition (OCR) is a subfield of Image Processing which is concerned with extracting text from images or scanned documents. In this project, we have chosen to focus on recognizing handwritten digits available in the MNIST database. The challenge in this project is to use basic Image Correlation, also known as Matrix Matching, techniques in order to maximize the accuracy of the handwritten digits recognizer without going through sophisticated techniques like machine learning.

Keywords: Image Processing, Optical Character Recognition, Handwritten Digits, Image Correlation, Matrix Matching, Machine Learning.

Table of Content

Introduction 6

Background 8

- Image Processing 8
  - ✓ Optical Character Recognition (OCR) - History 8
- Methods Used in OCR 9
  - ✓ Machine Learning 9
    - o Artificial Neural Network 9
    - o Support Vector Machine 9
  - ✓ Image Correlation 10
  - ✓ Feature Extraction 10
- **Tools** 11
  - ✓ Anaconda 11
    - o MNIST Database 11
    - o Feasibility Study 12
- **Methodology** 12
  - ✓ Getting Familiar with the Tools 12
  - ✓ Creating the reference and test set 13
  - ✓ Different Versions 13
- **Data** 14
  - ✓ Dataset Format 14
  - ✓ Reference Set 14
  - ✓ Test Set 15
- **System Analysis** 16
  - ✓ Overall Description 16
    - o Problem Description 16
    - o Existing System 16
    - o Disadvantages 16
    - o Proposed system 17
    - o Advantage of proposed System 17
    - o Architecture diagram 17
  - ✓ System Configuration 17
    - o Hardware Requirements 17
    - o Software Requirements 17
- **System Design** 18
  - ✓ Use Case Diagram 18
  - ✓ Data Flow Diagram 18
- **Implementation** 19
- **Challenges and Limitations** 21
- **Conclusion and Future Work** 22
- **References** 23
- **Appendix** 24

1. Introduction

It is easy for the human brain to process images and analyse them. When the eye sees a certain image, the brain can easily segment it and recognize its different elements. The brain automatically goes through that process, which involves not only the analysis of this images, but also the comparison of their different characteristics with what it already knows in order to be able to recognize these elements. There is a field in computer science that tries to do the same thing for machines, which is Image Processing.

Image processing is the field that concerns analysing images so as to extract some useful information from them. This method takes images and converts them into a digital form readable by computers, it applies certain algorithms on them, and results in a better quality images or with some of their characteristics that could be used in order to extract some important information from them.

Image processing is applied in several areas, especially nowadays, and several software have been developed that use this concept. Now we have self-driven cars which can detect other cars and human beings to avoid accidents. Also, some social media applications, like Facebook, can do facial recognition thanks to this technique. Furthermore, some software use it in order to recognise the characters in some images, which is the concept of optical character recognition, which we will be discussing and discovering in this project.

One of the narrow fields of image processing is recognizing characters from an image, which is referred to as Optical Character Recognition (OCR). This method is about reading an image containing one or more characters, or reading a scanned text of typed or handwritten characters and be able to recognize them. A lot of research has been done in this field in order to find optimal techniques with a high accuracy and correctness. The most used algorithms that proved a very high performance are machine learning algorithms like Neural Networks and Support Vector Machine.

One of the main applications of OCR is recognizing handwritten characters. In this project, we will focus on building a mechanism that will recognize handwritten digits. We will be reading images containing handwritten digits extracted from the MNIST database and try to recognize which digit is represented by that image. For that we will use basic Image Correlation techniques, also referred to as Matrix Matching. This approach is based on matrices manipulations, as it reads the images as matrices in which each element is a pixel. It overlaps the image with all the images in the reference set and find the correlation between them in order to be able to determine the digit it represents.

The goal of this project is to apply and manipulate the basic image correlation techniques to build program and keep polishing and enhancing in order to investigate to which extent it can get improved. This would allow us to see how far we can go, in terms of accuracy and performance, but using just the very simple and basic techniques of matrix matching and without going into complicated methods like machine learning.

2. Background

In this section, we will present the main concepts that this project is concerned with.

2.1. Image Processing

Image processing is a very wide field within computer science which deals mainly with analysing images and trying to get some information out of them. The image to be processed is imported then analysed using some computations, which, by the end, results either in an image with a better quality or some of the characteristics of this image depending on the purpose of this analysis [1]. This is a very wide field within computer science, which also has several other subfields of which Optical Character Recognition that we will be mainly dealing with throughout this project.

2.2. Optical Character Recognition (OCR) - History

It is easy for the naked eye to recognize a character when spotted in any document; however, computers cannot identify the characters from an image or scanned document. In order to make this possible, a lot of research has been done, which resulted in the development of several algorithms that made this possible. One of the fields that specialize in character recognition under the light of Image Processing is Optical Character Recognition (OCR).

In Optical Character Recognition, a scanned document or an image is read and segmented in order to be able to decipher the characters it contains [2]. The images are taken and are preprocessed so as to get rid of the noise and have unified colors and shades, then the characters are segmented and recognized one by one, to finally end up with a file containing encoded text containing these characters, which can be easily read by computers [2].

Optical Character Recognition dates back to the early 1900s, as it was developed in the United States in some reading aids for the blind [3]. In 1914, Emanuel Goldberg was able to implement a machine able to convert characters into “standard telegraph code” [4]. In the 1950s, David Shepard, who was at that time an engineer at the Department of Defense, developed a machine that he named Gismo, which is able to read characters and translate them into machine language [5]. In 1974, Ray Kurzweil decided to develop a machine that would read text for blind and visually impaired people under his company, Kurzweil Computer Products. There are several softwares and programs, nowadays, which use OCR in several different applications. In 1996, the United States Postal Services were able to develop a mechanism, HWAI, which recognizes handwritten mail addresses [6].

2.3. Methods Used in OCR

A lot of research has been done in the field of OCR, and still being done, which resulted in the development of several algorithms which enable computers to recognize characters from images or scanned texts. Many of these techniques have attained very high efficiency and a low error rate. However, these algorithms are still being investigated and improved for a better performance.

2.3.1. Machine Learning

Machine learning is a field that concerns making programs learn and know how to behave in different situations using data. One of its applications is Optical Character Recognition.

2.3.1.1. Artificial Neural Network

An Artificial Neural Network (ANN) is a system that mimics the human's biological neural network in the brain. It is an algorithm used for machine learning, which means it uses data to learn how to respond to different inputs. The ANN can be seen as a box, which takes one or more inputs and gives one output. Inside the box, there exist several interconnected nodes. The input is fed into the program, which goes through the several layers and nodes of the ANN and gives an output using a transfer function [7].

Artificial Neural Networks are used for OCR and have proved a very high accuracy rate. In this case, the ANN would "recognize a character based on its topological features such as shape, symmetry, closed or open areas, and number of pixels" [8]. The high accuracy of this kind of algorithms is mainly thanks to its ability of learning from the training set, which would contain characters with similar features.

Some Neural Networks have proven a very high performance. An implementation of the ANN done by Simard, Steinkraus, and Platt has reduced the error rate of recognizing handwritten digits from the MNIST dataset to a percentage as low as 0.7% [9]

2.3.1.2. Support Vector Machine

Support Vector Machine (SVM) is an algorithm that belongs to machine learning as well. SVMs are known as high performance pattern classifiers. While Neural Networks aim at minimizing the training error, SVMs have as goal to minimize the "upper bound of the generalization error" [10]. The learning algorithm in this technique is based on classification and regression analysis.

This kind of classifier has been used in the recognition of very complex characters like the Khmer language and has proved a very high performance.

2.3.2. Image Correlation

Image Correlation is a technique used to recognize characters from images. This approach, also referred to as Matrix Matching, uses mathematical computations in order to analyse the images [11].

By using this technique, the images are read as matrices, where each element represents a pixel, which makes it easier to manipulate them using mathematical approaches. The image to be identified is loaded as a matrix and compared to the images in the reference set. The test image is overlapped with each image in the reference set to be able to see how it matches with each one of them so as to tell which one represents it the most. The decision can be made by seeing the pixels that match and the ones left out from either one of the two images.

This technique has many challenges and limitations, as it only overlaps the images and tries to see how much they look alike. By using this method, problems arise when having characters of different sizes, or when one of them is rotated by a certain angle.

2.3.3. Feature Extraction

Feature extraction is a technique based on pattern recognition. The main idea of feature extraction is analysing the images and derive some characteristics from these images that identify each specific element [12]. An example of these characteristics would be the curvatures, the holes, the edges, etc. In the case of digits recognition, these features could be the holes inside the digits (for example for the eight, the six, and maybe the two as well) as well as the angles between some straight lines (for example in the one, the four, and the seven). Whenever an unknown image is to be recognized, its features are compared to these so that it can be classified.

3. Tools

This project's main objective is to be able to read the images containing the handwritten digits and be able to identify those digits using basic image correlation techniques. These images are normally represented and read as matrices, in which every element portrays a pixel. The image correlation technique takes these matrices and compares them using some algorithms so as to identify the match that represents the digit we are trying to figure out. This project will be mainly using matrices and heavy numerical computations, That is why it is very important to consider the tools that would provide us with a suitable environment for performing these computations.

3.1. Anaconda

Anaconda is a free and open source software that uses a high-level programming language. It has the same functionalities as Matlab and is compatible with it. It offers a very simple and suitable interface to exert some mathematical computations. It provides some tools to solve mathematical problems like some common linear algebra problems [13]. It is also very efficient when it comes to the use of resources, i.e., time and memory, when it comes to these operations. Also, it is very easy to use it when dealing with matrices, as it provides with many functions and operations that make it less costly to manipulate them. In this project, we will deal with images as matrices, in which each element represents a pixel, that is why it is very necessary for us to choose a tool that will make our computations easier and more efficient in terms of time and memory resources. Both Matlab and Anaconda are very easy to learn and work with and provide a suitable environment for this kind of projects. We have opted for Anaconda as it is free and open source.

3.2. MNIST Database

The MNIST database, which stands for the Modified National Institute of Standards and Technology database, is a very large dataset containing several thousands of handwritten digits. This dataset was created by mixing different sets inside the original National Institute of Standards and Technology (NIST) sets, so as to have a training set containing several types and shapes of handwritten digits, as the NIST set was divided into those written by high school students and others written by the Census Bureau workers [14]. The MNIST dataset has been the target of so many research done in recognizing handwritten digits. This allowed the development and improvements of many different algorithms with a very high performance, such as machine learning classifiers.

In order to be able to implement our recognizer and test its performance, it is necessary to have a suitable dataset which contains a large number of handwritten digits. This dataset should be able to allow us to discover the challenges and limitation of the image correlation technique and push us to look for ways and rules

to enhance it and assess its accuracy. We have opted for this dataset to be used for testing our program since it has proved a great reliability and importance in the field.

3.3. Feasibility Study

From a technical perspective, since this project makes heavy use of numerical computations, using Anaconda is a wise choice as it will make the program more efficient. This software will also provide us with some libraries to read and manipulate the images that will make the implementation process easier.

As for the dataset to use in the testing of the project, we have chose the MNIST Database. This database contains thousands of handwritten digits that have been used in the development of programs with a similar aim. This dataset is open for public use with no charges. It is also very convenient for our project and will help us reduce the time by using directly as a test set without having to make one ourselves.

Since all the tools to be used in this project are free of charge and very easy to use, we can conclude that this project is very feasible in terms of financial resources as well as effort and time.

4. Methodology

4.1. Getting Familiar with the Tools

The first step we had to go through while working on this project was getting familiar with the tools used, i.e., Anaconda and the MNIST dataset. After setting up the environment for Anaconda to work perfectly and downloading the dataset, I have started experimenting with both in order to get familiar with them and know how to use them easily in the future.

Since all the programming is mainly done in Anaconda, we had to download it along with its Graphical User Interface into the computer, and learn a little bit about its functions and how to use it. Anaconda is a free software which makes it very easy to work with matrices and vectors and is very efficient in performing calculations on them. I have started learning how to use it and looking for its main functions that I will be using in the implementation of the project. For that, I have used some random images of digits to see how they can be read and modified as well as how to apply some computations on them.

Moreover, I had to investigate the format of the MNIST dataset and get familiar with its representation. The MNIST dataset, which was used to create our test set, contains thousands of handwritten digits, represented as matrices. It has been used in the development of several programs and projects with the same aim as ours. After downloading the file which contains the handwritten digits, I have loaded it on Anaconda in order to visualize the images and figure out how to use and manipulate them.

4.2. Creating the reference and test set

One of the main steps in the project is creating the reference and the test set that will both be used in the implementation phase.

The test set is to be used in order to assess the performance of the program and evaluate its success or error rate. It is to be taken from the MNIST dataset, since it contains the handwritten digits that we intend to recognize and identify.

As for the reference set, it is used to compare the test images and be able to identify the digit they represent. It is to be created using different fonts.

4.3. Different Versions

After having a look at the dataset and deciding on tools to be used for the implementation, we have started the development of our mechanism by developing a very basic version. After that, we have started identifying the challenges and problems we have faced and kept enhancing it. We have ended up with several versions different one from another, in each new version we increase the accuracy of the program by improving the method and introducing some new rules.

5. Data

It is very necessary to know the kind of data we are using before we start the design and the implementation of the program. That is why we had to have a look at its format to understand how it is represented before creating the reference and the test set.

5.1. Dataset Format

The dataset that I have downloaded from the MNIST database contains 60,000 images of handwritten digits, from zero to nine, all grouped in one file. Each of the images is of size 28 by 28 pixels and represents a digit. I have noticed that there is no pattern or order to the way the images were organized in the file. The images are represented as matrices, of which the elements represent the pixels. Also, each image has a label that indicates the digit represented. This label was very helpful later on in order to be able to create the test set. Furthermore, the data did not contain noise or any major problems to deal with, that is why it was used without preprocessing it.

Figure 6.1.1. Example of the MNIST dataset [15]

5.2. Reference Set

To be able to recognize the digit represented by a certain image, it is required to compare it with other images containing known digits to be able to make the decision. For that it is necessary to create a reference set which will contain all these images.

That is to say, each image we would want to recognize is to be compared to the images in the reference set. The image with the highest match is the one that represents the right number. Since handwritten digits differ from a person to another, the reference set needs to have digits with different fonts. That is why, we have created six images of each digit using the online image editor *pixlr.com*, each one with a different font. The reference set contains images with the same dimensions as the ones in the MNIST dataset, i.e., 28 by 28 pixels. Furthermore, these images have a black background and a white font, which made it easier to use and manipulate them later on using Anaconda. Furthermore, to make the comparison easier, we have regrouped each six images representing the same digit under one file. So the resulting reference set was ten files, each one representing a digit from zero to nine, and containing six images of that digit in different fonts.

The pixels of these images are then changed into zeros and ones, which makes the overlapping of the images easier. The black background was initially represented as zeros, so it is left the same. As for the pixels of the white font, each one of them was represented with a different non zero value depending on the shade of white. These non zero values are all converted into ones. The following image displays the digit "2" reference set. Rest of the reference sets are in Appendix A.

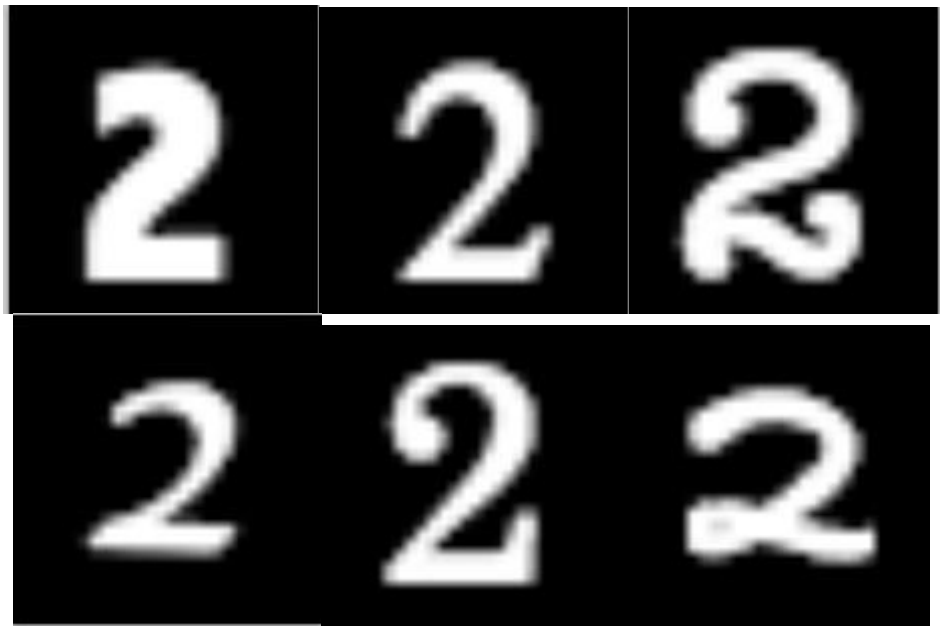


Figure 6.2.1. File of 2's in Reference Set

5.3. Test Set

The program to be developed needs to be tested against some images that contain handwritten digits so as to be able to assess its performance and calculate its success rate. That is why it is very necessary to create a test set. The test set

represents an example of the images containing the handwritten digits which will have to be compared to the images in the reference set so as to identify them.

This set was formed using the file from the MNIST database. The original file contained 60,000 images representing different digits. This made it difficult to look for each number using the label for the testing of the program. In order to make it easier to access each digit we want, we have decided to store a number of images from each digit in a separate file. That is why we have stored 20 images of each digit in ten different files. That is to say, the resulting test set was in the form of ten files, each one of them represents a digit and contains 20 images of it. These images were extracted from the initial file by reading them and their labels using Anaconda.

In order to make the manipulation of the matrices/images easier, we had to make some modifications in the elements of all the matrices representing the test set as well. The black pixels were originally represented as zeros, so they were left the same. As for the white ones, each of them had a different non zero number, so we turned them all into ones.

6. System Analysis:

6.1. OVERALL DESCRIPTION

6.1.1. PROBLEM DEFINITION

This system provides with an interface between data in digital and analog format. Images with characters can be provided to the software which will then be recognised by the software from the images and converted to digital format which can then by system. The software increases accuracy while not increasing time required for execution.

6.1.2. EXISTING SYSTEMS

The currently existing system uses Convolutional Neural Network which, While has accuracy, takes too much time for recognition of characters. This is used widely in other applications as an interface between the software and hardware.

6.1.2.1. DISADVANTAGE

- Takes too much time for recognition of software
- Does not have high enough accuracy to be used for commercial purposes
- Hardware dependence
- Unexplained behavior of the network**.**

6.1.3. PROPOSED SYSTEM

In this project, we plan to use Support Vector Machine algorithm to perform recognition of characters. SVM algorithm is more time-efficient than CNN algorithm and consumes less time for recognition of characters. But as trade-off, it has lower accuracy compared to SVM. The main purpose of this project is to increase accuracy of SVM algorithm while not increasing time required of execution.

6.1.3.1. ADVANTAGES OF PROPOSED SYSTEM

- Currently used system ,i.e., Support Vector Machine(SVM) has a longer running time and is time-efficient
- It can be used in high-dimensional spaces
- When SVM is used in character recognition, it becomes relatively memory-efficient.
- Can be used commercially

6.1.3.2. ARCHITECUTRE DIAGRAM:

6.2. SYSTEM CONFIGURATION

6.2.1. HARDWARE REQUIREMENTS

1. 8 GB RAM
2. 512 GB ROM+SSD
3. NVIDIA graphics card(For Large data).

6.2.2 SOFTWARE REQUIREMENTS:

1. Jupyter Notebook-Code visualisation
2. Online retail dataset/database in MS-EXCEL

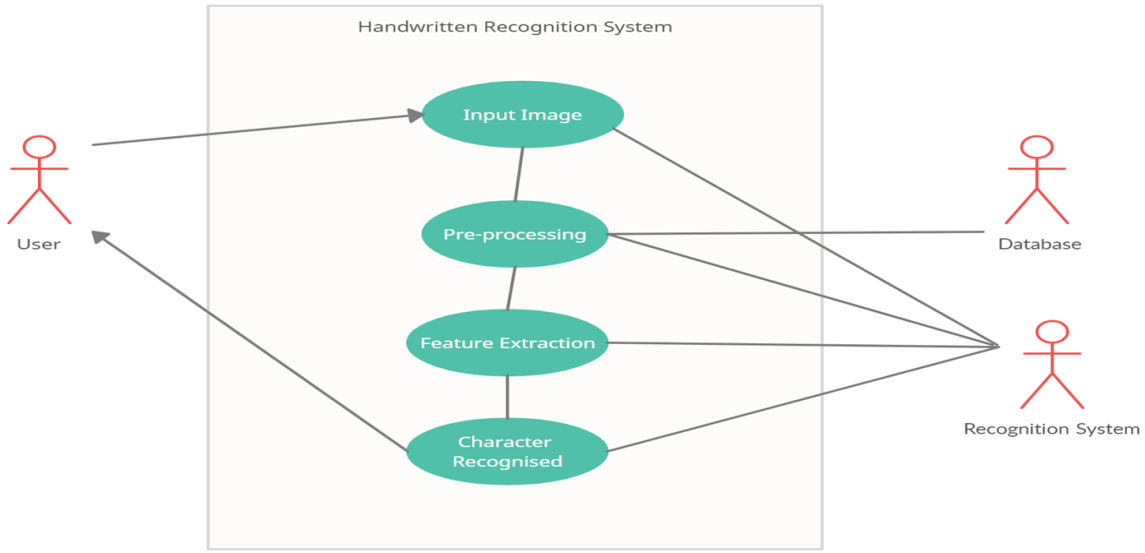
7. System Design:

Description:

This system aims to allow the system to convert characters from images and convert them to digital format so that the software is able to recognise the characters. It will allow for wide range of applications in all fields as it allows to optically recognise characters.

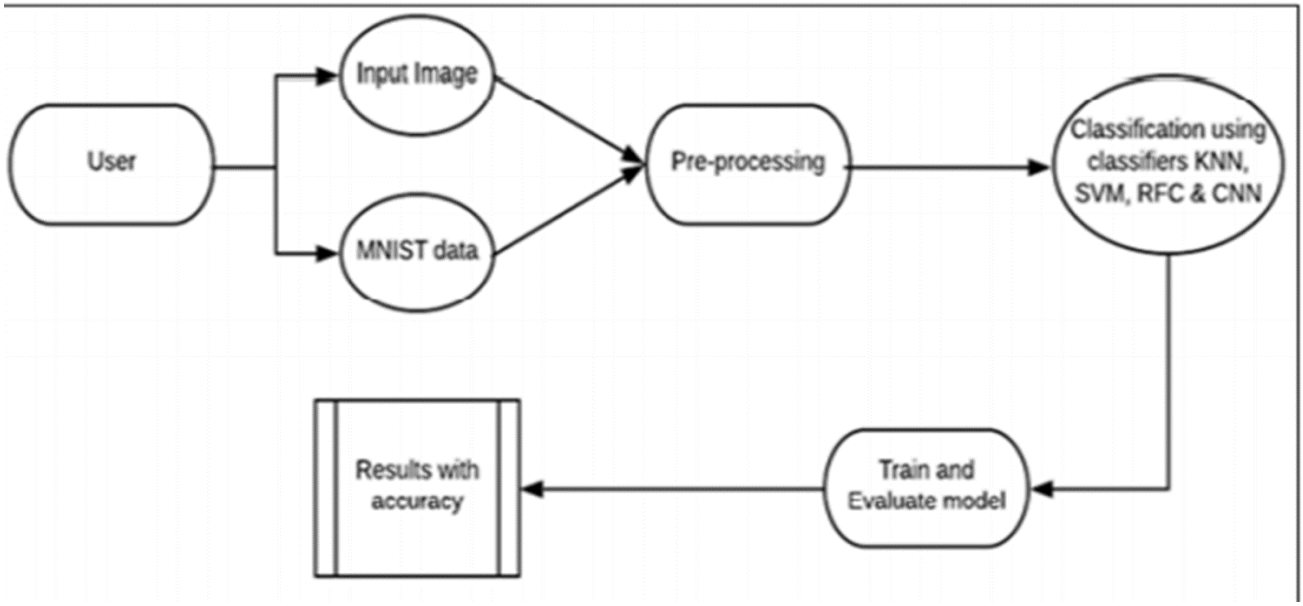
7.1 Use Case Diagram:

Use case diagram at its simple representation of a user's interaction with the system and its different relations / functionaries. It in fact provides a higher level view of the system.



7.2 Data Flow Diagram:

Data flow diagram shows the flow of data in the system. It shows how the data is used for activities in the system.



8. Implementation

In this project we aim at building a mechanism that would recognize handwritten digits from the MNIST dataset. We have opted for the Image Correlation technique, also referred to as Matrix Matching. The goal of this project is to use the basic and simple concepts of this methods and see how good can we make the accuracy rate without using complex techniques such as machine learning.

We have started the implementation of the program using a very simple and basic method, which is explained in further details under the section Version 1, then we have calculated its error rate. Afterwards, we have tried to spot the problems in the mechanism and find the limitations of the technique in order to improve it, and that is how we ended up with a second version of the program. We kept doing the same thing, each time trying to improve the previous version, which enabled us to keep improving the program and reach a higher accuracy and performance.

8.1.Version 1- Extracting the Reference Set from the MNIST Database

The initial reference set we have worked with contained images of types digits with different fonts. The decision of taking different fonts was mainly because to maximize the chances of a correct match, since the handwritten digits are written in different manners.

However, this was not a very accurate approach since the typed digits differ a lot from the handwritten ones. This approach resulted in a very low performance. That is why, we have decided to change the reference set, from the set of files containing the typed digits, into a set of images from the initial MNIST database, but not the same ones we have in the test set (Appendix C).

By following this change of reference set, and by taking just the first maximum without introducing any rules to it, we have an accuracy rate of 52.5%. We can see that the performance has changed drastically by changing the reference set.

In this approach, we would not only take the first maximum as the perfect match for the unknown digit, but we, also, take a look at the second maximum and analyse the relationship between the test digit and these two maximums as well as the pixels left out of all of them in every overlap so as to conclude the rules to be used to improve our mechanism.

That is why, the first step we did is to display matrices containing the maximum overlap with each digit along with the number of pixels left out from the test image and the number of pixels left out from the reference image (Appendix D). These results were used so as to be able to generate the rules.

After looking at the results of the first and second maximum, we have printed the pixels left out from both the test and the reference image, in order to be able to see the relationships between them and how they can help improve the efficiency of the program. We ended up extracting the following rules:

- ❏ If the first maximum is 8, and the second maximum is 1, then we look at the number of pixels left out from reference image of the 8 after the overlap with the test image of 1. If this number is significantly high, then we consider that the unknown digit is a 1, otherwise it is considered to be 8.

- ❏ If the first maximum is 6, and the second maximum is 3, then we look at the number of pixels left out from the reference image of the 6 after overlapping it with the test image of the 3. If this number is significantly high, then we consider the unknown digit to be a 3, otherwise we take it as a 6.

- ❏ If the first maximum is 8, and the second maximum is 7, then we consider the number of pixels left out from the reference image representing the 8. If this number is significantly high, then we consider the unknown digit to be 7, otherwise, we consider it as 8.

- ❏ If the first maximum is 7, and the second maximum is 6, then we there is a possibility it is either a 7 or 9. That is why we look at the number of pixels left out from both the reference and the test image of the 9. If the number of pixels left out from the reference image of 9 is almost half the number of pixels left out from the test image of 9, then it is most likely to be a 9, otherwise, we consider it as a 7.

- ❏ If none of these rules apply, then we take the first maximum as the digit we are trying to identify.

  By applying these rules the accuracy rate of the program went from 52.5% to 57%.

9. Challenges and Limitations

This project was my first encounter with Optical Character Recognition (OCR). That is why, while working on this project I was faced with many challenges and issues. First of all, it took me a long while to understand all the concepts and get familiar with them, from image processing to OCR, to all of the techniques and algorithms used in it. Furthermore, the data we were dealing with was very problematic in terms of the way the digits are written. Since some of the digits were rotated by an angle, some of them were thicker or thinner than the rest, and some of the digits were not well centered or were written in confusing ways. In addition to that, overcoming these challenges was not easy since we were only using basic image correlation techniques. We have tried to maximize the success rate, but we have only reached 57%, which is not a very high performance.

10. Conclusion and Future Work

Optical Character Recognition is a very broad field concerned with turning an image or a scanned document containing a set of characters into an encoded text that could be read by machines. In this project, we have attempted to build a recognizer for handwritten digits using the MNIST dataset. The challenge of this project was to be able to come up with some basic image correlation techniques, instead of some sophisticated algorithms, and see to what extent we can make this mechanism accurate. We have tried several versions and kept trying to improve each one in order to reach a higher performance rate. The last version has reached a rate of 57% accuracy. Unfortunately, we could not compare the performance of the mechanism we have built to some others that have already been designed and/or implemented before because we did not find any academic paper that tackles this method. The performance we have reached is far less than that of machine learning, which reaches a performance rate of 99.3%; however, it could be further improved and made into a better one. The goal of this project was to explore the field of OCR and try to come up with some techniques that could be used without going into deep computations, and even if the final result is not very reliable, it still provides an accuracy way better than random.

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 a 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 a 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 a higher efficiency.

11. References

[1] G. Anbarjafari, “1. Introduction to image processing,” Sisu@UT. [Online]. Available:

https://sisu.ut.ee/imageprocessing/book/1.

[2] S. Tanner, “Deciding whether Optical Character Recognition is feasible” [Online].

Available: http://www.odl.ox.ac.uk/papers/OCRFeasibility_final.pdf

[3] H. F. Schantz, The history of OCR, optical character recognition.

Manchester Center, VT:

Recognition Technologies Users Association, 1982.

[4] History of Computers and Computing, Internet, Dreamers, Emanuel Goldberg. [Online].

Available: https://history-computer.com/Internet/Dreamers/Goldberg.html.

[5] “Optical Character Recognition: What you Need to Know”. Phoenix Software

International. Available: http://www.phoenixsoftware.com/pdf/ocr_data_entry.pdf [6] S. N. Srihari, & E. J. Kuebert. Integration of Hand-Written Address Interpretation

Technology into the United States Postal Service Remote Computer Reader System.

Available: http://www.cedar.buffalo.edu/papers/articles/HWAI-RCR97.pdf

[7] V. Sharma, S. Rai, & A. Dev (2012). International Journal of Advanced

Research in Computer Science and Software Engineering. Available:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.468.9353&rep=rep1&type= pdf

[8] V. Shrivastava, & N. Sharma (2012). Artificial Neural Network Based

Optical Character Recognition. Available:

https://pdfs.semanticscholar.org/e0df/4d4c89af84b6caa250ba26e7f355258968de.pdf

[9] P. Y. Simard, D. Steinkraus, & J. Platt. Best Practices for Convolutional

Neural Networks Applied to Visual Document Analysis. Available:

https://www.microsoft.com/en-us/research/publication/best-practices-for-convolutional- neural-networks-applied-to-visual-document-analysis/?from=http%3A%2F%2Fresearch

.microsoft.com%2Fapps%2Fpubs%2F%3Fid%3D68920

[10] H. Byun, & S. W. Lee (2012). Applications of Support Vector Machines for Pattern Recognition: A Survey. Available:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.723.5893&rep=rep1&type= pdf

[11] R. Cintron, & V. Saouma (2008). Strain Measurements with the Digital Image Correlation System Vic-2D. Available:

https://nees.org/site/resources/pdfs/cintron_final_paper.pdf

[12] P. Singh, & S. Budhiraja. Feature Extraction and Classification Techniques in O.C.R.

Systems for Handwritten Gurmukhi Script – A Survey. Available:

http://www.ijera.com/papers/Vol%201%20issue%204/BQ01417361739.pdf

[13] "About". GNU Anaconda. Available: https://www.gnu.org/software/anaconda/about.html

[14] Y. LeCun, C. Cortes, C. J. C. Burges. The MNIST Database of Handwritten Digits.

Available: http://yann.lecun.com/exdb/mnist/

[15] "MNIST database," Wikipedia, 24-Apr-2018. [Online]. Available:

https://en.wikipedia.org/wiki/MNIST_database.

[16] "Landmarks in Postal Research at CEDAR". CEDAR University of Buffalo. Available:

http://www.cedar.buffalo.edu/~srihari/PostalResearch.pdf

APPENDIX:

Code:

```
from keras.datasets import mnist

import matplotlib.pyplot as plt

import cv2

import numpy as np

from keras.models import Sequential
```

```
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from keras.optimizers import SGD, Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.utils import to_categorical
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils import np_utils
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
from sklearn.utils import shuffle
```

```
# Read the data...
data = pd.read_csv(r"C:\Users\anant\Downloads\A_Z Handwritten Data\A_Z Handwritten Data.csv").astype('float32')

# Split data the X - Our data , and y - the prdict label
X = data.drop('0',axis = 1)
y = data['0']

# Reshaping the data in csv file so that it can be displayed as an image...

train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)
train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))

print("Train data shape: ", train_x.shape)
```

```
print("Test data shape: ", test_x.shape)


# Dictionary for getting characters from index values...

word_dict =
{0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P',16:'
Q',17:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X', 24:'Y',25:'Z'}



# Plotting the number of alphabets in the dataset...


train_yint = np.int0(y)

count = np.zeros(26, dtype='int')

for i in train_yint:

    count[i] +=1


alphabets = []

for i in word_dict.values():

    alphabets.append(i)


fig, ax = plt.subplots(1,1, figsize=(10,10))

ax.barh(alphabets, count)


plt.xlabel("Number of elements ")

plt.ylabel("Alphabets")

plt.grid()

plt.show()



#Shuffling the data ...

shuff = shuffle(train_x[:100])
```

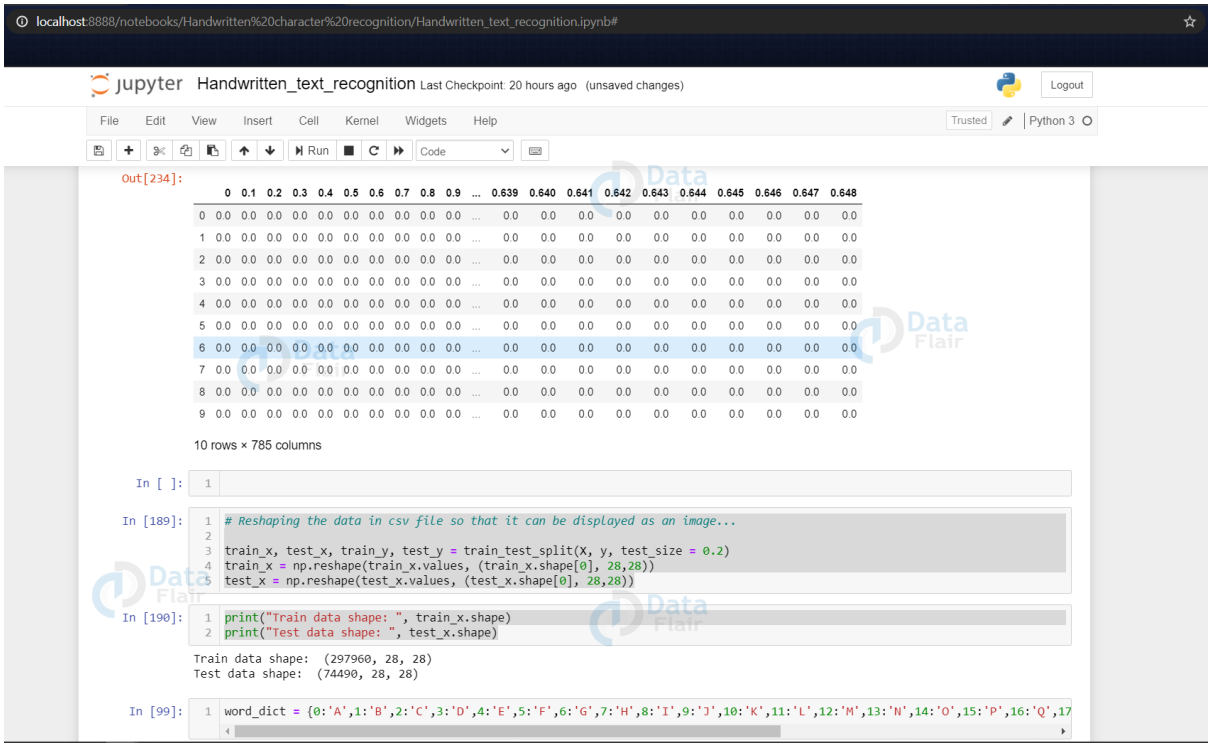
```
fig, ax = plt.subplots(3,3, figsize = (10,10))
axes = ax.flatten()

for i in range(9):
    axes[i].imshow(np.reshape(shuff[i], (28,28)), cmap="Greys")
plt.show()



#Reshaping the training & test dataset so that it can be put in the model...

train_X = train_x.reshape(train_x.shape[0],train_x.shape[1],train_x.shape[2],1)
print("New shape of train data: ", train_X.shape)

test_X = test_x.reshape(test_x.shape[0], test_x.shape[1], test_x.shape[2],1)
print("New shape of train data: ", test_X.shape)



# Converting the labels to categorical values...

train_yOHE = to_categorical(train_y, num_classes = 26, dtype='int')
print("New shape of train labels: ", train_yOHE.shape)

test_yOHE = to_categorical(test_y, num_classes = 26, dtype='int')
print("New shape of test labels: ", test_yOHE.shape)



# CNN model...

model = Sequential()
```

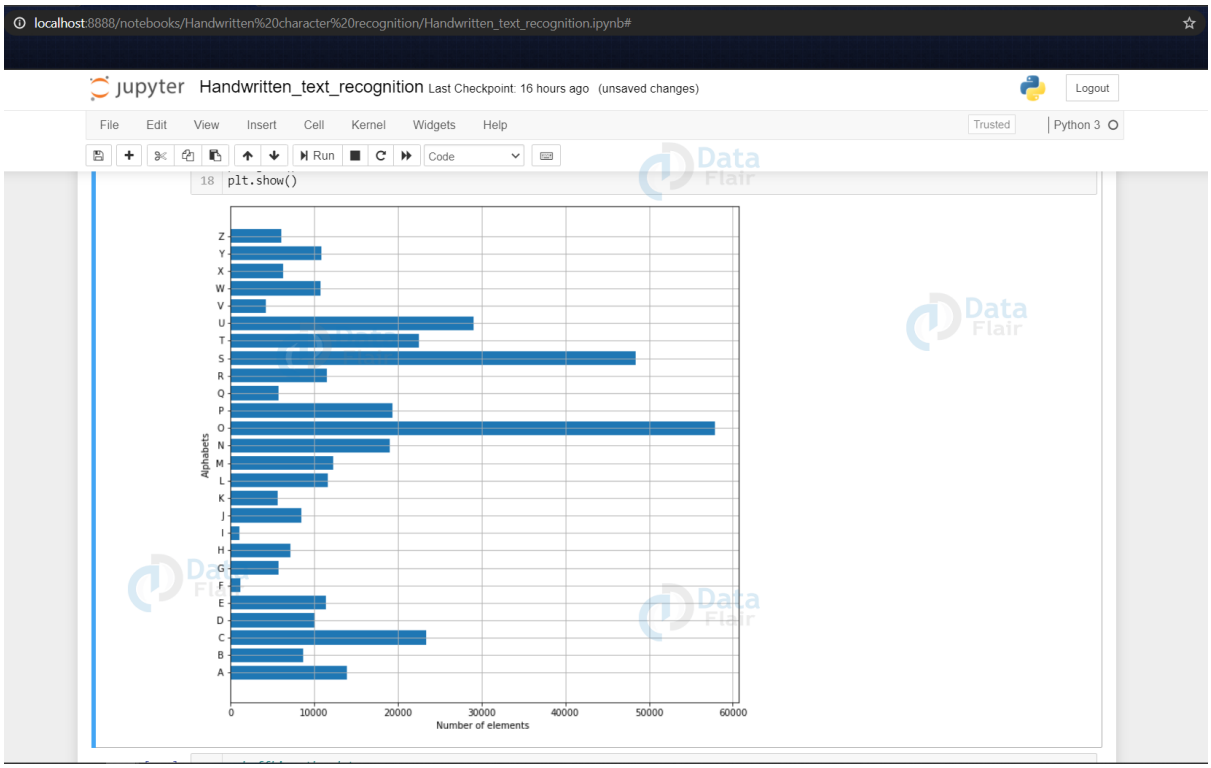
```
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation ="relu"))
model.add(Dense(128,activation ="relu"))

model.add(Dense(26,activation ="softmax"))



model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, min_lr=0.0001)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, verbose=0,
mode='auto')



history = model.fit(train_X, train_yOHE, epochs=1, callbacks=[reduce_lr, early_stop],
validation_data = (test_X,test_yOHE))



model.summary()
model.save(r'model_hand.h5')
```

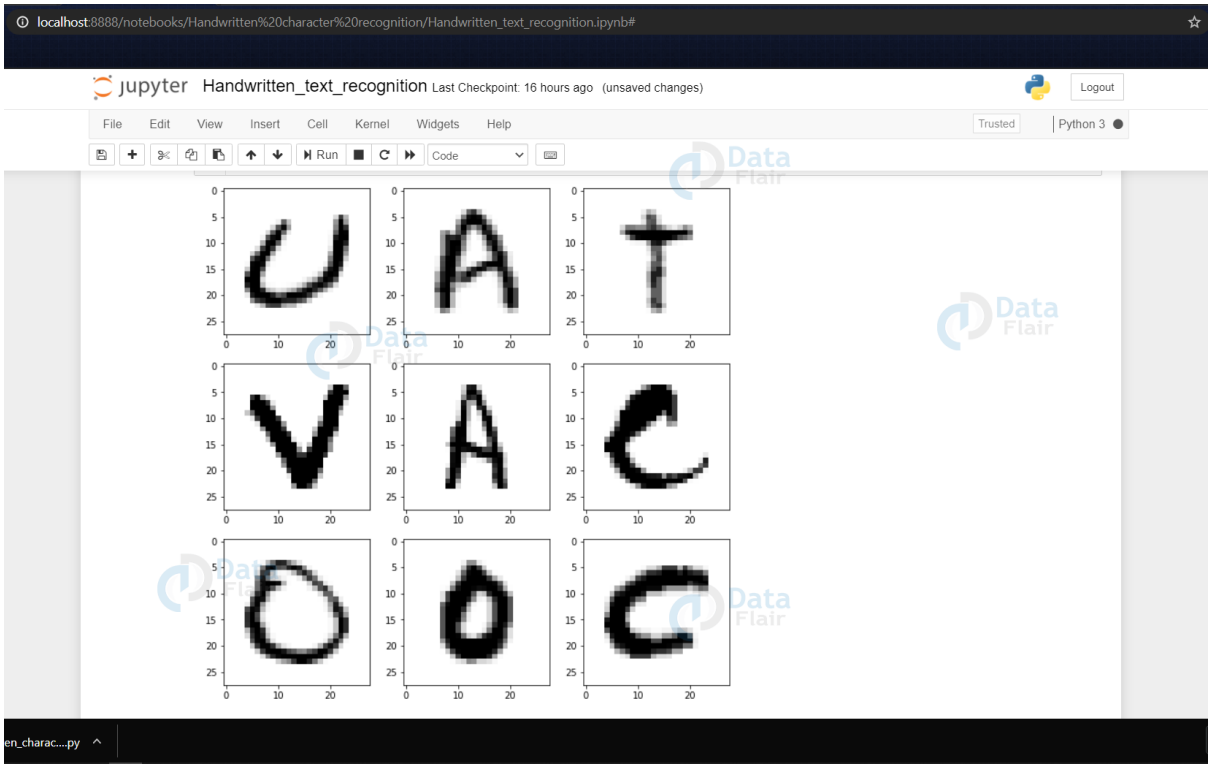
```
# Displaying the accuracies & losses for train & validation set...

print("The validation accuracy is :", history.history['val_accuracy'])
print("The training accuracy is :", history.history['accuracy'])
print("The validation loss is :", history.history['val_loss'])
print("The training loss is :", history.history['loss'])



#Making model predictions...

pred = model.predict(test_X[:9])
print(test_X.shape)


# Displaying some of the test images & their predicted labels...

fig, axes = plt.subplots(3,3, figsize=(8,9))
axes = axes.flatten()

for i,ax in enumerate(axes):
    img = np.reshape(test_X[i], (28,28))
    ax.imshow(img, cmap="Greys")
    pred = word_dict[np.argmax(test_yOHE[i])]
    ax.set_title("Prediction: "+pred)
    ax.grid()
```

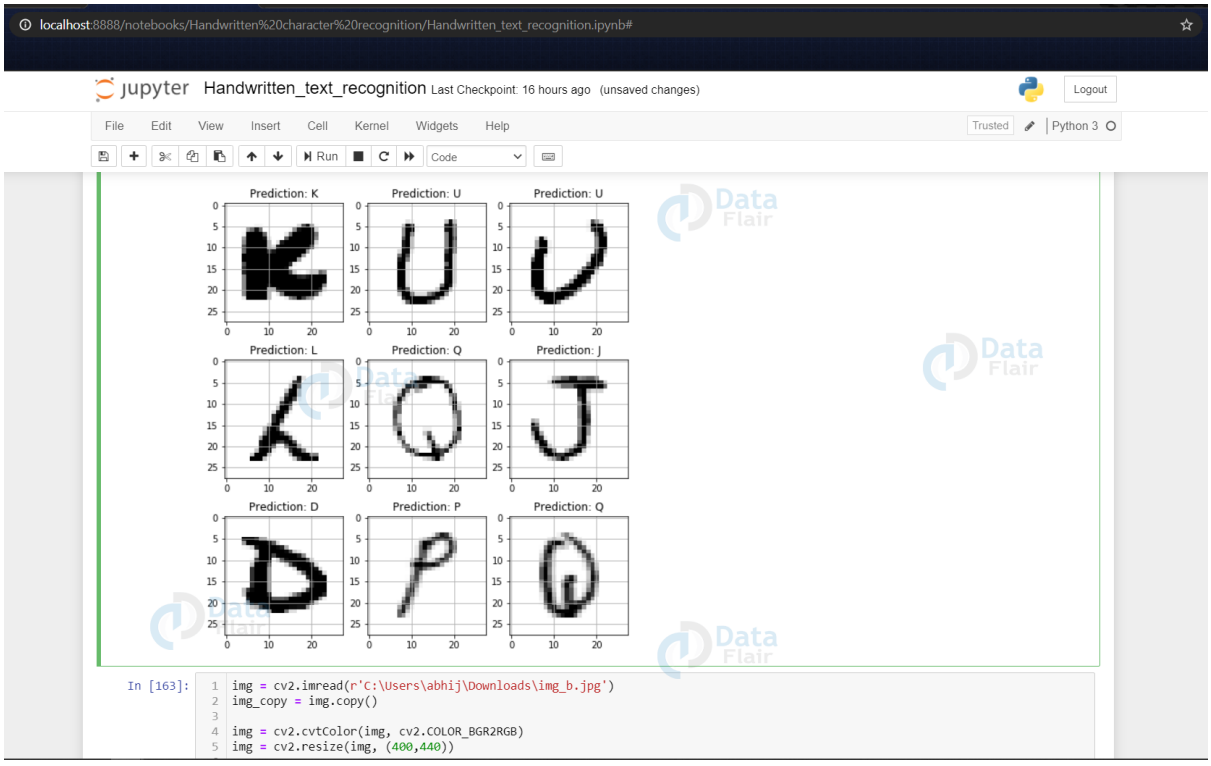
```
# Prediction on external image...

img = cv2.imread(r'Downloads\a.jpg')
img_copy = img.copy()

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400,440))

img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
_, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)

img_final = cv2.resize(img_thresh, (28,28))
img_final =np.reshape(img_final, (1,28,28,1))



img_pred = word_dict[np.argmax(model.predict(img_final))]


cv2.putText(img, "Dataflair _ _ _ ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color = (0,0,230))
cv2.putText(img, "Prediction: " + img_pred, (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3, color = (255,0,30))
cv2.imshow('Dataflair handwritten character recognition _ _ _ ', img)



while (1):
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
cv2.destroyAllWindows()
```

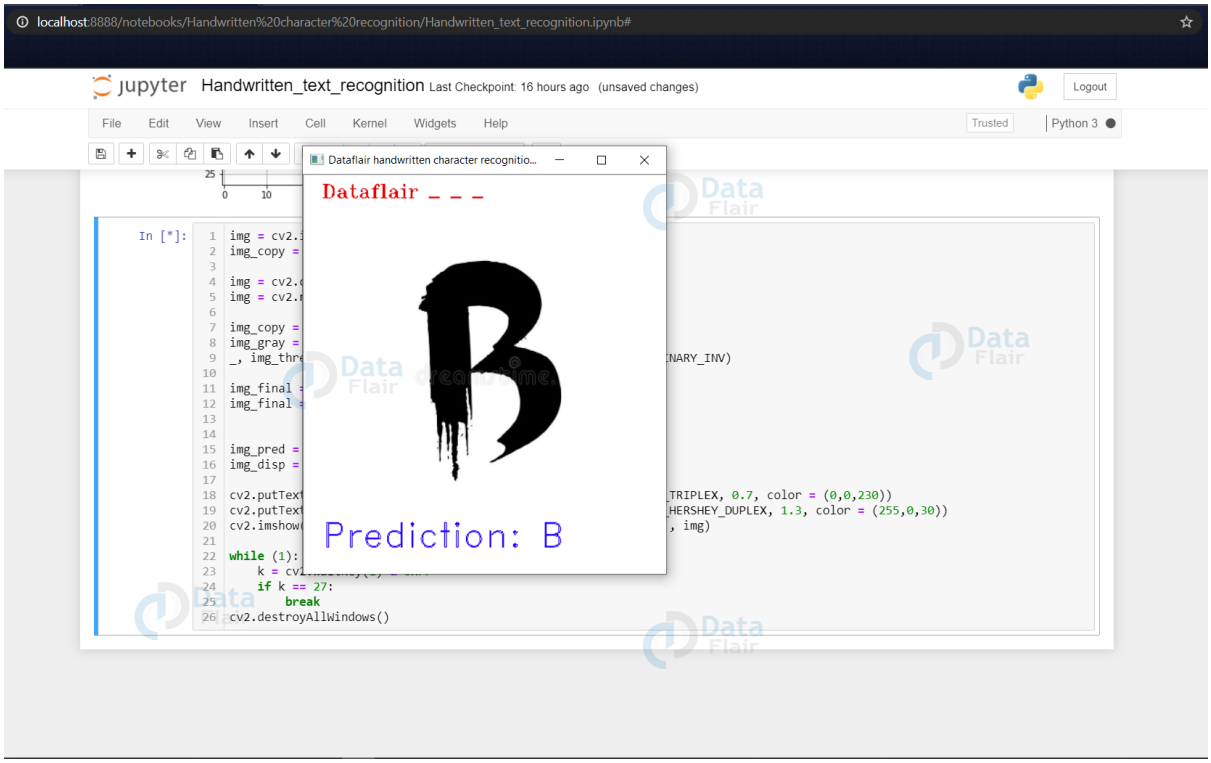
Output:


localhost:8888/notebooks/Handwritten%20character%20recognition/Handwritten_text_recognition.ipynb#
jupyter Handwritten_text_recognition Last Checkpoint: 20 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Run Code
Out[234]:
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 ... 0.639 0.640 0.641 0.642 0.643 0.644 0.645 0.646 0.647 0.648
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
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
2 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
3 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
4 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
5 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
6 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
7 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
8 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
9 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
10 rows × 785 columns
In [ ]:
In [189]:
Reshaping the data in csv file so that it can be displayed as an image...
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)
train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))
In [190]:
print("Train data shape: ", train_x.shape)
print("Test data shape: ", test_x.shape)
Train data shape: (297960, 28, 28)
Test data shape: (74490, 28, 28)
In [99]:
word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P',16:'Q',17


localhost:8888/notebooks/Handwritten%20character%20recognition/Handwritten_text_recognition.ipynb#
jupyter Handwritten_text_recognition Last Checkpoint: 16 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Run Code
18 plt.show()
Alphabets
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
0 10000 20000 30000 40000 50000 60000
Number of elements


localhost:8888/notebooks/Handwritten%20character%20recognition/Handwritten_text_recognition.ipynb#
jupyter Handwritten_text_recognition Last Checkpoint: 16 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
en_charac....py


localhost:8888/notebooks/Handwritten%20character%20recognition/Handwritten_text_recognition.ipynb#
jupyter Handwritten_text_recognition Last Checkpoint: 16 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Prediction: K
Prediction: U
Prediction: U
Prediction: L
Prediction: Q
Prediction: J
Prediction: D
Prediction: P
Prediction: Q
In [163]:
1 img = cv2.imread(r'C:\Users\abhij\Downloads\img_b.jpg')
2 img_copy = img.copy()
3
4 img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
5 img = cv2.resize(img, (400,440))


localhost:8888/notebooks/Handwritten%20character%20recognition/Handwritten_text_recognition.ipynb#
jupyter Handwritten_text_recognition Last Checkpoint: 16 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Dataflair handwritten character recognitio...
Dataflair _ _ _
Prediction: B
In [*]:
img = cv2.
img_copy =
img = cv2.
img = cv2.
img_copy =
img_gray =
_, img_thre
NARY_INV)
img_final =
img_final =
img_pred =
img_disp =
cv2.putText
TRIPLEX, 0.7, color = (0,0,230))
cv2.putText
HERSHEY_DUPLEX, 1.3, color = (255,0,30))
cv2.imshow
, img)
while (1):
k = cv
if k == 27:
break
cv2.destroyAllWindows()