PYTHON LIBRARY FOR CHARACTER RECOGNITION

Project Team

Sl. No.	Reg. No.	Student Name
1	16ETCS002114	Sarah Z Anwar
2	16ETCS002115	Satyam Agrawal
3	16ETCS002144	Ashish Kumar
4	16ETCS002161	Saurav Kumar

Supervisors: Ms. Pallavi R Kumar

MAY – 2019

B. Tech. in Computer Science and Engineering

FACULTY OF ENGINEERING AND TECHNOLOGY

M. S. RAMAIAH UNIVERSITY OF APPLIED SCIENCES
Bengaluru -560 054

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

This is to certify that the Project titled *"Python Library for Character Recognition"* is a bonafide work carried out in the *Department of Computer Science and Engineering* by Ms. Sarah Z Anwar bearing Reg. No. 16ETCS002114 in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.

MAY – 2019

Ms. Pallavi R Kumar
Assistant Professor – Dept. of CSE

Dr. Raghavendra V. Kulkarni   **Dr. M. Arulanantham**
Professor and Head – Dept. of CSE   **Professor and Dean-FET**

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

This is to certify that the Project titled "Python Library for Character Recognition" is a bonafide work carried out in the Department of Computer Science and Engineering by Mr. Ashish Kumar bearing Reg. No. 16ETCS002144 in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.

MAY – 2019

Ms. Pallavi R Kumar
Assistant Professor – Dept. of CSE

Dr. Raghavendra V. Kulkarni   **Dr. M. Arulanantham**
Professor and Head – Dept. of CSE   **Professor and Dean-FET**

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

This is to certify that the Project titled "Python Library for Character Recognition" is a bonafide work carried out in the Department of Computer Science and Engineering by Mr. Satyam Agrawal bearing Reg. No. 16ETCS002115 in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.

MAY – 2019

Ms. Pallavi R Kumar
Assistant Professor – Dept. of CSE

Dr. Raghavendra V. Kulkarni   **Dr. M. Arulanantham**
Professor and Head – Dept. of CSE   **Professor and Dean-FET**

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

This is to certify that the Project titled "Python Library for Character Recognition" is a bonafide work carried out in the Department of Computer Science and Engineering by Mr. Saurav Kumar bearing Reg. No. 16ETCS002161 in partial fulfilment of requirements for the award of B. Tech. Degree in Computer Science and Engineering of Ramaiah University of Applied Sciences.

MAY – 2019

Ms. Pallavi R Kumar
Assistant Professor – Dept. of CSE

Dr. Raghavendra V. Kulkarni   **Dr. M. Arulanantham**
Professor and Head – Dept. of CSE   **Professor and Dean-FET**

<u>Declaration</u>

Python Library for Character Recognition

The project work is submitted in partial fulfilment of academic requirements for the award of B. Tech. Degree in the Department of Computer Science and Engineering of the Faculty of Engineering and Technology of Ramaiah University of Applied Sciences. The project report submitted herewith is a result of our own work and in conformance to the guidelines on plagiarism as laid out in the University Student Handbook. All sections of the text and results which have been obtained from other sources are fully referenced. We understand that cheating and plagiarism constitute a breach of University regulations, hence this project report has been passed through plagiarism check and the report has been submitted to the supervisor.

Sl. No.	Reg. No.	Student Name	Signature
1	16ETCS002114	Sarah Z Anwar	
2	16ETCS002115	Satyam Agrawal	
3	16ETCS002144	Ashish Kumar	
4	16ETCS002161	Saurav Kumar	

Date: **08 MAY 2019**

Acknowledgements

We, the team members, express heartfelt gratitude to all the faculty members and the people who have helped us in any way to build this project.

Summary

Handwritten character recognition is a field of research in artificial intelligence, computer vision, and patter recognition. In the field of Artificial Intelligence, scientists have made many enhancements that helped a lot in the development of millions of smart devices. They brought a revolutionary change in the field of image processing where one of the biggest challenges was to identify documents in both printed as well as handwritten formats. A computer performing handwritten recognition is said to be able to acquire and detect characters in paper documents, pictures, touch-screen devices and other sources and convert them into machine-encoded form.

In the presented model, we will be handling the issue of machine reading English alphabetical figures. We have tried developing such a model that corresponds to the ability of human beings to identify such characters. The objective is to make a system that can classify a given input correctly into the corresponding character class. In today's world, just the development of any model is not the end of the task, the real task lies is making the model efficient, keeping that in mind, we have made the model into a library, which can be imported and used by anyone easily.

The model can be used to convert handwritten or typed data into electronic format. It can be used to store the document in well-ordered form with clarity of writing and it also saves space, protects the document from getting destroyed by mites and saves search time. The data then can be used anywhere, in any field, like database, data analysis, etc. enhancing progress in those fields. To implement it we have used Deep Neural Network, which is a part of supervised machine learning, this technique is mostly used in classification problem. The model consists of 3 hidden layers and an input layer. It gives an accuracy of 99.45% on training model, which indicates the training model used works very accurately for training the model but the test accuracy for the model is not very good for reason like less number of test cases, not use of Convolutional Neural Network etc.

Table of Contents

List of Tables

List of Figures

_____

Lists of Code Listings

Abbreviation and Acronyms

_____

AI	Artificial Intelligence
ANN	Artificial Neural network
CNN	Convolutional Neural network
CV	Computer Vision
CSV	Comma Separated Values
Colab	Collaboration
DNN	Deep Neural Network
GD	Gradient Descent
KNN	K Nearest Neighbours
ML	Machine Learning
OCR	Optical Character Recognition
PDA	Personal Digital Assistant
PIL	Python Image Library
ReLU	Rectified Linear Unit
TF	TensorFlow

1. Introduction

This chapter introduces the motivation for the project, its context and scope. It identifies the domain of application, justifying its necessity and potential benefits to the society and organizations. It outlines the assumptions and likely observation of the system.

1.1 Introduction:

Handling large amount of manual files are not an easy task, the room full of document that are very important and may be necessary to be looked into any time. To search a file from the room full of files sounds like a hectic job but this is the situation on most of the government offices in our country and many other sectors and companies. A person looking for a file may lose temper may not be able to find the file at all or may spoil other file during the search due to frustration, all these are possible since it's a human searching for a file in the large pile of files.

This intuition helps us realize the necessity of a character recognition library in every office. The library can be used to convert all the files and documents into electronic form which can be stored in a hard drive which consumes way less amount of space to store all the files. The file stores in the disk are easy to search since there exists so many tools to search through the files hence we have a whole room empty which can be used for various different works also reducing the stress of the person who is trying to search for a file. The files which could be damaged or destroyed by mites will be protected from any physical damage. The files will also be portable easily and can have many copied of the files if it is needed to be present in different places at the same time. The data obtained from the files can also be used for data analysis and data interpretation for other research and study.

In these times, there is a great requirement of this kind of a recognition system. In today's world, where our main focus or emphasis is on employing efficient and cost-effective methods, many people are now using technology to perform tasks that have previously been laborious as well as time consuming. The essence of our modern gadgets is their efficiency as well as their role in reducing labour intensity. Computers are now used as a well-ordered and well regulated alternative for taking and managing notes and for eradicating potential issues regarding clarity of handwriting or misplacing sheets of papers.

Additional advantages of preferring technology involve the comfort of making such notes better at a later time, as well as reducing the wastage of physical space. One more example regarding the benefits of the present technology is utilization of tablets for both professionals and students, conserving the need for several books and various important documents, once more preventing wastage of extra physical space and letting all necessary data be kept inside a small piece of equipment. Data management has gained considerable prominence in engineering executions. Given that the industry sector is consistently increasing throughout the past several decades, a significantly greater demand over automated machinery can be observed. These automated machineries depend heavily towards data management in the form of character identification utilities. A simple example for such can be comparing data management akin to organization of mails at a relevant distribution complex, under which all data is processed by OCR platforms. Then, decisions are undertaken through automated machinery arms to sort the correct mail towards the correct lane so that all mails are sorted in relevance to the areas they need be delivered.

1.2 Scope:

Character identification is a procedure that allows potency of diverse functions featuring mitigated manual input, thus limiting bias and mitigating mistakes. This project will involve developing a library function that helps detect the character image. It will consist of surveying and studying the literature available for character recognition, Image processing techniques for developing the test cases and Neural network approach for implementation of the model.

The objectives will be formulated and methods and methodologies to accomplish each objective will be specified. In this project, TensorFlow has been used which provide abstract function for implementation and development of the model. Instead of dealing with the nitty-gritty details of implementing algorithms, or figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application. TensorFlow takes care of the details behind the scenes.

The artefacts of the system will be specified, visualized, constructed using a Flowchart and its algorithm. The requirements of the system will be gathered and formulated. The algorithm will be implemented using Python 3.5 over the google collaboration platform which provide necessary hardware to develop a deep neural network model as the available hardware are not sufficient to satisfy the requirement for the model training. The training and test images of handwritten text document will be imported using Google drive.

The implementation will be tested, its performance, accuracy and correctness of the detection, recognition will be studied upon and concluded with the reasons of the drawback.

A well-documented report will be written in stages that documents the cumulative effort. The report will also discuss the benefits, accomplishments, limitations and required future work for the system.

1.3 Organization of the report :

The second chapter, Background Theory, gives a brief information about the previously used concepts involved in the development of the tool. It describes the technology and algorithms used in the presented model and justify the reason for choosing them. The theory is written in different sub-sections, with each sub-section pertaining to different techniques, describing its use in the system and other applications.

The third chapter contains the title, aim and objective of the project. Every project/application has an aim, which serves the purpose for the development of the application. The objectives of the project for achieving the aim are mentioned, and the steps of progress and development are described. To complete the objectives, there are certain methods and methodologies that are followed. The chapter describes each objective's methods and methodologies and the corresponding resources required.

The fourth chapter, Problem Solving, introduces the approach used and describes how the tool was developed step by step in a way that accomplishes each of the objectives using the mentioned methods and methodologies. The data collection methods are mentioned along with the modelling of the system. The implementation of the system is introduced and explained, as well as how the objectives are achieved by the implementation. The testing process to understand the performance and accuracy is described subsequently.

The fifth chapter, Results, presents the results of the implemented and executed system. It discusses the objectives accomplished by the implemented system and how it meets

the outlined objectives. It provides a detailed explanation for the output given by the system.

The sixth chapter, Project Costing, reports the expenses incurred. The chapter enlists the cost incurred to develop the library function. The cost of the design, implement, test and document the effort is also provided.

The seventh chapter, Conclusion and Future Work, concludes the project work, providing a short summary of the work done, and the difficulties faced during the development. It notes the benefits and gains provided by the tool, and why companies/organizations should require it. The future work required is discussed as to how the tool can be extended, improved and augmented to provide further functionalities, better performance and accuracy and other domains of application.

2. Background Theory

Based on the proposed work in chapter 1, it is essential to understand Deep Learning. This chapter introduces Deep learning and how a neural network works for classification problems. The chapter briefly describes the origin and improvements in character recognition problem over the years. It also explains the working of Deep learning and its applications. The chapter describes the available techniques for character recognition and justifies the choice of techniques used to implement the model.

2.1 Introduction to deep learning:

Deep learning is part of a broader family of machine learning methods based on the layers used in artificial neural networks. Deep learning is an aspect of artificial intelligence (AI) that is concerned with emulating the learning approach that human beings use to gain certain types of knowledge. At its simplest, deep learning can be thought of as a way to automate predictive analytics. While traditional machine learning algorithms are linear, deep learning algorithms are stacked in a hierarchy of increasing complexity and abstraction.

2.2 Brief History:

The history of Deep Learning can be traced back to 1943, when Walter Pitts and Warren McCulloch created a computer model based on the neural networks of the human brain. They used a combination of algorithms and mathematics they called "threshold logic" to mimic the thought process. Henry J. Kelley is given credit for developing the basics of a continuous Back Propagation Model in 1960. In 1962, a simpler version based only on the chain rule was developed by Stuart Dreyfus. While the concept of back propagation (the backward propagation of errors for purposes of training) did exist in the early 1960s, it was clumsy and inefficient, and would not become useful until 1985.
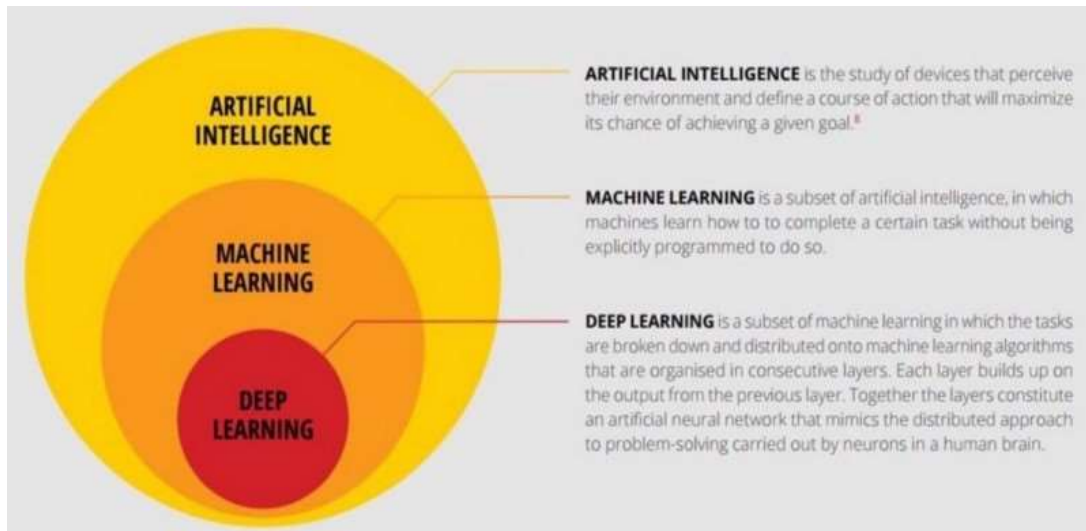
Figure 2.1 Deep learning a subset of machine learning which is a subset of artificial intelligence [5].

The earliest efforts in developing Deep Learning algorithms came from Alexey Grigoryevich Ivakhnenko (developed the Group Method of Data Handling) and Valentin Grigor'evich Lapa (author of Cybernetics and Forecasting Techniques) in 1965. They used models with polynomial (complicated equations) activation functions, that were then analyzed statistically. From each layer, the best statistically chosen features were then forwarded on to the next layer (a slow, manual process).

Back propagation, the use of errors in training Deep Learning models, evolved significantly in 1970. This was when Seppo Linnainmaa wrote his master's thesis, including a FORTRAN code for back propagation. Unfortunately, the concept was not applied to neural networks until 1985. This was when Rumelhart, Williams, and Hinton demonstrated back propagation in a neural network could provide "interesting" distribution representations. Philosophically, this discovery brought to light the question within cognitive psychology of whether human understanding relies on symbolic logic (computationalism) or distributed representations (connectionism). In 1989, Yann LeCun provided the first practical demonstration of backpropagation at Bell Labs. He combined

convolutional neural networks with back propagation onto read "handwritten" digits. This system was eventually used to read the numbers of handwritten checks.

2.3 About machine learning:

Machine learning is the idea that there are generic algorithms that can tell you something interesting about a set of data without you having to write any custom code specific to the problem. Instead of writing code, you feed data to the generic algorithm and it builds its own logic based on the data. It can be broadly divided into two main categories –Supervised Learning and Unsupervised learning.

2.3.1 Supervised learning

This algorithm is used to predict the future using the past information. This algorithm starts with training with a known dataset and generates a function to predict the output. The precision of the prediction function improves with better data and appropriate size of the dataset. The System compares the actual output with the predicted output in order to learn and modify its parameters.

2.3.2 Unsupervised learning

These algorithms are opposite to that of supervised algorithms. It has only the input data and no information about the output. The algorithm studies the data to recognize the structure and distribution. These can be classified as clustering and association algorithms. Clustering algorithms attempt to identify inherent groupings while association algorithms attempt to discover rules that describe larger portions of the data.

2.4 About TensorFlow

Google's Tensorflow is the most famous deep learning library in the world. It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**.

You can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output. It allows developers to create large-scale neural networks with many layers. **TensorFlow** is mainly **used** for: Classification, Perception, Understanding, Discovering, Prediction and Creation.

2.5 Working of Deep learning:

Deep learning models work in layers and a typical model at least have three layers. Each layer accepts the information from previous and pass it on to the next one. Deep learning models tend to perform well with amount of data where as old machine learning models stops improving after a saturation point. One of differences between machine learning and deep learning model is on the feature extraction area. Feature extraction is done by human in machine learning whereas deep learning model figure out by itself.

2.5.1 Activation Function:

Activation functions are functions that decide, given the inputs into the node, what should be the node's output? Because it's the activation function that decides the actual output, we often refer to the outputs of a layer as its "activations". One of the simplest activation functions is the Heaviside step function. This function returns a 0 if the linear combination is less than 0. It returns a 1 if the linear combination is positive or equal to zero.

$$f(H) = \begin{cases} 0, & H < 0 \\ 1, & H \geq 0 \end{cases}$$

The output unit returns the result of $f(H)$, where h is the input to the output unit.

2.5.2 Weights:

When input data comes into a neuron, it gets multiplied by a weight value that is assigned to this particular input. These weights start out as random values, and as the neural network learns more about what kind of input data leads to a student being accepted into a university, the network adjusts the weights based on any errors in categorization that the previous weights resulted in. This is called training the neural network.

We can associate weight as m (slope) in the original linear equation:

$$y = mx + b$$

2.5.3 Neural Network

A neural network is a connected graph with input neurons, output neurons, and weighted edges. A neural network is a graph of neurons. A neuron has inputs and outputs. It consists of connections, each connection transferring the output of a neuron to the input of another neuron. Each connection is assigned a weight. The propagation function computes the input of a neuron from the outputs of predecessor neurons. The propagation function is leveraged during the forward propagation stage of training.

2.5.4 Forward propagation

By propagating values from the first layer (the input layer) through all the mathematical functions represented by each node, the network outputs a value. This process is called a forward pass.
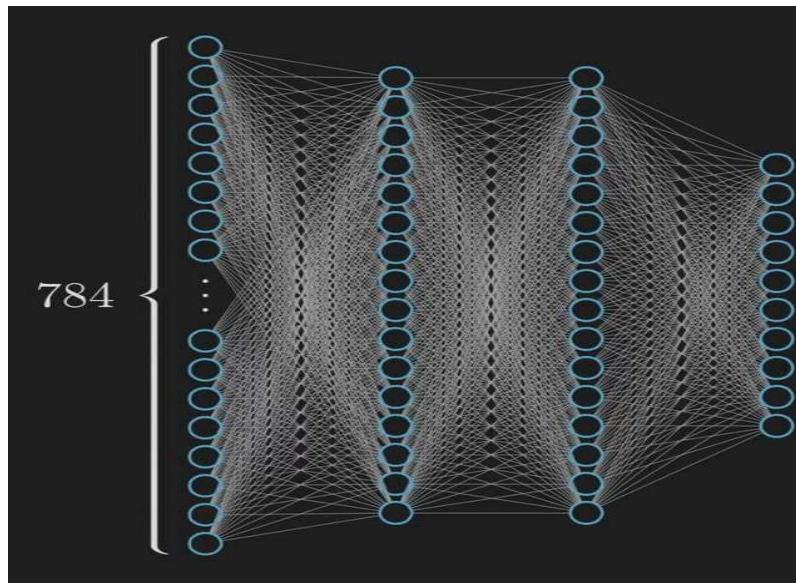
Figure 2.2 Neural Network [6].

2.5.5 Optimization Algorithm

Optimization algorithm is used to find the minimum error by minimizing a "cost" function. There are various optimization algorithms that can be used in different situations. The most commonly used one is Gradient Descent. Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm. It takes multiple small steps towards our goal and change the weights in steps that reduce the error. Fastest way to get to the goal is by taking step in the steepest direction.

2.5.6 Backpropagation

Forward propagate is used to get the output and compare it with the real value to get the error. Now, to minimise the error, you propagate backwards by finding the derivative of error with respect to each weight and then subtracting this value from the weight value. This is called back propagation. We have seen how to update weights with gradient descent. The back propagation algorithm is just an extension of that, using the

chain rule to find the error with the respect to the weights connecting the input layer to the hidden layer.

2.5.7 Regularization

Regularisation is the technique used to solve the over-fitting problem. Over-fitting happens when model is biased to one type of datset. There are different types of regularisation techniques, I think the mostly used regularisation is dropout. We can set the number of neurons for the dropout. Their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

2.6 Why Deep Learning for Classification problem?

The major distinguishing factor of deep learning compared to more traditional methods is the ability of the performance of the classifiers to large scaled with increased in quantities of data.

Older machine learning algorithms typically plateau in performance after it reaches a threshold of training data. Deep learning is one-of-a-kind algorithm whose performance continues to improve as more the data fed, the more the classifier is trained on resulting in outperforming more than the traditional models/ algorithm.

The execution time is comparatively more for deep learning, as it needed to be trained with lots of data. The major drawback of this ability to scale with additional training data is a need for trusted data that can be used to train the model. While the world is generating exponentially more data every year, the majority of this data is unstructured, and therefore currently unusable.

The software learns, in a very realistic sense, to recognize patterns in digital representations of images, sounds, censor data and other data. We are pre-training data, in order to classify or predict and build a train/training set and test set (we know the result). An optimal point of prediction is obtained such that, our prediction gives a satisfying result.

The neurons are based out in different level and made to make their prediction at each level and most-optimal predictions, and then use the data in order to give a best-fit outcome. It is considered as true intelligence on machine.
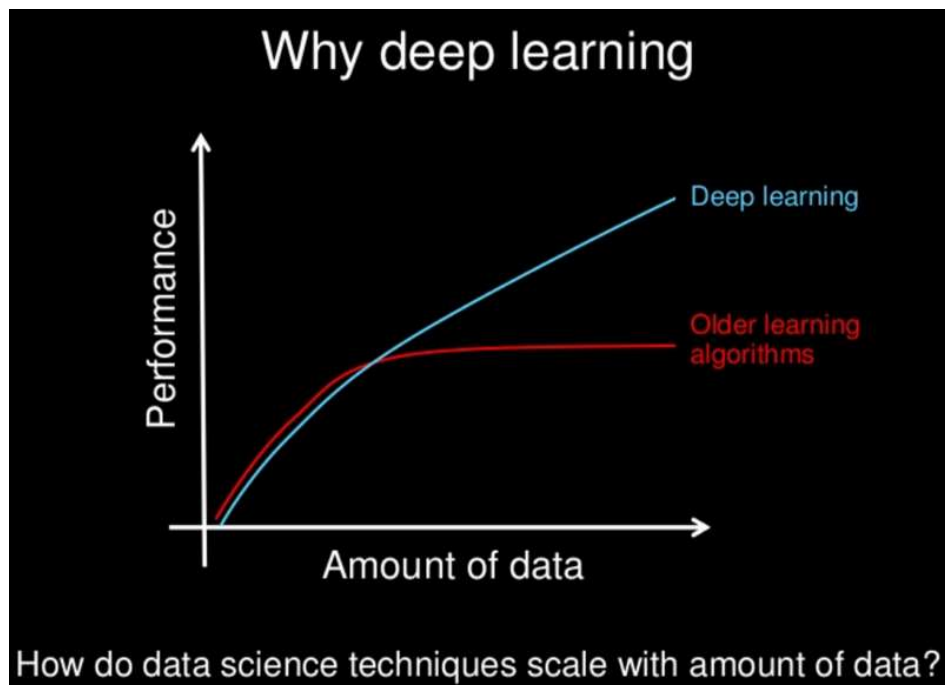
Figure 2.3 Why deep learning? [7].

2.7 Application of Deep Learning

1. Computer vision and pattern recognition

2. Robotics — Deep Learning systems have been taught to play games and even made to taught WIN games.

3. Facial recognition

4. Precision agriculture

5. Fashion technology

6. Autonomous vehicles

7. Drone and 3D mapping

8. Post estimation in Sports analytics & Retail markets

9. Security & Surveillance

10. Satellite imagery

11. Audio / Voice recognition

12. Restoring sound in videos

13. Text OCR on documents, Predicting the result of legal case a team of researchers from British and America built an algorithm by feeding with few examples and information, that was able predict a court's decision.

2.8 The available techniques for character recognition:

2.8.1. Online Direction based algorithm

On-line handwriting recognition involves the automatic conversion of text as it is written on a special digitizer or PDA, where a sensor picks up the pen-tip movements as well as pen-up/pen-down switching. That kind of data is known as digital ink and can be regarded as a dynamic representation of handwriting. The obtained signal is converted into letter codes, which are usable within computer and text-processing applications. The elements of an on-line handwriting recognition interface typically include a pen or stylus for the user to write with, a touch sensitive surface, which may be integrated with, or adjacent to, an output display and a software application, which interprets the

movements of the stylus across the writing surface, translating the resulting, strokes into digital text.

2.8.2. K-NN Classifier and DTW-Based Dissimilarity Measure

Classification is carried out by evaluating the dissimilarity measures between the pre-processed input character and all the training samples and then applying the nearest neighbour rule. Its major drawback is that it is computationally heavy, especially with large prototype sets and complicated similarity measures. For similarity measure, it uses DTW measure between two strokes. DTW is an elastic matching technique that gives distance measure between character pairs. A stroke is a sequence of sample points from pen down to pen up events. The distance between strokes is calculated by considering all possible alignments between them, and finding the alignment for which the total distance is minimum using dynamic programming.

2.8.3. Clustering

The goal of a clustering analysis is to divide a given set of data or objects into a cluster, which represents subsets or a group. The membership functions do not reflect the actual data distribution in the input and the output spaces. They may not be suitable for fuzzy pattern recognition. To build membership functions from the data available, a clustering technique may be used to partition the data, and then produce membership functions from the resultant clusters. Thus, the characters with similar features are in one cluster. Thus, in recognition process, the cluster is identified first and then the actual character.

2.8.4. Feature Extraction

The idea of feature point extraction algorithm is to identify characters based on features that are somewhat similar to the features humans use to identify characters. Programmers must manually determine the properties they feel are important. Some example properties might be Aspect Ratio, Percent of pixels above horizontal half point,

Percent of pixels to right of vertical half point, Number of strokes , Average distance from image centre , Is reflected y axis , Is reflected x axis. Researchers have used many methods of feature extraction for handwritten characters. Shadow code, fractal code, profiles, moment, template, structural (points, primitives), wavelet, directional feature etc., have been addressed in the literature as features. This approach gives the recognizer more control over the properties used in identification. Yet any system using this approach requires substantially more development time than a neural network because the properties are not learned automatically. Selection of a feature extraction method is probably the single most important factor in achieving high recognition performance in character recognition systems.

2.8.5. Pattern matching

In Pattern Matching, a character is identified by analysing its shape and comparing its features that distinguish each character. Various handwritten characters from forms or peripheral devices etc are recognized with the help of various pre-processing and image enhancement techniques. These characters are further more specifically recognized by Pattern matching using Neural Network. The extraction of discriminative features is crucial to the recognition performance of pattern matching. Though sophisticated classification is now available for character recognition, pattern matching still survives for, e.g., coarse classification, and the evaluation of features with pattern matching makes sense for various classifiers.

2.8.6. Artificial Neural Network

A neural network is a set of connected input/output units in which each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input values. Neural Network learning is also known as connectionist learning due to the connection between units. Neural network uses Back propagation, which is a technique and a supervised

algorithm that learns by first computing the output using a feed forward network, then, calculating the error signal and propagating the error backwards through the network.

2.9 Conclusion

The character recognition methods have developed remarkably in the last decade. A variety of techniques have emerged, influenced by developments in related fields such as image recognition and face recognition. In this paper, we have proposed an organization of these methods under two basic strategies. It is hoped that this comprehensive discussion will provide insight into the concepts involved, and perhaps provoke further advances in the area. The difficulty of performing accurate recognition is determined by the nature of the text to be read and by its quality. Generally, improper segmentation rates for unconstrained material increase progressively from machine print to handprint to cursive writing.

3. Aim and Objectives

Based on the survey in the previous chapter and the proposal in the first chapter, it is now convenient to create a roadmap for the development of the proposed project. This chapter states the title and aim of the project. The objectives to be met in the development of the system are discussed, along with the corresponding methods and methodologies to accomplish objectives.

3.1. Title

❖ PYTHON LIBRARY FOR CHARACTER RECOGNITION

3.2. Aim

❖ Design and develop a library for English Character Recognition

3.3. Objectives

❖ To conduct literature survey on Character Recognition/ textual image processing

❖ To design the methodology to be followed for developing the character recognition model.

❖ To develop and implement the algorithm that recognizes a manual character into a digital text document.

❖ To test and validate the developed model for various textual images

❖ To document the report by unifying all the results and outcomes.

3.4. Methods and Methodology/Approach to attain each objective

Objective No.	Statement of the Objective	Method/ Methodology	Resources Utilised
1	To conduct literature survey on Character Recognition/ textual image processing	- Existing books, papers and articles will be surveyed for understanding the working of Image Processing. - Existing books, papers and research papers will be surveyed to identify factors considered for learning Neural-network for recognition/ classification. - Existing books, papers will be surveyed for state of art in algorithms and models that can be applied in character recognition.	Books on handwritten recognition and image processing. Technical articles on Neural network, Deep learning, Convolutional Neural Network and image processing. Internet, YouTube videos, Coursera tutorials
2	To design the methodology to be followed for developing the character	- Based on the research, a suitable way to develop the model will be taken up. - Based on the different	Used Dia for creating the flow chart. Used PowerPoint for presentations

	recognition model.	procedures that will be deduced, a flowchart will be created. - A library will be made following the flowchart of the model.	
3	To develop and implement the algorithm that recognizes a manual character into a digital text document.	- Based on the designed flowchart, a model is developed using python programing language. - Using the developed model, the library is created that will considerably give an output of the class of the character recognized.	Jupyter notebook, Python, Anaconda, Google Colab, Google Drive, Laptop, Camera.
4	To test and validate the developed model for various textual images	- Based on the design, the library will be tested on various images of the characters. - The performance analysis of the library with different set of test cases will be studied.	Jupyter notebook, Python, Anaconda, Google Colab, Google Drive, Laptop, Camera.
5	To document the report by unifying	- Develop a scientific project report as per the	PowerPoint for presentations and

all the results and outcomes.	template specified. - Display and demonstrate in university organized project exhibition and demonstrations.	Microsoft word for documentation.

3.5 Summary

This chapter specifies the objectives of the proposed project. It also lists the methods and methodologies to be adopted in the order to achieve the specified objectives. These include literature survey, then designing the prototype followed by implementing, testing and analyzing the developed tool, and then generating a technical report over it.

4. Library development procedure

Based on the methodology discussed in the previous chapter, this chapter details the development process of the prototype, following the methodologies to accomplish each objective as described in the previous chapter. It provides the deliverables obtained at the end of each stage in the process. The chapter describes the analysis and design models obtained and flow charts for implementation. It explains the implemented prototype system with code listings. The test stages are clearly explained, providing the test suite, objective and results of testing. Finally, the analysis of the system for accuracy and scalability is described.

4.1 Problem solving Approach

The character recognition problem is broken down into three modules the image pre-processing for test cases, the training of the model and the prediction (testing). The training data is obtained from kaggle.com, which consists of 372449 images standardized and converted into csv form. The training model is developed using the flowchart developed during the design of the model. The model uses deep learning for training the model and test image are produced by us manually for predicting the result and accuracy of the model.

4.2 Requirements and Assumptions

The model is required to take handwritten document (image) as input dataset, pre-process the image, pass through the trained the neural network model to recognize the characters and then store the recognized characters into a text document.

The model cannot be as accurate as human at classifying the characters so to have a better performance of the model we restricted the model to capital English character recognition only. The model was train under various assumption for the user to get the maximum benefit from it. The assumptions made for the model are as follows:

- The input image is assumed to contain only English characters.

- The input image is assumed to be a text written on un-ruled white paper.

- The input image is assumed to contain text that are not cursive.

- The input image is assumed to contain text that does not contain noise e.g. ~~Ashish~~.

These assumptions make the model more reliable and specific to its application. The model cannot be as good as human so we have to define the model in such a way that the model should at least return all that it promises. Failing to fulfil the promise made is a big failure on the part of the model and the developer.
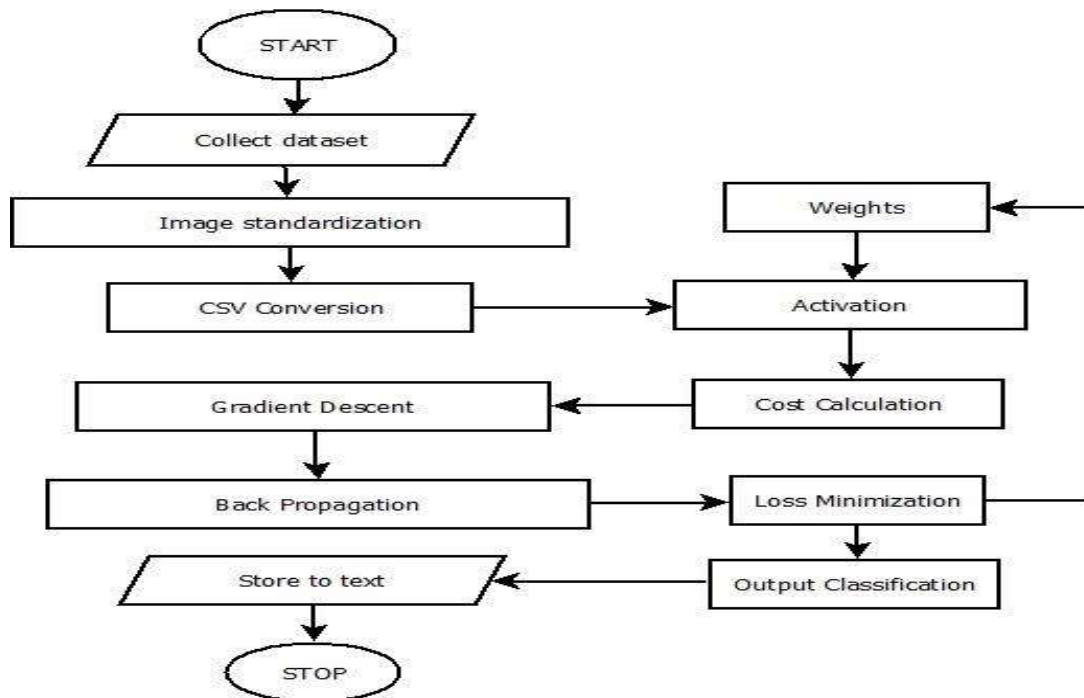
4.3 Flowchart

Figure 4.1 Flowchart

In the figure 4.1, the flowchart shows the flow of the data and processes used during the implementation of the model.

The first step is to collect the data required for training the model then the data is standardized into grey-scale and converted or first converted into csv than standardized. If the data is already in csv format we should check the values of the data whether is standardized or not, if not standardized then standardize it before using it to train the model. After the dataset is stored as csv format it is passed into the model where the model trains itself by varying the values initialized to weight to minimize the error. To do that the model passed through a number of steps. The activation step is the linear output of the input data and weight, which helps in determining how correct, or incorrect is the model. The next step is cost calculation where actually the error is calculated. Then comes the gradient descent step where the weights are adjusted to decrease the error. The next step is backpropagation where the derivative of the error is calculated it is an extension to the gradient descent step. The next step is loss minimisation where the weight values are used to reduce the error is the model. This continues till the error is minimum and we get an output class as the output of the model with the trained values of the weights. The output class can be used to classify the characters and store it into text document.

4.3 Image pre-processing

Pre-processing is required for coloured, binary or grey-level images containing text. Most of the algorithms of character recognition works on binary/grey – level image because the computation is difficult for coloured images. Images may contain background or watermark or any other thing different from text making it difficult to extract the text from the scanned image. So, Pre-processing helps in removing the above difficulties.

The result after Pre-processing is the binary/grey – level image containing text only. Thus, to achieve this, several steps are needed.

4.3.1   Noise reduction

When the document is scanned, it might be contaminated by additive noise and these low quality images will affect the next step of image processing. Therefore, a pre-

processing step is required to improve the quality of images before sending them to subsequent stages of image processing. So, it is very essential to remove all of these errors so that the information can be retrieved in the best way. There are many kinds of noise in images. One additive noise called 'Salt and Pepper Noise', the black points and white points sprinkled all over an image, typically looks like salt and pepper, which can be found in almost all images. Noise reduction techniques can be categorized in two major groups as filtering, morphological operations.

4.3.1.1 Filtering

It aims to remove noise and diminish spurious points, usually introduced by uneven writing surface and/or poor sampling rate of the data acquisition device. Various spatial and frequency domain filters can be designed for this purpose.

4.3.1.2 Morphological Operations

Morphological operations are commonly used as a tool in image processing for extracting image components that are useful in the representation and description of region shape. Morphological operations can be successfully used to remove the noise on the document images due to low quality of paper and ink, as well as erratic hand movement.

4.3.2   Thresholding

It is a process of converting a grayscale input image to a binary image by using an optimal threshold. The purpose of thresholding is to extract those pixels from some image, which represent an object (either text or other line image data such as graphs, maps). Though the information is binary, the pixels represent a range of intensities. Thus, the objective of binarization is to mark pixels that belong to true foreground regions with a single intensity and background regions with different intensities.

4.3.3 Implementation

First, open the target image by providing its location with the help of **'image'** function, which is present in package **'PIL'** as shown in the figure below.

```
[ ]    from PIL import Image
       with open('location of the target image', 'r+b') as f:
```

Figure 4.1: Loading the required image.

Now change the pixel size of the image according to the requirement, in our case, the image should be in **'28x28'** pixel size using the **'resizeimage'** function which is present in **'resizeimage'** package as shown in the figure below.

```
[ ]    from PIL import Image
       from resizeimage import resizeimage
       with open('location of the target image', 'r+b') as f:
           with Image.open(f) as image:
               cover = resizeimage.resize_cover(image, [28, 28], validate=False)
```

Figure 4.2: Resizing the image.

Now save the image which has the new dimensions into some location using the **'save'** function which is present in same package i.e., **'resizeimage'** as shown in the figure below.

```
[ ]    from PIL import Image
       from resizeimage import resizeimage
       with open('location of the target image', 'r+b') as f:
           with Image.open(f) as image:
               cover = resizeimage.resize_cover(image, [28, 28], validate=False)
               cover.save('location of the new (changed) image', image.format)
```

Figure 4.3: Saving the resized image.

Now open the new image (which has 28x28 dimension) using **'imread'** function which is present in **'cv2'** (cv2 is also called OpenCV) package as shown in the figure below. Next step is to extract the pixel values of the image. As the image is a coloured image so, each pixel will have three value i.e., RGB, one value for each colour (Red, Green and Blue). But, we need single value from each pixel. Therefore, the image will be converted to

grayscale values using the **'cvt.Color'** function, which is present in same package i.e., **'cv2'** as shown in the figure below.

```
[ ]    import cv2
       from PIL import Image
       from resizeimage import resizeimage
       with open('location of the target image', 'r+b') as f:
           with Image.open(f) as image:
               cover = resizeimage.resize_cover(image, [28, 28], validate=False)
               cover.save('location of the new (changed) image', image.format)
       img = cv2.imread('location of the new (changed) target image')
       grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Figure 4.4: Converting the resized image to grayscale.

Next step is to store these pixel values into a csv file. The output of the grayscale conversion is the nested list (list of lists) which is not supported by CSV file. CSV file only supports the data, which is in array format. Therefore, the extracted grayscale values (which are in nested list format) has to be converted to array format with the help of the **'numpy'** array conversion. 'Numpy' is a package, which has function **'array'** to convert nested list into array as shown in the figure below.

```
[ ]    import cv2
       import numpy as np
       from PIL import Image
       from resizeimage import resizeimage
       with open('location of the target image', 'r+b') as f:
           with Image.open(f) as image:
               cover = resizeimage.resize_cover(image, [28, 28], validate=False)
               cover.save('location of the new (changed) image', image.format)
       img = cv2.imread('location of the new (changed) target image')
       grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
       npArray = np.array([ elem for singleList in grayscaled for elem in singleList])
```

Figure 4.5: Converting gray-scaled values to array.

Now store the pixel values (which are now in array format) into the CSV file (the data has comma i.e., **' , '** as the delimiter) by providing the location of the file with the help of, **'savetxt'** function which is present in **'numpy'** package as shown in the figure below.

```
[ ]  import cv2
     import numpy as np
     from PIL import Image
     from resizeimage import resizeimage
     with open('location of the target image', 'r+b') as f:
         with Image.open(f) as image:
             cover = resizeimage.resize_cover(image, [28, 28], validate=False)
             cover.save('location of the new (changed) image', image.format)
     img = cv2.imread('location of the new (changed) target image')
     grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
     npArray = np.array([ elem for singleList in grayscaled for elem in singleList])
     np.savetxt('location of the csv file', npArray, delimiter=",")
```

Figure 4.6: Saving grayscale values into CSV file.

4.4 The Model

4.4.1 Training dataset

The training dataset consists of 3, 72,449 image collected from kaggle.com. The dataset consists of images that were in 28*28 pixel that has been converted into csv format. The data contains images of 26 classes, which are the 26 capital letters of the English alphabet.
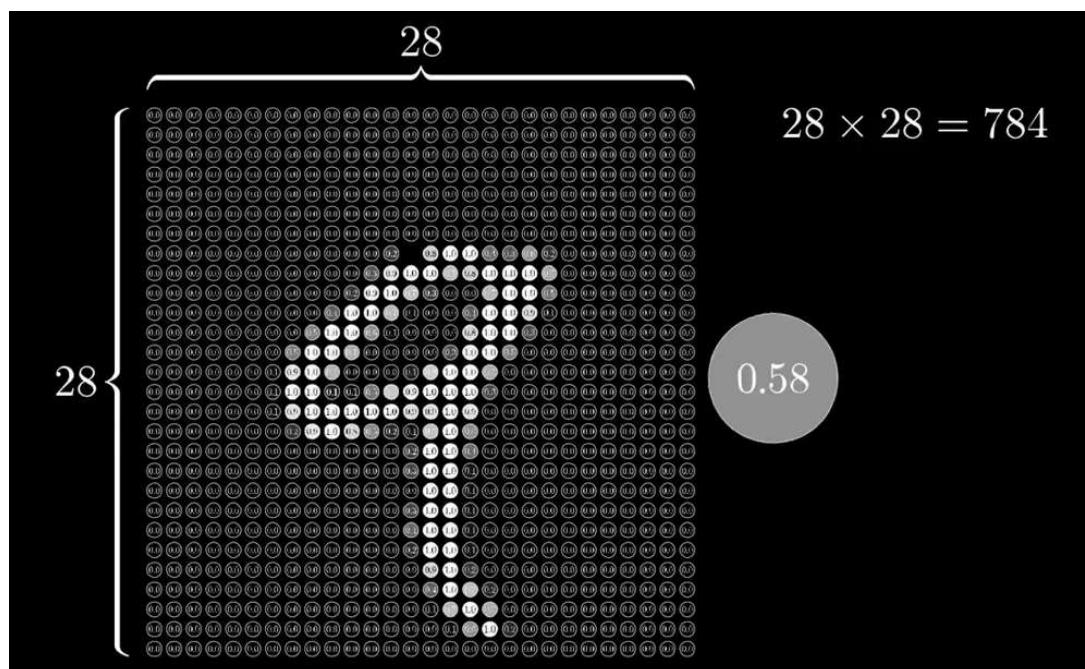

Figure 4.2 Pixel wise representation of the image from the dataset [8].

Classifying such a big dataset into so many classes requires the neural network to be big and well trained. To accomplish this, we have used deep learning consisting of three hidden layers where the first layer contains 100 neurons, the second layer contains 50 neurons, and the last layer of the output layer contains 26 neurons for the 26 classes. Where the input layer has 784(28*28) units. Each neuron in the different layers are supposed to identify different features from the image. The feature of the image like

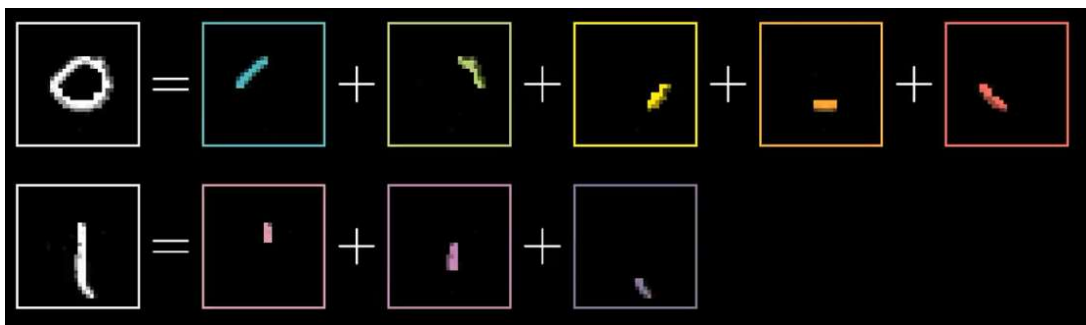O and I can be broken down into following:

Figure 4.3 Feature extraction [8]

4.4.2 Weights and biases

The weights and biases are the variables of the model, which are supposed to be initialized with certain value. They act as a support value to the input features from the dataset. These identify the values of the feature in an image. The weights in the trained model were initialized with Xavier Initializer. The Xavier initializer work better than any other random initializer for the weights and biases specially, when coupled with ReLU activation function. Whereas the bases are always initialized with zeros hence it the model also it has been initialized with zeros. The weights and biases are the parameters in the model, which are trained when the model trains, they are trained by minimizing the cost of error. The biases for an example/neuron remains same but the weights differ for each value of the input/input neuron. The following snippet show the initialization of the weights and biases in the model:

```
W1 = tf.get_variable("W1", [100,784], initializer = tf.contrib.layers.xavier_initializer(seed = 1))
b1 = tf.get_variable("b1", [100,1], initializer = tf.zeros_initializer())
W2 = tf.get_variable("W2", [50,100], initializer = tf.contrib.layers.xavier_initializer(seed = 1))
b2 = tf.get_variable("b2", [50,1], initializer = tf.zeros_initializer())
W3 = tf.get_variable("W3", [26,50], initializer = tf.contrib.layers.xavier_initializer(seed = 1))
b3 = tf.get_variable("b3", [26,1], initializer = tf.zeros_initializer())
```

Figure 4.7 weights and biases initialization.

4.4.3 Forward propagation

Forward propagation is the propagation of values of input layer through each neuron of each layer in the network to the output of the output layer is forward propagation. Each layer in the network has an activation function that is performed on the input values to the layer and the weights and biases for that neuron of that layer. There are many activation functions that are useful in different types of classification problem.
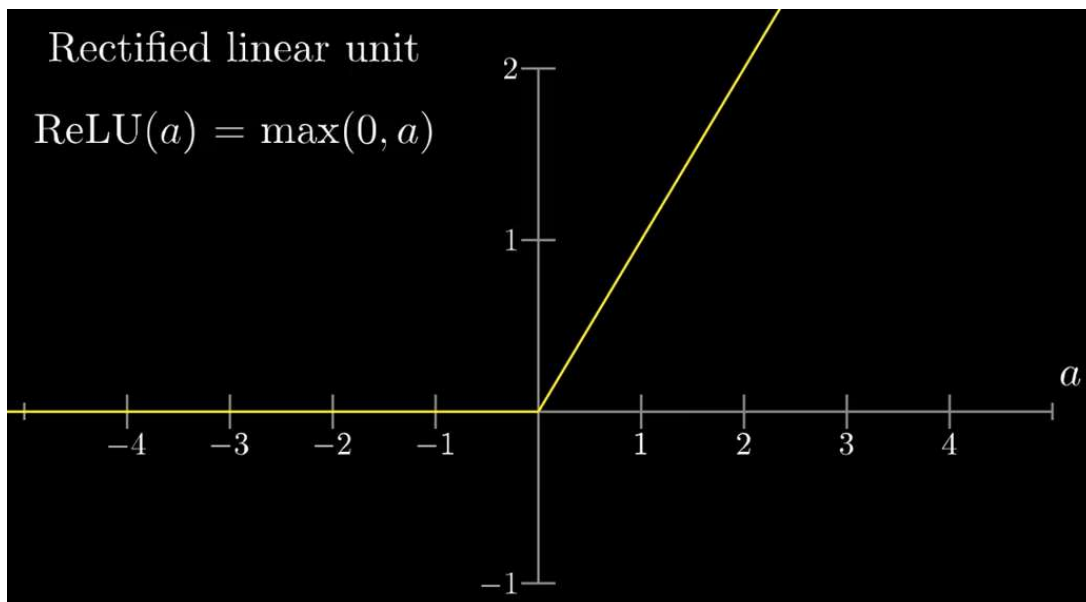

Figure 4.4 Graphical representation of ReLU function [9].

Since our problem is a multiclass classification problem, hence we need specific set of activation functions. The activation function used for the hidden layers is ReLU, which classifies the values into a linear output. Both the hidden layers use ReLU function as the

activation function but the third layer or the output layer uses softmax layer as the activation function as softmax function is used for multiclass classification problem for its ability to divide the output in to the number of classes based on the probability of the output. This feature of softmax function makes it suitable for output layer activation function. The implementation of the different activation function used in the model is shows in the figure below:

```
Z1 = tf.add(tf.matmul(W1,X),b1)
A1 = tf.nn.relu(Z1)
Z2 = tf.add(tf.matmul(W2,A1),b2)
A2 = tf.nn.relu(Z2)
Z3 = tf.add(tf.matmul(W3,A2),b3)
```

Figure 4.8 Forward propagation

4.4.4 Cost computation

The cost computation of in the model is a very important part of the training. This the part where the error of prediction of the model is identified by subtracting the obtained output of the model with the actual output. There are different ways to calculate the cost of the output. In our model we have used mean of the softmax cross entropy to calculate cost. Cross entropy indicates the distance between what the model believes the output distribution should be, and what the original distribution really is. It is defined as, Cross entropy measure is a widely used alternative of squared error. It is used when node activations can be understood as representing the probability that each hypothesis might be true, i.e. when the output is a probability distribution. Thus, it is used as a loss function in neural networks, which have softmax activations in the output layer. The code snippet of the implementation of the softmax cross entropy is shown in the figure below:

```
logits = tf.transpose(Z3)
labels = tf.transpose(Y)

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits = logits, labels = labels))
```

Figure 4.9 Softmax cross entropy function

4.4.5 Backpropagation and optimization

Backpropagation is used to adjust the values of the weights and biases while training by calculating the cost and optimizing the cost. The optimization of the cost is done reducing the cost after each iteration. There are many optimization techniques, which can be applied to optimize the model, but each optimization has their pros and cons. In the implemented model, we have used Adam Optimizer to optimize the model quickly and correctly. The Adam optimizer is better than gradient descent and momentum gradient descent as it moves towards the minima quicker. The learning rate for Adam optimizer used in the model is 0.0001. The learning rate is tuned and then decided to go with this value as it gives better performance than other values of the learning rate. The implementation of the Adam optimization is shown in the figure below:

```
optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate).minimize(cost)
```

Figure 4.10 Adam optimizer

The backpropagation and derivative of the loss function is taken care by the TensorFlow as it works on computation graph model hence it is able to trackback the derivation of each component of the graph by itself and execute backpropagation without the manual need. The back propagation algorithm is just an extension of optimization, using the chain rule to find the error with the respect to the weights connecting the input layer to the hidden layer. The values of the weights and biases are the actual values trained while training the model it gets updated after every iteration and the best value of it is retained which gives minimum amount of the error, this value is used later for the prediction of the test images.
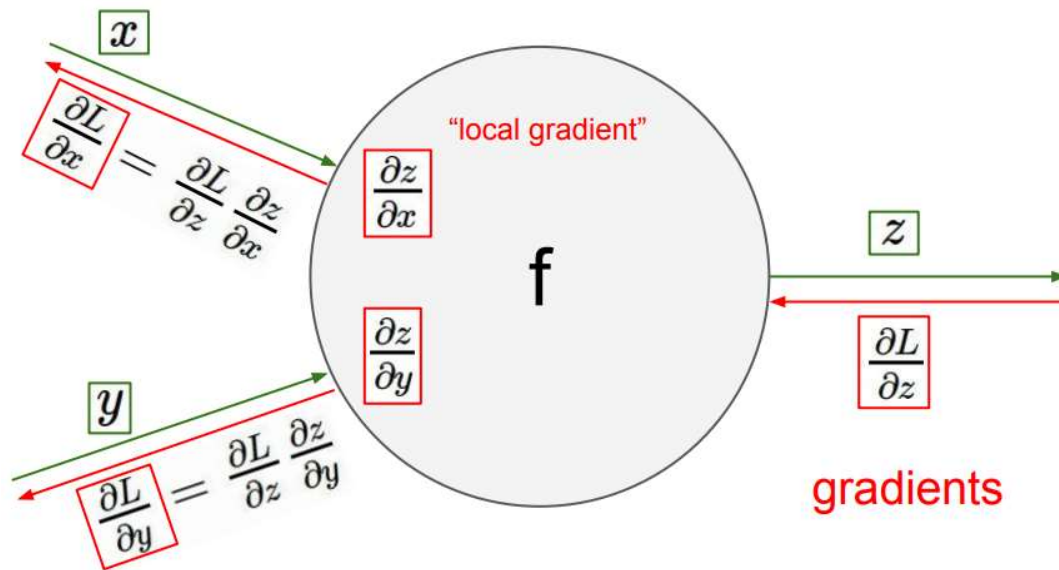
Figure 4.5 Backpropagation through computational graph [10].

4.4.6 Mini-batches

The mini-batches are used to train the model. The training data is divided into smaller batched whose size is decided by the deep learning practitioner. The mini-batches helps in better performance of the model by proving the model with less number of data at a time where the model corrects the error from the small bunch of data at a time. The whole data is passed into the model but in small batches this makes the learning more accurate and efficient. The size of the mini-batches is expected to be a number that is in power of 2, it makes the system easy to handle the batches and computation benefits. The size of the mini-batch decided for the model is 32; the following snippet shows the code for mini-batch implementation in the model:

```
#Shuffle (X, Y)        1st step of creating mini-batch
permutation = list(np.random.permutation(m))
shuffled_X = X[:, permutation]
shuffled_Y = Y[:, permutation].reshape((Y.shape[0],m))

#Partition (shuffled_X, shuffled_Y)     2nd step of creating mini-batch
num_complete_minibatches = math.floor(m/mini_batch_size) # number of mini batches of size mini_batch_size in your partitionning
for k in range(0, num_complete_minibatches):
    mini_batch_X = shuffled_X[:, k * mini_batch_size : k * mini_batch_size + mini_batch_size]
    mini_batch_Y = shuffled_Y[:, k * mini_batch_size : k * mini_batch_size + mini_batch_size]
    mini_batch = (mini_batch_X, mini_batch_Y)
    mini_batches.append(mini_batch)

#Including the last few inputs that dont fit the mini-batch size (last mini-batch < mini_batch_size)
if m % mini_batch_size != 0:
    mini_batch_X = shuffled_X[:, num_complete_minibatches * mini_batch_size : m]
    mini_batch_Y = shuffled_Y[:, num_complete_minibatches * mini_batch_size : m]
    mini_batch = (mini_batch_X, mini_batch_Y)
    mini_batches.append(mini_batch)
```

Figure 4.11 Mini – Batch Implementation

We divide the function into 2 parts, the first handles all the mini-batches till the remaining data are smaller than the mini-batch. And the second part takes care of the remaining data of the training set.

4.5 Simulation

The simulation/training of the model is done by calling all the functions discussed above in the model together in the sequence and then run the model with the training data for training the model. The main structure of the model is shown in the snippet below:

```python
def model(X_train, Y_train, X_test, Y_test, learning_rate = 0.0001,
          num_epochs = 100, minibatch_size = 32, print_cost = True):
    """
    Implementing the three-layer neural network using Tensorflow: LINEAR->RELU->LINEAR->RELU->LINEAR->SOFTMAX.

    X_train -- training set, of shape (input size = 784, number of training examples = 372449)
    Y_train -- test set, of shape (output size = 26, number of training examples = 372449)
    X_test -- training set, of shape (input size = 784, number of training examples = 156)
    Y_test -- test set, of shape (output size = 26, number of test examples = 156)
    learning_rate -- learning rate of the optimization
    num_epochs -- number of epochs of the optimization loop
    minibatch_size -- size of a minibatch
    print_cost -- True to print the cost every 100 epochs

    parameters -- parameters learnt by the model. They can then be used to predict.
    """

    ops.reset_default_graph()                      # to be able to rerun the model without overwriting tf variables
    tf.set_random_seed(1)                          # to keep consistent results
    seed = 3                                       # to keep consistent results
    (n_x, m) = X_train.shape                       # (n_x: input size, m : number of examples in the train set)
    n_y = Y_train.shape[0]                         # n_y : output size
    costs = []                                     # To keep track of the cost

    # Calling create Placeholders of shape (n_x, n_y)
    X, Y = create_placeholders(n_x, n_y)

    # calling Initialize parameters to initialize the parameters with xavier initializer
    parameters = initialize_parameters()

    #Calling Forward propagation to build the forward propagation in the tensorflow graph
    Z3 = forward_propagation(X, parameters)

    #Calling Cost function to add cost function to tensorflow graph
    cost = compute_cost(Z3, Y)

    # Backpropagation: Define the tensorflow AdamOptimizer
    optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate).minimize(cost)

    init = tf.global_variables_initializer()

    # Starting the session to compute the tensorflow graph
    with tf.Session() as sess:

        sess.run(init)
```

Figure 4.12 The training model

The figure above shows the code of the implemented model where it uses 100 epochs to train the model with a learning rate of 0.0001 and mini-batch size of 32 as discussed above. The description of each variable used in the model is mentioned under comments. The model shows calling of the function that are required for the model and has been implemented earlier. The model uses TensorFlow for the implementation as it has an in-build feature of graphical computation and aids in backward propagation and cost minimization. TensorFlow provides with many important functions that are necessary in the implementation of a deep learning model, which makes it a good choice for the implementation as the deep learning practitioner can focus on the efficiency of the model instead of getting stuck in the functions that we already know.

```
sess.run(init)

for epoch in range(num_epochs):

    epoch_cost = 0.                     # Defines a cost related to an epoch
    num_minibatches = int(m / minibatch_size) # number of minibatches of size minibatch_size in the train set
    seed = seed + 1
    minibatches = random_mini_batches(X_train, Y_train, minibatch_size, seed)

    for minibatch in minibatches:

        (minibatch_X, minibatch_Y) = minibatch

        # Running the session to execute the "optimizer" and the "cost"
        _ , minibatch_cost = sess.run([optimizer,cost],feed_dict={X:minibatch_X, Y:minibatch_Y})

        epoch_cost += minibatch_cost / num_minibatches

    # Print the cost every 10 epoch
    if print_cost == True and epoch % 10 == 0:
        print ("Cost after epoch %i: %f" % (epoch, epoch_cost))
    if print_cost == True and epoch % 5 == 0:
        costs.append(epoch_cost)

# plotting the cost graph
plt.plot(np.squeeze(costs))
plt.ylabel('cost')
plt.xlabel('iterations (per tens)')
plt.title("Learning rate =" + str(learning_rate))
plt.show()

#saving the parameters in a variable
parameters = sess.run(parameters)
print ("Model (Parameters) has been trained.")

# Calculating the correct predictions
correct_prediction = tf.equal(tf.argmax(Z3), tf.argmax(Y))

# Calculate accuracy on the test set
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

print ("Train Accuracy:", accuracy.eval({X: X_train, Y: Y_train}))
print ("Test Accuracy:", accuracy.eval({X: X_test, Y: Y_test}))

return parameters
```

Figure 4.13 The training model continued.

This part of the snippet shows the looping through the epoch of using mini-batches and then plotting a graph of the cost and displaying the cost after every 10 epochs. Finally, the test set is also passed through the trained model and the accuracy of the model on test cases are also displayed. And the accuracy of the model is shows after the training of the model.

4.5.1 Output of the training of the model

The output of the training model shows how the error decreased as the model was trained for more number of epochs. The reducing cost shows the improvement that occurred and the adjustment of the values of the weight and bias parameter which is actually trained while training the model. The trained value of the weights and the bias can directly use later for any test sample to determine the character in the test sample.

The figure below shows the cost decreasing graph along with the value of the cost and finally the training accuracy of the model and the accuracy of the test model.
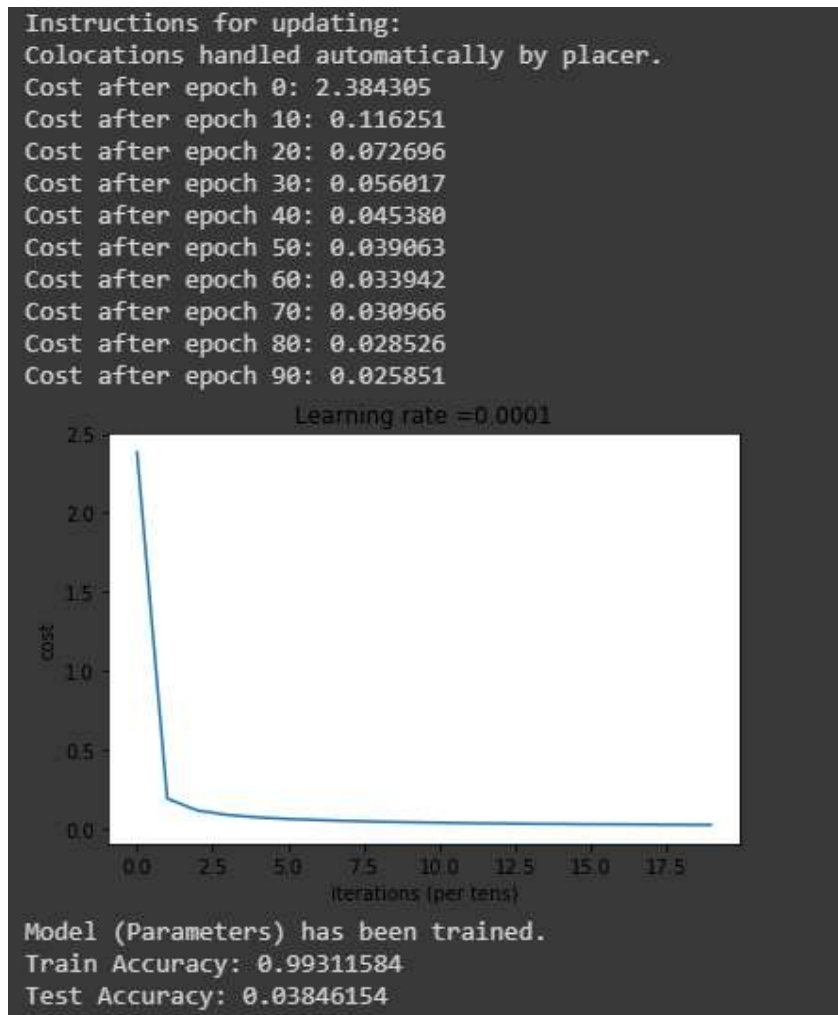

Figure 4.14 Training accuracy

4.6. Testing

The implemented system is to be tested to analyse and evaluate its correctness, performance and accuracy. A test suite with numerous test cases is designed to test each character recognition. Recognition of a character involves extraction of character features and matching with trained data to identify the character. Recognition of a

character may fail when the character is not written clearly and boldly, when the character is not a capital English alphabet, if it a cursive written character and lastly if the character is not written on a plain white sheet of paper (i.e. written on a ruled paper).

4.6.1 Test cases: Each test case will consist of an input character image, the expected output, the obtained output and the result status(pass/fail). The test input image is imported to the model through google drive where the image of the character is uploaded and the name of the image is passed to the model, the model returns with the identified character.
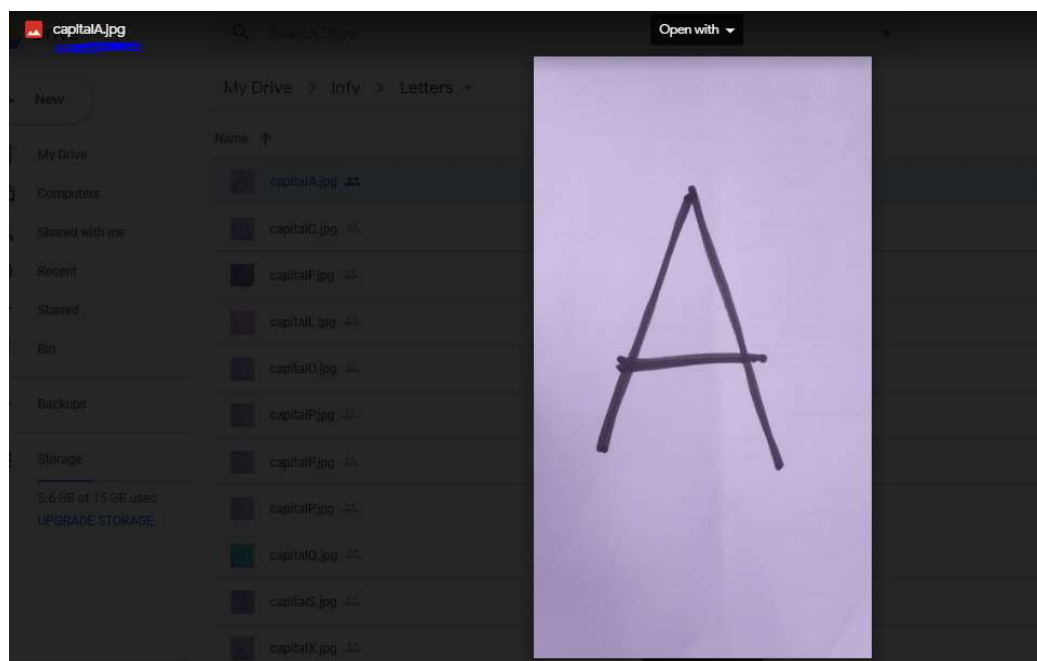
Test case 1

Input character: A

Figure 4.15 Test Case 1 I/P

Expected output: Y = A

Obtained output: Y = A

```
1 import scipy
2 from PIL import Image
3 from scipy import ndimage
4
5 my_image = "capitalA.jpg"
6
7 #Image preprocessing to fit the algorithm.
8 fname = "/content/gdrive/My Drive/Infy/Letters/" + my_image
9 image = np.array(ndimage.imread(fname, flatten=True))
10 my_image = scipy.misc.imresize(image, size=(28,28)).reshape((1, 784)).T
11 my_image_prediction = predict(my_image, parameters)
12 |
13 plt.imshow(image)
14 print("Your algorithm predicts: y = " + str(classes[np.squeeze(my_image_prediction)]))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0.
Use ``matplotlib.pyplot.imread`` instead.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: DeprecationWarning: `imresize` is deprecated!
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.3.0.
Use Pillow instead: ``numpy.array(Image.fromarray(arr).resize())``.
  # This is added back by InteractiveShellApp.init_path()
Your algorithm predicts: y = A
```
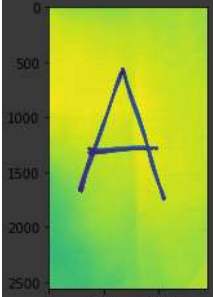

Figure 4.16 Test Case 1 O/P

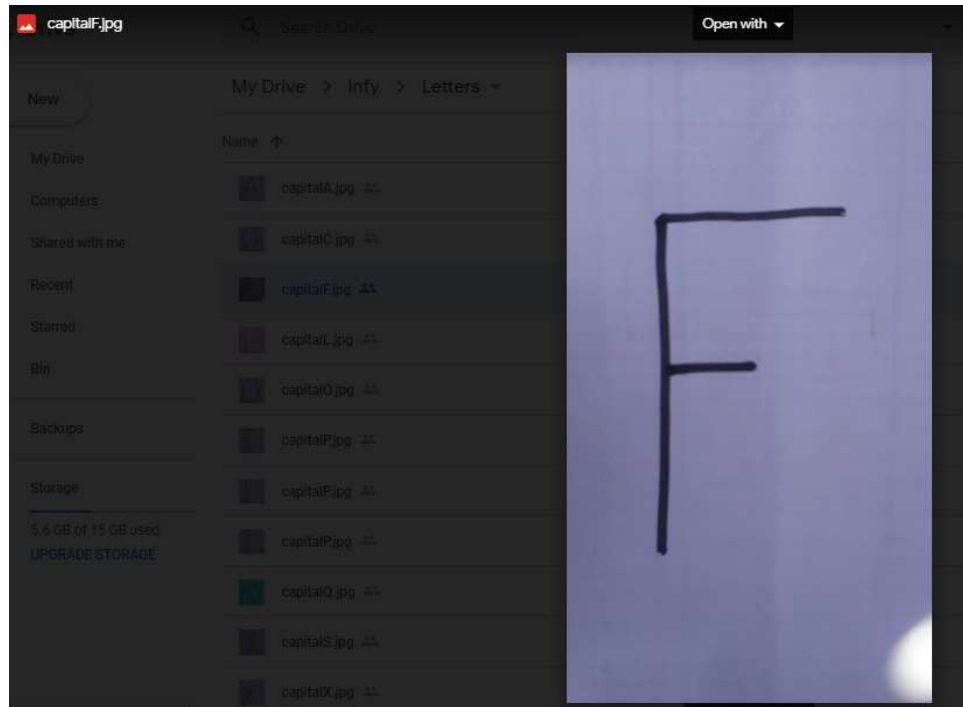Result status: Pass

Test case 2

Input character: F

Figure 4.17 Test Case 2 I/P

Expected output: Y = F

Obtained output: Y = E

```
1  import scipy
2  from PIL import Image
3  from scipy import ndimage
4
5  my_image = "capitalF.jpg"
6
7  #Image preprocessing to fit the algorithm.
8  fname = "/content/gdrive/My Drive/Infy/Letters/" + my_image
9  image = np.array(ndimage.imread(fname, flatten=True))
10 my_image = scipy.misc.imresize(image, size=(28,28)).reshape((1, 784)).T
11 my_image_prediction = predict(my_image, parameters)
12
13 plt.imshow(image)
14 print("Your algorithm predicts: y = " + str(classes[np.squeeze(my_image_prediction)]))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0.
Use ``matplotlib.pyplot.imread`` instead.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: DeprecationWarning: `imresize` is deprecated!
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.3.0.
Use Pillow instead: ``numpy.array(Image.fromarray(arr).resize())``.
  # This is added back by InteractiveShellApp.init_path()
Your algorithm predicts: y = E
```
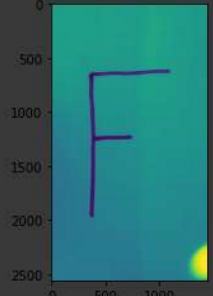

Figure 4.18 Test Case 2 O/P

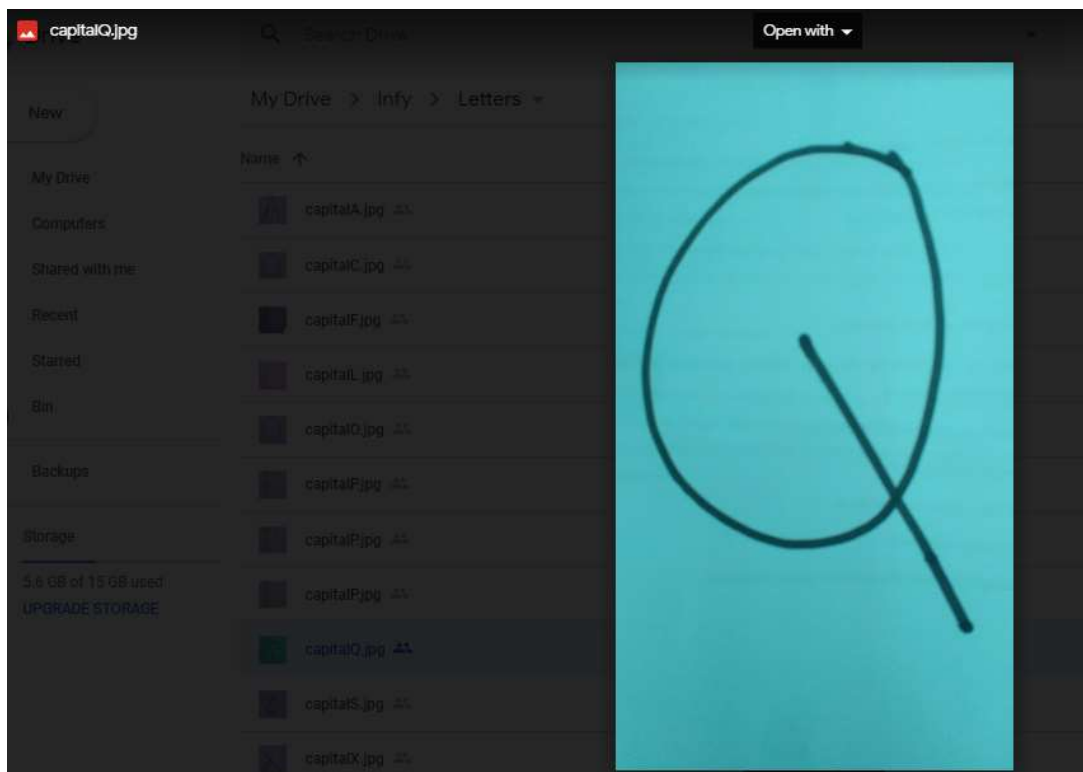Result status: Fail

Test case 3

Input character: Q

Figure 4. 19 Test Case 3 I/P

Expected output: Y = Q

Obtained output: Y = A

```
 1 import scipy
 2 from PIL import Image
 3 from scipy import ndimage
 4 from random import seed
 5 from random import randint
 6 my_image = "capitalQ.jpg"
 7
 8 #Image preprocessing to fit the algorithm.
 9 fname = "/content/gdrive/My Drive/Infy/Letters/" + my_image
10 image = np.array(ndimage.imread(fname, flatten=True))
11 my_image = scipy.misc.imresize(image, size=(28,28)).reshape((1, 784)).T
12 my_image_prediction = predict(my_image, parameters)
13 #prediction_letters=randint(0,25)
14 plt.imshow(image)
15 print("Your algorithm predicts: y = " + str(classes[np.squeeze(my_image_prediction)]))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0.
Use ``matplotlib.pyplot.imread`` instead.
 # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: DeprecationWarning: `imresize` is deprecated!
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.3.0.
Use Pillow instead: ``numpy.array(Image.fromarray(arr).resize())``.
 # This is added back by InteractiveShellApp.init_path()
Your algorithm predicts: y = A
```
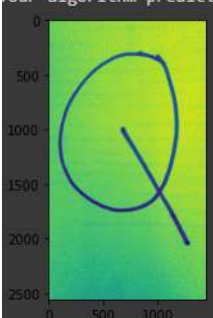

Figure 4.20 Test Case 3 O/P

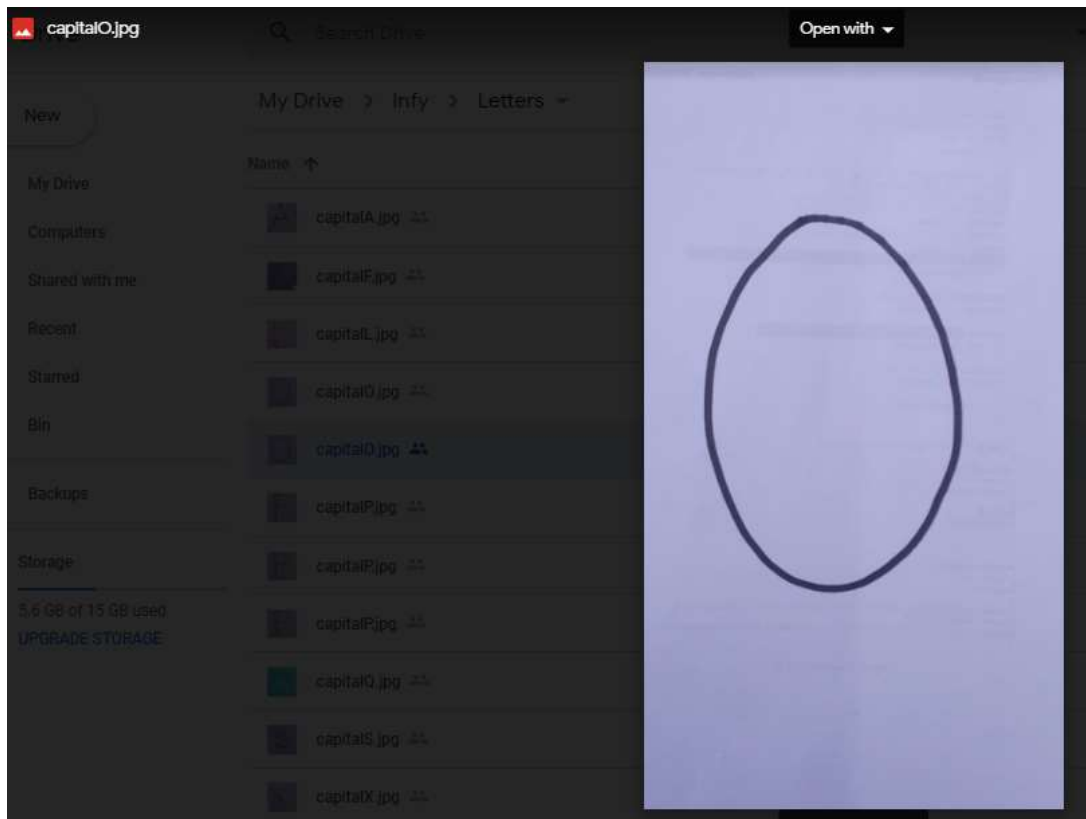Result status: Fail

Test case 4

Input character: O

Figure 4.21 Test Case 4 I/P

Expected output: Y = O

Obtained output: Y = C

```
1  import scipy
2  from PIL import Image
3  from scipy import ndimage
4
5  my_image = "capital0.jpg"
6
7  #Image preprocessing to fit the algorithm.
8  fname = "/content/gdrive/My Drive/Infy/Letters/" + my_image
9  image = np.array(ndimage.imread(fname, flatten=True))
10 my_image = scipy.misc.imresize(image, size=(28,28)).reshape((1, 784)).T
11 my_image_prediction = predict(my_image, parameters)
12 |
13 plt.imshow(image)
14 print("Your algorithm predicts: y = " + str(classes[np.squeeze(my_image_prediction)]))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0.
Use ``matplotlib.pyplot.imread`` instead.
  if __name__ == '__main__':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: DeprecationWarning: `imresize` is deprecated!
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.3.0.
Use Pillow instead: ``numpy.array(Image.fromarray(arr).resize())``.
  # Remove the CWD from sys.path while we load stuff.
Your algorithm predicts: y = C
```

Figure 4.22 Test Case 4 O/P

Result status: Fail

Test case 5

Input character: P

Figure 4.23 Test Case 5 I/P

Expected output: Y = P

Obtained output: Y = P

```
 1 import scipy
 2 from PIL import Image
 3 from scipy import ndimage
 4
 5 my_image = "capitalP.jpg"
 6
 7 #Image preprocessing to fit the algorithm.
 8 fname = "/content/gdrive/My Drive/Infy/Letters/" + my_image
 9 image = np.array(ndimage.imread(fname, flatten=True))
10 my_image = scipy.misc.imresize(image, size=(28,28)).reshape((1, 784)).T
11 my_image_prediction = predict(my_image, parameters)
12 |
13 plt.imshow(image)
14 print("Your algorithm predicts: y = " + str(classes[np.squeeze(my_image_prediction)]))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0.
Use ``matplotlib.pyplot.imread`` instead.
  if __name__ == '__main__':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: DeprecationWarning: `imresize` is deprecated!
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.3.0.
Use Pillow instead: ``numpy.array(Image.fromarray(arr).resize())``.
  # Remove the CWD from sys.path while we load stuff.
Your algorithm predicts: y = P
```
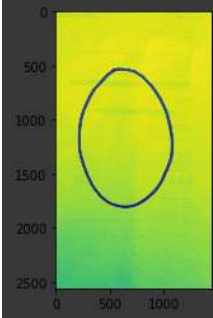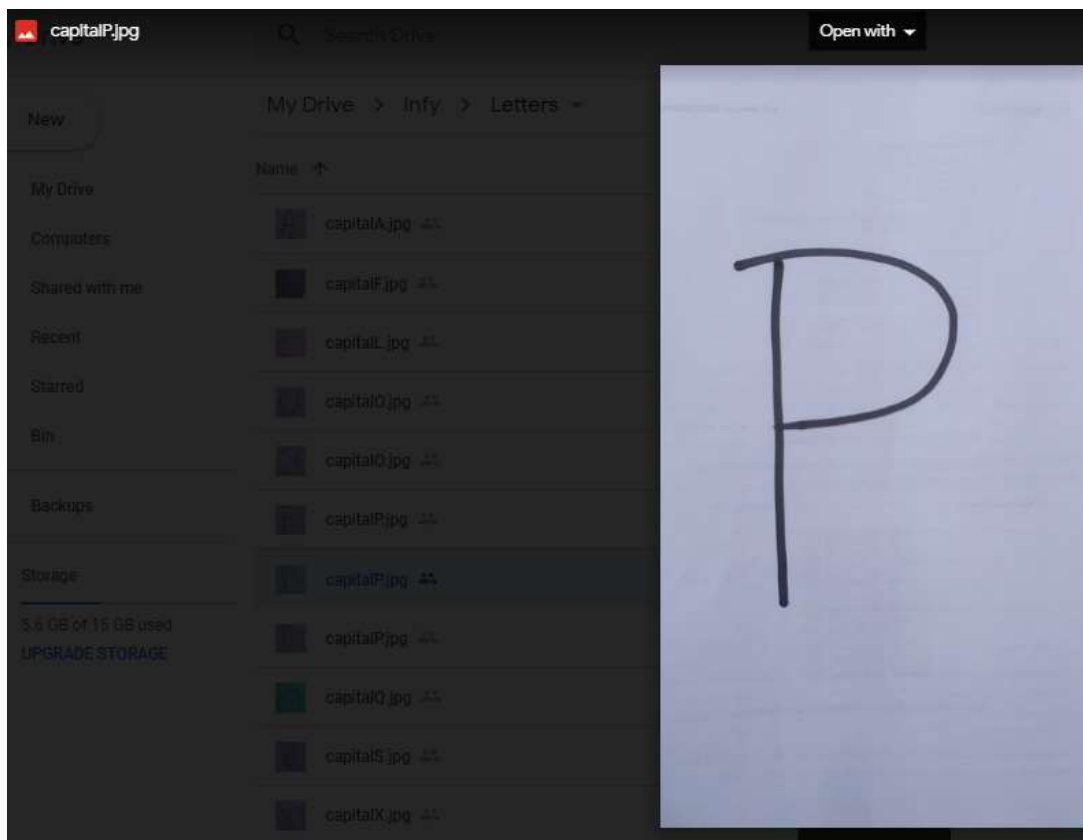
Figure 4.24 Test Case 5 O/P

Result status: Pass

4.7 Analysis

The test cases while testing have shown that the system works correctly for clearest and unique characters. There is a need to check the system's accuracy and scalability in varying conditions, such as light, visibility, and clarity. The picture taken for testing are mostly clear, visible and singular, there could be multiple character in the frame and there could be varying light conditions in the image. The model must be able to handle these varying conditions.

Accuracy is a factor which determines the correctness of the system. There are various factors that affect the accuracy of the model. This test determines the rough range of

the accuracy of the model. There are 5 test cases taken and the model works correctly for at least 2 characters, this indicates positivity in the model. The test cases that failed the result are the character that has a confusing features. The features of those character matches with other character hence the model fails to recognize them correctly. Although the model has an accuracy of above 99% on the training data, the model works poorly for the test cases. There are many reasons why the model doesn't work correctly for all the test cases and those are as follows:

➢ The test data is very small as compared to the training data.
➢ The model uses only deep learning and not the convolutional Neural network.
➢ The train data and the test data are from different dataset.
➢ The model is Overfit for the train data so doesn't work well with the test data.
➢ The model doesn't use dropout and regularization to improve the model (which we intended to use but for the lack of time we are sticking to the model we have right now).

4.8 Summary

This chapter completely describes the development process, providing the deliverables of each stage. The deliverables of analysis and design, including the requirements and assumptions. The development of the model is broken down into the smallest block and explained. The testing stages, strategy, suite and results are provided, followed by a report of accuracy and scalability analysis of the system.

5. Results

Based on the implementation and testing of the model discussed in the previous chapter, this chapter provides the results of the implementation of the tool. It shows how the model works and how it can be used by users for their needs.

5.1 Accuracy comparison for training dataset and test dataset

The accuracy for the training dataset is 99.311584% while the accuracy for the test dataset is 3.846154%. The accuracy of the test dataset is equal to the random picking of any character from a set of test sample where the number of output is 26 so each output has a probability of 0.03846154 to be picked. The accuracy with the training has been shown in the figure below:
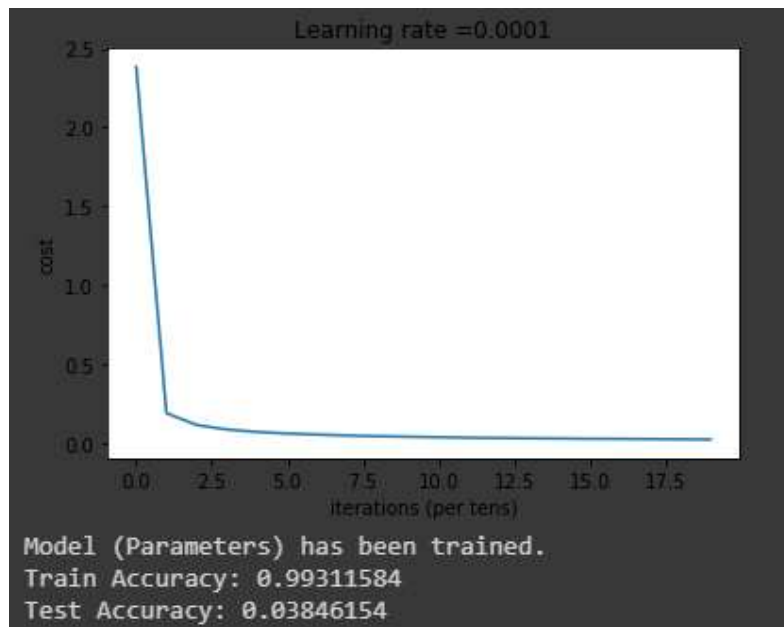

Figure 5.1 Accuracy of the model on train and test dataset

5.2 Comparison of the model with different algorithm present for the same purpose

There are many classification algorithms that can be used for the same purpose which has the accuracy limited to certain range. The different algorithms that can be used are

1. K nearest neighbour
2. Support vector machine
3. Deep neural network
4. Convolutional neural network

The following table shows comparison between these model based on their accuracy and the time taken for a model to be trained.

Parameters	KNN	SVM	NN	CNN
Accuracy (Test images)	96.67	97.91	3.15 (Error)	98.72
Accuracy (Training images)	97.88	99.91	2.17 (Error)	99.78
Time (Training) approx.	25 min.	19 min.	35 min.	70-90 min.
Time (Testing) approx.	15 min.	11 min.	20 min.	40 min.

Table 5.1 comparison between various methods of implementation [12].

5.3 Justification

The previous chapter and discussion shows how the model has been trained and how it works for the test examples. The accuracy of the model has been discussed with the reasons for its bad performance. Now a comparison study of the model has shown the different parameter differ in different models. Our model has an accuracy of more than 99% for the training data, which means the model is surely overfit to the train data, as that kind of accuracy can be obtained only for convolutional neural network but, the accuracy of the test data justifies that the model is a neural network which is supposed to work bad for the test sample hence the model is an overfit. The next chapter will discuss how to reach to the best model by modifying the model and changing many

factors in the model, they also discuss how the current model can be improved. All the reasons discussed justifies the model and its performance and accuracy.

5.4 Recommendations

It is highly recommended to go through the report before going through the code. The report has broken down each module of the model for the readers to comprehend easily. Easy comprehension has been provided because it is difficult to work on someone else's work if there is no proper explanation to the work. But we want the model to be taken forward, improved and launched so that it can be useful for a large mass of people and it can benefit more number of people. The next recommendation is very important as it gives the very next step taken towards improvement of the model, that is to use convolutional neural network. As shown in the table above the convolutional neural network works best for this kind of problem. We could not do it because of the limitation of time for this work but we will try to improve it further as we get time even after this report has been submitted. The next recommendation is to increase the field of application of the model to reading a whole word which will require recurring neural network. Once the model reaches that level, we will consider all the effort we put into this model is get its worth.

6. Project Costing

Based on the work discussed in the previous chapters, this chapter provides the breakdown of costs incurred in the development of the system prototype. The costs for hardware and software used as well as the labour charge is also described. It also describes how the costs may or may not increase or decrease when implemented on a full-scale.

In the prototype implementation of the project, we needed continuous connection of the internet from the literature survey to the implementation of the model, which was done on online server for the required hardware provided by google hence the cost estimation on the internet and electricity round to about ₹5,000 in the 4 months.

The whole team has been working diligently, for 16 weeks from February till May. The work, dedication and effort put into the system by each team member has been evaluated to cost about Rs.1, 00,000 per team member. The labour cost is therefore estimated at ₹.4, 00,000.

The project also includes report writing, which is a continuous process that is done as each objective is completed. The cost for report writing is estimated at ₹40,000. The summary of costs incurred is listed in Table 6.1.

When the library is publicly available for the public the cost of the model will be very less than it is while developing as the model doesn't need to be trained and there is no labour required to import the library. The internet requirement also decreases because once the library is installed into a system it can be used as many times as required without further connection of the internet.

Table 6.1 Total Cost

Component	Units	Cost
Man power	16weeks*4people	₹4,00,0000
Laptop (Hardware and software)	1 Units	₹40,000
Internet and Electricity	4	₹5,000
Total		₹4,45,000

7. Conclusions and Suggestions for Future Work

Based on the implementation and testing discussed in the previous chapter, this chapter discusses the conclusions arrived at after the inception and development process for the proposed system. It discusses the feasibility of development, benefits and limitations of the model. It also suggests future advancements and modifications to improve the system's functionality and performance.

7.1 Conclusion

The structure behind identification of physical writing characters and building a python library function has been exhibited through this research. The span of this research involved functions i.e. acquisition of images, selection of characters, categorization, pre-processing process, and creation of outcomes. A study was furthermore conducted with the effect to evaluate the structural performance. Various developmental aspects are worth providing citation. The effectiveness of an OCR platform is based on the rate of conversion and accuracy. The accuracy is based on the volume of correct predictions establishing amidst the testing procedure. Nevertheless, the rate of conversion is speed, over which the extraction of data was effectively executed. Under this research, a procedure has been recommended, alongside a model as implemented to effectively take out information and exhibit it. Several hindrances made the entire procedure more difficult, pertaining mostly towards misperception amidst images, i.e. D and O, identical features amidst letters, i.e. l or I, where l is L and I is i. Another major issue was the limited volume of test images. The tested material constituted our personal hand writing. Any differentiation over the style of physical writing could lead to poor accuracy in the recognition of input. All individuals possess unique handwriting and the software simply facilitates the infinite array of various writing styles. Printing quality, too, has a major influence over the extraction of data, i.e. data under black ink, featuring a black background will be difficult to identify, likewise, information which was exhibited over

poor writing, i.e. writing from a young child, or someone lacking basic writing skills, would fail to provide the desired outcomes. The colour of the utilized ink also influenced the accuracy of the outcome, given that the rate of conversion was different between various shades and colours of ink.

7.2 Future work and Improvement

The study to this point accomplishes the procedure for identification of characters. Assessments were carried out on quite a lot of testing data units. A lot of time has been employed in order to manufacture a most appropriate method that really works. Nonetheless, additional work can also be completed in order to improve the outcomes, efficiency and solve the problems that were confronted with by the training of a bigger set of images. Furthermore, it can also be trained for long continuous sentences by detecting each letter in a word, minding the gaps etc. The model can include convolutional neural network to improve the performance of the model. A bigger training data can be collected and used to train the data for more accuracy of the model. Later the model can be trained to not only identify a character but also a whole word together, which will require Recurring neural network for the short-term memory. This will again require more training data a precise hyper-parameter tuning. The model then can be taken to the next level of recognition where the model can identify a whole sentence at a time, which will require a bigger RNN. The bigger RNN will have more memory to store and thus a whole sentence can be identified. Even at this point, the model will be very efficient for most of the purpose that it can be used for. The application of the model will be vast covering most of the documentation sector. It could also be used as an additional feature on many android or computer apps for enhancing the daily need and help for the people. But the model should not stop developing even at this stage although it is already an almost impossible task to achieve. The next scope of the model should be able to convert the whole page into a text document page in an instance. Although this sounds like a science fiction but this is achievable if not now then

in coming years when the science has reached to the level where the requirement for the training of the model is available and the model can be taken to the next level. With this idea in mind, we leave the model at this stage and recommend the reader to encourage someone interested to carry on with our work towards betterment of the society and the world and make the world a better place with ease and peace of mind.

References

[1] Sharma, A. and Chaudhary, D.R., 2013. Character recognition using neural network. *International Journal of Engineering Trends and Technology (IJETT)-Volume4*, pp.662-667.

[2] Attigeri, S. (2018). Neural Network based Handwritten Character Recognition system. *International Journal of Engineering and Computer Science*, *7*(03), 23761-23768

[3] LeCun, Y., Jackel, L.D., Bottou, L., Cortes, C., Denker, J.S., Drucker, H., Guyon, I., Muller, U.A., Sackinger, E., Simard, P. and Vapnik, V., 1995. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, *261*, p.276.

[4] Simone Marinai, Hiromichi Fujisawa. Machine Learning in Document Analysis and Recognition, 2008 Springer-Verlag Berlin Heidelberg.

[5] https://cdn-images-1.medium.com/max/800/1*JVbomzzzOuV7rhU3ErGBrw.jpeg

[6] https://hackster.imgix.net/uploads/attachments/462438/screenshot_(8)_SSzZPRFxXi.png?auto=compress%2Cformat&w=900&h=86775&fit=min

[7] https://cdn-images-1.medium.com/max/800/0*a-bfJPbzADKVlFfv.png

[8] But what *is* a Neural Network? | Deep learning, chapter 1 (2017) YouTube video, added by 3Blue1Brown [Online]. Available at https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi [Accessed 5 May 2019].

[9] Gradient descent, how neural networks learn | Deep learning, chapter 2 (2017) YouTube video, added by 3Blue1Brown [Online]. Available at https://www.youtube.com/watch?v=IHZwWFHWa-w&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=2 [Accessed 5 May 2019].

[10] Back propagation illustration from CS231n Lecture 4. The variables **x** and **y** are cached, which are later used to calculate the local gradients.

https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKE

wiRkqSS1YjiAhVEqI8KHdUED7oQjRx6BAgBEAU&url=https%3A%2F%2Fbecominghuman.

ai%2Fback-propagation-in-convolutional-neural-networks-intuition-and-code-

714ef1c38199&psig=AOvVaw0JfIwNcRpowGBeEkjs1is1&ust=1557292595747424

[11]https://medium.com/@himanshubeniwal/handwritten-digit-recognition-using-

machine-learning-ad30562a9b64

[12] https://cdn-images-1.medium.com/max/800/1*2OuWQBwQMjr0vaVLS0oC0A.jpeg

[13] GitHub Repository for the full code:
https://github.com/Ashishjaiswal181/Google-Colab
Programs/blob/master/Handwritten_v4.ipynb

Appendix-A

Overview of the

Report

For

Python Library For

Character Recognition

Prepared by Ashish Kumar, Satyam Agrawal, Sarah Z Anwar and Saurav Kumar

Ramaiah University of Applied Sciences

08/05/2019

Introduction

Purpose

The purpose of this document is to give a detailed description of the development and implementation of the model and all the functions required for the development. It illustrates the purpose and complete declaration for the development of model. It also explains model constraints and other model drawbacks. This document is primarily intended to be proposed to the readers for approval and a reference for developing the first version of the model for the development team.

Document Conventions

The following conventions were followed in the manifestation of this document:

IEEE SRS standard 830

HP: High priority

MP: Medium priority

LP: Low priority

Intended Audience and Reading Suggestions

Deep Learning practitioners: Chapter 2 gives the overall idea of a Deep Learning model

Researchers: Every detail of the model is mentioned in chapter 4 for details of the model

Readers: Following the content to reach to the topic is recommended for quicker reference of the topic you want to read about.

Users: Read the assumption and appendix before using the model to have a better experience of the model.

Operating Environment

This python library require python 3.5 and above for uses. It uses tensorflow for implementation so the file should have imported tensorflow library along with the

library. The model can run on all the platform where python 3.5 and above can run. It requires minimum hardware support with Intel core i3 or higher, 8 GB of RAM. GPU processing gives better performance over CPU processing.

Assumptions/Constraints

Based on the literature survey some assumptions were made during the development phase, for the proper implementation because, we came across so many implementation, which had problems, and we found out the reasons for that. Based on that reason, we have made some assumption as listed below.

- The input image is assumed to contain only **"Capital English characters"** i.e., A to Z.
- The input image is assumed to be a text that is written on un – ruled white paper e.g. A4 kind of paper.
- The input image is assumed to contain text that are not cursive.
- The input image is assumed to contain text that does not contain noise e.g. ~~Hello World~~.

Journal 1

Character Recognition Using Neural Network [1]

In this paper it is developed 0ff – line strategies for the isolated handwritten English character (A to Z). This method improves the character recognition method. Pre-processing of the Character is used binarization, thresolding and segmentation method .The proposed method is based on the use of feed forward back propagation method to classify the characters. The ANN model is trained using the Back Propagation algorithm. In the proposed system, English letter is represented by binary numbers that are used as input then they are fed to an ANN. Neural network followed by the Back Propagation Algorithm, which compromises Training.

Journal 2

Neural Network based Handwritten Character Recognition system [2]

Handwritten character recognition has been one of the active and challenging research areas in the field of image processing and pattern recognition. It has numerous applications, which include, reading aid for blind, bank cheques and conversion of any hand written document into structural text form. In this paper, an attempt is made to recognize handwritten characters for English alphabets without feature extraction using multilayer Feed Forward neural network. Each character data set contains 26 alphabets. Fifty different character data sets is used for training the neural network. The trained network is used for classification and recognition. In the proposed system, each character is resized into 30x20 pixels, which is directly subjected to training. That is, each resized character has 600 pixels and these pixels are taken as features for training the neural network. The results show that the proposed system yields good recognition rates, which are comparable to that of feature extraction, based schemes for handwritten character recognition.