HANDWRITTEN DIGIT RECOGNITION

A project report submitted in the 3rd Academic Year

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

B. Hari Priya	(5191411008)
A. Hamsweetha	(5191411004)
E. Bala Subrahmanyam	(5191411018)

Under the Esteemed Guidance of

Mr. **I. MAHESH KUMAR SWAMY** **B. Tech, M. Tech**

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ENGINEERING AND TECHNOLOGY PROGRAM

GAYATRI VIDYA PARISHAD COLLEGE FOR DEGREE AND PG COURSES (A)

Rushikonda, Visakhapatnam - 45

(Approved by AICTE| Accredited by NBA| Accredited by NAAC| Affiliated to Andhra University)

An ISO 9001:2015 Certified Institution

2019-2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GAYATRI VIDYA PARISHAD COLLEGE FOR DEGREE AND PG COURSES (A)

Rushikonda, Visakhapatnam - 45

CERTIFICATE

This is to certify that the project report entitled "**HANDWRITTEN DIGIT RECOGNITION**" being submitted by B. Hari Priya (5191411008), A. Hamsweetha (5191411004), E. Bala Subrahmanyam (5191411018),in the partial fulfilment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Gayatri Vidya Parishad College for Degree and PG Courses, Visakhapatnam is a record of bonafied work carried out under my guidance and supervision.

Project Guide

I.Mahesh Kumar Swamy

Assistant Professor

B. Tech, M.Tech

Head of the Department

Dr. N.V.Ramana Murty

Professor

M. Tech, Ph. D

External Examiner

VISION AND MISSION OF THE INSTITUTE

VISION

Creating Human Excellence for a Better Society

MISSION

Unfold into a World class organization with strong academic and research base, producing responsible citizens to cater to the changing needs of the society.

VISION AND MISSION OF THE DEPARTMENT

VISION

"Create and sustain as a Centre of excellence in Computer Science and Engineering and allied areas of research to pave the way for providing better technocrats to the society."

MISSION

"To provide strong *conceptual base (M1)* for acquiring programming skills, to develop high *computational innovative skills (M2)* and to empower *leadership qualities* with strong *ethical base (M3)"*

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Ability to develop hardware and software problem solving capabilities by inculcating strong fundamental, analytical, and communication skills.

PEO2: Ability to build confidence to address the challenges in their work and to apply innovative technical tools through interaction with professional bodies.

PEO3: Ability to gain sufficient depth of knowledge, Self-learning skills, technical skills and leadership qualities through participating in personality development activities, so as to sustain in multidisciplinary environment.

PEO4: To be able to respond to societal needs in professional and ethical concerns.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: System Inception and Elaboration: Conceptualize the software and/or hardware systems, system components and processes/procedures through requirement analysis, Modelling/Design of the system using various architectural/design patterns, Standard Notations, procedures and algorithms.

PSO 2: System Construction: Implement the systems, Procedures and Processes using the state of the art technologies, standards, tools and Programming Paradigms.

PSO 3: System Testing and Deployment: Verify and Validate the Systems, Procedures and Processes using various testing and verification techniques and tools.

PSO 4: Quality and Maintenance: Manage the quality through various product development strategies under revision, transition and operation through maintainability, flexibility, testability, portability, reusability, interoperability, correctness, reliability, efficiency, integrity and usability to adapt the system to the changing structure and behavior of the systems/environments.

DECLARATION

We hereby declare that the project entitled **"HANDWRITTEN DIGIT RECOGNITION"** submitted in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering, to Engineering and Technology Program, Gayatri Vidya Parishad College for Degree & PG Courses (A). We assure that this project is not submitted in any other University or College.

Name & Signature of the Students

B. Hari Priya (5191411008)
A. Hamsweetha (5191411004)
E. Bala Subrahmanyam (5191411018)

ACKNOWLEDGEMENTS

With great pleasure we want to take this opportunity to express our heartfelt gratitude to all the people who helped in making main project work a grand success.

First of all we express our deep sense of gratitude to Sri. Guide Name, Assistant Professor, for his constant guidance throughout our main project work.

We are grateful to Project Co-ordinator Name, Assistant Professor, for his valuable suggestions and guidance given by him during the execution of this main project.

We would like to thank Dr. N.V.Ramana Murty, Professor, Head of the Department of Computer Science and Engineering, for being moral support throughout the period of our study.

We are highly indebted to Dr. B.V.Ramana Murty, Director and Prof. S Rajani, Principal for giving us the permission to carry out this main project in the college.

We would like to thank all the Teaching and Non-Teaching staff of CSE Department for sharing their knowledge with us.

B. Hari Priya (5191411008)

A. Hamsweetha (5191411004)

E. Bala Subrahmanyam (5191411018)

ABSTRACT

The reliance of humans over machines has never been so high such that from object classification in photographs to adding sound to silent movies everything can be performed with the help of deep learning and machine learning algorithms. Likewise, Handwritten digit recognition is one of the significant areas of research and development with a streaming number of possibilities that could be attained.

Handwritten digit recognition is one of the practically important issues in pattern recognition applications. The applications of digit recognition include in postal mail sorting, bank check processing, form data entry, etc. The heart of the problem lies within the ability to develop an efficient algorithm that can recognize handwritten digits, and which is submitted by users by the way of a scanner, tablet, and other digital devices. This project presents an approach to off-line handwritten digit recognition based on different machine learning technique. The main objective of this project is to ensure effective and reliable approaches for recognition of handwritten digits.

As the manually written digits are not of a similar size, thickness, position, and direction, in this manner, various difficulties must be considered to determine the issue of handwritten digit recognition. The uniqueness and assortment in the composition styles of various individuals additionally influence the example and presence of the digits. It is the strategy for perceiving and arranging transcribed digits. The aim of this project is to implement a classification algorithm to recognize the handwritten digits. Deep Learning algorithms with help of CNN are utilized along with keras and tensorflow are implemented to create a handwritten digit recognizing model with an accuracy of 98.67%.

TABLE OF CONTENTS

CHAPTER	Page No
1. Introduction	1
1.1 Introduction	1
1.2 Applications	2
1.3 Introduction to Convolution Neural Networks (CNN)	2
1.4 Introduction to OpenCV	7
1.5 Libraries Used	10
2. Literature Survey	13
3. Requirements and Analysis	14
3.1 Existing System	14
3.2 Proposed System	14
3.3 Functional Requirements	15
3.4 Non-Functional Requirements	16
3.5 Analysis	17
4. System Design	20
4.1 Design Goals	20
4.2 System Architecture	21
4.3 System Design	21
4.4 UML Diagrams	22
4.5 Data Flow Diagrams	26
4.6 Flow Chart	28
5. Implementation	29
5.1 Coding Approach	29
5.2 Information Handling	30
5.3 Programming Style	30
5.4 Verification and Validation	30
5.5 Implementation of Project	31
6. Testing	32
6.1 Testing Activities	32
6.2 Testing Types	32

6.3 Test Cases 35
7. Results 39
7.1 Canvas 39
7.2 Drawing Digit and Prediction 40
8. Sample Code 41
8.1 Code for application 41
8.2 Code for training the Model 43
9. Conclusion 45
10. References 46

LIST OF TABLES

Title	Page No
6.3 Test cases	35

LIST OF FIGURES

Title	Page No
1.3.1 Convolution Neural Network Architecture	3
1.3.2.1 visual representation of convolutional layers	5
1.6.1 visual representation of convolutional layers	12
3.4.1.9 Python Language Logo	17
4.3.1 System's Block Diagram	21
4.4.1.1 Use Case Diagram	23
4.4.2.1 Class Diagram	24
4.4.3.1 Activity Diagram	25
4.4.4.1 Sequence Diagram	36
4.5.1.1 DFD Level 0	27
4.5.2.1 DFD Level 1	27
4.6.1 Flow chart of the system	28
6.3.1 digit 0 drawn on canvas	36
6.3.2 digit 1 drawn on canvas	36
6.3.3 digit 2 drawn on canvas	37
6.3.4 digit 3 drawn on canvas	37
6.3.5 digit 4 drawn on canvas	37
6.3.6 digit 5 drawn on canvas	37
6.3.7 digit 6 drawn on canvas	38
6.3.8 digit 7 drawn on canvas	38
6.3.9 digit 8 drawn on canvas	38
6.3.10 digit 9 drawn on canvas	38
7.1.1 Canvas	39
7.1.2 Digit drawing and Prediction	40

1. INTRODUCTION

1.1 Introduction

Recognition is identifying or distinguishing a thing or an individual from the past experiences or learning. Similarly, Digit Recognition is nothing but recognizing or identifying the digits in any document. Digit recognition framework is simply the working of a machine to prepare itself or interpret the digits. Handwritten Digit Recognition is the capacity of a computer to interpret the manually written digits from various sources like messages, bank cheques, papers, pictures, and so forth and in various situations for web-based handwriting recognition on PC tablets, identifying number plates of vehicles, handling bank cheques, digits entered in any forms etc. Machine Learning provides various methods through which human efforts can be reduced in recognizing the manually written digits. Deep Learning is a machine learning method that trains computers to do what easily falls into place for people: learning through examples. With the utilization of deep learning methods, human attempts can be diminished in perceiving, learning, recognizing and in a lot more regions. Using deep learning, the computer learns to carry out classification works from pictures or contents from any document. Deep Learning models can accomplish state-of-art accuracy, beyond the human level performance. The digit recognition model uses large datasets to recognize digits from distinctive sources. Handwriting recognition of characters has been around since the 1980s. The task of handwritten digit recognition, using a classifier, has extraordinary significance and use such as – online digit recognition on PC tablets, recognize zip codes on mail, processing bank check amounts, numeric sections in structures filled up by hand (for example - tax forms) and so on. There are diverse challenges faced while attempting to solve this problem. The handwritten digits are not always of the same size, thickness, or orientation and position relative to the margins. The main objective was to actualize a pattern characterization method to perceive the handwritten digits provided in the MINIST data set of images of handwritten digits (0-9).

1.2 Applications

Application of handwritten Character recognition: (a) National ID number recognition system (b) Postal office automation with code number recognition on Envelope (c) Automatic license plate recognition and (d) Bank automation.

1.3 Introduction to Convolutional Neural Networks (CNN)

Convolutional Neural Networks are very similar to ordinary Neural Networks. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. Convolutional Neural Networks (CNNs) are analogous to traditional ANNs in that they are comprised of neurons that self-optimise through learning. Each neuron will still receive an input and perform a operation (such as a scalar product followed by a non-linear function) - the basis of countless ANNs. From the input raw image vectors to the final output of the class score, the entire of the network will still express a single 26 perceptive score function (the weight). The last layer will contain loss functions associated with the classes, and all of the regular tips and tricks developed for traditional ANNs still apply. The only notable difference between CNNs and traditional ANNs is that CNNs are primarily used in the field of pattern recognition within images. This allows us to encode image-specific features into the architecture, making the network more suited for image-focused tasks - whilst further reducing the parameters required to set up the model. One of the largest limitations of traditional forms of ANN is that they tend to struggle with the computational complexity required to compute image data. Common machine learning benchmarking datasets such as the MNIST database of handwritten digits are suitable for most forms of ANN, due to its relatively small image dimensionality of just 28 × 28. With this dataset a single neuron in the first hidden layer will contain 784 weights (28×28×1 where 1 bare in mind that MNIST is normalised to just black and white values), which is manageable for most forms of ANN. If you consider a more substantial coloured image input of 64 × 64, the number of weights on just a single neuron of the first layer increases substantially to 12, 288. Also take into account that to deal with this scale of input, the network will also need to be a lot larger than one used to classify colour-normalised MNIST digits, then you will understand the drawbacks of using such models.

1.3.1 Convolutional Neural Networks (CNN) Architecture:

CNNs are feedforward networks in that information flow takes place in one direction only, from their inputs to their outputs. Just as artificial neural networks (ANN) are biologically inspired, so are CNNs. The visual cortex in the brain, which consists of alternating layers of simple and complex cells (Hubel & Wiesel, 1959, 1962), motivates their architecture. CNN architectures come in several variations; however, in general, they consist of convolutional and pooling (or subsampling) layers, which are grouped into modules. Either one or more fully connected layers, as in a standard feedforward neural network, follow these modules. Modules are often stacked on top of each other to form a deep model. It illustrates typical CNN architecture for a toy image classification task. An image is input directly to the network, and this is followed by several stages of convolution and pooling. Thereafter, representations from these operations feed one or more fully connected layers. Finally, the last fully connected layer outputs the class label. Despite this being the most popular base architecture found in the literature, several architecture changes have been proposed in recent years with the objective of improving image classification accuracy or reducing computation costs. Although for the remainder of this section, we merely fleetingly introduce standard CNN architecture.

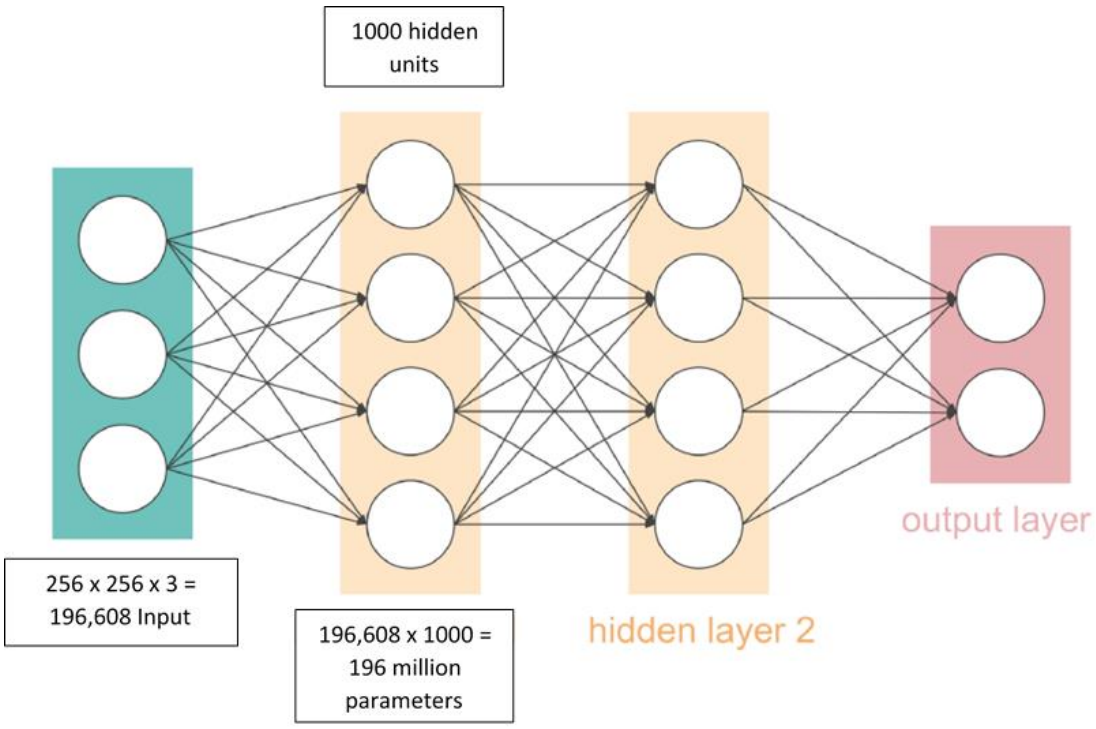

Fig 1.3.1 Convolution Neural Network Architecture

1.3.2 Overall Architecture:

CNNs are comprised of three types of layers. These are convolutional layers, pooling layers and fully connected layers. When these layers are stacked, a CNN architecture has been formed. A simplified CNN architecture for MNIST classification is illustrated in Figure 2. input 0 9 convolution w/ReLu pooling output fully-connected w/ ReLu fully-connected ... Fig. 2: An simple CNN architecture, comprised of just five layers The basic functionality of the example CNN above can be broken down into four key areas. 1. As found in other forms of ANN, the input layer will hold the pixel values of the image. 2. The convolutional layer will determine the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The rectified linear unit (commonly shortened to ReLu) aims to apply an 'elementwise' activation function such as sigmoid to the output of the activation produced by the previous layer. 3. The pooling layer will then simply perform down sampling along the spatial dimensionality of the given input, further reducing the number of parameters within that activation. 4. The fully-connected layers will then perform the same duties found in standard ANNs and attempt to produce class scores from the activations, to be used for classification. It is also suggested that ReLu may be used between these layers, as to improve performance. Through this simple method of transformation, CNNs are able to transform the original input layer by layer using convolutional and down sampling techniques to produce class scores for classification and regression purposes. However, it is important to note that simply understanding the overall architecture of a CNN architecture will not suffice. The creation and optimisation of these models can take quite some time, and can be quite confusing. We will now explore in detail the individual layers, detailing their hyperparameters and connectivities.

1.3.2.1 Convolutional Layers:

The convolutional layers serve as feature extractors, and thus they learn the feature representations of their input images. The neurons in the convolutional layers are arranged into feature maps. Each neuron in a feature map has a receptive field, which is connected to a neighborhood of neurons in the previous layer via a set of trainable weights, sometimes referred to as a filter bank. Inputs are convolved with the learned weights in order to compute a new feature map, and the convolved results are sent

through a nonlinear activation function. All neurons within a feature map have weights that are constrained to be equal; however, different feature maps within the same convolutional layer have different weights so that several features can be extracted at each location. As the name implies, the convolutional layer plays a vital role in how CNNs operate. The layers parameters focus around the use of learnable kernels. These kernels are usually small in spatial dimensionality, but spreads along the entirety of the depth of the input. When the data hits a convolutional layer, the layer convolves each filter across the spatial dimensionality of the input to produce a 2D activation map. These activation maps can be visualised. As we glide through the input, the scalar product is calculated for each value in that kernel. From this the network will learn kernels that 'fire' when they see a specific feature at a given spatial position of the input. These are commonly known as activations.

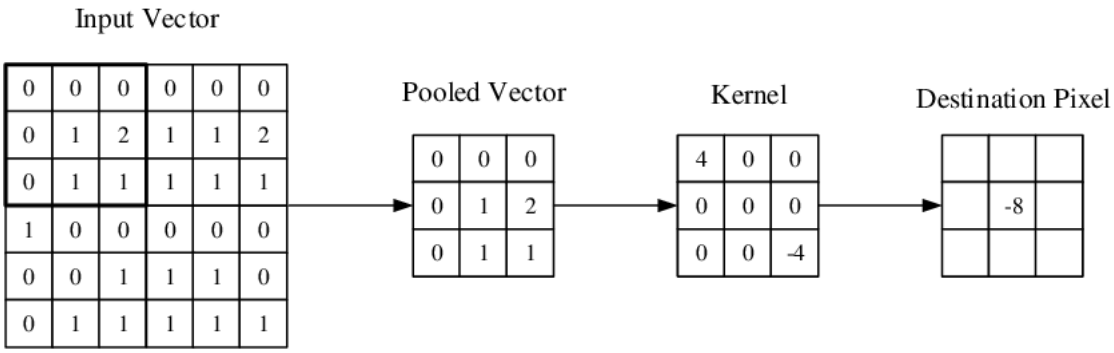

Fig 1.3.2.1 visual representation of convolutional layers

1.3.2.2 Pooling Layers:

The purpose of the pooling layers is to reduce the spatial resolution of the feature maps and thus achieve spatial invariance to input distortions and translations. Initially, it was common practice to use average pooling aggregation layers to propagate the average of all the input values, of a small neighbourhood of an image to the next layer. However, in more recent models, max pooling aggregation layers propagate the maximum value within a receptive field to the next layer. Pooling layers aim to gradually reduce the dimensionality of the representation, and thus further reduce the number of parameters and the computational complexity of the model. The pooling layer operates over each activation map in the input, and scales its dimensionality using the "MAX" function. In most CNNs, these come in the form of max-pooling layers with kernels of 31 a dimensionality of 2 × 2 applied with a stride of 2 along the spatial dimensions of the input. This scales the activation map down to 25% of the original size - whilst

maintaining the depth volume to its standard size. Due to the destructive nature of the pooling layer, there are only two generally observed methods of max-pooling. Usually, the stride and filters of the pooling layers are both set to 2 × 2, which will allow the layer to extend through the entirety of the spatial dimensionality of the input. Furthermore overlapping pooling may be utilised, where the stride is set to 2 with a kernel size set to 3. Due to the destructive nature of pooling, having a kernel size above 3 will usually greatly decrease the performance of the model. It is also important to understand that beyond max-pooling, CNN architectures may contain general pooling. General pooling layers are comprised of pooling neurons that are able to perform a multitude of common operations including L1/L2-normalisation, and average pooling. However, this tutorial will primarily focus on the use of max-pooling.

1.3.2.3 Fully Connected Layers:

Several convolutional and pooling layers are usually stacked on top of each other to extract more abstract feature representations in moving through the network. The fully connected layers that follow these layers interpret these feature representations and perform the function of high-level reasoning. For classification problems, it is standard to use the softmax operator on top of a DCNN. While early success was enjoyed by using radial basis functions (RBFs), as the classifier on top of the convolutional towers found that replacing the softmax operator with a support vector machine (SVM) leads to improved classification accuracy. The fully-connected layer contains neurons of which are directly connected to the neurons in the two adjacent layers, without being connected to any layers within them. This is analogous to way that neurons are arranged in traditional forms of ANN. Despite the relatively small number of layers required to form a CNN, there is no set way of formulating a CNN architecture. That being said, it would be idiotic to simply throw a few of layers together and expect it to work. Through reading of related literature, it is obvious that much like other forms of ANNs, CNNs tend to follow a common architecture.. 32 Convolutional Neural Networks differ to other forms of Artificial Neural Network in that instead of focusing on the entirety of the problem domain, knowledge about the specific type of input is exploited. This in turn allows for a much simpler network architecture to be set up. This paper has outlined the basic concepts of Convolutional Neural Networks, explaining the layers required to build one and detailing how best to structure the network in most image analysis tasks. Research in the field of image analysis using neural networks has somewhat slowed in

recent times. This is partly due to the incorrect belief surrounding the level of complexity and knowledge required to begin modelling these superbly powerful machine learning algorithms. The authors hope that this paper has in some way reduced this confusion, and made the field more accessible to beginners.

1.4 Introduction to OpenCV:

OpenCV stands for Open supply pc Vision Library is associate open supply pc vision and machine learning software system library. The purpose of creation of OpenCV was to produce a standard infrastructure for computer vision applications and to accelerate the utilization of machine perception within the business product [6]. It becomes very easy for businesses to utilize and modify the code with OpenCV as it is a BSD-licensed product. It is a rich wholesome library as it contains 2500 optimized algorithms, which also includes a comprehensive set of both classic and progressive computer vision and machine learning algorithms. These algorithms is used for various functions such as discover and acknowledging faces. Identify objects classify human actions. In videos, track camera movements, track moving objects. Extract 3D models of objects, manufacture 3D purpose clouds from stereo cameras, sew pictures along to provide a high-resolution image of a complete scene, find similar pictures from a picture information, remove red eyes from images that are clicked with the flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality. Officially launched in 1999 the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.

- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.

- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008. The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months and development is now done by an independent Russian team supported by commercial corporations. 39 40 In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site. On May 2016, Intel signed an agreement to acquire ITSEEZ, a leading developer of OpenCV. OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real time computer vision. Originally developed by Intel, it was later supported by Willow Garage then ITSEEZ (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

OpenCV's application areas include:

♣ 2D and 3D feature toolkits

♣Ego motion estimation

♣ Facial recognition system

♣ Gesture recognition

♣ Human–computer interaction (HCI)

♣ Mobile robotics

♣ Motion understanding

♣ Object identification

♣ Segmentation and recognition

♣ Stereopsis stereo vision: depth perception from 2 cameras

♣ Structure from motion (SFM)

♣ Motion tracking

♣ Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

♣ Boosting Decision tree learning

♣ Gradient boosting trees

♣ Expectation-maximization algorithm

♣ k-nearestneighbor algorithm

♣ Naive Bayes classifier

♣ Artificial neural networks

♣ Random forest

♣ Random forest

♣ Support vector machine (SVM)

♣ Deep neural networks (DNN)

1.4.1 Libraries in OpenCV:

1.4.1.1 NumPy:

NumPy is an acronym for "Numeric Python" or "Numerical Python". It is an open-source extension module for Python, which provides fast precompiled functions for mathematical and numerical routines. Furthermore, NumPy enriches the programming language Python with powerful data structures for efficient computation of multi-dimensional arrays and matrices. The implementation is even aiming at huge matrices

and arrays. Besides that, the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

♣ A powerful N-dimensional array object Sophisticated (broadcasting) functions

♣ Tools for integrating C/C++ and Fortran code

♣Useful linear algebra, Fourier Transform, and random number capabilities.

1.4.1.2 Numpy Array:

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

1.4.1.3 SciPy:

SciPy (Scientific Python) is often mentioned in the same breath with NumPy. SciPy extends the capabilities of NumPy with further useful functions for minimization, regression, Fourier transformation and many others. NumPy is based on two earlier Python modules dealing with arrays. One of these is Numeric. Numeric is like NumPy a Python module for high-performance, numeric computing, but it is obsolete nowadays. Another predecessor of NumPy is Numarray, which is a complete rewrite of Numeric but is deprecated as well. NumPy is a merger of those two, i.e. it is build on the code of Numeric and the features of Numarray.

1.5 Libraries Used:

1.5.1 TensorFlow: TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

1.5.2 keras: Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast

experimentation. Being able to go from idea to result as fast as possible is key to doing good research.

1.5.3 tkinter:

Tkinter tutorial provides basic and advanced concepts of Python Tkinter. Our Tkinter tutorial is designed for beginners and professionals.

Python provides the standard library Tkinter for creating the graphical user interface for desktop-based applications.

1.5.4 matplotlib:

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

1.6 Datasets

Handwritten character recognition is an expansive research area that already contains detailed ways of implementation which include major learning datasets, popular algorithms, features scaling and feature extraction methods. MNIST dataset (Modified National Institute of Standards and Technology database) is the subset of the NIST dataset which is a combination of two of NIST's databases: Special Database 1 and Special Database 3. Special Database 1 and Special Database 3 consist of digits written by high school students and employees of the United States Census Bureau, respectively. MNIST contains a total of 70,000 handwritten digit images (60,000 - training set and 10,000 - test set) in 28x28 pixel bounding box and anti-aliased. All these images have corresponding Y values which apprises what the digit is.

Fig 1.6.1 visual representation of convolutional layers

2. LITERATURE SURVEY

Anuj Dutt in his paper demonstrated that utilizing Deep Learning systems, he had the capacity to get an extremely high measure of accuracy. By utilizing the convolutional Neural Network with Keras and Theano as backend, he was getting an accuracy of 98.72%. In addition, execution of CNN utilizing Tensorflow gives a stunningly better consequence of 99.70%. Even though the complication of the procedure and codes appears to be more when contrasted with typical Machine Learning algorithms yet the accuracy he got is increasingly obvious. In a paper published by Saeed AL-Mansoori, Multilayer Perceptron (MLP) Neural Network was implemented to recognize and predict handwritten digits from 0 to 9. The proposed neural system was trained and tested on a dataset achieved from MNIST.

3. REQUIREMENTS AND ANALYSIS

3.1 Existing system

These days, an ever-increasing number of individuals use pictures to transmit data. It is additionally mainstream to separate critical data from pictures. Image Recognition is an imperative research area for its generally used applications. In general, the field of pattern recognition, one of the difficult undertakings is the precise computerized recognition of human handwriting. Without a doubt, this is a very difficult issue because there is an extensive diversity in handwriting from an individual to another individual. Although, this difference does not make any issues to people, yet, anyway it is increasingly hard to instruct computers to interpret general handwriting. For the image recognition issue, for example, handwritten classification, it is essential to make out how information is depicted onto images. Handwritten Recognition from the MNIST dataset is well known among scientists as by utilizing different classifiers for various parameters, the error rate has been decreased, for example, from linear classifier (1-layer NN) with 12% to 0.23% by a board of 35 convolution neural systems. The scope of this is to implement a Handwritten Digit Recognition framework and think about the diverse classifiers and different techniques by concentrating on how to accomplish close to human performance. For an undertaking of composing diverse digits (0-9) for various people the general issue confronted would be of digit order issue and the closeness between the digits like 1 and 7, 5 and 6, 3 and 8, 9 and 8 and so forth. Additionally, individuals compose a similar digit from various perspectives, the uniqueness and assortment in the handwriting of various people likewise impact the development and presence of the digits.

3.2 Proposed system

Handwritten Digit Recognition is the ability of a computer to recognize the human handwritten digit from different sources like images or papers and classify them into 10 predefined class (0-9). So, through this project i.e., Handwritten Digit Recognition, we can achieve the solution of this problem. The goal is to correctly identify and classify digits (numeric) from a dataset of tens of thousands of handwritten images. This can be done by using deep learning/CNN.

3.2.1 Project scope

The scope of the project is to build a reliable model that correctly predicts the handwritten digits.

3.2.2 Project objectives

The model has some objectives to achieve. They are as follows

- To train a model with a huge number of handwritten digit images from MNIST dataset.
- To create a reliable model that correctly predicts the handwritten digits using Convolutional Neural Networks (CNN).

3.3 Functional Requirements

The functional requirements describe the inputs and outputs of the application. The functional requirements of this project are as follows:

- MNIST Dataset.
- Convolutional Neural Networks.
- User Interface to draw the digits that are needed to be predicted.

3.3.1 Actors

Actors represent external entities that interact with the system. An actor can be human or an external system. During this activity, developers identify the actors involved in this system. In this project, user is:

User: Any person

3.3.2 Use cases

Use cases are used during requirement elicitation and analysis to represent the functionality of the system. Use cases focus on the behaviour of the system from an external point of view. A use case describes a function provided by the system that yields a visible result for an actor. An actor describes any entity that interacts with the system.

The identification of actors and use cases results in the definition of the boundary of the system, which is, in differentiating the tasks accomplished by the system and the tasks accomplished by its environment. The actors are outside the boundary of the system, whereas the use cases are inside the boundary of the system.

Actors are external entities that interact with the system. Use cases describe the behaviour of the system as seen from an actor's point of view. Actors initiate a use case to access the system functionality. The use case then initiates other use cases and gathers more information from the actors. When actors and use cases exchange information, they are said to be communicate.

To describe a use case, we use a template composed of six fields:

Use Case Name	:	The name of the use case
Participating Actors	:	The actors participating in the particular use case
Flow of events	:	Sequence of steps describing the function of use case
Exit Condition	:	Condition for terminating the use case
Quality Requirements	:	Requirements that do not belong to the use case but constraint the functionality of the system.

3.4. Non-Functional Requirements:

- Dependability
- Quick response
- User friendliness

3.4.1 Hardware Specifications

No hardware devices used.

3.4.2 Software Specifications

Operating System : Windows OS

Coding language : Python

Python

Python is an interpreted, high-level, general purpose programming language. Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional, and procedural. It also has a comprehensive standard library.

Python interpreters are available for many operating systems. Python, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Fig. 3.4.1.9 Python Language logo

3.5 ANALYSIS

The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished. This phase defines the problem that the customer is trying to solve. The deliverable result at the end of this phase is a requirement document. Analysis object model is represented by class and object diagrams. Analysis focuses on producing a model of the system, called the Analysis

model, which is correct, complete, consistent, and verifiable. The analysis model is composed of three individual models: The Functional Model represented by use cases and scenarios, the Analysis Object Model, represented by class and object diagrams, and the Dynamic Model, represented by state chart and sequence diagrams. The analysis model represents the system under development from the user's point of view. The analysis object model is a part of the analysis and focuses on the individual concepts that are manipulated by the system, their properties and their relationships.

3.5.1 Entity Objects

The Analysis object model consists of entity, boundary and control objects. Entity objects represent the persistent information tracked by the system. Participating objects form the basis of the analysis model.

3.5.2 Boundary Objects

Boundary object is the object used for interaction between the user and the system. Moreover, it is an interface used to communicate with the system. Boundary objects represent the system interface with the actors. In each use case, each actor interacts with at least one boundary object. The boundary object collects the information from the actor and translates into an interface model from that can be used by the objects and by the control objects.

3.5.3 Control Objects

Control objects are responsible for coordinating entity objects and boundary objects. A control object is creating at the beginning of the use cases and ceases to exist at its end. Control objects usually do not have a concrete counterpart in the real world. Control object is a responsible for collecting information from the boundary objects and dispatching it to entity object. Here only authorized users can operate, and details obtained are verified. The control objects in this project are admission, result, placement, company, and eligibility details.

3.5.4 Object Interaction

Interaction diagrams model the behaviour of use cases by describing the way groups of objects interact to complete the task. The two kinds of interaction diagrams

are sequence and collaboration diagrams. Sequence diagrams generally show the sequence of events that occur.

3.5.5 Object Behaviour

State chart diagrams are used to describe the behaviour of a system. They define different states of an object during its lifetime. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system and these states are changed by events. State chart diagram describes the flow of control from one state to another state. State chart diagrams are very important for describing the states. States can be identified as the condition of objects when an event occurs.

4. SYSTEM DESIGN

System Design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. In System design, developers:

- Define design goals of the project
- Decompose the system into smaller sub systems
- Design hardware/software strategies
- Design persistent data management strategies
- Design global control flow strategies
- Design access control policies and
- Design strategies for handling boundary conditions.

System design is not algorithmic. It is decomposed of several activities. They are:

- Identify Design Goals
- Design the initial subsystem decomposition
- Refine the subsystem decomposition to address the design goals.

System Design is the transform of analysis model into a system design model. Developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams. Developers also select strategies for building the system, such as the hardware/software platform on which the system will run, the persistent data management strategy, the goal control flow the access control policy and the handling of boundary conditions. The result of the system design is model that includes a clear description of each of these strategies, subsystem decomposition, and a UML deployment diagram representing the hardware/software mapping of the system.

4.1 Design Goals

Design goals are the qualities that the system should focus on. Many design goals can be inferred from the non-functional requirements or from the application domain.

User friendly: The system is user friendly because it is easy to use and understand.

Reliability: Proper checks are there for any failure in the system if they exist.

4.2 System Architecture

As the complexity of systems increases, the specification of the system decomposition is critical. Moreover, subsystem decomposition is constantly revised whenever new issues are addressed. Subsystems are merged into alone subsystem, a complex subsystem is split into parts, and some subsystems are added to take care of new functionality. The first iterations over the subsystem decomposition can introduce drastic changes in the system design model.

4.3 System Design

The reason behind this document is to investigate the design possibilities of the proposed system, such as architecture design, block diagram, sequence diagram, data flow diagram and user interface design of the system to define the steps such as pre-processing, feature extraction, segmentation, classification and recognition of digits.

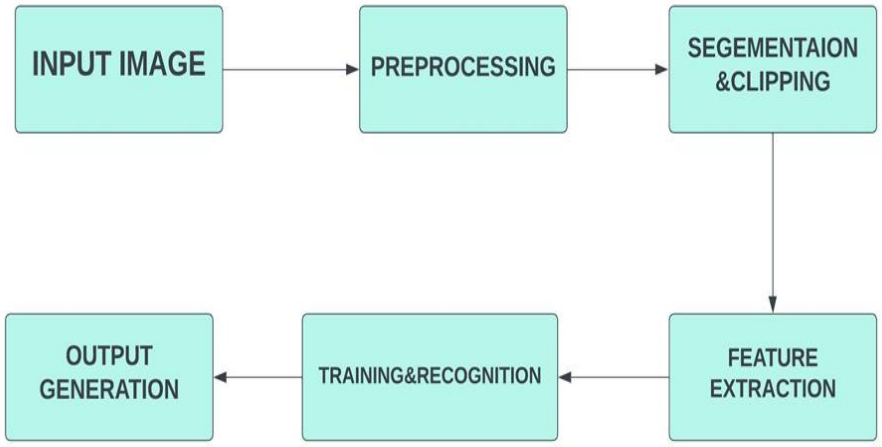

Fig 4.3.1 Architecture of the Proposed System

The above Figure illustrates the architecture diagram of the proposed system. The proposed model contains the four stages to classify and detect the digits: A. Pre-processing B. Segmentation C. Feature Extraction D. Classification and Recognition.

4.4 UML Diagrams

A UML diagram is a diagram based on the UML (Unified Modelling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.

UML is a modern approach to modelling and documenting software. In fact, it's one of the most popular business process modelling techniques. It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

Mainly UML has been used as a general-purpose modelling language in the field of software engineering. However, it has row found its way into the documentation of several business processes or workflows. For example, activity diagrams, a type of UML diagram, can be used as a replacement for flowcharts. They provide both a more standardized way of modelling workflows as well as a wider range of features to improves readability.

4.4.1 Use Case Diagram

In the Unified Modelling Language (UML), a use case diagram can summarize the details of your system's users(also known as actors) and their interactions with the system. To build one, use a set of specialized symbols and connectors. An effective use case diagram can help the team discuss and represent:

- Scenarios in which system or application interacts with people, organizations, or external systems.
- Goals that your system or application helps those entities (known as actors) achieve.
- The scope of the system.

UML use case diagrams are ideal for:

- Representing the goals of system-user interactions.
- Defining and organizing functional requirements in a system.
- Specifying the context and requirements of a system.

- Modelling the basic flow of events in a use case.

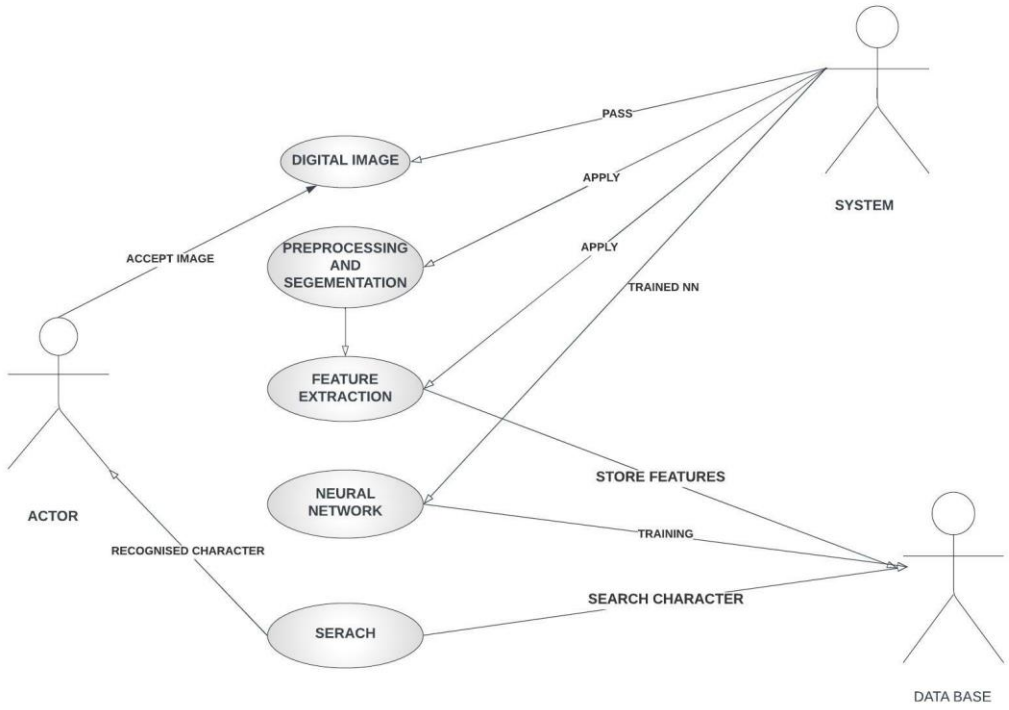

Fig. 4.4.1.1 Use Case Diagram

4.4.2 Class Diagram

Class diagrams are one of the most useful types of diagrams in UML as they clearly map out the structure of a particular system by modelling its classes, attributes, operations, and relationships between objects. It is a static diagram that represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

The class shape itself consists of a rectangle with three rows. The top row contains the name of the class, the middle row contains the attributes of the class, and the bottom section express the methods or operations that the class may use. Classes and subclasses are grouped together to show the static relationship between each object.

The purpose of the class diagram can be summarized as:

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.

- Base for component and deployment diagrams.
- Forward and reverse engineering.

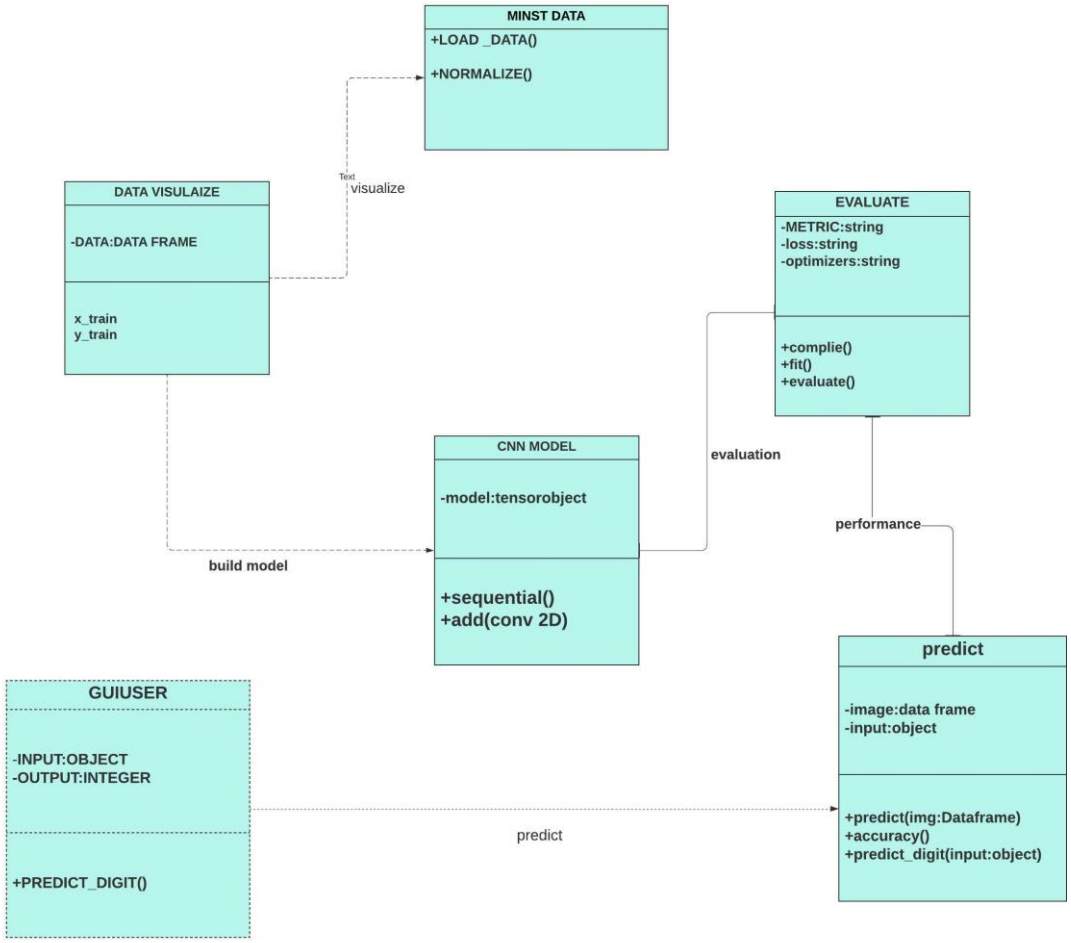

Fig. 4.4.2.1 Class Diagram

4.4.3 Activity Diagram

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc. Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques.

The purpose of an activity diagram can be described as

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

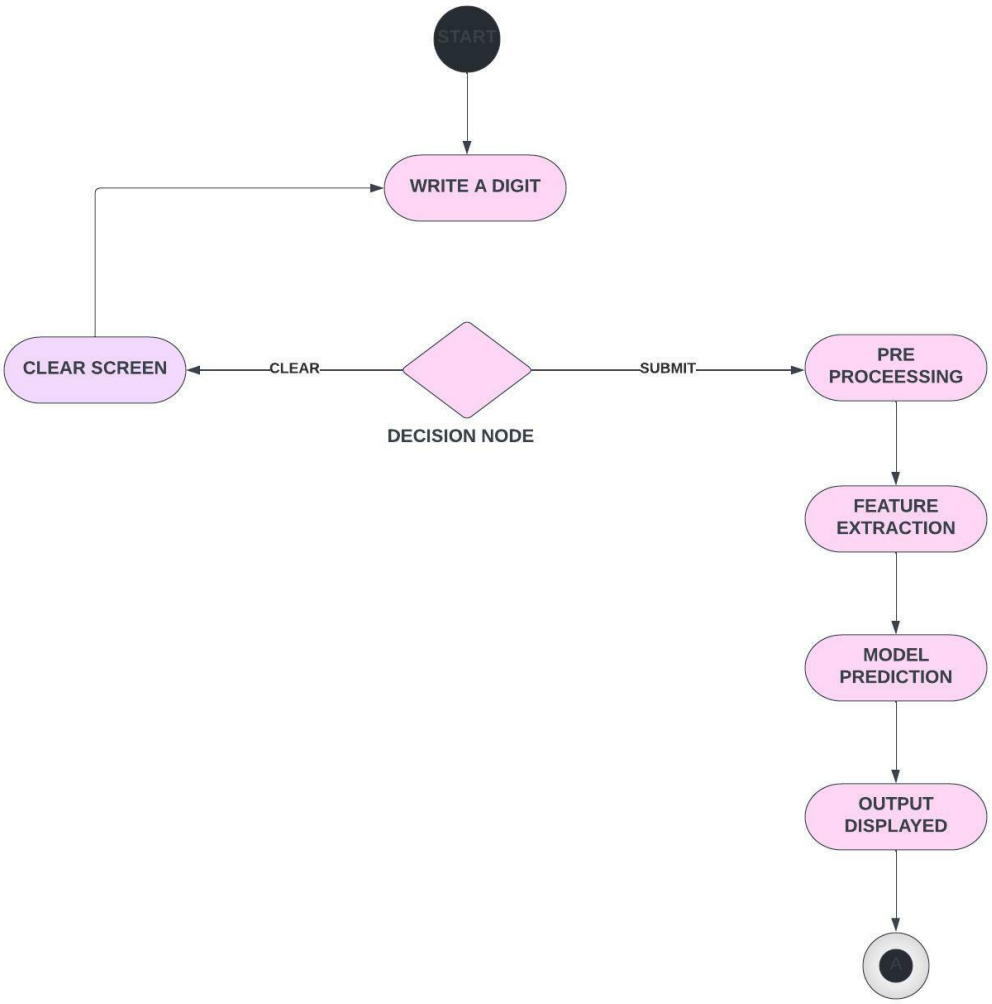

Fig. 4.4.3.1 Activity Diagram

4.4.4 Sequence Diagram

A sequence diagram is the most commonly used interaction diagram. It simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

Sequence diagrams can be useful references for businesses and other organizations. Purpose of sequence diagrams are:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

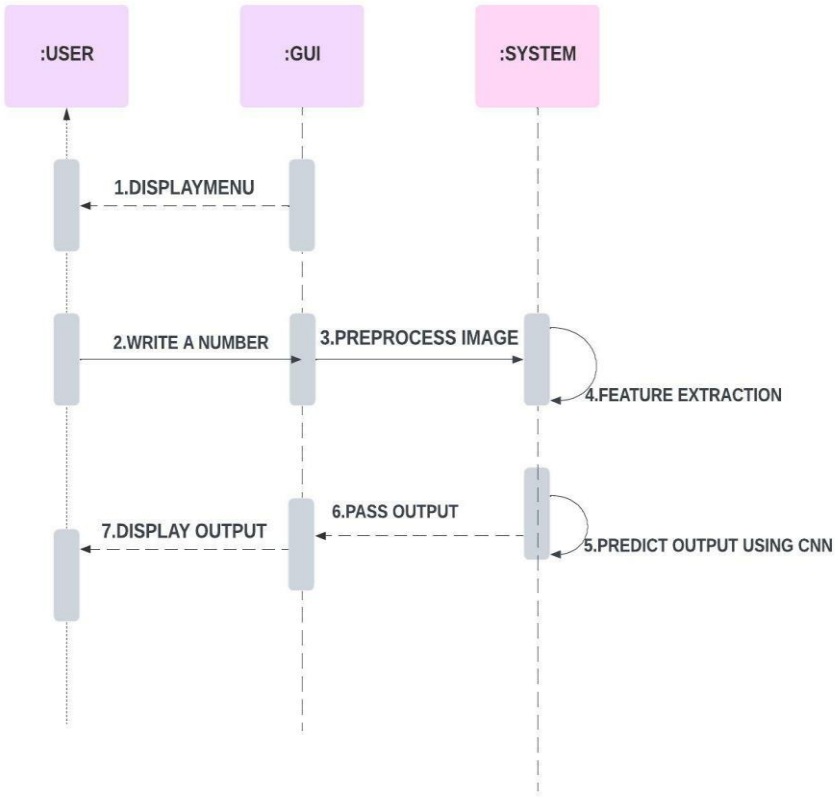

Fig. 4.4.4.1 Sequence Diagram

4.5 Data Flow Diagrams

A Data Flow Diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process

overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled.

4.5.1 DFD Level-0

DFD Level 0 is also called as Context Diagram. It's a basic overview of the whole system or process being analysed or modelled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. In our project the external entity is driver.

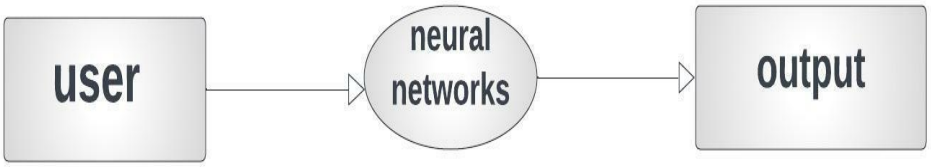

Fig. 4.5.1.1 DFD Level 0

4.5.2 DFD Level-1

DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into the subprocesses.

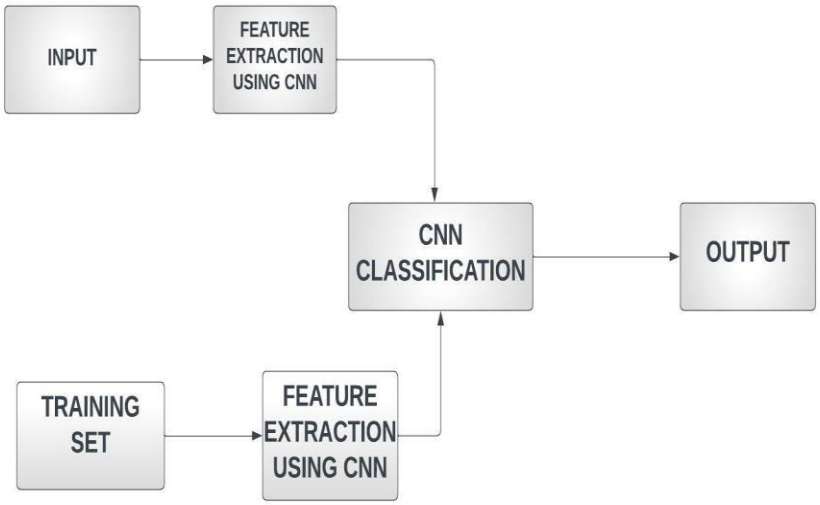

Fig 4.5.2.1 DFD Level 1

4.6 Flow Chart

A flowchart is a type of diagram that represents an algorithm, workflow or process. Flowchart can also be define as a diagrammatic representation of an algorithm. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analysing, designing, documenting or managing a process or program in various fields.

The first step in flow of this system involves loading the digit dataset from MNIST. Once the device is loaded, the images from the dataset are pre-processed. It is followed by training and testing the model. The predictions are carried out.

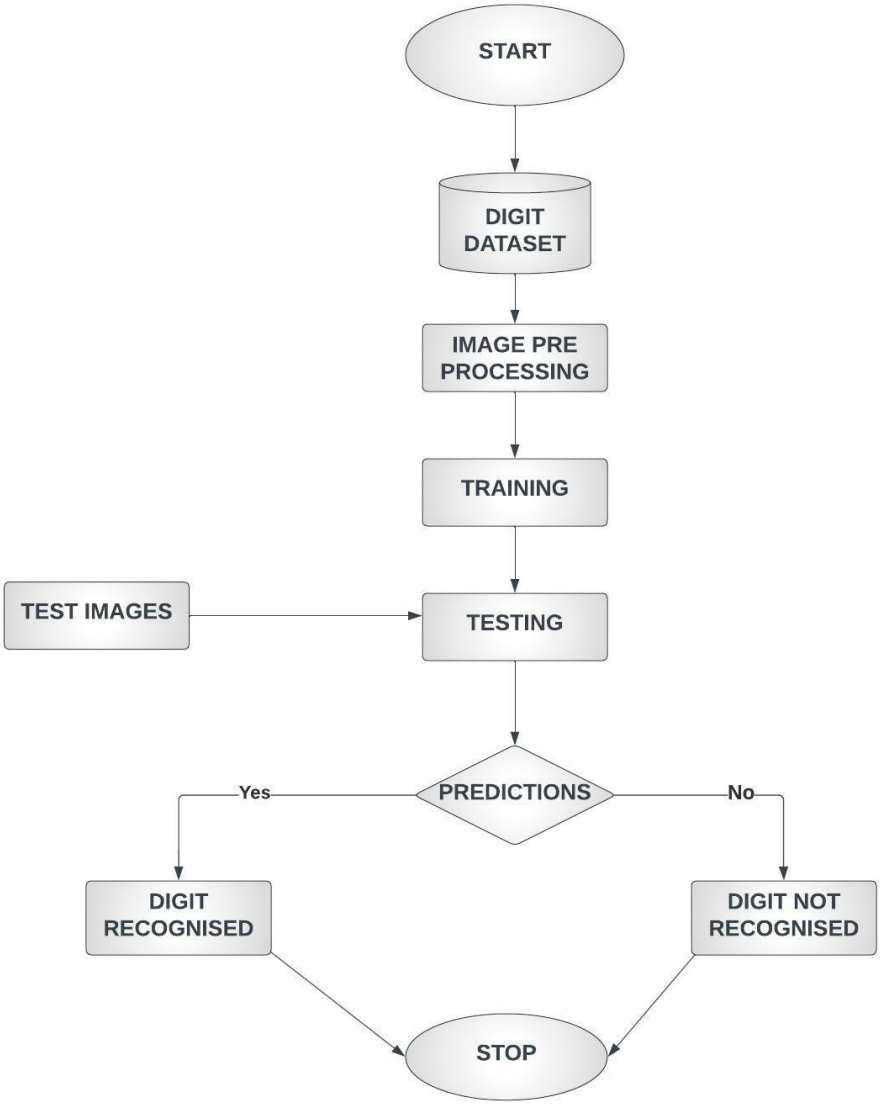

Fig. 4.6.1 Flow chart of the system

5. IMPLEMENTATION

The objective of the coding or programming phase is to translate the design of the system produced during the design phase into code in a given programming language, which can be executed by a computer and that performs the computation specified by the design.

The coding phase affects both testing and maintenance. The goal of coding is not to reduce the implementation cost, but the goal should be to reduce the cost of later phases.

5.1 Coding Approach

There are two major approaches for coding any software system. They are Top-Down approach and bottom up approach.

Bottom-up Approach can best suit for developing the object-oriented systems. During system design phase, we decompose the system into an appropriate number of subsystems, for which objects can be modelled independently. These objects exhibit the way the subsystems perform their operations.

Once objects have been modelled, they are implemented by means of coding. Even though related to the same system as the objects are implemented of each other, the Bottom-Up approach is more suitable for coding these objects. In this approach, we first do the coding of objects independently and then we integrate these modules into one system to which they belong.

This code will first allow the admin to login with a valid user id and password. After this, admin will select the admission or academic module and enter all the students' details branch wise. TPO will select placement module and enter the details based on academic performance. Reports are generated accordingly and can be viewed by teaching staff and students.

5.2 Information Handling

Any software system requires some amount of information during its operation selection of appropriate data structures can help us to produce the code so that objects of the system can better operate with the available information decreased complexity.

5.3 Programming Style

Programming style deals with act of rules that a programmer must follow so that the characteristics of coding such as Traceability, Understandability, Modifiability, and Extensibility can be satisfied. In this current system, we followed the coding rules for naming the variables and methods.

5.4 Verification and Validation

Verification is the process of checking the product built is right. Validation is the process of checking whether the right product is built. During the Development of the system, Coding for the object has been thoroughly verified from different aspects regarding their design, in the way they are integrated etc. The various techniques that have been followed for validation discussed in testing the current system.

Validations applied to the entire system at two levels:

Form level Validation:

Validations of all the inputs given to the system at various points in the forms are validated while navigating to the next form. System raises appropriate custom and pre-defined exceptions to alert the user about the errors occurred or likely to occur.

Field level Validation:

Validations at the level of individual controls are also applied wherever necessary. System pops up appropriate and sensuous dialogs wherever necessary. In this project**,** validations are performed on each individual control. If any one of text field is not filled or any wrong, click occurs then system will generate appropriate exceptions.

5.5 Implementation of Project

Training The Model

A. Pre-Processing: The role of the pre-processing step is it performs various tasks on the input image. It basically upgrades the image by making it reasonable for segmentation. For the most part, noise filtering, smoothing and standardization are to be done in this stage. The pre-processing additionally characterizes a smaller portrayal of the example. Binarization changes over a gray scale image into a binary image. The initial approach to the training set images that are to be processed in order to reduce the data, by thresholding them into a binary image.

B. Segmentation: Once the pre-processing of the input images is completed, sub-images of individual digits are formed from the sequence of images. Each individual digit is resized into pixels.

C. Feature Extraction: After the completion of pre-processing stage and segmentation stage, the pre-processed images are represented in the form of a matrix which contains pixels of the images that are of very large size. In this way it will be valuable to represent the digits in the images which contain the necessary information. This activity is called feature extraction. In the feature extraction stage redundancy from the data is removed.

D. Classification and Recognition: In the classification and recognition step the extracted feature vectors are taken as an individual input to CNN classifier.

User Interface

In this module an application is developed for the user to interact with. User is provided with a Canvas. User can draw a digit between 0-9. When the button Predict is clicked, the digit is recognized. If the user wants to try again, he can click the clear button which erases the drawing canvas.

6. TESTING

Testing is the process of finding differences between the expected behaviour specified by system models and the observed behaviour of the system. Testing is a critical role in quality assurance and ensuring the reliability of development and these errors will be reflected in the code, so the application should be thoroughly tested and validated.

Unit testing finds the differences between the object design model and its corresponding components. Structural testing finds differences between the system design model and a subset of integrated subsystems. Functional testing finds differences between the use case model and the system.

Finally, performance testing, finds differences between non-functional requirements and actual system performance. From modelling point of view, testing is the attempt of falsification of the system with respect to the system models. The goal of testing is to design tests that exercise defects in the system and to reveal problems.

6.1 Testing Activities

Testing a large system is a complex activity and like any complex activity. It has to be broke into smaller activities. Thus, incremental testing was performed on the project i.e., components and subsystems of the system were tested separately before integrating them to form the subsystem for system testing.

6.2 Testing Types

Unit Testing

Unit testing focuses on the building blocks of the software system that is the objects and subsystems. There are three motivations behind focusing on components. First unit testing reduces the complexity of overall test activities allowing focus on smaller units of the system, second unit testing makes it easier to pinpoint and correct faults given that few components are involved in the rest. Third unit testing allows parallelism in the testing activities, that is each component are involved in the test. Third unit testing allows parallelism in the testing activities that is each component can be tested independently of one another. The following are some unit testing techniques.

- **Equivalence testing**:

  It is a black box testing technique that minimizes the number of test cases. The possible inputs are partitioned into equivalence classes and a test case is selected for each class.

- **Boundary testing:**

  It is a special case of equivalence testing and focuses on the conditions at the boundary of the equivalence classes. Boundary testing requires that the elements be selected from the edges of the equivalence classes.

- **Path testing:**

  It is a white box testing technique that identifies faults in the implementation of the component the assumption here is that exercising all possible paths through the code at least once. Most faults will trigger failure. This acquires knowledge of source code.

Integrating Testing

Integration testing defects faults that have not been detected. During unit testing by focusing on small groups on components two or more components are integrated and tested and once tests do not reveal any new faults, additional components are added to the group. This procedure allows testing of increasing more complex parts on the system while keeping the location of potential faults relatively small. I have used the following approach to implements and integrated testing.

Top-down testing strategy unit tests the components of the top layer and then integrated the components of the next layer down. When all components of the new layer have been tested together, the next layer is selected. This was repeated until all layers are combined and involved in the test.

Validation Testing

The systems completely assembled as package, the interfacing have been uncovered and corrected, and a final series of software tests are validation testing. The validation testing is nothing but validation success when system functions in a manner that can be reasonably expected by the customer. The system validation had done by series of Black-box test methods.

System Testing

1. System testing ensures that the complete system compiles with the functional requirements and non-functional requirements of the system, the following are some system testing activities.
2. Functional testing finds differences between the functional between the functional requirements and the system. This is a black box testing technique. Test cases are divided from the use case model.
3. Performance testing finds differences between the design and the system the design goals are derived from the functional requirements.
4. Pilot testing the system is installed and used by a selected set of users – users exercise the system as if it had been permanently installed.
5. Acceptance testing, I have followed benchmarks testing in a benchmark testing the client prepares a set of test cases represent typical conditions under which the system operates. In our project, there are no existing benchmarks.

Installation testing, the system is installed in the target environment.

Black Box Testing also known as Behavioural Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional. This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behaviour or performance errors
- Initialization and termination errors

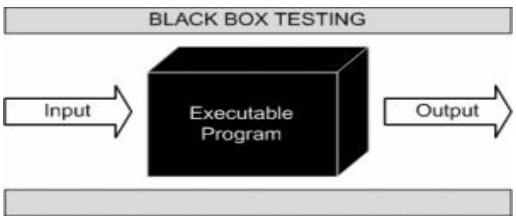

Fig. 6.2.1 Black Box Testing

6.3 Test cases

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass/Fail
1	Digit one is drawn on the canvas	Digit	1	1	PASS
2	Digit two is drawn on the canvas	Digit	2	2	PASS
3	Digit three is drawn on the canvas	Digit	3	3	PASS
4	Digit four is drawn on the canvas	Digit	4	4	PASS
5	Digit five is drawn on the canvas	Digit	5	5	PASS
6	Digit six is drawn on the canvas	Digit	6	6	PASS
7	Digit seven is drawn on the canvas	Digit	7	7	PASS
8	Digit eight is drawn on the canvas	Digit	8	8	PASS

9	Digit nine is drawn on the canvas	Digit	9	9	PASS
10	Digit zerois drawn on the canvas	Digit	0	0	PASS

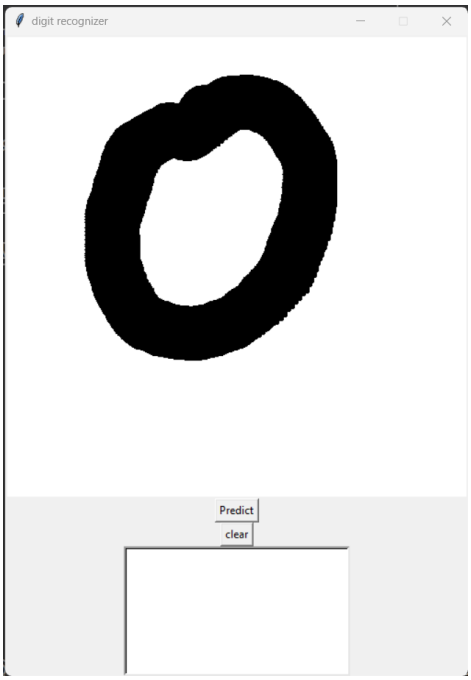

Fig 6.3.1 digit 0 drawn on canvas

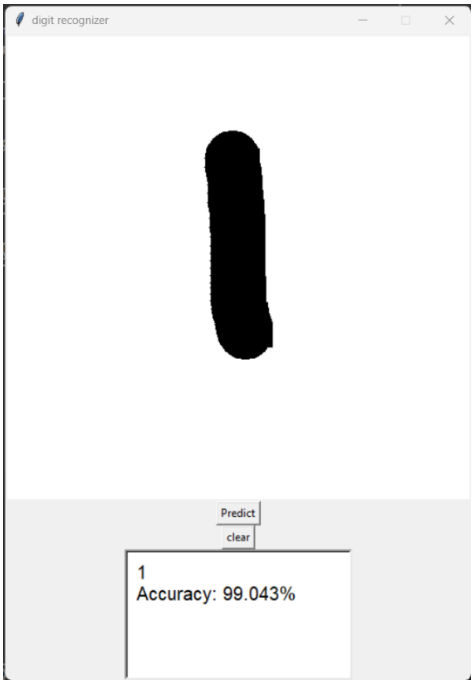

Fig 6.3.2 digit 1 drawn on canvas

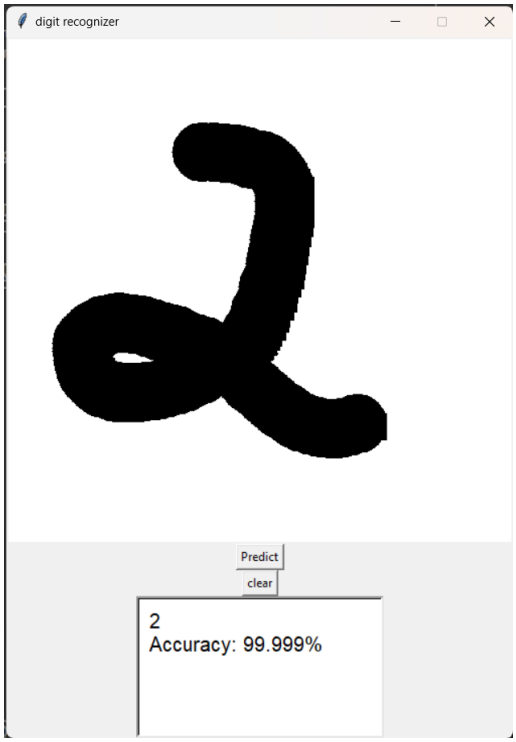

Fig 6.3.3 digit 2 drawn on canvas

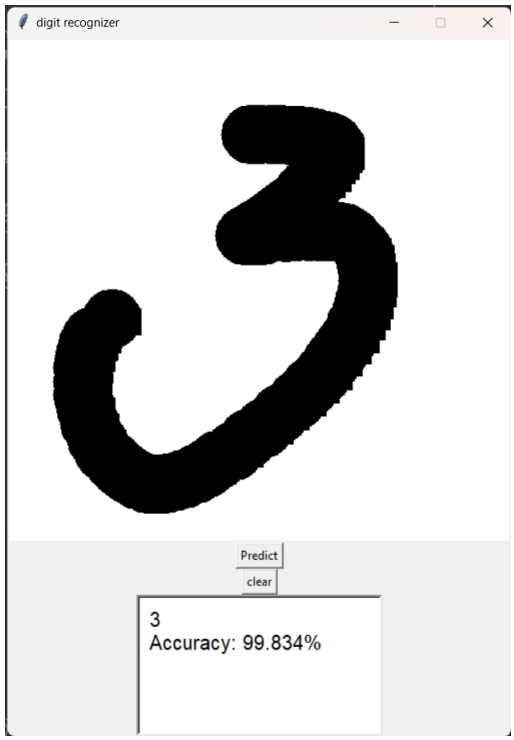

Fig 6.3.4 digit 3 drawn on canvas

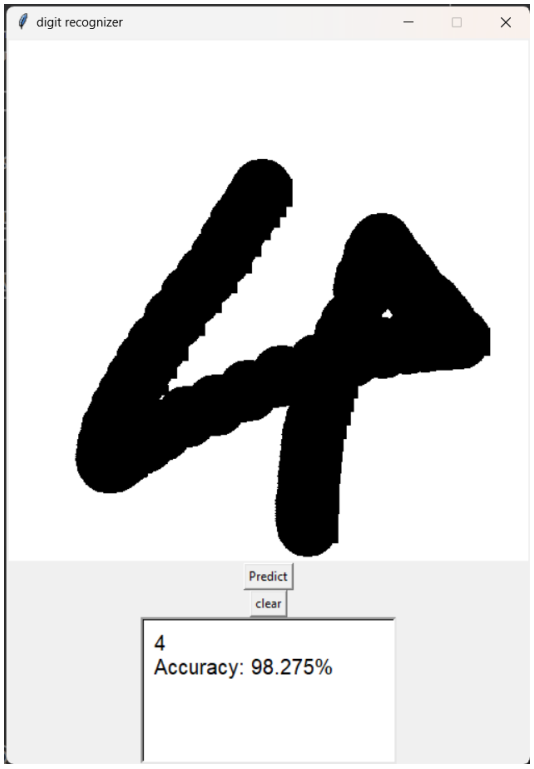

Fig 6.3.5 digit 4 drawn on canvas

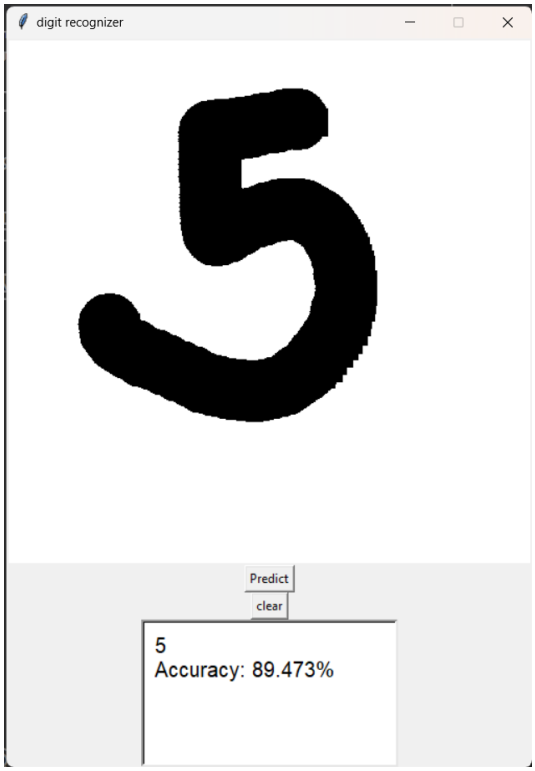

Fig 6.3.6 digit 5 drawn on canvas

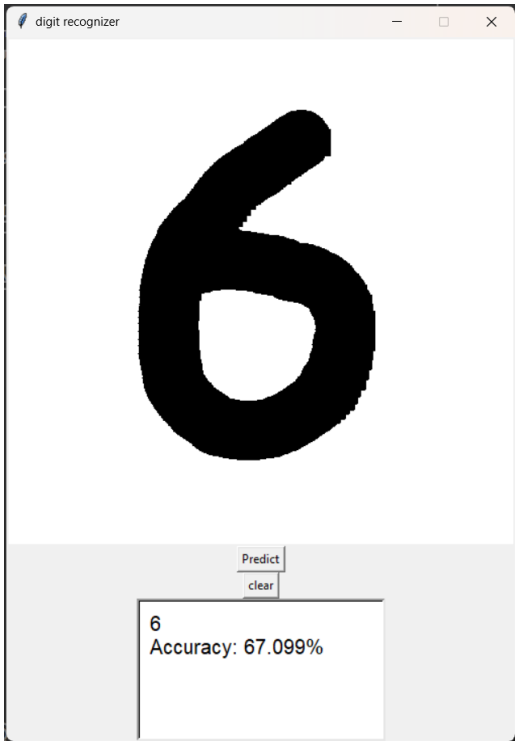

Fig 6.3.7 digit 6 drawn on canvas

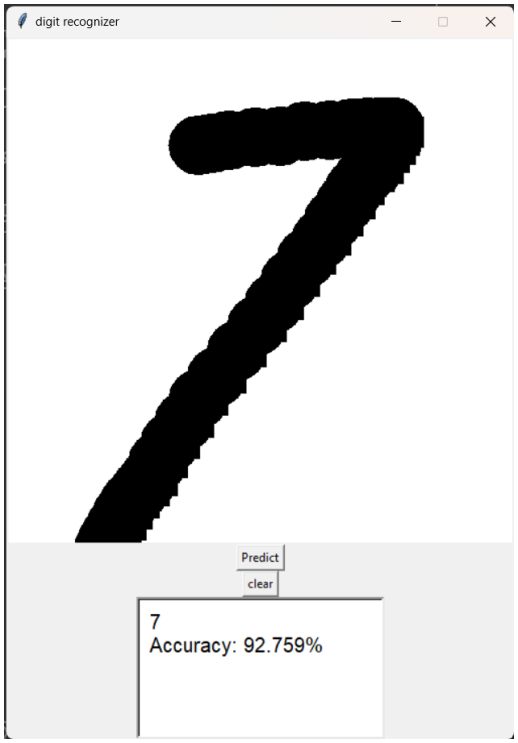

Fig 6.3.8 digit 7 drawn on canvas

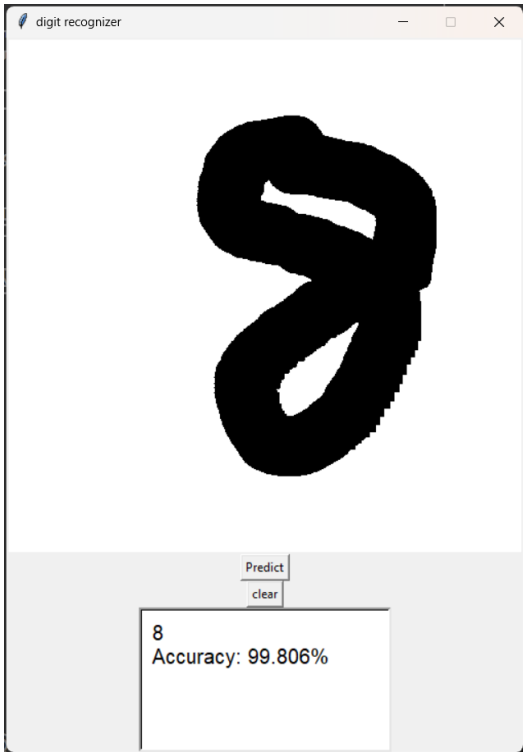

Fig 6.3.9 digit 8 drawn on canvas

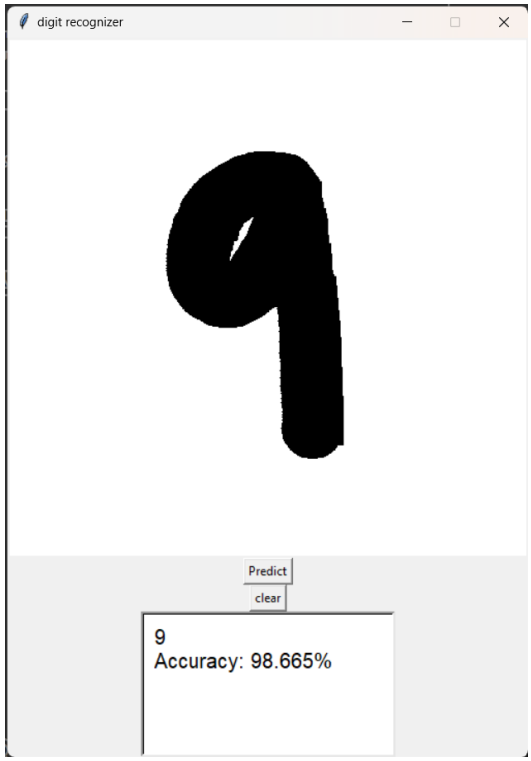

Fig 6.3.10 digit 9 drawn on canvas

7. RESULTS

7.1 canvas

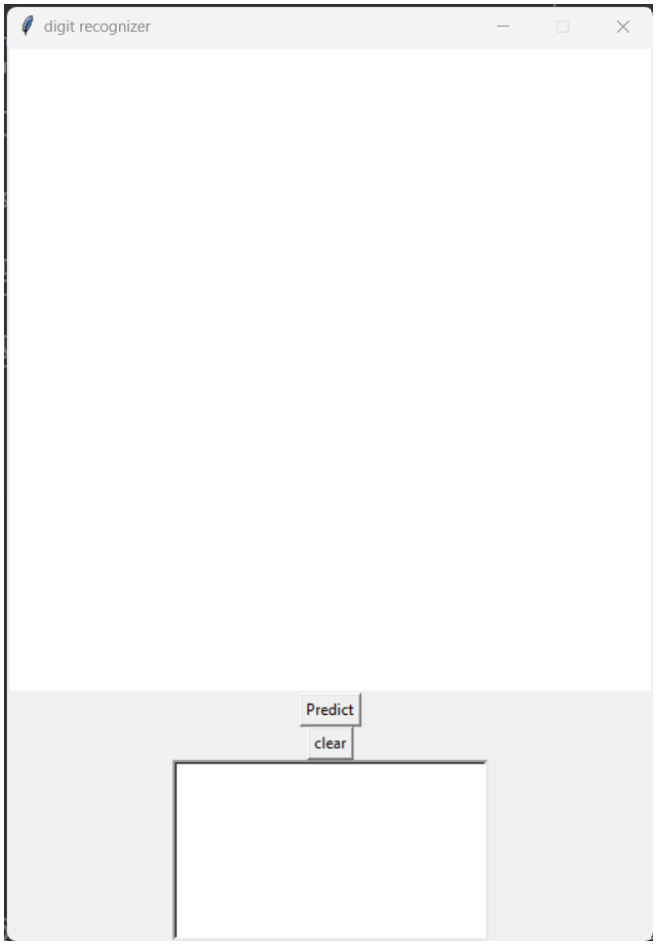

Fig. 7.1.1 canvas

7.2-digit drawing and prediction

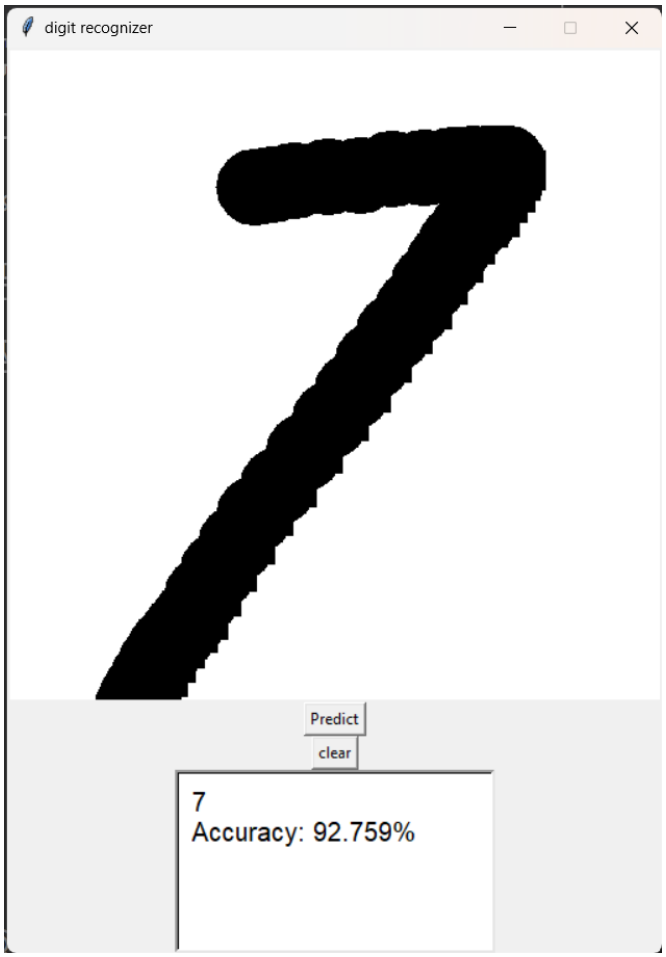

Fig. 7.1.2 digit drawing and prediction

8. SAMPLE CODE

8.1 Code for Application

#tensorflowtesting.py

```
import tensorflow as tf

from keras.models import load_model

import cv2
import numpy as np

classes=[0,1,2,3,4,5,6,7,8,9]

model=tf.keras.models.load_model('bestmodel.h5')
def testing():
    img=cv2.imread('image.png',0)
    img=cv2.bitwise_not(img)
##cv2.imshow('img',img)
    img=cv2.resize(img,(28,28))
    img=img.reshape(1,28,28,1)
    img=img.astype('float32')
    img=img/255.0

    pred=model.predict(img)
    return pred
```

#gui.py

```
from tensorflowTesting import testing
##import tensorflow as tf
from keras.models import load_model
import cv2
import numpy as np
import os

from PIL import ImageTk, Image, ImageDraw
import PIL
import tkinter as tk
from tkinter import *

classes=[0,1,2,3,4,5,6,7,8,9]
width = 500
height = 500
center = height//2
white = (255, 255, 255)
green = (0,128,0)

def paint(event):
    x1, y1 = (event.x - 10), (event.y - 10)
```

```
    x2, y2 = (event.x + 10), (event.y + 10)
    cv.create_oval(x1, y1, x2, y2, fill="black",width=40)
    draw.line([x1, y1, x2, y2],fill="black",width=40)
def model():
    filename = "image.png"
    image1.save(filename)
    pred=testing()
    print('argmax',np.argmax(pred[0]),'\n',
          pred[0][np.argmax(pred[0])],'\n',classes[np.argmax(pred[0])])
    txt.insert(tk.INSERT,"{}\nAccuracy:
{}%".format(classes[np.argmax(pred[0])],round(pred[0][np.argmax(pred[0])]*100,3))
)

def clear():
    cv.delete('all')
    draw.rectangle((0, 0, 500, 500), fill=(255, 255, 255, 0))
    txt.delete('1.0', END)

root = Tk()
##root.geometry('1000x500')

root.resizable(0,0)
cv = Canvas(root, width=width, height=height, bg='white')
cv.pack()

# PIL create an empty image and draw object to draw on
# memory only, not visible
image1 = PIL.Image.new("RGB", (width, height), white)
draw = ImageDraw.Draw(image1)

txt=tk.Text(root,bd=3,exportselection=0,bg='WHITE',font='Helvetica',
            padx=10,pady=10,height=5,width=20)

cv.pack(expand=YES, fill=BOTH)
cv.bind("<B1-Motion>", paint)

##button=Button(text="save",command=save)
btnModel=Button(text="Predict",command=model)
btnClear=Button(text="clear",command=clear)
##button.pack()
btnModel.pack()
btnClear.pack()
txt.pack()
root.title('digit recognizer')
root.mainloop()
```

8.2 Code for training the model

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout

(X_train, y_train), (X_test, y_test)=mnist.load_data()

X_train.shape , y_train.shape , X_test.shape , y_test.shape

def plot_input_img(i):
    plt.imshow(X_train[i] , cmap ='binary')
    plt.title(y_train[i])
    plt.show()

for i in range(10):
    plot_input_img(i)

# pre process the images
X_train= X_train.astype(np.float32)/255
X_test = X_test.astype(np.float32)/255

#reshape/ expanding the dimensions of the image to 28 28 1
X_train = np.expand_dims(X_train, -1)
X_test = np.expand_dims(X_test, -1)

#convert classes to one hot vectors
y_train = tf.keras.utils.to_categorical(y_train)
```

```
y_test = tf.keras.utils.to_categorical(y_test)

model = Sequential()

model.add(Conv2D(32, (3,3), input_shape = (28,28,1), activation = 'relu'))
model.add(MaxPool2D((2,2)))
model.add(Conv2D(64, (3,3), activation = 'relu'))
model.add(MaxPool2D((2,2)))
model.add(Flatten())
model.add(Dropout(0.25))
model.add(Dense(10, activation="softmax"))

model.summary()

model.compile(optimizer = 'adam', loss = keras.losses.categorical_crossentropy , metrics = ['accuracy'] )

#callbacks
from keras.callbacks import EarlyStopping, ModelCheckpoint
#EarlyStopping
es=EarlyStopping(monitor='val_accuracy' ,min_delta=0.01, patience = 4, verbose= 1 )
#model check point
mc=ModelCheckpoint("./bestmodel.h5", monitor= "val_accuracy", verbose= 1, save_best_only= True)
cb = [es,mc]

#model training
his = model.fit(X_train, y_train, epochs= 50, validation_split=  0.3, callbacks = cb)

model_S = keras.models.load_model("D://handwritten//bestmodel.h5")

score = model_S.evaluate(X_test,y_test)
print(f" the model accuracy is {score[1]} ")
```

9. CONCLUSION

Handwritten digit recognition is the first step to the vast field of Artificial Intelligence and Computer Vision. In this project, the Handwritten Digit Recognition using Deep learning methods has been implemented. The performance of CNN for handwritten recognition performed significantly. The results can be made more accurate with more convolution layers and a greater number of hidden neurons. Using Keras as backend and Tensorflow as the software, a CNN model is able to give accuracy of about 98.78%. The loss percentage in both training and evaluation is less than 0.1, which is negligible. The only challenging part is the noise present in the real-world image, which needs to look after. Here we demonstrate a model which can recognize handwritten digit. Later it can be extended for character recognition and real-time person's handwriting.

10. REFERENCES

1. Recognition of Handwritten Digit using Convolutional Neural Network in Python with TensorFlow and Comparison of Performance for Various Hidden Layers by Fathma Siddique, Shadman Sakib, Md. Abu Bakr Siddique in 2019.

2. Recognition of Handwritten Digit using Convolutional Neural Network (CNN) By Md. Anwar Hossain & Md. Mohon Ali in 2019.

3. Handwritten Digit Recognition Using Convolutional Neural Networks by Haider AI-Wzwazy in 2016.

4. Handwritten Digit Recognition using CNN Vijayalaxmi R Rudraswamimath , Bhavanishankar in 2019.

5. Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras by Jason Brownlee in 2016.