A NOVEL METHOD FOR HANDWRITTEN DIGIT RECOGNITION SYSTEM

A PROJECT REPORT

Submitted by

GOWTHAM R	**(612719104025)**
MOHAN P	**(612719104039)**
MURUGAN P	**(612719104042)**
TAMILARASAN T	**(612719104068)**

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

THE KAVERY ENGINEERING COLLEGE

MECHERI, SALEM-636 453.

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2023

BONAFIDE CERTIFICATE

Certified that this project report "**A NOVEL METHOD FOR HANDWRITTEN DIGIT RECOGNITION SYSTEM**" is the bonafide work of **GOWTHAM R** (612719104025), **MOHAN P** (612719104039), **MURUGAN P** (612719104042), **TAMILARASAN T** (612719104068), who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Mr. M. BALAMURUGAN M.E., (Ph.D)	**Mr. M. Umapathy M.E.**
Head of the department	**Supervisor**
Assistant Professor,	Assistant Professor,
Department of Computer Science and Engineering,	Department of Computer Science and Engineering,
The Kavery Engineering College,	The Kavery Engineering College,
Mecheri,	Mecheri,
Salem- 636453.	Salem- 636453.

Submitted for VIVA-VOICE Examination held on _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We wish our heartfelt thanks to our respected Management for the Blessings and constant support over our project period.

We wish to express our sincere thanks to our respected Principal **Dr. V. DURAISAMY M.E., Ph.D, FIE**. For all the blessing and help Provided during the period of project work.

We are in deep gratitude to our department faculty who have always been supporting us through thick and thin respected Project Coordinator and Head of the Department **Mr. M. BALAMURUGAN M.E., (Ph.D)** for the continuous support for the project.

We wish to express our sincere thanks to our respected Project Guide **Mr. M. UMAPATHY, M.E.,** Assistant professor for her constant help and creative ideas to complete this project.

We would like to extend warmest thanks to all our **Department Faculty Members** and **supporting faculties** for helping this project's successful completion unflinching support and encouragement from the members of our Family and Friends. We must thank them all from our depth and heart.

ABSTRACT

Character recognition is crucial in the contemporary world. It can resolve more difficult issues while also simplifying human tasks. Character identification from handwritten is one illustration. This system is used extensively throughout the globe to identify zip codes or postal codes for mail sorting. Handwritten symbols can be recognized using a variety of methods. In this paper, two methods—pattern recognition and convolutional neural networks—are studied. Both techniques are described, and various implementations of each strategy are also covered. Methods for pattern recognition include Bayesian Decision Theory, Nearest Neighbor Rule, and Linear Classification or Discrimination. Neural networks are used for shape identification, Chinese character recognition, and handwritten digit recognition. Use of neural networks for training and identification.

We aim to develop a novel method for handwritten digit recognition using a combination of pattern recognition and convolutional neural networks (CNN). We will explore the effectiveness of these two methods individually and in conjunction with each other, to identify the most efficient and accurate approach. The proposed system will be trained on a large dataset of handwritten digits, and its performance will be evaluated based on various metrics such as accuracy, precision, recall, and F1-score. The ultimate goal is to develop a robust and reliable system that can be used in various applications, including postal mail sorting, bank check processing, and document digitization. The results of this research could potentially contribute to the advancement of character recognition technology and improve the efficiency of various industries that rely on this technology.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The Handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavors. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image. Nevertheless, in these algorithms, appropriate filter size selection, data preparation, limitations in datasets, and noise have not been taken into consideration. As a consequence, most of the algorithms have failed to make a noticeable improvement in classification accuracy. To address the shortcomings of these algorithms, our paper presents the following contributions:

- Firstly, after taking the domain knowledge into consideration, the size of the effective receptive field (ERF) is calculated. Calculating the size of the ERF helps us to select a typical filter size which leads to enhancing the classification accuracy of our CNN.
- Secondly, unnecessary data leads to misleading results and this, in turn, negatively affects classification accuracy. To guarantee the dataset is free from any redundant or irrelevant variables to the target variable, data preparation is applied before implementing the data classification mission.
- Thirdly, to decrease the errors of training and validation, and avoid the limitation of datasets, data augmentation has been proposed.
- Fourthly, to simulate the real-world natural influences that can affect image quality, we propose to add an additive white Gaussian noise with $\sigma = 0.5$ to the MNIST dataset.

As a result, our CNN algorithm achieves state-of-the-art results in handwritten digit recognition, with a recognition accuracy of 99.98%, and 99.40% with 50% noise. A neural network is a model inspired by how the brain works. It consists of multiple layers having many activations, this activation resembles neurons of our brain. A neural network tries to learn a set of parameters in a set of data which could help to recognize the underlying relationships. Neural networks can adapt to changing input, so the network generates the best possible result without needing to redesign the output criteria.

1.2 PURPOSE

Handwritten character recognition is one of the practically important issues in pattern recognition applications. The applications of digit recognition include postal mail sorting, bank check processing, form data entry, etc. The applications of digit recognition include postal mail sorting, bank check processing, form data entry, etc. This paper presents an efficient handwritten digit recognition system based on CONVOLUTIONAL NEURAL NETWORKS(CNN). These features are very simple to implement compared to other methods. Pattern recognition is a data analysis method that uses machine learning algorithms to automatically recognize patterns and regularities in data. Handwritten digit recognition using MNIST dataset is a major project made with the help of Neural Network. It basically detects the scanned images of handwritten digits. We have taken this a step further where our handwritten digit recognition system not only detects scanned images of handwritten digits but also allows writing digits on the screen with the help of an integrated GUI for recognition.

CHAPTER 2

LITERATURE SURVEY

1. HANDWRITTEN CHARACTER RECOGNITION USING NEURAL NETWORK:

AUTHOR: CHIRAG PATEL, RIPAL PATEL, PALAK PATEL.

YEAR: 2011

Description:

Objective of this paper is to recognize the characters in a given scanned document and study the effects of changing the Models of ANN. Today Neural Networks are mostly used for Pattern Recognition tasks. The paper describes the behaviors of different Models of Neural Network used in OCR. OCR is a widespread use of Neural Networks. We have considered parameters like number of Hidden Layers, size of Hidden Layer and epochs. We have used a Multilayer Feed Forward network with Back propagation. In Preprocessing we have applied some basic algorithms for segmentation of characters, normalizing of characters and De-skewing. We have used different Models of Neural Network and applied the test set on cache to find the accuracy of the respective Neural Network.

2. HANDWRITTEN CHARACTER RECOGNITION USING GRADIENT FEATURES:

AUTHOR: ASHUTOSH AGGARWAL, RAJNEESH RANI, RENUDHIR.

YEAR: 2012

Description:

Feature extraction is an integral part of any recognition system. The aim of feature extraction is to describe the pattern by means of a minimum number of features that are effective in discriminating between pattern classes. The gradient measures the magnitude and direction of the greatest change in intensity in a small neighborhood of each pixel. (In what follows, "gradient" refers to both the gradient magnitude and direction). Gradients are computed by means of the Sobel operator.In this paper an effort is made towards recognition of English Characters and obtained recognition accuracy of 94%. Due to its logical simplicity, ease of use and high recognition rate. Gradient Features should be used for recognition purposes.

3. CHARACTER RECOGNITION USING MATLAB'S NEURAL NETWORK TOOLBOX:

AUTHOR: KAULESHWAR PRASAD, DEVVRAT C. NIGAM, ASHMIKA LAKHOTIYA AND DHEEREN UMRE.

YEAR: 2013

Description:

Recognition of Handwritten text has been one of the active and challenging areas of research 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 handwritten document into structural text form. In this paper we focus on recognition of the English alphabet in a given scanned text document with the help of Neural Networks. Using the Matlab Neural Network toolbox, we tried to recognize handwritten characters by projecting them on different sized grids. The first step is image acquisition which acquires the scanned image followed by noise filtering, smoothing and normalization of the scanned image, rendering an image suitable for segmentation where the image is decomposed into sub images. Feature Extraction improves recognition rate and misclassification. We use character extraction and edge detection algorithms for training the neural network to classify and recognize the handwritten characters.

4. NEURAL BASED HANDWRITTEN CHARACTER RECOGNITION:

AUTHOR: HARMANDLU M, MURALI MOHAN K.R,KUMAR H.

YEAR: 1999

Description:

This paper explores the existing ring based method (W.I.Reber, 1987), the new sector based method and the combination of these, termed the Fusion method for the recognition of handwritten English capital letters. The variability associated with the characters is accounted for by way of considering a fixed number of concentric rings in the case of the ring based approach and a fixed number of sectors in the case of the sector approach. Structural features such as end points, junction points and the number of branches are used for the pre classification of characters, the local features such as normalized vector lengths and angles derived from either ring or sector approaches are used in the training using the reference characters and subsequent recognition of the test characters. The recognition rates obtained are encouraging.

5. A FEATURE EXTRACTION TECHNIQUE BASED ON CHARACTER GEOMETRY FOR CHARACTER RECOGNITION:

AUTHOR: DHINESH DHILEEP

YEAR: 2012

Description:

This paper describes a geometry based technique for feature extraction applicable to segmentation-based word recognition systems. The proposed system extracts the geometric features of the character contour. These features are based on the basic line types that form the character skeleton. The system gives a feature vector as its output. The feature vectors generated from a training set were then used to train a pattern recognition engine based on Neural Networks so that the system can be benchmarked.

6. A REVIEW OF GRADIENT-BASED AND EDGE-BASED FEATURE EXTRACTION METHODS FOR OBJECT DETECTION:

AUTHOR: SHENG WANG

YEAR: 2011

Description:

In computer vision research, object detection based on image processing is the task of identifying a designated object on a static image or a sequence of video frames. Projects based on such research works have been widely adapted to various industrial and social applications. The field to which those applications apply includes but not limited to, security surveillance, intelligent transportation system, automated manufacturing, and quality control and supply chain management. In this paper, we are going to review a few most popular computer vision methods based on image processing and pattern recognition. Those methods have been extensively studied in various research papers and their significance to computer vision research has been proven by subsequent research works. In general, we categorize those methods into gradient-based and edge based feature extraction methods, depending on the low level features they use. In this paper, the definitions for gradient and edge extended. Because an image can also be considered as a grid of image patches, it is therefore reasonable to incorporate the concept of granules to gradient for a review.

7. A COMPARISON OF THREE CLASSIFICATION ALGORITHMS FOR THE IDENTIFICATION OF HANDWRITTEN DIGITS:

AUTHOR: MAIWAN B. ABDULRAZZAQ, JWAN NAJEEB SAEED

YEAR: 2019

Description:

Handwritten digits recognition is considered as a core to a diversity of emerging applications. It is used widely by computer vision and machine learning researchers for performing practical applications such as computerized bank check numbers reading. However, executing a computerized system to carry out certain types of duties is not easy and it is a challenging matter. Recognizing the numeral handwriting of a person from another is a hard task because each individual has a unique handwriting way. The selection of the classifiers and the number of features play a vast role in achieving best possible accuracy of classification. This paper presents a comparison of three classification algorithms namely Naive Bayes (NB), Multilayer Perceptron (MLP) and K_Star algorithm based on correlation features selection (CFS) using NIST handwritten dataset. The objective of this comparison is to find out the best classifier among the three ones that can give an acceptable accuracy rate using a minimum number of selected features. The accuracy measurement parameters are used to assess the performance of each classifier individually, which are precision, recall and F-measure. The results show that K_Star algorithm gives better recognition rate than NB and MLPas it reached the accuracy of 82.36%.

8. RECOGNITION OF HANDWRITTEN DIGITS WITH CLASSIFICATION OF DECISION TREE: A MACHINE LEARNING METHOD:

AUTHOR: PRAMOD SEKHARAN NAIR, TSEHAY ADMASSU

YEAR: 2019

Description:

Handwritten digits recognition is an area of machine learning, in which a machine is trained to identify handwritten digits. One method of achieving this is with a decision tree classification model. A decision tree classification is a machine learning approach that uses the predefined labels from the past known sets to determine or predict the classes of the future data sets where the class labels are unknown. In this paper we have used the standard kaggle digits dataset for recognition of handwritten digits using a decision tree classification approach. And we have evaluated the accuracy of the model against each digit from 0 to 9.

9. DEVELOPMENT OF A HIGH PRECISION HANDWRITTEN DIGIT RECOGNITION DETECTOR BASED ON A CONVOLUTION-NEURAL NETWORK:

AUTHOR: SAVITA AHLAWAT, AMIT CHOUDHARY, ANAND NAYYAR, SAURABH SINGH AND BYUNGUN YOON.

YEAR: 2020

Description:

Traditional systems of handwriting recognition have relied on handcrafted features and a large amount of prior knowledge. Training an Optical character recognition (OCR) system based on these prerequisites is a challenging task. Research in the handwriting recognition field is focused around deep learning techniques and has achieved breakthrough performance in the last few years. In addition, we aim to evaluate various SGD optimization algorithms in improving the performance of handwritten digit recognition. A network's recognition accuracy increases by incorporating ensemble architecture. Here, our objective is to achieve comparable accuracy by using a pure CNN architecture without ensemble architecture, as ensemble architectures introduce increased computational cost and high testing complexity. Thus, a CNN architecture is proposed in order to achieve accuracy even better than that of ensemble architectures, along with reduced operational complexity and cost. Moreover, we also present an appropriate combination of learning parameters in designing a CNN that leads us to reach a new absolute record in classifying MNIST handwritten digits. We carried out extensive experiments and achieved a recognition accuracy of 99.87% for a MNIST dataset.

10. MCS HOG FEATURES AND HANDWRITTEN DIGIT RECOGNITION SYSTEM BASED ON SVM.:

AUTHOR: HAMAYUN A. KHAN.

YEAR: 2017

Description:

Digit Recognition is an essential element of the process of scanning and converting documents into electronic format. In this work, a new Multiple-Cell Size (MCS) approach is being proposed for utilizing Histogram of Oriented Gradient (HOG) features and a Support Vector Machine (SVM) based classifier for efficient classification of Handwritten Digits. The HOG based technique is sensitive to the cell size selection used in the relevant feature extraction computations. Hence a new MCS approach has been used to perform HOG analysis and compute the HOG features. The system has been tested on the Benchmark MNIST Digit Database of handwritten digits and a classification accuracy of 99.36% has been achieved using an Independent Test set strategy. A Cross-Validation analysis of the classification system has also been performed using the 10-Fold Cross-Validation strategy and a 10-Fold classification accuracy of 99.26% has been obtained. The classification performance of the proposed system is superior to existing techniques using complex procedures since it has achieved at par or better results using simple operations in both the Feature Space and in the Classifier Space. The plots of the system's Confusion Matrix and the Receiver Operating Characteristics (ROC) show evidence of the superior performance of the proposed new MCS HOG and SVM based digit classification system.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Handwritten digit recognition is the capability of computer applications to recognize the human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different shapes and sizes. The handwritten digit recognition system is a way to tackle this problem which uses the image of a digit and recognizes the digit present in the image. Convolutional Neural Network model created using Tensorflow library over the MNIST dataset to recognize handwritten digits. Handwritten Digit Recognition is the capability of a computer to fete the mortal handwritten integers from different sources like images, papers, touch defenses, etc and classify them into 10 predefined classes (0-9). This has been a Content of bottomless- exploration in the field of deep literacy. Number recognition has numerous operations like number plate recognition, postal correspondence sorting, bank check processing, etc. In Handwritten number recognition, we face numerous challenges. because of different styles of jotting of different peoples is not an optic character recognition. The main existing problem is, the model can not recognize multiple digit numbers. This exploration provides a comprehensive comparison between different machine literacy and deep literacy algorithms for the purpose of handwritten number recognition. For this, we've used Support, Multilayer Perceptron, and Convolutional Neural Network. The comparison between these algorithms is carried out on the basis of their delicacy, crimes, and testing training time corroborated by plots and maps that have been constructed using matplotlib for visualization.

3.2 PROPOSED SYSTEM

Problem Statement:

Mohan is a cashier, he needs a way to quickly enter the account details which are written by account holders in the challan so that account holders don't have to wait long.

Ideation / Solution Description:

Using MNIST dataset and Convolutional Neural Network to perform digits recognition. All of those digits can be converted into electronic words in a text document format, and this data only needs a fraction of the physical storage space of the physical copies.

Novelty / Uniqueness:

Unlike OCR, which recognizes all the characters, it can accurately recognize the digits.

Social Impact / Customer Satisfaction:

This system saves time and workload in sectors which use this technology.

Business Model (Revenue Model):

It is very useful in the banking sectors, In the banking sector, numerical detail in cheque can be easily recognized. Pin code details are easily obtained in the postal system.

Scalability of the Solution:

This system has no restriction on the number of digits to be recognized.

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 SYSTEM REQUIREMENTS

This project needs the help of hardware and software requirements to be fit in the computer or the laptop. The user and the toolkits and hardware and software requirements are required also.

4.1.1 Software Requirements:

- Python (Version - 3.9.10)
- VS Code Editor.
- Web Browser (Chrome, Brave, Edge, Opera etc...)
- Operating system - Microsoft Windows 7 SP 1 or above, Linux or Mac.
- Microsoft Visual Studio 2010.

4.1.2 Hardware Requirements:

- Dual Core and up, 15” Monitor.
- Hard Disk minimum of 40 GB.
- RAM minimum of 2 GB.

4.2 FUNCTIONAL REQUIREMENTS

Image Data:

Handwritten digit recognition refers to a computer's capacity to identify human handwritten digits from a variety of sources, such as photographs, documents, touch screens, etc., and categories them into ten established classifications (0-9). In the realm of deep learning, this has been the subject of countless studies.

Website:

Web hosting makes the code, graphics, and other items that make up a website accessible online. A server hosts every website you've ever visited. The type of hosting determines how much space is allotted to a website on a server. Shared, dedicated, VPS, and reseller hosting are the four basic varieties.

Modified National Institute of Standards and Technology dataset:

The abbreviation MNIST stands for the MNIST dataset. It is a collection of 60,000 tiny square grayscale photographs, each measuring 28 by 28, comprising handwritten single digits between 0 and 9.

Digit Classifier Model:

To train a convolutional network to predict the digit from an image, use the MNIST database of handwritten digits and get the training and validation data first.

Cloud:

The cloud offers a range of IT services, including virtual storage, networking, servers, databases, and applications. In plain English, cloud computing is described as a virtual platform that enables unlimited storage and access to your data over the internet.

4.3 NON-FUNCTIONAL REQUIREMENTS

Usability:

One of the very significant problems in pattern recognition applications is the recognition of handwritten characters. Applications for digit recognition include filling out forms, processing bank checks, and sorting mail.

Security:

The system generates a thorough description of the instantiation parameters, which might reveal information like the writing style, in addition to a categorization of the digit. The generative models are capable of segmentation driven by recognition.

Reliability:

The samples are used by the neural network to automatically deduce rules for reading handwritten digits. Furthermore, the network may learn more about handwriting and hence enhance its accuracy by increasing the quantity of training instances. Numerous techniques and algorithms, such as Deep Learning/CNN, SVM, Gaussian Naive Bayes, KNN, Decision Trees, Random Forests, etc., can be used to recognize handwritten numbers.

Availability:

The features for handwritten digit recognition have been Acquainted. These features are based on shape analysis of the digit image and extract slant or slope information. They are effective in obtaining good recognition of accuracy.

Scalability:

The scalability in the task of handwritten digit recognition, using a classifier, has great importance and it makes use of online handwriting recognition on computer tablets, recognizing zip codes on mail for postal mail sorting, processing bank check amounts, and numeric entries in forms filled up manually.

CHAPTER 5

SYSTEM DESIGN

5.1 SOLUTION & TECHNICAL ARCHITECTURE

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to find the best tech solution to solve existing business problems. Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders. Define features, development phases, and solution requirements. Provide specifications according to which the solution is defined, managed, and delivered.

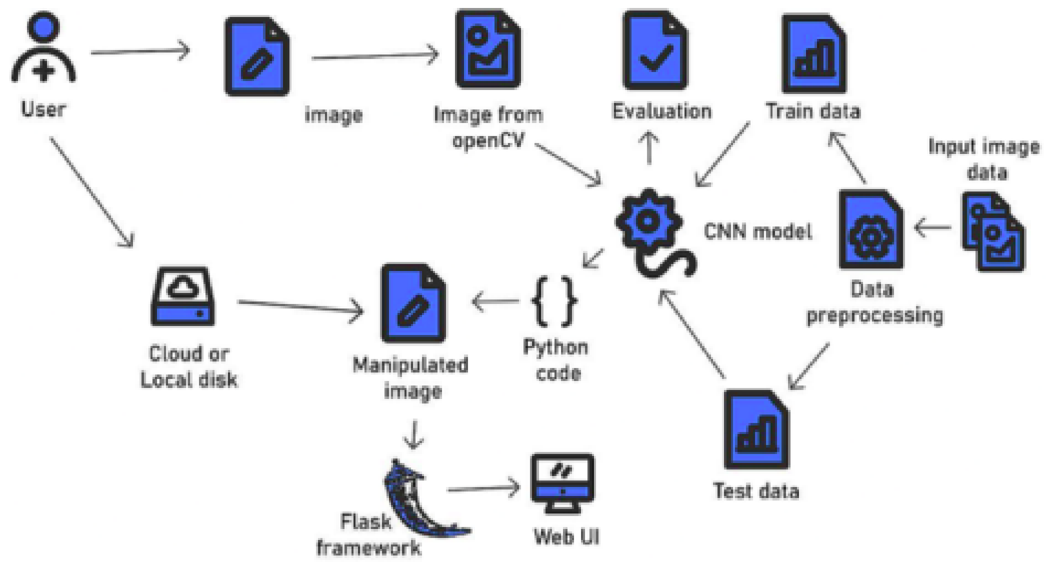


Fig 5.1: Technical Architecture

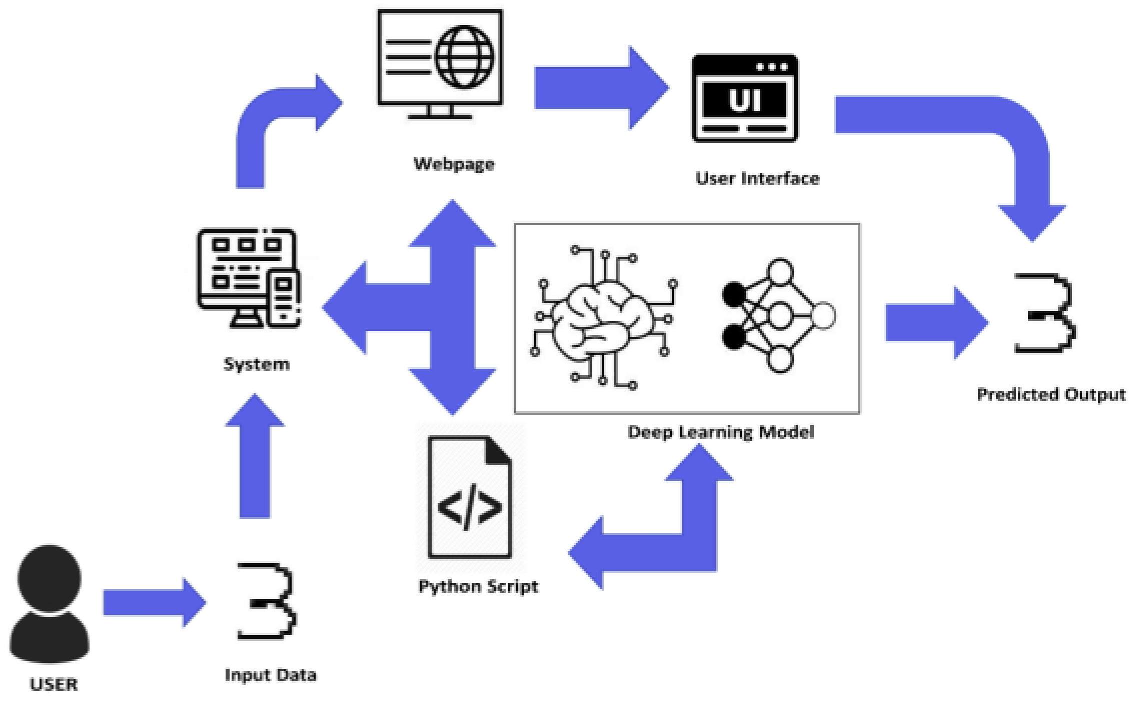


Fig 5.2: Technical Architecture

5.2 DATAFLOW DIAGRAM

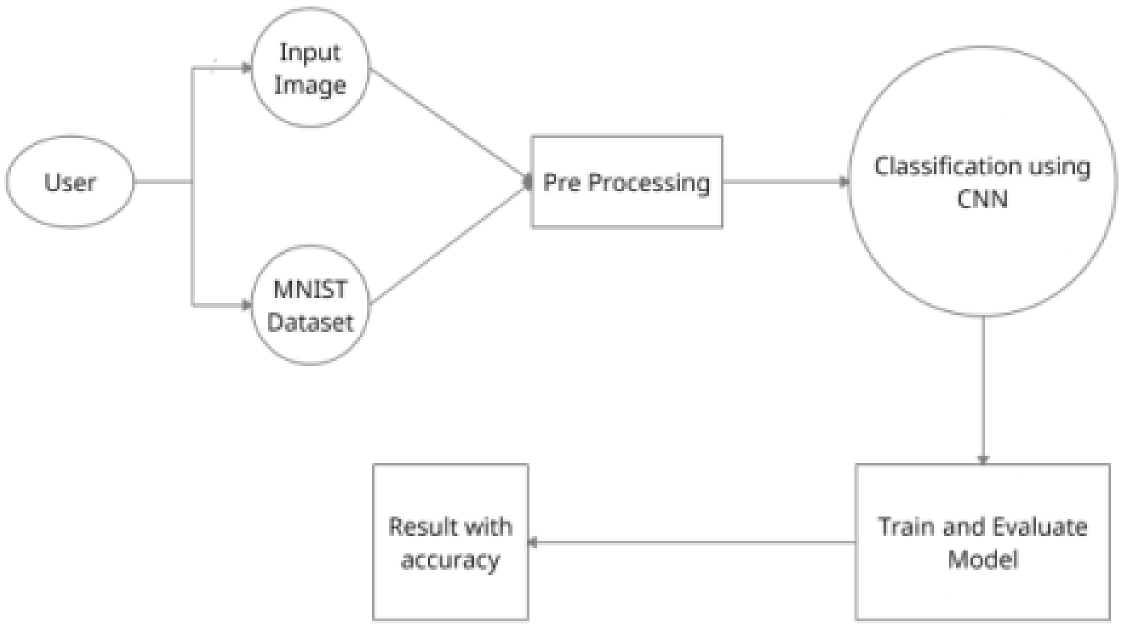


Fig 5.3: Dataflow Diagram

5.3 SEQUENCE DIAGRAM

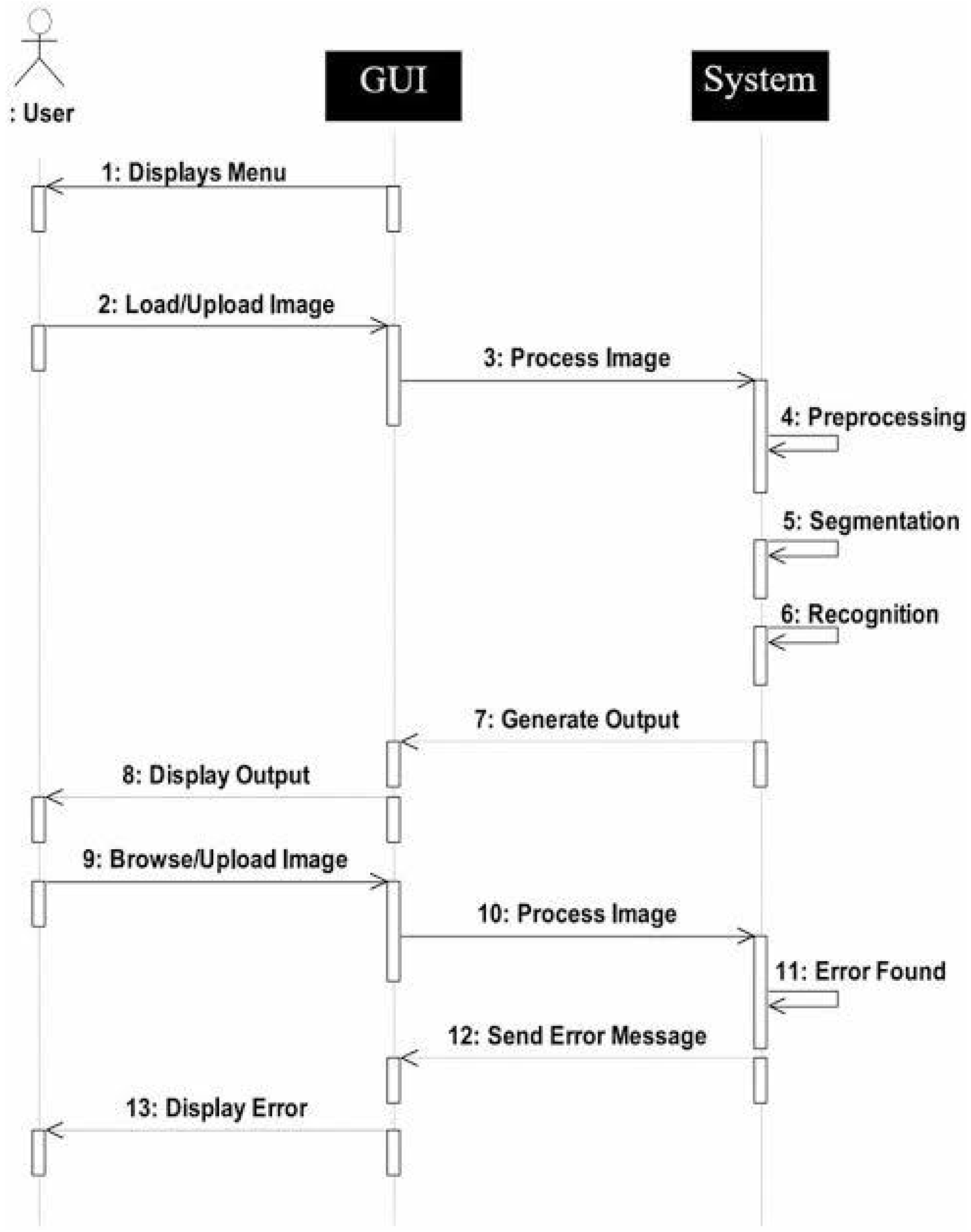


Fig 5.4: Sequence Diagram

5.4 USECASE DIAGRAM

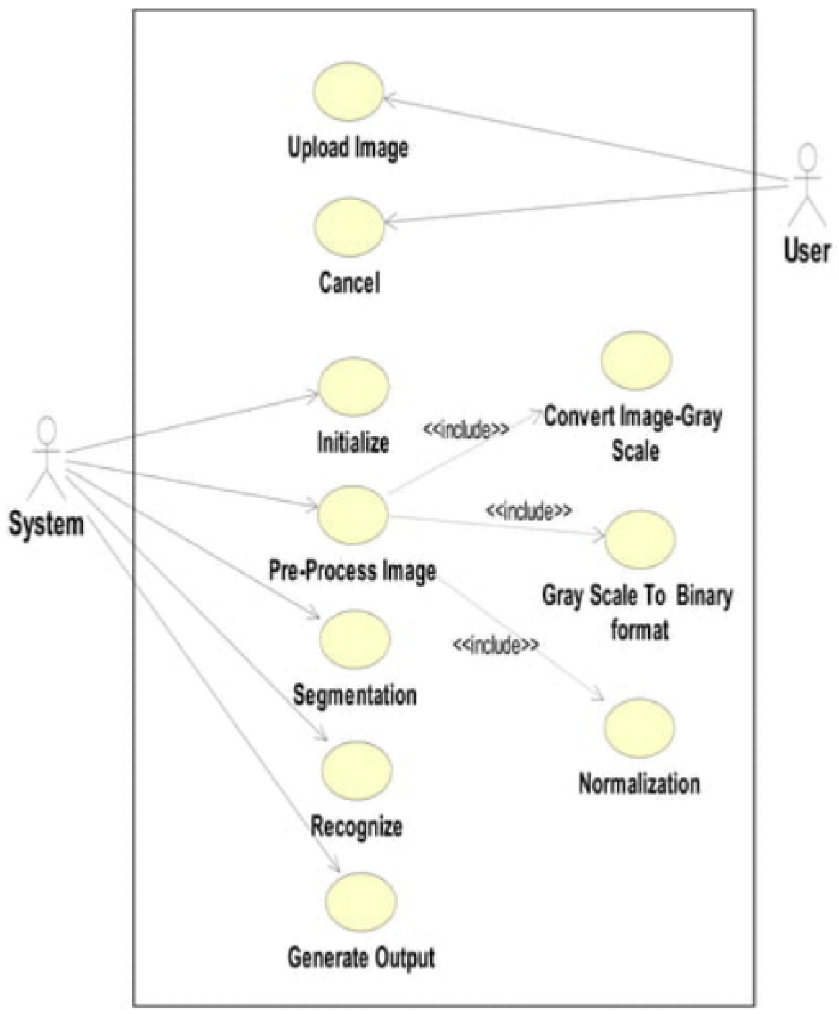


Fig 5.5: Usecase Diagram

CHAPTER 6

SYSTEM MODULES

6.1 LIST OF MODULES

Modules used to develop this "A Novel Method For Handwritten Digit Recognition System" project are,

- TensorFlow
- Keras
- Matplotlib
- NumPy
- SciPy
- OpenCV
- Pillow
- Flask

6.1.1 Tensorflow

TensorFlow is an open-source machine learning framework developed by the Google Brain team. It is designed to enable developers and researchers to build, train, and deploy machine learning models efficiently. TensorFlow is widely used in both industry and academia for a variety of tasks such as image and speech recognition, natural language processing, and reinforcement learning.

One of the key features of TensorFlow is its ability to define and manipulate symbolic graphs. A TensorFlow graph represents a computational graph of mathematical operations, where each node represents an operation and edges represent the data flow between them. This allows for efficient parallel execution on both CPUs and GPUs, making it an ideal platform for training deep learning models.

TensorFlow also provides a high-level API, called Keras, which simplifies the process of building and training neural networks. Keras provides a user-friendly interface for designing, training, and deploying neural networks, and supports a wide range of architectures, including convolutional networks, recurrent networks, and transformers.

Another notable feature of TensorFlow is its support for distributed computing. TensorFlow allows for the training and inference of large-scale machine learning models across multiple devices and machines, which is essential for training complex models that require large amounts of data and computing resources.

Overall, TensorFlow is a powerful and flexible machine learning framework that has become a popular choice for building and deploying machine learning models at scale.

6.1.2 Keras

Keras is a high-level open-source neural network API written in Python. It was developed with a focus on enabling fast experimentation with deep neural networks. Keras was originally developed as a user-friendly interface on top of lower-level deep learning libraries, including TensorFlow, Theano, and CNTK.

One of the key features of Keras is its simplicity and ease of use. It provides a user-friendly interface for designing, training, and deploying neural networks. With Keras, users can easily define a neural network architecture using a simple and intuitive syntax. Keras supports a wide range of neural network architectures, including convolutional networks, recurrent networks, and transformers.

Keras also includes a set of pre-built layers and utilities, which can be used to build complex neural network models quickly and easily. It also includes a range of pre-trained models that can be used for transfer learning.

Another notable feature of Keras is its ability to work seamlessly with TensorFlow, which is a popular deep learning library. Keras can be used as a high-level API on top of TensorFlow, enabling fast and efficient experimentation with deep neural networks.

Keras also supports distributed training, allowing users to train large-scale neural networks across multiple devices and machines.

Overall, Keras is a powerful and user-friendly neural network API that makes it easy to build and deploy deep learning models for a wide range of tasks, including image classification, natural language processing, and speech recognition.

6.1.3 Matplotlib

Matplotlib is a popular open-source data visualization library for the Python programming language. It provides a wide range of tools for creating high-quality plots and charts, making it an essential tool for data scientists and researchers.

Matplotlib provides a range of plotting functions, such as line plots, scatter plots, bar charts, histograms, and many more. It allows for customization of every aspect of the plot, including the axes, labels, colors, and annotations. The library also supports multiple subplots and figure layouts, making it easy to create complex visualizations.

One of the strengths of Matplotlib is its ability to produce publication-quality plots. The library provides a range of customization options, such as font sizes, line styles, and colors, that allow users to produce high-quality plots that are suitable for use in scientific papers and presentations.

Matplotlib also supports a range of file formats for saving plots, including PNG, PDF, SVG, and EPS. Additionally, the library integrates well with other

Python libraries, such as NumPy and Pandas, which makes it easy to create visualizations from data stored in these libraries.

Overall, Matplotlib is a powerful and flexible data visualization library that is widely used by data scientists, researchers, and developers. It provides a range of tools for creating high-quality plots and charts that can be customized to meet specific requirements.

6.1.4 NumPy

NumPy is a Python library for numerical computing that provides support for large, multi-dimensional arrays and matrices, as well as a large collection of mathematical functions to operate on these arrays. NumPy is widely used in scientific computing, data analysis, and machine learning applications.

One of the key features of NumPy is its ability to efficiently store and manipulate large arrays of numerical data. NumPy arrays are implemented in C and provide a fast and efficient way to store and manipulate large datasets. NumPy arrays can be created from Python lists, tuples, or other arrays, and can be of any dimensionality. They are optimized for numerical operations, such as addition, subtraction, multiplication, and division, as well as advanced mathematical operations such as linear algebra and Fourier transforms.

Another important feature of NumPy is its ability to integrate with other Python libraries. NumPy arrays can be easily converted to and from Pandas dataframes, which makes it easy to work with data in both NumPy and Pandas. NumPy arrays can also be used with other popular Python libraries, such as Matplotlib and Scikit-learn, to visualize data and perform machine learning tasks.

NumPy also provides a wide range of mathematical functions for manipulating arrays, including trigonometric functions, exponential and logarithmic functions, statistical functions, and many others. These functions are optimized for fast execution and are essential for many scientific computing and data analysis tasks.

Overall, NumPy is a powerful and versatile library that provides a solid foundation for numerical computing in Python. Its efficient array operations, broad mathematical function support, and integration with other Python libraries make it a must-have tool for any data scientist or scientific computing practitioner.

6.1.5 SciPy

Scipy is an open-source scientific computing library for Python that provides a wide range of tools for scientific and engineering applications. Scipy is built on top of NumPy, which is another popular Python library for numerical computing.

Scipy provides a comprehensive set of functions for optimization, integration, linear algebra, statistics, signal processing, and more. These functions are optimized for speed and accuracy and are implemented using efficient algorithms and techniques from scientific computing.

One of the key features of Scipy is its optimization module, which provides a variety of optimization algorithms for solving unconstrained and constrained optimization problems. These algorithms include gradient-based methods, such as the L-BFGS-B and conjugate gradient methods, as well as non-gradient-based methods, such as the Nelder-Mead and Powell methods.

Scipy also provides a powerful integration module, which includes functions for numerical integration, differential equations, and numerical Fourier transforms. These functions are optimized for speed and accuracy and are particularly useful for scientific applications such as physics, engineering, and biology.

In addition to these core features, Scipy also provides a range of statistical functions for descriptive statistics, hypothesis testing, probability distributions, and more. Scipy also includes functions for signal processing, image processing, and sparse matrix operations.

Overall, Scipy is a powerful and versatile library that provides a wide range of tools for scientific computing and data analysis. Its comprehensive set of functions and efficient algorithms make it a popular choice among scientists, engineers, and researchers who work with numerical data.

6.1.6 OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. It was originally developed by Intel in 1999 and has since become one of the most popular open-source computer vision libraries.

OpenCV provides a vast array of algorithms for processing and analyzing images and videos. It includes a variety of image processing functions, such as filtering, edge detection, thresholding, and morphological operations. It also includes feature detection and extraction functions, such as SIFT (Scale-Invariant Feature Transform) and SURF (Speeded-Up Robust Features), which are used for object recognition and tracking.

OpenCV supports multiple programming languages, including C++, Python, Java, and MATLAB, making it accessible to a wide range of developers and researchers. It also supports multiple platforms, including Windows, Linux, macOS, Android, and iOS.

In addition to its image processing capabilities, OpenCV provides tools for machine learning and computer vision applications. It includes algorithms for classification, clustering, regression, and deep learning, as well as tools for camera calibration, stereo vision, and 3D reconstruction.

Overall, OpenCV is a powerful and versatile library for computer vision and image processing, widely used in both academic and industrial applications, including robotics, automotive, healthcare, and entertainment industries.

6.1.7 Pillow

Pillow is a Python library for processing images. It provides a set of image processing operations such as reading and writing images in various formats, resizing and cropping images, applying filters, adjusting colors and contrast, and much more. Pillow is a fork of the Python Imaging Library (PIL) and is backward-compatible with PIL.

Pillow makes it easy to work with image data in Python, whether it is a single image or a large collection of images. It supports many image file formats, including JPEG, PNG, BMP, GIF, and TIFF. Pillow provides a consistent API for manipulating these image formats, which makes it easier for developers to write image processing code that works across different file formats.

Pillow is widely used in many different domains, including web development, data science, and computer vision. It is particularly useful for tasks that involve working with large amounts of image data, such as training machine learning models that use images as input.

Overall, Pillow is a powerful and flexible library for processing images in Python. It provides a wide range of image processing operations and supports many different image file formats, making it an essential tool for many Python developers.

6.1.8 Flask

Flask is a lightweight, open-source Python web framework that allows developers to easily build web applications. It is popular for its simplicity, flexibility, and scalability, making it an ideal choice for small to medium-sized web applications.

Flask provides a simple and intuitive interface for building web applications. It comes with a built-in development server, which makes it easy to test and debug applications locally. Flask is also extensible, allowing developers to add new features and functionality through third-party libraries.

One of the key features of Flask is its routing system. Flask allows developers to define URL routes for their application, which map to specific Python functions that handle the corresponding requests. This makes it easy to build RESTful APIs and web applications that respond to different HTTP requests.

Flask also supports the use of templates, which allow developers to create dynamic web pages that can be customized based on user input or other dynamic factors. Flask supports several templating engines, including Jinja2, which is a powerful and flexible templating engine that allows for complex rendering of HTML, XML, and other markup languages.

Another notable feature of Flask is its support for extensions. Flask extensions are third-party libraries that provide additional functionality to the framework. Some popular Flask extensions include Flask-WTF for form handling, Flask-SQLAlchemy for database integration, and Flask-Login for user authentication.

Overall, Flask is a lightweight and flexible Python web framework that provides an easy-to-use interface for building web applications. It is popular among developers who value simplicity, flexibility, and scalability in their web development projects.

CHAPTER 7
SOFTWARE TESTING

7.1 TEST CASES

A test case is a document, which has a set of test data, preconditions, expected results and postconditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

Test Case acts as the starting point for the test execution, and after applying a set of input values, the application has a definitive outcome and leaves the system at some end point or also known as execution postcondition.

A test case is a basic concept in software testing, but there are similar terms that might cause confusion for beginners or individuals less familiar with quality assurance.

Test case vs **Test Scenario**. As the name implies, a test scenario describes a situation or functionality that requires testing. For example, a test scenario might be, "Verify login functionality." Test scenarios typically have their own ID numbers for tracking. QA teams often derive test cases (low-level actions) from test scenarios (high-level actions); and test scenarios typically come from software and business requirements documentation. The below table shows the test cases, test scenarios and test status of this project,

Test Case ID	Feature Type	Component	Test Scenario	Expected Result	Actual Result	Status
HP_TC_001	UI	Home Page	Verify UI elements in the Home Page	The Home page must be	Working as expected	PASS

				displayed properly		
HP_TC_ 002	UI	Home Page	Check if the UI elements are displayed properly in different screen sizes	The Home page must be displayed properly in all sizes	The UI is not displayed properly in mobile screens	FAIL
HP_TC_ 003	Functio nal	Home Page	Check if user can upload their file	The input image should be uploaded to the applicatio n successful ly	Working as expected	PASS
HP_TC_ 004	Functio nal	Home Page	Check if user cannot upload unsupported files	The applicati0 n should not allow user to select a non image file	User is able to upload only JPEG, JPG, PNG, and SVG	PASS
/HP_TC _005	Functio nal	Home Page	Check if the page redirects to the result	The page should redirect to	Working as expected	PASS

			page once the input is given	the results page		
BE_TC_001	Functional	Backend	Check if all the routes are working properly	All the routes should properly work	Working as expected	PASS
M_TC_001	Functional	Model	Check if the model can handle various image sizes	The model should rescale the image and predict the results	Working as expected	PASS
M_TC_002	Functional	Model	Check if the model predicts the digit	The model should predict the number	Working as expected	PASS
M_TC_003	Functional	Model	Check if the model can handle complex input image	The model should predict the number in the complex image	The model fails to identify the digit since the model is not built to handle such data	FAIL

RP_TC_ 001	UI	Result Page	Verify UI elements in the Result Page	The Result page must be displayed properly	Working as expected	PASS
RP_TC_ 002	UI	Result Page	Check if the input image is displayed properly	The input image should be displayed properly	Working as expected	PASS
RP_TC_ 003	UI	Result Page	Check if the result is displayed properly	The result should be displayed properly	Working as expected	PASS
RP_TC_ 004	UI	Result Page	Check if the other predictions are displayed properly	The other prediction s should be displayed properly	Working as expected	PASS

7.2 UNIT TESTING

UNIT TESTING is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

In SDLC, STLC, V Model, Unit testing is the first level of testing done before integration testing. Unit testing is a White Box testing technique that is usually performed by the developer. During the testing, the program to be tested was performed with certain defined test scenarios and the output of the program for the test cases was evaluated to determine whether the program is performing as expected. In case of any error the irregularities were detected and corrected for optimal functioning by employing following testing approaches and phases and correction was recorded for future references. Though, in a practical world due to time crunch or reluctance of developers to test, QA engineers also do unit testing.

Unit testing focuses verification effort on the unit of software design. Employing the unit test paradigms, developed in the development design phase of the system design as a guide, significant control conduits are validated for uncovering or unleashing errors in the developed system models. The developed interfaces of individual modules were tested to ensure proper flow of the information into and out of the modules under consideration. Boundary conditions have been verified properly. The every composing and autonomous paths have been executed for assuring that all statements in the module can be effectively executed minimum once and all error-handling approaches were taken into consideration. The individual unit has been tested in depth under varying parameters so as to ensure that the developed system would not have to face any failure with varied implementation conditions.

The system model testing has been accomplished while developing individual models and ultimately the individual unit was verified to be functional as per expectations in varied functional circumstances. In our research or project work we have prepared a few individual functions that are further integrated to accomplish overall research objectives.

7.3 INTEGRATION TESTING

INTEGRATION TESTING is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.

Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as 'I & T' (Integration and Testing), 'String Testing' and sometimes 'Thread Testing'. The aim of integration testing is to test the interfaces between the modules and expose any defects that may arise when these components are integrated and need to interact with each other.

Integration testing is the second level of the software testing process after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Unit testing uses modules for testing purposes, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.

Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as integration testing.

Integration testing is conducted after unit testing, where the functional correctness of the smallest piece of code, or unit, is tested. Each unit can be logically isolated in the software. The smaller the unit, the more granular insights unit testing can reveal.

The main difference between unit testing and integration testing is that in unit testing, individual modules are tested. In integration testing, these modules are combined and tested as a single unit to check the functionality of the overall application.

7.4 SYSTEM TESTING

SYSTEM TESTING is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

After the integration testing, the software was entirely combined as a single unit system model; the errors caused due to interfaces have been unleashed and the raised errors have been corrected in the final series of software tests, validation tests begin. Validation test proceeds when the software functions in such a manner which can be expected by the user or customer. In this testing the system was tested according to the system requirement specification. System testing was in fact a succession of diverse tests whose principal rationale was to fully work out the computer-based system. Though, the individual test has a diverse principle all work to bear out that all comprising the system.

After the integration testing, the software was entirely combined as a single unit system model; the errors caused due to interfaces have been unleashed and the raised errors have been corrected in the final series of software tests, validation tests begin. Validation test proceeds when the software functions in such a manner which can be expected by the user or customer. In this testing the system was tested according to the system requirement specification. System testing was in fact

a succession of diverse tests whose principal rationale was to fully work out the computer-based system. Though, the individual test has a diverse principle all work to bear out that all comprising the system.

There are mainly two widely used methods for software testing, one is White box testing which uses internal coding to design test cases and another is black box testing which uses GUI or user perspective to develop test cases.

- White Box Testing.
- Black Box Testing.

System testing falls under Black box testing as it includes testing of the external working of the software. Testing follows the user's perspective to identify minor defects.

7.5 USER ACCEPTANCE TESTING

USER ACCEPTANCE TESTING is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. Acceptance Testing is done in the final phase of testing after functional, integration and system testing are done.

The main Purpose of Acceptance Testing is to validate end to end business flow. It does not focus on cosmetic errors, spelling mistakes or system testing. Acceptance Testing is carried out in a separate testing environment with production-like data setup. It is a kind of black box testing where two or more end-users will be involved.

Acceptance testing is formal testing based on user requirements and function processing. It determines whether the software is conforming specified requirements and user requirements or not. It is conducted as a kind of Black Box testing where the number of required users involves testing the acceptance level of the system. It is the fourth and last level of software testing.

User acceptance testing (UAT) is a type of testing, which is done by the customer before accepting the final product. Generally, UAT is done by the customer (domain expert) for their satisfaction, and checks whether the application is working according to given business scenarios, real-time scenarios.

In this, we concentrate only on those features and scenarios which are regularly used by the customer or mostly user scenarios for the business or those scenarios which are used daily by the end-user or the customer.

However, the software has passed through three testing levels (Unit Testing, Integration Testing, System Testing) But still there are some minor errors which can be identified when the system is used by the end user in the actual scenario. Acceptance testing is the squeezing of all the testing processes that have been done previously.

Once the application is bug-free, we hand it to the customer, no customer accepts the application blindly before using it. Hence, they do one round of testing for their satisfaction, which is known as user acceptance testing.

7.5.1 Defect Analysis

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Total
By Design	1	0	1	0	2
Duplicate	0	0	0	0	0
External	0	0	2	0	2
Fixed	4	1	0	1	6
Not Reproduced	0	0	0	1	1
Skipped	0	0	0	1	1
Won’t fix	1	0	1	0	2
Total	6	1	4	3	14

7.5.2 Test Case Analysis

Section	Total Cases	Not Tested	Fail	Pass
Client Application	10	0	3	7
Security	2	0	1	1
Performance	3	0	1	2
Exception Reporting	2	0	0	2

CHAPTER 8

CODING & SOLUTIONING

Requirements.txt

1. Flask==2.2.2
2. keras==2.10.0
3. matplotlib==3.5.3
4. numpy==1.21.6
5. opencv-python==4.6.0.66
6. Pillow==9.3.0
7. scipy==1.7.3
8. tensorflow==2.10.0

Create_model.py

```
# importing required libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPool2D
from keras import optimizers
from keras.datasets import mnist
from keras.utils import to_categorical
import keras

# loading mist hand written dataset
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

# Applying threshold for removing noise
_, X_train_th = cv2.threshold(X_train,127,255,cv2.THRESH_BINARY)
_, X_test_th = cv2.threshold(X_test,127,255,cv2.THRESH_BINARY)

# Reshaping
X_train = X_train_th.reshape(-1,28,28,1)
X_test = X_test_th.reshape(-1,28,28,1)

# Creating categorical output from 0 to 9
y_train = to_categorical(y_train, num_classes = 10)
y_test = to_categorical(y_test, num_classes = 10)

# cross checking shape of input and output
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

# Creating CNN model
input_shape = (28,28,1)
number_of_classes = 10

# Applying CNN layers
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,
3),activation='relu',input_shape=input_shape))
```

```
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))
model.add(Dense(number_of_classes, activation='softmax'))

# compiling the model
model.compile(loss=keras.losses.categorical_crossentropy,
          optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

model.summary()

# fitting the model
history = model.fit(X_train, y_train,epochs=10, shuffle=True,
               batch_size = 200,validation_data= (X_test, y_test))

# saving the model
model.save("models\digit_classifier.h5")
```

Templates

Index.html

```
<!-- HOME PAGE -->
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
shrink-to-fit=no">
    <title>Home - HandWritten Digit Recognition System</title>
    <!-- Bootstrap CDN - CSS -->
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/css/bootstrap.min.css"

integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iu
XoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
    <style>
        .container {
            height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        .display-4 {
            font-weight: 350;
        }

        .lead {
            /* font-weight: normal; */
            text-align: justify;
        }
    </style>
</head>

<body>

    <div class="container">
        <div class="jumbotron">
            <!-- Project Name -->
            <h1 class="display-4">
                A Novel Method for Handwritten Digit Recognition System
            </h1>
            <!-- Project Description -->
            <p class="lead">
                Handwritten digit recognition system is a technology that is much needed
in this world as of today. This
                system is used to recognize the digits from the images that we provide as
a input. Before proper
                implementation of this technology we have relied on writing digits with
our own hands which can result
                in errors. It is difficult to store and access physical data with efficiency.
This project presents
                recognizing the handwritten digits from the famous MNIST dataset.
Here, we will be using artificial
                neural network / convolutional neural network.
            </p>
            <!-- Routes to recognize page -->
            <a href="/recognize" class="btn btn-primary btn-lg">Recognize</a>
        </div>
    </div>

    <!-- Bootstrap CDN - JavaScript -->
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRv
H+8abtTE1Pi6jizo"
        crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"

integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiS
njAK/l8WvCWPIPm49"
        crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/js/bootstrap.min.js"

integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMi
Z6OW/JmZQ5stwEULTy"
        crossorigin="anonymous"></script>
</body>
</html>
```

Recognize.html

```
<!-- RECOGNIZE PAGE -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
shrink-to-fit=no">
    <title>Recognize - HandWritten Digit Recognition System</title>
    <!-- Bootstrap CDN - CSS -->
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/css/bootstrap.min.css"

integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iu
XoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
    <style>
        .container {
            height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
        }
        .display-4 {
            font-weight: 350;
        }
        .lead {
            /* font-weight: normal; */
            text-align: justify;
        }
    </style>
</head>
<body>

    <!-- Back Button -->
    <nav class="navbar navbar-light bg-light">
        <a class="navbar-brand" href="/" style="font-size: 23px;">
          <img src={{back_url}} width="30" height="30" alt="">
        </a>
      </nav>

    <div class="container">
        <div class="jumbotron">
            <h1 class="display-4">Handwritten digit recognizer </h1>
            <p class="lead">This is the page where you can get the result by uploading
image from your local storage.
            </p>

            <!-- Gets image input -->
            <form method="POST" enctype="multipart/form-data">
                <input type="file" name="file" class="file" required>
                <button class="btn btn-success active">Recognize</button>
            </form>
            <br>
          <!-- Displays result -->
          {% if display_output %}
            <img src="{{uploaded_img_path}}" class="rounded mx-auto d-block"
height="175" width="175">
            <br>
            <div class="alert alert-{{alert_box_color}} alert-dismissible fade show"
role="alert">
              <strong>{{display_output}}</strong>
              <button type="button" class="close" data-dismiss="alert"
aria-label="Close">
                <span aria-hidden="true">&times;</span>
              </button>
            </div>
          {% endif %}
        </div>
      </div>

      <!-- Bootstrap CDN - JavaScript -->
      <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"

integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRv
H+8abtTE1Pi6jizo"
        crossorigin="anonymous"></script>
      <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"
```

```
integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiS
njAK/l8WvCWPIPm49"
        crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/js/bootstrap.min.js"

integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMi
Z6OW/JmZQ5stwEULTy"
        crossorigin="anonymous"></script>
</body>
</html>
```

Recognizer.py

```
# importing required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage
import math
from keras.models import load_model

# loading pre trained model
model = load_model("models/digit_classifier.h5")

# predicting the digit
def predict_digit(img):
    test_image = img.reshape(-1, 28, 28, 1)
    return np.argmax(model.predict(test_image))

# refining each digit
def image_refiner(gray):
    org_size = 22
    img_size = 28
    rows, cols = gray.shape

    if rows > cols:
        factor = org_size / rows
        rows = org_size
        cols = int(round(cols * factor))
    else:
        factor = org_size / cols
        cols = org_size
        rows = int(round(rows * factor))
    gray = cv2.resize(gray, (cols, rows))

    # get padding
    colsPadding = (
        int(math.ceil((img_size - cols) / 2.0)),
        int(math.floor((img_size - cols) / 2.0)),
    )
    rowsPadding = (
        int(math.ceil((img_size - rows) / 2.0)),
        int(math.floor((img_size - rows) / 2.0)),
    )

    # apply padding
    gray = np.lib.pad(gray, (rowsPadding, colsPadding), "constant")
    return gray

# getting the predicted image
def get_output(path):
    img = cv2.imread(path, 2)
    img_org = cv2.imread(path)

    ret, thresh = cv2.threshold(img, 127, 255, 0)
    contours, hierarchy = cv2.findContours(
        thresh, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE
    )

    predicted_values = []

    for j, cnt in enumerate(contours):
        epsilon = 0.01 * cv2.arcLength(cnt, True)
        approx = cv2.approxPolyDP(cnt, epsilon, True)

        hull = cv2.convexHull(cnt)
        k = cv2.isContourConvex(cnt)
        x, y, w, h = cv2.boundingRect(cnt)

        if hierarchy[0][j][3] != -1 and w > 10 and h > 10:
            # cropping each image and process
            roi = img[y : y + h, x : x + w]
            roi = cv2.bitwise_not(roi)
            roi = image_refiner(roi)
            th, fnl = cv2.threshold(roi, 127, 255, cv2.THRESH_BINARY)

            # getting prediction of cropped image
            pred = predict_digit(roi)
            predicted_values.append(pred)

    return predicted_values

if __name__ == "__main__":
    recognized_digits = get_output(path="static/images/1.png")
    print(recognized_digits)
```

App.py

```
# importing required libraries
from flask import Flask, render_template, url_for, request
from werkzeug.utils import secure_filename
import os
from recognizer import get_output

# instantiating the flask application
app = Flask(__name__)

# allowing only specified image files
```

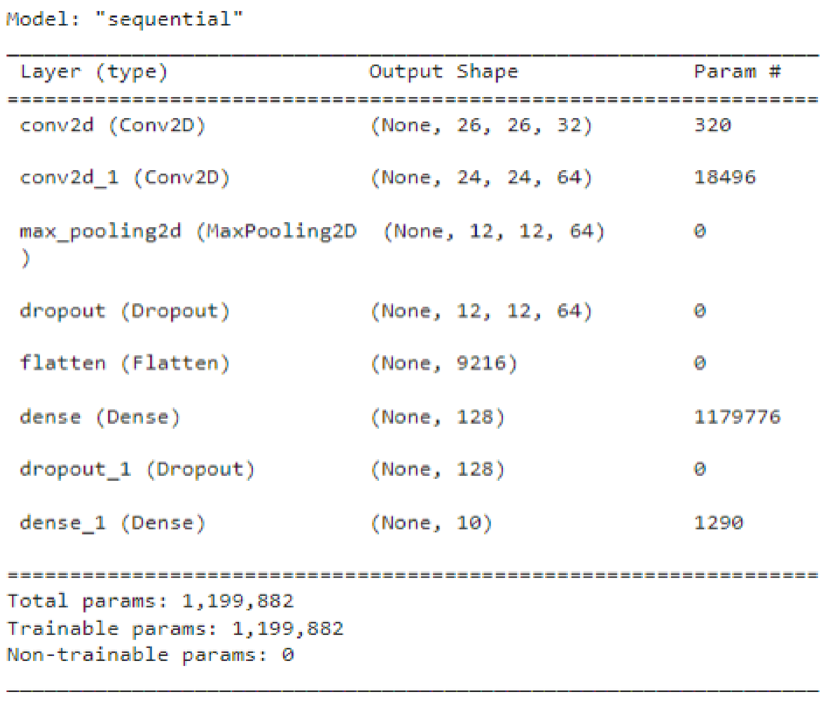
```
ALLOWED_EXTENSIONS = ["jpg", "jpeg", "png", "svg"]
# folder path to store the uploaded image
UPLOAD_FOLDER = "static/images/"

# configuring application
app.secret_key = os.urandom(24)
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

def is_allowed_file(filename: str) -> bool:
    """
    Checking the upload file is valid or not
    """
    return ("." in filename) and (
        filename.rsplit(".", 1)[1].lower() in ALLOWED_EXTENSIONS
    )

# routes to home page
@app.route("/")
def index():
    return render_template("index.html")

# routes to recognize page
@app.route("/recognize", methods=["GET", "POST"])
def recognize():
    alert_box_color = "dark"
    display_output = None
    uploaded_img_path = None
    back_url = url_for("static", filename="icons/back.svg")

    if request.method == "POST":
        file = request.files["file"]
        if file and is_allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config["UPLOAD_FOLDER"], filename))

            image_url = url_for("static", filename=f"images/{file.filename}")
            output = get_output(path=image_url[1:])

            if len(output) > 1:
                output = list(map(str, output))
                display_output = f"Recognized digits are {', '.join(output)}"
                uploaded_img_path = image_url[1:]
            else:
                display_output = f"Recognized digit is {output[0]}"
                uploaded_img_path = image_url[1:]

        else:
            alert_box_color = "danger"
            display_output = f"Choosen file format is not supported. Choose only {',
'.join(ALLOWED_EXTENSIONS)} images."

    return render_template(
        "recognize.html",
        back_url=back_url,
        alert_box_color=alert_box_color,
        display_output=display_output,
        uploaded_img_path=uploaded_img_path,
    )

if __name__ == "__main__":
    app.run(debug=True)
```

CHAPTER 9

RESULTS

9.1 PERFORMANCE METRICS

9.1.1 Model Summary



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2D  (None, 12, 12, 64)       0
 )

 dropout (Dropout)           (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

9.1.2 Accuracy

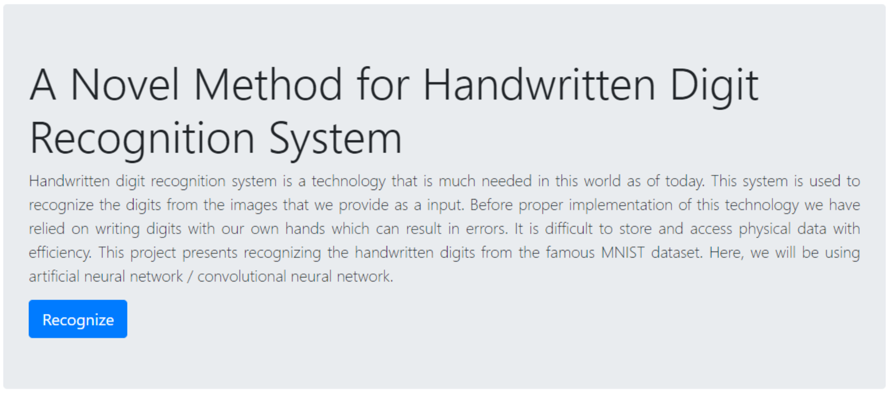
Accuracy	0.9861000180244446
Loss	0.04915023967623710 6

9.1.3 Confusion Matrix

0	0	1	2	3	4	5	6	7	8	9
0	975	0	1	0	0	0	2	0	1	1
1	0	1138	1	0	0	0	0	2	0	0
2	1	1	1022	0	1	0	0	5	2	0
3	0	0	0	1006	0	3	0	0	0	1
4	0	0	0	0	980	0	0	0	1	1
5	0	0	0	2	0	889	1	0	0	0
6	2	2	0	0	1	1	952	0	0	0
7	0	2	1	1	0	0	0	1021	1	2
8	3	0	2	1	0	2	1	2	962	1
9	1	2	0	0	10	5	0	7	3	981

9.2 OUTPUT SCREENSHOTS

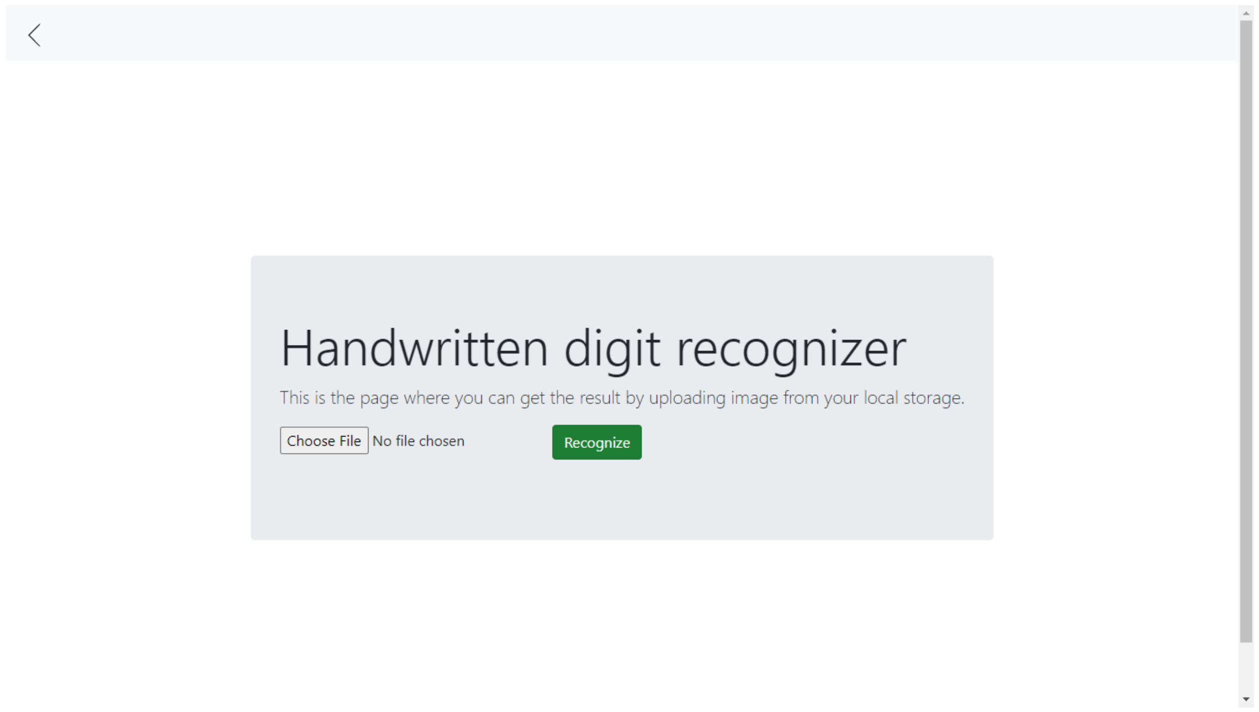
Home Page



A Novel Method for Handwritten Digit Recognition System

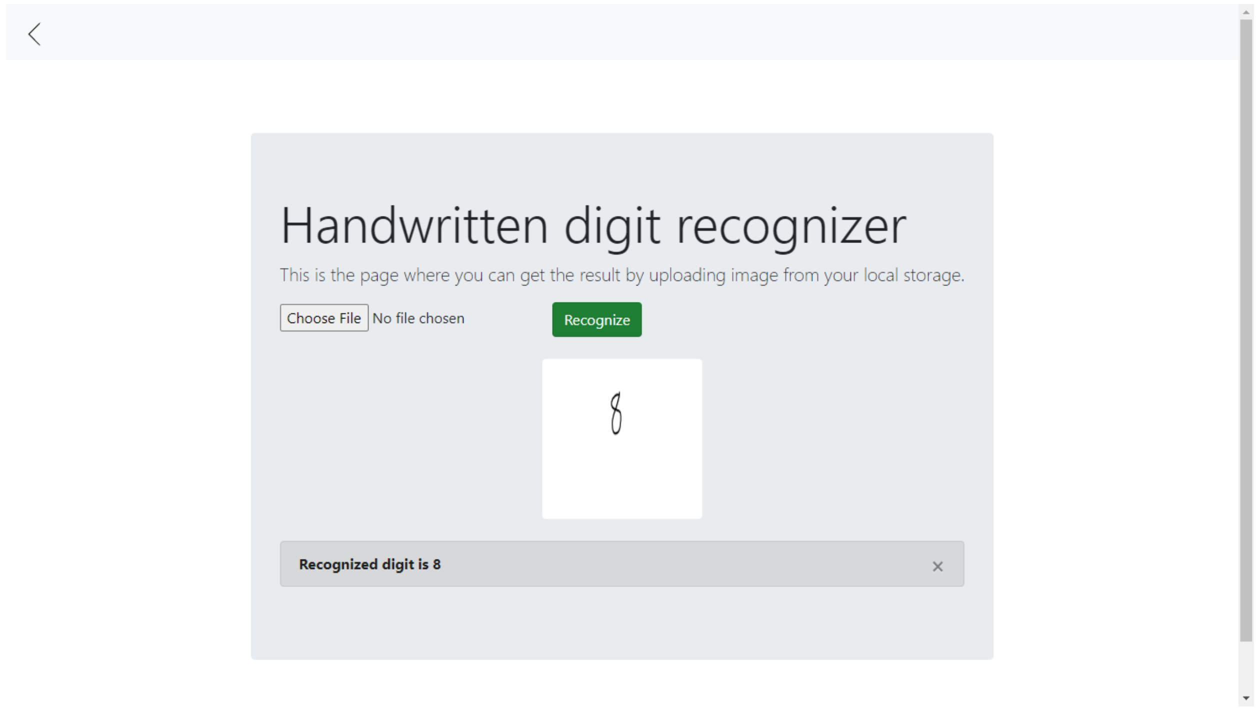
Handwritten digit recognition system is a technology that is much needed in this world as of today. This system is used to recognize the digits from the images that we provide as a input. Before proper implementation of this technology we have relied on writing digits with our own hands which can result in errors. It is difficult to store and access physical data with efficiency. This project presents recognizing the handwritten digits from the famous MNIST dataset. Here, we will be using artificial neural network / convolutional neural network.

Recognize Page


Handwritten digit recognizer

This is the page where you can get the result by uploading image from your local storage.

Recognize Page After Uploading an Image


Handwritten digit recognizer

This is the page where you can get the result by uploading image from your local storage.

Recognized digit is 8

CHAPTER 10

ADVANTAGES & DISADVANTAGES

10.1 ADVANTAGES

The main motivation behind using convolutional layers is that it is typically true of images that pixels in close proximity are more related with each other than with pixels that are a greater distance away. Thus, compared to fully connected layers, convolutional layers give a better indication of general features that appear in an image by taking advantage of this spatial structure of images.

10.1.1 Shift Invariance:

- A major shortcoming of fully interconnected networks is their dependence on position of a feature in an image. Such a network would recognize an image, but not its slightly shifted self. Training shift invariance in a fully connected network is possible, and involves extensive expansion of training data, but it's significantly more efficient to use convolution, which naturally has this property.
- Convolutional layers detect a given feature, regardless of its position on an image. Because the MNIST data set is centered and normalized, a fully connected network can still work, but a network with convolutional layers is able to handle data that is not properly centered or normalized.

10.1.2 Computationally Efficient:

- Another consequence of using convolutional networks is that there are fewer parameters involved, making the network more computationally efficient to train.

- For any given neuron in a fully connected hidden layer, there is a weight and a bias associated with each neuron in the previous layer, and as such, the number of parameters scales as the number of neurons squared, assuming a similar number of neurons in each interconnected layer.
- Electronic data storage.
- More organized files.
- Easier data retrieval.
- Power full high-level feature.

10.2 DISADVANTAGES:

- Not always accurate.
- Spacing of letter of words.
- Different languages.
- May lack robustness when writing variations are large.
- Requires large amounts of training data.
- Does not model temporal variations very well.

CHAPTER 11

CONCLUSION

In this paper, the Handwritten Digit Recognition using Deep learning methods has been implemented. The most widely used Machine learning algorithms, KNN, SVM, RFC and CNN have been trained and tested on the same data in order to acquire the comparison between the classifiers.

Utilizing these deep learning techniques, a high amount of accuracy can be obtained. Compared to other research methods, this method focuses on which classifier works better by improving the accuracy of classification models by more than 99%. Using Keras as backend and Tensorflow as the software, a CNN model is able to give accuracy of about 98.72%. In this initial experiment, CNN gives an accuracy of 98.72%, while KNN gives an accuracy of 96.67%, while RFC and SVM are not that outstanding.

CHAPTER 12

FUTURE SCOPE

This project is far from complete and there is a lot of room for improvement. Some of the improvements that can be made to this project are as follows:

- Add support to detect digits from multiple images and save the results.
- Improve model to detect digits from complex images.
- Add support to different languages to help users from all over the world.

This project has endless potential and can always be enhanced to become better. Implementing this concept in the real world will benefit several industries and reduce the workload on many workers, enhancing overall work efficiency.

CHAPTER 13

REFERENCES

1. Pranit Patil, Bhupinder Kaur, “Handwritten Digit Recognition Using Various Machine Learning Algorithms and Models”, International Journal of Innovative Research in Computer Science & Technology (IJIRCST), Volume-8, Issue-4, July-2020.
2. Rohini. M, Dr. D. Surendran, “A NOVEL METHOD FOR HAND WRITTEN DIGIT RECOGNITION USING DEEP LEARNING”, INTERNATIONAL JOURNAL OF CURRENT ENGINEERING AND SCIENTIFIC RESEARCH (IJCESR), Volume-6, Issue-6, 2019.
3. Ayesha Siddiqa, Chakrapani. D. S, “A Recognition System for Handwritten Digits Using CNN”, International Journal of Science and Research (IJSR), Volume-10, Issue-10, October-2021.
4. Ritik Dixit, Rishika Kushwah, Samay Pashine, “Handwritten Digit Recognition using Machine and Deep Learning Algorithms”, arXiv:2106.12614v1 [cs.CV] 23 June 2021.
5. Wang, Y., Wang, R., Li, D. et al. Improved Handwritten Digit Recognition using Quantum K-Nearest Neighbor Algorithm. Int J Theor Phys 58, 2331–2340 (2019).
6. M. B. Abdulrazzaq and J. N. Saeed, "A Comparison of Three Classification Algorithms for Handwritten Digit Recognition," 2019 International Conference on Advanced Science and Engineering (ICOASE), Zakho - Duhok, Iraq, 2019, pp. 58-63, doi: 10.1109/ICOASE.2019.8723702.
7. Assegie, Tsehay & Nair, Pramod. (2019), "Handwritten digits recognition with decision tree classification: a machine learning approach", International Journal of Electrical and Computer Engineering (IJECE). 9. 4446.

10.11591/ijece.v9i5.pp4446-4451. K. Elissa, “Title of paper if known,” unpublished.

8. D. Ge, X. Yao, W. Xiang, X. Wen and E. Liu, "Design of High Accuracy Detector for MNIST Handwritten Digit Recognition Based on Convolutional Neural Network," 2019 12th International Conference on Intelligent Computation Technology and Automation (ICICTA), Xiangtan, China, 2019, pp. 658-662, doi: 10.1109/ICICTA49267.2019.00145.
9. Al-Wzwazy, Haider. (2016), "Handwritten Digit Recognition Using Convolutional Neural Networks", International Journal of Innovative Research in Computer and Communication Engineering. 4.
10. Hafiz, Abdul & Bhat, Ghulam. (2020), "Reinforcement Learning Based Handwritten Digit Recognition with Two-State Q-Learning", 2007.01193.
11. Khan, H. (2017) MCS HOG Features and SVM BasedHandwritten Digit Recognition System. Journal of Intelligent Learning Systems and Applications, 9, 21-33.
12. M. Y. W. Teow, "Understanding convolutional neural networks using a minimal model for handwritten digit recognition," 2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems (I2CACIS), Kota Kinabalu, 2017, pp. 167-172, doi: 10.1109/I2CACIS.2017.8239052.
13. Alsaafin, A. and Elnagar, A. (2017) A Minimal Subset Of Features Using Feature Selection for Handwritten Digit Recognition. Journal of Intelligent Learning Systems and Applications, 9, 55-68.
14. Wu S., Wei W., Zhang L. (2018) Comparison of Machine Learning Algorithms for Handwritten Digit Recognition. In: Li K., Li W., Chen Z., Liu Y. (eds) Computational Intelligence and Intelligent Systems. ISICA

2017. Communications in Computer and Information Science, vol 874. Springer, Singapore.

15. R. Jantayev and Y. Amirgaliyev, "Improved Handwritten Digit Recognition method using Deep Learning Algorithm," 2019 15th International Conference on Electronics, Computer and Computation (ICECCO), Abuja, Nigeria, 2019, pp. 1-4, doi: 10.1109/ICECCO48375.2019.9043235.