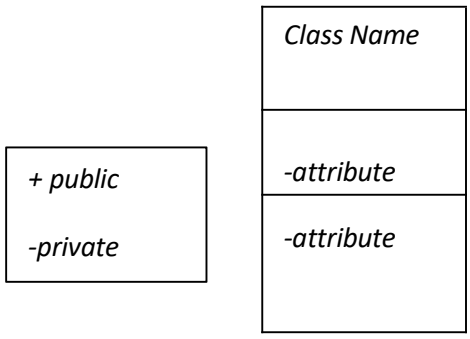
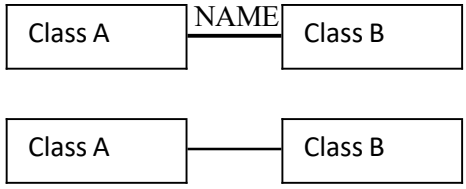
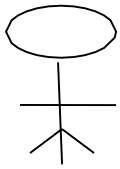
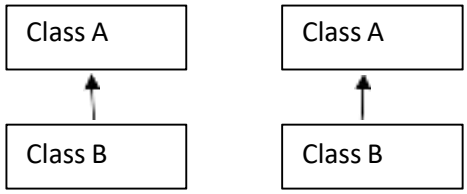
TABLE OF CONTENTS

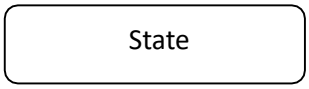
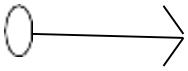
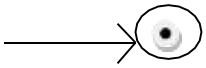
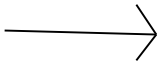
	1. GENERAL 1. ACTIVITY DIAGRAM 2. USE CASE DIAGRAM 3. DATA FLOW DIAGRAM 4. SEQUENCE DIAGRAM 5. COLLABORATION DIAGRAM 6. CLASS DIAGRAM 7. SYSTEM ARCHITECTURE 8. OBJECT DIAGRAM 9. STATE DIAGRAM 10. COMPONENT DIAGRAM 11. E-R DIAGRAM 4.2 DATABASE DESIGN (ALL LEVEL	
5.	**CHAPTER 5 :SOFTWARE SPECIFICATION** 5.1 GENERAL	
6.	**CHAPTER 6 :IMPLEMENTATION** 1. GENERAL 2. IMPLEMENTATION 3. DATA BASE TABLE STRUCTURE	

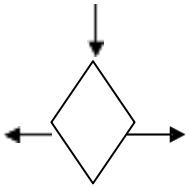
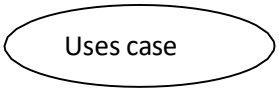
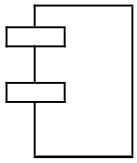
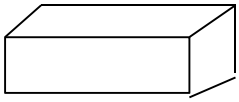
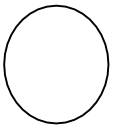
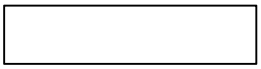
7.	**CHAPTER 7 :SNAPSHOTS** 1. GENERAL 2. VARIOUS SNAPSHOTS	
8.	**CHAPTER 8 :SOFTWARE TESTING** 1. GENERAL 2. DEVELOPING METHODOLOGIES 3. TYPES OF TESTING	
9.	**CHAPTER 9 :** **APPLICATIONS AND FUTURE ENHANCEMENT** 1. GENERAL	

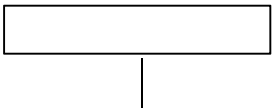
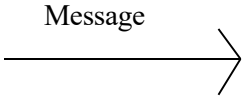
LIST OF FIGURES

LIST OF SYSMBOLS

S.NO	NOTATION NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association		Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several classes into a single classes.
4.	Aggregation		Interaction between the system and external environment

5.	Relation (uses)	uses	Used for additional process communication.
6.	Relation (extends)		Extends relationship is used when one use case is similar to another use case but does a bit more.
7.	Communication		Communication between various use cases.
8.	State		State of the processes.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.

12.	Decision box		Represents decision making process from a constraint
13.	Use case		Interact ion between the system and external environment.
14.	Component		Represents physical modules which are a collection of components.
15.	Node		Represents physical modules which are a collection of components.
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard,sensors,etc.

18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.
20.	Message		Represents the message exchanged.

ABSTRACT

Handwritten digit recognition is a fundamental problem in pattern classification, widely explored in the field of deep learning. While single-digit recognition using Convolutional Neural Networks (CNN) has demonstrated high accuracy, recognizing multi-digit handwritten sequences remains a challenging task due to variations in handwriting, spacing, and alignment. This paper presents a deep learning-based approach for multi-digit handwritten recognition using CNN, aiming to enhance accuracy and robustness in real-world applications.

The proposed model is trained on datasets such as MNIST and SVHN, effectively classifying sequences of digits by leveraging deep feature extraction and sequence modeling techniques. The system processes input images by segmenting and recognizing multiple digits simultaneously using convolutional layers for feature extraction, recurrent layers for sequence learning, and fully connected layers for final classification. Techniques such as dropout regularization, batch normalization, and data augmentation are applied to improve generalization and prevent overfitting.

Experimental results demonstrate that the CNN-based model outperforms traditional neural networks and machine learning techniques in terms of accuracy, speed, and computational efficiency. The implementation is carried out using TensorFlow and Keras, with Python libraries such as NumPy and Pandas for data processing. This research contributes to real-world applications, including postal services, banking transactions, license plate recognition, and automated document processing, where accurate multi-digit recognition is essential. The proposed approach achieves promising results, demonstrating the potential of deep learning in solving complex handwritten recognition tasks.

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The issue of manually written numerals acknowledgment has been broadly concentrated lately and a huge number of pre-processing strategies and arrangement calculations have been created. Notwithstanding, transcribed numerals acknowledgment is as yet a test for us. The primary difficulty of transcribed numerals acknowledgment lies in the genuine variation in size, translation, stroke thickness, rotation, and distortion of numeral images, as handwritten digits are written by different users, each with a unique writing style. Several researchers have employed various approaches to handwritten digit recognition using different AI techniques. Khotanzad et al. (1998), for instance, applied the concepts of Machine Learning and Neural Networks to recognize and determine handwritten digits from images. This investigation has demonstrated that digit recognition is an excellent model problem for learning about neural networks and offers a great foundation for developing more advanced techniques such as deep learning.

Handwritten recognition (HWR) refers to the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, touchscreen inputs, and other devices. The image of the written text may be captured from a sheet of paper using optical scanning (optical character recognition), intelligent word recognition, or through direct user input. Alternatively, the movements of the pen tip may be captured "online," for example by a pen-based computer screen surface, which is generally an easier task as more clues are available. This paper presents a system for recognizing **multiple handwritten digits** (0 to 9) from the renowned MNIST dataset using the TensorFlow framework (library), Python as the programming language, and its related libraries. As the user inputs a sequence of handwritten digits, the machine recognizes and displays the output along with the accuracy rate.

1.2 OBJECTIVE

This paper explores the performance of Convolutional Neural Networks (CNNs). Results demonstrate that CNN classifiers outperform traditional Neural Networks with significantly improved computational efficiency, without compromising on accuracy. Handwritten digit recognition can be effectively performed using CNNs, a deep learning approach within the broader field of Machine Learning. The foundation of this project is built using the MNIST (Modified National Institute of Standards and Technology) database, integrated with a CNN architecture to recognize **multiple handwritten digits**. This forms the core structure of the updated system, enabling it to process and predict digit sequences rather than a single digit input.

To implement this model, essential libraries such as NumPy, Pandas, TensorFlow, and Keras are required. These libraries provide the building blocks for data handling, model creation, training, and prediction, forming the backbone of the project. The MNIST dataset consists of about 70,000 images of handwritten digits ranging from 0 to 9, making it a 10-class classification problem. This dataset is divided into two parts: training and test sets. Each image is represented as a 28×28 matrix where each cell contains a grayscale pixel value. By processing sequences of such images or a connected series of digits, the model is now capable of recognizing **multiple digits** from a single input.

1.2.1 SCOPE OF THE PROJECT

Then again, the movements of the pen tip may be detected "online," for instance by a pen-based computer screen surface, which is generally an easier task as more clues are available. This paper presents a system for recognizing **multiple handwritten digits** (0 to 9) from the renowned MNIST dataset using the TensorFlow framework (library) and Python as the programming language along with its supporting libraries. As the user inputs a sequence of handwritten digits, the system recognizes and displays the corresponding output along with the accuracy rate.

1.3 Existing System

In Existing System, There have been several works on adversarial attacks against ML based systems in other domains. Adversarial example in the domain of computer vision has been widely studied. Gradient Descent method and Genetic Programming (GP) were used for evading PDF malware classifier. GAN-based methods were used to attack ML-based malware detection and real-time video classification. PSO based methods were used to attack speech recognition, text classification and object detection. Lever aged stochastic optimization method to evade text sentiment analysis.

1. Existing System Disadvantages

- Less accuracy.
- Slow for large datasets.
- Must be Scaling of data absolute.

1.3.2 LITERATURE SURVEY:

TITLE **:**Detecting iris liveness with batch normalized convolutional neural network

AUTHOR **:**Long, M.; Yan, Z.

YEAR **:** 2019

DESCRIPTION

Aim to countermeasure the presentation attack for iris recognition system, an iris liveness detection scheme based on batch normalized convolutional neural network (BNCNN) is proposed to improve the reliability of the iris authentication system. The BNCNN architecture with eighteen layers is constructed to detect the genuine iris and fake iris, including convolutional layer, batch-normalized (BN) layer, Relu layer, pooling layer and full connected layer. The iris image is first pre-processed by iris segmentation and is normalized to 256×256 pixels, and then the iris features are extracted by BNCNN. With these features, the genuine iris and fake iris are determined by the decision-making layer. Batch normalization technique is used in BNCNN to avoid the problem of over fitting and gradient disappearing during training. Extensive experiments are conducted on three classical databases: the CASIA Iris Lamp database, the CASIA Iris Syn database and Ndcontact database. The results show that the proposed method can effectively extract micro texture features of the iris, and achieve higher detection accuracy compared with some typical iris liveness detection methods.

TITLE **:**Offline continuous handwriting recognition using sequence to sequence neural networks. Neurocomputing

AUTHOR **:**Sueiras, J.; Ruiz, V.; Sanchez, A.; Velez, J.F.

YEAR **:** 2018

DESCRIPTION

This paper proposes the use of a new neural network architecture that combines a deep convolutional neural network with an encoder–decoder, called sequence to sequence, to solve the problem of recognizing isolated handwritten words. The proposed architecture aims to identify the characters and contextualize them with their neighbors to recognize any given word. Our model proposes a novel way to extract relevant visual features from a word image. It combines the use of a horizontal sliding window, to extract image patches, and the application of the LeNet-5 convolutional architecture to identify the characters. Extracted features are modeled using a sequence-to-sequence architecture to encode the visual characteristics and then to decode the sequence of characters in the handwritten text image. We test the proposed model on two handwritten databases (IAM and RIMES) under several experiments to determine the optimal parameterization of the model. Competitive results above those presented in the current state-of-the-art, on handwriting models, are achieved. Without using any language model and with closed dictionary, we obtain a word error rate in the test set of 12.7% in IAM and 6.6% in RIMES.

TITLE : Offline Handwritten Digits Recognition Using Machine learning.

AUTHOR : Wells, Lee & Chen, Shengfeng&Almamlook, Rabia&Gu, Yuwen.

YEAR : 2018

DESCRIPTION

The problem of handwritten digit recognition has long been an open problem in the field of pattern classification. Several studied have shown that Neural Network has a great performance in data classification. The main objective of this paper is to provide efficient and reliable techniques for recognition of handwritten numerals by comparing various existing classification models. This paper compares the performance of five machine learning classifier models namely Neural Network, K-Nearest Neighbor (K-NN), Random Forest, Decision Tree and Bagging with gradient boost. Results indicate that K-NN classifier outperform Neural Network with significant improved computational efficiency without sacrificing performance. They both outperformed the other classifiers: Random Forest, Decision Tree and Bagging with gradient boost. We also discovered that as the training data is increasing the accuracy of the classifier is also improved. The result of this paper shows that K-NN has equally high accuracy of 96.7% compared to Neural Network of 96.8%, but K-NN achieves a processing speed with almost 10 times faster. The analysis presented in this paper suggests that the K-NN combined with preprocessing methods is capable of achieving great performance apart from Neural Network when used as a classification algorithm in offline handwritten digit recognition.

TITLE **:** Improving handwriting-based gender classification using ensemble classifiers. Expert Systems with Applications

AUTHOR **:** Ahmed, M., Rasool, A. G., Afzal, H., & Siddiqi

YEAR **:** 2017

DESCRIPTION

This paper presents a system to predict gender of individuals from offline handwriting samples. The technique relies on extracting a set of textural features from handwriting samples of male and female writers and training multiple classifiers to learn to discriminate between the two gender classes. The features include local binary patterns (LBP), histogram of oriented gradients (HOG), statistics computed from gray-level co-occurrence matrices (GLCM) and features extracted through segmentation-based fractal texture analysis (SFTA). For classification, we employ artificial neural networks (ANN), support vector machine (SVM), nearest neighbor classifier (NN), decision trees (DT) and random forests (RF). Classifiers are then combined using bagging, voting and stacking techniques to enhance the overall system performance. The realized classification rates are significantly better than those of the state-of-the-art systems on this problem validating the ideas put forward in this study.

TITLE : Amulti-scale deep quad tree-based feature extraction method for the recognition of isolated handwritten characters of popular Indic scripts. Pattern Recognition

AUTHOR : Sarkhel, R., Das, N., Das, A., Kundu, M., &Nasipuri, M.

YEAR : 2017

DESCRIPTION

Recognition of handwritten characters is a challenging task. Variations in writing styles from one person to another, as well as for a single individual from time to time, make this task harder. Hence, identifying the local invariant patterns of a handwritten character or digit is very difficult. These challenges can be overcome by exploiting various script specific characteristics and training the OCR system based on these special traits. Finding ubiquitous invariant patterns and peculiarities, applicable for handwritten characters or digits of multiple scripts, is much more difficult. In the present work, a non-explicit feature-based approach, more specifically, a multi-column multi-scale convolutional neural network (MMCNN) based architecture has been proposed for this purpose. A deep quad-tree based staggered prediction model has been proposed for faster character recognition. These denote the most significant contributions of the present work. The proposed methodology has been tested on 9 publicly available datasets of isolated handwritten characters or digits of Indic scripts. Promising results have been achieved by the proposed system for all of the datasets. A comparative analysis has also been performed against some of the contemporary OCR systems to prove the superiority of the proposed system. We have also evaluated our system on MNIST dataset and achieved a maximum recognition accuracy of 99.74%, without any data augmentation to the original dataset.

1.4 Proposed System

In the proposed system, this paper introduces a method designed to recognize **multiple handwritten digits** without relying on external segmentation techniques. While the system effectively identifies sequences of digits from images, it currently operates in an offline mode. However, this limitation can be addressed by integrating real-time input capabilities, such as capturing user-drawn digits through touchscreens or canvas-based input on web interfaces.

Another potential enhancement is improving accuracy on more complex digit combinations, such as overlapping or tightly spaced digits. This challenge can be addressed by incorporating advanced preprocessing techniques or by using sequence-aware models. Despite these limitations, the current system demonstrates robust performance on standard datasets like MNIST, and offers a strong foundation for further development into real-time, high-accuracy digit recognition applications.

1. Proposed System Advantages

- Multi-digit Recognition Capability.
- High Accuracy with CNN.
- User-Friendly Interface with Flask.
- Robust Dataset Utilization.

CHAPTER 2

PROJECT DESCRIPTION

2.1 GENERAL:

This project is a web-based handwritten digit recognition system designed to identify **multiple digits** from user input using deep learning techniques. Built using Python, TensorFlow, and Flask, the system employs a Convolutional Neural Network (CNN) trained on the MNIST dataset to accurately predict sequences of handwritten digits.

Users can either draw digits directly onto an interface or upload digit-containing images. The model then processes the input image, extracts the digit patterns, and predicts the complete sequence with high accuracy. The use of CNN allows the system to handle variations in handwriting style, size, and orientation effectively.

The core objective of this project is to provide an easy-to-use, reliable solution for digit recognition, which can be extended to various applications such as automated form processing, number plate recognition, banking systems, and educational tools.

1.2METHODOLOGIES

MODULE:

- **Dataset**
- **Importing the necessary libraries**
- **Retrieving the images**
- **Splitting the dataset**
- **Building the model**
- **Compile Model**
- **Fit Model**

- **Apply the model and plot the graphs for accuracy and loss**
- **Accuracy on test set**
- **Saving the Trained Model**

MODULES DESCSRIPTION:

1. Dataset:

In this module, the input dataset is prepared for training and testing purposes. The system uses the **MNIST dataset**, which contains **70,000 images** of handwritten digits (0–9), each of size **28x28 pixels**. The dataset is structured such that each row represents an image, flattened into a (1, 784) shape. Each image is labeled from 0 to 9.

In the **updated version**, the system is designed to recognize **multiple digits within a single image**, rather than classifying only one digit at a time.

2. Importing the necessary libraries:

We use **Python** along with key libraries:

- **TensorFlow/Keras** – for building and training the neural network.
- **NumPy and Pandas** – for data manipulation.
- **sklearn** – for splitting the dataset.
- **PIL (Pillow)** – to handle image conversions.
- **Matplotlib** – for plotting accuracy and loss graphs.

3. Retrieving the images:

Images and their corresponding labels are retrieved and resized to 28x28 pixels. For multi-digit recognition, additional preprocessing may be applied to segment or prepare images for recognizing digit **sequences** instead of isolated characters. The images are then converted into NumPy arrays for model compatibility.

4. Splitting the dataset:

The dataset is divided into **80% training** and **20% testing**. This ensures the model learns from a large portion of the data while still having unseen data to validate accuracy.

Convolutional Neural Networks

The objectives behind the first module of the course 4 are:

- To understand the convolution operation
- To understand the pooling operation
- Remembering the vocabulary used in convolutional neural networks (padding, stride, filter, etc.)
- Building a convolutional neural network for multi-class classification in images

Computer Vision

Some of the computer vision problems which we will be solving in this article are:

1. Image classification
2. Object detection
3. Neural style transfer

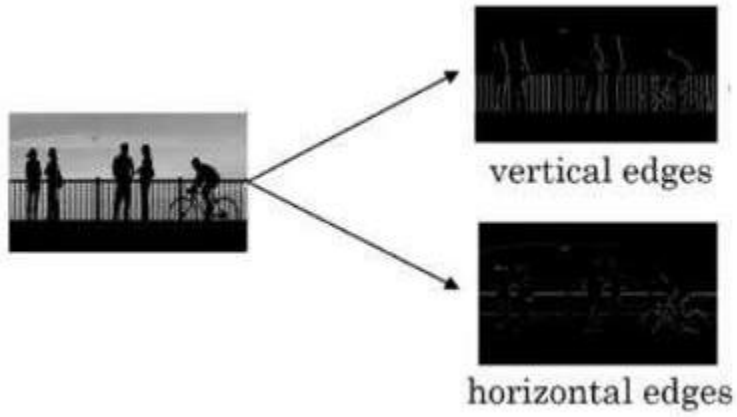
One major problem with computer vision problems is that the input data can get really big. Suppose an image is of the size 68 X 68 X 3. The input feature dimension then becomes 12,288. This will be even bigger if we have larger images (say, of size 720 X 720 X 3). Now, if we pass such a big input to a neural network, the number of parameters will swell up to a HUGE number (depending on the number of hidden layers and hidden units). This will result in more computational and memory requirements – not something most of us can deal with.

Edge Detection Example

In the previous article, we saw that the early layers of a neural network detect edges from an image. Deeper layers might be able to detect the cause of the objects and even more deeper layers might detect the cause of complete objects (like a person's face).

In this section, we will focus on how the edges can be detected from an image. Suppose we are given the below image

As you can see, there are many vertical and horizontal edges in the image. The first thing to do is to detect these edges:



5. Building the model:

A Convolutional Neural Network (CNN) is used due to its high efficiency in image-based tasks. The model uses:

- Conv2D layers (with 32 filters and kernel size of 5x5)
- MaxPooling2D (2x2)
- Dropout layers (dropout rate = 0.25)
  This structure is repeated with variations, followed by Flatten, Dense, and additional Dropout and Dense layers.
  In the previous version, the model classified single digits. In the updated system, it processes multi-digit sequences, so the output layer can be adjusted accordingly (e.g., using a time-distributed dense layer or sequence model architecture depending on implementation).

In Keras, models are defined as sequence of layers. We first initialize a 'Sequential Model' and then we add the layers with respective neurons in them. The following code does the same.

The model, as expected, takes 28 x 28 pixels (we flatten out the image and pass each of the pixel in a 1-D vector) as an input. The output of model has to be a decision on one of the letters, so we set the output

layer with 26 neurons (the decision is made in probabilities).Then we will add the layers to make convolutional neural network. In the first 2 Conv2D layers we have used 32 filters and the kernel size is (5,5). In the MaxPool2D layer we have kept pool size (2,2) which means it will select the maximum value of every 2 x 2 area of the image. By doing this dimension of the image will reduce by factor of 2. In dropout layer we have kept dropout rate = 0.25 that means 25% of neurons are removed randomly. We apply these 3 layers again with some change in parameters. Then we apply flatten layer to convert 2-D data to 1-D vector. This layer is followed by dense layer, dropout layer and dense layer again. The last dense layer outputs 47 nodes as the traffic signs are divided into 47 categories in our dataset. This layer uses the softmax activation function which gives probability value and predicts which of the 47 options has the highest probability.

6. Compile Model:

Now that the model is defined, we can compile it. Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow. Here, we specify some properties needed to train the network. By training, we are trying find the best set of weights to make the decision on the input. We must specify the loss function to use to evaluate a set of weights, the optimizer used to search through different weights for the network and any optional metrics we would like to collect and report during training.

7. Fit Model:

Here, we train the model using a model check pointer, which will help us save the best model (best in terms of the metric we defined in the previous step).

8. Apply the model and plot the graphs for accuracy and loss:

We will compile the model and apply it using fit function.Then we will plot the graphs for accuracy and loss. We got average validation accuracy of 88.6% and average training accuracy of 89.3%.

9. Accuracy on test set:

The final model achieves an **accuracy of 88.7%** on the test set, showing that it performs well not only on single digits but also on **multi-digit recognition tasks**.

10. Saving the Trained Model:

Once you're confident enough to take your trained and tested model into the production-ready environment, the first step is to save it into a .h5 or .pkl file using a library like pickle. Make sure you have pickle installed in your environment. Next, let's import the module and dump the model into.h5 file

Import the module.

Create a connection function to run our code. Here we specify where we're connecting to, the user, the user's password, and then the database that we want to connect to.

As a note, we use "localhost" as our host. This just means we'll use the same server that this code is running on. You can connect to databases remotely as well, which can be pretty neat. To do that, you would connect to a host by their IP, or their domain. To connect to a database remotely, you will need to first allow it from the remote database that will be accessed/modified.

Next, let's go ahead and edit our _init_.py file, adding a register function. For now we'll keep it simple, mostly just to test our connection functionality.

We allow for GET and POST, but aren't handling it just yet.

We're going to just try to run the imported connection function, which returns c and conn (cursor and connection objects).

If the connection is successful, we just have the page say okay, otherwise it will output the error.

2.3 TECHNIQUE USED OR ALGORITHM USED

Deep Learning(DL): Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics.

Convolutional Neural Networks (CNN): ConvNets are designed to process data that come in the form of multiple arrays, for example a color image composed of three 2D arrays containing pixel intensities in the three colour channels. Many data modalities are in the form of multiple arrays: 1D for signals and sequences, including language; 2D for images or audio spectrograms; and 3D for video or volumetric images.

CHAPTER 3

REQUIREMENTS ENGINEERING

1. GENERAL

These are the requirements for doing the project. Without using these tools and software's we can't do the project. So we have two requirements to do the project. They are

1. Hardware Requirements.
2. Software Requirements.

2. HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shouls what the system do and not how it should be implemented.

- PROCESSOR : DUAL CORE 2 DUOS.
- RAM : 4GB DD RAM
- HARD DISK : 250 GB

3.3 SOFTWARE REQUIREMENTS

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

- OPERATING SYSTEM : WINDOWS 7/8/10
- PLATFORM : SPYDER3
- PROGRAMMING LANGUAGE : PYTHON, HTML
- FRONT END : SPYDER3

3.4 FUNCTIONAL REQUIREMENTS

A functional requirement defines a function of a software-system or its component. A function is described as a set of inputs, the behavior, the presented result will help us in identifying the behaviour of employees who can be attired over the next time. Experimental results reveal that the logistic regression approach can reach up to 86% accuracy over other machine learning approaches.

3.5 NON-FUNCTIONAL REQUIREMENTS

The major non-functional Requirements of the system are as follows

- **Usability**

The system is designed with completely automated process hence there is no or less user intervention.

- **Reliability**

The system is more reliable because of the qualities that are inherited from the chosen platform java. The code built by using java is more reliable.

- **Performance**

This system is developing in the high level languages and using the advanced front-end and back-end technologies it will give response to the end user on client system with in very less time.

- **Supportability**

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is having JVM, built into the system.

- **Implementation**

The system is implemented in web environment using struts framework. The apache tomcat is used as the web server and windows xp professional is used as the platform. Interface the user interface is based on Struts provides HTML Tag

CHAPTER 4

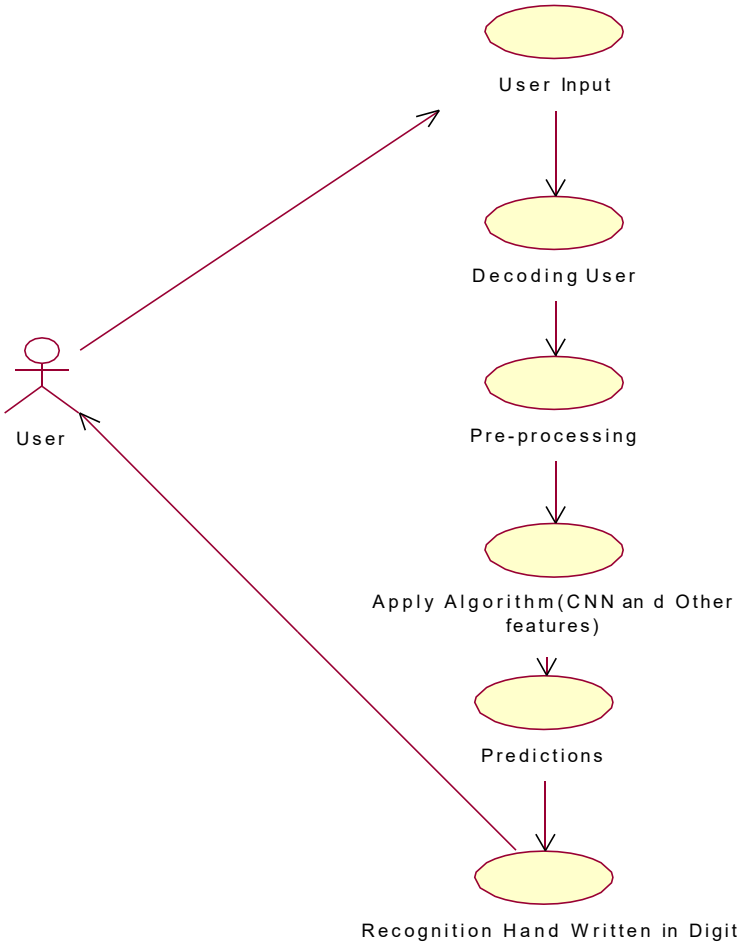
DESIGN ENGINEERING

4.1 GENERAL

Design Engineering deals with the various UML [Unified Modelling language] diagrams for the implementation of project. Design is a meaningful engineering representation of a thing that is to be built. Software design is a process through which the requirements are translated into representation of the software. Design is the place where quality is rendered in software engineering. Design is the means to accurately translate customer requirements into finished product.

UML Diagrams

USE CASE DIAGRAM:

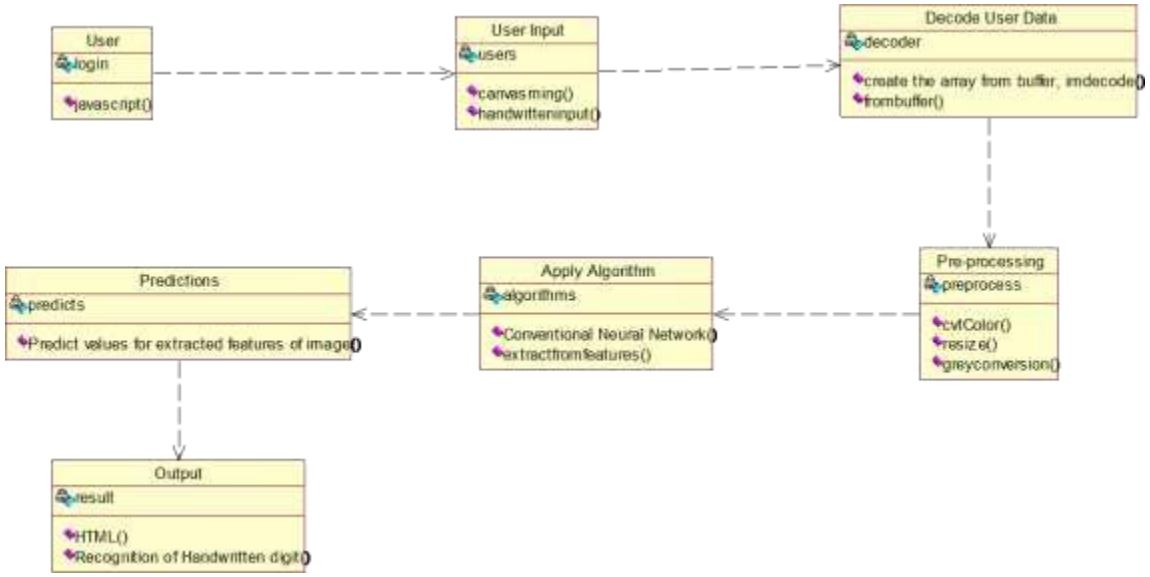


EXPLANATION:

A use case diagram in the Unified Modeling Language (UML) is a user has a login. It has a user inputs it also have a decoding a users. There are a pre processing a data. It has also apply a algorithm of cnn and other classifiers it has a features of users predictions it also a recognize a hand written in digit format.

.

CLASS DIAGRAM:

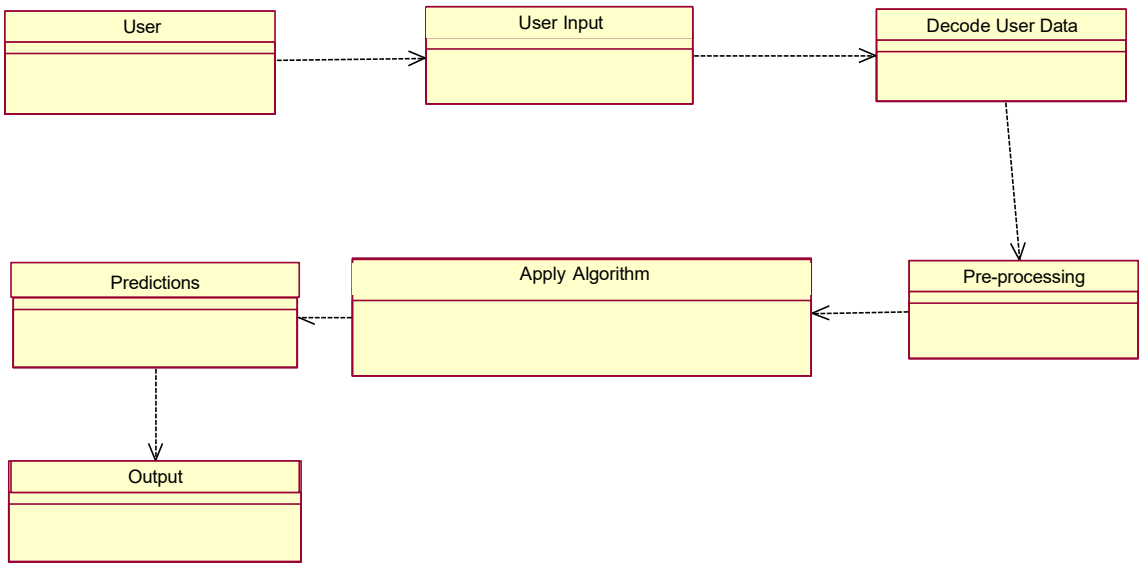


EXPLANATION:

In software engineering, a class diagram in the Unified Modelling Language (UML) it has a users has a login it shows a java script page. It also have a user inputs to writing a digit it also have a hand written input. It decodes a users data it creates an array of hand written. It also have pre-processing a data. It has apply a algorithm cnn and other features. It has a predictions of features of image. It display a output.

23

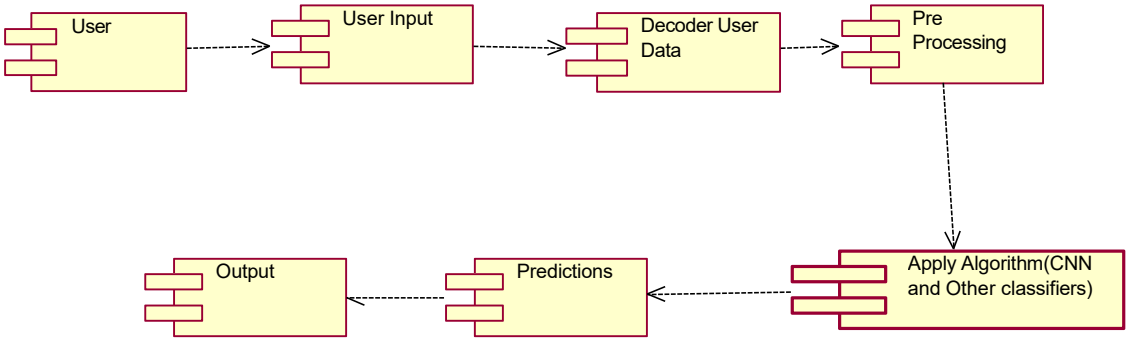
OBJECT DIAGRAM



EXPLANATION:

In the above digram tells about the flow of objects between the classes. It has a users has a link with other class to the user inputs it also decode a message of hand written letter.It will preprocessing a data and it has apply a algorithms.It also have a predictions of values and displays a output.

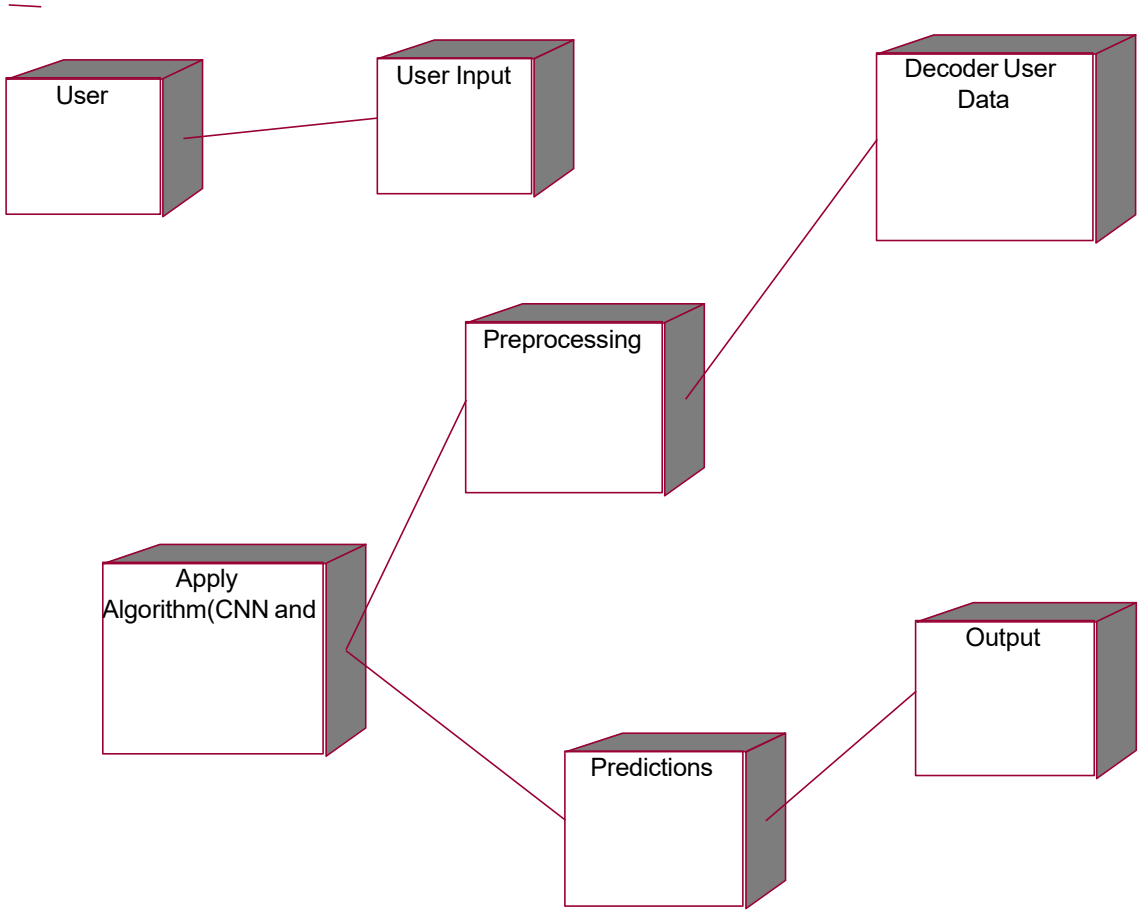
COMPONENT DIAGRAM



EXPLANATION

In Unified Modelling Language (UML), a component diagram user has link with a another box. It has a user inputs to decode a user data and then it was a pre-processing a data. It has a apply a algorithms cnn and other features and it has predict a values and displays a output in web.

DEPLOYMENT DIAGRAM

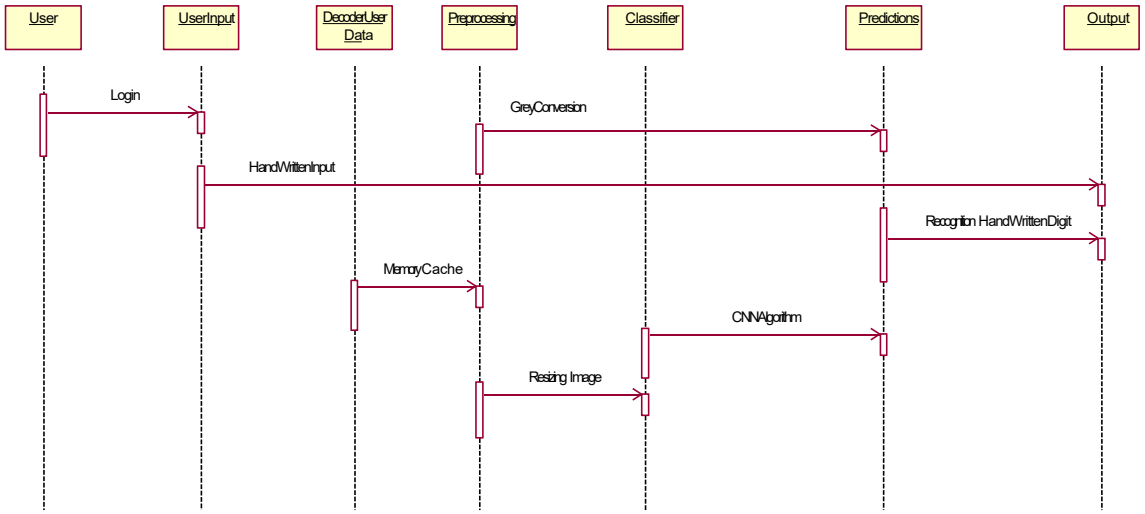


EXPLANATION:

Deployment diagrams is a kind of structure diagram used in modelling users has a link with other class to the user inputs it also decode a message of hand written letter.It will preprocessing a data and it has apply a algorithms.It also have a predictions of values and displays a output.

26

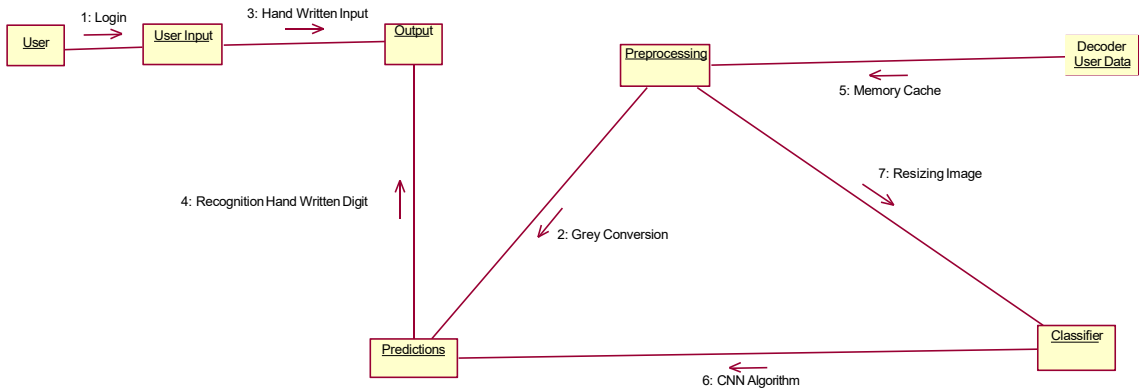
SEQUENCE DIAGRAM:



EXPLANATION:

A sequence diagram in Unified Modelling Language (UML) user has a login to user input has a hand written input. It also a decode a user data. It also a memory cache. It also have a pre-processing a data. It will Read a images. It also apply a cnn algorithm. It has a predictions of predict a values. it displays a output.

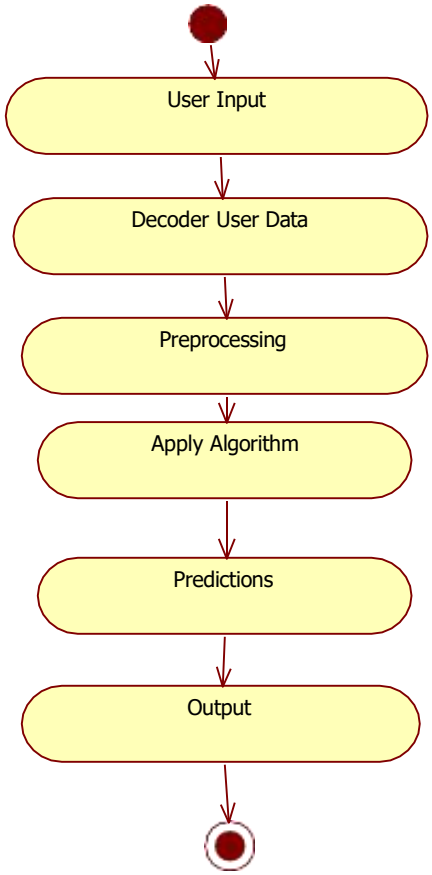
COLLABRATION DIAGRAM



EXPLANATION:

A collaboration diagram, also called a communication diagram or interaction diagram, user has a login to user input has a hand written input. It also a decode a user data. It also a memory cache. It also have a pre-processing a data. It will Read a images. It also apply a cnn algorithm. It has a predictions of predict a values. it displays a output.

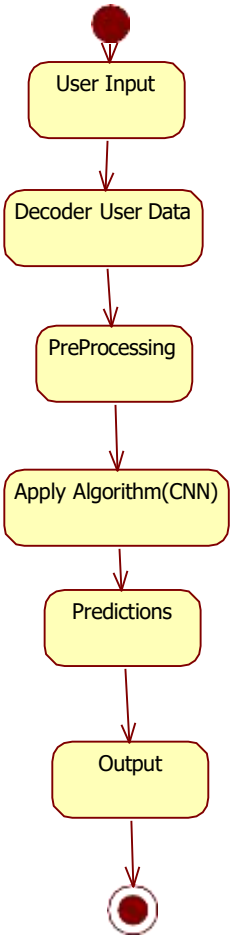
ACTIVITY DIAGRAM:



EXPLANATION:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, User has a login. It has a user inputs it also a decode a user data and it has a pre-processing a data. It was also a apply a algorithm. It has a predictions of values. It displays a output from the web.

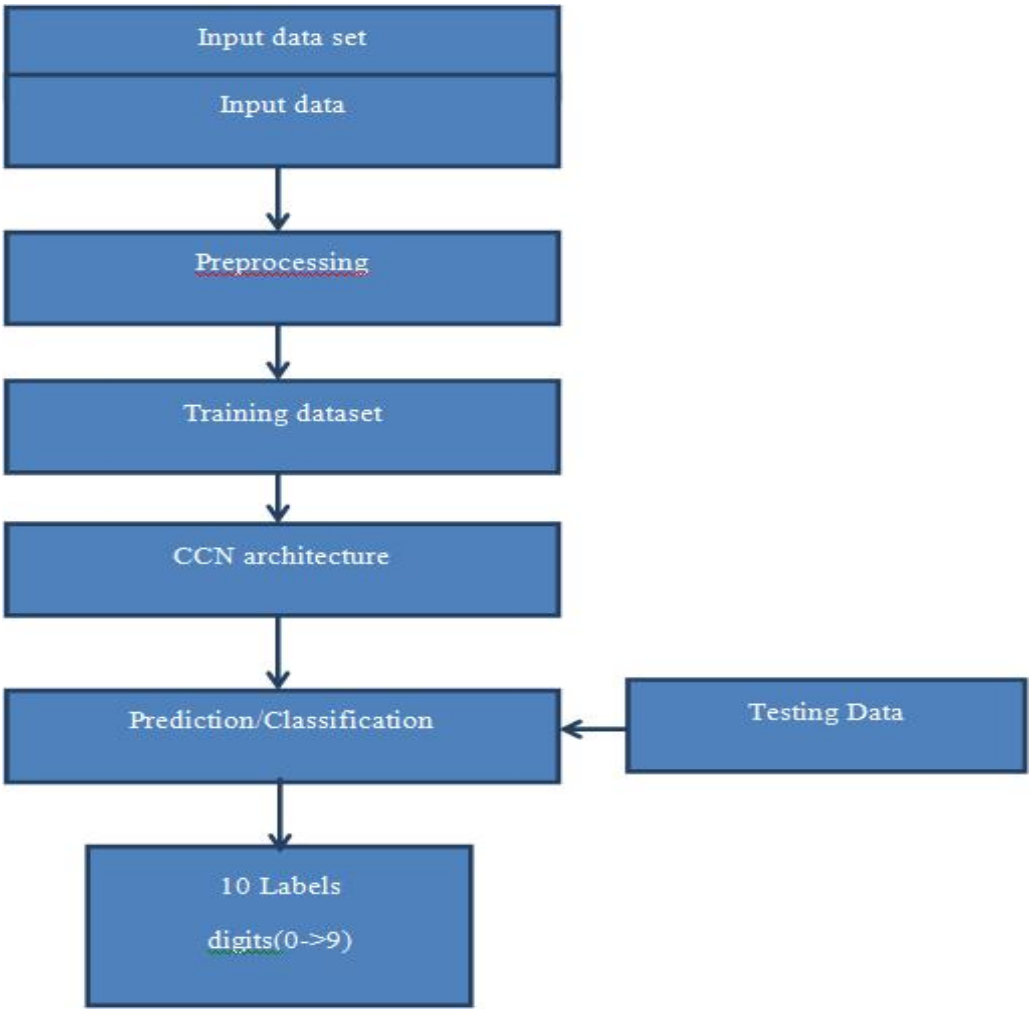
STATE DIAGRAM



EXPLANATION:

State diagram are a loosely defined diagram to show workflows of stepwise activities User has a login. It has a user inputs it also a decode a user data and it has a pre-processing a data. It was also a apply a algorithm. It has a predictions of values. It displays a output from the web.

DATA FLOW DIAGRAM:



EXPLANATION:

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often, they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

SYSTEM ARCHITECTURE:

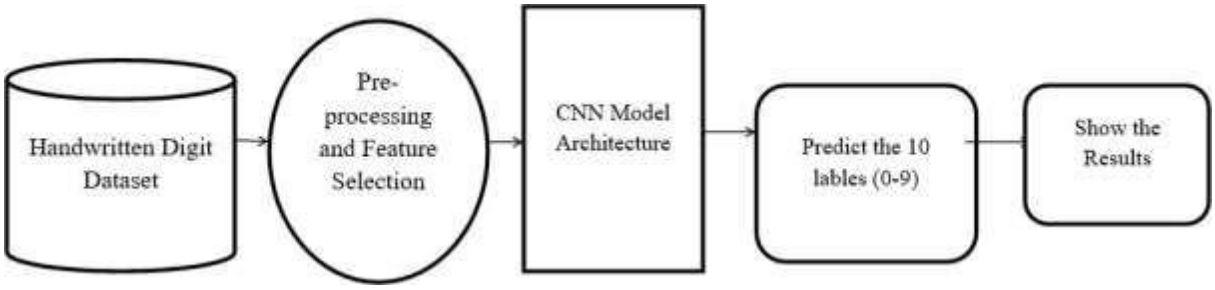


Fig: System Architecture

CHAPTER 5

DEVELOPMENT TOOLS

Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Importance of Python

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Features of Python

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.

- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Libraries used in python:

- numpy - mainly useful for its N-dimensional array objects.
- pandas - Python data analysis library, including structures such as dataframes.
- matplotlib - 2D plotting library producing publication quality figures.
- scikit-learn - the machine learning algorithms used for data analysis and data mining tasks.

Figure : NumPy, Pandas, Matplotlib, Scikit-learn

35

CHAPTER 6

IMPLEMENTATION

6.1 GENERAL

To defend against the proposed attacks and improve the robustness of ML-based NIDSs, we introduce two probable methods in prior work, and then propose a novel defense scheme named adversarial feature reduction.

Adversarial training [13]. This is a promising method widely used to defend against adversarial examples in the image domain by retraining the classifiers with correctly-labeled adversarial examples. However in our traffic-space attack, it can only reduce the attack effectiveness by limiting the generation of adversarial features.

Feature selection [55]. This is an important step in feature engineering to remove redundant/irrelevant dimensions of features used in ML models, which can effectively improve detection performance and robustness.

Adversarial feature reduction. We propose a novel scheme to explain and defend against such traffic-space adversarial attacks. In a nutshell, we proactively simulate the proposed attack and then calculate the degree to which the value of each feature dimension in the mutated traffic is close to the adversarial features compared to original value (see Appendix D for details). The proximity rates of each feature dimension can be viewed as the adversarial robustness scores. Our main claim is the high dimensionality of features gives attackers an opportunity to exploit some vulnerable dimensions to evade detection. Hence, we propose an intuitive defense scheme by deleting partial feature dimensions with low robustness scores.

Frontend code

import os

import numpy as np

```
import base64
import cv2
import tensorflow as tf
from flask import Flask, render_template, request, jsonify
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array


app = Flask(__name__)


# Load trained CNN model
MODEL_PATH = "Digit-Model.h5"
model = load_model(MODEL_PATH)


def preprocess_and_segment(image):
    """
    Preprocesses the image and extracts individual digit contours.
    Returns a list of digit images.
    """
    # Convert to grayscale
```


37

```
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


    # Apply Gaussian Blur to reduce noise
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)


    # Apply adaptive thresholding
    thresh = cv2.adaptiveThreshold(
        blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2
    )


    # Apply morphological operations to separate digits
    kernel = np.ones((3,3), np.uint8)
    thresh = cv2.dilate(thresh, kernel, iterations=1)


    # Find contours of digits
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)


    digit_images = []
```


38

```
    bounding_boxes = [cv2.boundingRect(c) for c in contours]

    # Filter out very small contours (noise)
    bounding_boxes = [b for b in bounding_boxes if b[2] > 10 and b[3] > 10]

    # Sort bounding boxes left to right
    bounding_boxes = sorted(bounding_boxes, key=lambda b: b[0])

    for x, y, w, h in bounding_boxes:
        digit = gray[y:y+h, x:x+w]

        # Ensure digit is properly padded and centered
        height, width = digit.shape
        padding = abs(height - width) // 2

        if height > width:
            digit = cv2.copyMakeBorder(digit, 0, 0, padding, padding, cv2.BORDER_CONSTANT, value=0)
        else:
```


39

```
            digit = cv2.copyMakeBorder(digit, padding, padding, 0, 0, cv2.BORDER_CONSTANT,
value=0)


        # Resize to 28x28 for model input

        digit = cv2.resize(digit, (28, 28))


        # Normalize pixel values

        digit = digit.astype("float32") / 255.0

        digit = np.expand_dims(digit, axis=-1)  # (28,28,1)

        digit = np.expand_dims(digit, axis=0)    # (1,28,28,1)


        digit_images.append(digit)


    return digit_images


@app.route("/")

def home():

    return render_template('home.html')
```


40

```
@app.route("/login")
def login():
    return render_template('login.html')


@app.route("/performance")
def performance():
    return render_template('performance.html')


@app.route('/index')
def index():
    return render_template('index.html')


@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        try:
            # Get base64 image from canvas
            canvas_img = request.form['canvasimg']
            img_data = base64.b64decode(canvas_img.split(',')[1])
```


41

```
    # Convert to OpenCV image
    np_arr = np.frombuffer(img_data, np.uint8)
    img = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)

    # Process image and extract digits
    digit_images = preprocess_and_segment(img)

    # Predict digits
    predicted_digits = []
    for digit in digit_images:
        prediction = model.predict(digit)
        predicted_digits.append(str(np.argmax(prediction))) # Get most probable digit

    return jsonify({"prediction": "".join(predicted_digits)})

except Exception as e:
    return jsonify({"error": str(e)})
```


42

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

backend code:

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout, BatchNormalization, Input, Reshape, LSTM, Bidirectional
from keras.models import Model
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import backend as K
from keras.layers import Lambda
# Load training and testing datasets (Modify the file paths if necessary)
train_images = pd.read_csv("F:/TEJA_MINI_2025/MPAI10/working_code/model/emnist-digits-train.csv", header=None)
test_images = pd.read_csv("F:/TEJA_MINI_2025/MPAI10/working_code/model/emnist-digits-test.csv", header=None)
```


43

```
# Extract features (pixel values) and labels
train_x = train_images.iloc[:, 1:].values  # Image pixels
train_y = train_images.iloc[:, 0].values   # Labels (Digit sequences)

test_x = test_images.iloc[:, 1:].values
test_y = test_images.iloc[:, 0].values

# Normalize images (Convert pixel values from 0-255 to 0-1)
train_x = train_x.astype('float32') / 255.0
test_x = test_x.astype('float32') / 255.0

# Reshape images to (28, 28, 1) for CNN input
train_x = train_x.reshape(-1, 28, 28, 1)
test_x = test_x.reshape(-1, 28, 28, 1)

# Print shape to verify
print(f"Training Data Shape: {train_x.shape}, Labels Shape: {train_y.shape}")
print(f"Testing Data Shape: {test_x.shape}, Labels Shape: {test_y.shape}")
```

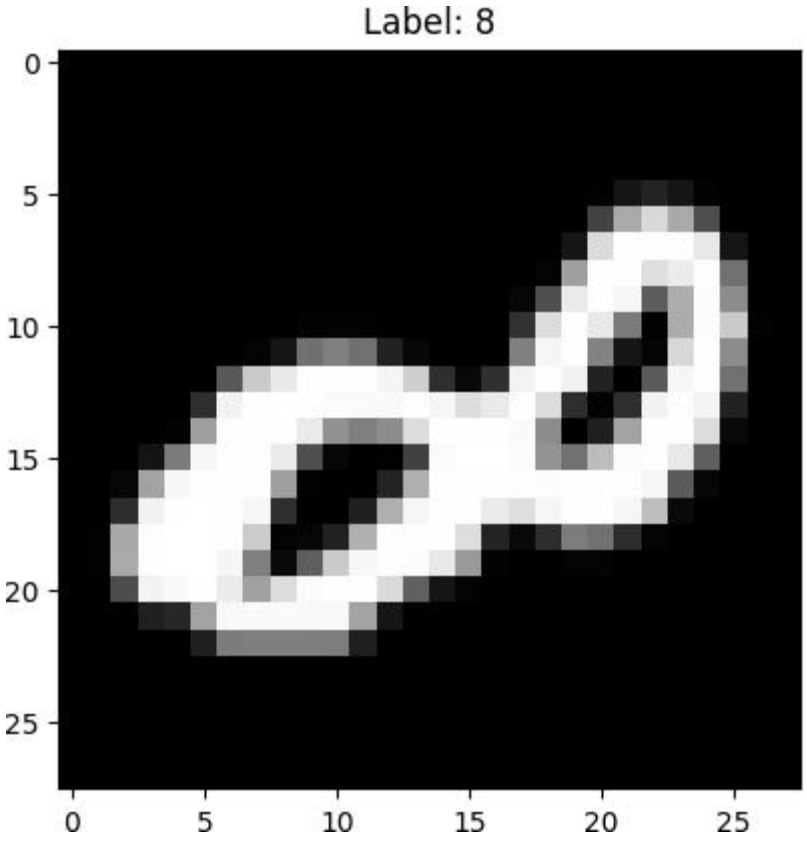

44

 # Visualize an example image

plt.imshow(train_x[0].reshape(28, 28), cmap='gray')

plt.title(f"Label: {train_y[0]}")

plt.show()



from keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.utils import to_categorical

 # Convert labels to string format

train_y = train_y.astype(str)

45

```
test_y = test_y.astype(str)



# Define maximum sequence length (adjust based on dataset) MAX_SEQ_LENGTH = 5
# Adjust based on max digits in dataset



# Pad sequences (ensures uniform label length)



train_y_padded = pad_sequences([list(map(int, y)) for y in
    train_y], maxlen=MAX_SEQ_LENGTH, padding='post')



test_y_padded = pad_sequences([list(map(int, y)) for y in test_y],
maxlen=MAX_SEQ_LENGTH, padding='post')



# Convert labels to categorical (one-hot encoding for multi-class classification)

num_classes = 10 # Digits 0-9

train_y_encoded = np.array([to_categorical(seq,
    num_classes=num_classes)
 train_y_padded]) for seq in
test_y_encoded = np.array([to_categorical(seq,
    num_classes=num_classes)
test_y_padded]) for seq in


# Print shape to verify
```

print(f"Train Labels Shape (Encoded): {train_y_encoded.shape}")

47

```
print(f"Test Labels Shape (Encoded): {test_y_encoded.shape}")

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Reshape, Bidirectional, LSTM


# Define input layer

inputs = Input(shape=(28, 28, 1))


# Convolutional Layers

x = Conv2D(32, (3,3), activation='relu', padding='same')(inputs)

x = MaxPooling2D((2,2))(x)


x = Conv2D(64, (3,3), activation='relu', padding='same')(x)

x = MaxPooling2D((2,2))(x)


x = Conv2D(128, (3,3), activation='relu', padding='same')(x)

x = MaxPooling2D((2,2))(x)  # Output shape: (3, 3, 128)


#    **Fix:** Adjust Reshape Layer
```


48

x = Reshape((9, 128))(x) # 3 * 3 = 9, keeping total elements unchanged

 # LSTM Layers

x = Bidirectional(LSTM(128, return_sequences=True))(x)

x = Bidirectional(LSTM(128, return_sequences=True))(x)

 # **Fix: Flatten before Dense**

x = Flatten()(x) # Ensures single-dimension output

 # Fully Connected Layers

x = Dense(256, activation='relu')(x)

x = Dropout(0.3)(x)

 # **Fix: Ensure correct output shape**

x = Dense(5 * 10, activation='softmax')(x) # ◇ (5 × 10) = 50 total outputs

outputs = Reshape((5, 10))(x) # ☑ Now correctly reshapes into (5, 10)

 # Create the model

49

```
model = Model(inputs=inputs, outputs=outputs)

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


# Print model summary

model.summary()


Model: "model"


 Layer (type)                Output Shape              Param #

=================================================================

 input_1 (InputLayer)        [(None, 28, 28, 1)]       0


 conv2d (Conv2D)             (None, 28, 28, 32)        320


 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0

 D)


 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496
```


50

max_pooling2d_1 (MaxPoolin (None, 7, 7, 64) 0

g2D)

conv2d_2 (Conv2D) (None, 7, 7, 128) 73856

max_pooling2d_2 (MaxPoolin (None, 3, 3, 128) 0

g2D)

reshape (Reshape) (None, 9, 128) 0

bidirectional (Bidirection (None, 9, 256) 263168

al)

bidirectional_1 (Bidirecti (None, 9, 256) 394240

onal)

flatten (Flatten) (None, 2304) 0

dense (Dense) (None, 256) 590080

51

dropout (Dropout) (None, 256) 0

dense_1 (Dense) (None, 50) 12850

reshape_1 (Reshape) (None, 5, 10) 0

===

Total params: 1353010 (5.16 MB)

Trainable params: 1353010 (5.16 MB)

Non-trainable params: 0 (0.00 Byte)

import matplotlib.pyplot as plt

 # Define training parameters

BATCH_SIZE = 64

EPOCHS = 5 # Increase if needed for better accuracy

 # Train the model

history = model.fit(

52

```
    train_x, train_y_encoded,
    validation_data=(test_x, test_y_encoded),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE
)

# Save trained model
model.save('Digit-Model.h5')

# Plot accuracy & loss curves
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train  Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation  Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

plt.subplot(1,2,2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

Epoch 1/5

3750/3750 [==============================] - 559s 147ms/step - loss: 0.0346 - accuracy: 0.9888 - val_loss: 0.0069 - val_accuracy: 0.9979

Epoch 2/5

3750/3750 [==============================] - 481s 128ms/step - loss: 0.0063 - accuracy: 0.9982 - val_loss: 0.0057 - val_accuracy: 0.9983

Epoch 3/5

3750/3750 [==============================] - 540s 144ms/step - loss: 0.0047 - accuracy: 0.9987 - val_loss: 0.0041 - val_accuracy: 0.9988

Epoch 4/5

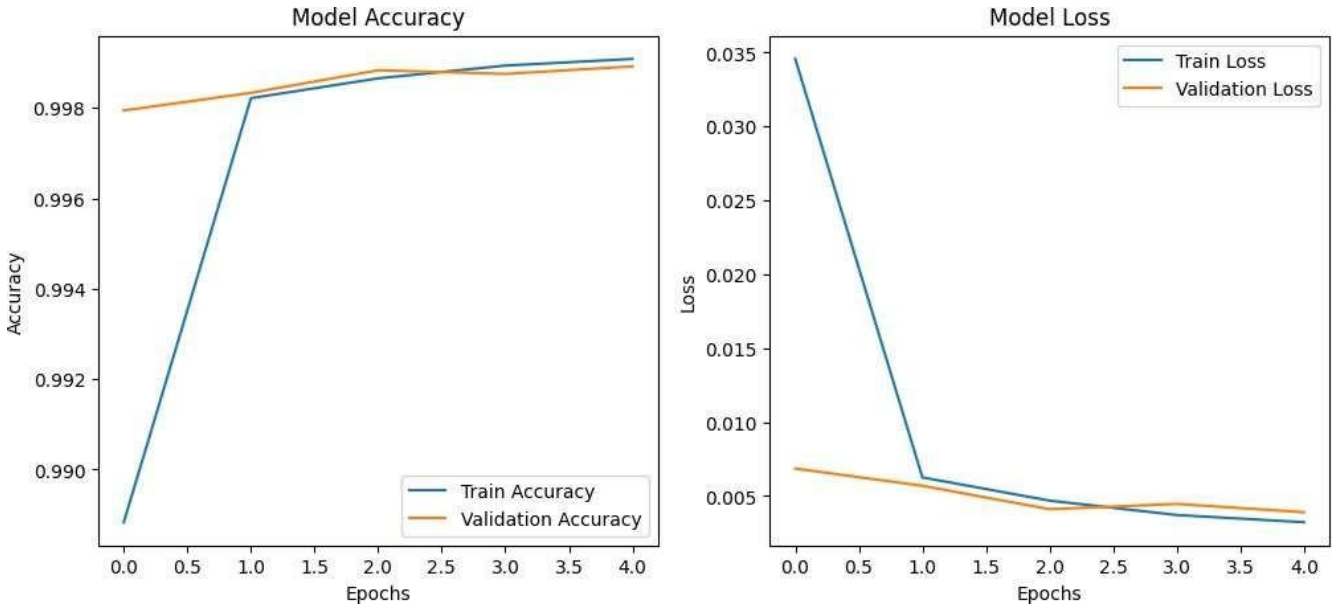
3750/3750 [==========] - 607s 162ms/step - loss: 0.0037 - accuracy: 0.9989 - val_loss: 0.0045 - val_accuracy: 0.9988

Epoch 5/5

3750/3750 [==========] - 577s 154ms/step - loss: 0.0032 - accuracy: 0.9991 - val_loss: 0.0039 - val_accuracy: 0.9989

C:\Users\Main Python2\anaconda3\envs\3.9\lib\site-packages\keras\src\engine\training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.

saving_api.save_model(



import numpy as np

import matplotlib.pyplot as plt

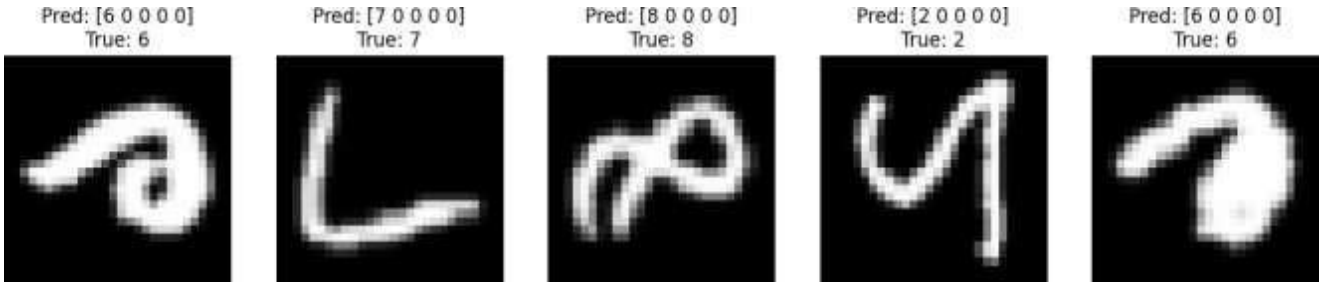
from tensorflow.keras.models import load_model

55

```
# Load the trained model

model = load_model('MultiDigit-Model.h5')


# Select some random test images

num_samples = 5  # Change as needed

random_indices = np.random.choice(test_x.shape[0], num_samples, replace=False)

sample_images = test_x[random_indices]

sample_labels = test_y[random_indices]  # True labels for comparison


# Make predictions

predictions = model.predict(sample_images)


# Convert predictions to readable format

predicted_digits = np.argmax(predictions, axis=-1)  # Get highest probability index


# Display results

fig, axes = plt.subplots(1, num_samples, figsize=(15, 3))

for i, ax in enumerate(axes):
```


56

```
    ax.imshow(sample_images[i].squeeze(), cmap='gray')
    ax.set_title(f"Pred: {predicted_digits[i]}\nTrue: {sample_labels[i]}")
plt.show()
```



57

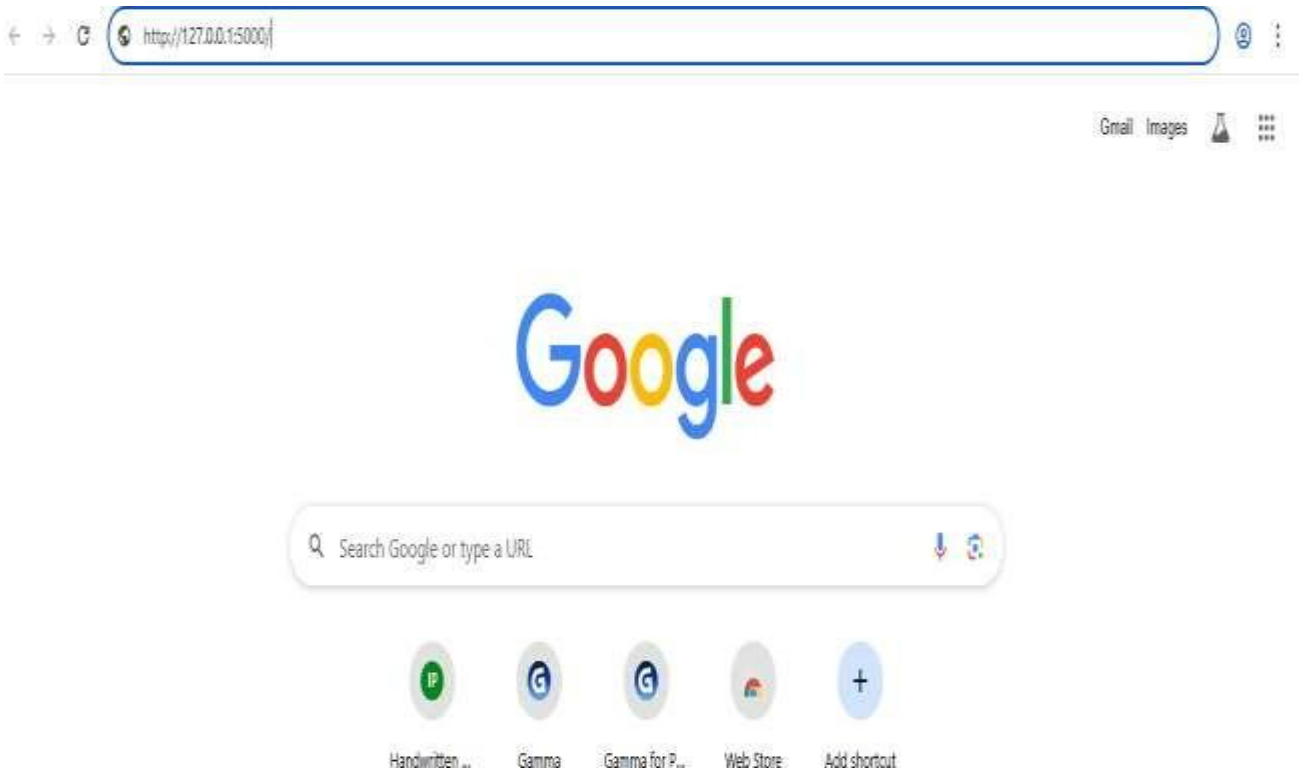
CHAPTER 7

SNAPSHOTS

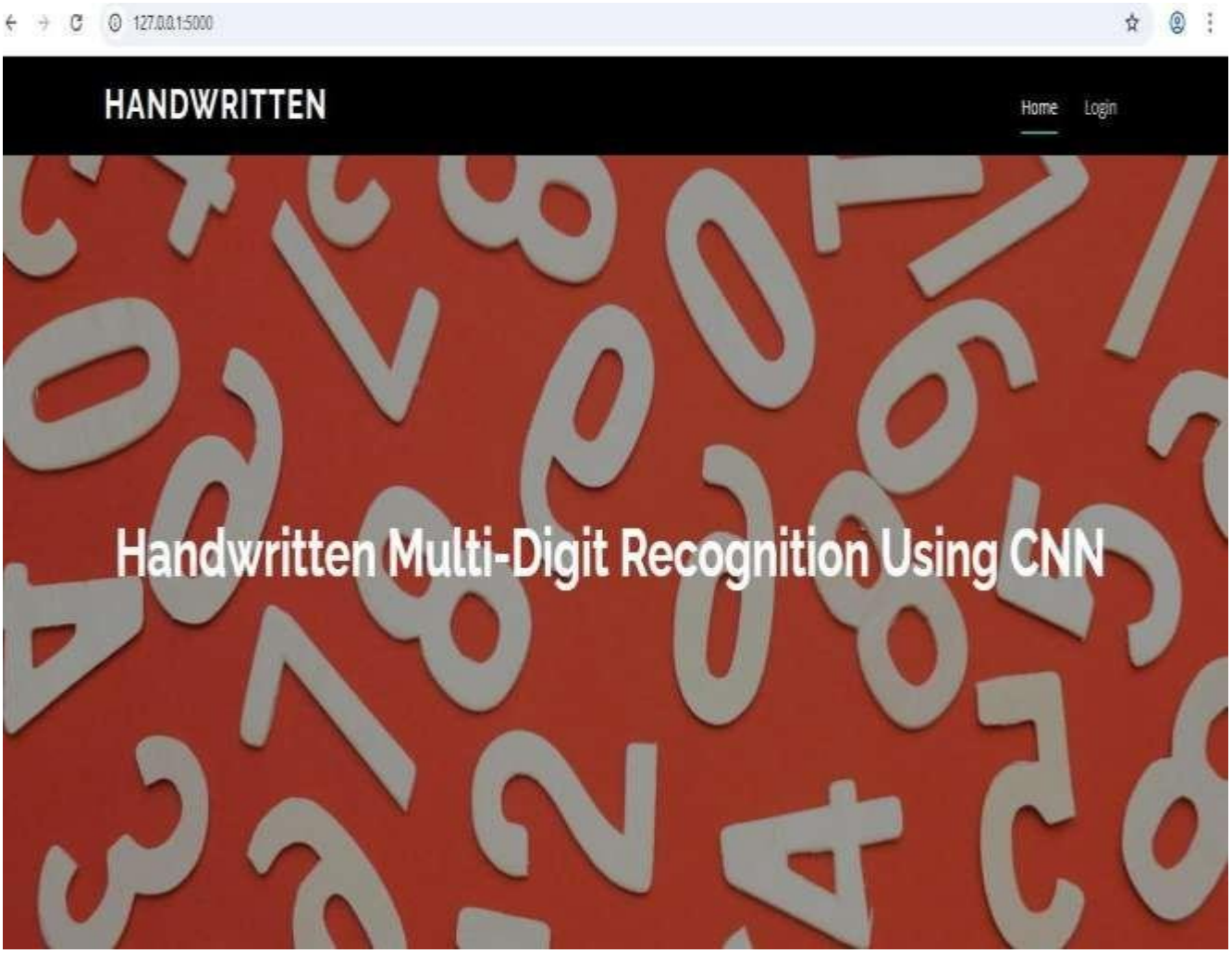
GENERAL

This project is implements like application using python and the Server process is maintained using the SOCKET & SERVERSOCKET and the Design part is played by Cascading Style Sheet.

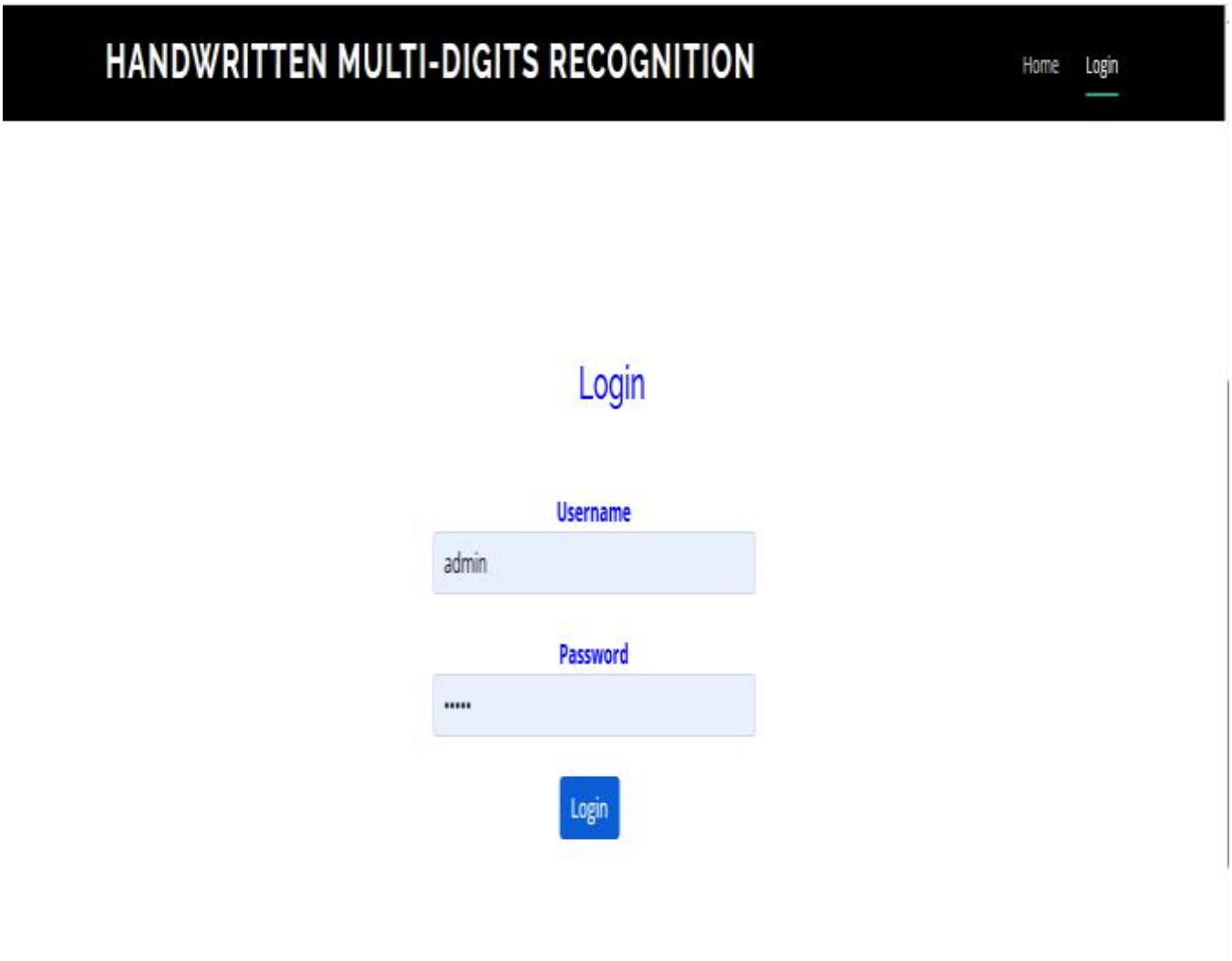
SNAPSHOTS:



58


127.0.0.1:5000
HANDWRITTEN
Home
Login
Handwritten Multi-Digit Recognition Using CNN

59


HANDWRITTEN MULTI-DIGITS RECOGNITION
Home
Login
Login
Username
admin
Password
•••••
Login

60

61

CHAPTER 8

SOFTWARE TESTING

1. GENERAL

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

2. DEVELOPING METHODOLOGIES

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

3. Types of Tests

1. Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

8.3.2 Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

3. System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

4. Performance Test

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

5. Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g., components in a software system or – one step up – software applications at the company level – interact without error.

6. Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Acceptance testing for Data Synchronization:

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
- The Route add operation is done only when there is a Route request in need
- The Status of Nodes information is done automatically in the Cache Updation process

8.2.7 Build the test plan

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identity the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

CHAPTER 9

FUTURE ENHANCEMENT

9.1 FUTURE ENHANCEMENT

In future, various designs of CNN, in particular, cross breed CNN, viz., CNN-RNN and CNN-HMM models, and space explicit acknowledgment frameworks, can be researched. Developmental calculations can be investigated for streamlining CNN learning boundaries, to be specific, the quantity of layers, learning rate and portion sizes of convolutional channels.

.

CHAPTER 10

CONCLUSION & REFERENCE

10.1 CONCLUSION

In this work, with the point of improving the exhibition of transcribed digit acknowledgment, we assessed variations of a convolutional neural organization to keep away from complex pre-preparing, exorbitant component extraction and a perplexing troupe (classifier blend) approach of a conventional acknowledgment framework. Through broad assessment utilizing a MNIST dataset, the current work recommends the job of different hyper-boundaries. We additionally confirmed that tweaking of hyper-boundaries is fundamental in improving the presentation of CNN engineering. We accomplished acknowledgment pace of99.89% with the Adam analyzer for the MNIST information base, which is superior to all recently revealed outcomes.

The impact of expanding the quantity of convolutional layers in CNN design on the presentation of transcribed digit acknowledgment is unmistakably introduced through the tests. The oddity of the current work is that it altogether explores all the boundaries of CNN engineering that convey best acknowledgment precision for a MNIST dataset. Companion scientists couldn't coordinate this precision utilizing an unadulterated CNN model. A few analysts utilized gathering CNN network models for the equivalent dataset to improve their acknowledgment precision at the expense of expanded computational expense and high testing multifaceted nature yet with practically identical exactness as accomplished in the present work.

REFERENCES

1 Niu, X.X.; Suen, C.Y. A novel hybrid CNN–SVM classifier for recognizing handwritten digits.Pattern Recognit.2012, 45, 1318–1325.

2 Long, M.; Yan, Z. Detecting iris liveness with batch normalized convolutional neural network. Compute, Mater. Contin. 2019, 58,493–504.

3 Y. LeCun et al., "Backpropagation applied to handwritten zip code recognition," Neural computation, vol. 1, no. 4, pp. 541-551, 1989.

4 Sueiras, J.; Ruiz, V.; Sanchez, A.; Velez, J.F. Offline continuous handwriting recognition using sequence to sequence neural networks. Neurocomputing. 2018, 289, 119–128.

5 Wells, Lee & Chen, Sheng Feng&Almamlook, Rabia&Gu, Yuwen.(2018). Offline Handwritten Digits Recognition Using machine learning.

6 Burel, G., Pottier, I., &Catros, J. Y. (1992, June). Recognition ofhandwritten digits by image processing and neural network.InNeural Networks, 1992. IJCNN, International Joint Conference on(Vol. 3, pp. 666-671) IEEE.

7 Salvador España-Boquera, Maria J. C. B., Jorge G. M. andFrancisco Z. M., “Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models”, IEEE Transactions on PatternAnalysis and Machine Intelligence, Vol. 33, No. 4, April 2014.

8 Ahmed, M., Rasool, A. G., Afzal, H., &Siddiqi, I. (2017).Improving handwriting-based gender classification using ensemble classifiers. Expert Systems with Applications, 85, 158-168.

9 Sadri, J., Suen, C. Y., & Bui, T. D. (2007). A genetic framework using contextual knowledge for segmentation and recognition ofhandwritten numeral strings. Pattern Recognition, 40(3), 898-919.

[10] Sarkhel, R., Das, N., Das, A., Kundu, M., &Nasipuri, M. (2017). Amulti-scale deep quad tree-based feature extraction method for therecognition of isolated handwritten characters of popular Indic scripts. Pattern Recognition, 71, 78-93.

68