Neural Network Digit Classification

By

Hammad Arif

2021-GCUF-031

Zain Ali

2021-GCUF-028

Project Code

CSI-631

Project Advisor

Prof Arooj

DEPARTMENT OF COMPUTER SCIENCE

***Government College University Faisalabad* 2025**

Acknowledgement:

Firstly, I would like to express my deepest gratitude to **Allah Almighty** for granting me the strength, patience, and ability to complete this Final Year Project successfully.

I would also like to extend my heartfelt thanks to my **Final Year Project Supervisor, Prof. Arooj,** esteemed teacher at **Superior College Wazirabad**, for his valuable guidance, unwavering support, and insightful suggestions throughout the development of this project. His expertise and encouragement have played a vital role in the successful completion of this system.

Furthermore, I am sincerely grateful to my **beloved parents** for their continuous moral support, prayers, and motivation. Their constant encouragement and belief in my capabilities helped me overcome challenges at various stages of the project journey.

Without the support of these individuals, this project would not have reached its successful completion.

Zain Ali (2021 – GCUF - 028)

Hammad Arif (2021 – GCUF – 031)

Certificate:

This is to certify that the Final Year Project entitled **"Neural Network Digital Classification"** has been completed and submitted by the following Students:

- **Zain Ali**	Roll No. **2021-GCUF-028**
- **Hammad Arif**	Roll No. **2021-GCUF-031**

This project is a bona fide piece of work carried out by the above-mentioned students under the supervision of **Prof Arooj** in partial fulfillment of the requirements for the degree of **Bachelor of Science in Computer Science**.

I further certify that this work is original and has not been submitted to **Government College University Faisalabad** for the award of any degree.

Internal Examiner

Name: __________________

Signature ___________________

External Examiner:

Name: ______________________

Signature:____________________

Declaration:

We hereby declare that the Final Year Project titled **"Neural Network digital Classification"** is the result of our own independent work carried out under the supervision of **Prof.Arooj**, at **Superior College Wazirabad**, affiliated with **Government College University Faisalabad (GCUF).**

This project has been completed and submitted in partial fulfillment of the requirements for the degree of **Bachelor of Science in Computer Science (Session 2021–2025).**

We further declare that this project report has not been submitted previously to any other university or institution for the award of any degree, diploma, or certificate.

Zain Ali

Hammad Arif

Table of Content:

List of Tables:

List of Figures:

Chapter 01: Introduction to the problem

1.1 Introduction:

Digit classification is a common task in the field of machine learning and computer vision. It involves recognizing handwritten numbers (0 to 9) from images and predicting the correct digit. This technology is used in many real-life applications such as postal code reading, automatic bank check processing, and digitizing handwritten documents.

In this project, we use a neural network to classify handwritten digits. Neural networks are designed to mimic the way the human brain learns patterns. By training the model on a large dataset of digit images, the system learns to identify and classify digits with high accuracy.

We use the MNIST dataset, which contains thousands of images of handwritten digits, to train and test our model. This dataset is widely used for testing the performance of digit recognition systems. The goal of the project is to build a model that can recognize digits accurately and learn how deep learning techniques can be applied to real-world problems.

1.2 Background:

Before neural networks were widely used, recognizing handwritten digits was done using manual methods or rule-based systems. These methods often gave poor results and were hard to update for new data.

Neural networks have changed the way we solve image-related problems. They are very good at finding patterns in images and have been successfully used for handwriting recognition.

The MNIST dataset, created by Yann LeCun and others, contains 70,000 grayscale images of handwritten digits (0 to 9). It is a common benchmark used to test how well machine learning models perform in digit classification tasks.

1.3 Purpose:

The main purpose of this project is to build and test a neural network that can recognize handwritten digits with high accuracy. It also aims to help understand

How deep learning models work and how they can be used to solve real-world problems. The project aims to:

- Build a neural network model that can recognize handwritten numbers.
- Learn how deep learning works in real-life problems.
- Train and test the model to make sure it works well.
- Possibly create a simple app to show how it works in action.

1.4 Scope:

This project is limited to the classification of handwritten digits (0–9) using image data from the MNIST dataset. It involves building, training, and evaluating a neural network model to recognize these digits.

The main tasks covered in the project include:

- Using the MNIST dataset to train and test the model.
- Creating and testing different types of neural networks.
- Evaluating the accuracy of the model using test data.
- Creating a simple interface where users can draw a digit and get predictions from the model.

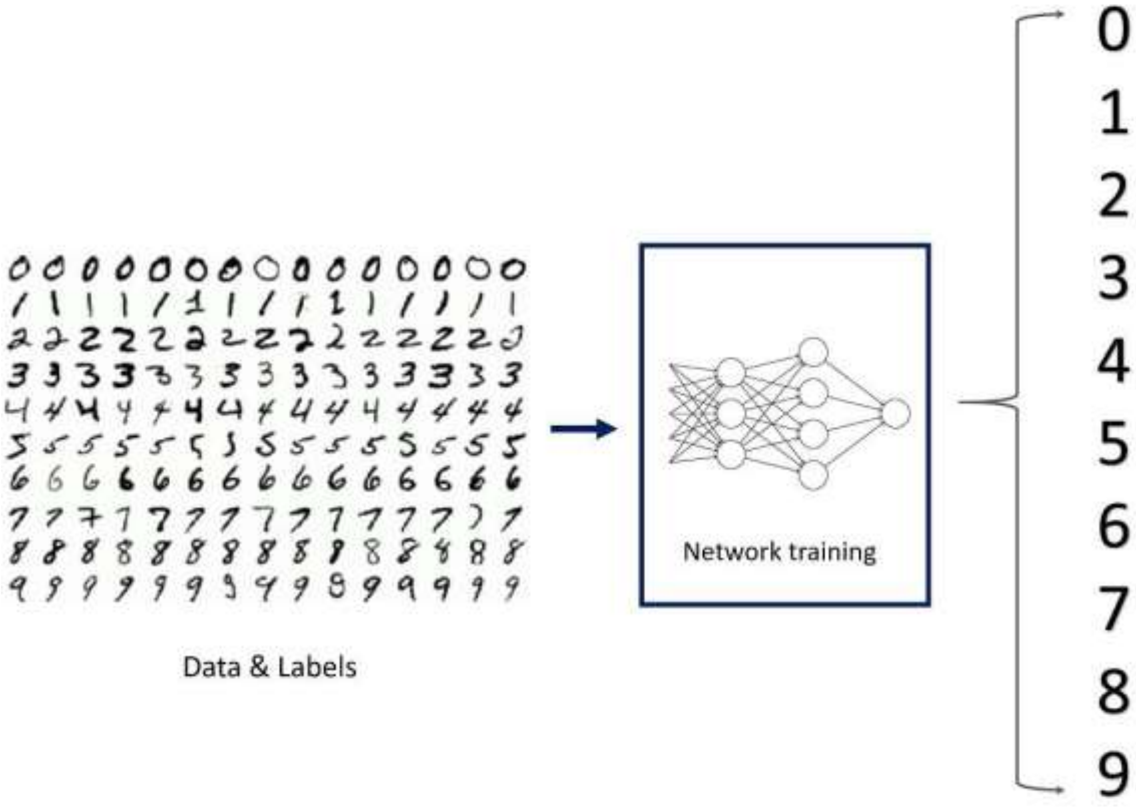
The project does not include the recognition of letters, symbols, or digits from complex real-world images. However, the techniques used here can be extended to such tasks in future work.

1.5 Objective:

The main objectives of this project are:

- To create a neural network model that can recognize handwritten digits.
- To learn how to preprocess image data and train a deep learning model.
- To test the model and measure how accurately it performs.
- To try different models and choose the best one.

- To document the process clearly for learning and presentation.



1.6 Intended Audience and Reading Suggestions:

This document is intended for the following audience:

- **Academic Evaluators and Faculty Members**: to review the project structure, technical depth, and learning outcomes.
- **Students and Learners**: who are interested in understanding how neural networks are used in digit recognition.
- **Developers**: looking to explore practical applications of deep learning in image classification tasks.

- **Researchers**: who may use this project as a base for further study or experimentation in the field of computer vision and AI?

Reading Suggestions:

- Provides an overview of the project, including its purpose, scope, and objectives.
- Reviews existing research and similar projects in the area of digit recognition.
- Details the system design, neural network architecture, and implementation steps.
- Covers results and evaluation, showing how well the model performs.

1.7 Document Convention List:

Convention for main Title heading

Font face: Time New Roman

Font style: Bold

Font size: 18

Convention for main heading

Font face: Time New Roman

Font style: Bold

Font size: 16

Convention for body content

Font face: Time New Roman

Font size: 14

Document is justified

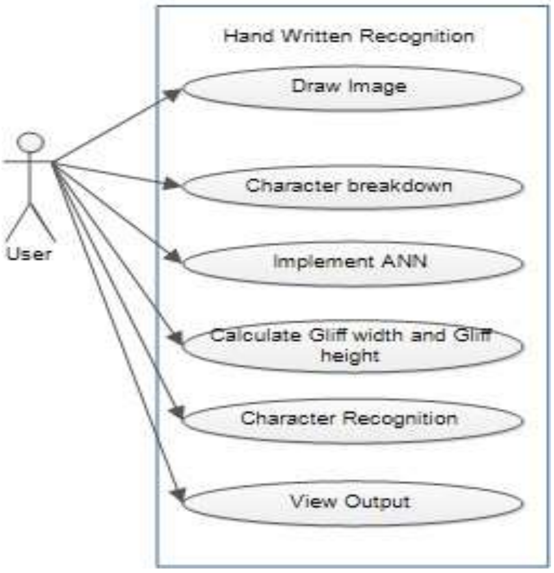
used bullets points

Chapter 02: Software Requirement Specification

2.1 Overall Description

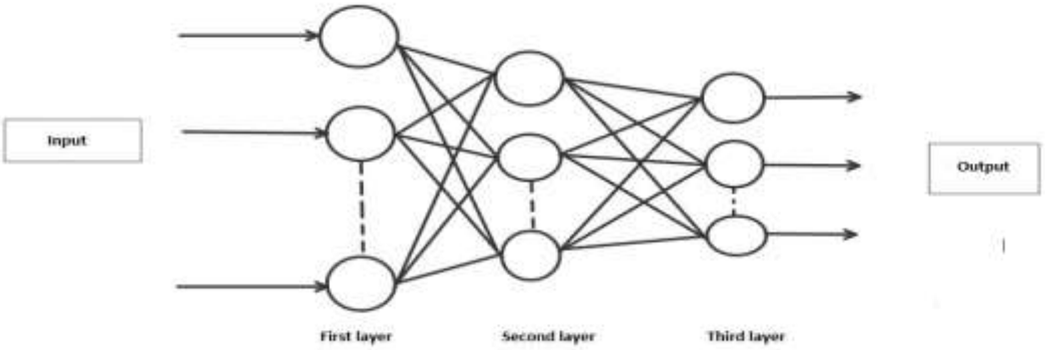
This section provides a general overview of the project. It explains how the system works, what it is used for, and the environment in which it will run. It does not go into technical detail but gives readers a basic understanding of the software, its features, limitations, and the assumptions made during development.

Use case diagram



Product Perspective

This digit recognition system is software that takes handwritten digits as input and predicts the correct digit using a neural network. It uses the MNIST dataset for training and testing. Although it's an independent system, it can be integrated into other applications like school grading systems, document readers, or mobile scanning apps.



Product Features

Here’s what this software can do:

- **Understand Handwritten Numbers** Accurately predicts digits from 0 to 9 from an image.
- **Prepare Images:** It changes the image into a simple format so the computer can understand it better.
- **Train and Learn:** The program learns how to recognize digits by looking at thousands of examples.
- **Show the Result:** It tells the user what digit it sees in the image.
- **Show Accuracy:** It shows how good the model is at recognizing digits.
- **GUI:** A simple screen where users can draw or upload a digit for testing.

Design and Implementation Constraints

These are things the project needs or rules it must follow:

- **Programming Language:** We use Python, along with tools like NumPy pandas, pillow.
- **Computer Power:** It can run on a normal computer, but works faster on a GPU (graphics card).

- **Image Type:** Only works with 28x28 black-and-white images, like those in the MNIST dataset.
- **Model Size:** The model is not too complex so it can run faster and be easier to understand

Assumptions and Dependencies

These are things we believe to be true and what the project depends on:

- The digit images should be neat and centered, like the MNIST examples.
- Python and the required libraries (like NumPy pandas and pillow) are already installed.
- The system will be used on a local computer or laptop.
- The MNIST dataset is accurate and works well for training the system.

2.2 System Features

This part explains what the system can do. Each feature is like a step the system follows to recognize a handwritten digit.

Feature 1: Upload or Draw a Digit
The user can upload an image of a handwritten digit or draw one using a mouse or touchscreen.

Feature 2: Prepare the Image
The system cleans and resizes the image to the right size (28x28 pixels) and turns it into data the computer can understand.
It makes sure all images look similar so the system can recognize them better.

Feature 3: Predict the Digit

The trained neural network looks at the image and guesses what digit it is (from 0 to 9).

This is the main goal to know what number was written in the image.

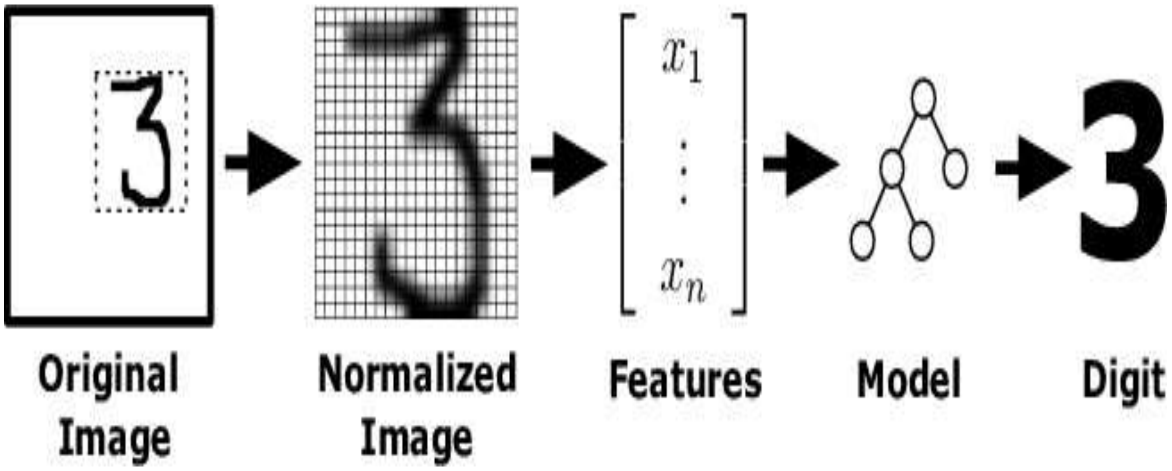
Feature 4: Show the Result

After guessing, the system shows the user the number it thinks is in the picture. The user can see the result and know if the system was correct.

Feature 5: Train the Model

Developers can train the neural network using many examples (like from the MNIST dataset) so it learns how digits usually look.

Training helps the system improve its accuracy in recognizing digits.



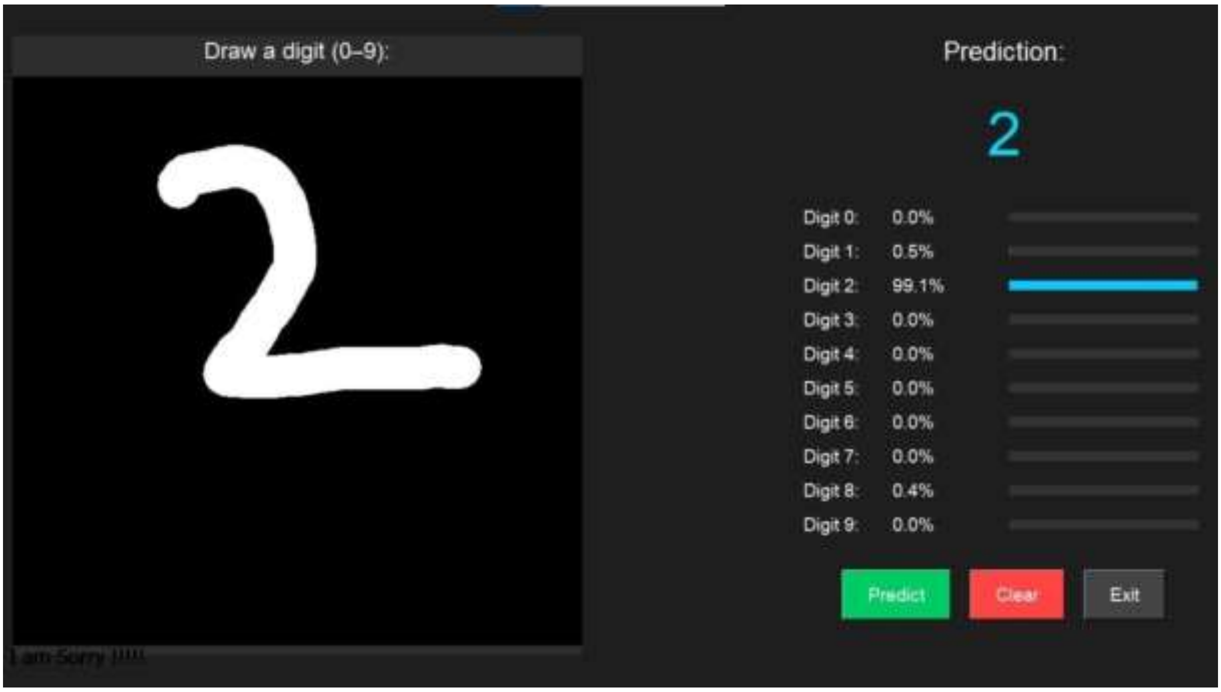
2.3 External Interface Requirements

This section describes how the system interacts with users, hardware, software, and other systems.

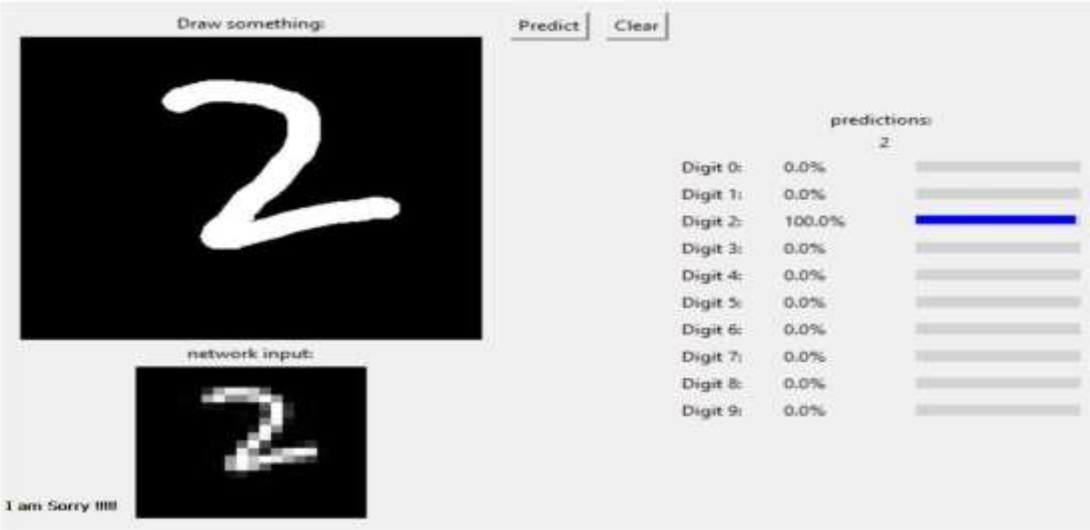
User Interface

The system will have a simple and user-friendly interface. Users can upload a digit image or draw it directly after processing the result will be shown clearly on the screen.

- Simple buttons for uploading or drawing
- Clear display of prediction result
- Display of prediction confidence (e.g., 95%)



```
Prediction:
2
Digit 0:  0.0%
Digit 1:  0.5%
Digit 2:  99.1%
Digit 3:  0.0%
Digit 4:  0.0%
Digit 5:  0.0%
Digit 6:  0.0%
Digit 7:  0.0%
Digit 8:  0.4%
Digit 9:  0.0%
I am Sorry !!!!!
```



```
predictions:
2
Digit 0:  0.0%
Digit 1:  0.0%
Digit 2:  100.0%
Digit 3:  0.0%
Digit 4:  0.0%
Digit 5:  0.0%
Digit 6:  0.0%
Digit 7:  0.0%
Digit 8:  0.0%
Digit 9:  0.0%
I am Sorry !!!!!
```

Hardware Interface

The system runs on general computer hardware.

- Input Device: Mouse or touchscreen (for drawing digits), keyboard
- Output Device: Monitor or screen
- Minimum RAM: 4 GB
- Processor: Any modern processor (i3 or above recommended)

Software Interface

The system uses several software tools and libraries:

- **Operating System:** Windows, Linux, or macOS
- **Programming Language:** Python
- **Libraries:** (NumPy pandas and pillow)

2.4 Other Non-Functional Requirements

This part talks about how the system should work how fast, safe, and secure it should be.

Performance Requirements

- The system should give the result within 1 or 2 seconds after the user gives an input.
- It should be correct most of the time**,** at least 85% accurate**.**
- It should work smoothly on normal computers without slowing down.

Safety Requirements

- The system should be safe to use and should not harm the computer.

- If the user gives a wrong image or does something by mistake, the system should not crash**.**
- It should show easy-to-understand error messages when something go wrong.

Security Requirements

- If used on the internet, the user's images and data should be private and protected**.**
- No one else should be able to change or access the system without permission.
- If any data is saved, it should be stored safely.

Chapter 03: Analysis (use case model)

3.1 Actor Identification

For the Neural Network Digit Classification System, the following actors and use cases have been identified:

Actors:

- **User:**
  This is the person who interacts with the system to classify handwritten digits. The user can upload digit images and receive classification results.

- **Model:**
  The internal component responsible for processing the input and providing digit classification using the trained neural network.

Primary Use Cases:

- **Upload Digit Image:** The user uploads an image of a handwritten digit.
- **Preprocess Image:** The system processes the uploaded image to convert it into a suitable format for classification.
- **Classify Digit:** The neural network classifies the processed image and predicts the digit.
- **Display Result:** The system shows the predicted digit to the user.

3.2 Use case Diagram

Candidate Actors:

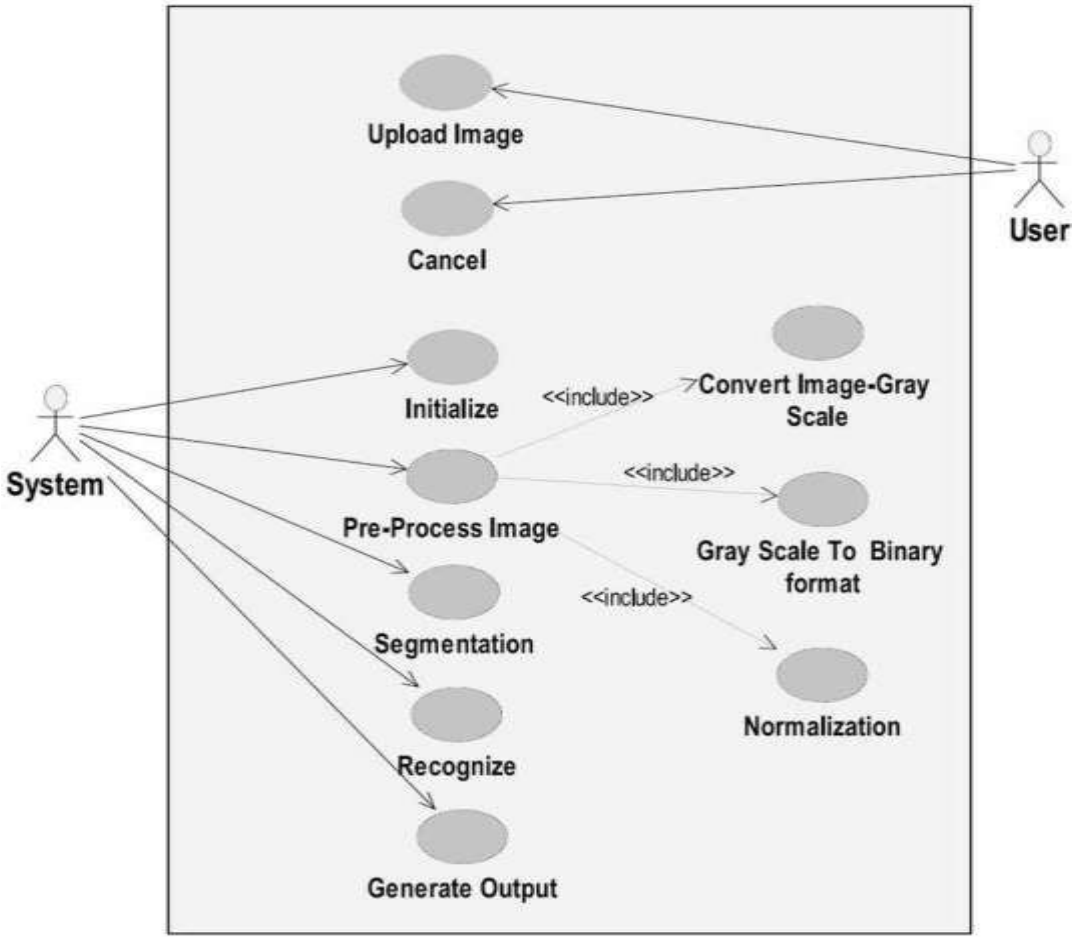
- User
- System (Neural Network Model)

Candidate Use Cases:

- Upload Image
- Preprocess Image
- Classify Digit
- Display Result

Use Case Relationships:

- **User** → Upload Image → System
- **System** → Preprocess Image → Classify Digit → Display Result



3.3 Event Flow

This section explains the steps for each main action in the system. Each use case shows what the user does and how the system reacts.

Upload Image

- User selects a digit image from their device.
- User uploads the image to the system.
- System stores the image for processing.

Preprocess Image

- System converts the image to grayscale.
- System resizes the image to 28x28 pixels.
- System normalizes pixel values for the neural network.

Classify Digit

- The preprocessed image is sent to the neural network.
- The model analyzes the image and predicts a digit.
- System selects the digit with the highest confidence.

Display Result

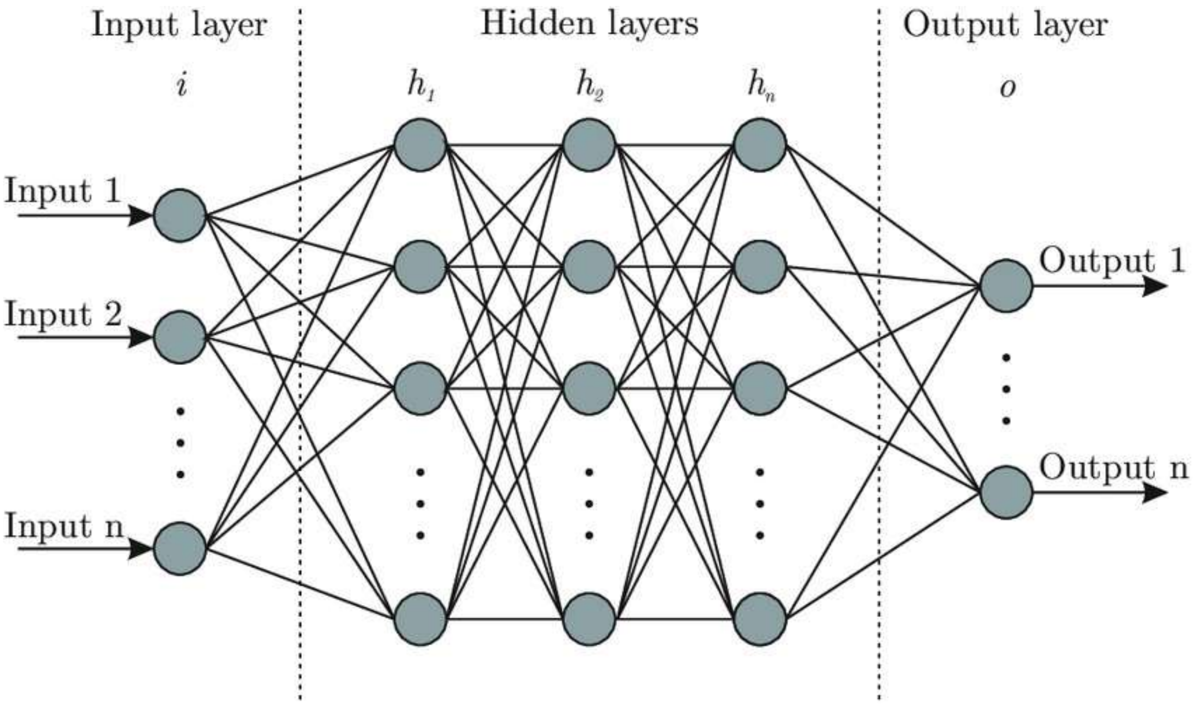
- System shows the predicted digit to the user.
- Shows a confidence score or probability.

Chapter 4: System Design

4.1 Architecture Diagram

The architecture diagram provides a high-level overview of the system components and their interactions. The system is divided into layers including input layer, hidden layers, and output layer.

Input Layer (784 neurons)
↓
Hidden Layer 1 (128 neurons)
↓
Hidden Layer 2 (64 neurons)
↓
Output Layer (10 neurons)

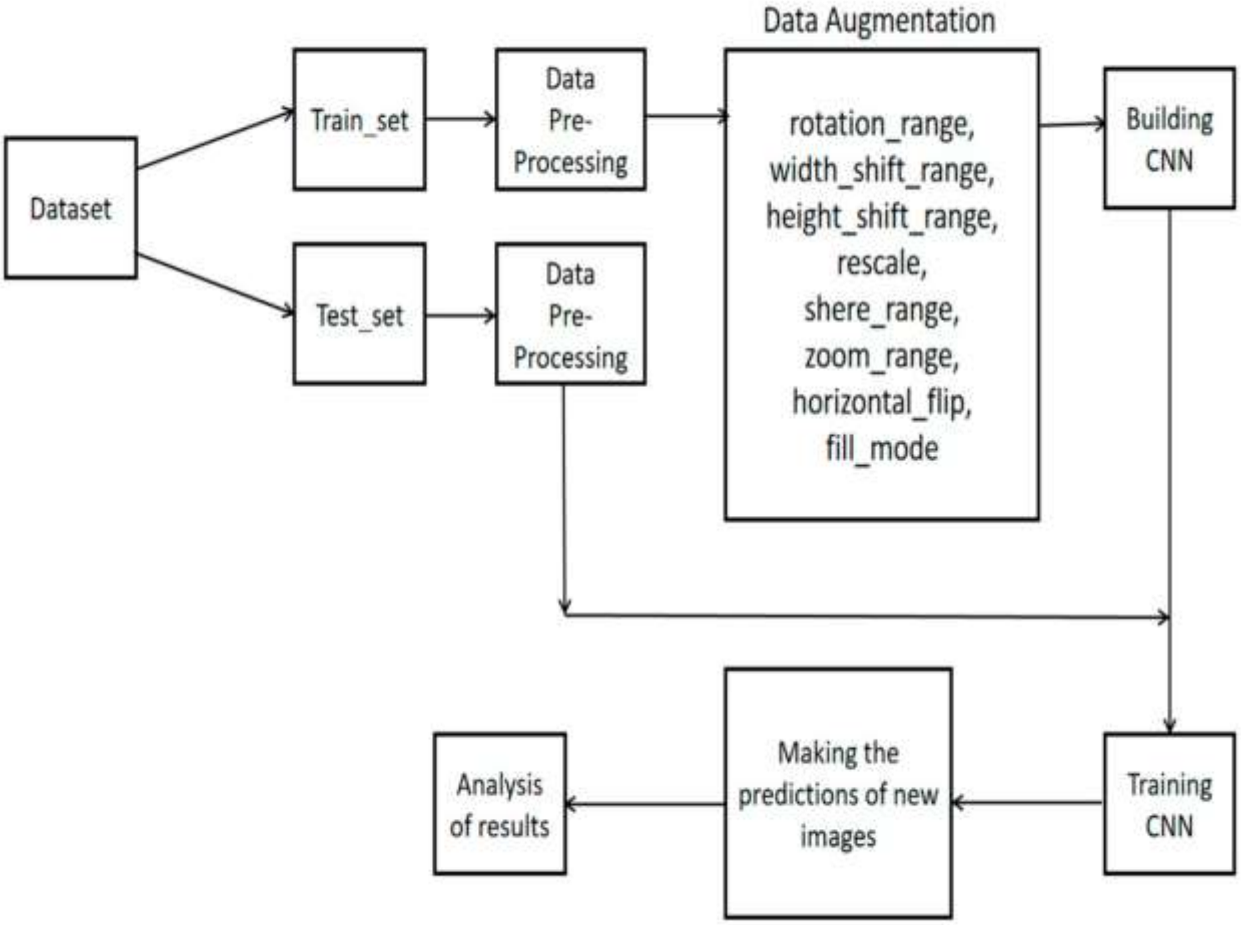


4.2 Data Flow Diagram (DFD)

The DFD represents the flow of data between different modules of the system.

DFD Levels:

- **Level 0:** Shows the interaction between the user and the system.
- **Level 1:** Details the processes such as image preprocessing, model prediction, and displaying the output.

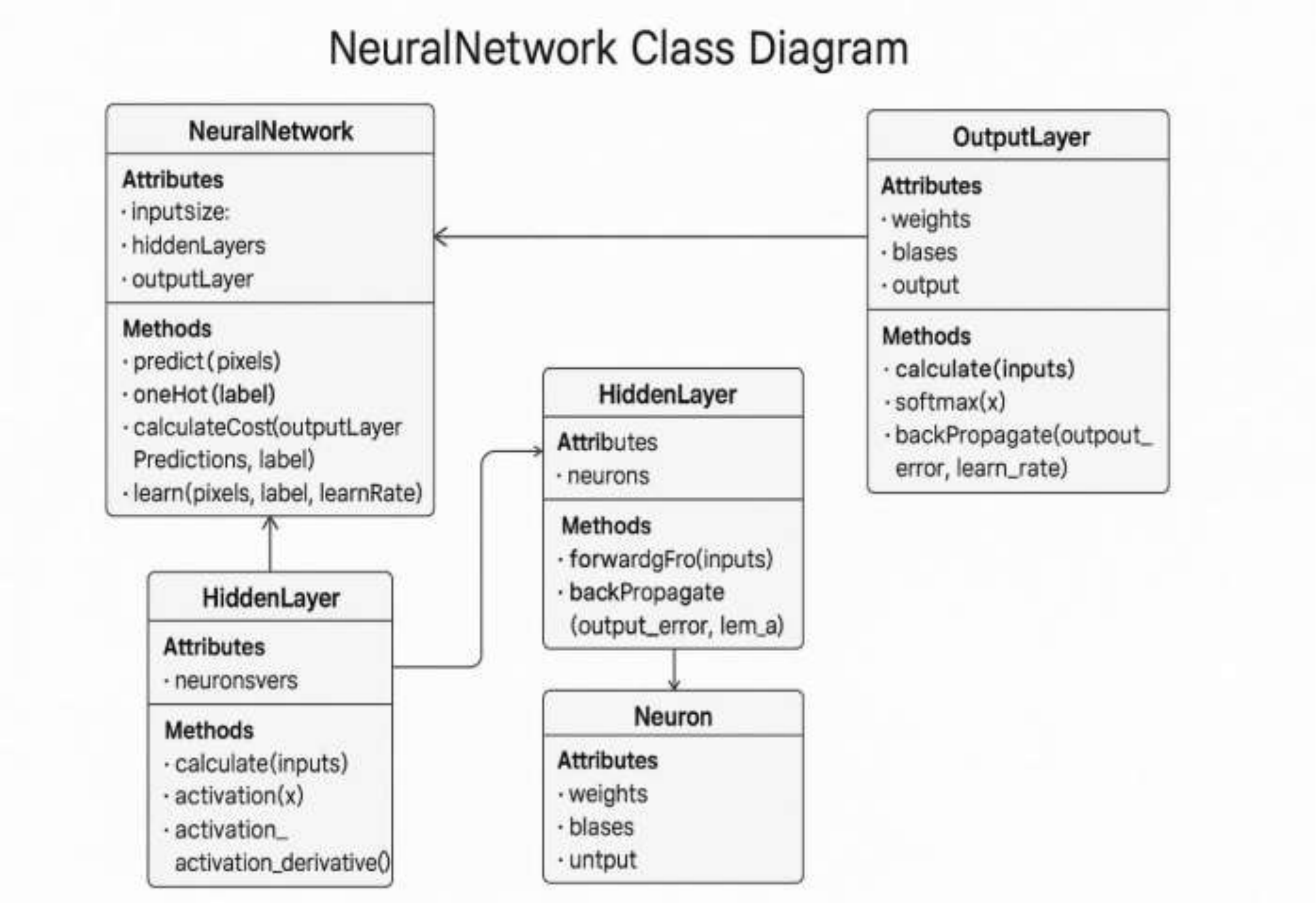

Data Augmentation
Dataset
Train_set
Test_set
Data Pre-Processing
Data Pre-Processing
rotation_range, width_shift_range, height_shift_range, rescale, shere_range, zoom_range, horizontal_flip, fill_mode
Building CNN
Training CNN
Making the predictions of new images
Analysis of results

4.3 Class Diagram

A Class Diagram is a part of UML (Unified Modeling Language). It shows the structure of
The system by displaying:

- The **main classes** used in the project
- Their **attributes** (data they hold)
- Their **methods** (what they can do)
- The **relationships** between different classes

In this project, the class diagram represents how different parts of the Neural Network Digit Classification system interact with each other.



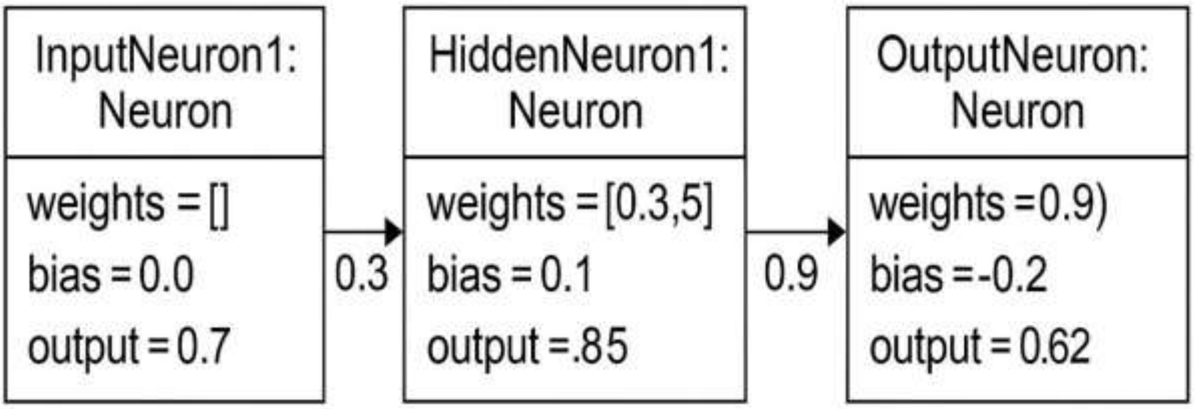
4.4 Object Diagram

An object diagram is like a snapshot of the system at one moment. It shows the real objects (not just the blueprint like a class diagram) and how they are connected and interact with each other.

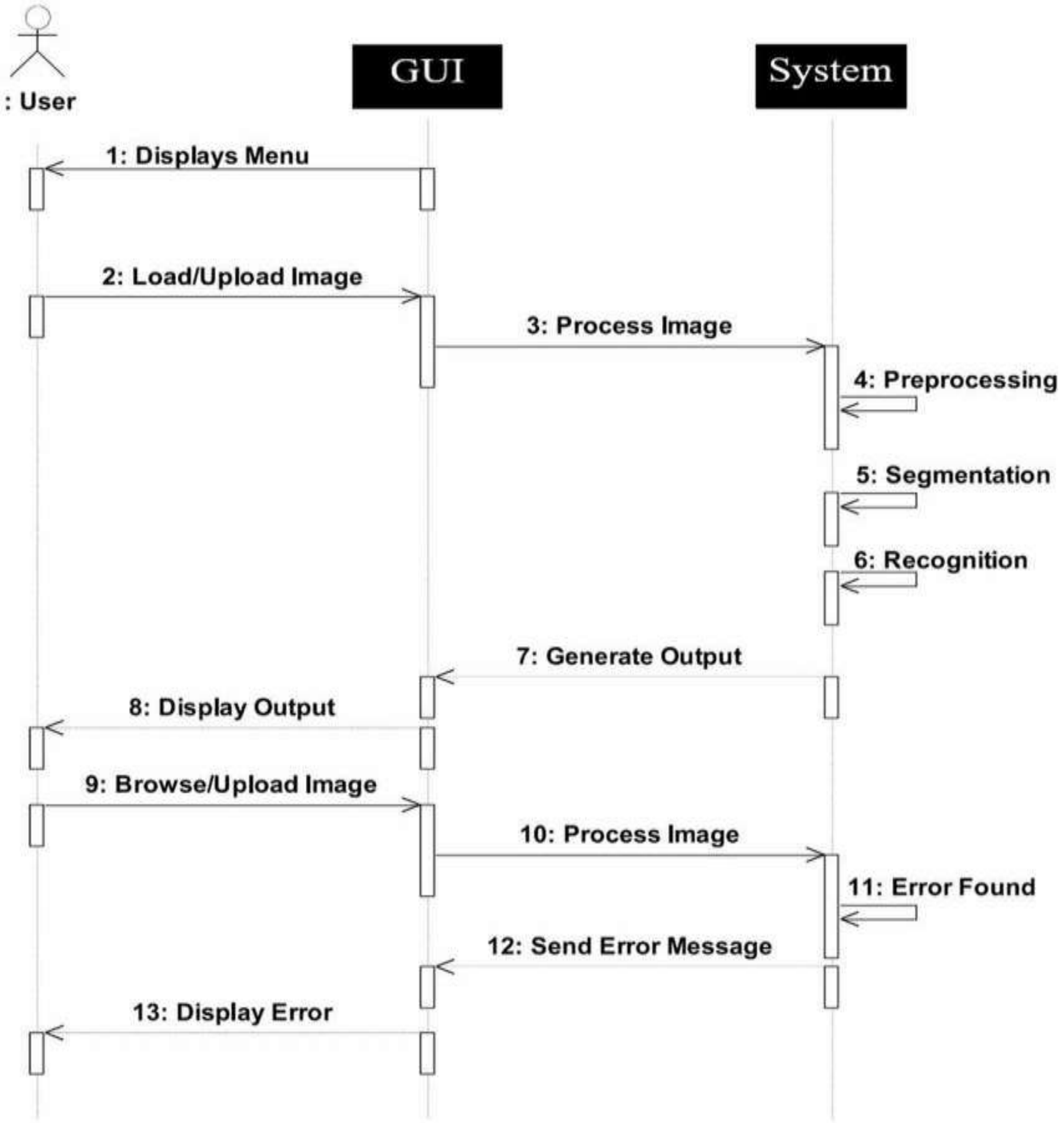
While a class diagram shows what kinds of things the system has (like templates), the object diagram shows the actual things that are being used at a specific time.

It helps you:

- Understand how the system works at a specific time.
- See the real values of data (like the actual image uploaded or the result predicted).
- Show how different parts of the system talk to each other.



4.5 Sequence Diagram

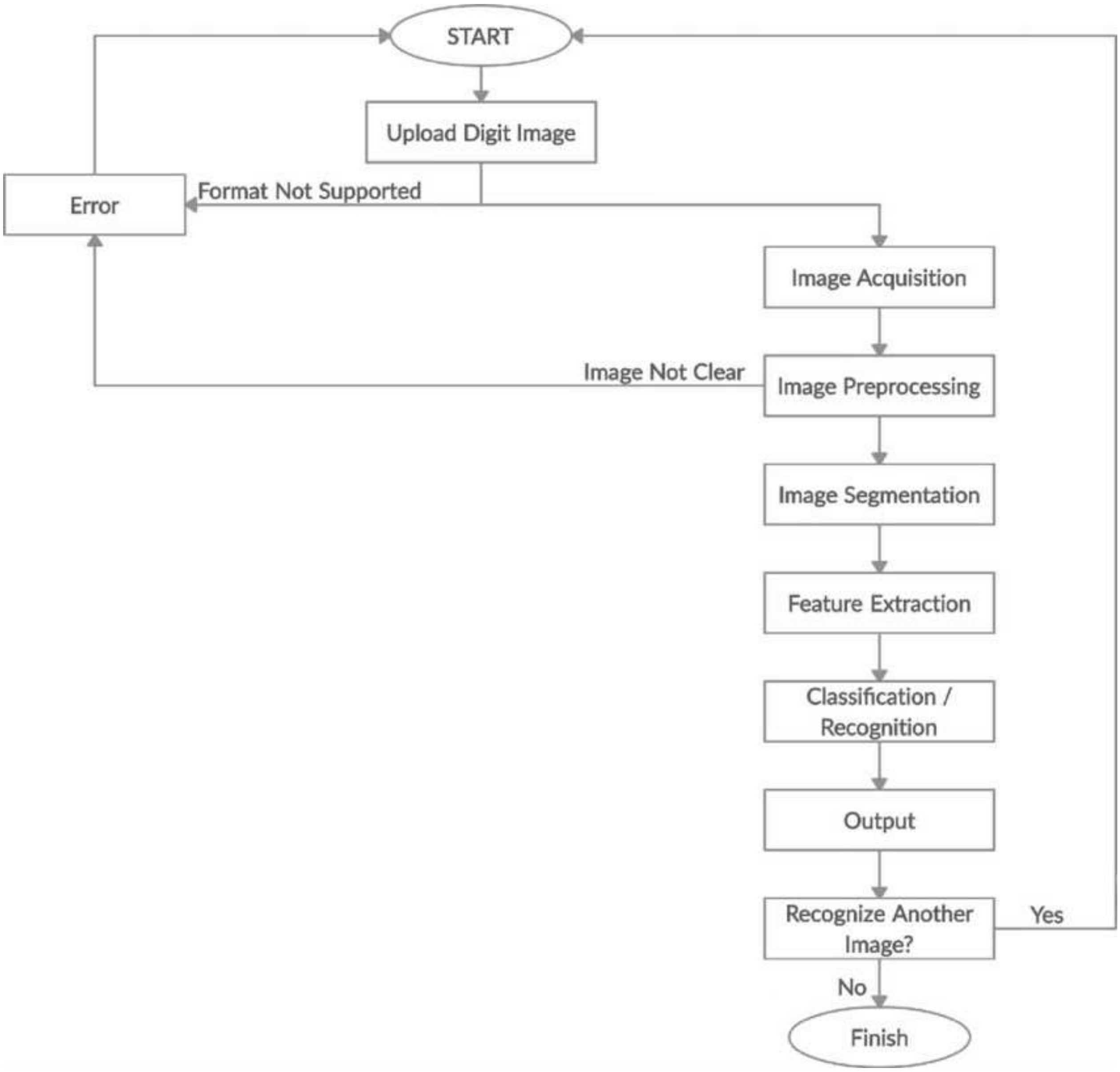


4.6 Activity Diagram

The activity diagram shows the flow of activities in the system.

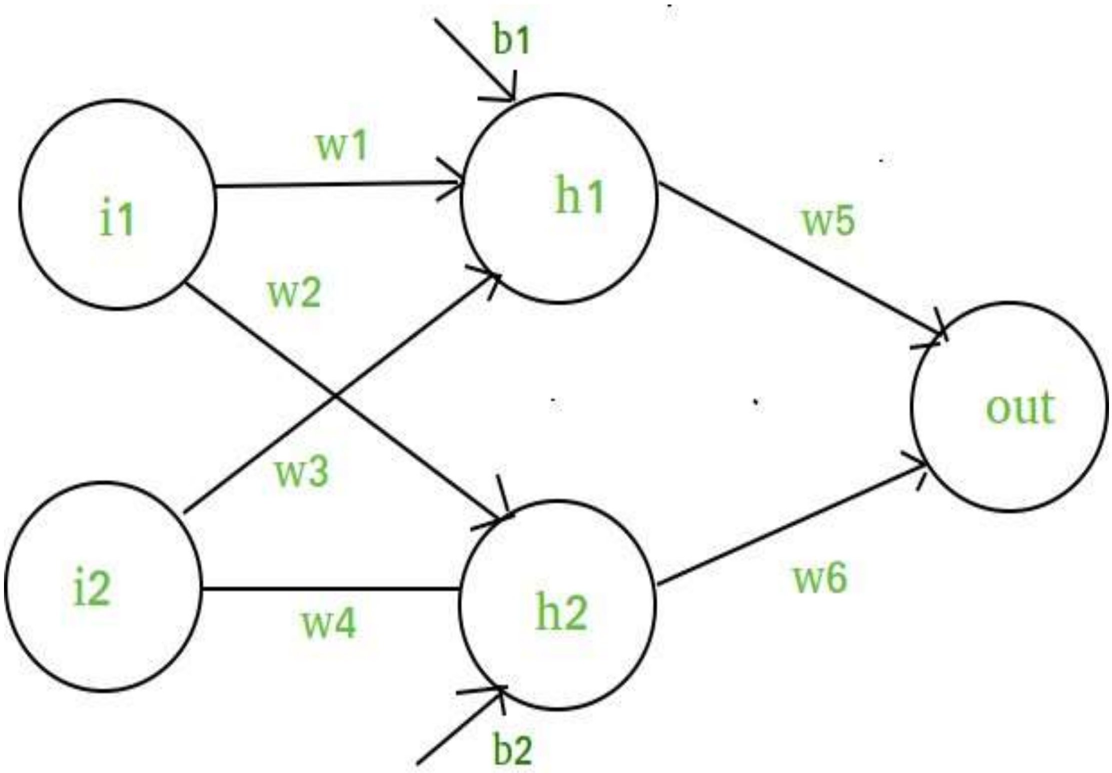
Flow:

- Start
- Upload image
- Process image
- Predict digit
- Show result
- End



4.7 Collaboration Diagram

The collaboration diagram shows how different parts (objects) of the system talk to each other during the process.



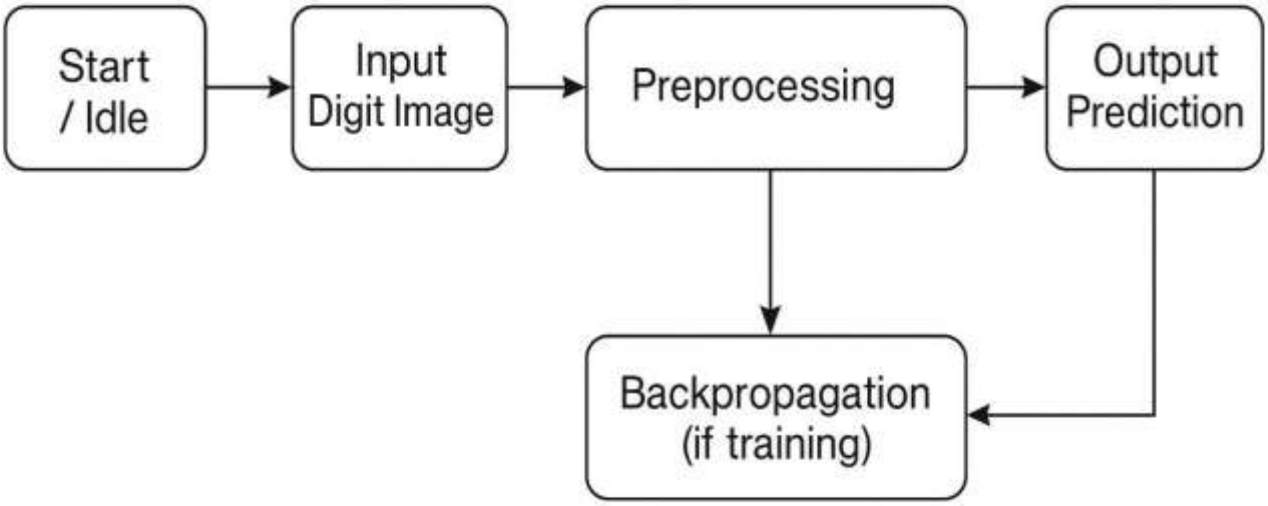
4.8 State Transition Diagram

This diagram shows the different states the system can be in and how it moves from one state to another.

Example states:

- Idle state
- Image uploaded
- Image processing

- Predicting digit
- Showing result



Chapter [illegible]: Implementation

This chapter explains how the system was actually built and how its different parts work together. It covers the internal components, their roles, and how they are connected to complete the task of digit classification.

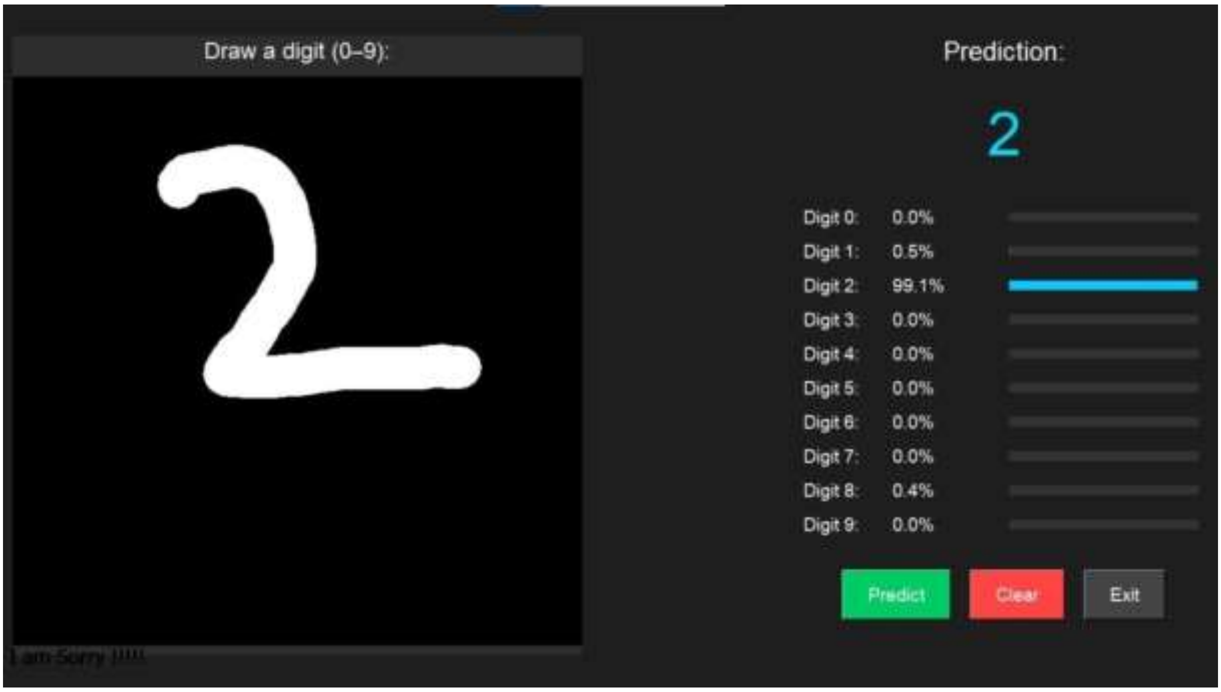
Chapter No 5: Implementation

5.1 Component Diagram

A Component Diagram is a UML diagram that shows the main parts (components) of the system and how they interact. In this project, the component diagram explains the structure of the software system used for digit classification.

5.1.1 GUI Drawing App

- This is the part where the **user draws a digit** using a mouse or finger.
- It gives the system a **handwritten digit as input**.
- It includes a drawing area, a "Predict" button, and a "Clear" button.
- The digit drawn is passed to the next component for processing.



```
2

Digit 0:   0.0%
Digit 1:   0.5%
Digit 2:   99.1%
Digit 3:   0.0%
Digit 4:   0.0%
Digit 5:   0.0%
Digit 6:   0.0%
Digit 7:   0.0%
Digit 8:   0.4%
Digit 9:   0.0%
```

5.1.2 Digit Preprocessor

- This part cleans and prepares the digit image before giving it to the neural network.
- It does the following tasks:
  - **Resize** the image to 28x28 pixels (like MNIST format)
  - **Convert** to black and white (grayscale)

  - **Normalize** the pixel values (between 0 and 1)
  - **Invert** the image if the background is black and digit is white

5.1.3 Trained Network

- This is the brain of the system. It is a Neural Network model that has already been trained to understand handwritten digits.
- It takes the cleaned image from the preprocessor and predicts the number (0 to 9).
- The network is trained using a mnist dataset.

5.1.4 Model Loader

- This part loads the trained neural network model when the system starts.
- The model is saved in a file (like .h5 or .pkl) after training.
- It ensures we don't need to train the model every time.

5.1.5 Network Trainer

- This component is used only during the training phase**.**
- It adjusts the internal weights based on mistakes, so that it improves over time.

5.1.6 Dataset Component

- This is the collection of digit images used for training and testing.
- We use the MNIST Dataset which contains:
  - 60,000 images for training
  - 10,000 images for testing

How Components Work Together

Here's how the system works step-by-step:

1. The user **draws a digit** in the GUI Drawing App.

2. The **Digit Preprocessor** cleans the image.

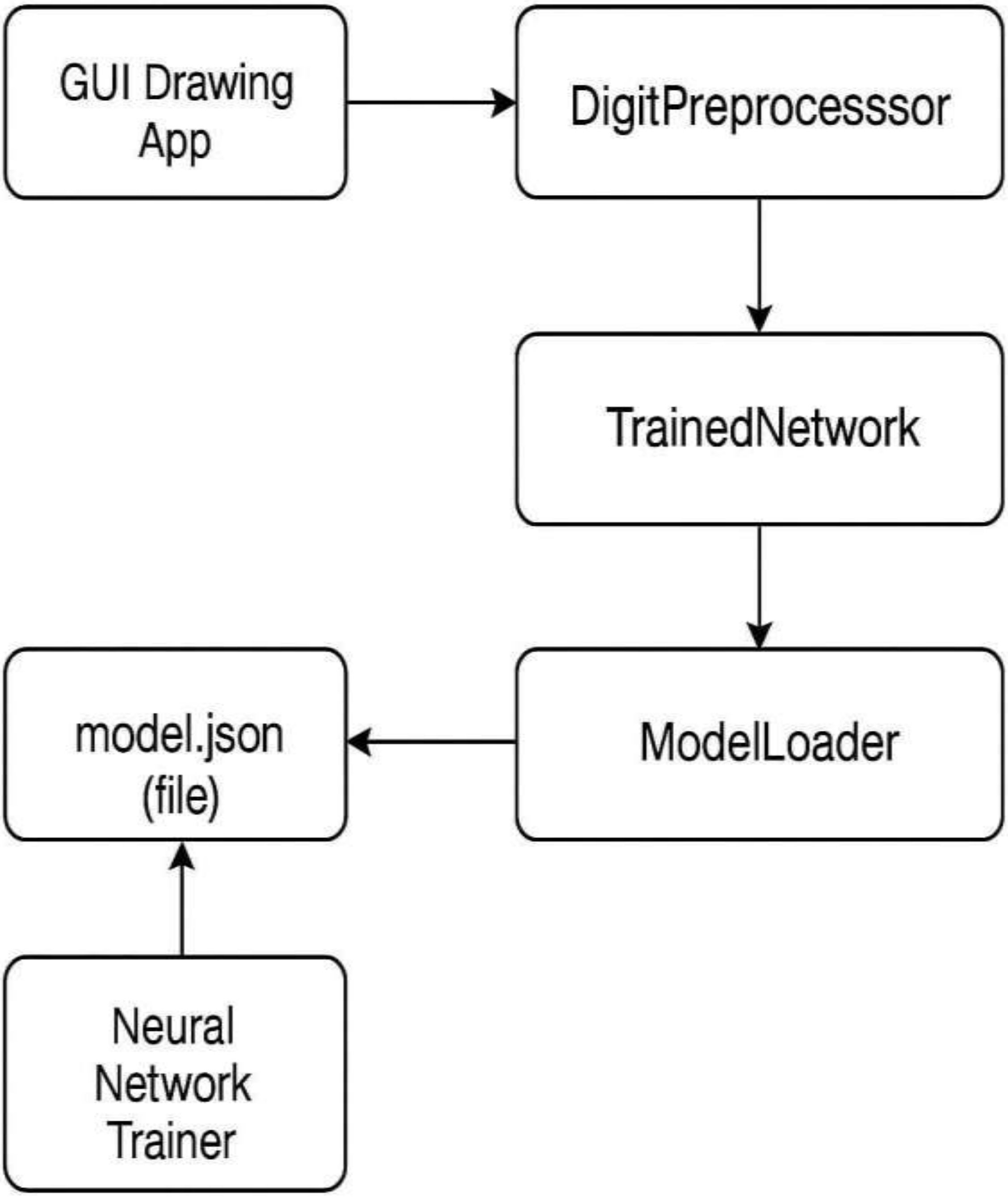
3. The **Model Loader** loads the Trained Network.

4. The cleaned digit is sent to the **Trained Network**.

5. The network **predicts the digit**.

6. The result is shown to the user.

5.2 Deployment Diagram



This chapter explains how we tested our digit recognition system and what results we got. We checked how well the system works, how accurate it is, how fast it predicts, and how much error (cost) it makes during training.

Chapter No 6: Testing (Software Quality Attributes)

6.1 Testing Strategy

To make sure the system is working properly, we tested it using the following methods:

1. Unit Testing

- Each part of the system was tested separately.
- Example: We tested the digit preprocessor to check if it correctly resizes and normalizes images.

2. Integration Testing

- After testing individual components, we tested how well they work together.
- Example: We checked if the drawing app sends the digit to the neural network smoothly.

3. System Testing

- We tested the full system from start to end.
- We drew digits and checked if the system gave the correct prediction.

4. Test Dataset

- We used the MNIST test dataset which has 10,000 images of handwritten digits that the model had never seen before.
- This helps us check how well the model works on real-life data.

6.2 Accuracy Metrics

We used different methods to check how accurate and reliable our model is:

1. Accuracy

- This is the percentage of correct predictions made by the system.

- **Result**: Our model gave around 90–95% accuracy on the MNIST test dataset.

2. Confusion Matrix

- This shows how many digits were correctly or incorrectly predicted.
- For example, it tells us if the system confused a "4" with a "9".

3. Loss and Accuracy Curves

- These graphs show how the model improved during training.
- **Loss**: Lower is better. It shows how many mistakes the model is making.
- **Accuracy**: Higher is better. Shows how often the model gets it right.

4. Prediction Speed

- The model predicted each digit in less than 1 second.
- This makes it fast and usable in real-time applications.

6.3 Visualization of Results

We also used images and graphs to show how the model performs:

1. Sample Predictions

- We showed some test digit images along with the predicted results.
- Example: A drawn “3” image → Output: “Predicted: 3 (95% confidence)”

2. Accuracy Graph

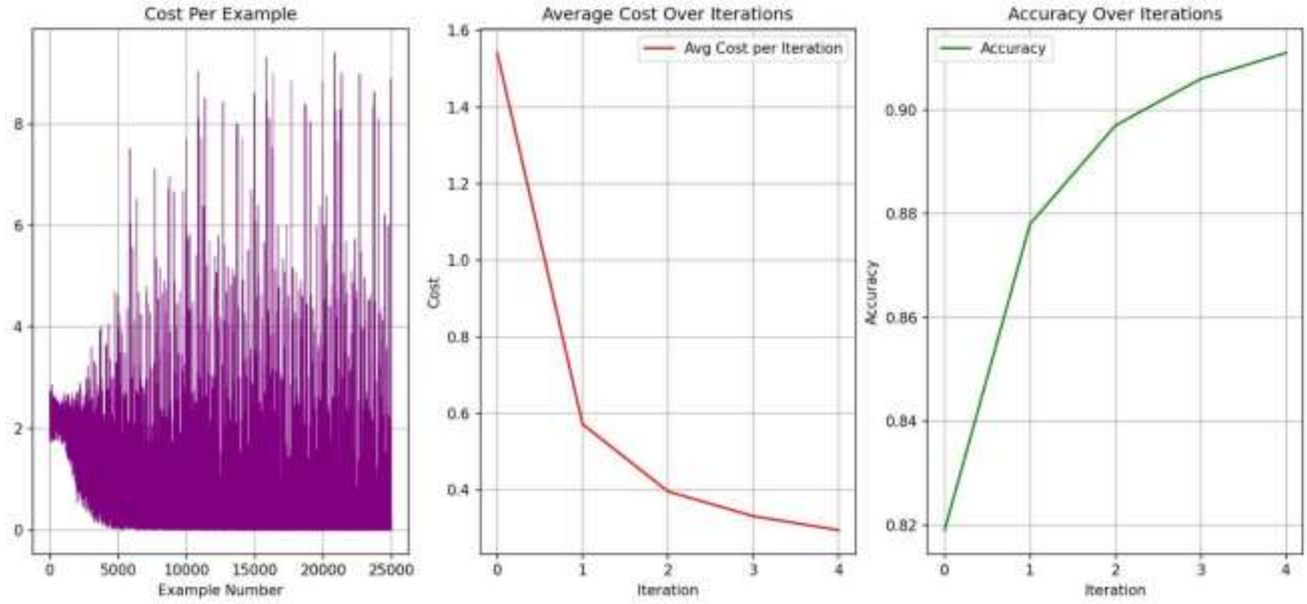
- Shows how the accuracy improved after each training step.
- Helps understand how fast the model is learning.

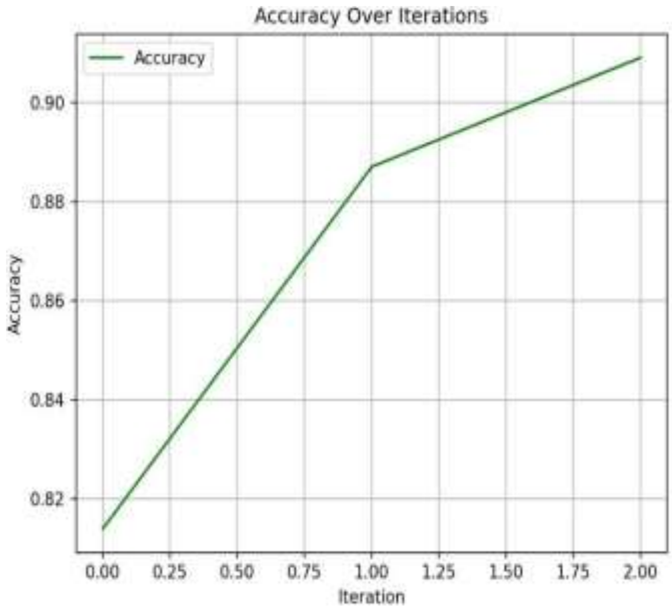
3. Cost Graph

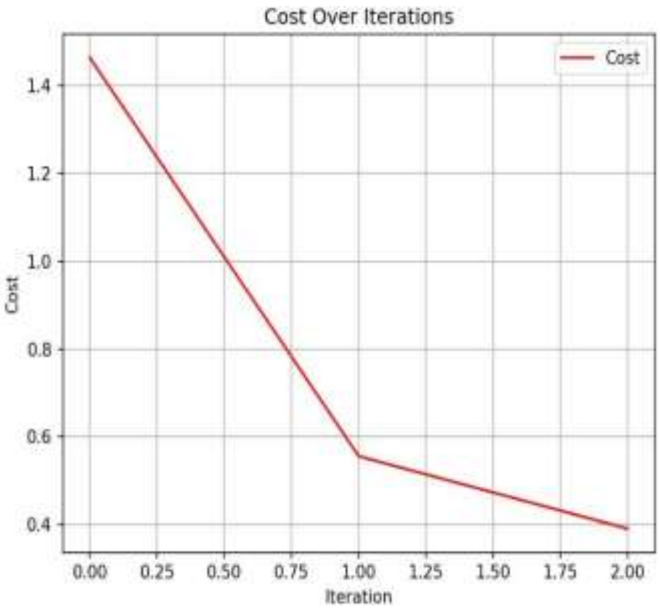
- Shows how the model’s errors decreased over time.

4. Confusion Matrix

- A square chart showing which digits were predicted correctly and where the model got confused.
- Example: Sometimes it mistakes a “9” for a “4”.




Accuracy Over Iterations
Accuracy
0.90
0.88
0.86
0.84
0.82
Accuracy
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Iteration


Cost Over Iterations
Cost
1.4
1.2
1.0
0.8
0.6
0.4
Cost
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Iteration

Chapter 7: Tools and Technologies

In this chapter, we describe the tools, software, and technologies used to build and run our digit classification system. These tools helped us write the code, train the model, and test the system effectively.

7.1 Programming Language

We used the Python programming language for this project.

Because,

- Python is easy to write and read.
- It has many libraries for machine learning and image processing.
- It is widely used in AI, data science, and neural networks.

Main Python Libraries Used:

Library	Use / Purpose
NumPy	Used for mathematical operations, arrays, and matrix calculations (core part of neural network math)
Pandas	Used to organize and analyze data (like training logs or accuracy reports)
Pillow (PIL)	Used for image loading, resizing, and converting images to grayscale
Tkinter	Used to build the GUI where the user can draw a digit using the mouse

7.2 Operating Environment

This section describes the system and software setup used for running the project.

Operating System

- Windows 10 Pro (64-bit)

Laptop Model

- Lenovo ThinkPad

Processor

- Intel Core i7

RAM

- 12 GB RAM

Storage

- 500 GB SSD

Item	Details
Operating System	Windows 10 Pro
Laptop Model	Lenovo ThinkPad
Processor	Intel Core i7
RAM	12 GB
Storage	500 GB SSD

Chapter No 8

8.1 References

1. Michal Nielsen, Neural network and deep learning, Available at:
http://neuralnetworksanddeeplearning.com

2. 3Blue1Brown, neural network (YouTube video), 2017Available at:
https://www.youtube.com/watch?v=aircAruvnKk

3. Sebastian lague, neural networks (YouTube playlist), 2021 Available at:
https://www.youtube.com/playlist?list=PLFtAvWs\

4. Python software foundation, tkinter documentation Available at:
https://docs.python.org/library/tkinter.html

5. Pillow Developers, Pillow documentation Available at:
https://pillow.org/readthedocs.io/

6. NumPy Developers, documentation Available at:
https://numpy.org/doc/

7. PyTorch Developers PyTorch Documentation Available at:
https://www.kaggle.com/dataset/hojjatk/mnist_dataset

8. Yann Lecun, MNIST database of handwritten digits. Available at:
https://www.kaggle.com/datasets/crawford/emnist

9.Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik, EMNIST Available at:
https ://www.kaggle.com //arxiv.org/abs.

Chapter 9: Appendices

9.1 Appendix – User Documentation

This section explains how to use the Neural Network Digit Classification system step-by-step.

Step 1 – Open the Application

- Run main.py in Python or click on the application executable file.

Step 2 – Upload an Image

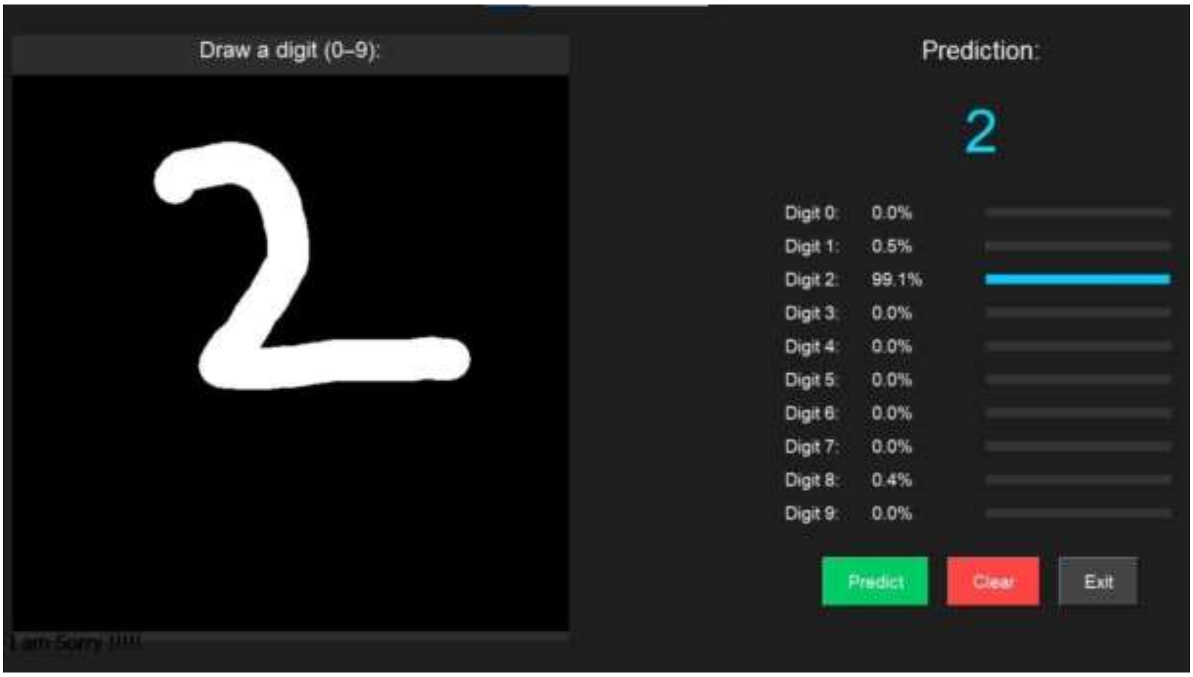
- Click the "Browse" button and select a handwritten digit image (28×28 pixels recommended).

Step 3 – Predict the Digit

- After uploading the image, click the "Predict" button.
- The neural network will process the image and display the predicted digit.

Step 4 – Clear

- Use the "Clear" button to reset and upload another image.



```
Prediction:

2

Digit 0:   0.0%
Digit 1:   0.5%
Digit 2:   99.1%
Digit 3:   0.0%
Digit 4:   0.0%
Digit 5:   0.0%
Digit 6:   0.0%
Digit 7:   0.0%
Digit 8:   0.4%
Digit 9:   0.0%
```

I am Sorry !!!!

Accuracy Results

The trained model achieved the following accuracy:

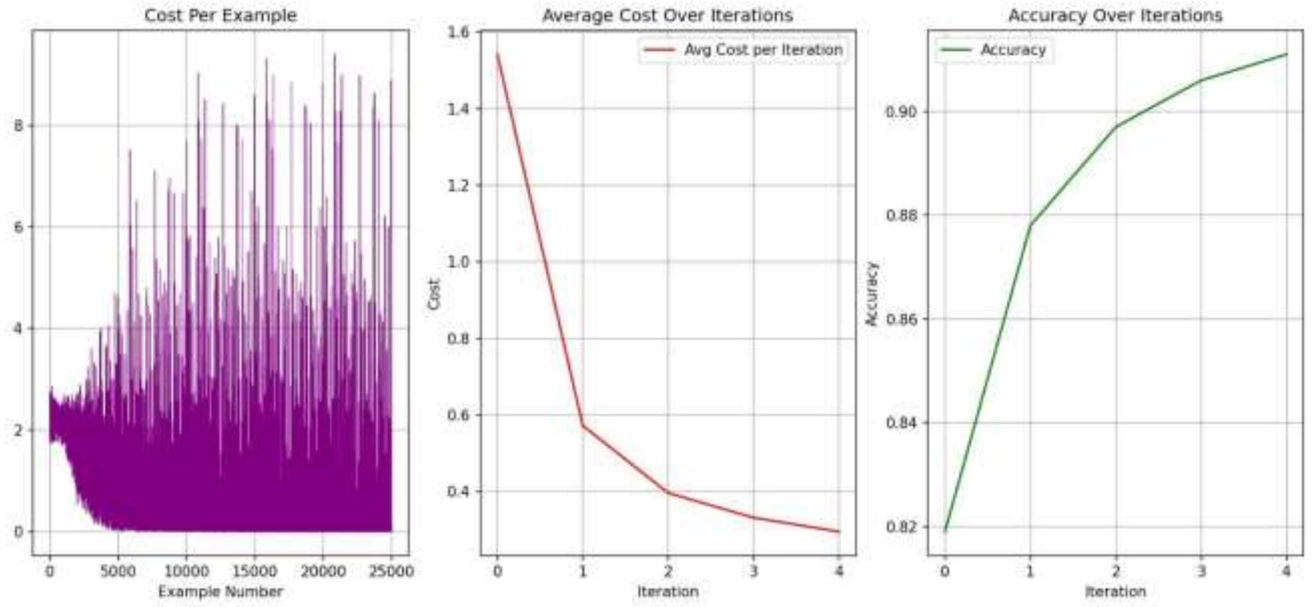
- **Training Accuracy:** 85-90%
- **drawing Accuracy:** 70-75%
- **Test Accuracy:** 90-95%

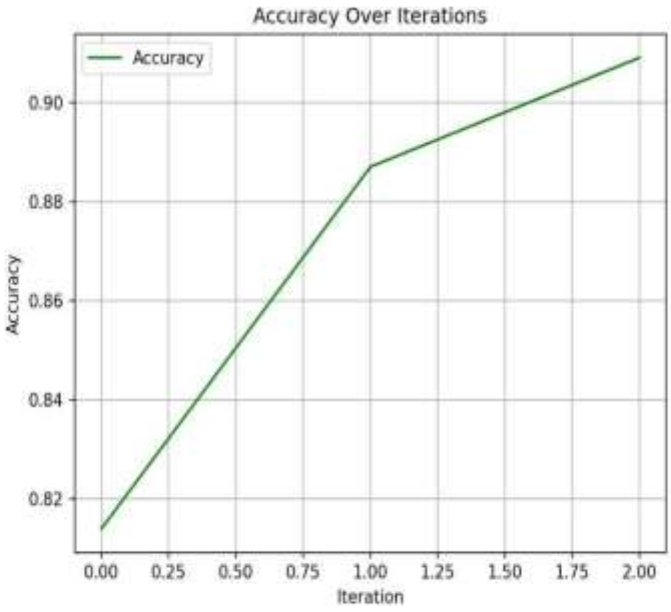
This shows that the model is highly effective in predicting handwritten digits.

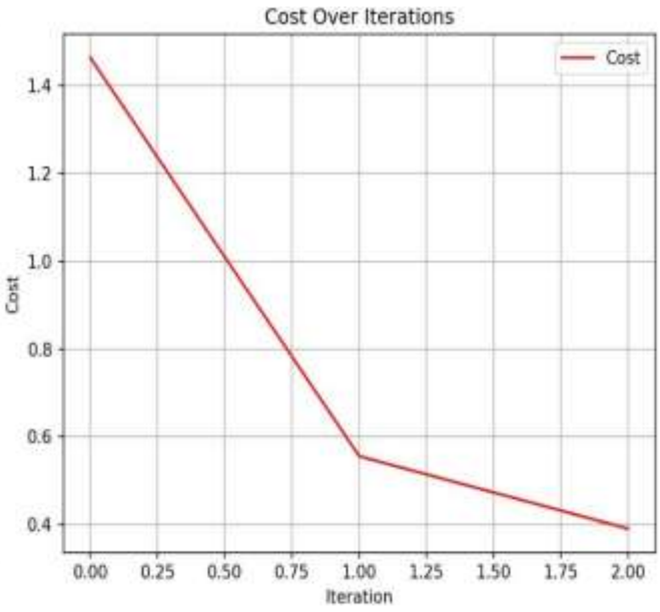
Visualization of Results (Graph)

To better understand the model's performance, the following graphs were generated:

- **Accuracy Graph:** Shows how training and validation accuracy improved over epochs.
- **Loss Graph:** Shows how the error decreased during training.




Accuracy Over Iterations
Accuracy
0.90
0.88
0.86
0.84
0.82
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Iteration
Accuracy


Cost Over Iterations
Cost
1.4
1.2
1.0
0.8
0.6
0.4
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Iteration
Cost

9.2 Appendix – Supervisory_Committee

For Approval of any two Consultant Teachers	
Teacher Consulted **Name:** ________________ **Designation:** ________________ **Comments:**	**Teacher Consulted** **Name:** ________________ **Designation:** ________________ **Comments:**
Signature: ________________________	**Signature:** ________________________

For Official use only:

Date: ____

Approved By: ____

Group ID: ____

Meeting Required: Date: _________________________________

Time: ________________

Place: ________________

REJECTED Remarks:

____

____

____

Project Title (if revised): ______________________

Project Coordinator: ________________________

Project Supervisor: _________________________

Thank You