



GREEN UNIVERSITY OF BANGLADESH
Department of Computer Science and Engineering (CSE)
Semester: Summer 2025 B.Sc. in CSE (Day)

Handwritten Digit Recognition Using Artificial Neural Networks

Course Title: Machine Learning Lab
Course Code: CSE_412
Section: 221_D18

Students Details

| Name | ID |
|-------------------|-----------|
| Md. Monirul Islam | 221002154 |
| Prince Sarker | 221002240 |

Submission Date: 24/08/2025
Course Teacher's Name: Sudip Chandra Ghoshal

[For teachers use only: **Don't write anything inside this box**]

| <u>Lab Project Status</u> | |
|---------------------------|-------------------|
| Marks: | Signature: |
| Comments: | Date: |

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 2 |
| 1.1 | Overview | 2 |
| 1.2 | Motivation | 2 |
| 1.3 | Problem Statement | 2 |
| 1.3.1 | Complex Engineering Problem | 3 |
| 1.4 | Objectives | 3 |
| 1.5 | Application | 4 |
| 2 | Implementation of the Project | 5 |
| 2.1 | Introduction | 5 |
| 2.2 | Tools and Technology | 5 |
| 2.3 | Implementation | 5 |
| 3 | Performance Evaluation | 16 |
| 3.1 | Output | 16 |
| 4 | Conclusion | 17 |
| 4.1 | Discussion | 17 |
| 4.2 | Limitations and Future Work: | 17 |
| 4.3 | Resources | 17 |

1. Introduction

1.1 Overview

This project is about teaching a computer to recognize numbers written by hand. We use a dataset called MNIST, which has thousands of small images of digits from 0 to 9. Each image is 28×28 pixels in black and white. We build an Artificial Neural Network (ANN) that can look at these images and decide which digit they represent. Before giving the images to the model, we make them ready by resizing, converting to grayscale, and scaling the pixel values between 0 and 1. The ANN then learns from many examples and improves its ability to guess the correct digit. After training, we test it with new images to see how well it performs. This project shows how machines can learn patterns from data and use them to make predictions.

1.2 Motivation

Handwritten digit recognition is important because numbers are still written manually in many areas. Typing them into a computer is slow and can cause errors. Automating this process saves time, reduces mistakes, and improves efficiency. Using Artificial Neural Networks (ANN), computers can learn patterns in handwritten digits and recognize them accurately.

- ✓ Reduce time and errors in manual data entry.
- ✓ Automate tasks like reading cheques, postal codes, and forms.
- ✓ Enable faster and more efficient banking, postal, and educational processes.
- ✓ Demonstrate practical applications of machine learning and ANN.

1.3 Problem Statement

Handwritten digits vary greatly from person to person in style, size, thickness, and orientation. This variation makes it challenging for computers to recognize digits without proper training. The goal is to build a system that can learn from examples and correctly classify digits despite these differences. The MNIST dataset provides a standard set of images to train and test such a system.

We aim to develop a tool that:

- ✓ Handwritten digits have diverse shapes and styles.
- ✓ Computers cannot recognize digits accurately without training.
- ✓ The system should generalize well to unseen handwriting.
- ✓ Use ANN to achieve high accuracy in classification tasks.

1.3.1 Complex Engineering Problem

The following Table 1.1 shows the main project attributes that this project focuses on. Only the attributes that are relevant to our work are filled in the right column.

Table 1.1: Summary of the attributes touched by the mentioned projects

| Name of the P Attributes | How the Project Addresses It |
|--|--|
| P1: Depth of knowledge required | Requires understanding of machine learning concepts, artificial neural networks, activation functions, and image preprocessing techniques. The project applies these concepts to classify handwritten digits accurately. |
| P3: Depth of analysis required | Involves analyzing model performance using accuracy, loss, and confusion matrix. Also requires evaluating the effect of preprocessing and ANN architecture on results. |
| P7: Interdependence | The performance of the ANN depends on proper preprocessing, network design, and training parameters. Each step is interconnected and affects overall accuracy. |

1.4 Objectives

The main objectives of the project are:

- ✓ To build an ANN model to recognize handwritten digits.
- ✓ To prepare the images so the model can understand them (resize, grayscale, normalize).
- ✓ To train the model using the MNIST dataset.
- ✓ To test the model on new images to check accuracy.
- ✓ To show that ANN can be used for real-world number recognition tasks like bank checks or postal codes.

1.5 Application

Handwritten digit recognition using ANN has many practical uses in everyday life and industry. Some common applications include:

- ✓ **Banking:** Reading handwritten amounts on cheques and deposit slips automatically.
- ✓ **Postal Services:** Sorting letters and parcels by recognizing handwritten postal/ZIP codes.
- ✓ **Education:** Digitizing handwritten exam sheets, answer scripts, or notes.
- ✓ **Data Entry Automation:** Converting handwritten forms into editable digital text efficiently.
- ✓ **Mobile Apps:** Recognizing handwritten numbers in note-taking apps or calculator tools.

2. Implementation of the Project

2.1 Introduction

This project implements a full pipeline for handwritten digit recognition using the MNIST dataset. The system trains a compact Artificial Neural Network (ANN/CNN) to classify 28×28 grayscale images of digits (0–9), evaluates the model on a held-out test set, and deploys it via a Pygame interface so users can draw a digit and receive a real-time prediction.

The implementation emphasizes:

- Reproducible preprocessing (normalization to [0,1], consistent centering/cropping).
- A small, accurate ANN that achieves high accuracy on MNIST.
- Robust evaluation (accuracy, per-class metrics, confusion matrix, error analysis).
- Usability via an interactive digit board powered by Keras + OpenCV + Pygame.

2.2 Tools and Technology

The development of the project relies on a combination of programming tools, libraries, and environments to ensure effective implementation, scalability, and usability. The following technologies were used:

- **Python:** The core programming language for implementing data processing and clustering logic.
- **TensorFlow / Keras:** Model definition, training, checkpointing.
- **Pandas:** For reading and manipulating structured data from CSV files.
- **OpenCV (cv2):** Image preprocessing for the UI (cropping, resizing, thresholding).
- **Pygame:** An interactive drawing surface for inference.
- **NumPy:** For efficient numerical operations and array handling.
- **Matplotlib & Seaborn:** For plots (loss/accuracy curves, confusion matrix, t-SNE/PCA).
- **Scikit-learn:** Metrics (classification report, confusion matrix), optional K-Means.
- **Jupyter Lab and Google Colab:** For cloud-based and local development environments.

Dataset

- **MNIST:** 60,000 train images and 10,000 test images of digits, 28×28 grayscale.

2.3 Implementation

Importing Required Libraries

```
In [25]: import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Flatten
import matplotlib.pyplot as plt
```

Loading and Inspecting the MNIST Dataset with Keras

```
In [26]: (X_train,y_train),(X_test,y_test) = keras.datasets.mnist.load_data()
print("X_train Shape: ",X_train.shape)
print("y_train Shape: ",y_train.shape)
print("X_test Shape: ",X_test.shape)
print("y_test Shape: ",y_test.shape)
```

```
X_train Shape: (60000, 28, 28)
y_train Shape: (60000,)
X_test Shape: (10000, 28, 28)
y_test Shape: (10000,)
```

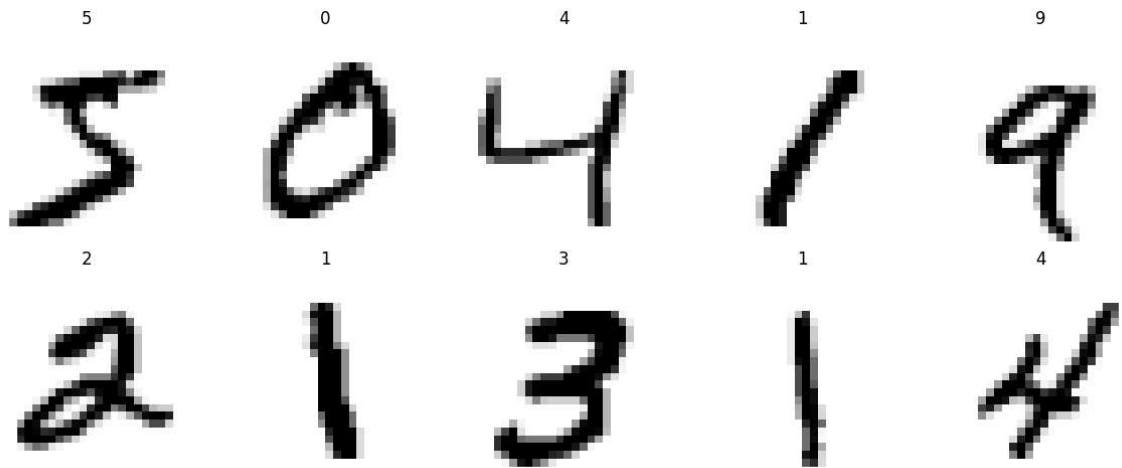
Visualizing Sample Images from X_train with Corresponding Labels from y_train

```
In [10]: import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_train[i], cmap="binary")
    plt.title(y_train[i])
    plt.axis("off")

plt.tight_layout()
plt.show()
```



```
In [11]: X_train = X_train/255
X_test = X_test/255
```

Simple Feedforward Neural Network for 28x28 Input Images

```
In [13]: from tensorflow.keras.layers import Input
model = Sequential([
    Input(shape=(28, 28)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(32, activation='relu'),
    Dense(10, activation='softmax')
])
```

Summary of the Fully Connected Neural Network Model

```
In [14]: model.summary()
```

Model: "sequential_1"

| Layer (type) | Output Shape | |
|---------------------|--------------|--|
| flatten_1 (Flatten) | (None, 784) | |
| dense_3 (Dense) | (None, 128) | |
| dense_4 (Dense) | (None, 32) | |
| dense_5 (Dense) | (None, 10) | |



Total params: 104,938 (409.91 KB)


Trainable params: 104,938 (409.91 KB)


Non-trainable params: 0 (0.00 B)


```
In [15]: model.compile(  
    loss='sparse_categorical_crossentropy',  
    optimizer='adam',  
    metrics=['accuracy']  
)
```


Training the Model on the Dataset for 25 Epochs with 20% Validation Split


```
In [16]: history = model.fit(X_train,y_train,epochs=25,validation_split=0.2)
```


Epoch 1/25
1500/1500  3s 1ms/step - accuracy: 0.9185 - loss: 0.2791 - val_accuracy: 0.9613 - val_loss: 0.1383


Epoch 2/25
1500/1500  2s 1ms/step - accuracy: 0.9653 - loss: 0.1156 - val_accuracy: 0.9703 - val_loss: 0.0998


Epoch 3/25
1500/1500  2s 1ms/step - accuracy: 0.9760 - loss: 0.0806 - val_accuracy: 0.9741 - val_loss: 0.0894


Epoch 4/25
1500/1500  2s 1ms/step - accuracy: 0.9813 - loss: 0.0588 - val_accuracy: 0.9707 - val_loss: 0.1016


Epoch 5/25
1500/1500  2s 1ms/step - accuracy: 0.9857 - loss: 0.0452 - val_accuracy: 0.9692 - val_loss: 0.1106


Epoch 6/25
1500/1500  2s 1ms/step - accuracy: 0.9886 - loss: 0.0361 - val_accuracy: 0.9743 - val_loss: 0.0948


Epoch 7/25
1500/1500  2s 1ms/step - accuracy: 0.9906 - loss: 0.0295 - val_accuracy: 0.9722 - val_loss: 0.1120


Epoch 8/25
1500/1500  2s 1ms/step - accuracy: 0.9918 - loss: 0.0249 - val_accuracy: 0.9728 - val_loss: 0.1061


Epoch 9/25
1500/1500  2s 1ms/step - accuracy: 0.9927 - loss: 0.0219 - val_accuracy: 0.9751 - val_loss: 0.1038


Epoch 10/25
1500/1500  2s 1ms/step - accuracy: 0.9943 - loss: 0.0173 - val_accuracy: 0.9743 - val_loss: 0.1153


Epoch 11/25
1500/1500  2s 1ms/step - accuracy: 0.9935 - loss: 0.0192 - val_accuracy: 0.9732 - val_loss: 0.1217


Epoch 12/25
1500/1500  2s 1ms/step - accuracy: 0.9951 - loss: 0.0143 - val_accuracy: 0.9732 - val_loss: 0.1252


Epoch 13/25
1500/1500  2s 1ms/step - accuracy: 0.9954 - loss: 0.0135 - val_accuracy: 0.9772 - val_loss: 0.1151


Epoch 14/25
1500/1500  2s 1ms/step - accuracy: 0.9950 - loss: 0.0149 - val_accuracy: 0.9754 - val_loss: 0.1264

Epoch 15/25
1500/1500  2s 1ms/step - accuracy: 0.9971 - loss: 0.0094 - val_accuracy: 0.9714 - val_loss: 0.1394

Epoch 16/25
1500/1500  2s 1ms/step - accuracy: 0.9964 - loss: 0.0106 - val_accuracy: 0.9746 - val_loss: 0.1360

Epoch 17/25
1500/1500  2s 1ms/step - accuracy: 0.9966 - loss: 0.0111 - val_accuracy: 0.9781 - val_loss: 0.1288

Epoch 18/25
1500/1500  2s 1ms/step - accuracy: 0.9969 - loss: 0.0099 - val_accuracy: 0.9744 - val_loss: 0.1556

Epoch 19/25
1500/1500  2s 1ms/step - accuracy: 0.9965 - loss: 0.0104 - val_accuracy: 0.9744 - val_loss: 0.1556

```

ccuracy: 0.9767 - val_loss: 0.1335
Epoch 20/25
1500/1500 ————— 2s 1ms/step - accuracy: 0.9973 - loss: 0.0089 - val_a
ccuracy: 0.9740 - val_loss: 0.1688
Epoch 21/25
1500/1500 ————— 2s 1ms/step - accuracy: 0.9974 - loss: 0.0077 - val_a
ccuracy: 0.9753 - val_loss: 0.1582
Epoch 22/25
1500/1500 ————— 2s 1ms/step - accuracy: 0.9971 - loss: 0.0092 - val_a
ccuracy: 0.9751 - val_loss: 0.1581
Epoch 23/25
1500/1500 ————— 2s 1ms/step - accuracy: 0.9978 - loss: 0.0074 - val_a
ccuracy: 0.9740 - val_loss: 0.1759
Epoch 24/25
1500/1500 ————— 2s 1ms/step - accuracy: 0.9969 - loss: 0.0092 - val_a
ccuracy: 0.9743 - val_loss: 0.1807
Epoch 25/25
1500/1500 ————— 2s 1ms/step - accuracy: 0.9977 - loss: 0.0064 - val_a
ccuracy: 0.9769 - val_loss: 0.1610

```

```
In [27]: y_prob = model.predict(X_test)
```

```
313/313 ————— 0s 706us/step
```

```
In [18]: y_pred = y_prob.argmax(axis=1)
```

Model Accuracy on the Test Set

```
In [19]: from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
Out[19]: 0.9771
```

Confusion Matrix and Classification Report for Model Evaluation

```
In [20]: from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

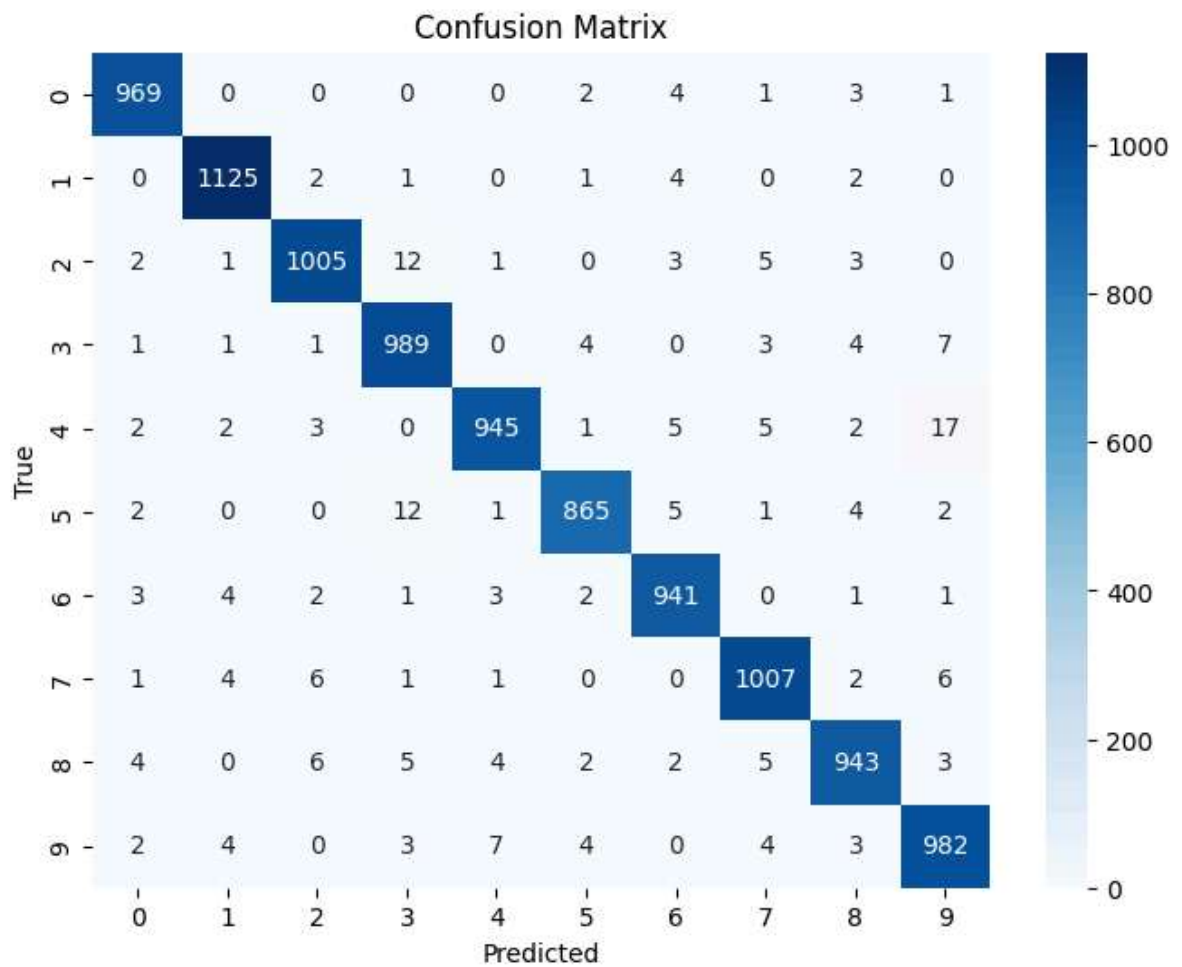
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

cm = confusion_matrix(y_test, y_pred_classes)

plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred_classes))
```

313/313 ————— 0s 596us/step



Classification Report:

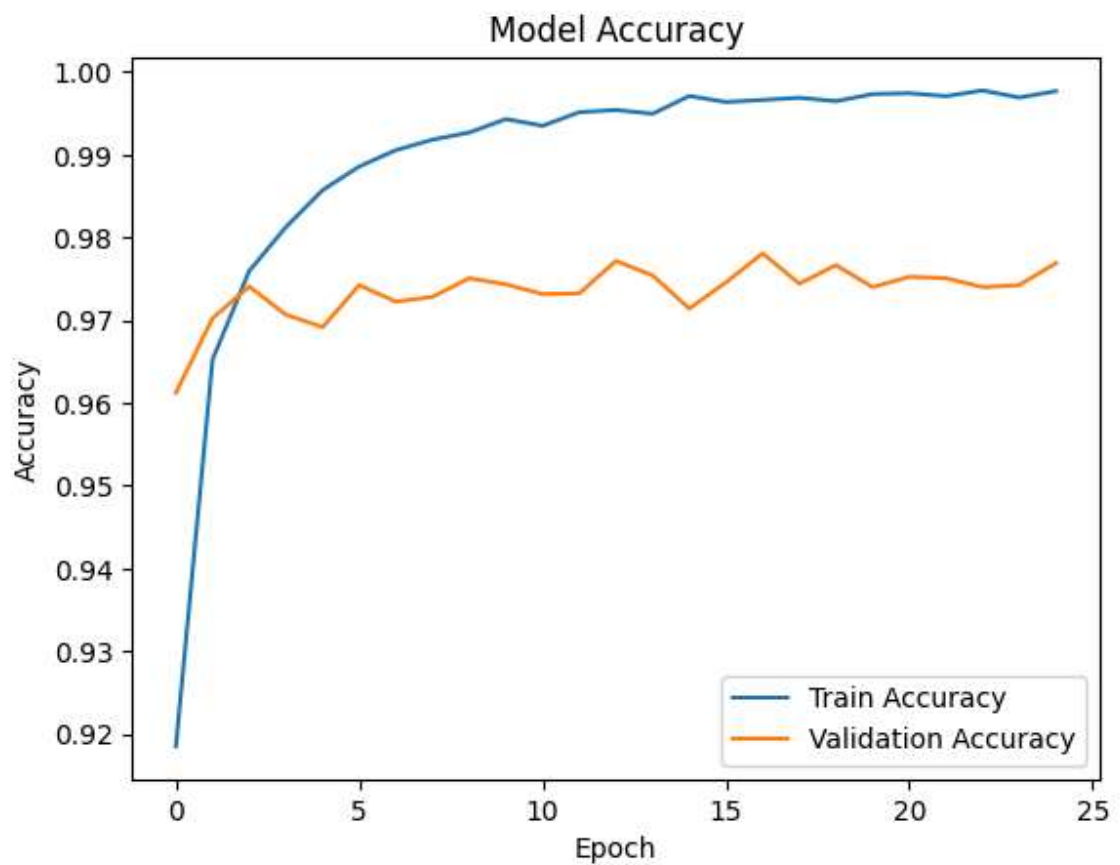
| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.98 | 0.99 | 0.99 | 980 |
| 1 | 0.99 | 0.99 | 0.99 | 1135 |
| 2 | 0.98 | 0.97 | 0.98 | 1032 |
| 3 | 0.97 | 0.98 | 0.97 | 1010 |
| 4 | 0.98 | 0.96 | 0.97 | 982 |
| 5 | 0.98 | 0.97 | 0.98 | 892 |
| 6 | 0.98 | 0.98 | 0.98 | 958 |
| 7 | 0.98 | 0.98 | 0.98 | 1028 |
| 8 | 0.98 | 0.97 | 0.97 | 974 |
| 9 | 0.96 | 0.97 | 0.97 | 1009 |
| accuracy | | | 0.98 | 10000 |
| macro avg | 0.98 | 0.98 | 0.98 | 10000 |
| weighted avg | 0.98 | 0.98 | 0.98 | 10000 |

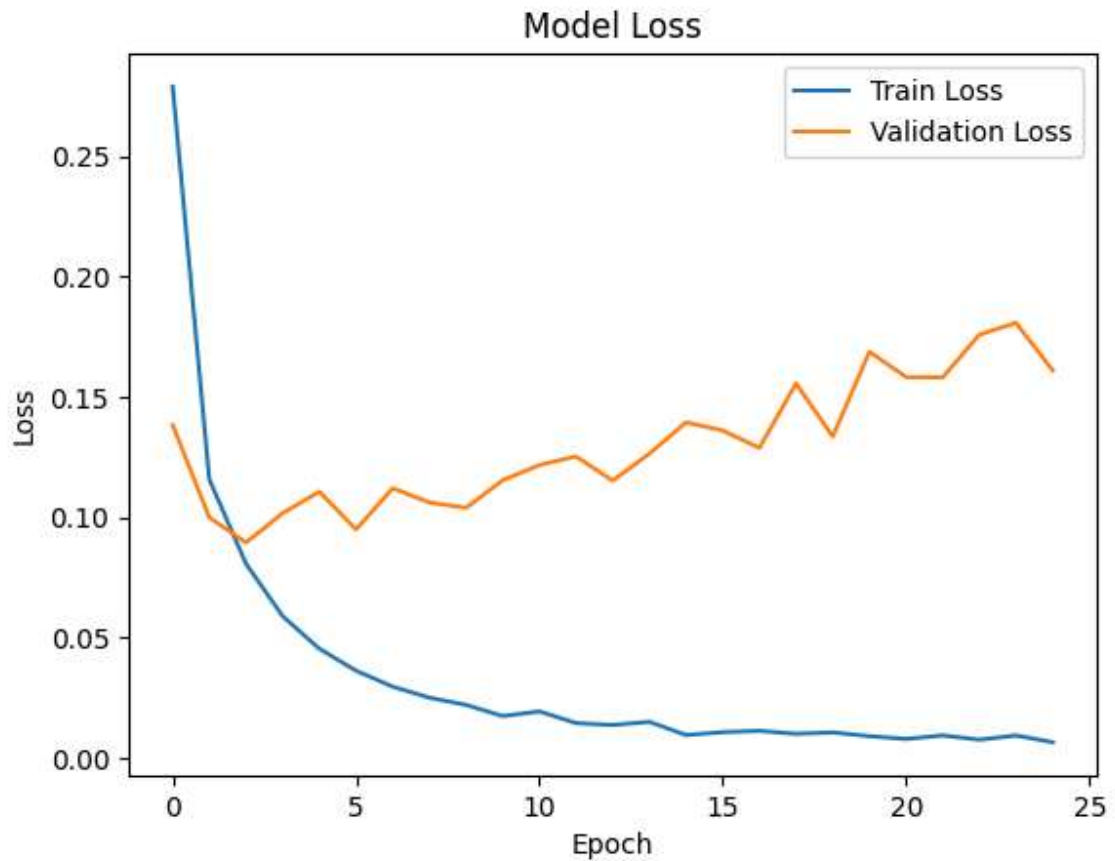
Training and Validation Accuracy & Loss Over Epochs

```
In [22]: import matplotlib.pyplot as plt
```

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





Saving and Loading the Trained Model

```
In [23]: model.save("my_model.keras")

from keras.models import load_model
model_s = load_model("my_model.keras")
```

```
In [24]:
```

Interactive Handwritten Digit Recognition with Pygame

```
In [ ]: import pygame, sys
from pygame.locals import *

import numpy as np
from keras.models import load_model
import cv2

WINDOWSIZE_X = 640
WINDOWSIZE_Y = 640

BOUNDARYINC = 5
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
```

```

IMAGESAVE = False

MODEL = load_model('my_model.keras')
LABELS = {0: 'Zero', 1: 'One',
           2: 'Two', 3: 'Three',
           4: 'Four', 5: 'Five',
           6: 'Six', 7: 'Seven',
           8: 'Eight', 9: 'Nine'}

pygame.init()

FONT = pygame.font.Font('freesansbold.ttf', 18)
DISPLAYSURF = pygame.display.set_mode((WINDOWSIZE_X, WINDOWSIZE_Y))

pygame.display.set_caption('Digit Board')

iswriting = False

number_xcord = []
number_ycord = []

image_cnt = 1
PREDICT = True

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

        if event.type == MOUSEMOTION and iswriting:
            x, y = event.pos
            pygame.draw.circle(DISPLAYSURF, WHITE, (x, y), 4, 0)

            number_xcord.append(x)
            number_ycord.append(y)

        if event.type == MOUSEBUTTONDOWN:
            iswriting = True

        if event.type == MOUSEBUTTONUP:
            iswriting = False
            number_xcord = sorted(number_xcord)
            number_ycord = sorted(number_ycord)

            rect_min_x, rect_max_x = max(number_xcord[0] - BOUNDARYINC, 0), min(WIND
            rect_min_y, rect_max_y = max(number_ycord[0] - BOUNDARYINC, 0), min(WIND

            number_xcord = []
            number_ycord = []

            img_array = np.array(pygame.PixelArray(DISPLAYSURF))[rect_min_x:rect_ma

```

```

if IMAGESAVE:
    cv2.imwrite(f'image_{image_cnt}.png', img_array)
    image_cnt += 1
if PREDICT:
    image = cv2.resize(img_array, (28, 28))
    image = np.pad(image, (10, 10), 'constant', constant_values=0)
    image = cv2.resize(image, (28, 28))/255

    label = str(LABELS[np.argmax(MODEL.predict(image.reshape(1, 28, 28, 3)))]

    textSurface = FONT.render(label, True, RED, WHITE)
    textRecObj = textSurface.get_rect()
    textRecObj.left, textRecObj.bottom = rect_min_x, rect_max_y

    DISPLAYSURF.blit(textSurface, textRecObj)

if event.type == KEYDOWN:
    if event.unicode == 'n':
        DISPLAYSURF.fill(BLACK)

pygame.display.update()

```

In []:

3. Performance Evaluation

3.1 Output

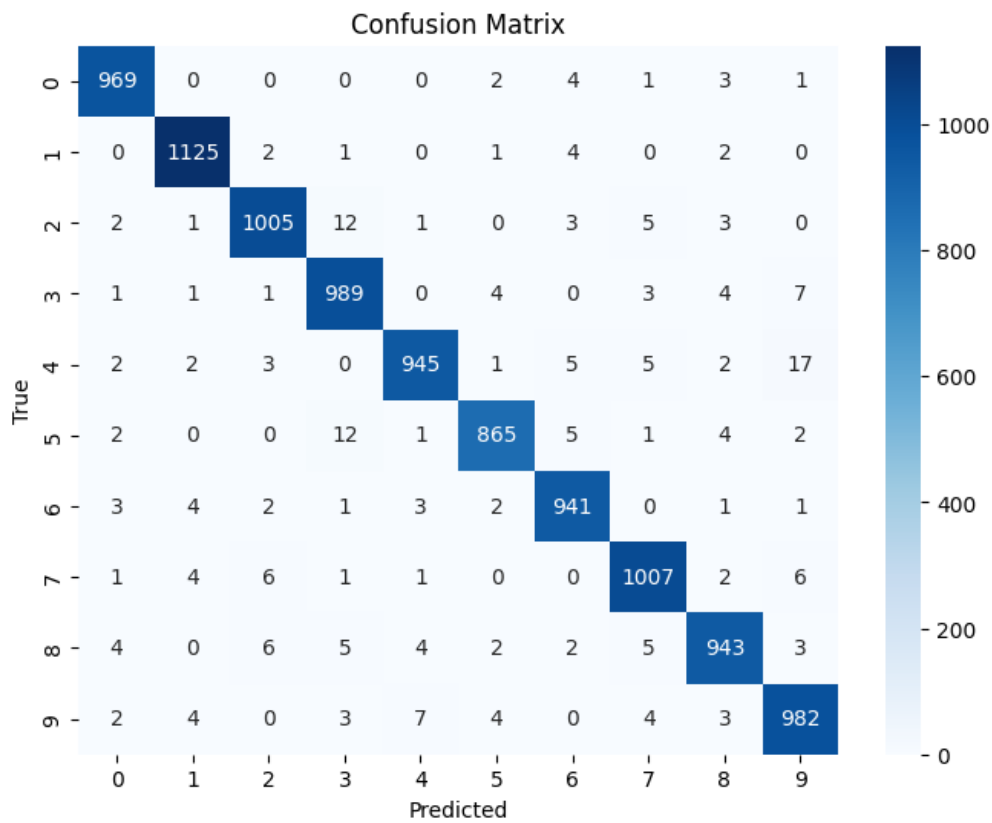


Figure 3.1: Confusion Matrix

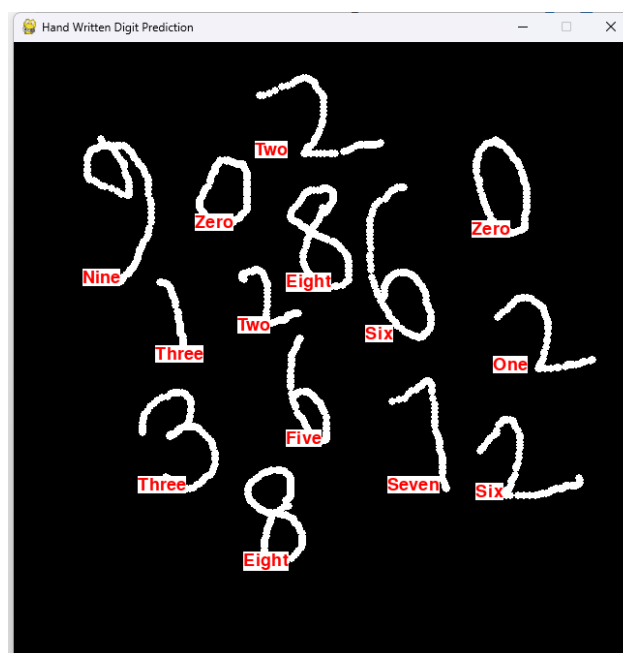


Figure 3.2: Hand Written Digit Prediction

4. Conclusion

4.1 Discussion

This project demonstrates a complete system for recognizing handwritten digits using an artificial neural network (ANN). A small ANN model trained on the MNIST dataset can achieve high accuracy while remaining lightweight. Preprocessing the input images—such as cropping, centering, and scaling—is very important. The system’s user interface (UI) must apply the same preprocessing steps used during training to work correctly. We implemented an interactive interface using Pygame, which allows users to draw digits and see the model’s predictions. This makes the system practical for applications like forms, educational tools, or demonstrations.

4.2 Limitations and Future Work:

The model may not perform as well on real-world digits that differ from MNIST, such as digits written with different pens or on varied backgrounds. To improve performance, we can apply data augmentations like rotations, shifts, or elastic distortions, and train with extended datasets like EMNIST, which includes letters as well as digits. For mobile or embedded use, the ANN can be converted to TensorFlow Lite and optimized using quantization to reduce model size and latency. Future work can also explore recognizing multiple characters in a sequence, segmentation of digits from images, and sequence models such as CTC decoders to read numbers like amounts or ZIP codes directly.

4.3 Resources

Project Resources

GitHub Repository:

Implementation and source code can be accessed here:

[GitHub — Handwritten Digit Recognition Using Artificial Neural Network](#)

References

- [1] GeeksforGeeks. (2025). *K means Clustering – Introduction*. Retrieved from <https://www.geeksforgeeks.org/k-means-clustering-introduction/>
- [2] BytePlus. (2023). *How K-means Clustering is Revolutionizing Educational Analytics*. Retrieved from <https://www.byteplus.com/en/topic/479185>
- [3] ResearchGate. (2023). *Clustering Student Performance Data Using k-Means Algorithms*. Retrieved from https://www.researchgate.net/publication/367543523_CLUSTERING_STUDENT_PERFORMANCE_DATA_USING_k-MEANS_ALGORITHMS
- [4] Wikipedia. (2025). *Educational data mining*. Retrieved from https://en.wikipedia.org/wiki/Educational_data_mining