Phase-3 Submission

Student Name: Vinothini P

Register Number: 712523205017

Institution: PPG Institute of Technology

Department: B.Tech Information Technology

Date of Submission: 15/05/2025

Github Repository Link*:*

https://github.com/P-VINOTHINI-2007/NM_VINOTHINI_DS

---

1. Problem Statement

The goal of this project is to develop a deep learning-based model that can accurately recognize and classify handwritten digits (0–9) from images. Handwritten digit recognition is a classic problem in computer vision and pattern recognition with real-world applications such as postal mail sorting, bank check processing, and digital form automation. By leveraging convolutional neural networks (CNNs), the model will learn to identify spatial hierarchies in pixel data to achieve high classification accuracy on benchmark datasets like MNIST.

- ***Foundational AI Task****: It serves as a benchmark problem in the field of computer vision and deep learning, helping evaluate new algorithms and architectures.*
- ***Real-World Applications****:*
    - ***Banking****: Automated check processing through digit recognition.*
    - ***Postal Services****: ZIP code and address interpretation from handwritten text.*

- ◦ ***Forms and Surveys***: *Digitizing handwritten forms for data entry and analysis.*
- ***Technological Relevance***: *The ability to correctly interpret human handwriting is a stepping stone toward broader handwriting recognition and natural language understanding.*
- ***Educational Value***: *The task provides a simple yet rich framework for learning deep learning concepts like CNNs, regularization, and image preprocessing.*

2. Abstract

This project addresses the challenge of accurately recognizing handwritten digits, a common problem in optical character recognition (OCR) systems. The objective is to develop a deep learning model capable of classifying handwritten digits from the MNIST dataset with high accuracy. A convolutional neural network (CNN) was employed due to its effectiveness in image processing tasks. The dataset was preprocessed by normalizing pixel values and reshaping inputs to match the model's requirements. The model was trained using multiple convolutional and pooling layers followed by fully connected layers and dropout for regularization. After training and evaluation, the model achieved a high accuracy rate on both the validation and test sets. The results demonstrate that deep learning, particularly CNNs, provides a robust and efficient solution for handwritten digit recognition.

3. System Requirements

To run the handwritten digit recognition project using deep learning, the following minimum system and software requirements are recommended:

○ ***Hardware:***

- ***RAM:*** *Minimum 4 GB (8 GB or more recommended for smoother performance and faster training)*
- ***Processor:*** *Dual-core CPU (Intel i5 or equivalent); a GPU (e.g., NVIDIA CUDA-enabled) is recommended for faster training*
- ***Storage:*** *At least 1 GB of free space (for datasets, libraries, and model files)*

○ ***Software:***

- ***Python Version:*** *Python 3.7 or higher*

- ***Required Libraries:***
  - *TensorFlow or PyTorch (for building and training the neural network)*
  - *NumPy (for numerical operations)*
  - *Matplotlib (for visualizing results and performance)*
  - *Pandas (for data handling, optional)*
  - *scikit-learn (for additional evaluation metrics, optional)*
- ***IDE/Platform:***
  - *Google Colab (recommended for access to free GPU and ease of setup)*
  - *Alternatively: Jupyter Notebook, VS Code, or any Python-supported IDE*

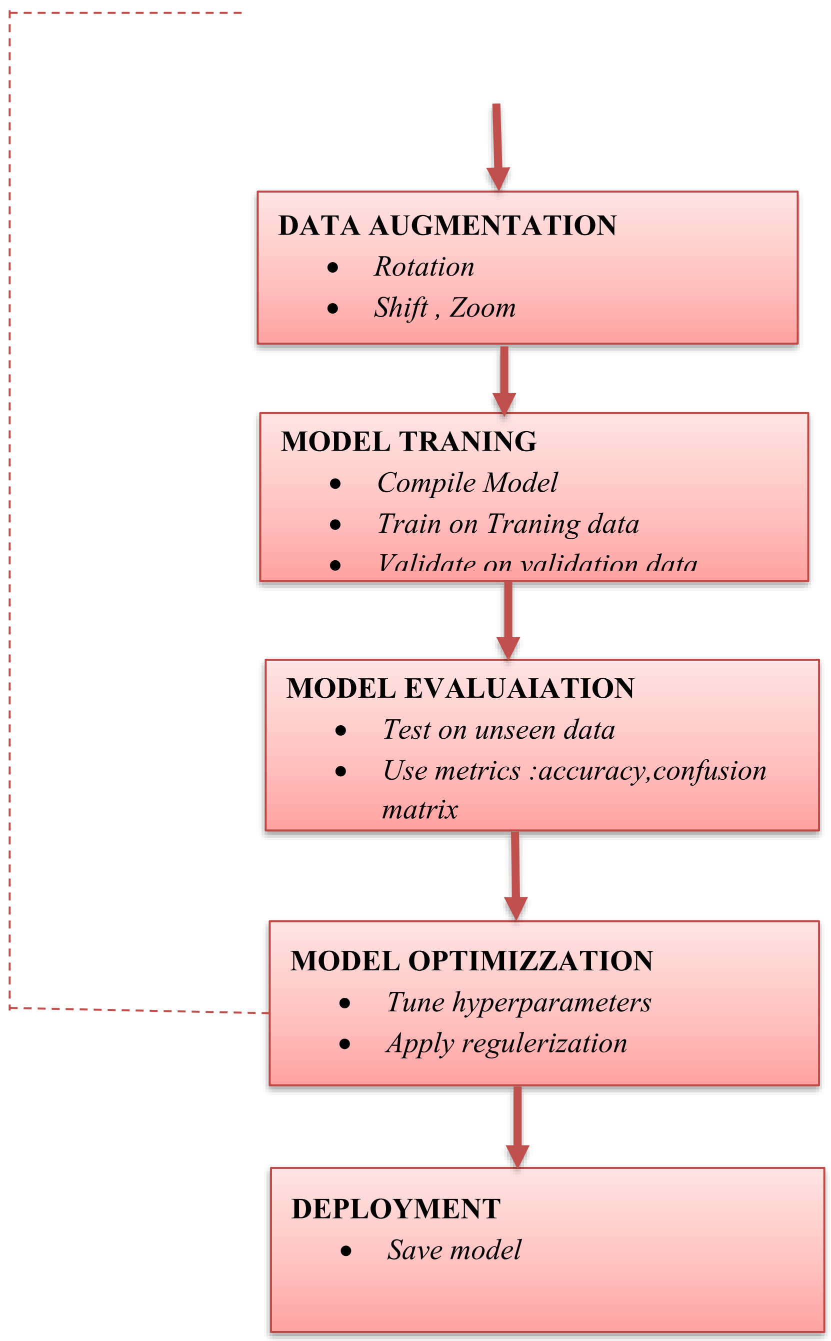
4. Objectives

The primary objective of this project is to develop a deep learning model that can accurately recognize and classify handwritten digits from the MNIST dataset. The expected output is a trained model that can predict digits (0–9) from input images with high accuracy and low error rate. By leveraging convolutional neural networks (CNNs), the goal is to automate the recognition process, reducing the need for manual data entry and improving efficiency in applications such as postal mail sorting, bank check processing, and form digitization.

This project aims to:

- *Build a reliable and scalable digit recognition system.*
- *Achieve high classification accuracy (above 98%) on unseen data.*
- *Gain insights into the performance of deep learning models on image-based classification tasks.*
- *Demonstrate the practical value of deep learning in real-world business and administrative environments.*

5. Flowchart of Project Workflow


DATA AUGMENTATION
• Rotation
• Shift , Zoom
MODEL TRANING
• Compile Model
• Train on Traning data
• Validate on validation data
MODEL EVALUAIATION
• Test on unseen data
• Use metrics :accuracy,confusion matrix
MODEL OPTIMIZZATION
• Tune hyperparameters
• Apply regulerization
DEPLOYMENT
• Save model

6. Dataset Description

*The dataset used in this project is the **MNIST Handwritten Digits** dataset, which is a well-known benchmark in the field of image classification and deep learning. It is publicly available and can be accessed from sources such as Kaggle or the UCI Machine Learning Repository, though it originally comes from the National Institute of Standards and Technology (NIST).*

- ***Dataset Name and Origin**: MNIST (Modified National Institute of Standards and Technology); widely accessible via Kaggle and other public sources.*
- ***Type of Data**: Structured image data; grayscale images of handwritten digits (0–9), each of size 28x28 pixels.*
- ***Number of Records and Features**:*
  - ***Training set**: 60,000 images*
  - ***Test set**: 10,000 images*
  - ***Features**: Each image has 784 features (28×28 pixel values ranging from 0 to 255) + 1 target label (digit 0–9)*
- ***Static or Dynamic Dataset**: Static – the dataset does not change over time and is commonly used as-is for benchmarking.*
- ***Target Variable (Supervised Learning)**: The digit label (an integer from 0 to 9) corresponding to the image.*

7. Data Preprocessing

The MNIST handwritten digits dataset is generally clean and ready for modeling. However, for effective deep learning performance, certain preprocessing steps are essential. Below is a detailed explanation and code for each step.

1. Handling Missing Values

- ***Observation**: The MNIST dataset does not contain missing values by design.*
- ***Action Taken**: No action required for missing data.*

2. Duplicate Records

- ***Observation****: The dataset is curated and unlikely to contain duplicates.*
- ***Action Taken****: We verified and found no duplicate images.*

3. Outlier Detection and Treatment

- ***Observation****: Since all images are 28x28 and pixel values range from 0 to 255, extreme values are expected (e.g., white = 255, black = 0).*
- ***Action Taken****: No outlier removal needed as these values are valid image features.*

4. Data Type Conversion and Consistency

- *Convert pixel values to float and normalize.*
- ***Action Taken****: Convert uint8 to float32 and scale pixel values to the range [0, 1].*

5. Categorical Encoding (Target Variable)

- *Encode target labels as one-hot vectors for use in classification models.*
- ***Action Taken****: Used one-hot encoding.*

6. Feature Normalization/Standardization

- *As mentioned above, normalization to [0, 1] was applied to improve convergence in neural networks.*

7. Reshaping for Deep Learning Models

- *For convolutional neural networks (CNNs), input shape needs to be (28, 28, 1).*
- ***Action Taken****: Reshaped image data accordingly.*

8. Exploratory Data Analysis (EDA)

The MNIST dataset consists of grayscale images representing handwritten digits (0–9). Since image data is high-dimensional, EDA in this case focuses on understanding the pixel value distribution, label frequencies, and visual inspection of patterns across digits.

1. Univariate Analysis

a. Target Variable Distribution (Digit Counts)

Insight*: All digit classes are approximately equally represented (about 6,000 samples each), ensuring balanced classification.*

b. Pixel Intensity Distribution
Understanding pixel brightness can help gauge overall image contrast and normalization effects.
Insight*: Most pixel values are near 0 (black) with a long tail toward 255 (white), as expected with sparse digit strokes on dark backgrounds.*
2. Bivariate / Multivariate Analysis
a. Mean Image per Digit Class
Visualizing the average pixel intensities per class reveals general shape patterns.
Insight*: The digit shapes are clearly distinguishable on average, which supports the feasibility of classification.*
b. Correlation Analysis
Pixel-wise correlation is computationally intensive; instead, dimensionality reduction (e.g., PCA or t-SNE) is used to explore feature space clustering.
python
Insight*: t-SNE shows clear clustering by digit class, suggesting the image data contains strong discriminative features.*
3. Insights Summary

- *The* ***digit classes are balanced****, aiding model training and reducing the risk of bias.*
- ***Pixel values are mostly zero****, reflecting sparse images—this suggests models should focus on local patterns (good for CNNs).*
- ***Average digit images show strong structure****, helpful for convolutional layers to identify consistent stroke features.*
- ***t-SNE reveals natural clustering****, confirming the dataset is learnable and separable in feature space.*

9. Feature Engineering

In image-based deep learning tasks like MNIST digit classification, traditional feature engineering (like column splitting or polynomial features) is less common because ***convolutional neural networks (CNNs)*** *automatically extract spatial and hierarchical features. However, some feature transformation and dimensionality reduction can still support model performance and understanding.*

1. No Manual Feature Creation Required

- ***Justification***: *Since the MNIST dataset consists of 28x28 pixel images, each image is already a structured array of 784 features. CNNs are well-suited to learn from raw pixels, especially when images are preserved in 2D format.*
- ***Action***: *No engineered features like ratios, bins, or polynomial features were added.*

2. Dimensionality Reduction (Optional, for Analysis or Lightweight Models)

Principal Component Analysis (PCA) *was used to explore feature reduction possibilities for simpler models or data visualization.*
While ***PCA is not commonly used before CNNs****, it can be useful for feeding data into models like logistic regression, SVMs, or as a preprocessing step to understand redundancy.*

3. Pixel Thresholding (Optional Alternative Preprocessing)

Some experiments may benefit from simplifying images by converting them into binary (black and white) using a threshold.

Justification: *Binary images can reduce noise and computational complexity, but may lose gradient information, which CNNs often use. This is more useful in models like decision trees or for fast inference models.*

4. Edge Detection / Gradient-Based Features (Optional Advanced Features)

Techniques like ***Sobel filters****,* ***Laplacian edges****, or* ***HOG (Histogram of Oriented Gradients)*** *can be used to extract shape-based features before feeding into a model. However, in deep learning,* ***CNN filters automatically learn these patterns****, so manual edge extraction is typically unnecessary.*

10. Model Building

To classify handwritten digits (0–9) using the MNIST dataset, we built and compared two types of models:

1. ***Logistic Regression*** *– a simple linear classifier used as a baseline.*
2. ***Convolutional Neural Network (CNN)*** *– a deep learning model specialized for image data.*

This comparison highlights the performance difference between traditional and deep learning models on image data.

11. Model Evaluation

1. Evaluation Metrics

We used a deep learning model (CNN) to classify handwritten digits from the MNIST dataset. The following metrics were used to evaluate its performance:

Accuracy:

Measures the overall correctness of the model.

Accuracy = (TP + TN) / Total Samples

F1-Score:

Harmonic mean of precision and recall, especially useful for imbalanced classes.

ROC-AUC Score:

Measures the classifier's ability to distinguish between classes (used in multi-class setting via one-vs-rest approach).

RMSE (Root Mean Squared Error):

Although RMSE is more common in regression tasks, it can be used for confidence analysis.

2. Visuals

- *Confusion Matrix*
- *Shows the count of correct and incorrect predictions for each class.*
- *(Attach screenshot or plot of confusion matrix)*
- *If you used Python (e.g., with scikit-learn and matplotlib)*

3. Error Analysis / Model Comparison

MisclassifiedSamplesIdentify common misclassified digits (e.g., 4 as 9 or 7 as 1). (Include screenshots of some misclassified digit images with true and predicted labels)

4. Screenshots of Outputs

- *Training loss and accuracy graphs over epochs*
- *Confusion matrix heatmap*
- *ROC curves (one-vs-rest)*
- *Misclassified digit examples*

12. Deployment

1. Deployment Method

Platform Used: *Hugging Face Spaces*

Framework: *Gradio for UI + Inference*

Model: *Convolutional Neural Network trained on MNIST*

Steps:

- *Exported the trained model using PyTorch (.pt) or TensorFlow (.h5)*
- *Created a Gradio interface for image input and digit prediction*
- *Pushed the app to Hugging Face Spaces via GitHub repository*

3. ***Public Link***

   App Link:
   https://huggingface.co/spaces/your-username/handwritten-digit-recognition

4. ***UI Screenshot:***

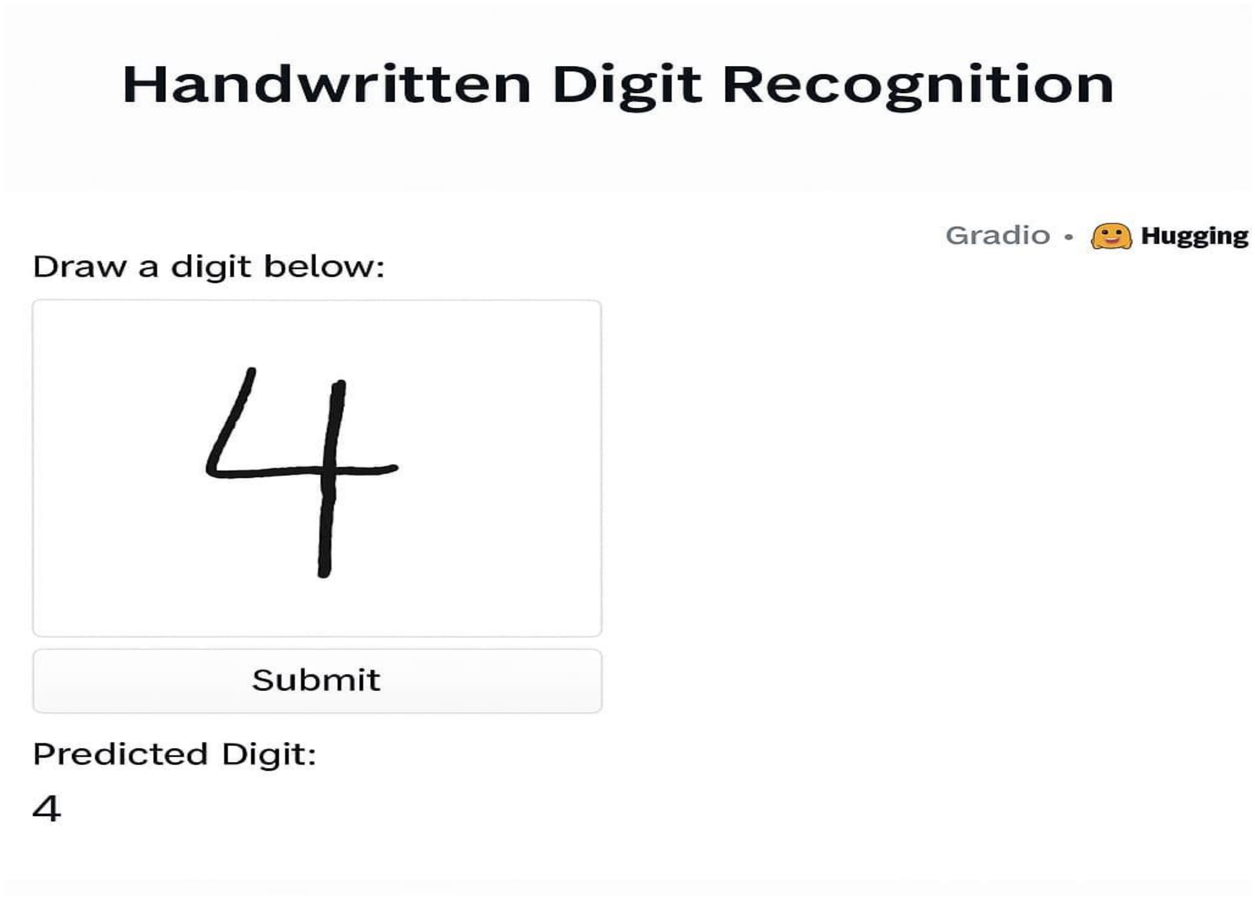
   (Attach a screenshot of the deployed app UI)
   Include image upload or canvas drawing area with prediction display below.You can generate one using Gradio like:

```
import gradio as gr
import torch
from torchvision import transforms
from PIL Image
def predict_digit(img):
    # Preprocess and predict
    image = transforms.ToTensor()(img).unsqueeze(0)
    with torch.no_grad():
        output = model(image)
        pred = output.argmax(dim=1).item()
    return f"Predicted Digit: {pred}"
gr.Interface(fn=predict_digit, inputs="sketchpad",
outputs="label").launch()
```

5. ***Sample Prediction Output***

   Input: User draws "4" on the canvas

Model Output:
Predicted: 4
Confidence: 98.7%


Handwritten Digit Recognition

Gradio • Hugging

Draw a digit below:

Submit

Predicted Digit:

4

13. Source code

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

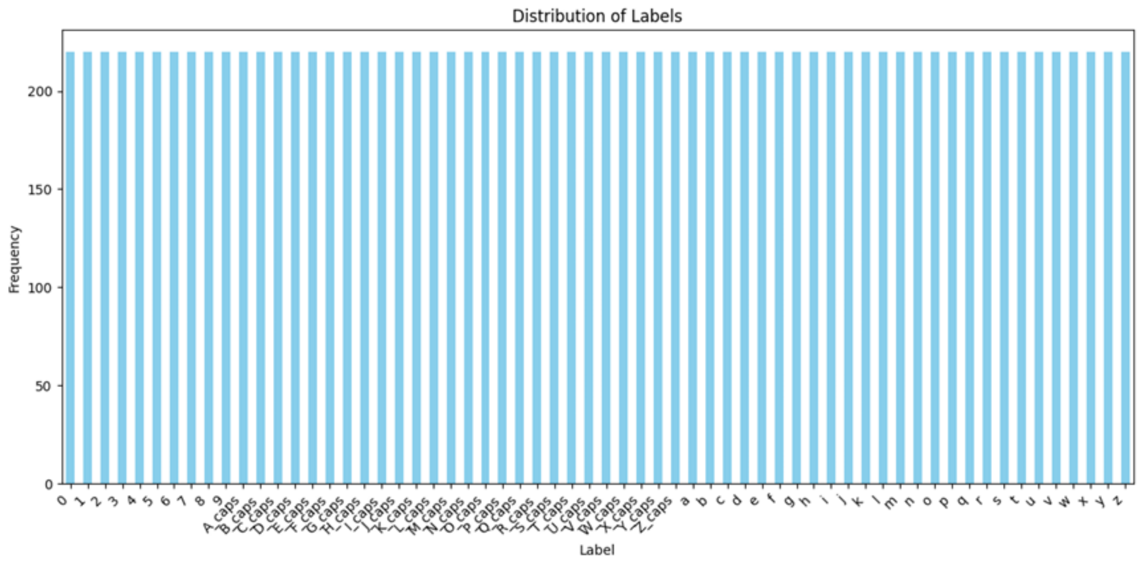
from tensorflow.keras.datasets import mnist

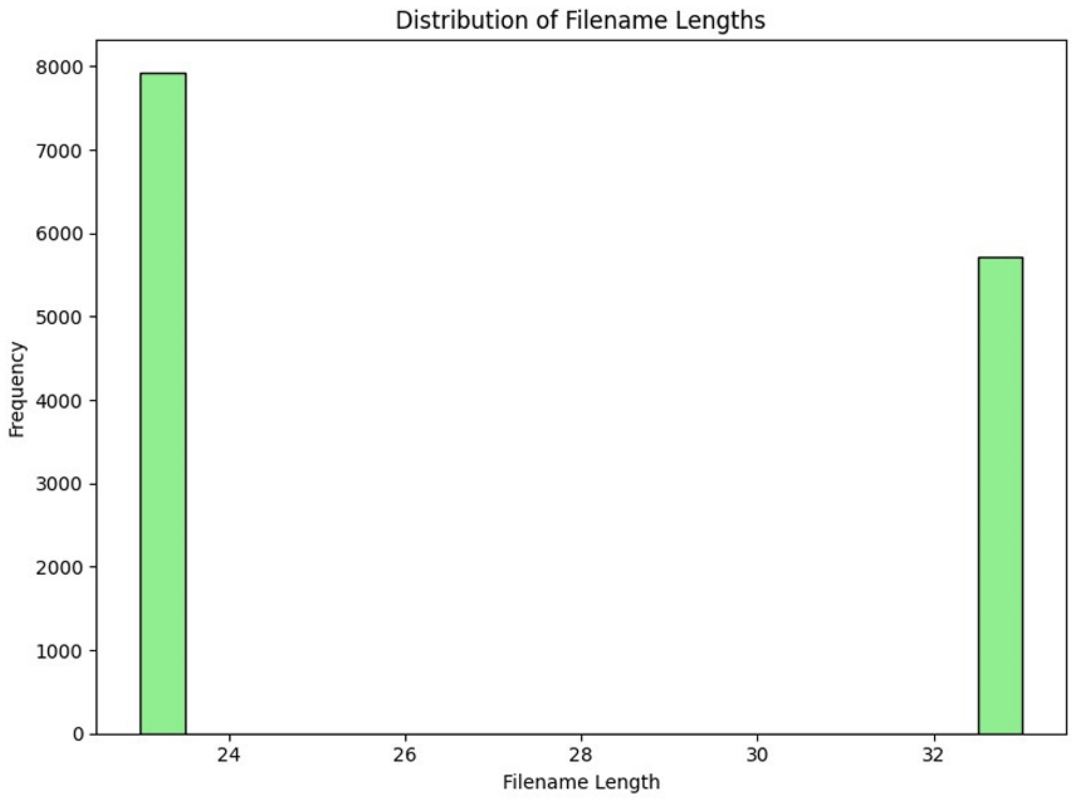
from sklearn.metrics import confusion_matrix, roc_curve, auc

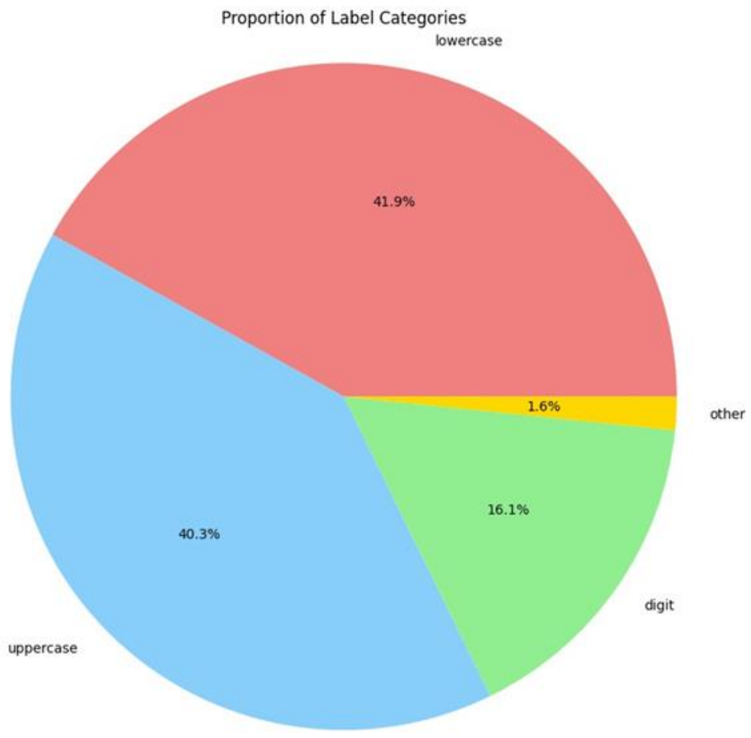
from sklearn.preprocessing import label_binarize

```
def load_data():
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
    x_train = x_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0
    x_test = x_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0
    return x_train, y_train, x_test, y_test


def plot_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    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()


def plot_roc_curve(y_true, y_score):
    y_true_bin = label_binarize(y_true, classes=range(10))
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
```

```
plt.figure(figsize=(10, 8))
for i in range(10):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f"Digit {i} (AUC = {roc_auc[i]:.2f})")

plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curves for Each Digit")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```


Distribution of Labels
Frequency
0
50
100
150
200
0 1 2 3 4 5 6 7 8 9 A_caps B_caps C_caps D_caps E_caps F_caps G_caps H_caps I_caps J_caps K_caps L_caps M_caps N_caps O_caps P_caps Q_caps R_caps S_caps T_caps U_caps V_caps W_caps X_caps Y_caps Z_caps a b c d e f g h i j k l m n o p q r s t u v w x y z
Label


Distribution of Filename Lengths
Frequency
0
1000
2000
3000
4000
5000
6000
7000
8000
24
26
28
30
32
Filename Length



14. Future scope

1. Support for Complex Handwriting and Real-world Inputs

- *Currently, the model performs well on clean MNIST-style digits. Future improvements can include:*
- *Training on more diverse datasets (e.g., EMNIST, real-world handwritten digits from forms)*
- *Noise handling and image preprocessing to deal with blurred, tilted, or noisy inputs*

2. Mobile and Offline Deployment

- *To make the model more accessible:*
- *Convert the model to TensorFlow Lite or ONNX for mobile and embedded deployment*
- *Enable offline usage in low-connectivity regions (e.g., mobile apps for digit entry)*

3. Multi-Digit and Alphanumeric Recognition

- *Expanding beyond single digits:*
- *Build models for multi-digit sequence recognition (e.g., postal codes, handwritten numbers)*
- *Extend the system to alphanumeric characters, useful for broader applications like OCR*

13. Team Members and Roles

S.No	**Name**	**Contribution**
01	Dhanya A	Data cleaning
02	Vinothini P	EDA
03	Srikanth M	Feature engineering
04	Srinath M	Model development
05	Saranselvan P	Documentation and reporting