Phase-2 Submission Template

Student Name: Priyadharshini.S

Register Number: 622423121033

Institution: Salem College Of Engineering And Technology

Department: Biomedical Engineering

Date of Submission: 08.05.2025

Github Repository Link:https://github.com/Ramachandran0706/Nm-Project.git

---

Title: Recognizing handwritten digits with deep learning for smarter AI applications

1.Problem Statement

The aim of this project is to develop an intelligent system capable of accurately recognizing handwritten digits using deep learning techniques. By leveraging Convolutional Neural Networks (CNNs), the system will be trained on digit image datasets to classify and interpret handwritten numbers. This recognition system can be integrated into various AI-powered applications such as smart form processing, automated banking systems, educational tools, and data digitization platforms, thereby enhancing efficiency and reducing human error in digit recognition task.

2. Project Objectives

.* *To design and develop a deep learning model capable of accurately recognizing handwritten digits using Convolutional Neural Networks (CNNs).*

* *To train the model using a standardized dataset such as MNIST, ensuring high accuracy and generalization on unseen data.*

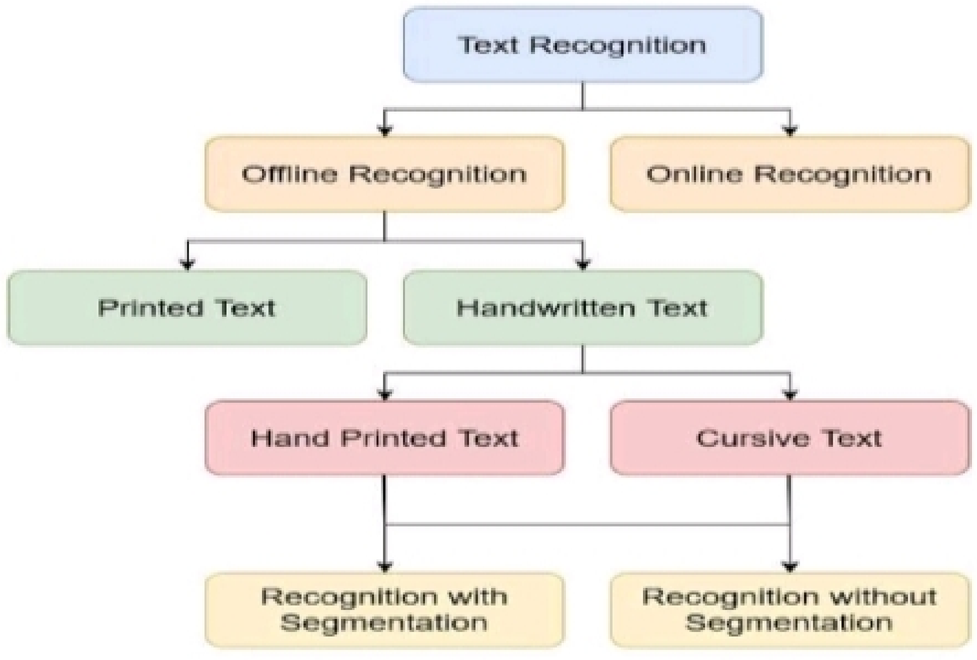
***** To evaluate the performance of the model using appropriate metrics such as accuracy, precision, recall, and confusion matrix.**

***** To integrate the digit recognition system into practical AI applications, demonstrating real-world use cases such as automated form filling and smart data entry.**

***** To explore model optimization techniques for deployment on resource-constrained devices (e.g., mobile phones or edge devices).**

****To enhance understanding of AI in pattern recognition, particularly in the field of optical character recognition (OCR).***

3.FLOW CHART



4. Data Description

For this project, the MNIST (Modified National Institute of Standards and Technology) dataset is used. It is a benchmark dataset widely used for training and testing machine learning models on handwritten digit recognition tasks.

Dataset Name: MNIST Handwritten Digit Dataset

Total Samples: 70,000 images

Training Set: 60,000 images

Test Set: 10,000 images

Image Format: Grayscale

Image Size: 28 x 28 pixels

Label Range: 0 to 9 (each representing a digit)

Data Type:

Input: Pixel values (0–255)

Output: Digit class labels (0–9)

5. Data Preprocessing

Before training the deep learning model, the MNIST dataset undergoes several preprocessing steps to ensure the data is clean, normalized, and in a format suitable for training:

Reshaping:

Each image is originally 28x28 pixels. For CNNs, the images are reshaped to include the channel dimension
Shape: (28, 28, 1) for grayscale input.

Normalization:

Pixel values range from 0 to 255.Normalize pixel values to a range of 0 to 1 by dividing by 255:

image = image / 255.0

Label Encoding:

The digit labels (0 to 9) are converted to one-hot encoded vectors.
Example:

Digit 3 → [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

Train-Test Split: Dataset is typically already split:60,000 images for training 10,000 images for testing.

Shuffling:

Training data is shuffled to ensure randomness and prevent model bias.

Augmentation (Optional):

Data augmentation techniques like rotation, zoom, or shift can be applied to increase diversity and improve model robustness.

Exploratory Data Analysis (EDA)

Data augmentation techniques like rotation, zoom, or shift can be applied to increase diversity and improve model robustness.

7. Feature Engineering

Raw Pixel Values as Features:

relationships Each image (28x28 pixels) provides 784 features (flattened). In CNNs, features are extracted directly from image matrices (28x28x1), preserving spatial.

Automatic Feature Extraction via CNN Layers:

Convolutional Layers extract local features like edges, curves, and shapes. Pooling Layers reduce dimensionality and retain important spatial features. Activation Functions (e.g., ReLU) introduce non-linearity, allowing the network to learn complex patterns.

Hierarchical Feature Learning:

Early layers detect simple features (edges, corners).Deeper layers learn complex features (digit shapes and structures).

Flattening and Dense Layers:

After feature maps are learned, they are flattened and passed to fully connected layers to classify the digits.

Dimensionality Reduction (Optional):

Techniques like PCA (Principal Component Analysis) can be used before model training for analysis, but are usually not needed with CNNs.

8. Model Building

*Model Architecture**: A typical CNN architecture used for digit recognition may include the following layers***

Input Layer:Accepts 28x28 grayscale images (shape: 28x28x1)

Convolutional Layer 1:Applies multiple filters (e.g., 32 filters, 3x3 kernel)

Activation function: ReLU

Max Pooling Layer 1:

Reduces spatial dimensions (e.g., 2x2 pool size)

Convolutional Layer 2:

More filters (e.g., 64 filters, 3x3 kernel)

Activation function: ReLU

Max Pooling Layer 2:

Further downsampling

Flatten Layer:

Converts 2D feature maps into a 1D feature vector

Fully Connected (Dense) Layer:

Example: 128 neurons, ReLU activation

Compilation:

Loss Function: Categorical Cross-Entropy

Optimizer: Adam (adaptive learning rate optimizer)

Metrics: Accuracy

Output Layer:10 neurons (for digits 0–9), with Softmax activation for classification

Training the Model:

Epochs: Typically 10–20

Batch Size: Commonly 32 or 64

Validation Split: To monitor overfitting during training

Evaluation:

Use the test set to evaluate model performance.
Key metrics: Accuracy, Confusion Matrix, Precision, Recall

9. Visualization of Results & Model Insights

Accuracy and Loss Curves

Plot Training vs. Validation Accuracy and Loss: *Helps monitor overfitting and underfitting during training.*

Tools: matplotlib or seaborn

Confusion Matrix

Shows how well the model classifies each digit.Helps identify specific digits the model struggles with (e.g., 5 and 6).

Tool: sklearn.metrics.confusion_matrix

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_true, y_pred)

ConfusionMatrixDisplay(cm).plot()
```

Sample Predictions

Display test images with predicted and actual labels.

Useful for visual confirmation of correct and incorrect predictions.

```
plt.imshow(image, cmap='gray')

plt.title(f"Predicted: {pred}, Actual: {actual}")
```

Misclassified Images

Visualize examples where the model made wrong predictions.Helpful for improving model or preprocessing techniques.

Filter (Kernel) Visualization

Visualize filters from convolutional layers to understand what features the model is learning (e.g., edges, curves).

Feature Maps (Activation Maps)

Show outputs of intermediate layers to observe how features are progressively abstracted.Useful for deep insight into CNN behavior.

10. Tools and Technologies Used

Programming Language:Python

Widely used in AI/ML projects due to its simplicity and vast ecosystem of libraries.

Libraries and Frameworks:

TensorFlow or Keras

For building and training deep learning models (CNNs).

NumPy

For numerical operations and array handling.

Pandas

For data handling and analysis (optional, for custom datasets).

Matplotlib / Seaborn

For data and results visualization (accuracy curves, confusion matrix).

scikit-learn

For performance metrics and evaluation (e.g., confusion matrix, classification report).

Development Environment:Jupyter Notebook / Google ColabFor interactive development and experimentation.

Anaconda (optional)

For environment management and package installation.

Hardware (Optional):

GPU Acceleration (e.g., NVIDIA GPU / Google Colab GPU)

To speed up training of deep learning models.

11. Team Members and Contributions

RAMACHANDRAN D

- ✓ ***Data Cleaning***
- ✓ ***Feature Engineering***
- ✓ ***Documentation and Reporting***

PRIYADHARSHINI

- ✓ ***Exploratory Data Analysis (EDA)***
- ✓ ***Model Development***

PRABAVATHI

- ✓ ***Interface & Deployment***
- ✓ ***Model Development***