PHASE-2

Student Name: HEMALAKCHANA .N

Register Number: 510123106014

Institution: Adhiparaskathi College of Engineering

Department: B.E.Electronics and Communication Engineering

Date of Submission: 08-05-2025

GitHub Repository Link: https://github.com/Hema-lakchana/phase2.git

Recognizing handwritten digits with deep learning for smarter AI application

1.Problem Statement

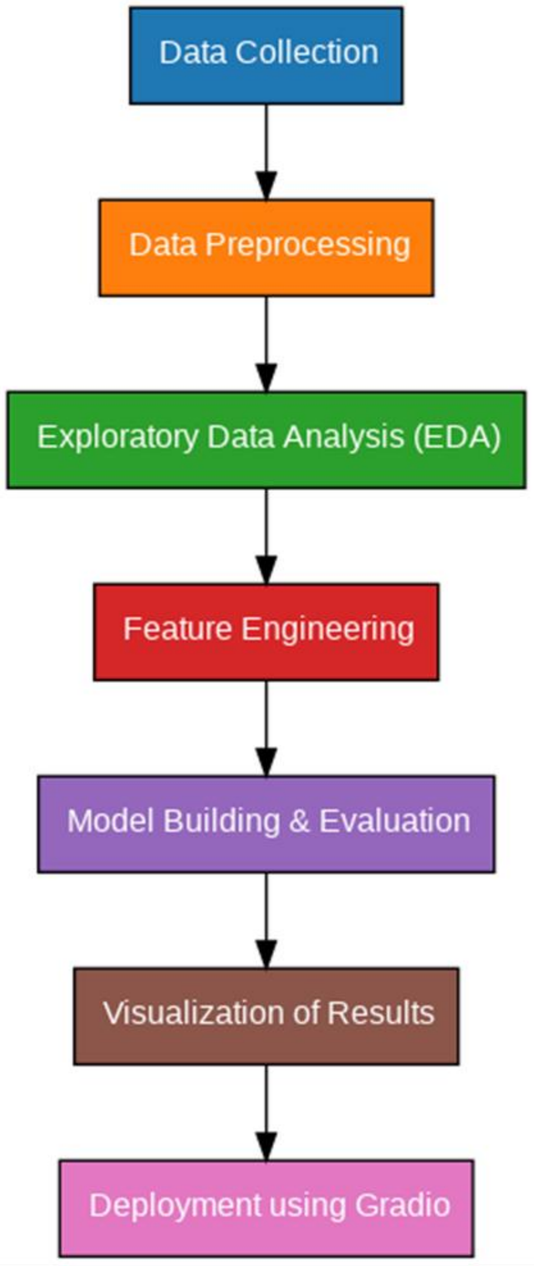
Recognizing handwritten digits accurately is a critical step toward developing smarter AI applications such as automated form reading, postal address recognition, and check processing. The goal is to build a deep learning model capable of identifying digits (0-9) from images of handwritten numbers.The goal is to ensure high classification accuracy, robustness against variations in handwriting, and efficiency in training and inference. This phase also includes evaluating the model using standard performance metrics and preparing the trained model for real-world AI applications.

2.Project Objectives

To design, train, and evaluate a deep learning model (using Convolutional Neural Networks - CNNs) that can classify handwritten digits from image data with high accuracy, contributing to AI applications that require optical character recognition (OCR). Implement a Convolutional Neural Network (CNN) architecture to capture spatial patterns and features of handwritten digit.

Assess the model's performance on a test set to measure accuracy, loss, and other relevant metrics, ensuring the model's robustness and precision in recognizing digits.

3.Flowchart of the Project Workflow


Data Collection
Data Preprocessing
Exploratory Data Analysis (EDA)
Feature Engineering
Model Building & Evaluation
Visualization of Results
Deployment using Gradio

4.Data Description

1. Dataset Overview

Dataset: MNIST (Modified National Institute of Standards and Technology)

Training Set: 60,000 grayscale images of handwritten digits.

Test Set: 10,000 grayscale images of handwritten digits.

2.Data Splitting

Training Set: 60,000 images used to train the deep learning model.

Validity Set: A subset (e.g., 10,000 images) of the training data used to tune the model and prevent overfitting.

Test Set:10,000 images used to evaluate the final model's performance

5. Data Preprocessing

1. Loading Data

Dataset: The dataset (e.g., MNIST) is typically provided in the form of images (28x28 pixels) along with labels (0-9).

Loading Libraries: Python libraries such as TensorFlow, Keras, or PyTorch are used to load the dataset efficiently. Common functions like tf.keras.datasets.mnist.load_data() (in TensorFlow/Keras) or torchvision.datasets.MNIST (in PyTorch) are used to load the data in training and test splits.

6. Exploratory Data Analysis (EDA)

- ***Understanding Data Structure***

Dataset Overview: The MNIST dataset consists of 60,000 training images and 10,000 test images, each of size 28x28 pixels.

Labels: Each image corresponds to a digit (0–9), which is the label for the task of classification.

Training data: (60000, 28, 28)

Test data: (10000, 28, 28)

Labels: (60000,) for training and (10000,) for testing.

- ***Distribution of Digits***

Purpose: To check if the dataset is balanced across all digit classes (0–9). Ideally, the data should have roughly the same number of samples for each digit to avoid class imbalance.

Visualization: *Plot a histogram or bar chart to visualize the distribution of labels across the training and test sets.*

7. Feature Engineering

- Image pixel intensities directly serve as features
- Data augmentation (e.g., rotation, shift, zoom) to increase robustness
- Flattening (for non-CNN models, optional)
- No manual feature extraction needed due to CNN Justify each feature added or removed.

8.Model Building

1. Model Selection

Chosen Architecture: Convolutional Neural Network (CNN)

Justification:

- Excellent at capturing spatial hierarchies in images.
- Effective at recognizing patterns like edges, curves, and shapes.
- Well-suited for grayscale image classification (e.g., MNIST).

2.Model Compilation

- Model.Compile(optimizer='adam',loss='categorical_crossentropy'
- metrics=['accuracy'])

Optimizer: Adam (adaptive learning rate and efficient convergence)Loss Function: categorical_crossentropy (suitable for multi-class classification)

Metric: accuracy (to evaluate performance)

9. Visualization of Results & Model Insights

- Training and Validation Accuracy & Loss Curves
- **Purpose:** To observe how well the model learned during training and whether overfitting or underfitting occurred.
- Example:

```
import matplotlib.pyplot as plt

# Plot accuracy
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()

# Plot loss
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()
    plt.ylabel('Accuracy')
```

10. Tools and Technologies Used

- **Programming Language:** Python
- **IDE/Notebook:** Google Colab, Jupyter Notebook, VS Code, etc.
- **Libraries:** pandas, numpy, seaborn, matplotlib, scikit-learn, XGBoost, etc.
- **Visualization Tools:** Plotly, Tableau, Power BI.]

11. Team Members and Contributions

- **CHARUMATHI.V-**Projectcoordinator and model developer
- **HEMALAKCHANA.N-**Data analyst and preprocessing lead
- **SARASWATHI.P-** Evaluation and metrics specialist
- **SWATHI.K-**Visualization and Reporting
- **SANGAVI.S-**Documentation and   presentation coordinator