Github Link: https://github.com/KaviyaK-04/Project-Deep-Learning.git

Project Title: Recognizing Handwritten Digits with Deep Learning for Smarter AI Applications

PHASE-II

Student Name: Kaviya K

Register Number:623023104019

Institution: Tagore Institute of Engineering and Technology -Salem

Department: Computer Science and Engineering

Date of Submission:08-05-2025

1. Problem Statement

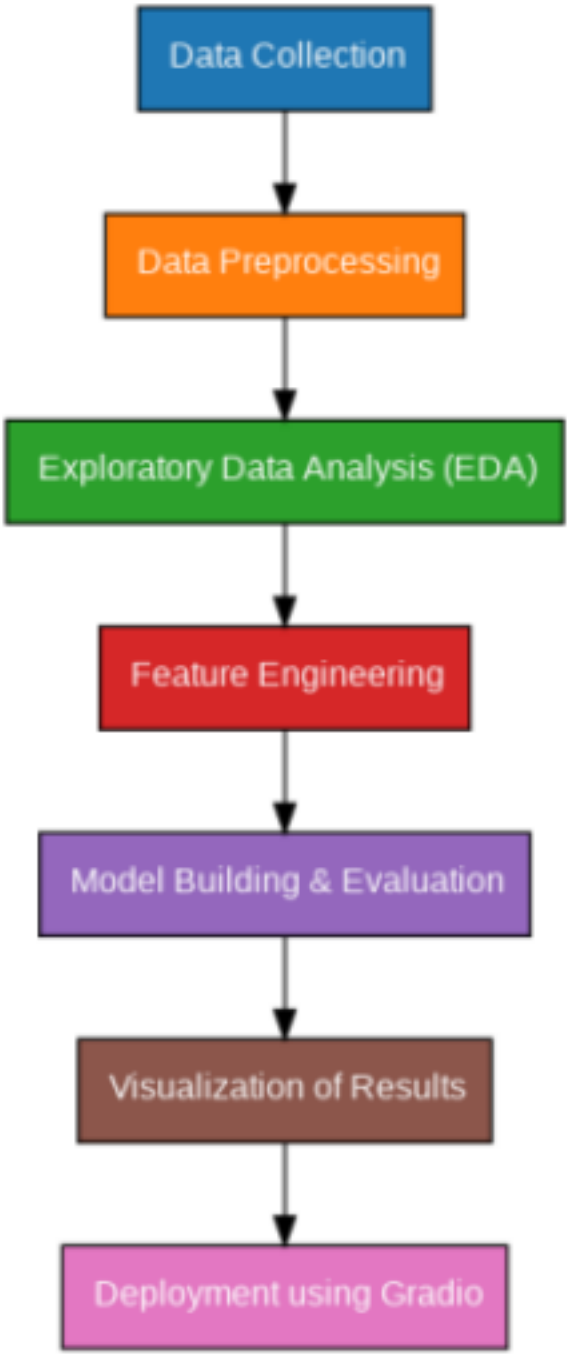
In many real-world applications, automated systems must accurately interpret handwritten input, such as digits on forms, checks, or postal codes. However, handwriting varies significantly between individuals, making this a challenging task. This project aims to develop a deep learning model that can accurately recognize handwritten digits using image data, thereby enhancing the intelligence and efficiency of AI-powered systems in fields like finance, education, and logistics.

2. Project Objectives

- To develop a deep learning model capable of recognizing handwritten digits from image data.
- To train the model using a standard dataset such as MNIST for accurate digit classification.
- To evaluate the model's performance using accuracy and loss metrics.

- To optimize the model for improved prediction accuracy and reduced error rate.

3. Flowchart of the Project Workflow



4. Data Description

- Dataset Name: MNIST Handwritten Digits Dataset
- Source: [Yann LeCun's website / Kaggle / TensorFlow Datasets]
- Type of Data: Image data (grayscale images of handwritten digits)
- Records and Features: 70,000 total images (60,000 training, 10,000 testing), each 28x28 pixels
- Target Variable: Digit class (0 through 9)
- Static or Dynamic: Static dataset
- Attributes Covered: Pixel intensity values (0 to 255)
- Dataset Link: https://www.kaggle.com/datasets/mloey1/ahcd1

5. Data Preprocessing

Load the MNIST dataset

→ Use a library like TensorFlow or Keras to import the dataset.

Normalize the images

→ Convert pixel values from **0–255** to **0–1** by dividing by 255.

Reshape the images

→ Change the shape from (28, 28) to (28, 28, 1) to match CNN input format.

One-hot encode the labels

→ Convert digit labels (0–9) into binary vectors (e.g., 3 → [0,0,0,1,0,0,0,0,0,0]).

Split the data

→ Use predefined training (60,000) and test (10,000) sets from MNIST.

6. Exploratory Data Analysis (EDA)

Univariate Analysis:

Image Visualization:

→ Display random samples of digit images with their labels to visually confirm data quality.

Label Distribution:

→ Plot a count plot/bar chart of digit classes (0–9) to ensure the dataset is balanced.

Pixel Intensity Distribution:

→ Use histograms to analyze how pixel values (0–255) are distributed across images.

Bivariate/Multivariate Analysis:

Average Digit Visualization:

→ Calculate and plot the average image per digit to understand stroke patterns.

T-SNE or PCA Projection (optional):

→ Reduce image data to 2D using PCA or t-SNE for visual clustering of digits.

Heatmaps of Digit Similarities:

→ Compute and visualize how similar different digits are based on pixel averages.

Key Insights:

- The dataset is well-balanced across all digit classes (0–9).
- Some digits (like 1 and 7) are visually distinct, while others (like 4 and 9) may look similar.
- The high dimensionality of images requires deep learning (e.g., CNNs)

7. Feature Engineering

- **Normalize Pixel Values**

  Scale pixel intensities from **0–255** to **0–1** by dividing by 255.

  ➤ Helps improve neural network training.

- **Reshape Images**

  Add a channel dimension to each image:

  From **(28,28)** to **(28,28,1)**.

  ➤ Required for convolutional layers in CNNs.

- **One-Hot Encode Labels**

  Convert digit labels (0–9) into binary vectors for classification.

- **Data Augmentation (Optional)**

  Apply transformations like rotation, zoom, and shift.

8. Model Building

- **Algorithm used:**

  Convolutional Neural Network (CNN)

  ➢ Specifically designed for image classification tasks

➢ Automatically extracts spatial features from image pixels

- **Model Selection Rationale:**

  CNNs are highly effective for image data.
  Able to learn edges, shapes, and patterns through layers.
  Better accuracy and generalization than traditional machine learning on image tasks.

- **Train-Test Split:**

  Standard MNIST Split:

  ➢ 60,000 training images

  ➢ 10,000 test images

  Or optionally use `train_test_split()` for a custom ratio (e.g., 80/20)

- **Evaluation Metrics:**

  Accuracy: % of correctly classified digits

  Confusion Matrix: Visual breakdown of predictions vs actual classes

  Precision, Recall, F1-Score (optional): For deeper error analysis

9. Visualization of Results & Model Insights

Model Performance Visualization:

- Accuracy Curve:

  ➢ Plot accuracy vs. epochs to visualize how well the model learns over time.

  ➢ Shows whether the model is improving with each training step.

- Loss Curve:

  ➢ Plot loss vs. epochs to check if the model is converging or overfitting.

  ➢ Helps identify early stopping points or the need for more epochs.

Confusion Matrix:

- Purpose: Visualize the classification performance across all digits (0-9).
- How: Plot a heatmap of true vs predicted values.

➢ This shows where the model struggles (e.g:confusion b/w 4 and 9).

Model Comparison:

- Performance Metrics:

  ➢ Plot metrics like accuracy, precision, recall, and F1-score for each digit.

➢ Compare performance across epochs or with different architectures. **User**

Testing / Deployment:

- Gradio Interface (optional):

  ➢ Create a Gradio interface where users can draw digits and get predictions from the trained model.

  ➢ This gives insight into how well the model performs with real-time data.

10. Tools and Technologies Used

Programming Language: Python 3

➢ Python is widely used for machine learning and deep learning tasks.

Notebook Environment: Google Colab

➢ Provides a cloud-based Jupyter notebook with free access to GPUs for faster

model training.

Key Libraries:

● **TensorFlow / Keras**

➢ For building and training the convolutional neural network (CNN) model.

● **NumPy**

➢ For numerical operations, especially for data manipulation.

● **Matplotlib, Seaborn**

➢ For visualizing training progress, loss/accuracy curves, and confusion

matrices. ● **Scikit-learn**

➢ For preprocessing tasks like train-test splitting, data scaling, and performance metrics evaluation.

● **Gradio (optional)**

➢ For creating an interactive interface where users can draw digits and get predictions from the model.

11. Team Members & Contributions

1. Gomathi B

○ Responsibilities:

- **Data Cleaning:** Ensured the dataset was in the proper format and handled any missing or corrupted data.

- **Data Preprocessing:** Normalized the image pixels and reshaped the data for the deep learning model.

2. Kaviya K

- Responsibilities:

  - **Exploratory Data Analysis (EDA):** Performed EDA by visualizing digit samples and understanding pixel distributions.

  - **Model Evaluation:** Worked on model performance evaluation, including accuracy, confusion matrix, and error analysis.

3. Gokila K

- Responsibilities:

  - **Model Development:** Designed and trained the Convolutional Neural Network (CNN) for handwritten digit recognition.

  - **Hyperparameter Tuning:** Experimented with different architectures, learning rates, and dropout rates to improve the model's performance.

4. ***Indhumathi K***

- Responsibilities:

  - **Documentation and Reporting:** Compiled project documentation, including methodology, findings, and results.

  - **User Interface:** Developed a Gradio interface for real-time digit prediction from user input.