Github Link: https://github.com/Dhashvandhan399/Handwriting-digits-with-smart-ai-application-

Project Title: Recognising hand written digits with deep learning for smarter ai application

PHASE-3

Student Name: Dhashvandhan.R

Register Number: 422723104024

Institution: V.R.S College of Engineering and Technology

Department: Computer science and Engineering

Date of Submission: 17.05.2025

1. Problem Statement

The goal is to develop a deep learning model capable of accurately recognizing and classifying handwritten digits (0–9) from images. Handwritten digits vary greatly in style, size, and orientation due to differences in individual writing patterns, making this a complex pattern recognition problem. Traditional image processing techniques struggle with such variability, whereas deep learning—especially Convolutional Neural Networks (CNNs)—can automatically learn relevant features from the raw pixel data.

This problem falls under the category of image classification, where the input is a grayscale image of a digit, and the output is a label representing the digit (0 through 9). Solving this problem enables the development of intelligent systems in fields like automated data entry, postal sorting, check recognition, and educational tools.

2. Abstract

Handwritten digit recognition is a fundamental challenge in the field of computer vision and pattern recognition, with broad applications in automation and intelligent systems. This project explores the use of deep learning techniques, particularly Convolutional Neural Networks (CNNs), to accurately classify handwritten digits from the MNIST dataset. Unlike traditional machine learning methods that rely on manual feature extraction, CNNs automatically learn spatial hierarchies of features directly from input images, making them highly effective for image classification tasks.

The system is trained on a large set of labeled digit images and is evaluated based on its accuracy and generalization to unseen data. With proper tuning, the model achieves high accuracy, demonstrating the capability of deep learning in recognizing patterns in unstructured data. The

success of this approach has implications for real-world applications such as automated postal systems, check verification, and educational grading tools. This project highlights the potential of deep learning to enhance machine intelligence and enable smarter, more adaptive AI applications.

3. System Requirement

1. Hardware Requirements
   Processor (CPU): Intel i5/i7 or AMD Ryzen 5/7 (Minimum: Dual-core CPU)

Graphics Processing Unit (GPU): NVIDIA GPU with CUDA support (e.g., GTX 1050 or better) for faster training (optional but recommended)

RAM: Minimum 8 GB (Recommended: 16 GB or more)

Storage: At least 5 GB of free disk space (for datasets, models, and environment)

Display: Any standard monitor (1080p recommended for visualization)

2. Software Requirements

Operating System: Windows 10/11, macOS, or any Linux distribution (Ubuntu preferred)

Programming Language: Python 3.7 or higher

IDE/Editor: Jupyter Notebook, VS Code, or PyCharm

4. Objectives

1. To understand the challenges involved in handwritten digit recognition, including variations in writing styles, shapes, and noise in input data.

2. To explore and apply deep learning techniques, specifically Convolutional Neural Networks (CNNs), for image classification tasks.

3. To implement a robust model that can classify handwritten digits (0–9) from the MNIST dataset with high accuracy.

4. To train and validate the model using appropriate metrics such as accuracy, precision, recall, and confusion matrix analysis.

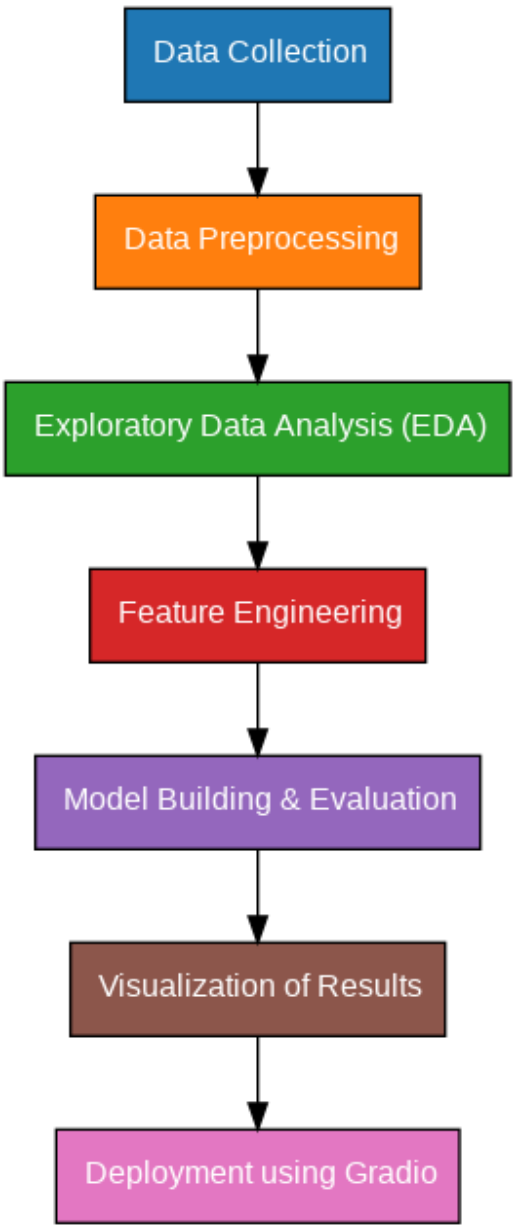
5. To evaluate the model's generalization ability on unseen data and improve its performance through techniques like data augmentation, regularization, and hyperparameter tuning.

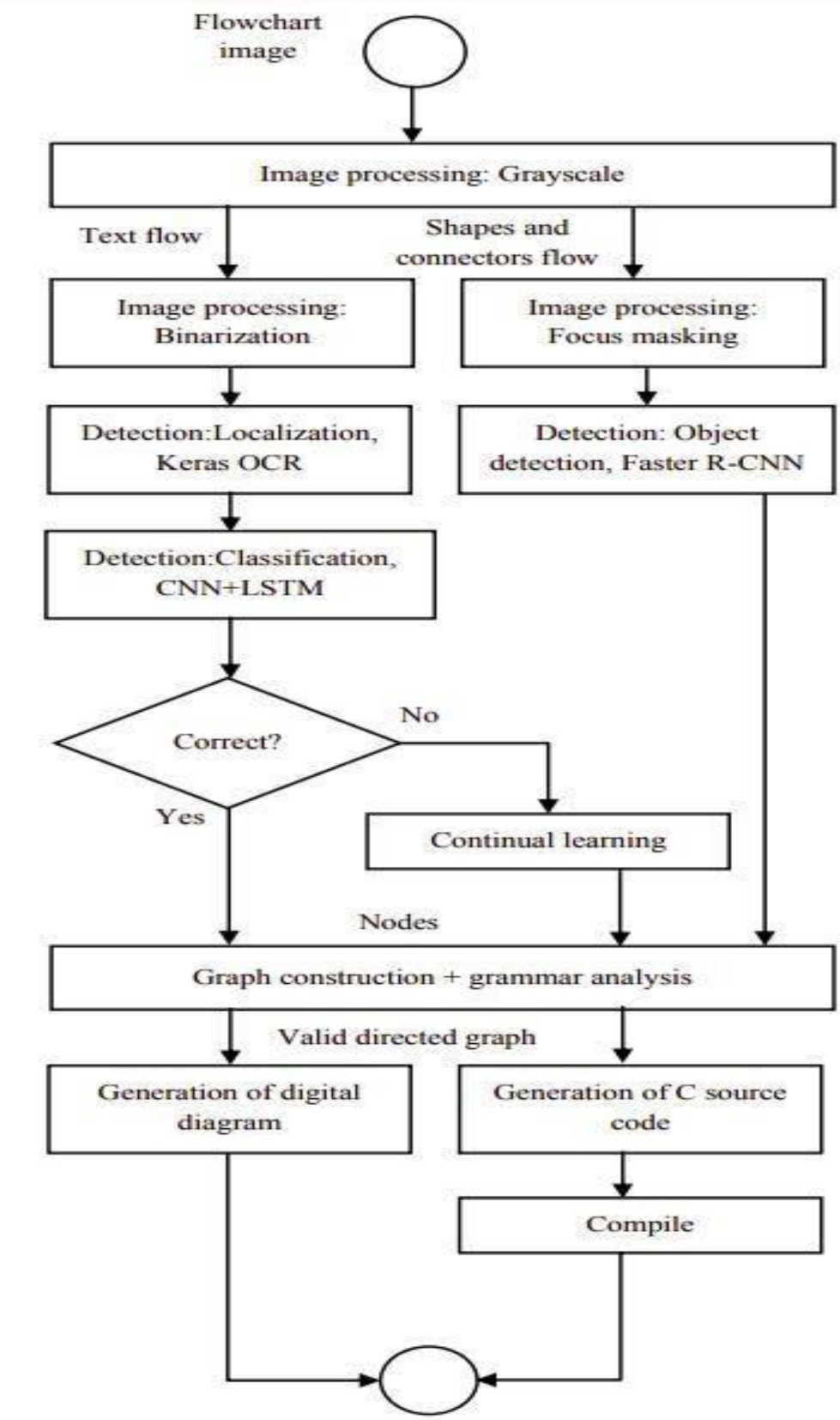
6. To demonstrate the practical applications of the developed system in real-world use cases like postal code recognition, check reading, and automated form processing.

7. To contribute towards building smarter AI systems that can interpret and interact with humanlike data efficiently.

5. Flowchart of the Project Workflow

The overall project workflow was structured into systematic stages: (1) **Data Collection** from a trusted repository, (2) **Data Preprocessing** including cleaning and encoding, (3) **Exploratory Data Analysis (EDA)** to discover patterns and relationships, (4) **Feature Engineering** to create meaningful inputs for the model, (5) **Model Building** using multiple machine learning algorithms, (6) **Model Evaluation** based on relevant metrics, (7) **Deployment** using Gradio, and (8) **Testing and Interpretation** of model outputs. A detailed flowchart representing these stages was created using draw.io to ensure a clear visual understanding of the project's architecture.




Flowchart image
Image processing: Grayscale
Text flow
Shapes and connectors flow
Image processing: Binarization
Image processing: Focus masking
Detection:Localization, Keras OCR
Detection: Object detection, Faster R-CNN
Detection:Classification, CNN+LSTM
Correct?
No
Yes
Continual learning
Nodes
Graph construction + grammar analysis
Valid directed graph
Generation of digital diagram
Generation of C source code
Compile

6. Dataset Description

Source:

tensorflow.keras.datasets torchvision.datasets

http://yann.lecun.com/exdb/mnistSample dataset **Dataset**

Type:

Supervised, Image Classification **Data**

Format:

Grayscale images

Image size: 28 x 28 pixels

File format: typically provided in .idx or .csv formats **Number**

of Samples:

Training set: 60,000 images

Test set: 10,000 images Total:

70,000 labeled images

Classes:

10 digit classes: 0 through 9 **Label**

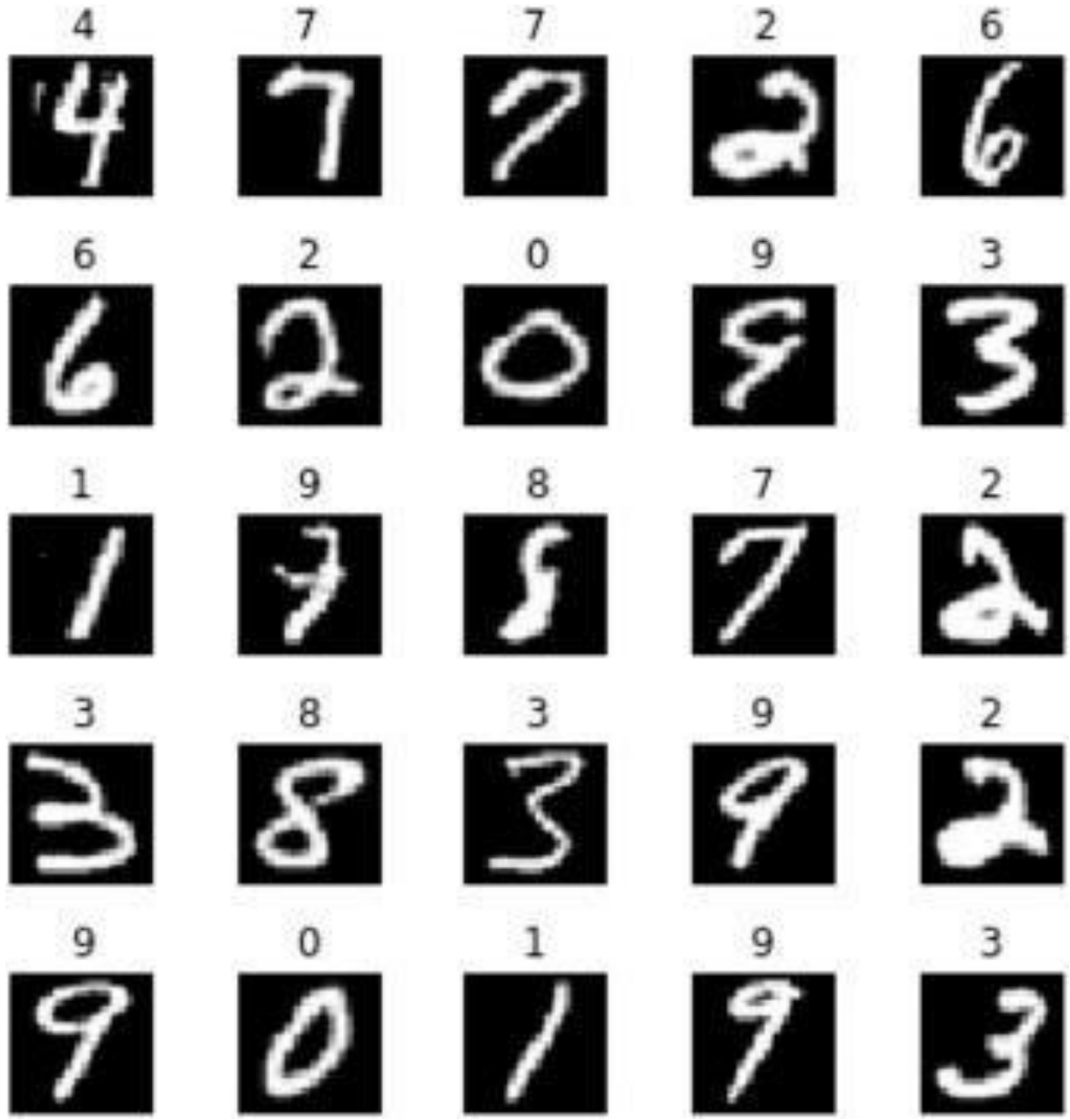
Format:

Each image is associated with a single digit label (0–9), indicating the correct class.

Features:

Pixel values range from 0 (black) to 255 (white).

Each image is a 2D matrix of 784 features (28x28 pixels flattened).



7. Data Preprocessing

For recognizing handwritten digits with deep learning, data preprocessing usually involves

1. **Grayscale normalization:** Convert the digit images to grayscale (if not already) and normalize pixel values to a range like 0 to 1 by dividing by 255. This helps the model learn more effectively.

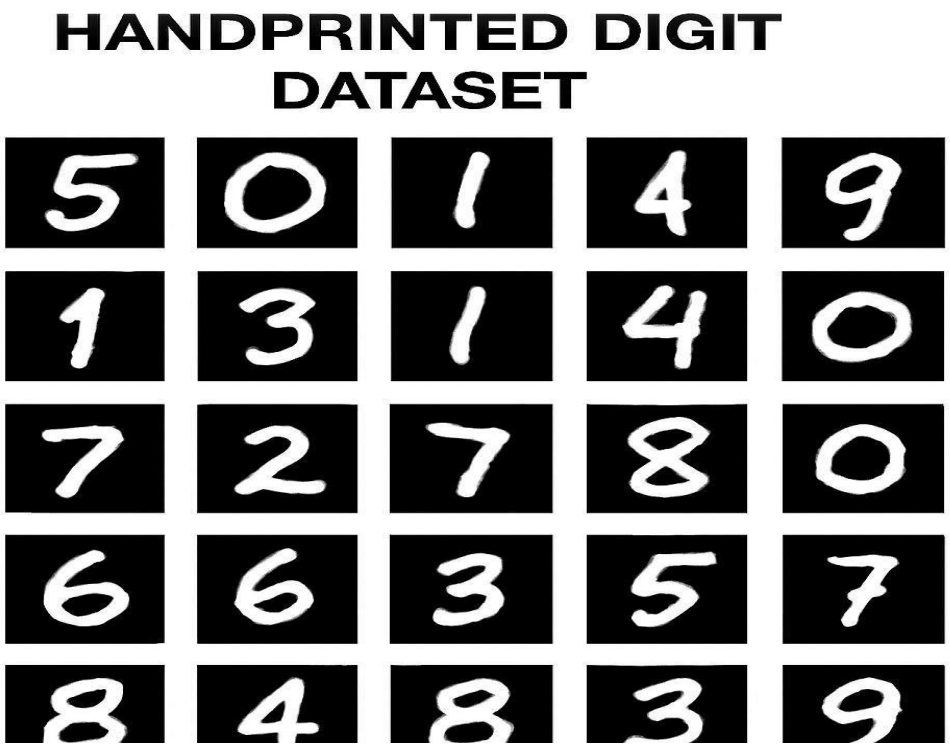
2. **Resizing images:** Ensure all images have the same size (commonly 28x28 pixels for MNIST) so they fit the input layer of the neural network.

3. **Flattening or reshaping:** Depending on the model, images might be flattened into vectors or kept as 2D arrays for convolutional layers.

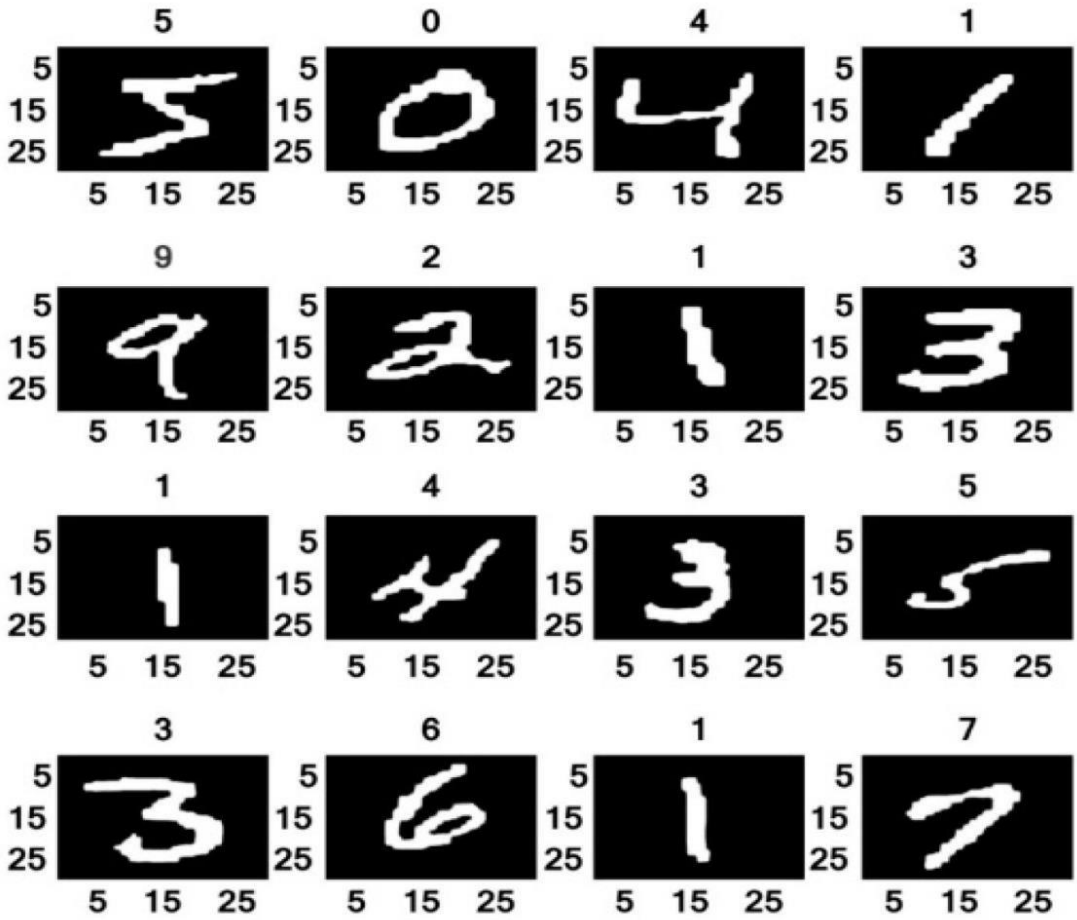
4. **Label encoding:** Convert the digit labels (0-9) into a format suitable for training, often onehot encoding for classification.

5. **Data augmentation (optional):** Apply transformations like rotation, shifting, or zooming to artificially increase training data diversity and improve model robustness.

6. **Splitting data:** Divide the dataset into training, validation, and test sets to evaluate model performance fairly.



,



8. Exploratory Data Analysis (EDA)

1. Understanding the Dataset

Identify the number of samples and their dimensions (e.g., 28x28 images).

Check how many total images are available.

Verify the number of classes (digits 0–9).

2. Visualizing Sample Images

Display random digit images to see variations in handwriting.

Plot one example per class to understand how each digit looks.

3. Class Distribution

Plot a bar chart showing how many samples exist for each digit (0–9).

Check for class imbalance, which can affect model training.

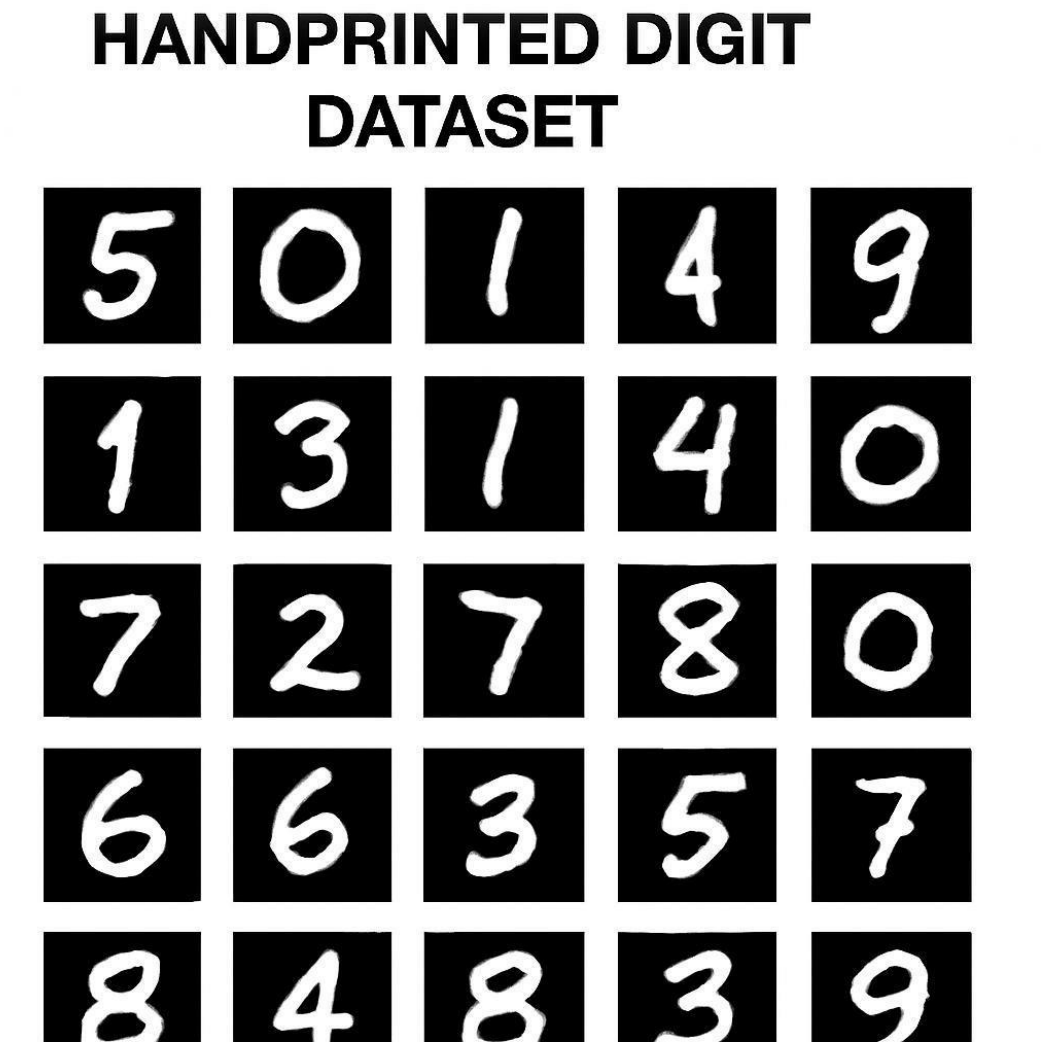
4. Pixel Value Distribution

Analyze pixel intensity values (usually 0 to 255).

Use histograms to understand the spread and detect noise.

5. Image Structure

Calculate average images per class to observe typical pattern



9. Feature Engineering

1. Pixel Normalization:

Scale pixel values from 0–255 to 0–1 (or -1 to 1) to help the neural network train efficiently.

2. Shape & Structure Features (less common in CNNs):

Calculate statistical features such as:

Mean and standard deviation of pixel values.

Total number of white pixels.

Center of mass (to understand alignment).

Symmetry (horizontal or vertical).

3. Edge Detection (if using classical models):

Apply filters like Sobel or Canny to extract edges of digits.

4. Histogram of Oriented Gradients (HOG):

A powerful technique for classical ML (e.g., SVMs) that captures shape and texture.

5. Image Augmentation (a form of feature enrichment):

Rotate, scale, shift, or shear images to create new training samples and expose the model to variations.

6. Dimensionality Reduction (optional):

Techniques like PCA (Principal Component Analysis) can be used to reduce input size while preserving important features.

10. Model Building

Convolutional Neural Network (CNN): Ideal for image data due to its ability to detect spatial patterns.

Frameworks like TensorFlow or PyTorch are commonly used

11. Model Evaluation

1. Accuracy

The most common metric for classification.

Measures the percentage of correctly predicted digits.

2. Confusion Matrix

Shows a table of actual vs. predicted digit classes.

Helps identify specific digits the model struggles with.

3. Classification Report

Provides precision, recall, and F1-score for each class (digit 0–9).

4. Loss Curve & Accuracy Curve

Plots of training and validation loss/accuracy over epochs.

Helps detect overfitting or underfitting.

5. Error Analysis

Visualize misclassified images to understand failure cases and improve the model.

12. Deployment

Steps for Model Deployment

1. Save the Trained Model

Save the model in a format that can be reused later (e.g., .h5 for Keras models)

model.save('digit_recognition_model.h5')

2. Load the Model in a Backend Environment

3. Create a Prediction API (e.g., using Flask)

4. Frontend (Optional)

Create a simple UI (e.g., with HTML/JavaScript) where users can draw digits or upload images.

5. Host the App

Use services like Heroku, Render, AWS, or Google Cloud to deploy the backend API.

Ensure the model runs efficiently and securely.

6. Monitor and Update

Track performance with logs and user feedback.

Update the model if accuracy drops or if new data becomes available.

13. Source Code

```
from flask import Flask, request, jsonify import
numpy as np from tensorflow.keras.models
import load_model from PIL import Image
import io

app = Flask(__name__) model =
load_model("digit_model.h5")

@app.route('/predict', methods=['POST']) def
predict():     if 'file' not in request.files:         return
jsonify({'error': 'No file uploaded'}), 400     file =
request.files['file']     img =
Image.open(file).convert('L').resize((28, 28))     img =
np.array(img).astype('float32') / 255.0     img =
img.reshape(1, 28, 28, 1)     prediction =
model.predict(img)     digit =
int(np.argmax(prediction))     return
jsonify({'prediction': digit})

if __name__ == '__main__':  app.run(debug=True)
```

Sample output

14. Future Scope

1. Integration with Real-World Applications

Banking: Automated check processing and signature verification.

Postal Services: Automated ZIP code reading for mail sorting.

Education: Digit recognition for online exams or digitized handwritten notes.

2. Expansion to Multilingual Handwriting

Extend recognition to non-Latin scripts (e.g., Devanagari, Arabic, Chinese characters) using similar CNN-based architectures.

3. Improved Accuracy with Advanced Models

Incorporate more advanced architectures like ResNet, Capsule Networks, or Vision Transformers to boost accuracy and robustness.

4. On-Device Inference

Deploy lightweight models on mobile devices or IoT edge devices for real-time offline digit recognition.

5. Handwriting Style Adaptation

Implement few-shot learning or meta-learning to adapt to new handwriting styles with minimal new data.

6. End-to-End OCR Integration

Combine with Optical Character Recognition (OCR) systems for full document understanding and digit-text parsing.

7. Use in Education and Special Needs

Assistive technologies for students with disabilities (e.g., dysgraphia) by converting handwriting to readable text or voice.

8. Security and Fraud Detection

Use in verifying handwritten signatures and detecting forgeries by analyzing subtle handwriting features.

13. Team Members and Roles

Dhashvandhan.R- *"Data Preprocessing & EDA Lead Model Development & Optimization Specialist"*
Responsible for cleaning and preparing the dataset, handling missing values, Encoding features, and conducting exploratory data analysis.

Dharanidharan.M*-Focuses on implementing collaborative and content-based filtering models, hybrid Approaches, and tuning model parameters for optimal performance.*

Dhanush kumar.V-*"Feature Engineering & Evaluation Analyst"*
Handles feature extraction and transformation, builds similarity matrices, and Evaluates model accuracy using metrics like RMSE, Precision@K, and Recall@K.

Dheenadhayalan.S- *"Project Coordinator & Deployment Engineer"*
Oversees overall project integration, coordinates tasks among team members, and Develops the Streamlit web app for deployment of the recommendation system.