Phase-2

RECOGNIZING HANDWRITTEN DIGITS WITH DEEP LEARNING FOR SMARTER AI APPLICATIONS

Student Name: G. KOMATHY

Register Number: 513523104039

Institution: Annai Mira College Of Engineering And Technology

Department: B.E Computer Science Engineering

Date of Submission: 05.05.2025

Github Repository Link:
https://github.com/priyakomathy/EBPS---DS---RECOGNISING-HAND-WRITTEN-DIGITS.git

1. Problem Statement

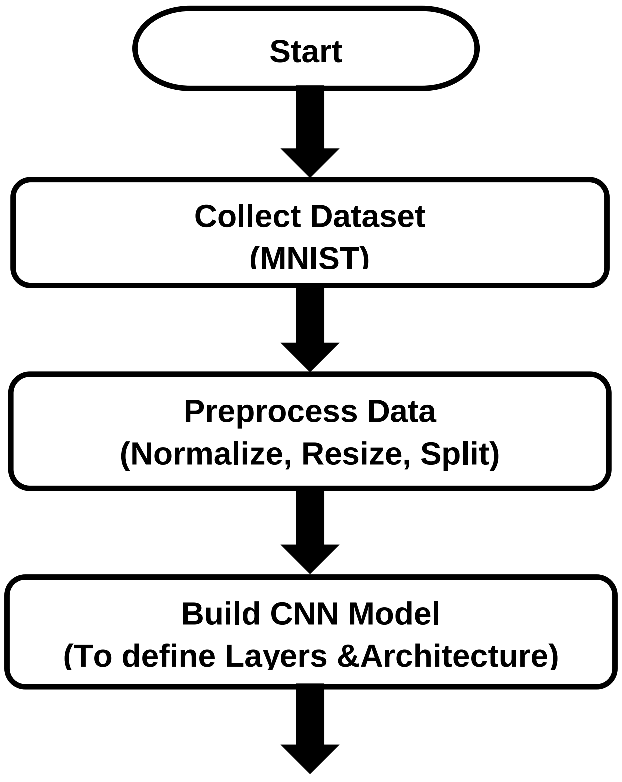
In many real-world scenarios, handwritten digits appear in documents, forms, and notes that need to be interpreted by machines. However, variations in human handwriting make this a complex task for traditional computer vision systems. This project addresses the challenge of accurately recognizing handwritten digits (0–9) using deep learning techniques, with the goal of enhancing the capabilities of intelligent systems.

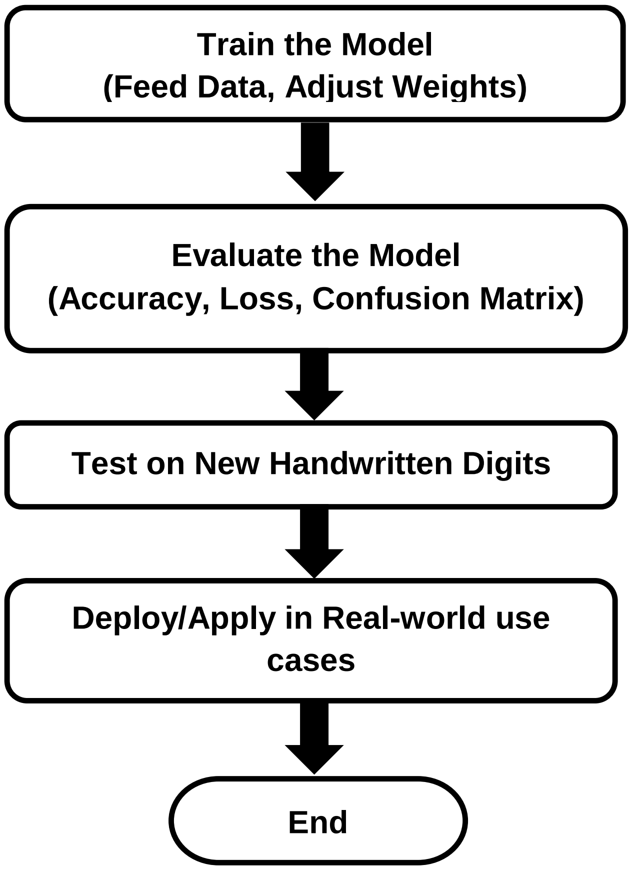
By developing a robust deep learning model, this solution aims to improve applications such as: Image classification for identifying digits in images. Optical Character Recognition (OCR) for digitizing handwritten content. Smart automation systems, such as automated grading, form processing, and document analysis.

2. Project Objectives

1. To build a deep learning model that can accurately recognize handwritten digits from 0 to 9.
2. To use Convolutional Neural Networks (CNNs) for effective image-based digit classification.
3. To test and evaluate the model’s accuracy using a standard dataset like MNIST.
4. To apply the model in real-life scenarios such as OCR, automated grading, and document analysis.

3. Flowchart of the Project Workflow


Start
Collect Dataset (MNIST)
Preprocess Data (Normalize, Resize, Split)
Build CNN Model (To define Layers &Architecture)


Train the Model
(Feed Data, Adjust Weights)
Evaluate the Model
(Accuracy, Loss, Confusion Matrix)
Test on New Handwritten Digits
Deploy/Apply in Real-world use cases
End

4. Data Description

1. Dataset Name and Source: MNIST dataset, available from Kaggle, TensorFlow, and other open sources.
2. Type of Data: Image data (unstructured) - grayscale images of handwritten digits.
3. Number of Records and Features: Totally 70,000 images, 60,000 for training and 10,000 for testing. Each image is 28x28 pixels (784 features).
4. Static or Dynamic dataset: This is a static dataset, the data does not change over time.
5. Target Variable (Supervised Learning): The digit label (0-9) representing the correct handwritten number in each image.

5. Data Preprocessing

1. Import the dataset (from TensorFlow/Kaggle).
2. Convert images to grayscale if needed (MNIST is already grayscale).
3. Normalize pixel values to a range between 0 and 1.
4. Reshape images to match the input format of the model (28x28 or 28x28x1).
5. Split the data into training and testing sets.
6. One-hot encode the labels (to convert digits 0–9 into vectors for training).

6. Exploratory Data Analysis (EDA)

1. Univariate Analysis: Plotted the count of each digit (0–9) to check if all classes are balanced. Checked pixel value distribution using histograms.
2. Bivariate/Multivariate Analysis: Displayed sample images with their labels to see how different digits look. Observed how pixel patterns relate to specific digits.
3. Insights Summary: The dataset is clean and balanced. Digits have clear visual patterns. CNN can learn these patterns to improve accuracy.

7. Feature Engineering

1. Used raw pixel values as features: Each image (28x28 pixels) gives 784 features when flattened.
2. Normalized pixel values: Scaled all pixel values from 0–255 to 0–1 for better model performance.
3. Reshaped data for CNN input: Changed shape from 784 to 28x28x1 to work with convolutional layers.
4. One-hot encoded labels: Converted digit labels (0–9) into one-hot format to help the model classify correctly.

5. No manual feature creation or splitting was needed, since CNNs automatically learn important patterns from image data.

8. Model Building

1. We used two models :
    i. CNN – best for image data like handwritten digits.
    ii. K-Nearest Neighbors (KNN) – a simple model that compares pixel patterns.
2. Why these models? CNN is powerful for deep learning on images. KNN is easy to understand and gives a good comparison.
3. We split the data into training and testing parts so we can check how well the model works on unseen data.
4. We trained the models and measured how good they are using: Accuracy, Precision, Recall, F1-score

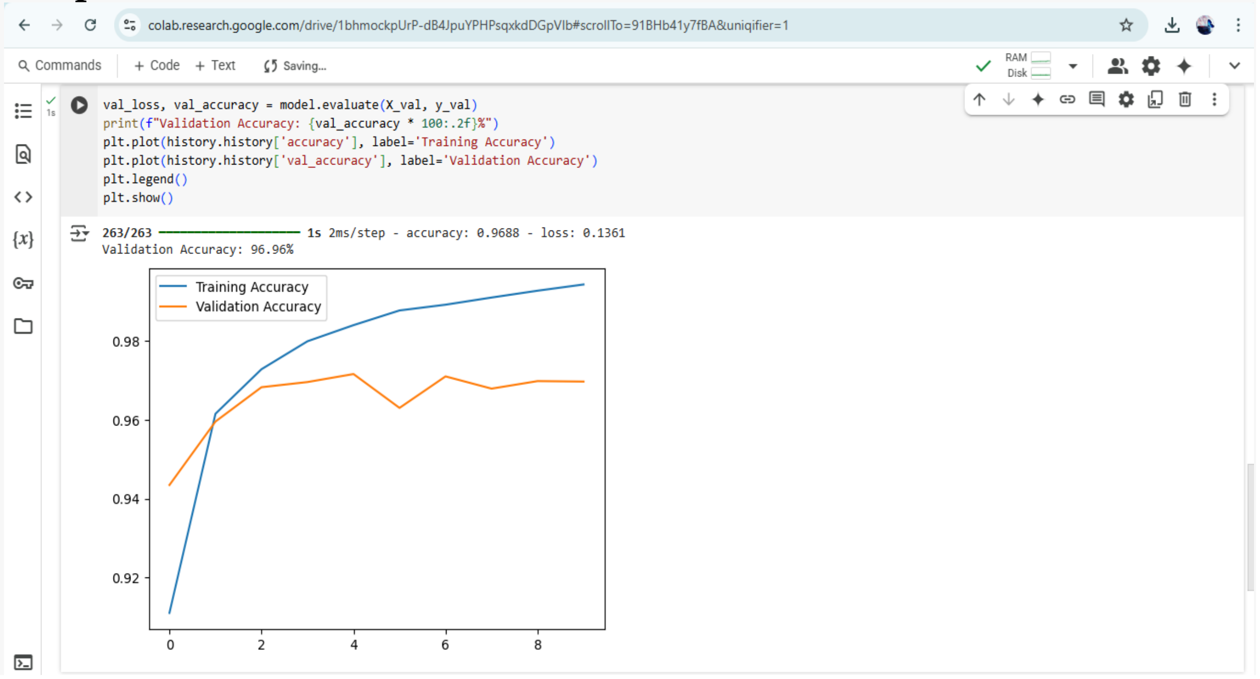
9. Visualization of Results & Model Insights

1. Confusion Matrix: Shows how many digits the model correctly predicted and where it got confused (e.g., mistaking 4 for 9).
2. Accuracy Comparison (Bar Chart): Displays how different models (like KNN, Random Forest) performed in terms of accuracy.
3. Feature Importance (if applicable): If using models like Random Forest, shows which pixel areas were most useful in predicting digits.
4. Sample Predictions with Images: Shows actual digit images alongside predicted labels, especially for wrong predictions, to understand errors.
5. Insights: Most models struggle with digits that look similar (e.g., 5 and 8).
6. Visuals help in choosing the best model and understanding its behavior.

Program:

```
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()
```

Output:



```
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()
```

```
263/263 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9688 - loss: 0.1361
Validation Accuracy: 96.96%
```

colab.research.google.com/drive/1bhmockpUrP-dB4JpuYPHPsqxkdDGpVlb#scrollTo=91BHb41y7fBA&uniqifier=1
Commands
+ Code
+ Text
Saving...
RAM
Disk
Training Accuracy
Validation Accuracy
0.98
0.96
0.94
0.92
0
2
4
6
8

10. Tools and Technologies Used

1. Python – Main programming language for coding.
2. Jupyter Notebook – For writing and running code interactively.
3. NumPy & Pandas – For data handling and processing.
4. Matplotlib & Seaborn – For plotting graphs and visualizing data.
5. Scikit-learn – For machine learning models and evaluation.
6. TensorFlow / Keras – For building deep learning models (like CNN).
7. MNIST Dataset – Standard dataset of handwritten digits used for training and testing.

11. Team Members and Contributions

1. S. M. Lekha Sri -Project Leader: Dataset selection, model training, and result evaluation.
2. M. Kumaran -Data Analyst: Data preprocessing, Exploratory Data Analysis and Feature Engineering.
3. G. Madhan Kumar - ML Engineer: Implemented machine learning algorithms and performance tuning.
4. G. Komathy - Documentation & Presentation: Prepared report, slides, and visualizations.