Handwritten Digit Recognition is one of the fundamental problems in computer vision and machine learning. It is the process of teaching machines to interpret and classify images of handwritten digits (0–9) into their respective categories.

Since handwritten styles vary from person to person (size, thickness, slanting, spacing, etc.), a simple rule-based program cannot handle such variations. Hence, advanced approaches like **Machine Learning** and **Deep Learning** are used to solve this problem.

In this project, we aim to build a model that can recognize digits from handwritten images using **Convolutional Neural Networks (CNNs)**. The project uses the **MNIST dataset**, which is a benchmark dataset containing thousands of digit samples.

This project is important because it forms the foundation for real-world applications such as:

* Reading postal addresses (ZIP codes).
* Bank cheque processing.
* Recognizing digits on forms and applications.
* Automated data entry systems.

**Objectives**

The main objectives of the project are:

1. Develop a recognition system that can correctly classify handwritten digits (0–9).
2. Understand image preprocessing techniques such as resizing, normalization, and reshaping.
3. Implement machine learning and deep learning models (MLP, CNN) for digit recognition.
4. Evaluate performance metrics like accuracy, loss, confusion matrix, and precision-recall.
5. Build a simple user interface (optional) that allows users to draw digits and receive predictions.

The project methodology can be divided into several phases:

### **Step 1: Data Collection**

* Use the **MNIST Dataset** which contains:
  + 60,000 training images.
  + 10,000 testing images.
* Each image is **28x28 pixels** in grayscale.

### **Step 2: Data Preprocessing**

* Convert pixel values (0–255) into **normalized values (0–1)** for faster training.
* Reshape the dataset to match the CNN input requirements: (28,28,1) format.
* Convert class labels into **one-hot encoded format** (e.g., digit 5 → [0,0,0,0,0,1,0,0,0,0]).

### **Step 3: Model Building**

We will use **Convolutional Neural Networks (CNNs)** because they are highly effective in image classification.

* **Input Layer**: Accepts 28x28 grayscale image.
* **Convolutional Layers**: Extracts spatial features (edges, curves, patterns).
* **Pooling Layers**: Reduces image size to avoid overfitting and speed up training.
* **Fully Connected Layers (Dense)**: Learns patterns and relationships.
* **Output Layer**: 10 neurons with **Softmax activation** (for digits 0–9).

### **Step 4: Model Training**

* Use training data (60,000 samples).
* Apply **Backpropagation and Gradient Descent** to minimize error.
* Optimizer: **Adam / SGD.**
* Loss Function: **Categorical Crossentropy**.
* Train for multiple **epochs** until accuracy stabilizes.

### **Step 5: Model Testing & Evaluation**

* Test with the **10,000 unseen images.**
* Measure accuracy (expected > 98% with CNN).
* Plot **confusion matrix** to see which digits get misclassified.
* Compare performance with simpler models (Logistic Regression, SVM, MLP).

## ****Planning****

## **Set up environment**: Install Python, TensorFlow/Keras, NumPy, Matplotlib.

1. **Load Dataset**: Use keras.datasets.mnist to load data.
2. **Preprocess**: Normalize, reshape, and one-hot encode.
3. **Build CNN Model**: Define architecture with Conv2D, MaxPooling, Flatten, Dense layers.
4. **Compile & Train**: Use Adam optimizer, categorical crossentropy loss, track accuracy.
5. **Evaluate**: Check test set accuracy, visualize misclassified samples.