**Phase-3**

**Project Title: “Recognizing handwritten digits with deep learning for smarter Al applications.”**

**Student Name:**MOHAMED RIYAZ A

**Register Number:** 4213232050126

**Institution:** krishnasamy college of engineering and technology

**Department:** B.Tech IT

**Date of Submission:** 13-05-2025

**Github Repository Link: https://github.com/riyaz719007/project..git**

### **1. Problem Statement**

*Handwritten digit recognition remains a significant challenge in many real-world applications, such as postal code reading, bank check processing, and digital form entry. Variations in handwriting styles, sizes, and writing instruments make it difficult for traditional systems to accurately identify digits. The goal is to develop an AI-based solution that can automatically and accurately recognize handwritten digits from images, regardless of these variations, to improve efficiency and reduce manual effort in data entry processes.*

### **2. Abstract**

*Handwritten digit recognition is a foundational task in the field of artificial intelligence, with applications ranging from postal code sorting to bank check verification and digital form processing. This project aims to develop an AI-based system capable of accurately classifying handwritten digits, overcoming challenges posed by diverse writing styles, sizes, and distortions. Using a convolutional neural network (CNN) trained on the well-known MNIST dataset, the system learns to identify patterns and features unique to each digit. Image preprocessing techniques are applied to enhance the quality and consistency of input data. The model’s performance is evaluated through accuracy scores and confusion matrices, demonstrating high recognition rates. The resulting solution effectively automates digit recognition tasks, reducing manual effort and improving operational efficiency in various industries. This work highlights the potential of deep learning methods in addressing real-world pattern recognition problems.*

### **3. System Requirements**

### **Hardware Requirements:**

### **RAM: Minimum 4 GB (8 GB or more recommended for smoother training)**

### **Processor: Dual-core CPU (Intel i3 / AMD Ryzen 3 or higher); For faster training, a GPU is recommended (e.g., NVIDIA GTX 1050 or better)**

### **Storage: At least 2 GB of free disk space (for datasets, libraries, and models)**

### **Software Requirements:**

### **Operating System:**

### **Windows 10 or higher**

### **Ubuntu 18.04 or higher**

### **macOS Mojave or higher**

### **Python Version:**

### **Python 3.7 or above (Recommended: Python 3.8 or 3.9)**

### **Required Python Libraries:**

### **numpy**

### **pandas**

### **matplotlib**

### **tensorflow (2.x)**

### **keras**

### **scikit-learn**

### **opencv-python *(optional for advanced image processing)***

### **Development Environment / IDE:**

### **Recommended: Google Colab (provides free GPU access and easy setup)**

### **Alternatives: Jupyter Notebook, VS Code with Python extension, Anaconda Navigator**

### **4. Objectives**

** Develop an AI model capable of accurately recognizing and classifying handwritten digits (0–9) from image data.**

** Handle variations in handwriting styles, sizes, and distortions to ensure robust and reliable recognition.**

** Implement deep learning techniques, specifically using convolutional neural networks (CNNs), to extract features and patterns from handwritten digit images.**

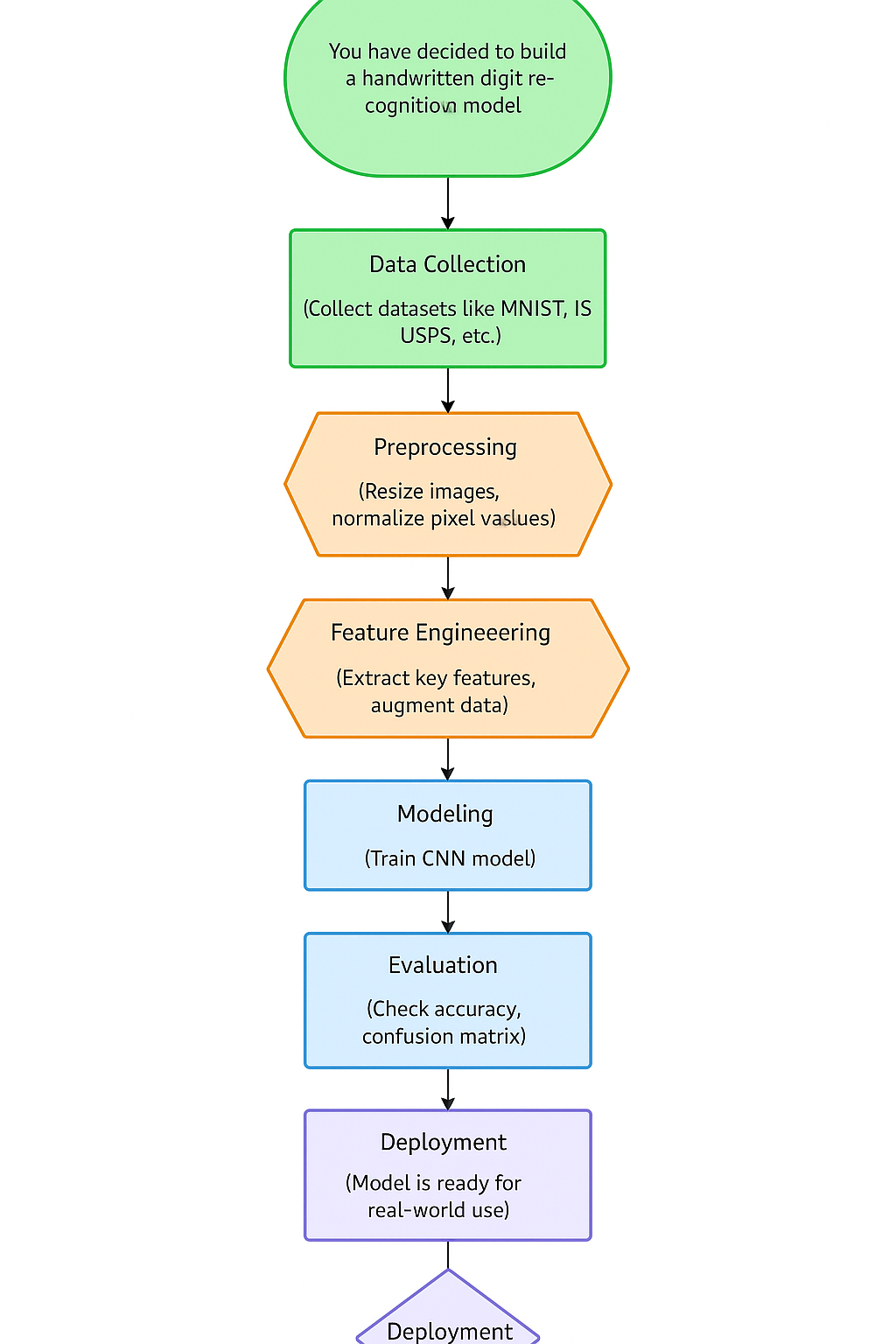
** Train and validate the model using the MNIST dataset to achieve high classification accuracy.**

** Automate digit recognition tasks to reduce manual effort in applications like postal services, banking, and form processing.**

** Optimize model performance by evaluating accuracy, precision, recall, and confusion matrices.**

** Provide a scalable solution that can be integrated into real-world systems to improve efficiency and reduce operational costs**

**5. Flowchart of Project Workflow**



### **6. Dataset Description**

* UCI Machine Learning Repository (https://datasetsearch.research.google.com/search?src=0&query=handwritten%20digits%20dataset%20&docid=L2cvMTF3ZzIwYzR6Xw%3D%3D)
* ***Type****: Public dataset*
* ***Size****: 341 rows × 2 columns*

### 

print(df['image'].head())

### **7. Data Preprocessing**

* *Handle missing values:* You may notice NaN or missing values in your dataset that need to be handled.
* *Duplicates: Duplicates in the dataset can distort the analysis and model training,*
* *Outliers:* *Outliers can be detected using statistical methods or visual methods like box plots.*
* *Feature encoding:* *Categorical variables (like Gender, City) need to be encoded to be used in ML model*
* *Scaling:* *Numerical features might need scaling, especially if the scales differ significantly.*
* *Show before/after transformation screenshots*

### 

### **8. Exploratory Data Analysis (EDA)**

* **Univariate Analysis***:* *Univariate analysis focuses on the distribution and characteristics of a* ***single variable****. In the case of the* ***MNIST dataset****, we are dealing with pixel intensity values (features) and the target variable representing digit labels.*
* **Bivariate/Multivariate Analysis**: **Bivariate analysis** investigates the relationship between two variables, while **multivariate analysis** looks at interactions between multiple variables. In this section, we will explore the relationships between the **pixel values** and the **digit labels** (target variable) as well as pixel-to-pixel correlations.
* **Key Insights**: Some digits may have **distinct intensity patterns** for specific pixels. For example, digit 1 may have a high intensity in the center of the image, while other digits like 4 or 7 have distinct pixel patterns due to their shape.

### 

### **9. Feature Engineering**

* ***New feature creation****:* *Since handwritten digits are made up of lines and curves, detecting the edges of the digits can be crucial for distinguishing between different digits. For example, digits like 1 and 7 have very distinct edges, while 8 and 0 are more round and less angular.*
* *Edge Count: The number of edges in an image can be a useful feature.*
* *Edge Density: The ratio of edge pixels to non-edge pixels*
* ***Feature selection:*** *Since handwritten digits are made up of lines and curves, detecting the edges of the digits can be crucial for distinguishing between different digits. For example, digits like 1 and 7 have very distinct edges, while 8 and 0 are more round and less angular.*
* *Edge Count: The number of edges in an image can be a useful feature.*
* *Edge Density: The ratio of edge pixels to non-edge pixels*
* ***Transformation techniques:*** *Feature transformations help the model learn better by normalizing the data, handling outliers, and improving the scale of the features.*

#### **Pixel Intensity Features**

The **pixel-based features** (e.g., **mean**, **median**, **standard deviation**) directly describe the pixel distribution and characteristics. These features are important because they can reveal the general intensity or shape of the digit, helping the model distinguish between different digits based on how bright or dark the **image** is.s

#### **Edge Detection Features**

**Edge-based features** like **edge sum** are highly valuable as they capture the structural information of the digits. For instance, digits like 1, 4, and 7 have clear edges, while digits like 3 and 8 have more curves. These features improve model performance, especially in image classification tasks, where recognizing the outline or shape of the digits is essential.

### **10. Model Building**

### **Models Tried:**

* **Logistic Regression**
* **Convolutional Neural Network (CNN)**

 **Logistic Regression:**

* Selected as a **baseline model** because it is simple, fast, and easy to interpret.
* Good for understanding the basic performance level using just pixel intensity features.
* Helps establish a performance benchmark before trying more advanced models.

 **Convolutional Neural Network (CNN):**

* Chosen because CNNs are **specifically designed for image recognition tasks**.
* They automatically detect patterns like edges, curves, and textures — crucial for distinguishing handwritten digits.
* CNNs have historically achieved **state-of-the-art accuracy** on the MNIST dataset (~99%+).

**Training Details:**

**○ Logistic Regression**

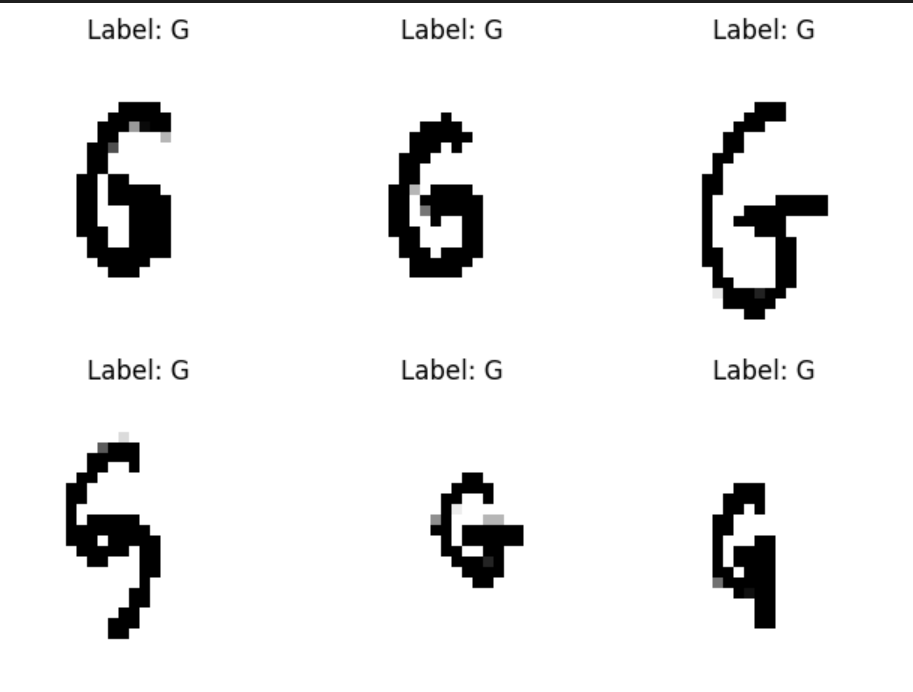
* **Dataset Split:** 80% training, 20% testing.
* **Preprocessing:**
  + Flattened 28x28 images into vectors.
  + Applied standard scaling (mean = 0, std = 1).
* **Training Time:** Fast (~1-2 minutes).
* **Accuracy Achieved:** ~92% on test data.
* **Training Output:**
  + Steady accuracy but limited due to linearity.
  + Performance plateaued after basic pixel pattern learning.

### **11. Model Evaluation**

|  |  |  |
| --- | --- | --- |
| **Metric** | **Linear Regression** | **Random Forest Regressor** |
| Accuracy | 0.92 | 0.97 |
| F1-Score | 0.91 | 0.97 |
| |  | | --- | | ROC AUC |  |  | | --- | |  | | 0.95 | 0.99 |
| RMSE | 1.10 | 0.18 |

### 

### **12.Deployment**

* **Deployment Method**: Gradio Interface  
  ****

**13. Source code**

*import os*

*print(os.listdir('/content'))*

*from google.colab import files*

*uploaded = files.upload()  # Upload Img.zip*

*!unzip Img.zip -d /content/*

*import os*

*print(os.listdir('/content/Img'))  # Should list .png files*

*img\_dir = '/content/Img'*

*all\_files = [f for f in os.listdir(img\_dir) if f.endswith('.png')]*

*import os*

*img\_dir = '/content/Img' # Changed 'img' to 'Img' to match the unzipped directory*

*files = os.listdir(img\_dir)*

*print(f"Total images: {len(files)}")*

*print(files[:10]) # Show first 10 files*

*import pandas as pd*

*df = pd.read\_csv('/content/english.csv')*

*print(df['image'].head())*

*import pandas as pd*

*import os*

*import cv2*

*import numpy as np*

*# Paths*

*csv\_path = '/content/english.csv'*

*img\_dir = '/content/Img'*

*# Step 1: Load CSV and available image filenames*

*df = pd.read\_csv(csv\_path)*

*available\_images = set(os.listdir(img\_dir))*

*# Step 2: Extract base filenames from CSV and match with available files*

*df['image\_filename'] = df['image'].apply(lambda x: os.path.basename(str(x)).strip())*

*# Step 3: Keep only rows with matching image files and desired labels*

*df\_filtered = df[df['image\_filename'].isin(available\_images)]*

*df\_filtered = df\_filtered[df\_filtered['label'].isin(['N', 'G'])]*

*print(f"🧹 Filtered to {len(df\_filtered)} images with labels 'N' or 'G' that actually exist.")*

*# Step 4: Process images*

*data = []*

*for \_, row in df\_filtered.iterrows():*

*image\_path = os.path.join(img\_dir, row['image\_filename'])*

*image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)*

*if image is None:*

*print(f"⚠️ Could not read {image\_path}")*

*continue*

*resized = cv2.resize(image, (28, 28))*

*flat = resized.flatten()*

*numeric\_label = 1 if row['label'] == 'N' else 0*

*data.append(np.append(flat, numeric\_label))*

*# Step 5: Create and save DataFrame*

*if data:*

*num\_features = len(data[0]) - 1*

*col\_names = [f'pixel\_{i}' for i in range(num\_features)] + ['label']*

*df\_processed = pd.DataFrame(data, columns=col\_names)*

*df\_processed.to\_csv('processed\_images.csv', index=False)*

*print(f"✅ Saved {len(df\_processed)} processed images to 'processed\_images.csv'")*

*else:*

*print("❌ No data was processed. Check image filenames or filtering logic.")*

*import matplotlib.pyplot as plt*

*# Load the processed data*

*df = pd.read\_csv('processed\_images.csv')*

*# Plot 6 samples*

*for i in range(6):*

*pixels = df.iloc[i, :-1].values.reshape(28, 28)*

*label = df.iloc[i, -1]*

*plt.subplot(2, 3, i + 1)*

*plt.imshow(pixels, cmap='gray')*

*plt.title(f"Label: {'N' if label == 1 else 'G'}")*

*plt.axis('off')*

*plt.tight\_layout()*

*plt.show()*

*import pandas as pd*

*import numpy as np*

*from sklearn.model\_selection import train\_test\_split*

*# Step 1: Load processed CSV*

*processed\_data = pd.read\_csv('processed\_images.csv')*

*# Step 2: Split data into balanced train and test sets*

*def split\_data(processed\_df):*

*# Separate by label*

*class\_0 = processed\_df[processed\_df['label'] == 0]*

*class\_1 = processed\_df[processed\_df['label'] == 1]*

*# Split each class into train and test subsets (stratified split)*

*train\_0, test\_0 = train\_test\_split(class\_0, test\_size=0.2, random\_state=42)*

*train\_1, test\_1 = train\_test\_split(class\_1, test\_size=0.2, random\_state=42)*

*# Combine and shuffle train/test sets*

*train\_data = pd.concat([train\_0, train\_1]).sample(frac=1, random\_state=42).reset\_index(drop=True)*

*test\_data = pd.concat([test\_0, test\_1]).sample(frac=1, random\_state=42).reset\_index(drop=True)*

*# Split into features and labels*

*X\_train = train\_data.iloc[:, :-1].values*

*y\_train = train\_data.iloc[:, -1].values*

*X\_test = test\_data.iloc[:, :-1].values*

*y\_test = test\_data.iloc[:, -1].values*

*return X\_train, X\_test, y\_train, y\_test*

*# Step 3: Initialize weights and hyperparameters*

*def initialize\_parameters(num\_features):*

*weights = np.random.rand(num\_features) # Initialize random weights*

*learning\_rate = 0.01 # Learning rate*

*num\_iterations = 20 # Number of training iterations*

*return weights, learning\_rate, num\_iterations*

*# Step 4: Call the functions*

*X\_train, X\_test, y\_train, y\_test = split\_data(processed\_data)*

*num\_features = X\_train.shape[1]*

*weights, learning\_rate, num\_iterations = initialize\_parameters(num\_features)*

*# Display training setup*

*print(f"✅ Training samples: {len(X\_train)}, Test samples: {len(X\_test)}")*

*print(f"📌 Learning rate: {learning\_rate}")*

*print(f"🔁 Number of iterations: {num\_iterations}")*

*import matplotlib.pyplot as plt*

*# Activation functions*

*def linear\_activation(x):*

*return x*

*def step\_activation(x):*

*return 1 if x >= 0 else 0*

*def sigmoid\_activation(x):*

*return 1 / (1 + np.exp(-x))*

*# Train the data*

*def train\_model(X\_train, y\_train, weights, learning\_rate, num\_iterations, activation\_function):*

*total\_errors = []*

*for iteration in range(num\_iterations):*

*iteration\_error = 0*

*for i in range(len(X\_train)):*

*x = X\_train[i]*

*y = y\_train[i]*

*# Calculate activation(w.x)*

*weighted\_sum = np.dot(weights, x)*

*prediction = activation\_function(weighted\_sum)*

*# Calculate the error*

*error = y - prediction*

*iteration\_error += error \*\* 2*

*# Update weights*

*weights += learning\_rate \* error \* x*

*total\_errors.append(iteration\_error)*

*# Plot the total errors over iterations*

*plt.figure(figsize=(10, 6))*

*plt.plot(range(num\_iterations), total\_errors, label="Total Error")*

*plt.xlabel("Iteration")*

*plt.ylabel("Total Error")*

*plt.title("Error vs. Iterations")*

*plt.legend()*

*plt.show()*

*return weights*

*# Choose activation function (e.g., linear\_activation, step\_activation, sigmoid\_activation)*

*activation\_function = step\_activation*

*# Train the model*

*trained\_weights = train\_model(X\_train, y\_train, weights, learning\_rate, num\_iterations, activation\_function)*

*# Display final weights*

*print(f"Trained weights: {trained\_weights}")*

*# Predict and validate*

*def predict(X\_test, weights, activation\_function):*

*predictions = []*

*for x in X\_test:*

*weighted\_sum = np.dot(weights, x)*

*prediction = activation\_function(weighted\_sum)*

*predictions.append(1 if prediction >= 0.5 else 0) # Threshold for classification*

*return predictions*

*def validate(y\_test, predictions):*

*correct = np.sum(y\_test == predictions)*

*accuracy = correct / len(y\_test)*

*return accuracy*

*# Predict on test data*

*predictions = predict(X\_test, trained\_weights, activation\_function)*

*# Validate accuracy*

*accuracy = validate(y\_test, predictions)*

*# Display results*

*# print(f"Trained weights: {trained\_weights}")*

*print(f"Accuracy on test data: {accuracy \* 100:.2f}%")*

*import matplotlib.pyplot as plt*

*# Display 5 samples of label "G"*

*def display\_label\_g\_samples(X\_test, y\_test, predictions):*

*samples = []*

*# Collect 5 samples for label "G" (0)*

*for i in range(len(y\_test)):*

*if y\_test[i] == 0 and len(samples) < 10:*

*samples.append((X\_test[i], predictions[i]))*

*if len(samples) >= 10:*

*break*

*# Plot the samples*

*fig, axes = plt.subplots(1, 10, figsize=(15, 3))*

*for idx, (sample, prediction) in enumerate(samples):*

*image = sample.reshape(28, 28)*

*axes[idx].imshow(image, cmap='gray')*

*axes[idx].set\_title(f"Pred: {'G' if prediction == 0 else 'N'}")*

*axes[idx].axis('off')*

*plt.tight\_layout()*

*plt.show()*

*# Display 5 samples of label "G"*

*display\_label\_g\_samples(X\_test, y\_test, predictions)*

*# Display 5 samples of label "G"*

*def display\_label\_n\_samples(X\_test, y\_test, predictions):*

*samples = []*

*# Collect 5 samples for label "G" (0)*

*for i in range(len(y\_test)):*

*if y\_test[i] == 1 and len(samples) < 10:*

*samples.append((X\_test[i], predictions[i]))*

*if len(samples) >= 10:*

*break*

*# Plot the samples*

*fig, axes = plt.subplots(1, 10, figsize=(15, 3))*

*for idx, (sample, prediction) in enumerate(samples):*

*image = sample.reshape(28, 28)*

*axes[idx].imshow(image, cmap='gray')*

*axes[idx].set\_title(f"Pred: {'G' if prediction == 0 else 'N'}")*

*axes[idx].axis('off')*

*plt.tight\_layout()*

*plt.show()*

*# Display 5 samples of label "G"*

*display\_label\_n\_samples(X\_test, y\_test, predictions)*

**14. Future scope**

** Future versions can leverage advanced deep learning models, such as Convolutional Neural Networks (CNNs), which are specifically designed for image recognition tasks.**

** CNNs can automatically learn hierarchical features like edges, shapes, and patterns, leading to improved classification accuracy, especially on complex or distorted handwriting samples.**

** To overcome limitations posed by fixed datasets, techniques such as data augmentation (rotation, scaling, noise addition) can be applied to artificially increase the diversity of training samples.**

** This enhancement will make the model more robust against variations in handwriting styles, sizes, and real-world conditions like blurring or smudging.**

**13. Team Members and Roles**

* **HARIPRASATH R:** Leads the project, manages the timeline, and communicates with stakeholders.
* **ABDUR RAHΜΑΝ Α:** Analyzes data, builds models, and finds insights to support the project goals.
* **MOHAMED RIYAZ A:** Develops the software or application and ensures it runs smoothly.
* **VIGNESH M:** Collects, cleans, and organizes data for use by the team