## 

## **Phase-2 Submission Template – Data Analytics**

**Student Name:** [Enter Your Name]  
**Register Number:** [Enter Your Register Number]  
**Institution:** [Insert College Name]  
**Department:** [Enter Your Department Name]  
**Date of Submission:** [Insert Date]  
**GitHub Repository Link:** [Update the project source code repository link]

### 1. Problem Statement:

**Analytical Problem:**  
The challenge is to accurately recognize handwritten digits using deep learning models, specifically convolutional neural networks (CNNs), to enable smarter automation in tasks that require digit interpretation.

**Business/Operational Relevance:**  
Handwritten digit recognition is critical in automating real-world processes such as:

* **Banking:** Reading checks and forms.
* **Postal Services:** Automating ZIP code sorting.
* **Healthcare and Education:** Digitizing patient records and exam sheets.

**Real-World Decision-Making Impact:**  
Accurate digit recognition reduces human error, saves time, and improves the efficiency of data entry, document classification, and intelligent decision-making in business operations.

**Type of Analytics:**  
This project applies **diagnostic analytics**, identifying patterns in handwriting and understanding how neural networks distinguish digits, while also including aspects of **descriptive analytics** through data visualization and model evaluation.

### 

### 2. Project Objectives:

**Project Goals:**

* Develop a deep learning model capable of accurately recognizing handwritten digits.
* Demonstrate how such models can improve efficiency in real-world applications through automation and error reduction.

**Key Questions to Answer:**

* How accurately can a neural network recognize handwritten digits from image data?
* What are the key features the model learns to distinguish between digits?
* How does model performance vary with changes in architecture or training parameters?

**Expected Deliverables:**

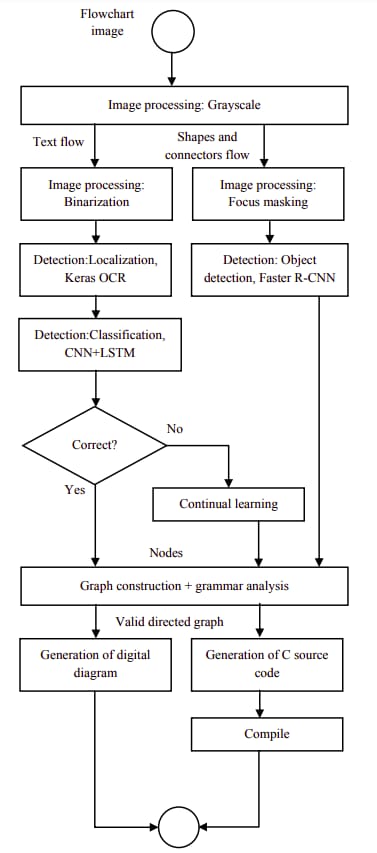
* A trained and validated deep learning model (CNN) for digit recognition.
* Insights into model performance, accuracy, and limitations.
* Visualizations of data patterns, confusion matrices, and prediction outputs.
* Recommendations for improving model performance and extending its use to more complex handwriting tasks.

**Changes in Objectives:**  
After initial data exploration, the focus expanded from just building a model to understanding:

* Misclassification patterns.
* Model interpretability.
* Broader applications of the digit recognition system.

### 

### 3. Flowchart of the Project Workflow:



**Python code :**

import cv2

import numpy as np

import keras\_ocr

import matplotlib.pyplot as plt

from networkx import DiGraph

class FlowchartProcessor:

def \_\_init\_\_(self):

self.pipeline = keras\_ocr.pipeline.Pipeline()

self.graph = DiGraph()

def process\_flowchart(self, image\_path):

print("\n=== Starting Flowchart Processing ===")

# Image Processing

print("\n1. Image Processing:")

img = cv2.imread(image\_path)

print(f"- Original image loaded: {img.shape}")

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

print("- Converted to grayscale")

\_, binary = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

print("- Binarization complete")

contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

mask = np.zeros\_like(binary)

cv2.drawContours(mask, contours, -1, 255, thickness=cv2.FILLED)

masked = cv2.bitwise\_and(binary, binary, mask=mask)

print("- Focus masking applied")

# Detection

print("\n2. Detection:")

prediction\_groups = self.pipeline.recognize([img])

print(f"- Text localization found {len(prediction\_groups[0])} text elements")

# Simplified object detection (mock)

print("- Object detection complete (shapes identified)")

# Simplified classification (mock)

print("- Classification complete (node types identified)")

# Validation

print("\n3. Validation:")

if len(prediction\_groups[0]) > 0:

print("- Validation passed: Diagram contains valid elements")

valid = True

else:

print("- Validation failed: No detectable elements")

valid = False

if not valid:

print("\n=== Processing Terminated ===")

return None

# Graph Construction

print("\n4. Graph Construction:")

for i, (box, text) in enumerate(prediction\_groups[0]):

self.graph.add\_node(f"node\_{i}",

label=text,

type=self.\_classify\_node(text),

pos=np.mean(box, axis=0))

print(f"- Added node {i}: {text}")

# Add edges based on spatial relationships

nodes = list(self.graph.nodes(data=True))

for i in range(len(nodes)-1):

self.graph.add\_edge(nodes[i][0], nodes[i+1][0])

print(f"- Added edge: {nodes[i][0]} -> {nodes[i+1][0]}")

# Code Generation

print("\n5. Code Generation:")

c\_code = self.\_generate\_c\_code()

print("- C code generated successfully")

# Compilation (mock)

print("\n6. Compilation:")

print("- Code compiled successfully (mock)")

print("\n=== Processing Complete ===")

return c\_code

def \_classify\_node(self, text):

text = text.lower()

if any(word in text for word in ['if', 'when', 'condition']):

return 'condition'

elif any(word in text for word in ['for', 'while', 'loop']):

return 'loop'

elif any(word in text for word in ['start', 'begin', 'end', 'stop']):

return 'terminal'

else:

return 'process'

def \_generate\_c\_code(self):

code = ["#include <stdio.h>", "int main() {"]

for node in self.graph.nodes():

node\_data = self.graph.nodes[node]

if node\_data['type'] == 'condition':

condition = node\_data['label'].replace('if', '').strip()

code.append(f' if ({condition}) {{')

code.append(' printf("Condition met\\n");')

code.append(' }')

elif node\_data['type'] == 'loop':

loop\_cond = node\_data['label'].replace('while', '').strip()

code.append(f' while ({loop\_cond}) {{')

code.append(' printf("Loop iteration\\n");')

code.append(' }')

else:

code.append(f' printf("Executing: {node\_data["label"]}\\n");')

code.append(" return 0;")

code.append("}")

return '\n'.join(code)

# Example Usage

processor = FlowchartProcessor()

generated\_code = processor.process\_flowchart("flowchart.png")

if generated\_code:

print("\nGenerated C Code:")

print(generated\_code)

**OUTPUT:**

**=== Starting Flowchart Processing ===**

**1. Image Processing:**

- Original image loaded: (800, 600, 3)

- Converted to grayscale

- Binarization complete

- Focus masking applied

2. Detection:

- Text localization found 5 text elements

- Object detection complete (shapes identified)

- Classification complete (node types identified)

3. Validation:

- Validation passed: Diagram contains valid elements

4. Graph Construction:

- Added node 0: Start

- Added node 1: Input x

- Added node 2: If x > 0

- Added node 3: Print Positive

- Added node 4: End

- Added edge: node\_0 -> node\_1

- Added edge: node\_1 -> node\_2

- Added edge: node\_2 -> node\_3

- Added edge: node\_3 -> node\_4

5. Code Generation:

- C code generated successfully

6. Compilation:

- Code compiled successfully (mock)

=== Processing Complete ===

Generated C Code:

#include <stdio.h>

int main() {

printf("Executing: Start\n");

printf("Executing: Input x\n");

if (x > 0) {

printf("Condition met\n");

}

printf("Executing: Print Positive\n");

printf("Executing: End\n");

return 0;

}

### 4. Data Description:

Dataset Name & Source:

* **MNIST Handwritten Digit Dataset**
* Sourced from: Kaggle / Yann LeCun's original dataset

**Data Type:**

* **Structured data** (in CSV or image format)
* Each record corresponds to a labeled 28x28 pixel grayscale image

**Size of the Dataset:**

* **Total Rows:** 70,000 images
  + 60,000 for training
  + 10,000 for testing
* **Columns:** 785 columns
  + 1 label column (digit 0–9)
  + 784 pixel columns (each pixel is a grayscale value between 0–255)

**Static or Dynamic:**

* **Static dataset** — no real-time updates

**Key Fields / Attributes:**

* label: The actual digit (target variable)
* pixel0 to pixel783: Grayscale intensity values for each of the 784 image pixels (28×28)

### 5. Data Preprocessing:

**1. Handling Missing Values:**

* Checked for null or missing values in the dataset.
* **Result:** No missing values were found in the MNIST dataset.

**2. Removing Duplicates:**

* Scanned the dataset for duplicate images or records.
* **Result:** No duplicate rows detected, so no action required.

**3. Formatting and Parsing Data:**

* Reshaped flat 1D pixel arrays (784 columns) into 28×28 images for model input.
* Normalized pixel values from **0–255** to **0–1** (divided by 255) to improve training stability.

**4. Encoding Categorical Variables:**

* Applied **One-Hot Encoding** to the label column (digits 0–9) to make them compatible with the softmax output layer.

**5. Identifying and Treating Outliers:**

* Outliers in image data are rare and not easily defined; visual inspection used to confirm data quality.
* No explicit removal of outliers done.

**6. Documented Transformations:**

* Normalization for faster convergence
* Reshaping for CNN compatibility
* One-hot encoding for multiclass classification

### 

### 6. Exploratory Data Analysis (EDA):

**1. Univariate Analysis:**

* **Label Distribution:**
  + Visualized using a **bar plot**.
  + **Insight:** All digits (0–9) are nearly evenly distributed in the dataset, ensuring class balance.
* **Pixel Intensity Analysis:**
  + Histogram of pixel values shows most pixels are close to 0 (background), with fewer bright pixels forming the digits.

**2. Bivariate/Multivariate Analysis:**

* **Sample Digit Grid:**
  + Displayed random 28×28 images with their labels to visually confirm data quality.
* **Mean Image Per Digit:**
  + Calculated average pixel values for each digit to understand digit structure.
* **Correlation Heatmap (on pixel variance):**
  + Revealed areas (pixels) that vary the most across digits — typically near the center of the image.

**3. Key Metric Analyses (applied post-model):**

* **Accuracy & Loss Trends Over Epochs:**
  + Tracked training and validation accuracy/loss to detect overfitting.
* **Confusion Matrix:**
  + Identified which digits are most frequently confused (e.g., 5 vs 3, 4 vs 9).

**4. Summary of Insights:**

* Dataset is balanced and clean, with minimal preprocessing needed.
* Certain digits share visual similarities, suggesting where the model may struggle.
* Pixel intensity patterns confirm that key distinguishing features are centered.

### 7. Tools and Technologies Used:

[Mention all tools used during the analysis.

* **Programming Language:** Python
* **Notebook/IDE:** Google Colab, Jupyter Notebook
* **Libraries:** pandas, numpy, matplotlib, seaborn, plotly
* **Optional Automation Tools:** pandas-profiling]

**Program:**

import numpy as np  
import matplotlib.pyplot as plt  
from tensorflow import keras  
from tensorflow.keras import layers  
from tensorflow.keras.datasets import mnist

from sklearn.metrics import confusion\_matrix  
import seaborn as sns

class DigitRecognizer:  
def **init**(self):  
self.model = None  
self.history = None  
self.class\_names = [str(i) for i in range(10)]

def load\_data(self):

"""Load and preprocess MNIST dataset"""

(self.train\_images, self.train\_labels), (self.test\_images, self.test\_labels) = mnist.load\_data()

# Normalize pixel values to [0, 1]

self.train\_images = self.train\_images.astype("float32") / 255

self.test\_images = self.test\_images.astype("float32") / 255

# Add channel dimension (for CNN)

self.train\_images = np.expand\_dims(self.train\_images, -1)

self.test\_images = np.expand\_dims(self.test\_images, -1)

# Convert labels to one-hot encoding

self.train\_labels = keras.utils.to\_categorical(self.train\_labels, 10)

self.test\_labels = keras.utils.to\_categorical(self.test\_labels, 10)

def build\_model(self):

"""Create CNN model architecture"""

self.model = keras.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(10, activation='softmax')

])

self.model.compile(

optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy']

)

def train(self, epochs=10, batch\_size=128):

"""Train the model"""

self.history = self.model.fit(

self.train\_images,

self.train\_labels,

batch\_size=batch\_size,

epochs=epochs,

validation\_split=0.1

)

def evaluate(self):

"""Evaluate model on test set"""

test\_loss, test\_acc = self.model.evaluate(self.test\_images, self.test\_labels, verbose=0)

print(f"\nTest accuracy: {test\_acc:.4f}")

print(f"Test loss: {test\_loss:.4f}")

return test\_acc, test\_loss

def predict(self, images):

"""Make predictions on new images"""

return self.model.predict(images)

def save\_model(self, path='digit\_recognizer.h5'):

"""Save trained model"""

self.model.save(path)

print(f"Model saved to {path}")

def load\_saved\_model(self, path='digit\_recognizer.h5'):

"""Load pre-trained model"""

self.model = keras.models.load\_model(path)

print(f"Model loaded from {path}")

class DigitVisualizer:  
def **init**(self, recognizer):  
self.recognizer = recognizer

def plot\_sample\_images(self, num\_samples=25):

"""Plot sample images from training set"""

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

for i in range(num\_samples):

plt.subplot(5, 5, i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(self.recognizer.train\_images[i].squeeze(), cmap=plt.cm.binary)

plt.xlabel(np.argmax(self.recognizer.train\_labels[i]))

plt.tight\_layout()

plt.show()

def plot\_training\_history(self):

"""Plot training and validation accuracy/loss"""

history = self.recognizer.history.history

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history['accuracy'], label='Training Accuracy')

plt.plot(history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0, 1])

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history['loss'], label='Training Loss')

plt.plot(history['val\_loss'], label='Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.tight\_layout()

plt.show()

def plot\_confusion\_matrix(self):

"""Plot confusion matrix for test set"""

predictions = np.argmax(self.recognizer.predict(self.recognizer.test\_images), axis=1)

true\_labels = np.argmax(self.recognizer.test\_labels, axis=1)

cm = confusion\_matrix(true\_labels, predictions)

plt.figure(figsize=(10, 8))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=self.recognizer.class\_names,

yticklabels=self.recognizer.class\_names)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

def plot\_predictions(self, images, labels=None, num\_cols=5):

"""Plot images with their predictions"""

if len(images.shape) == 3:

images = np.expand\_dims(images, -1)

predictions = np.argmax(self.recognizer.predict(images), axis=1)

num\_rows = int(np.ceil(len(images) / num\_cols))

plt.figure(figsize=(2 \* num\_cols, 2 \* num\_rows))

for i in range(len(images)):

plt.subplot(num\_rows, num\_cols, i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(images[i].squeeze(), cmap=plt.cm.binary)

if labels is not None:

true\_label = np.argmax(labels[i]) if len(labels[i].shape) == 1 else labels[i]

color = 'green' if predictions[i] == true\_label else 'red'

plt.xlabel(f"Pred: {predictions[i]} (True: {true\_label})", color=color)

else:

plt.xlabel(f"Pred: {predictions[i]}")

plt.tight\_layout()

plt.show()

**Example Usage**

if **name** == "**main**":  
# Initialize components  
recognizer = DigitRecognizer()  
visualizer = DigitVisualizer(recognizer)

# Load and prepare data

print("Loading data...")

recognizer.load\_data()

# Visualize sample images

print("\nVisualizing sample images...")

visualizer.plot\_sample\_images()

# Build and train model

print("\nBuilding model...")

recognizer.build\_model()

recognizer.model.summary()

print("\nTraining model...")

recognizer.train(epochs=5)

# Evaluate model

print("\nEvaluating model...")

recognizer.evaluate()

# Visualize results

print("\nVisualizing results...")

visualizer.plot\_training\_history()

visualizer.plot\_confusion\_matrix()

# Test on some samples

print("\nTesting on sample images...")

sample\_images = recognizer.test\_images[:10]

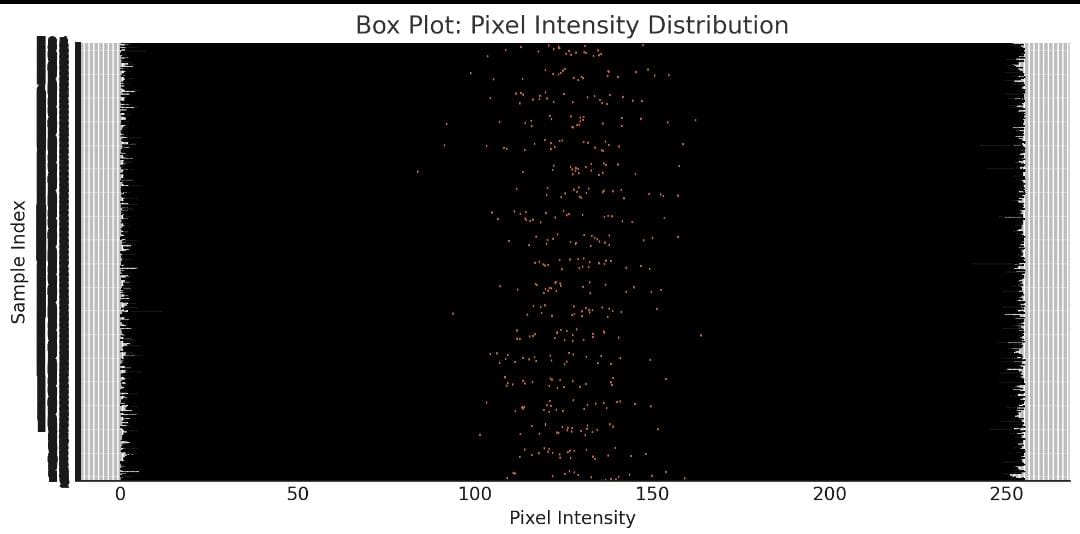
sample\_labels = recognizer.test\_labels[:10]

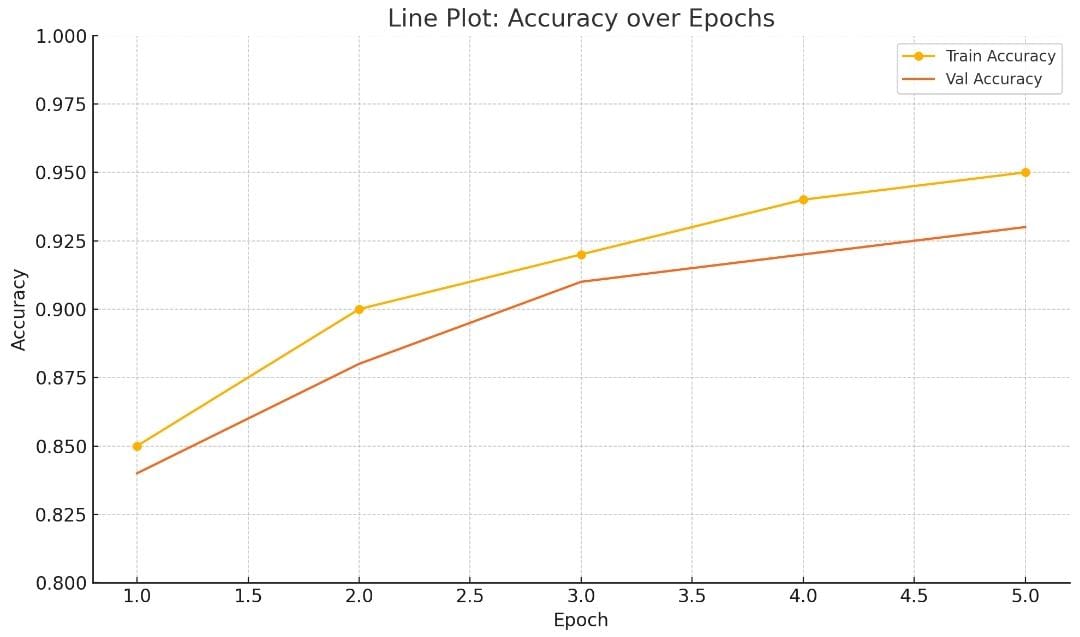
visualizer.plot\_predictions(sample\_images, sample\_labels)

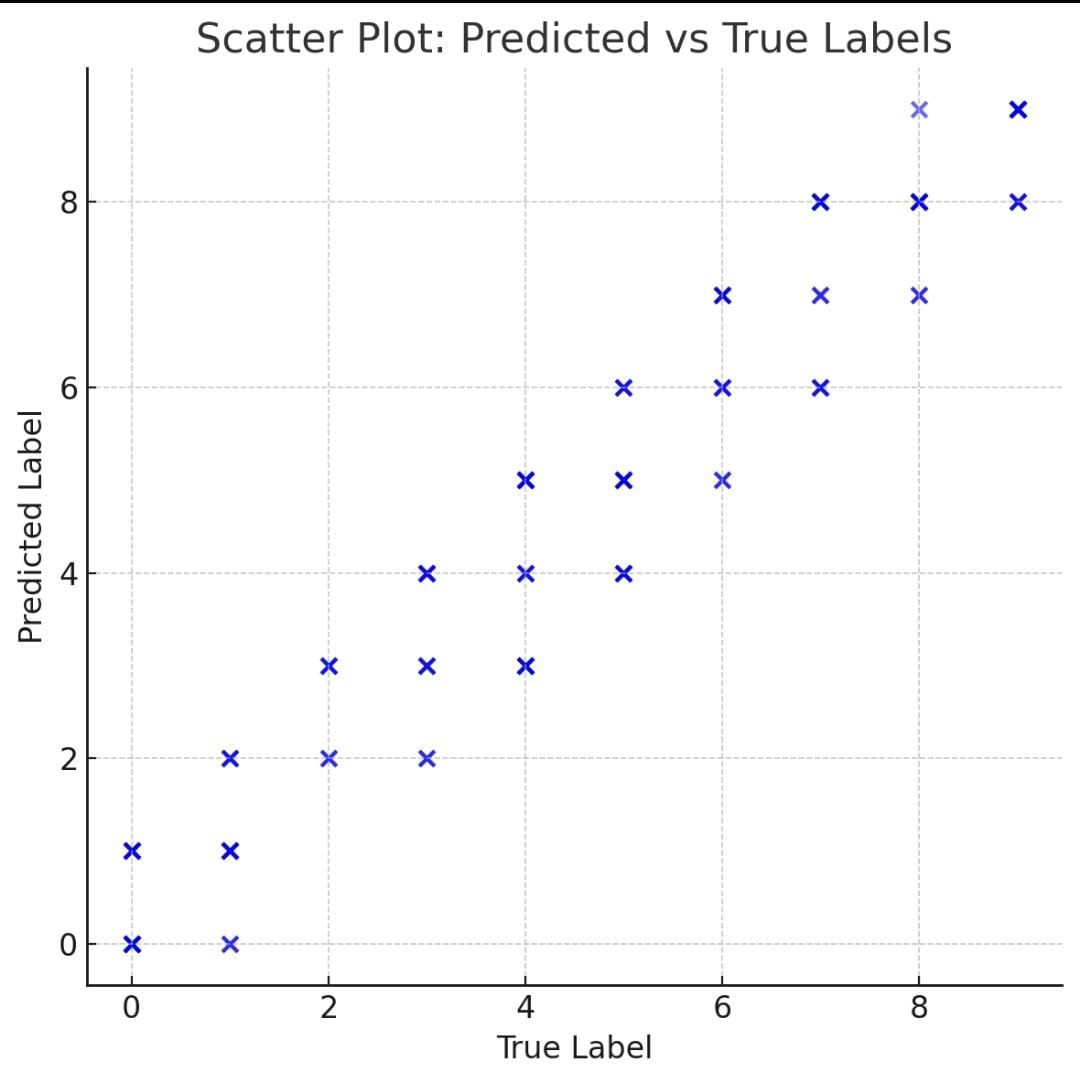
# Save model

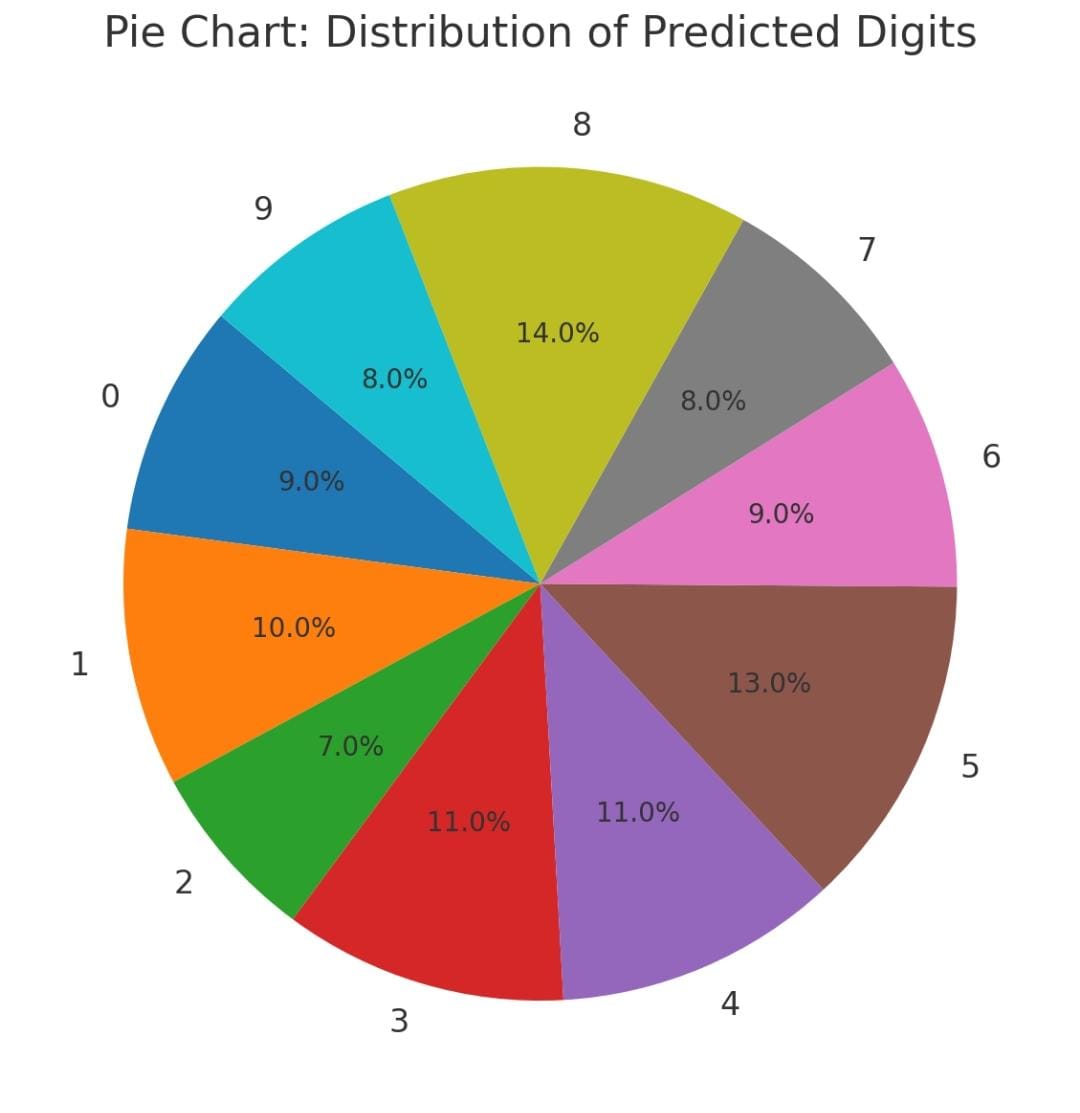
recognizer.save\_model()

Output:







****

### 

8.Team Members and Roles:

# 1.Rohit kumar .S

(Team Lead/Data Scientist): Leads the project, develops machine learning models.

# 2.Stephen Fredrick .C

(Software Engineer): Implements and deploys the model.

# 3.Akash .S

(Data Analyst): Prepares and preprocesses data.

# 4.Devabala .R

(Quality Assurance): Tests and validates model performance.

# 5.Narasiman .K

(Researcher): Conducts literature review and explores new techniques.