**Handwritten Digit Recognition Project**

**Overview**

Handwritten digit recognition is a classic problem in the field of computer vision and machine learning. The goal is to automatically recognize and classify digits (0 through 9) from images of handwritten digits. This problem is typically approached using image classification techniques.

**Dataset**

**MNIST Dataset**: The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used benchmark dataset for handwritten digit recognition. It contains:

* 60,000 training images and 10,000 testing images.
* Each image is a grayscale 28x28 pixel image of a single digit.

**Techniques**

1. **Convolutional Neural Networks (CNNs)**: CNNs are particularly effective for image classification tasks. They use convolutional layers to automatically learn spatial hierarchies of features from input images.
2. **Data Preprocessing**: Includes resizing, normalization, and sometimes augmentation of images to prepare them for model input.
3. **One-Hot Encoding**: Converts categorical labels into a binary matrix, used as the target for classification tasks.
4. **Batch Normalization**: Normalizes the outputs of layers to stabilize and accelerate training.
5. **Activation Functions**: Functions like ReLU (Rectified Linear Unit) and Softmax are used in CNNs to introduce non-linearity and produce probabilities for classification.

**Python Code**

The provided code demonstrates a complete pipeline for handwritten digit recognition, including model training, prediction, and a GUI for user interaction.

**1. Import Libraries**

python

Copy code

import numpy as np

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Input, BatchNormalization

from keras.utils import to\_categorical

from tensorflow.keras.optimizers import Adam

from PIL import Image, ImageDraw

import tkinter as tk

from tkinter import Button, Canvas, messagebox

**2. Load and Prepare Dataset**

python

Copy code

def load\_large\_dataset():

(X\_train, y\_train), (\_, \_) = mnist.load\_data()

return X\_train, y\_train

X\_train, y\_train = load\_large\_dataset()

y\_train = to\_categorical(y\_train, 10)

X\_train = X\_train.reshape(X\_train.shape[0], 28, 28, 1).astype('float32')

X\_train /= 255

* **Function**: Loads the MNIST dataset and preprocesses it.
* **Normalization**: Scales pixel values to the range [0, 1].

**3. Create and Train the CNN Model**

python

Copy code

def create\_model():

model = Sequential()

model.add(Input(shape=(28, 28, 1)))

model.add(Conv2D(64, (3, 3), activation='relu', kernel\_initializer='he\_uniform'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu', kernel\_initializer='he\_uniform'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2)))

model.add(Flatten())

model.add(Dense(256, activation='relu', kernel\_initializer='he\_uniform'))

model.add(Dense(10, activation='softmax'))

return model

model = create\_model()

model.compile(loss='categorical\_crossentropy', optimizer=Adam(learning\_rate=0.001), metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=25, batch\_size=128)

* **Model Architecture**: Consists of convolutional layers, max-pooling layers, batch normalization, and dense layers.
* **Training**: The model is trained for 25 epochs with a batch size of 128.

**4. Image Preprocessing and Prediction**

python

Copy code

def preprocess\_image(image):

image = image.convert('L')

image = image.resize((28, 28), Image.Resampling.LANCZOS)

image\_array = np.array(image)

image\_array = 255 - image\_array

image\_array = image\_array.astype('float32') / 255.0

image\_array = image\_array.reshape(1, 28, 28, 1)

return image\_array

def predict(image):

preprocessed\_image = preprocess\_image(image)

pred = model.predict(preprocessed\_image, batch\_size=1)

return pred.argmax()

* **Preprocessing**: Converts the image to grayscale, resizes it, inverts colors, normalizes, and reshapes it for the model.
* **Prediction**: Uses the model to predict the digit from the preprocessed image.

**5. Tkinter GUI Application**

python

Copy code

class DrawingApp:

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

self.root = root

self.root.title("Digit Drawing and Prediction")

self.canvas = Canvas(root, width=280, height=280, bg='white')

self.canvas.pack()

self.button\_clear = Button(root, text="Clear", command=self.clear)

self.button\_clear.pack(side='left')

self.button\_predict = Button(root, text="Predict", command=self.predict)

self.button\_predict.pack(side='left')

self.drawing = False

self.last\_x, self.last\_y = 0, 0

self.canvas.bind("<Button-1>", self.start\_drawing)

self.canvas.bind("<B1-Motion>", self.draw)

self.image = Image.new("L", (280, 280), color=255)

self.draw\_image = ImageDraw.Draw(self.image)

def start\_drawing(self, event):

self.drawing = True

self.last\_x, self.last\_y = event.x, event.y

def draw(self, event):

if self.drawing:

x, y = event.x, event.y

self.canvas.create\_line((self.last\_x, self.last\_y, x, y), fill='black', width=8)

self.draw\_image.line([self.last\_x, self.last\_y, x, y], fill=0, width=8)

self.last\_x, self.last\_y = x, y

def clear(self):

self.canvas.delete("all")

self.image = Image.new("L", (280, 280), color=255)

self.draw\_image = ImageDraw.Draw(self.image)

def predict(self):

prediction = predict(self.image)

messagebox.showinfo("Prediction", f"The model predicts: {prediction}")

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = DrawingApp(root)

root.mainloop()

* **GUI Application**: Provides an interface to draw digits, clear the canvas, and predict the digit using the trained model.
* **Drawing**: Allows users to draw digits with the mouse.
* **Prediction**: Displays the predicted digit in a message box.

**Summary**

This documentation covers the entire process of creating a handwritten digit recognition system using a CNN, from loading and preparing the data to building and training the model, preprocessing input images, and creating a user-friendly GUI application. This approach showcases a practical application of machine learning in image classification and provides an interactive tool for digit recognition.