Python Digit Recognizer GUI

ENGR 13300 Independent Project Report

Medha Shinde

LC5 – 11

Table of Contents

1. Introduction

For my ENGR 13300 Independent Project, I decided to create an image classifier for recognizing handwritten digits. It connects with my intended major of Computer Engineering with a concentration in Artificial Intelligence and Machine Learning, as the ability to develop image recognition systems has significant relevance in various AI applications, including computer vision, automated data entry, and more.

I relied on the Modified National Institute of Standards and Technology (MNIST) database to train a robust digit recognition model. The dataset consists of handwritten digits and is commonly used for training and testing image recognition models. For this reason, I felt it was a suitable dataset for training a digit recognition model using TensorFlow, a deep learning framework.

Once I trained the dataset, I set up a graphical user interface (GUI) using primarily Tkinter. Tkinter is the standard Python library used for creating GUIs, so it is fairly straightforward to implement. I set up the program to load the model trained from the MNIST dataset, prompt the user to draw a digit, and use the model to identify the digit. The model has an over 98% accuracy, which is adequate for identifying the input the majority of the time.

This report contains the user manual for how to use the program, such as how sample inputs and outputs may look like. It also explains how the program is able to identify the digit and which user-defined functions it implements, as well as the error-handling mechanisms. In the very end the source code is included, so users can run the same program on their own personal computers provided they have the necessary software/libraries installed.

2. User Manual

2.1 Setting Up

The program and interface are easy to begin using. There are code files: **'train_model.py'** and **'digit_recognizer.py.'** If the user has not run 'train_model.py' before, they will need to do so. This is because this code performs the essential task of training the convolutional neural network (CNN) on the MNIST dataset using Tensorflow and Keras. Without this pre-training, the digit recognizer code would be attempting to load a non-existent model, resulting in an error.

To do so, open the folder in which the two files are contained in terminal, and run the command **'python train_model.py.'** The resulting output should look like:

```
Epoch #0
1500/1500 [==============================] - 30s 17ms/step - loss: 0.1533 -
accuracy: 0.9534 - val_loss: 0.0578 - val_accuracy: 0.9827
Epoch #1
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0476 -
accuracy: 0.9849 - val_loss: 0.0496 - val_accuracy: 0.9853
Epoch #2
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0313 -
accuracy: 0.9900 - val_loss: 0.0538 - val_accuracy: 0.9838
Epoch #3
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0245 -
accuracy: 0.9921 - val_loss: 0.0416 - val_accuracy: 0.9886
Epoch #4
1500/1500 [==============================] - 37s 25ms/step - loss: 0.0180 -
accuracy: 0.9940 - val_loss: 0.0384 - val_accuracy: 0.9891
Epoch #5
1500/1500 [==============================] - 30s 20ms/step - loss: 0.0123 -
accuracy: 0.9959 - val_loss: 0.0424 - val_accuracy: 0.9887
Epoch #6
1500/1500 [==============================] - 33s 22ms/step - loss: 0.0119 -
accuracy: 0.9962 - val_loss: 0.0511 - val_accuracy: 0.9873
Epoch #7
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0093 -
accuracy: 0.9970 - val_loss: 0.0482 - val_accuracy: 0.9900
Epoch #8
1500/1500 [==============================] - 19s 13ms/step - loss: 0.0078 -
accuracy: 0.9973 - val_loss: 0.0492 - val_accuracy: 0.9898
Epoch #9
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0053 -
accuracy: 0.9984 - val_loss: 0.0572 - val_accuracy: 0.9884
313/313 [==============================] - 3s 7ms/step - loss: 0.0549 -
accuracy: 0.9884
```

The results indicate that the model has an overall accuracy of about 98.84%.

2.2 How it Works

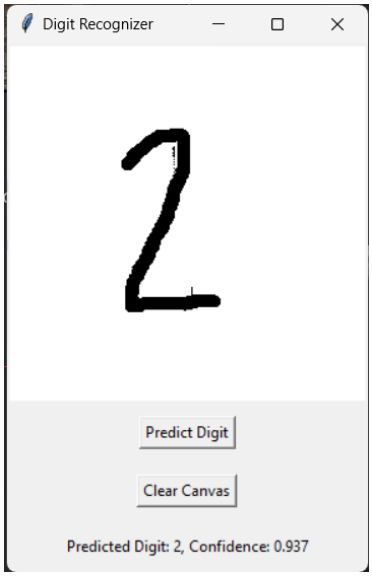
Once the model has been trained, the digit recognizer program can be run using the command `'python digit_recognizer.py.'` The Tkinter GUI window will open, allowing the user to draw a digit of their choice.

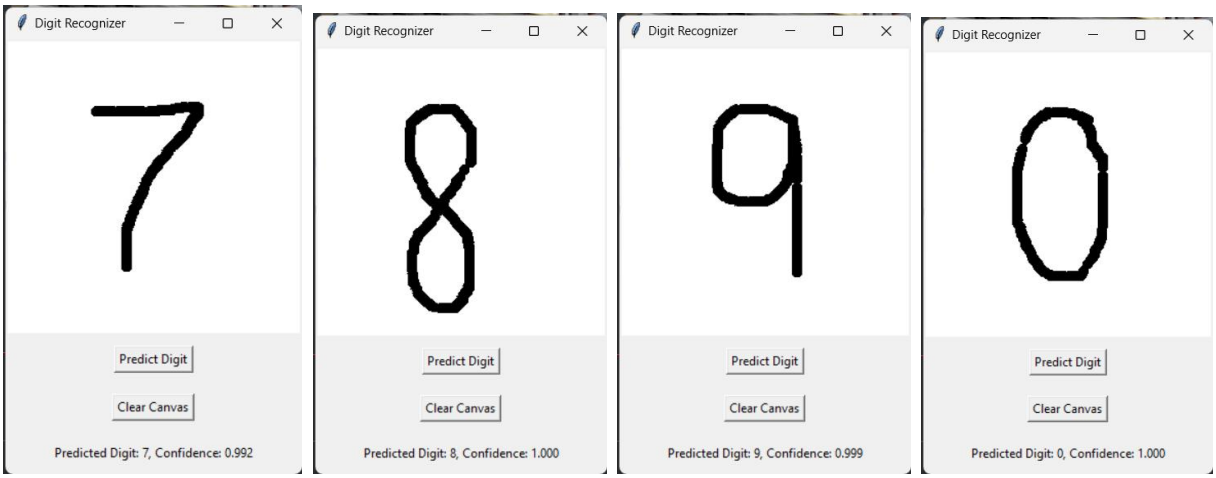
In terms of the specific user-defined functions used in this code, when the user begins the program:

A. The `main()` function orchestrates the Tkinter GUI setup, canvas creation, and event bindings, as well as establishing the Predict Digit and Clear Canvas buttons.
B. The `draw_on_canvas(event)` UDF enables real-time digit drawing by responding to the user input of drawing. It allows for drawn strokes to be dynamically updated on canvas.
C. The `start_drawing(event)` and `stop_drawing(event)` UDFs manage the drawing state, initiating drawing when the left mouse button is pressed upon and stopping when it is released.
D. The `predict_digit()` function is activated by the 'Predict Digit' button and is essential for retrieving the drawn image, preprocessing it, and employing the trained model to make predictions. It runs multiple predictions to maximize accuracy, and the highest confidence prediction is displayed on the Tkinter window.
E. The `clear_canvas()` function is activated by the 'Clear Canvas' button and wipes the canvas clean, clearing any previous prediction results.

2.3 Sample Inputs and Outputs

These are what sample inputs and outputs may look like. The program displays the predicted digit, as well as the confidence level of the program's prediction. The minimum confidence threshold for a prediction is set to 0.85, so that the program does not make wildly inaccurate predictions, such as if the input is invalid.





2.4 Error Handling

Apart from the minimum confidence threshold coded into the program, there are a couple of error-handling mechanisms. For example, if the user does not draw anything, or if the user scribbles something that is not a digit, the program will display an error message.

3. Project Code

3.1 Model Training Code

```
"""
================================================================================
ENGR 13300 Fall 2023

Program Description
    This program trains a convolutional neural network to accurately identify
    handwritten digits (0-9) using the MNIST dataset.

Assignment Information
    Assignment:     ENGR 13300 Independent Project
    Author:         Medha Shinde, shindem@purdue.edu
    Team ID:        LC5 - 11

Contributor:    None
    My contributor(s) helped me:
    [ ] understand the assignment expectations without
        telling me how they will approach it.
    [ ] understand different ways to think about a solution
        without helping me plan my solution.
    [ ] think through the meaning of a specific error or
        bug present in my code without looking at my code.
    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.

ACADEMIC INTEGRITY STATEMENT
I have not used source code obtained from any other unauthorized
source, either modified or unmodified. Neither have I provided
access to my code to another. The project I am submitting
is my own original work.
================================================================================
"""

# Import necessary libraries

import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
# Load MNIST data
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

model = models.Sequential()

# Convolutional layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Flattening step
model.add(layers.Flatten())

# Fully connected layers
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

callbacks = [
    EarlyStopping(patience=3, monitor='val_loss'),
    ModelCheckpoint(filepath='best_model.h5', save_best_only=True)
]

# Define the number of epochs
num_epochs = 10

# Training loop
for epoch in range(num_epochs):
    # Code to perform training for one epoch
    print(f"Epoch #{epoch}")
    history = model.fit(train_images[..., tf.newaxis], train_labels, epochs=1,
validation_split=0.2, callbacks=callbacks)

test_loss, test_acc = model.evaluate(test_images[..., tf.newaxis], test_labels)

# Save the best model

model.save('best_model.h5')
```

3.2 Program Code

```
"""
================================================================================
ENGR 13300 Fall 2023

Program Description
    This program loads the optimized model to recognize user-drawn digits via a
    Tkinter GUI canvas that handles real-time prediction and display.

Assignment Information
    Assignment:     ENGR 13300 Independent Project
    Author:         Medha Shinde, shindem@purdue.edu
    Team ID:        LC5 - 11

Contributor:    None

    My contributor(s) helped me:
    [ ] understand the assignment expectations without
        telling me how they will approach it.
    [ ] understand different ways to think about a solution
        without helping me plan my solution.
    [ ] think through the meaning of a specific error or
        bug present in my code without looking at my code.
    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.

ACADEMIC INTEGRITY STATEMENT
I have not used source code obtained from any other unauthorized
source, either modified or unmodified. Neither have I provided
access to my code to another. The project I am submitting
is my own original work.
================================================================================
"""

# Import necessary libraries

import tensorflow as tf
from tensorflow.keras.models import load_model
from tkinter import Tk, Canvas, Button, Label
from PIL import Image, ImageDraw
import numpy as np

# Load the trained model
try:
    model = load_model('best_model.h5')
    model_loaded = True
except Exception as e:
    print(f"Error loading the model: {e}")
    model_loaded = False

# Global variables for canvas and image
canvas = None
draw = None
confidence_threshold = 0.85
```

```
def predict_digit():
    global canvas

    if not model_loaded:
        result_label.config(text="Error: Model not loaded!")
        return

    try:
        # Get the image from the canvas
        filename = "image.png"
        canvas.postscript(file=filename, colormode='color')
        img = Image.open(filename)

        # Convert the image to grayscale
        img = img.convert('L')

        # Resize the image to 28x28 pixels
        img = img.resize((28, 28))

        # Invert the colors (black background, white drawing)
        img = Image.eval(img, lambda x: 255 - x)

        # Save the image for debugging (optional)
        img.save("drawn_image.png")

        # Convert the image to a numpy array
        img_array = np.array(img)

        # Normalize pixel values
        img_array = img_array / 255.0

        # Reshape the array to match the model's expected input shape
        img_array = img_array.reshape(1, 28, 28, 1)

        if np.max(img_array) == 0:
            # Handle the case where no drawing is detected
            result_label.config(text="No drawing detected!")
        else:
            # Make multiple predictions
            num_predictions = 10
            predictions = model.predict(img_array)

            # Check each prediction
            valid_predictions = []
            for _ in range(num_predictions):
                predictions = model.predict(img_array)
                predicted_class = np.argmax(predictions)
                confidence = predictions[0][predicted_class]

                if confidence >= confidence_threshold:
                    valid_predictions.append((predicted_class, confidence))

            if not valid_predictions:
```

```
                # No valid predictions found
                result_label.config(text="Not recognized as a digit!")
            else:
                # Display the highest confidence prediction
                best_prediction = max(valid_predictions, key=lambda x: x[1])
                predicted_class, confidence = best_prediction
                result_label.config(text=f"Predicted  Digit:  {predicted_class},
Confidence: {confidence:.3f}")

    except Exception as e:
        # Handle any errors that may occur during the prediction process
        result_label.config(text=f"Error predicting digit: {e}")

def draw_on_canvas(event):
    global canvas, draw

    if draw:
        x1, y1 = (event.x - 1), (event.y - 1)
        x2, y2 = (event.x + 1), (event.y + 1)
        canvas.create_oval(x1, y1, x2, y2, fill="black", width=8)

def start_drawing(event):
    global draw
    draw = True

def stop_drawing(event):
    global draw
    draw = False

def clear_canvas():
    global canvas, draw

    canvas.delete("all")
    result_label.config(text="Predicted Digit: ")

    # Reset the drawing
    draw = None

def main():
    # Create the main window
    global root
    root = Tk()
    root.title("Digit Recognizer")

    # Create a drawing canvas
    global canvas
    canvas = Canvas(root, width=280, height=280, bg="white")
    canvas.pack()

    # Bind mouse events to the canvas
    canvas.bind("<B1-Motion>", draw_on_canvas)
    canvas.bind("<ButtonRelease-1>", stop_drawing)
    canvas.bind("<Button-1>", start_drawing)

    # Create buttons
```

```
    predict_button = Button(root, text="Predict Digit", command=predict_digit)
    predict_button.pack(pady=10)

    clear_button = Button(root, text="Clear Canvas", command=clear_canvas)
    clear_button.pack(pady=10)

    # Display the predicted digit
    global result_label
    result_label = Label(root, text="Predicted Digit: ")
    result_label.pack(pady=10)

    # Run the Tkinter main loop
    root.mainloop()

if __name__ == "__main__":
    main()
```