Handwritten Digit Recognition App

DONE BY

DEEPU PRASAD

ABHINAND A S

THEJUIN P

JITHIN V M

Handwritten Digit Recognition App

1. Introduction

This Streamlit web application allows users to draw a digit (0–9) on a canvas and uses a Convolutional Neural Network (CNN) to recognize the digit. The model is trained on the MNIST dataset and saved for reuse.

2. Requirements

The following Python libraries are required:

- - streamlit
  - numpy
  - tensorflow
  - opencv-python
  - Pillow
  - streamlit-drawable-canvas

3. Model Setup

The application attempts to load a pre-trained CNN model from the local filesystem. If the model is not found, it will train a new model using the MNIST dataset, then save it as 'digit_model.keras'.

4. Model Architecture

The CNN model consists of:

- - Conv2D layer with 32 filters
  - MaxPooling2D
  - Conv2D with 64 filters
  - MaxPooling2D

- Flatten layer
- Dense layer with 64 neurons
- Output Dense layer with 10 neurons (softmax)

5. Canvas Interaction

The app includes a drawable canvas using `streamlit-drawable-canvas`. Users draw a digit, which is then processed and passed to the model.

6. Image Preprocessing

Steps include:

- - Convert RGBA to grayscale using OpenCV
  - Resize the image to 28x28 pixels
  - Invert the colors (black background, white digit)
  - Center the digit and pad to maintain aspect ratio
  - Normalize pixel values to [0, 1] range
  - Reshape to fit model input format (1, 28, 28, 1)

7. Prediction

The processed image is passed to the model for prediction. The app displays the digit with the highest confidence score.

8. Output

After prediction, the app shows the user's drawing and the predicted digit using styled HTML for better presentation.

9. Step-by-Step Code Explanation

1. Import Libraries

Essential libraries are imported for building the app, processing images, and using the ML model.

```
Users > nsti- > Downloads > m.py > ...
import streamlit as st
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist
from PIL import ImageOps, Image
import cv2
```

2. App Title and Instructions

Streamlit displays a title and a short description of how to use the
app.

```
# Title
st.title("✏️ Handwritten Digit Recognition")
st.write("Draw a digit (0-9) and click **Predict**")
```

3. Load or Retrain the Model

Tries to load a pre-trained Keras model from disk. If not found, it retrains using the MNIST
dataset

```
# Try to load model or train if failed
try:
    model = tf.keras.models.load_model(MODEL_PATH)
except Exception as e:
    st.warning("⚠️ Model not found or corrupted. Retraining the model...")
```

4. Model Training

If retraining is needed, it reshapes and normalizes the data, sets up a CNN model, compiles, fits, and saves it.

```
# Load data
(X_train, y_train), (_, _) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255
y_train = tf.keras.utils.to_categorical(y_train, 10)

# Set seed
tf.random.set_seed(42)
np.random.seed(42)

# Define model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    (import) Conv2D: Any
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=3, batch_size=64, validation_split=0.2, verbose=0)
model.save(MODEL_PATH)
st.success("✅ Model trained and saved successfully!")
```

5. Drawing Canvas

Uses `streamlit-drawable-canvas` to let users draw digits interactively.

```
# Drawing canvas
from streamlit_drawable_canvas import st_canvas

canvas_result = st_canvas(
    fill_color="white",
    stroke_width=10,
    stroke_color="black",
    background_color="white",
    height=280,
    width=280,
    drawing_mode="freedraw",
    key="canvas"
)
```

6. Predict Button

Once the user clicks Predict, the app checks if there's a drawing to process.

```

# Predict on drawing
if canvas_result.image_data is not None:
    img = canvas_result.image_data
```

7. Image Preprocessing

Processes the drawn image: converts to grayscale, resizes, inverts, and normalizes it.

```
    if st.button("Predict"):
        # Preprocess image
        img = cv2.cvtColor(img.astype(np.uint8), cv2.COLOR_RGBA2GRAY)
        img = cv2.resize(img, (28, 28))
        img = ImageOps.invert(Image.fromarray(img))
        img = np.array(img).astype("float32") / 255.0
```

8. Digit Centering and Padding

Attempts to extract the digit's bounding box, resizes it to 20x20, and pads it to 28x28.

```
    # Optional: Center and pad digit
    try:
        inverted_img = cv2.bitwise_not((img * 255).astype(np.uint8))
        coords = cv2.findNonZero(inverted_img)
        if coords is not None:
            x, y, w, h = cv2.boundingRect(coords)
            img = img[y:y+h, x:x+w]
            img = cv2.resize(img, (20, 20))
            img = np.pad(img, ((4, 4), (4, 4)), mode='constant', constant_values=0)
        else:
            st.warning("Empty drawing detected – please draw a digit before predicting.")
    except Exception as e:
        st.warning(f"Couldn't center the digit properly: {e}")

    img = img.reshape(1, 28, 28, 1)
```

9. Prediction

The final preprocessed image is reshaped and passed to the model. The predicted digit is extracted using `np.argmax()`.

```
    # Predict
    pred = model.predict(img)
    digit = np.argmax(pred)
```

10. Display Results

The app shows the user's drawing and the predicted digit using styled Streamlit/HTML output.

```
# Show results
st.image(canvas_result.image_data, caption="Your Drawing", width=150)
st.markdown(
    f"<h2 style='text-align: center; color: green;'>🖍️ Predicted Digit: {digit}</h2>",
    unsafe_allow_html=True
)
```

\

Conclusion

Uploaded the application file in github and deployed it using streamlit community

Application Link : [Digit Recogniser]

Output :