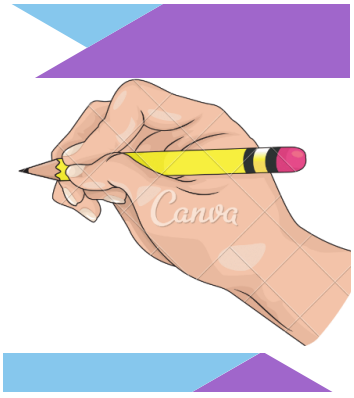
Handwritten Digit Recognition



Overview

Introduction

The Handwritten Digit Recognizer is a simple application that allows users to draw digits on a canvas using a graphical user interface (GUI) built with Tkinter. The application leverages a neural network model trained on the MNIST dataset to predict the digit drawn by the user.

Features

Canvas Drawing: Users can draw digits on the canvas using the left mouse button.

Prediction: The application predicts the drawn digit using a pre-trained neural network model.

Technical Details

Technologies Used

Python: The core programming language for the application.

NumPy: For numerical operations and array handling.

Pandas: For data manipulation and handling CSV files.

Matplotlib: For data visualization and plotting.

TensorFlow and Keras: For building and training the neural network model.

Tkinter: For building the graphical user interface.

PIL (Pillow): For image processing and handling.

Neural Network Model

The application uses a simple neural network with the following architecture:

Input Layer: Flatten layer to reshape the input image.

Hidden Layer: Dense layer with 128 neurons and ReLU activation.

Dropout Layer: Dropout layer with a dropout rate of 0.2 to prevent overfitting.

Output Layer: Dense layer with 10 neurons (corresponding to digits 0-9) and softmax activation.

Data Handling

Training Data: Loaded from a CSV file containing labeled digit images.

Testing Data: Loaded from another CSV file for potential future enhancements.

Data Preprocessing: Pixel values normalized to be between 0 and 1.

GUI Design

The GUI is created using Tkinter and includes:

Canvas: For drawing digits.

Buttons: "Clear" to clear the canvas and "Predict" to predict the drawn digit.

Prediction Label: Displays the predicted digit.

Image Processing

The application uses the PIL library for image processing:

Drawing Capture: Captures the content of the canvas.

Grayscale and Inversion: Converts the image to grayscale and inverts colors.

Resizing: Resizes the image to the required input size (28x28 pixels).

Normalization: Normalizes pixel values to be between 0 and 1.

Usage

Installation

Install the required Python libraries: NumPy, Pandas, Matplotlib, TensorFlow, Pillow.

Ensure the CSV files containing training and testing data are available.

Running the Application

Run the provided Python script.

Draw a digit on the canvas.

Click the "Predict" button to see the predicted digit.

Use the "Clear" button to erase the canvas and try again.


```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from tensorflow import keras

from tensorflow.keras import layers

from tkinter import Canvas, Tk, Button, Label, PhotoImage

from io import BytesIO

from PIL import Image, ImageOps, ImageTk

from PIL import ImageGrab
```

 # Load the training and testing datasets

```
train_data = pd.read_csv("F:\Documents\digittrain.csv")

test_data = pd.read_csv("F:\Documents\digittest.csv")
```

 # Separate labels from features in the training dataset

```
X_train = train_data.drop('label', axis=1)

y_train = train_data['label']


# Normalize pixel values to be between 0 and 1

X_train = X_train / 255.0

test_data = test_data / 255.0


# Split the training dataset into training and validation sets

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)


# Build the neural network model

model = keras.Sequential([

   layers.Flatten(input_shape=(784,)),  # Input layer

   layers.Dense(128, activation='relu'),  # Hidden layer with ReLU activation

   layers.Dropout(0.2),  # Dropout layer to prevent overfitting

   layers.Dense(10, activation='softmax')  # Output layer with softmax activation

])


# Compile the model

model.compile(optimizer='adam',

          loss='sparse_categorical_crossentropy',

          metrics=['accuracy'])


# Train the model

model.fit(X_train.values.reshape(-1, 784), y_train, epochs=10, validation_data=
(X_val.values.reshape(-1, 784), y_val))
```

```
# Create Tkinter GUI
root = Tk()
root.title("Handwritten Digit Recognizer")
canvas = Canvas(root, width=280, height=280, bg='white')
canvas.grid(row=0, column=0, columnspan=2)
label_prediction = Label(root, text="Prediction: ")
label_prediction.grid(row=1, column=0, columnspan=2)

# Function to handle drawing on the canvas
def draw(event):
    x1, y1 = (event.x - 10), (event.y - 10)
    x2, y2 = (event.x + 10), (event.y + 10)
    canvas.create_oval(x1, y1, x2, y2, fill='black', width=5)

# Function to predict the digit
def predict_digit():
    # Get the bounding box of the drawn content on the canvas
    x = root.winfo_rootx() + canvas.winfo_x()
    y = root.winfo_rooty() + canvas.winfo_y()
    x1 = x + canvas.winfo_width()
    y1 = y + canvas.winfo_height()

    # Capture the content of the canvas in the specified region
    img = ImageGrab.grab().crop((x, y, x1, y1))

    # Convert to grayscale and invert colors
```

```
    img = ImageOps.invert(ImageOps.grayscale(img))

    # Resize the image to 28x28 pixels
    resized_img = np.array(img.resize((28, 28), Image.LANCZOS))

    # Normalize pixel values
    resized_img = resized_img / 255.0
    try:
        # Reshape the image
        reshaped_img = resized_img.reshape(1, 784)
       # Make predictions
        prediction = model.predict(reshaped_img)
        predicted_label = np.argmax(prediction)
        # Display the prediction
        label_prediction.config(text=f"Prediction: {predicted_label}")
    except ValueError as e:
        print("Error:", e)

# Button to clear the canvas
button_clear = Button(root, text="Clear", command=lambda: canvas.delete("all"))
button_clear.grid(row=2, column=0)

# Button to predict the digit
button_predict = Button(root, text="Predict", command=predict_digit)
button_predict.grid(row=2, column=1)
```

 # Bind the draw function to the canvas

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

root.mainloop()

Result

Conclusion

The Handwritten Digit Recognizer provides a basic yet effective demonstration of digit recognition using a neural network. With further enhancements, it can serve as a foundation for more sophisticated applications in the field of image recognition and