****

Signature Verification using Python

*A project report submitted*

*to*

**MANIPAL ACADEMY OF HIGHER EDUCATION**

*For Partial Fulfilment of the Requirement for the*

*Award of the Degree*

*of*

**Bachelor of Technology**

*in*

**Information Technology**

*by*

**Rithika Narikimilli, Nihitha R Haresamudram**

**Reg. No. 225811256, 225811342**

*Under the guidance of*

Dr. Abhijit Das

Assistant Professor – Senior Scale

Department of I &T

Manipal Institute of Technology

Bengaluru, India

****

**Nov 2024**

**DECLARATION**

We hereby declare that this project work entitled Signature Verification using Python

is original and has been carried out by us in the Department of Information Technology of Manipal Institute of Technology, Bengaluru, under the

guidance of **Dr. Abhijit Das**, **Assistant Professor- Senior Scale**, Department of Information Technology, MIT, Bengaluru. No part of this work has been submitted for the award of a degree or diploma either to this University or to any other Universities.

Place: Bengaluru

Date :06-11-24

RITHIKA NARIKIMILLI

NIHITHA R HARESAMUDRAM

****

**CERTIFICATE**

This is to certify that this project entitled **Signature Verification using Python**

is a bonafide project work done by **Rithika Narikimilli (Reg.No.: 225811256)** and **Nihitha R Haresamudram (Reg.No.: 225811342)** at Manipal Institute of Technology, Bengaluru, independently under our guidance and supervision for the award of the Degree of Bachelor of Technology in Information Technology.

Dr. Abhijit Das Dr. Dayananda P

Assistant Professor – Senior Scale Professor & Head

Department of I &T Department of I & T

Manipal Institute of Technology Manipal Institute of Technology

Bengaluru, India Bengaluru, India

**Table of Contents**

Introduction

Objectives

Tools and Technology used

System Workflow

Implementation Details

Results

Challenges Faced

Future Enhancements

Conclusion

Codes

**1. Introduction**

Signature verification is a biometric method commonly used for identity authentication in various sectors, such as banking, legal documentation, and other applications requiring secure verification. This project focuses on creating a signature matching system that compares two input signatures and returns their similarity percentage. The system uses a combination of image processing techniques and a graphical user interface (GUI) to streamline the process.

**2. Objectives**

* To develop a user-friendly application that allows users to capture, upload, and compare signatures.
* To achieve accurate results by employing image processing techniques to measure signature similarity.
* To ensure a smooth user experience through an intuitive interface built using Tkinter.

**3. Tools and Technologies Used**

* **Programming Language**: Python
* **Libraries**:
  + **OpenCV**: For image processing, capture, and manipulation.
  + **Tkinter**: For the graphical user interface.
  + **skimage (SSIM)**: To compute the structural similarity between two images.
* **Hardware Requirements**: Camera for capturing signatures in real-time.

**4. System Workflow**

**4.1 User Interface**

The system provides an easy-to-use interface where users can:

* Capture a signature using their webcam.
* Browse and upload an image file from their local machine.
* Compare two signatures to check for similarity.

**4.2 Signature Capture and Upload**

* **Image Capture**: The system allows users to capture an image using a webcam. The user can press the space bar to capture the signature and ESC to exit the capture window.
* **Image Upload**: Alternatively, the user can browse their local system to select an image file of a signature. Supported formats include .jpg, .jpeg, and .png.

**4.3 Signature Comparison**

The similarity between the two signatures is calculated using **structural similarity index (SSIM)**:

* The images are converted to grayscale to simplify the comparison.
* Images are resized to 300x300 pixels for uniformity.
* SSIM is computed to determine the similarity score between the two images.
* Based on the calculated score, the system displays whether the signatures match or not.

**4.4 Result Display**

* If the similarity percentage is above the defined threshold (85%), a success message is displayed, indicating that the signatures match.
* If the similarity percentage is below the threshold, a failure message is shown, indicating that the signatures do not match.

**5. Implementation Details**

**5.1 Threshold Value**

* The similarity threshold is set to 85%, which means that signatures are considered a match if they are 85% or more similar. This value can be adjusted based on application requirements and testing results.

**5.2 Key Functions**

* **browsefunc()**: Allows users to browse and select an image file.
* **capture\_image\_from\_cam\_into\_temp ()**: Captures an image from the webcam and saves it as a temporary file.
* **captureImage()**: Manages the process of capturing or uploading a signature, displaying it in the interface.
* **checkSimilarity()**: Compares two signature images and displays the result.
* **match()**: The core function that performs the image similarity comparison using SSIM.

**6. Results**

After comparing two signatures, the system provides a percentage similarity score. If the score exceeds the threshold, it confirms that the signatures match. Otherwise, it highlights that they do not match.

**Example Result:**

* Signature 1: *captured/uploaded*
* Signature 2: *captured/uploaded*
* Similarity Score: 90%
* **Result**: Success, Signatures Match!

**7. Challenges Faced**

* **Image Preprocessing**: Resizing images and converting them to grayscale was necessary to ensure that the comparison was accurate, but this also led to slight variations in results.
* **Camera Integration**: Handling different camera resolutions and ensuring compatibility across systems required additional debugging.
* **Similarity Threshold**: Determining the appropriate threshold value for signature comparison involved testing across various signature samples.

**8. Future Enhancements**

* **Contour Detection**: Implementing contour detection can further improve the accuracy of the system by considering the geometric shapes of the signatures.
* **Machine Learning Integration**: Exploring ML models for signature verification could offer more robust results, especially for cases where signatures are partially distorted.
* **Multiple Signature Database**: Allowing users to compare signatures against a pre-existing database to handle multiple comparisons efficiently.

**9. Conclusion**

This project successfully demonstrates how Python, OpenCV, and Tkinter can be used to develop a functional and user-friendly signature matching system. While the current system achieves its objectives, there is potential for further improvement and expansion of the system to support more complex use cases.

**CODES:**

**main.py**

import tkinter as tk

from tkinter.filedialog import askopenfilename

from tkinter import messagebox

import os

import cv2

from numpy import result\_type

from signature import match

THRESHOLD = 85

def browsefunc(ent):

filename = askopenfilename(filetypes=([

("image", ".jpeg"),

("image", ".png"),

("image", ".jpg"),

]))

ent.delete(0, tk.END)

ent.insert(tk.END, filename) # add this

def capture\_image\_from\_cam\_into\_temp(sign=1):

cam = cv2.VideoCapture(0, cv2.CAP\_DSHOW)

cv2.namedWindow("test")

# img\_counter = 0

while True:

ret, frame = cam.read()

if not ret:

print("failed to grab frame")

break

cv2.imshow("test", frame)

k = cv2.waitKey(1)

if k % 256 == 27:

# ESC pressed

print("Escape hit, closing...")

break

elif k % 256 == 32:

# SPACE pressed

if not os.path.isdir('temp'):

os.mkdir('temp', mode=0o777) # make sure the directory exists

# img\_name = "./temp/opencv\_frame\_{}.png".format(img\_counter)

if(sign == 1):

img\_name = "./temp/test\_img1.png"

else:

img\_name = "./temp/test\_img2.png"

print('imwrite=', cv2.imwrite(filename=img\_name, img=frame))

print("{} written!".format(img\_name))

# img\_counter += 1

cam.release()

cv2.destroyAllWindows()

return True

def captureImage(ent, sign=1):

if(sign == 1):

filename = os.getcwd()+'\\temp\\test\_img1.png'

else:

filename = os.getcwd()+'\\temp\\test\_img2.png'

# messagebox.showinfo(

# 'SUCCESS!!!', 'Press Space Bar to click picture and ESC to exit')

res = None

res = messagebox.askquestion(

'Click Picture', 'Press Space Bar to click picture and ESC to exit')

if res == 'yes':

capture\_image\_from\_cam\_into\_temp(sign=sign)

ent.delete(0, tk.END)

ent.insert(tk.END, filename)

return True

def checkSimilarity(window, path1, path2):

result = match(path1=path1, path2=path2)

if(result <= THRESHOLD):

messagebox.showerror("Failure: Signatures Do Not Match",

"Signatures are "+str(result)+f" % similar!!")

pass

else:

messagebox.showinfo("Success: Signatures Match",

"Signatures are "+str(result)+f" % similar!!")

return True

root = tk.Tk()

root.title("Signature Matching")

root.geometry("500x700") # 300x200

uname\_label = tk.Label(root, text="Compare Two Signatures:", font=10)

uname\_label.place(x=90, y=50)

img1\_message = tk.Label(root, text="Signature 1", font=10)

img1\_message.place(x=10, y=120)

image1\_path\_entry = tk.Entry(root, font=10)

image1\_path\_entry.place(x=150, y=120)

img1\_capture\_button = tk.Button(

root, text="Capture", font=10, command=lambda: captureImage(ent=image1\_path\_entry, sign=1))

img1\_capture\_button.place(x=400, y=90)

img1\_browse\_button = tk.Button(

root, text="Browse", font=10, command=lambda: browsefunc(ent=image1\_path\_entry))

img1\_browse\_button.place(x=400, y=140)

image2\_path\_entry = tk.Entry(root, font=10)

image2\_path\_entry.place(x=150, y=240)

img2\_message = tk.Label(root, text="Signature 2", font=10)

img2\_message.place(x=10, y=250)

img2\_capture\_button = tk.Button(

root, text="Capture", font=10, command=lambda: captureImage(ent=image2\_path\_entry, sign=2))

img2\_capture\_button.place(x=400, y=210)

img2\_browse\_button = tk.Button(

root, text="Browse", font=10, command=lambda: browsefunc(ent=image2\_path\_entry))

img2\_browse\_button.place(x=400, y=260)

compare\_button = tk.Button(

root, text="Compare", font=10, command=lambda: checkSimilarity(window=root,

path1=image1\_path\_entry.get(), path2=image2\_path\_entry.get(),))

compare\_button.place(x=200, y=320)

root.mainloop()

**signature.py**

import cv2

from skimage.metrics import structural\_similarity as ssim

import matplotlib.pyplot as plt

def show\_images(img1, img2):

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)

    plt.imshow(img1, cmap='gray')

    plt.title("Image 1")

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.imshow(img2, cmap='gray')

    plt.title("Image 2")

    plt.axis('off')

    plt.show()

def match(path1, path2):

    # read the images

    img1 = cv2.imread(path1)

    img2 = cv2.imread(path2)

    # Check if images are loaded successfully

    if img1 is None:

        print(f"Error: Unable to load image at {path1}")

        return None

    if img2 is None:

        print(f"Error: Unable to load image at {path2}")

        return None

    # turn images to grayscale

    img1 = cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)

    img2 = cv2.cvtColor(img2, cv2.COLOR\_BGR2GRAY)

    # resize images for comparison

    img1 = cv2.resize(img1, (300, 300))

    img2 = cv2.resize(img2, (300, 300))

    # Display both images using matplotlib

    show\_images(img1, img2)

    # Calculate similarity value

    similarity\_value = ssim(img1, img2) \* 100  # SSIM value is between 0 and 1, convert to percentage

    similarity\_value = "{:.2f}".format(similarity\_value)

    return float(similarity\_value)

# Example usage

ans = match(r"D:\Code\Git stuff\Signature-Matching\assets\1.png",

            r"D:\Code\Git stuff\Signature-Matching\assets\3.png")

if ans is not None:

    print("Similarity Percentage:", ans)

**Output:**

A screenshot of a computer

Description automatically generated