INTERNATIONAL ISLAMIC UNIVERSITY CHITTAGONG

Department of Computer Science & Engineering (CSE)

Project Report

Project Title: **Signature Verification System**

Course Code: **CSE-4876**

Course Title: Pattern Recognition & Image Processing Lab

Submitted To	Submitted By
Mohammad Zainal Abedin **Assistant Professor, Dept. of CSE.** **International Islamic University Chittagong**	**Team Name** : Team_JRF **Team Member:** Mohammad Rahatul Islam(C193075) Jabed Iqbal Joy (C193049) Md Shah Ibne Fahad(C193048)

Abstract: Using Python, OpenCV, and Tkinter, the Signature Verification System is a cutting-edge software solution designed to make it easier to compare and validate signature images. The project aims to tackle the pressing requirement for dependable and effective signature authentication systems across several industries, such as banking, legal paperwork, and security access**.**

Introduction: The Signature Verification System is an innovative and secure solution designed to enhance document authenticity and reduce the risk of forgery in various applications. This project aims to develop a robust and efficient system that verifies the authenticity of handwritten signatures, ensuring the integrity of signed documents.

Project Overview: Our proposed Signature Verification System will be designed to address the following key objectives.

Enhanced Security: Implement a robust and reliable method for verifying the authenticity of signatures on documents, contracts, and other critical records.

Efficiency: Streamline the signature verification process to reduce manual effort and human error associated with traditional methods.

Accuracy: Employ state-of-the-art machine learning algorithms to achieve high accuracy in signature recognition and verification.

User-Friendly Interface: Develop an intuitive and user-friendly interface that allows users to easily submit and verify signatures.

Methodology:

Technologies Used

Python: Core programming language

OpenCV: Image processing library for image manipulation and feature extraction.

Tkinter: GUI toolkit for building the interactive user interface

CSV: Handling the dataset containing user information and signature paths

Implementation Details

GUI Development: Tkinter library is used to create a user-friendly interface with text fields, buttons, and functionalities.

Signature Comparison Algorithm: The project implements an algorithm using OpenCV to compare signatures. It reads, processes, and compares the similarity between two images.

Dataset Handling: The system interacts with a CSV file containing user information to retrieve user details and signature paths.

Code Structure Overview

The code consists of two main files:

main.py: Contains the GUI implementation using Tkinter and handles user interaction.

```
import tkinter as tk

from tkinter.filedialog import askopenfilename

from tkinter import messagebox

from tkinter import *

import os

import cv2

from numpy import result_type

from signature import match

import csv

# Mach Threshold

THRESHOLD = 80

path="demo.png"

def load_user_name():

    found=False

    global path

    for row in dataset:
        if(row[0] == userid_data.get()):
            found=True
            path=row[3]
    if(found==False):
        messagebox.showerror("Failure: User Not Found","User Not Found!!")
    else:
        username_label = tk.Label(root, text="User Name : "+row[1], font=10)
        username_label.place(x=10, y=180)
def browsefunc(ent):
    filename = askopenfilename(filetypes=([
        ("image", ".jpeg"),
        ("image", ".png"),
        ("image", ".jpg"),
    ]))
    ent.delete(0, tk.END)
    ent.insert(tk.END, filename)
    print(filename)  # add this
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
dataset= csv.reader(open('D:\Signature-Matching\database\IP_Dataset.csv','r'))
root = tk.Tk()
root.title("Signature Verification")
root.geometry("500x500")  # 300x200
```

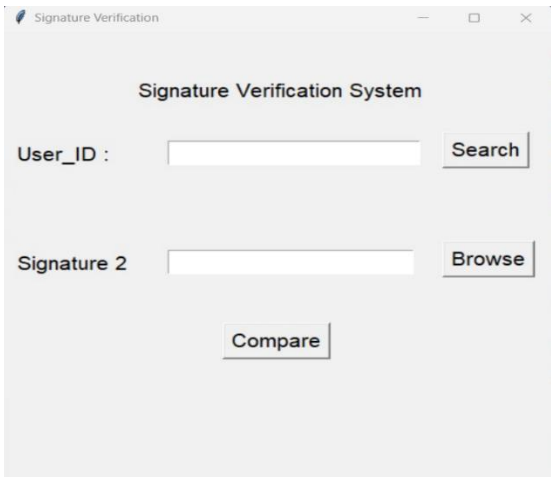
```
label = tk.Label(root, text="Signature Verification System", font=10)

label.place(x=120, y=50)

userid_label = tk.Label(root, text="User_ID : ", font=10)

userid_label.place(x=10, y=120)

userid_data = StringVar()

userid_textbox = tk.Entry(root,textvariable=userid_data ,font=('Arial',15))

userid_textbox.place(x=150, y=120,height=27,width=230)

search_user = tk.Button(root,command=lambda:load_user_name(), text="Search",
font=10)

search_user.place(x=400,y=110)

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=240)

img2_browse_button = tk.Button(root, text="Browse", font=10, command=lambda:
browsefunc(ent=image2_path_entry))

img2_browse_button.place(x=400, y=230)

compare_button = tk.Button(root, text="Compare", font=10, command=lambda:
checkSimilarity(window=root,path1=path,path2=image2_path_entry.get(),))

compare_button.place(x=200, y=320)

root.mainloop()
```

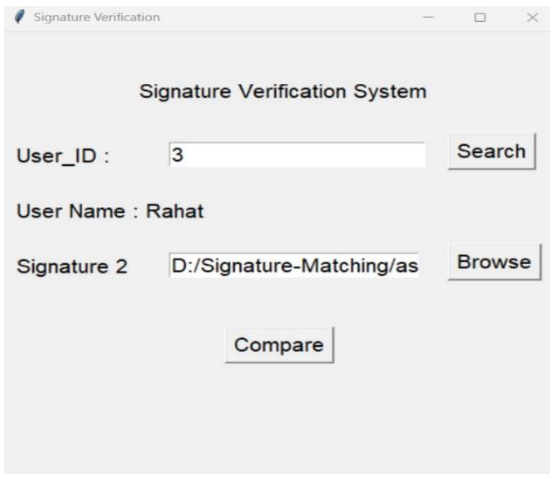
signature.py: Contains the signature comparison algorithm using OpenCV and SSIM (Structural Similarity Index).

```
import cv2

from skimage.metrics import structural_similarity as ssim

def match(path1, path2):

    # read the images

    img1 = cv2.imread(path1)

    img2 = cv2.imread(path2)

    # 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
cv2.imshow("One", img1)
cv2.imshow("Two", img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
similarity_value = "{:.2f}".format(ssim(img1, img2)*100)
# print("answer is ", float(similarity_value),
#       "type=", type(similarity_value))
return float(similarity_value)
```

Output





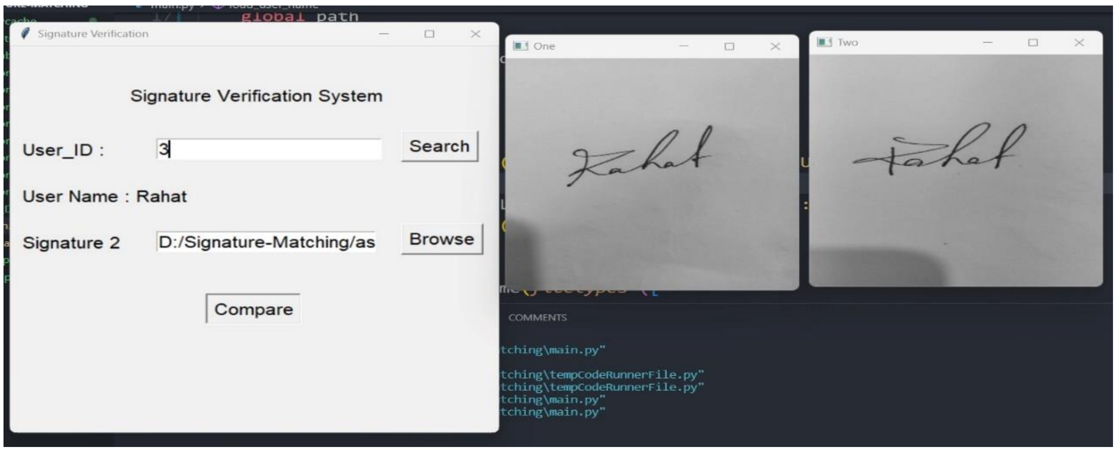
Signature Verification

Signature Verification System

User_ID : 3

User Name : Rahat

Signature 2 D:/Signature-Matching/as



Signature Verification

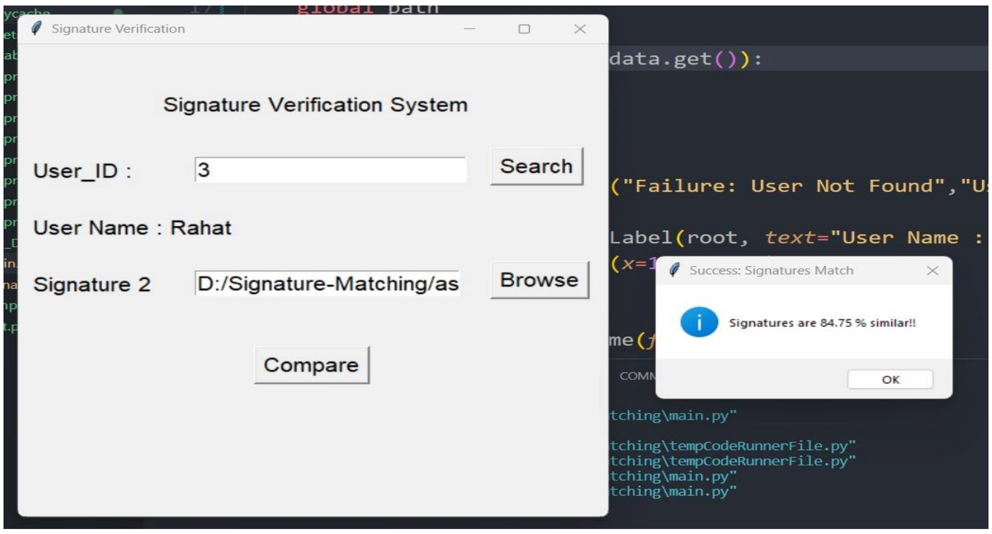
Signature Verification System

User_ID : 3

User Name : Rahat

Signature 2 D:/Signature-Matching/as

```
tching\main.py"
tching\tempCodeRunnerFile.py"
tching\tempCodeRunnerFile.py"
tching\main.py"
tching\main.py"
```



Signature Verification

Signature Verification System

User_ID : 3 Search

User Name : Rahat

Signature 2 D:/Signature-Matching/as Browse

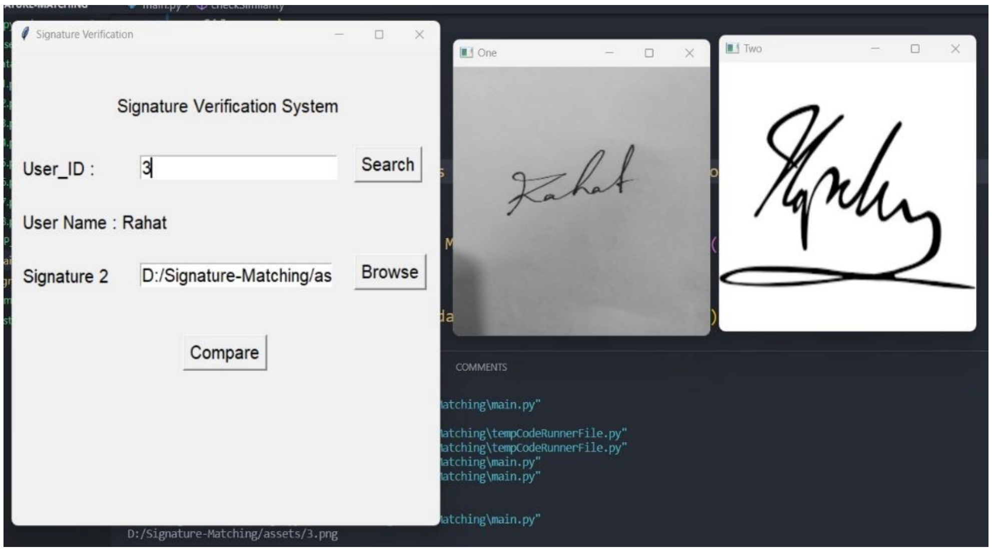
Compare

Success: Signatures Match

Signatures are 84.75 % similar!!

OK

```
data.get()):
("Failure: User Not Found","U
Label(root, text="User Name :
tching\main.py"
tching\tempCodeRunnerFile.py"
tching\tempCodeRunnerFile.py"
tching\main.py"
tching\main.py"
```



Signature Verification

Signature Verification System

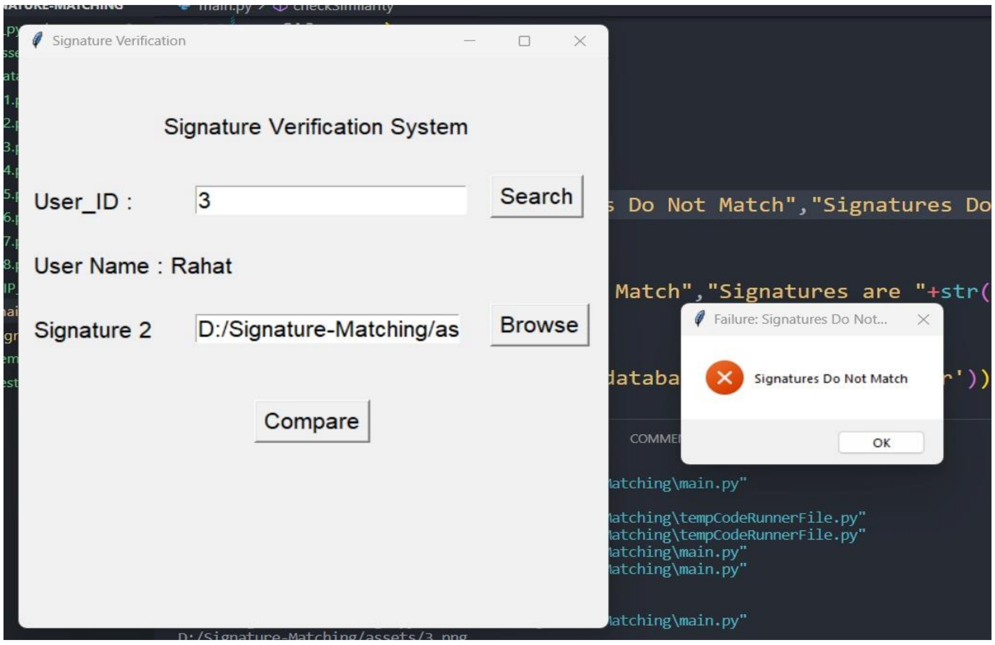
User_ID : 3 Search

User Name : Rahat

Signature 2 D:/Signature-Matching/as Browse

Compare

```
atching\main.py"
atching\tempCodeRunnerFile.py"
atching\tempCodeRunnerFile.py"
atching\main.py"
atching\main.py"
atching\main.py"
D:/Signature-Matching/assets/3.png
```



Signature Verification

Signature Verification System

User_ID : 3 Search

User Name : Rahat

Signature 2 D:/Signature-Matching/as Browse

Compare

Failure: Signatures Do Not...

Signatures Do Not Match

OK

```
s Do Not Match","Signatures Do
Match","Signatures are "+str(
databa
atching\main.py"
atching\tempCodeRunnerFile.py"
atching\tempCodeRunnerFile.py"
atching\main.py"
atching\main.py"
atching\main.py"
D:/Signature-Matching/assets/3.png
```

Applications:

Banking and Financial Institutions:

Document Authentication: Verify signatures on financial documents, cheques, or contracts to prevent fraud and ensure authenticity.

Legal and Contractual Processes:

Contract Verification: Authenticate signatures on legal documents to validate contracts and agreements efficiently.

Security and Access Control:

Identity Verification: Utilize the system for access control in secure environments by verifying signatures for identity validation.

Administrative and Governmental Bodies:

Official Document Validation: Validate signatures on official paperwork within administrative or governmental institutions to ensure accuracy.

Future Enhancements:

Advanced Matching Algorithms:

Machine Learning Integration: Incorporate machine learning models for more sophisticated signature analysis and enhanced accuracy.

User Authentication Integration:

Biometric Verification: Integrate biometric methods like fingerprint or face recognition for additional user verification layers.

Real-time Signature Capture:

Live Signature Comparison: Enable real-time signature capture and comparison using devices like tablets or touchscreens.

User Interface Refinement:

Enhanced User Experience: Improve the GUI design for better user interaction, clearer instructions, and intuitive navigation.

Security and Encryption Measures:

Data Security Protocols: Implement robust security measures to protect sensitive user data and signature information.

Scalability and Performance Optimization:

Optimized Processing: Enhance system performance for larger datasets and expedited signature comparisons.

Conclusion

The Signature Verification System provides a reliable way to confirm signatures in a variety of industries, including banking, legal documents, and access control. The system offers an easy-to-use interface that enables users to compare signatures quickly through the integration of Python, OpenCV, and Tkinter. Using image processing techniques, it can currently retrieve and compare signatures based on user ID. Accuracy and security are expected to be improved in the future by use of sophisticated algorithms and improved user verification. In general, this technology fosters trust and authenticity in document validation procedures across various industries by addressing the crucial need for trustworthy signature verification.