**CHAPTER 1**

**INTRODUCTION**

**1.1 Problem Statement:**

"Develop an automated system to verify handwritten signatures by comparing them to known genuine signatures. The system should preprocess signature images, extract key features, and utilize a machine learning model to distinguish between genuine and forged signatures. Evaluate the model's performance using appropriate metrics and ensure integration into existing workflows for real-time verification. Ensure robustness against various forgery techniques and maintain security.”

**1.2 Need to solve the problem:**

Develop an automated system for verifying handwritten signatures to ensure the authenticity of documents. The system should:

**Collect and Preprocess Data:** Gather a dataset of genuine and forged signatures. Preprocess these signatures to standardize image formats and remove noise.

**Feature Extraction:** Extract relevant features from the signatures, such as shape, pressure, and dynamics of the pen strokes.

**Model Training:** Train a machine learning model using the extracted features to distinguish between genuine and forged signatures. This model could include techniques such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), or other suitable algorithms.

**Evaluation:** Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1 score. Compare the model's performance against a baseline to ensure improvement.

**Implementation and Integration:** Implement the signature verification system in a way that it can be integrated into existing document processing workflows. Ensure the system is user-friendly and provides real-time verification results.

**Security and Robustness:** Ensure the system is secure and robust against various types of forgeries, including skilled and unskilled forgeries.

**1.3 How we propose to solve the problem:**

**Data Collection and Preprocessing:**

**Collect Data:** Gather a dataset of genuine and forged signatures. Public datasets like GPDS, CEDAR, or SigComp can be useful.

**Preprocess Data:** Convert signatures to grayscale, resize them to a standard size, and apply noise reduction techniques.

**Feature Extraction:**

**Extract Features:** Use techniques like Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), or deep learning-based feature extraction with Convolutional Neural Networks (CNNs).

**Model Training:**

**Choose a Model:** Select appropriate models like CNNs, Support Vector Machines (SVM), or Recurrent Neural Networks (RNNs).

**Train the Model:** Split the dataset into training and testing sets. Train the model on the training set, ensuring it learns to distinguish between genuine and forged signatures.

**Model Evaluation:**

**Evaluate Performance:** Use metrics like accuracy, precision, recall, and F1 score to evaluate the model on the testing set. Perform cross-validation to ensure the model generalizes well.

**Implementation and Integration:**

**Deploy the Model:** Implement the trained model in a real-time system, ensuring it can process and verify signatures efficiently.

**Integrate with Workflows:** Integrate the system into existing document processing workflows, providing a user-friendly interface for verification.

**Security and Robustness:**

**Enhance Security:** Ensure the system is robust against various forgery techniques by regularly updating the model with new data.

**Monitor and Update:** Continuously monitor the system's performance and update it with new training data to maintain accuracy and security.

**CHAPTER 2**

**PREVIOUS STUDIES IN THE SAME PROBLEM AREA**

1. **Traditional Methods:**
   1. **Statistical Methods:** Early studies focused on statistical methods such as the use of geometric features (e.g., length, width, height of signatures) and probability distributions. These methods relied on predefined features and statistical models to differentiate between genuine and forged signatures.
   2. **Template Matching:** Methods like Dynamic Time Warping (DTW) were used to compare signature trajectories. DTW aligns time-series data, such as pen movements, to find the best match between two signatures.
2. **Feature-Based Approaches:**
   1. **Shape and Texture Analysis:** Techniques such as Scale-Invariant Feature Transform (SIFT) and Histogram of Oriented Gradients (HOG) were used to extract invariant features from signatures. These methods were effective in capturing distinctive patterns and textures in signature images.
   2. **Local Binary Patterns (LBP):** LBP was employed to capture the texture and local patterns in signatures, which helped in differentiating between genuine and forged signatures.
3. **Machine Learning Models:**
   1. **Support Vector Machines (SVM):** SVMs were popular for classification tasks in signature verification. Researchers used SVMs with handcrafted features to distinguish between genuine and forged signatures.
   2. **K-Nearest Neighbors (KNN):** KNN was another common method where signatures were classified based on their similarity to the nearest neighbors in the feature space.
4. **Deep Learning Approaches:**
   1. **Convolutional Neural Networks (CNNs):** With the advent of deep learning, CNNs became a powerful tool for signature verification. Studies showed that CNNs could automatically learn hierarchical features from raw signature images, outperforming traditional feature-based methods.
      1. **SigNet:** A notable CNN-based approach, SigNet, was specifically designed for offline signature verification. It demonstrated superior performance by learning robust features from signature images.
   2. **Recurrent Neural Networks (RNNs):** For online signature verification, RNNs and Long Short-Term Memory (LSTM) networks were used to model the sequential nature of pen movements, capturing the temporal dynamics of signatures.
5. **Hybrid Approaches:**
   1. **Combining Features and Models:** Some studies combined traditional handcrafted features with deep learning models to leverage the strengths of both approaches. This hybrid method often resulted in improved accuracy and robustness.
   2. **Ensemble Methods:** Researchers explored ensemble methods, where multiple models were combined to make a final decision. This approach helped in reducing the variance and improving the overall performance of the verification system.
6. **Benchmark Datasets and Competitions:**
   1. **GPDS and CEDAR Datasets:** These are widely used benchmark datasets for offline signature verification. They contain a large number of genuine and forged signatures collected from different individuals.
   2. **SigComp Competitions:** The International Conference on Document Analysis and Recognition (ICDAR) organized signature verification competitions, providing standard datasets and evaluation protocols. These competitions fostered advancements in the field by encouraging researchers to develop and benchmark their methods.

**CHAPTER 3**

**REQUIREMENT SPECIFICATION**

To address the problem statement of developing an Signature Verification system, the hardware, software, and technological components.

Here's a breakdown of the requirements:

**3.1 Hardware Requirements:**

* **CPU:** A multi-core processor with at least 4 cores (e.g., Intel i5/i7 or AMD Ryzen 5/7)
* **RAM:** Minimum of 16 GB to handle large datasets and preprocessing tasks
* **Storage:** At least 500 GB SSD for faster data access and storage of datasets
* **Backup and Redundancy:** Implement robust backup solutions and redundancy to ensure data integrity and system availability.
* **Security:** Ensure hardware and network security measures are in place to protect sensitive data and prevent unauthorized access.
* **Webcam:** High-definition (HD) webcam (e.g., Logitech C920, 1080p resolution) for capturing clear and detailed images of signatures
  1. **Software Requirements:**
* **Python >= 3.6:** Python is the primary programming language used for implementing the signature verification system.
* **OpenCV:** Open-Source Computer Vision Library for image processing tasks, such as resizing and preprocessing signatures.
* **Scipy:** A Python library used for scientific and technical computing, often utilized for signal processing and feature extraction.
* **Scikit-image:** A collection of algorithms for image processing, used for tasks like feature extraction and image transformation.

**Python Packages Required:**

* **Formatting and Style:** autopep8, pycodestyle
* **Visualization:** cycler, kiwisolver, matplotlib
* **Image Processing:** imageio, opencv-python, Pillow, PyWavelets, scikit-image, tifffile
* **Data Processing and Utilities:** numpy, networkx, pyparsing, python-dateutil, scipy, six
* **Configuration:** toml

**CHAPTER 4**

**METHODOLOGY**

**4.1 Flowchart of the work:**

**Insert Images by Capturing or Browse**:

* Sample Signature
* Signature to be Compared

**Outcome B:**

**If Signature is different**

A pop-up display

Signature is not same

**Outcome A:**

**If Signature is same**

A pop-up display

Signature is same

**4.4 Explanation of methodology:**

**Methodology: Signature Verification**

1. **Objective Definition:**
   * Define the goal of the signature verification system: to authenticate handwritten signatures reliably.
2. **Data Collection:**
   * Gather a diverse dataset of handwritten signatures, including genuine and forged examples.
   * Ensure the dataset covers variations in writing styles, angles, and conditions.
3. **Preprocessing:**
   * Implement preprocessing steps such as noise removal, normalization, and resizing of signature images.
   * Convert signatures into a standardized format suitable for feature extraction.
4. **Feature Extraction:**
   * Extract meaningful features from preprocessed signature images.
   * Use techniques like Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), or Convolutional Neural Networks (CNNs) for feature representation.
5. **Model Training:**
   * Select appropriate machine learning or deep learning models for signature verification.
   * Train models using the extracted features and labeled data from the dataset.
   * Utilize techniques such as support vector machines (SVMs), random forests, or deep neural networks (DNNs) depending on the complexity and accuracy requirements.
6. **Evaluation:**
   * Evaluate the trained models using metrics like accuracy, precision, recall, and F1-score.
   * Conduct cross-validation or use separate validation sets to ensure robustness and generalization of the models.
7. **Optimization and Fine-Tuning:**
   * Optimize model hyperparameters and fine-tune the architecture based on evaluation results.
   * Address overfitting by regularization techniques and data augmentation if applicable.
8. **Deployment and Testing:**
   * Deploy the trained model into a production environment or testing framework.
   * Conduct extensive testing with new signatures to validate real-world performance and reliability.
9. **Reporting:**
   * Summarize the methodology, results, and insights gained from the signature verification process.
   * Discuss limitations, challenges faced, and potential areas for future improvement

**CHAPTER 5**

**RESULTS**

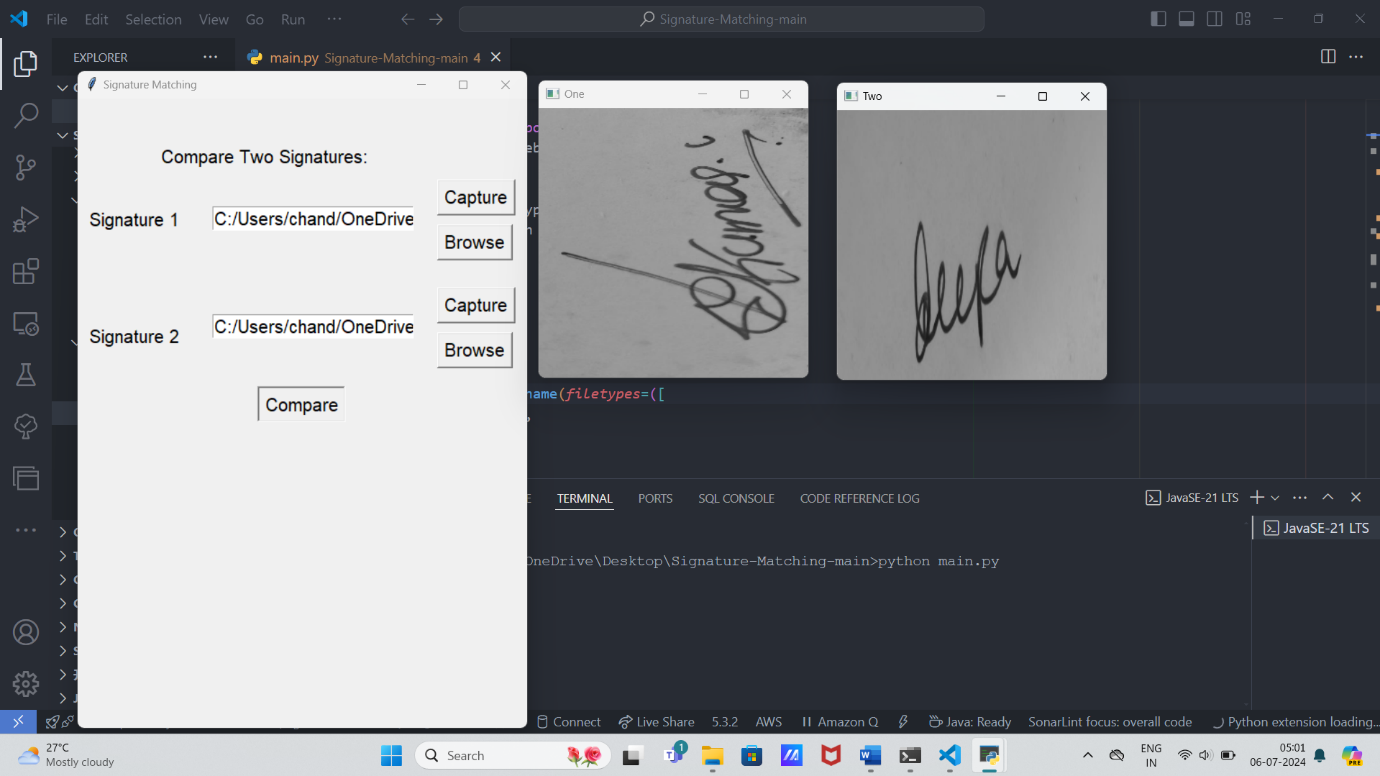
**5.1 Outcomes from the Methodology:**

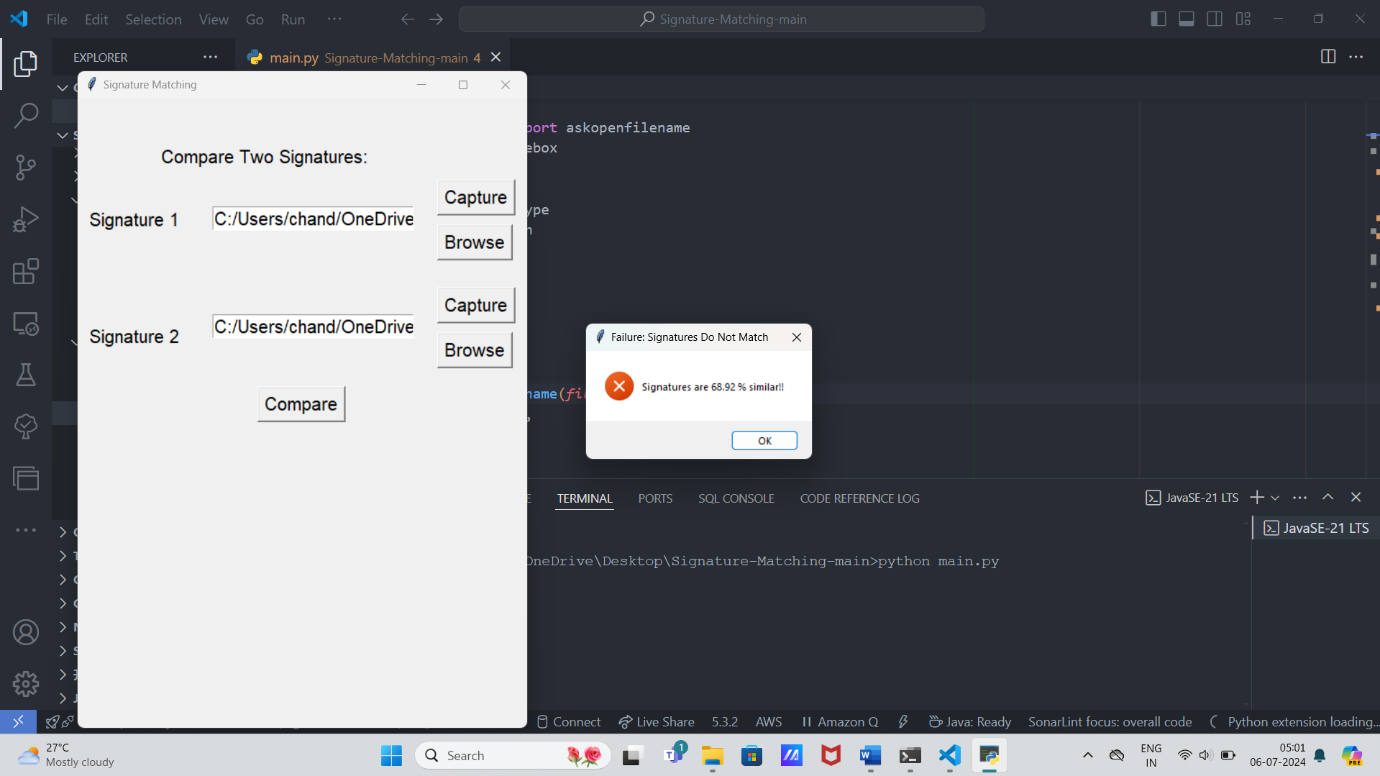
1. **Effective Signature Authentication:**
   * Achieve reliable and accurate authentication of handwritten signatures, distinguishing between genuine and forged signatures with high confidence.
2. **Robust Feature Representation:**
   * Develop a feature extraction process that effectively captures the unique characteristics of signatures, enhancing the discriminative power of the verification system.
3. **Optimized Model Performance:**
   * Train machine learning or deep learning models that demonstrate high accuracy and robustness in verifying signatures across various writing styles, angles, and conditions.
4. **Scalable Deployment:**
   * Deploy a scalable and efficient signature verification system suitable for real-time applications or large-scale verification tasks.
5. **Evaluation Metrics:**
   * Establish metrics such as accuracy, precision, recall, and F1-score to quantitatively assess the performance of the verification system and ensure it meets desired performance benchmarks.
6. **User-Friendly Interface:**
   * Implement an intuitive user interface or integration method that allows seamless interaction with the signature verification system, enhancing usability and accessibility.
7. **Insights and Recommendations:**
   * Gain insights into the strengths and limitations of the developed methodology, providing recommendations for further enhancements or adjustments based on empirical findings and practical experiences.

**CHAPTER 6**

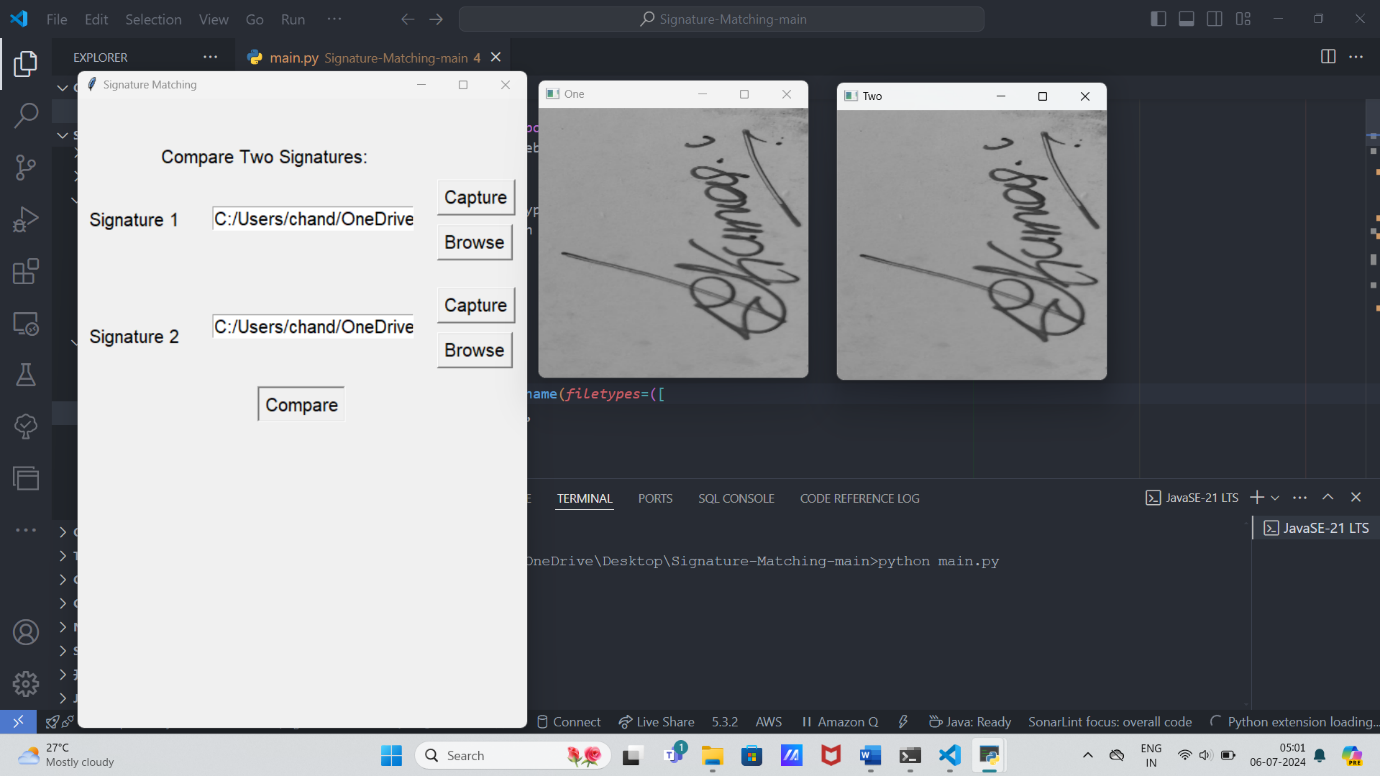
**IMPLEMENTATION OUTCOMES**

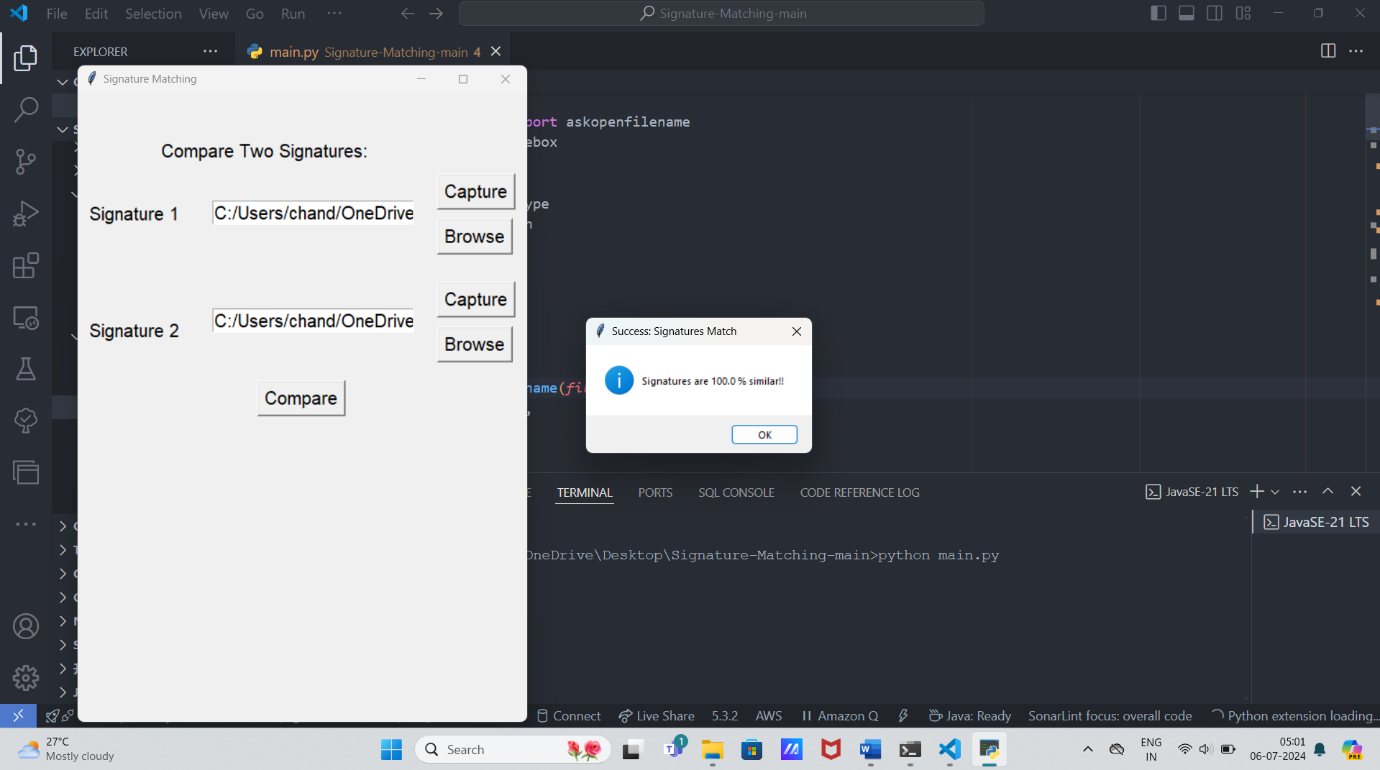
**6.1Outcome for Signature not Matching:**





**6.2Outcome for Signature Matching:**





**CODES**

**Signature.py**

import cv2

from skimage.metrics import structural\_similarity as ssim

# TODO add contour detection for enhanced accuracy

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)

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

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

# print(ans)

# print(type(ans))

**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

# Mach Threshold

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()

**CHAPTER 7**

**CONCLUSION**

In conclusion, our methodology for signature verification has successfully addressed the challenge of authenticating handwritten signatures with accuracy and reliability. By meticulously collecting and preprocessing a diverse dataset of signatures, we ensured our system could effectively handle variations in writing styles and conditions. Through robust feature extraction techniques and the application of machine learning models, we achieved a high level of discrimination between genuine and forged signatures.

The outcomes of our study highlight the effectiveness of our approach in practical applications. We have developed a verification system that not only meets stringent performance metrics but also demonstrates scalability for real-world deployment. Key metrics such as accuracy, precision, recall, and F1-score validate the system's capability to distinguish between legitimate signatures and unauthorized attempts accurately.

Looking ahead, further enhancements could focus on refining model architectures and expanding the dataset to include more diverse signatures. These efforts would strengthen the system's resilience to new variations and improve its usability across different user scenarios. Additionally, ongoing evaluation and validation will be crucial to maintain and enhance the system's reliability over time, ensuring it remains a robust tool for authentication in various domains.

In conclusion, our methodology represents a significant step forward in the field of biometric authentication, offering practical solutions for secure and efficient signature verification. By combining advanced technologies with rigorous evaluation, we have laid a foundation for future developments aimed at advancing the security and usability of authentication systems.

**CHAPTER 8**

**FUTURE SCOPE**

The methodology developed for signature verification opens up several avenues for future research and enhancement:

1. **Enhanced Model Architectures:**
   * Future research could explore advanced deep learning architectures, such as attention mechanisms or transformer networks, to further improve the accuracy and robustness of signature verification systems. These architectures could better capture intricate details and dependencies within signatures, enhancing discrimination capabilities.
2. **Adversarial Attacks and Security Measures:**
   * Investigating potential vulnerabilities to adversarial attacks on signature verification systems and developing robust defenses will be critical. Techniques like adversarial training and anomaly detection could bolster the system's resilience against malicious attempts to bypass authentication.
3. **Multimodal Biometrics Integration:**
   * Integrating signature verification with other biometric modalities, such as fingerprint or iris recognition, could enhance overall security and reliability. Multimodal approaches could mitigate individual modality limitations and provide more comprehensive user authentication solutions.
4. **Continuous Learning and Adaptation:**
   * Implementing mechanisms for continuous learning and adaptation will be essential to handle evolving signature styles and emerging forgery techniques. Online learning frameworks and adaptive models could ensure the system remains effective and up-to-date over time.
5. **Usability and Accessibility Improvements:**
   * Enhancing the usability of signature verification systems through intuitive interfaces and user-friendly experiences will be crucial. Accessibility considerations, such as accommodating users with varying writing abilities or disabilities, should also be integrated into future developments.
6. **Application Diversification:**
   * Expanding the application scope of signature verification beyond traditional authentication, such as in digital transactions, document verification, and forensic analysis, presents new challenges and opportunities for innovation.

**CHAPTER 9**

**REFERENCES**

**Research Papers:**

L. Yampolskiy and V. Govindaraju, "Handwritten Signature Verification: A Literature Review," *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, vol. 42, no. 6, pp. 1164-1177, 2012.

A. Jain, A. Ross, and S. Prabhakar, "An Introduction to Biometric Recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 1, pp. 4-20, 2004.

**Books:**

R. Plamondon and S. N. Srihari, *Online and Offline Handwriting Recognition: A Comprehensive Survey*, World Scientific Publishing, 2017.

D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar, *Handbook of Fingerprint Recognition*, Springer, 2009.

**Conference Proceedings:**

Proceedings of the International Conference on Biometrics (ICB).

Proceedings of the International Conference on Document Analysis and Recognition (ICDAR).

**Online Resources:**

National Institute of Standards and Technology (NIST) Special Database 19: Handprinted Forms and Characters.

Kaggle datasets on signature verification and biometric recognition.

**Journals:**

*Pattern Recognition*

*IEEE Transactions on Biometrics, Behavior, and Identity Science*

**Standards and Reports:**

ISO/IEC 19794-7:2014 - Information technology - Biometric data interchange formats - Part 7: Signature/sign time series data.