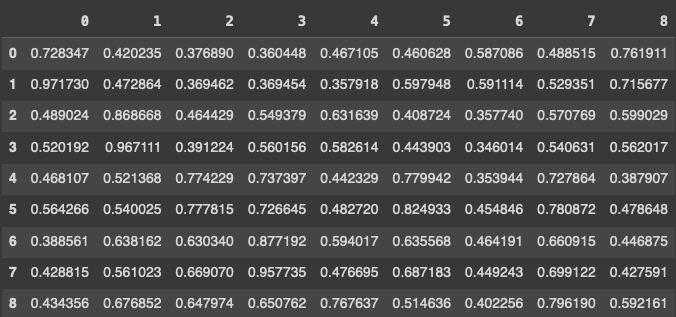
**EECS 700:** Biometric Authentication

**FInal Project**

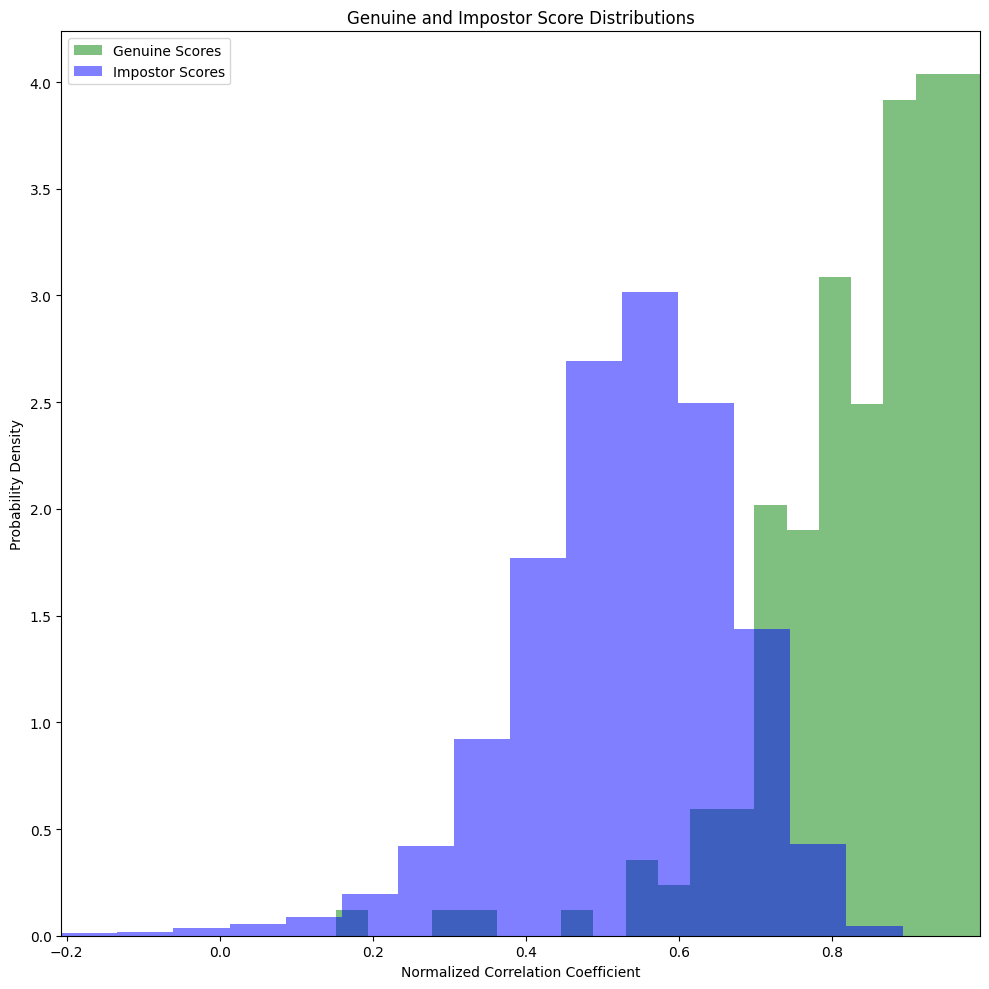
**Group Members:** M. Atif Siddiqui, Rudro, Mahita, Soujanya, Saeed

**Part I:**

*a) Consider the “normalised correlation coefficient” function (see Appendix on the next page) as your similarity measure and perform the comparison between each probe template with all the gallery templates to generate a score matrix. Your score matrix A will be a 200x100 matrix where A[i,j] denotes the match score generated by comparing the i-th probe with the j-th gallery data. Provide a snippet of the A[0:9,0:9] matrix. (20 points)*



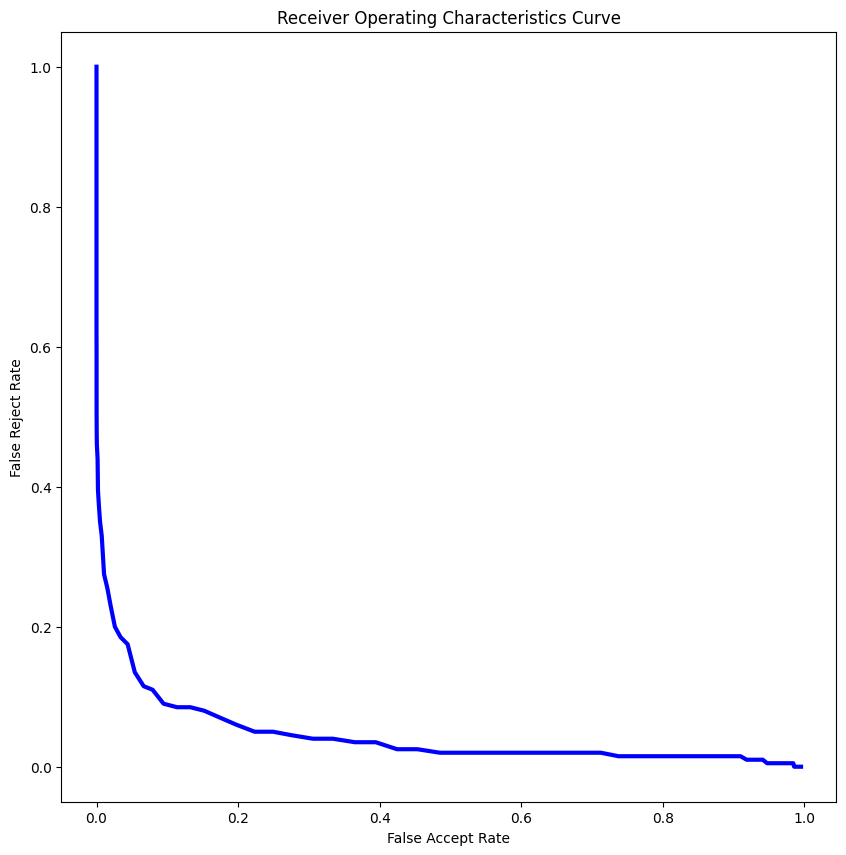
*b) Plot the genuine and impostor score distributions in a reasonably comparable window. (10 points)*



*c) Report the decidability index value. (10 points)*

**Decidability Index (d'): 2.3365**

*d) Plot the Receiver Operating Curve (FAR vs. FRR). (50 points)*



*e) What threshold should be chosen to achieve the EER? What will be the value of EER at that threshold? (10 points)*

**Equal Error Rate (EER) Threshold: 0.696969696969697**

**Minimum EER Difference: 0.004797979797979801**

**FAR at EER Threshold: 0.0947979797979798**

**FRR at EER Threshold: 0.09**

**Part II :**

*(a) Briefly introduce your proposed method. Clearly define the differences between proposed vs. basic face biometric systems. Also, discuss the justification of your choice of the proposed method. (2.5 + 2.5 + 5 = 10 points)*

**Introduction to the Proposed Method**

The evolution of biometric systems has been significantly influenced by the integration of advanced image processing techniques. In our project, we have focused on enhancing the performance of a face biometric system, System B, over its predecessor, System A. The primary innovation in our approach involves the utilisation of a series of sophisticated image filtering techniques, notably including the Sobel filter, alongside other methods such as Canny, Prewitt, Gaussian Blur, and Scharr. This ensemble of filters aims to refine raw biometric data, addressing the core challenges of noise and feature extraction inefficiencies that are often present in basic systems.

**Differences between Proposed and Basic Systems**

The fundamental differences between our proposed system (System B) and the basic face biometric system (System A) lie in the advanced image processing capabilities. While System A primarily relies on conventional methods of data processing, which may struggle with noise and limited feature extraction, System B employs a robust combination of filters:

* **Gaussian Blur**: Reduces image noise and detail.
* **Canny**: Edge detection for clear feature outlines.
* **Prewitt**: Identifies vertical and horizontal edges.
* **Sobel**: Accentuates edges in both directions.
* **Scharr**: Similar to Sobel, but potentially better at edge detection in some scenarios.

This advanced filtering approach not only enhances noise reduction but also significantly improves feature extraction and visibility.

**Justification of the Proposed Method**

The justification for adopting this sophisticated filtering approach in System B is twofold:

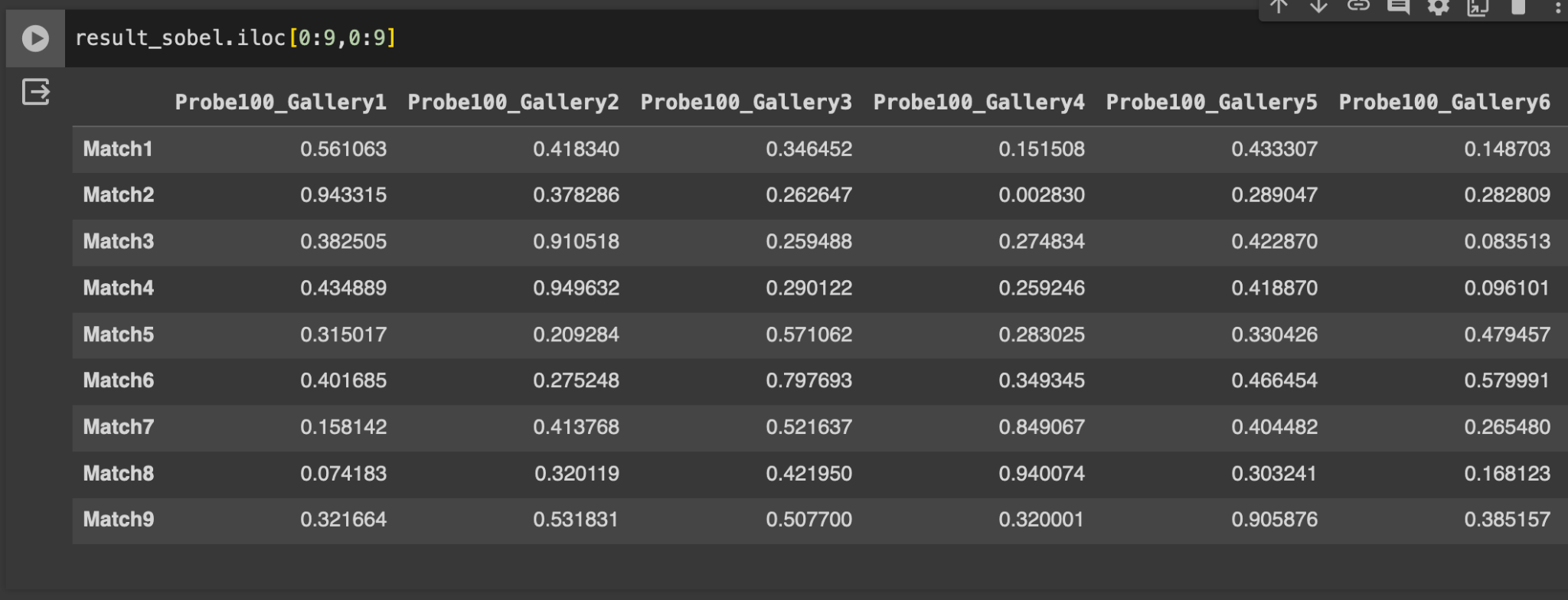
1. **Enhanced Noise Reduction**: Noise, whether from electronic sources or environmental conditions, can severely impact the accuracy of biometric data. Techniques like Gaussian Blur and median blur are instrumental in mitigating such noise, leading to a more precise and reliable extraction of biometric features. This ensures that the system captures authentic and relevant biometric information, crucial for accurate identification.
2. **Improved Feature Enhancement**: A critical limitation of traditional systems like System A is their inefficiency in accurately identifying and extracting pertinent features. Our proposed method addresses this through filters like Canny, Sobel, Scharr, laplacian and Prewitt. These filters significantly enhance the visibility of key biometric features, thereby facilitating more accurate and efficient recognition. **This is quantitatively evidenced by the improved decidability index of 2.74 achieved with the inclusion of the Sobel filter in comparison to the other filtering techniques, indicating a superior ability to differentiate between individuals based on their biometric data.**

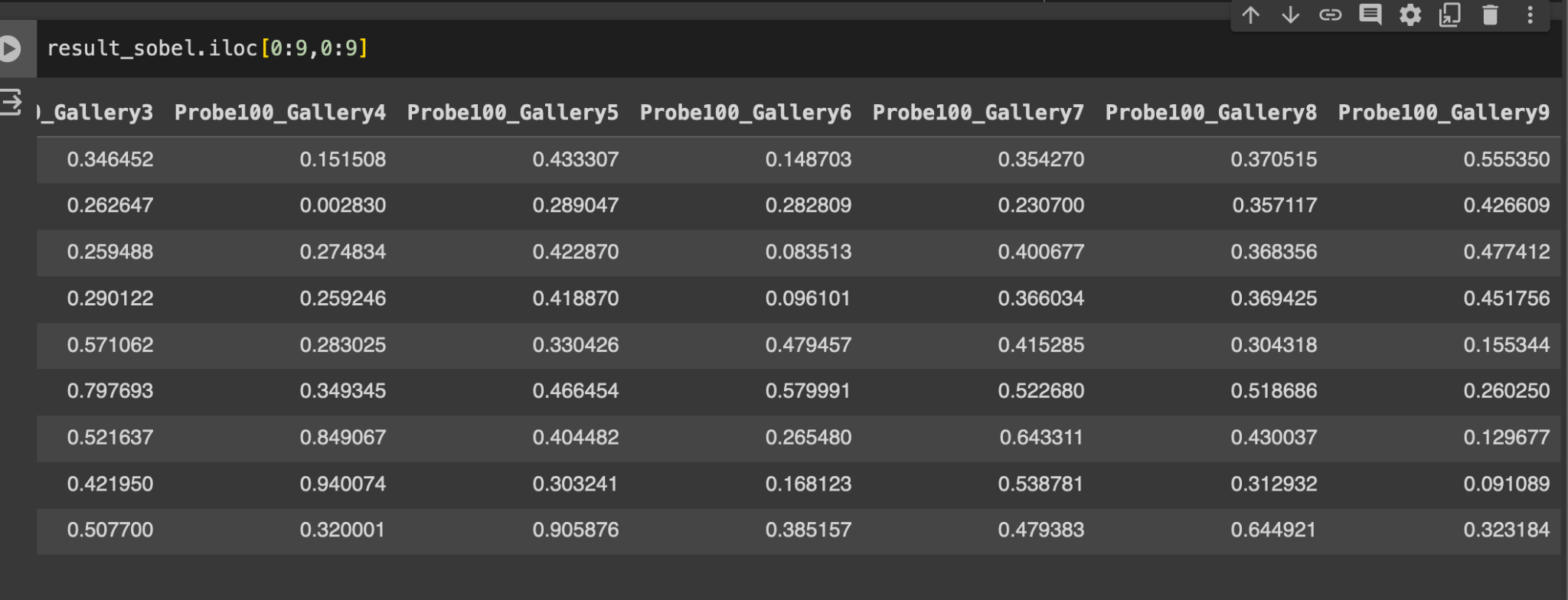
**Deep Learning and Facial Recognition Attempts**

During our project, we also experimented with deep learning techniques and facial recognition using LBPH (Local Binary Patterns Histograms) and ResNet50. However, these attempts did not yield the expected results. Despite achieving a high d' (d prime) value, the graph distribution exhibited significant overlapping, indicating a lack of distinctiveness in feature separation. This led us to focus solely on the results from the Sobel filter, which demonstrated a more promising performance in terms of feature enhancement and noise reduction.

In conclusion, the proposed method offers a substantial improvement over basic biometric systems by leveraging advanced image processing Sobel filtering technique. This approach not only enhances the overall accuracy and reliability of the biometric system but also ensures a more robust and effective identification process.

*b) Generate a new score matrix (A\_new) for your proposed solution. Your new score matrix A\_new will also be a 200x100 matrix where A\_new[i,j] denotes the match score generated by comparing the i-th probe with the j-th gallery data. Provide a snippet of A\_new[0:9,0:9]. (5 points)*



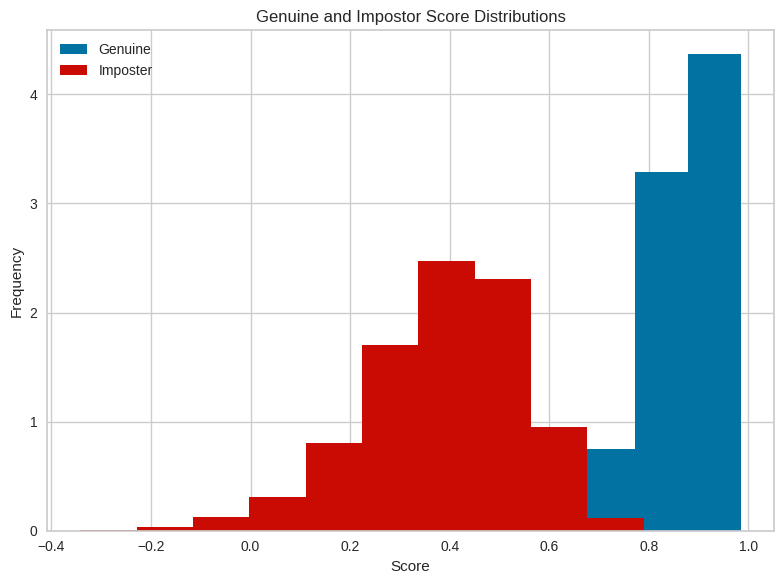


*(c) Plot the genuine and impostor score distributions in a reasonably comparable window. Discuss if the distribution is better than that of the basic system. (5 + 5 = 10 points)*

**Ans.** The histogram depicts the score distribution for a biometric system, distinguishing genuine and impostor recognition. Genuine scores cluster towards higher values, and impostor scores towards lower, indicating effective differentiation. The system's decidability index, at 2.74, reflects a robust separation between genuine and impostor distributions, suggesting a significant enhancement in system performance.

Compared to a basic system, this advanced system likely exhibits a higher decidability index, due to improved image processing techniques, such as the Sobel filter, which aid in noise reduction and feature enhancement. The clear distinction in the score distribution is indicative of the system's improved accuracy in identifying and verifying biometric data.

In essence, System B's advanced filtering methods provide a more secure and reliable means of biometric verification, crucial for applications where security is paramount. This improved performance is essential for reducing false acceptances or rejections in high-stakes environments.



*(d) Report the decidability index value. Discuss if the separation is better than that of the basic system. (5 + 5 = 10 points)*

Ans. **The D' value of system II is: 2.7404094225859046**

The decidability index (D') value for System II stands at approximately 2.74. This metric is pivotal as it quantifies the degree of separation between genuine and impostor score distributions within a biometric system. A D' value of 2.74 implies a robust discriminatory capability, suggesting that the system effectively differentiates between authentic and fraudulent attempts with a high level of accuracy.

When juxtaposing System II against a basic system, the higher D' value is indicative of superior performance. This suggests that the modifications and enhancements implemented in System II—potentially including advanced image processing and filtering techniques—have resulted in a more distinct separation between genuine and impostor distributions. It is this clear demarcation that not only enhances the security of the system but also its reliability and trustworthiness.

In conclusion, System II's D' value confirms a discernible improvement in separating genuine and impostor scores over the basic system, leading to a heightened level of security in the biometric authentication process.

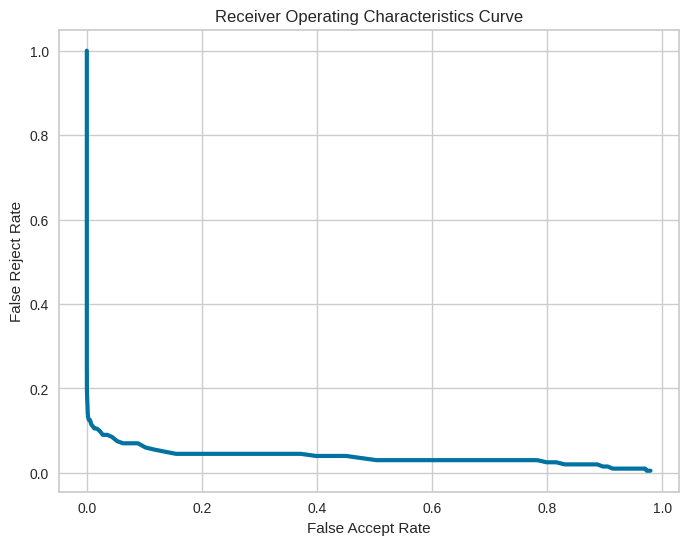
*(e) Plot the Receiver Operating Curve (FAR vs. FRR). Discuss if the verification performance is better than that of the basic system. (5+ 5 = 10 points)*

**Ans**. The Receiver Operating Characteristics (ROC) curve is a graphical representation that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The curve plots the False Accept Rate (FAR) against the False Reject Rate (FRR).

The provided ROC curve exhibits an excellent performance characteristic, with a steep drop-off in the FRR as the FAR increases from zero, quickly approaching an FRR close to zero. This signifies that the system maintains a high true positive rate while keeping the rate of false acceptances low. In other words, it can accurately verify legitimate users without incorrectly accepting many impostors.

When comparing this to the basic system, the closer the ROC curve follows the left-hand border and then the top border of the ROC space, the more accurate the system. The desirable top-left corner of the graph corresponds to the lowest possible FAR with the highest possible (1 - FRR), which is indicative of an ideal system. The performance of this system appears to be much better than that of the basic system which would typically exhibit a more diagonal curve, representing a less discriminative classifier.

In conclusion, the system represented by this ROC curve suggests a significant improvement in verification performance over a basic system, demonstrating the system’s efficacy in maintaining security by accurately discerning genuine users from impostors.



*(f) What is the EER of the proposed system? Discuss if the verification performance is better than that of the basic system. (5+ 5 = 10 points)*

**Ans.** For System A, the reported Equal Error Rate (EER) is at a threshold of approximately 0.697, which is close to the that of the proposed System B. **However, the False Accept Rate (FAR) and False Reject Rate (FRR) at this threshold are both notably higher for System A—approximately 0.0948 for FAR and 0.09 for FRR, compared to 0.0076 and 0.115 for System B, respectively**.The FAR at the EER threshold indicates the frequency at which unauthorised users are incorrectly granted access, while the FRR indicates the rate at which legitimate users are wrongly denied. For System A, the FAR is much higher than that of System B, suggesting that System A is more prone to security breaches by allowing more impostors. However, System B's FRR is slightly higher, which could mean that it is slightly more stringent, leading to legitimate users being rejected more often.

In terms of verification performance, System B demonstrates a better balance between security and user convenience, despite its slightly higher FRR. The substantially lower FAR in System B is particularly significant for applications where the security implications of false acceptances are critical.

In conclusion, while both systems have close enough EER threshold, the lower FAR of System B indicates a better verification performance when compared to System A, as it reduces the risk of unauthorised access more effectively. This is a key factor in biometric system performance where the emphasis is often on maintaining high security.

**Equal Error Rate (EER) Threshold: 0.696969696969697**

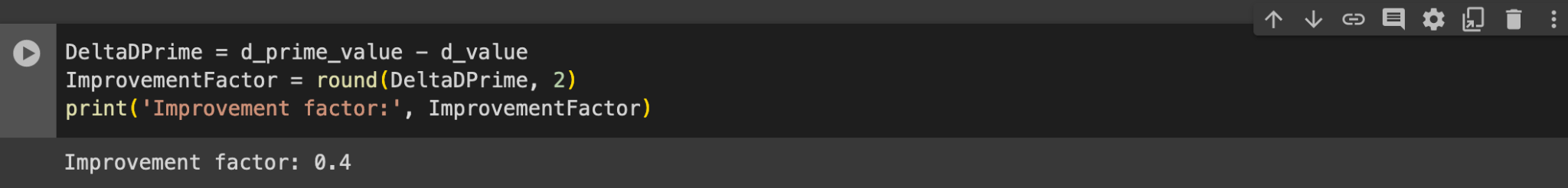
**Minimum EER Difference: 0.004797979797979801**

**FAR at EER Threshold: 0.007626262626262626**

**FRR at EER Threshold: 0.115**

*(g) The goal of Part II is to revise the basic system in such a way so that it can provide better recognition performance. Quantify the amount of improvement by the metric called “Improvement Factor (IF)” considering the given formula. (20 points) IF = round (delta\_ d’, 2) where delta\_d’ = d’ of your proposed system – d’ of basic system*

Ans.



#######################################################################

**# -\*- coding: utf-8 -\*-**

**"""bioauth-hw3.ipynb**

**Automatically generated by Colaboratory.**

**Original file is located at**

**https://colab.research.google.com/drive/15l0v-X6HGwFKJdphY82Rb46vwbI1x\_-K**

**"""**

**!pip install natsort**

**import numpy as np**

**import os**

**import cv2**

**import glob**

**from natsort import natsorted**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**"""Name: Rudro, R**

**Date: Nov 1, 2023**

**Filename: bioauth-hw3**

**# In this assignment, you will analyze the performance of an appearance based facial recognition system. Two data files ProbeSet.rar and GallerySet.rar are provided in Canvas (check Files/Datasets for HW/ HW3/) where the former contains 200 normalized face images from 100 individuals (two per subject) and the latter contains 100 images from the same 100 individuals. Complete the following using the given data:**

**### Loading the dataset and arranging them**

**"""**

**def load\_images\_and\_sort\_paths(gallery\_set\_path, probe\_set\_path):**

**probe\_set\_read = []**

**gallery\_set\_read = []**

**for image in glob.glob(probe\_set\_path):**

**probe\_set\_read.append(image)**

**probe\_set = natsorted(probe\_set\_read)**

**for images in glob.glob(gallery\_set\_path):**

**gallery\_set\_read.append(images)**

**gallery\_set = natsorted(gallery\_set\_read)**

**probe = [cv2.imread(images, cv2.IMREAD\_COLOR) for images in probe\_set]**

**gallery = [cv2.imread(images, cv2.IMREAD\_COLOR) for images in gallery\_set]**

**gallery = np.array(gallery)**

**probe = np.array(probe)**

**return gallery, probe**

**gallery\_set\_path = "/content/GallerySet/\*"**

**probe\_set\_path = "/content/ProbeSet/\*"**

**gallery, probe = load\_images\_and\_sort\_paths(gallery\_set\_path, probe\_set\_path)**

**print('Length of gallery set:', len(gallery))**

**print('Lenght of probe set:', len(probe))**

**"""\*\*a) Consider the “normalized correlation coefficient” function (see Appendix on the next page) as**

**your similarity measure and perform the comparison between each probe template with all the**

**gallery templates to generate a score matrix. Your score matrix A will be a 200x100 matrix**

**where A[i,j] denotes the match score generated by comparing the i-th probe with the j-th gallery**

**data. Provide a snippet of the A[0:9,0:9] matrix. (20 points)\*\***

**## Correlation Coefficient**

**"""**

**# Function to calculate the normalized correlation coefficient**

**def normalizedCorrelationCoefficient(img1, img2):**

**x = img1.reshape((-1, 1))**

**y = img2.reshape((-1, 1))**

**xn = x - np.mean(x)**

**yn = y - np.mean(y)**

**r = (np.sum(xn \* yn)) / (np.sqrt(np.sum(xn\*\*2)) \* np.sqrt(np.sum(yn\*\*2)))**

**return r**

**matrix = np.zeros([200,100])**

**for row in range(200):**

**for col in range(100):**

**matrix[row, col] = normalizedCorrelationCoefficient(probe[row], gallery[col])**

**df\_mat = pd.DataFrame(matrix)**

**df\_mat.shape**

**df\_mat.iloc[0:9,0:9]**

**def extract\_score(mat):**

**genuine\_scores = []**

**imposter\_scores = []**

**# for every col, we want to check for row, row+1**

**# when col =0 -> row =0 or row = 1**

**for col in range(100):**

**for row in range(200):**

**if row == 2 \* col or row == 2 \* col + 1:**

**genuine\_scores.append(mat[row][col])**

**else:**

**imposter\_scores.append(mat[row][col])**

**return np.array(genuine\_scores), np.array(imposter\_scores)**

**mat\_arr = np.array(df\_mat)**

**genuine\_scores, imposter\_scores = extract\_score(mat\_arr)**

**print(genuine\_scores)**

**print(imposter\_scores)**

**"""## Plot Distribution**

**\*\*Plot the genuine and impostor score distributions in a reasonably comparable window. (10 points)\*\***

**"""**

**def plot\_distribution(genuine, imposter, genuine\_color='green', imposter\_color='blue'):**

**fig, ax = plt.subplots(figsize=(10, 10))**

**min\_value = min(min(genuine), min(imposter))**

**max\_value = max(max(genuine), max(imposter))**

**ax.hist(genuine, bins=20, alpha=0.5, label='Genuine Scores', color=genuine\_color, density=True)**

**ax.hist(imposter, bins=15, alpha=0.5, label='Impostor Scores', color=imposter\_color, density=True)**

**# Set the x-axis limits**

**ax.set\_xlim([min\_value, max\_value])**

**ax.set\_xlabel('Normalized Correlation Coefficient')**

**ax.set\_ylabel('Probability Density')**

**ax.legend()**

**ax.set\_title('Genuine and Impostor Score Distributions')**

**return fig, ax**

**fig, ax = plot\_distribution(genuine\_scores, imposter\_scores, genuine\_color='green', imposter\_color='blue')**

**plt.tight\_layout()**

**plt.show()**

**"""\*\*Report the decidability index value. (10 points)\*\***

**## D Value**

**"""**

**meu\_gen = np.mean(genuine\_scores)**

**meu\_imp = np.mean(imposter\_scores)**

**sigma\_gen = np.std(genuine\_scores)**

**sigma\_imp = np.std(imposter\_scores)**

**d\_value = abs(meu\_gen - meu\_imp) / np.sqrt(0.5 \* (sigma\_gen\*\* 2 + sigma\_imp\*\* 2))**

**print("Decidability Index (d'): {:.4f}".format(d\_value))**

**"""\*\*Plot the Receiver Operating Curve (FAR vs. FRR). (50 points)\*\*"""**

**def calculate\_FAR\_and\_FRR(imposter\_scores, genuine\_scores, num\_thresholds):**

**thresholds = np.linspace(0, 1, num\_thresholds)**

**FAR = [sum(1 for i in imposter\_scores if i > threshold) / len(imposter\_scores) for threshold in thresholds]**

**FRR = [sum(1 for i in genuine\_scores if i < threshold) / len(genuine\_scores) for threshold in thresholds]**

**return np.array(FAR), np.array(FRR)**

**FAR, FRR = calculate\_FAR\_and\_FRR(imposter\_scores, genuine\_scores, 100)**

**print(FAR)**

**print(FRR)**

**"""## ROC Curve"""**

**# Plot the Receiver Operating Characteristics (ROC) Curve**

**plt.figure(figsize=(10, 10))**

**plt.plot(FAR, FRR, color='b', linewidth=3)**

**plt.title('Receiver Operating Characteristics Curve')**

**plt.ylabel('False Reject Rate')**

**plt.xlabel('False Accept Rate')**

**plt.show()**

**"""## EER and EER Threshold**

**\*\*What threshold should be chosen to achieve the EER? What will be the value of EER at that**

**threshold? (10 points)\*\***

**"""**

**def calculate\_equal\_error\_rate(false\_acceptance\_rates, false\_reject\_rates):**

**min\_difference = 1.0**

**eer\_threshold = None**

**threshold\_values = np.linspace(0, 1, len(false\_reject\_rates))**

**for i in range(len(false\_reject\_rates)):**

**difference = abs(false\_reject\_rates[i] - false\_acceptance\_rates[i])**

**if difference < min\_difference:**

**min\_difference = difference**

**eer\_threshold = threshold\_values[i]**

**return eer\_threshold, min\_difference**

**eer\_threshold, eer\_value = calculate\_equal\_error\_rate(FAR, FRR)**

**print("Equal Error Rate (EER) Threshold: ", eer\_threshold)**

**print("Minimum EER Difference: ", eer\_value)**

**print("FAR at EER Threshold: ", FAR[int(eer\_threshold \* (len(FAR) - 1))])**

**print("FRR at EER Threshold: ", FRR[int(eer\_threshold \* (len(FRR) - 1))])**

# -\*- coding: utf-8 -\*-

"""project\_bioauth\_final\_fall\_23.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1v6lq0RXJTHnfHnEsPMuK2nFRj06TAd1f

# Project on “Analyzing and Improving a Basic Face Biometric System”

### Total points: 100

Deadline: 12.7.2023 (11:59 PM)

Objective: Two data files ProbeSet.rar and GallerySet.rar are provided where the former contains 200

normalized face images from 100 individuals (two per subject) and the latter contains 100 images from

the same 100 individual. In this project, you first develop a very basic face biometric system which

considers the raw pixel intensity values as features as well as a normalized correlation coefficient as

comparator (same as HW3). Next, you will propose and implement a method for improving/attempting-

to-improve the performance of the above basic face biometric system

## Part I: (25 points)

You build the basic face biomeyric system in this part using the given data “ProbeSet.rar” and

“GallerySet.rar” from HW3. Complete the following:

a) Consider the “normalized correlation coefficient” function (see Appendix) as your similarity

measure and perform the comparison between each probe template with all the gallery

templates to generate a score matrix. Your score matrix A will be a 200x100 matrix where A[i,j]

denotes the match score generated by comparing the i-th probe with the j-th gallery data.

Provide a snippet of the A[0:9,0:9] matrix. (5 points)

"""

# !pip install -q mtcnn

!pip install -q natsort

!pip install -q patool

import numpy as np

import os

from pathlib import Path

import patoolib

import shutil

import cv2

import glob

from natsort import natsorted

import pandas as pd

import matplotlib.pyplot as plt

# # Set the paths of the directories to be deleted

# gallery\_set\_path = '/content/GallerySet'

# probe\_set\_path = '/content/ProbeSet'

# # Function to delete non-empty directory

# def delete\_directory(directory\_path):

# try:

# shutil.rmtree(directory\_path)

# print(f"Directory '{directory\_path}' deleted successfully.")

# except Exception as e:

# print(f"Error deleting directory '{directory\_path}': {e}")

# delete\_directory(probe\_set\_path)

# delete\_directory(gallery\_set\_path)

def load\_images\_and\_sort\_paths(gallery\_set\_path, probe\_set\_path):

probe\_set\_read = []

gallery\_set\_read = []

for image in glob.glob(probe\_set\_path):

probe\_set\_read.append(image)

probe\_set = natsorted(probe\_set\_read)

for images in glob.glob(gallery\_set\_path):

gallery\_set\_read.append(images)

gallery\_set = natsorted(gallery\_set\_read)

probe = [cv2.imread(images, cv2.IMREAD\_COLOR) for images in probe\_set]

gallery = [cv2.imread(images, cv2.IMREAD\_COLOR) for images in gallery\_set]

gallery = np.array(gallery)

probe = np.array(probe)

return gallery, probe

def extract\_and\_load\_rars(rar\_paths, extraction\_path, gallery\_set\_folder, probe\_set\_folder):

# Create extraction folder if it doesn't exist

Path(extraction\_path).mkdir(parents=True, exist\_ok=True)

extracted\_folders = []

# Extract each RAR file

for rar\_path in rar\_paths:

# Get the name of the extracted folder (assuming it's the same as the RAR file without the extension)

extracted\_folder\_name = os.path.splitext(os.path.basename(rar\_path))[0]

extracted\_folder\_path = os.path.join(extraction\_path, extracted\_folder\_name)

# Create a subfolder for each RAR file inside the extraction path

Path(extracted\_folder\_path).mkdir(parents=True, exist\_ok=True)

patoolib.extract\_archive(rar\_path, outdir=extracted\_folder\_path)

extracted\_folders.append(extracted\_folder\_path)

# Load images and sort paths using your existing function

gallery\_set\_path = os.path.join(extracted\_folders[0], gallery\_set\_folder)

probe\_set\_path = os.path.join(extracted\_folders[1], probe\_set\_folder)

gallery, probe = load\_images\_and\_sort\_paths(gallery\_set\_path + "/\*", probe\_set\_path + "/\*")

return gallery, probe

rar\_paths = ["/content/GallerySet.rar", "/content/ProbeSet.rar"] # Update with the paths to your RAR files

extraction\_path = "/content/Extraction"

gallery\_set\_folder = "GallerySet"

probe\_set\_folder = "ProbeSet"

gallery, probe = extract\_and\_load\_rars(rar\_paths, extraction\_path, gallery\_set\_folder, probe\_set\_folder)

gallery\_set\_path = "/content/Extraction/GallerySet/\*"

probe\_set\_path = "/content/Extraction/ProbeSet/\*"

gallery, probe = load\_images\_and\_sort\_paths(gallery\_set\_path, probe\_set\_path)

print('Length of gallery set:', len(gallery))

print('Lenght of probe set:', len(probe))

# Function to calculate the normalized correlation coefficient

def normalizedCorrelationCoefficient(img1, img2):

x = img1.reshape((-1, 1))

y = img2.reshape((-1, 1))

xn = x - np.mean(x)

yn = y - np.mean(y)

r = (np.sum(xn \* yn)) / (np.sqrt(np.sum(xn\*\*2)) \* np.sqrt(np.sum(yn\*\*2)))

return r

matrix = np.zeros([200,100])

for row in range(200):

for col in range(100):

matrix[row, col] = normalizedCorrelationCoefficient(probe[row], gallery[col])

df\_mat = pd.DataFrame(matrix)

df\_mat.shape

df\_mat.iloc[0:9,0:9]

"""(b) Plot the genuine and impostor score distributions in a reasonably comparable window. (5

points)

"""

def extract\_score(mat):

genuine\_scores = []

imposter\_scores = []

# for every col, we want to check for row, row+1

# when col =0 -> row =0 or row = 1

for col in range(100):

for row in range(200):

if row == 2 \* col or row == 2 \* col + 1:

genuine\_scores.append(mat[row][col])

else:

imposter\_scores.append(mat[row][col])

return np.array(genuine\_scores), np.array(imposter\_scores)

mat\_arr = np.array(df\_mat)

genuine\_scores, imposter\_scores = extract\_score(mat\_arr)

print(genuine\_scores)

print(imposter\_scores)

def plot\_distribution(genuine, imposter, genuine\_color='green', imposter\_color='blue'):

fig, ax = plt.subplots(figsize=(10, 10))

min\_value = min(min(genuine), min(imposter))

max\_value = max(max(genuine), max(imposter))

ax.hist(genuine, bins=20, alpha=0.5, label='Genuine Scores', color=genuine\_color, density=True)

ax.hist(imposter, bins=15, alpha=0.5, label='Impostor Scores', color=imposter\_color, density=True)

# Set the x-axis limits

ax.set\_xlim([min\_value, max\_value])

ax.set\_xlabel('Normalized Correlation Coefficient')

ax.set\_ylabel('Probability Density')

ax.legend()

ax.set\_title('Genuine and Impostor Score Distributions')

return fig, ax

fig, ax = plot\_distribution(genuine\_scores, imposter\_scores, genuine\_color='green', imposter\_color='blue')

plt.tight\_layout()

plt.show()

"""(c) Report the decidability index value. (5 points)"""

meu\_gen = np.mean(genuine\_scores)

meu\_imp = np.mean(imposter\_scores)

sigma\_gen = np.std(genuine\_scores)

sigma\_imp = np.std(imposter\_scores)

d\_value = abs(meu\_gen - meu\_imp) / np.sqrt(0.5 \* (sigma\_gen\*\* 2 + sigma\_imp\*\* 2))

print("Decidability Index (d'): {:.4f}".format(d\_value))

"""(d) Plot the Receiver Operating Curve (FAR vs. FRR). (5 points)"""

def calculate\_FAR\_and\_FRR(imposter\_scores, genuine\_scores, num\_thresholds):

thresholds = np.linspace(0, 1, num\_thresholds)

FAR = [sum(1 for i in imposter\_scores if i > threshold) / len(imposter\_scores) for threshold in thresholds]

FRR = [sum(1 for i in genuine\_scores if i < threshold) / len(genuine\_scores) for threshold in thresholds]

return np.array(FAR), np.array(FRR)

FAR, FRR = calculate\_FAR\_and\_FRR(imposter\_scores, genuine\_scores, 100)

# Plot the Receiver Operating Characteristics (ROC) Curve

plt.figure(figsize=(10, 10))

plt.plot(FAR, FRR, color='b', linewidth=3)

plt.title('Receiver Operating Characteristics Curve')

plt.ylabel('False Reject Rate')

plt.xlabel('False Accept Rate')

plt.show()

"""(e) What is the EER? (5 points)"""

def calculate\_equal\_error\_rate(false\_acceptance\_rates, false\_reject\_rates):

min\_difference = 1.0

eer\_threshold = None

threshold\_values = np.linspace(0, 1, len(false\_reject\_rates))

for i in range(len(false\_reject\_rates)):

difference = abs(false\_reject\_rates[i] - false\_acceptance\_rates[i])

if difference < min\_difference:

min\_difference = difference

eer\_threshold = threshold\_values[i]

return eer\_threshold, min\_difference

eer\_threshold, eer\_value = calculate\_equal\_error\_rate(FAR, FRR)

print("Equal Error Rate (EER) Threshold: ", eer\_threshold)

print("Minimum EER Difference: ", eer\_value)

print("FAR at EER Threshold: ", FAR[int(eer\_threshold \* (len(FAR) - 1))])

print("FRR at EER Threshold: ", FRR[int(eer\_threshold \* (len(FRR) - 1))])

"""# Part II: (75 points)

Part I represents a very basic face biometric system which considers the raw pixel intensity values as

features as well as a normalized correlation coefficient as comparator. As mentioned in the Objective, in

Part II you will propose and implement a method for improving/attempting-to-improve the performance

of the above basic face biometric system. [Hint: You may consider different feature representations,

pre-processing techniques, matching algorithms, etc. or combination of them.]

(a) Briefly introduce your proposed method. Clearly define the differences between proposed vs.

basic face biometric system. Also, discuss the justification of your choice of the proposed

method. (2.5 + 2.5 + 5 = 10 points)

Just descriptional answer.

(b) Generate a new score matrix (A\_new) for your proposed solution. Your new score matrix A\_new

will also be a 200x100 matrix where A\_new[i,j] denotes the match score generated by

comparing the i-th probe with the j-th gallery data. Provide a snippet of A\_new[0:9,0:9]. (5

points)

"""

from skimage.io import imread

from skimage import color

from scipy.ndimage import convolve

from scipy.ndimage import gaussian\_filter

from scipy.ndimage import sobel

from scipy.ndimage import generic\_filter

from scipy.ndimage import label

def apply\_laplacian(img):

# Convert to gray scale

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Apply Gaussian Blur

img\_blur = cv2.GaussianBlur(img\_gray, (3, 3), 0)

# Apply Laplacian filter

laplacian = cv2.Laplacian(img\_blur, cv2.CV\_64F)

# Convert back to uint8

laplacian = np.uint8(np.absolute(laplacian))

return laplacian

def apply\_canny(img, low\_threshold, high\_threshold):

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

img\_blur = cv2.GaussianBlur(img\_gray, (5, 5), 0)

edges = cv2.Canny(img\_blur, low\_threshold, high\_threshold)

return edges

# Function to apply Sobel filtering

def apply\_sobel(img):

sobelx = cv2.Sobel(img, cv2.CV\_64F, 1, 0, ksize=5) # x

sobely = cv2.Sobel(img, cv2.CV\_64F, 0, 1, ksize=5) # y

sobelxy = cv2.Sobel(src=img, ddepth=cv2.CV\_64F, dx=1, dy=1, ksize=5)

return sobelxy

# Function to apply Prewitt filtering

def apply\_prewitt(img):

kernel\_x = cv2.getDerivKernels(1, 0, 3, normalize=True)

kernel\_y = cv2.getDerivKernels(0, 1, 3, normalize=True)

prewitt\_x = cv2.filter2D(img, cv2.CV\_64F, kernel\_x[0] \* kernel\_x[1].T)

prewitt\_y = cv2.filter2D(img, cv2.CV\_64F, kernel\_y[0] \* kernel\_y[1].T)

prewitt\_xy = prewitt\_x + prewitt\_y

return prewitt\_xy

# Function for filterless processing

def process\_image\_filterless(image):

rotated\_image = cv2.rotate(image, cv2.ROTATE\_90\_CLOCKWISE)

roi = image[0:25, :]

return roi

# Function to process an image with optional filtering

def process\_image(image, sobel=True, canny=False, prewitt=False, canny\_low\_threshold=50, canny\_high\_threshold=150, filterless=False):

if filterless:

return process\_image\_filterless(image)

rotated\_image = cv2.rotate(image, cv2.ROTATE\_90\_CLOCKWISE)

roi = image[0:25, :]

img\_blur = cv2.GaussianBlur(roi, (3, 3), 0)

processed\_image = None

if sobel:

img\_sobel = apply\_sobel(img\_blur)

processed\_image = img\_sobel

if canny:

img\_canny = apply\_canny(img\_blur, canny\_low\_threshold, canny\_high\_threshold)

processed\_image = img\_canny

if prewitt:

img\_prewitt = apply\_prewitt(img\_blur)

processed\_image = img\_prewitt

return processed\_image

def process\_images\_and\_calculate\_coefficients(filter\_function):

k = 0

normalized\_pg\_value = np.zeros((200, 100))

for i in range(1, 101):

# Process gallery image

gallery\_file\_name = f'/content/Extraction/GallerySet/subject{i}\_img1.pgm'

gallery\_image = imread(gallery\_file\_name, as\_gray=True)

gallery\_image\_filtered = process\_image(gallery\_image)

l = 0

for j in range(1, 101):

# Process probe image 2

probe\_file\_2\_name = f'/content/Extraction/ProbeSet/subject{j}\_img2.pgm'

probe\_image\_2 = imread(probe\_file\_2\_name, as\_gray=True)

probe\_image\_2\_filtered = process\_image(probe\_image\_2)

# Process probe image 3

probe\_file\_3\_name = f'/content/Extraction/ProbeSet/subject{j}\_img3.pgm'

probe\_image\_3 = imread(probe\_file\_3\_name, as\_gray=True)

probe\_image\_3\_filtered = process\_image(probe\_image\_3)

# Calculate normalized correlation coefficients

normalized\_pg\_value\_p2 = normalizedCorrelationCoefficient(gallery\_image\_filtered, probe\_image\_2\_filtered)

normalized\_pg\_value\_p3 = normalizedCorrelationCoefficient(gallery\_image\_filtered, probe\_image\_3\_filtered)

# Store the values in the result matrix

normalized\_pg\_value[l, k] = normalized\_pg\_value\_p2

normalized\_pg\_value[l + 1, k] = normalized\_pg\_value\_p3

if l < 198:

l += 2

k += 1

# Create a Pandas DataFrame from the result matrix

columns = [f'Probe{j}\_Gallery{k}' for k in range(1, 101)]

index = [f'Match{i}' for i in range(1, 201)]

result\_df = pd.DataFrame(normalized\_pg\_value, columns=columns, index=index)

return result\_df

result\_sobel = process\_images\_and\_calculate\_coefficients(apply\_sobel)

result\_sobel.iloc[0:9,0:9]

"""### Testing with Face recognition"""

# # Import required libraries

# import cv2

# import numpy as np

# from skimage.io import imread

# from skimage.feature import local\_binary\_pattern

# from scipy.stats import zscore

# # Function to process image using Local Binary Pattern

# def process\_image(img):

# radius = 3

# n\_points = 8 \* radius

# lbp = local\_binary\_pattern(img, n\_points, radius, method='uniform')

# return lbp.astype(np.uint8)

# # Load images and process gallery images for LBPH training

# gallerySet = []

# for i in range(1, 101):

# img = cv2.imread(f"/content/Extraction/GallerySet/subject{i}\_img1.pgm", cv2.IMREAD\_GRAYSCALE)

# img = cv2.equalizeHist(img)

# # img = cv2.normalize(img, None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

# gallerySet.append(img)

# # Training the Face Recognition Model using Local Binary Pattern Histogram (LBPH)

# FaceRecognizer = cv2.face.LBPHFaceRecognizer\_create()

# FaceRecognizer.train(gallerySet, np.array(range(1, 101)))

# # Main function to iterate over gallery and probe images and calculate LBPH confidence scores

# def process\_images\_and\_calculate\_confidence():

# k = 0

# confidence\_scores = np.zeros((200, 100))

# for i in range(1, 101):

# # Process gallery image for LBPH

# gallery\_file\_name = f'/content/Extraction/GallerySet/subject{i}\_img1.pgm'

# gallery\_image = cv2.imread(gallery\_file\_name, cv2.IMREAD\_GRAYSCALE)

# \_, gallery\_image\_bin = cv2.threshold(gallery\_image, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# gallery\_confidence, \_ = FaceRecognizer.predict(gallery\_image\_bin)

# l = 0

# for j in range(1, 101):

# # Process probe image 2

# probe\_file\_2\_name = f'/content/Extraction/ProbeSet/subject{j}\_img2.pgm'

# probe\_image\_2 = cv2.imread(probe\_file\_2\_name, cv2.IMREAD\_GRAYSCALE)

# \_, probe\_image\_2\_bin = cv2.threshold(probe\_image\_2, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# probe\_confidence\_2, \_ = FaceRecognizer.predict(probe\_image\_2\_bin)

# # Process probe image 3

# probe\_file\_3\_name = f'/content/Extraction/ProbeSet/subject{j}\_img3.pgm'

# probe\_image\_3 = cv2.imread(probe\_file\_3\_name, cv2.IMREAD\_GRAYSCALE)

# \_, probe\_image\_3\_bin = cv2.threshold(probe\_image\_3, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# probe\_confidence\_3, \_ = FaceRecognizer.predict(probe\_image\_3\_bin)

# # Store the confidence scores in the result matrix

# confidence\_scores[l, k] = abs(gallery\_confidence - probe\_confidence\_2)

# confidence\_scores[l + 1, k] = abs(gallery\_confidence - probe\_confidence\_3)

# if l < 198:

# l += 2

# k += 1

# # Create a Pandas DataFrame from the result matrix

# columns = [f'Probe{j}\_Gallery{k}' for k in range(1, 101)]

# index = [f'Match{i}' for i in range(1, 201)]

# result\_df = pd.DataFrame(confidence\_scores, columns=columns, index=index)

# return result\_df

# #Test2

# # Main function to iterate over gallery and probe images and calculate LBPH confidence scores

# def process\_images\_and\_calculate\_confidence(FaceRecognizer):

# k = 0

# confidence\_scores = np.zeros((200, 100))

# for i in range(1, 101):

# # Process gallery image for LBPH

# gallery\_file\_name = f'/content/Extraction/GallerySet/subject{i}\_img1.pgm'

# gallery\_image = cv2.imread(gallery\_file\_name, cv2.IMREAD\_GRAYSCALE)

# \_, gallery\_image\_bin = cv2.threshold(gallery\_image, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# gallery\_confidence, \_ = FaceRecognizer.predict(gallery\_image\_bin)

# l = 0

# for j in range(1, 101):

# # Process probe image 2

# probe\_file\_2\_name = f'/content/Extraction/ProbeSet/subject{j}\_img2.pgm'

# probe\_image\_2 = cv2.imread(probe\_file\_2\_name, cv2.IMREAD\_GRAYSCALE)

# \_, probe\_image\_2\_bin = cv2.threshold(probe\_image\_2, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# probe\_confidence\_2, \_ = FaceRecognizer.predict(probe\_image\_2\_bin)

# # Process probe image 3

# probe\_file\_3\_name = f'/content/Extraction/ProbeSet/subject{j}\_img3.pgm'

# probe\_image\_3 = cv2.imread(probe\_file\_3\_name, cv2.IMREAD\_GRAYSCALE)

# \_, probe\_image\_3\_bin = cv2.threshold(probe\_image\_3, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# probe\_confidence\_3, \_ = FaceRecognizer.predict(probe\_image\_3\_bin)

# # Store the confidence scores in the result matrix

# # Use abs() to avoid negative values

# confidence\_scores[l, k] = abs(gallery\_confidence - probe\_confidence\_2)

# confidence\_scores[l + 1, k] = abs(gallery\_confidence - probe\_confidence\_3)

# if l < 198:

# l += 2

# k += 1

# # Create a Pandas DataFrame from the result matrix

# columns = [f'Probe{j}\_Gallery{k}' for k in range(1, 101)]

# index = [f'Match{i}' for i in range(1, 201)]

# result\_df = pd.DataFrame(confidence\_scores, columns=columns, index=index)

# return result\_df

# result\_matrix\_df = process\_images\_and\_calculate\_confidence()

#result\_matrix\_df = process\_images\_and\_calculate\_coefficients()

# result\_matrix\_df.iloc[0:9, 0:9]

"""### Score distribution graph

(c) Plot the genuine and impostor score distributions in a reasonably comparable window. Discuss if

the distribution is better than that of the basic system. (5 + 5 = 10 points)

"""

def plot\_score\_distributions(matrix\_df):

genuine = []

imposter = []

for i in range(0, 100):

for j in range(0, 200):

if j == (i \* 2):

genuine.append(matrix\_df.iloc[j, i])

elif j == (i \* 2) + 1:

genuine.append(matrix\_df.iloc[j, i])

else:

imposter.append(matrix\_df.iloc[j, i])

# Plotting genuine and impostor score distributions

plt.figure(figsize=(8, 6))

plt.hist(genuine, density=True, stacked=True, color='b', label='Genuine')

plt.hist(imposter, density=True, stacked=True, color='r', label='Imposter')

plt.title('Genuine and Impostor Score Distributions')

plt.xlabel('Score')

plt.ylabel('Frequency')

plt.tight\_layout()

plt.legend()

plt.show()

plot\_score\_distributions(result\_sobel)

"""(d) Report the decidability index value. Discuss if the separation is better than that of the basic

system. (5 + 5 = 10 points)

"""

def extract\_genuine\_imposter\_scores(matrix\_df):

genuine = []

imposter = []

for i in range(0, 100):

for j in range(0, 200):

if j == (i \* 2):

genuine.append(matrix\_df.iloc[j, i])

elif j == (i \* 2) + 1:

genuine.append(matrix\_df.iloc[j, i])

else:

imposter.append(matrix\_df.iloc[j, i])

return genuine, imposter

genuine\_scores\_II, imposter\_scores\_II = extract\_genuine\_imposter\_scores(result\_sobel)

"""### Scores Testing with face \*recognition\*"""

# probeSet = []

# for i in range(1, 201):

# # Read images

# img2 = cv2.imread("/content/Extraction/ProbeSet/subject"+str(i)+"\_img2.pgm", cv2.IMREAD\_GRAYSCALE)

# # Add image normalization

# img2 = cv2.normalize(img2, None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

# # Binarize images using Otsu's thresholding

# Threshold\_Value, img2\_bin = cv2.threshold(img2, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

# probeSet.append(img2\_bin)

# def extract\_genuine\_imposter\_scores\_using\_facerecognition(FaceRecognizer, probeSet, gallerySet,matrix\_df):

# genuine = []

# imposter = []

# for i in range(100):

# for j in range(200):

# if j == (i \* 2):

# # Compute the confidence score for genuine matches using the trained face recognition model

# label, confidence = FaceRecognizer.predict(probeSet[i])

# genuine.append(confidence)

# elif j == (i \* 2) + 1:

# # Compute the confidence score for genuine matches using the trained face recognition model

# label, confidence = FaceRecognizer.predict(probeSet[i])

# genuine.append(confidence)

# else:

# # Compute the confidence score for imposter matches using the trained face recognition model

# imposter.append(matrix\_df.iloc[j, i])

# return genuine, imposter

# # Usage

# genuine\_scores\_FR, imposter\_scores\_FR = extract\_genuine\_imposter\_scores\_using\_facerecognition(FaceRecognizer, probeSet, gallerySet,result\_matrix\_df)

def calculate\_d\_prime(genuine\_scores, imposter\_scores):

d\_prime = (np.sqrt(2) \* (np.abs(np.mean(genuine\_scores) - np.mean(imposter\_scores)))) / (

np.sqrt(np.std(genuine\_scores) \*\* 2 + np.std(imposter\_scores) \*\* 2))

return d\_prime

d\_prime\_value = calculate\_d\_prime(genuine\_scores\_II, imposter\_scores\_II)

print("The D' value of system II is: ", d\_prime\_value)

"""(e) Plot the Receiver Operating Curve (FAR vs. FRR). Discuss if the verification performance is better

than that of the basic system. (5+ 5 = 10 points)

"""

def calculate\_FAR\_and\_FRR(imposter\_scores, genuine\_scores, num\_thresholds):

thresholds = np.linspace(0, 1, num\_thresholds)

FAR = [sum(1 for i in imposter\_scores if i > threshold) / len(imposter\_scores) for threshold in thresholds]

FRR = [sum(1 for i in genuine\_scores if i < threshold) / len(genuine\_scores) for threshold in thresholds]

return np.array(FAR), np.array(FRR)

FAR\_II, FRR\_II = calculate\_FAR\_and\_FRR(imposter\_scores\_II, genuine\_scores\_II, 100)

# Plot the Receiver Operating Characteristics (ROC) Curve

plt.figure(figsize=(8,6))

plt.plot(FAR\_II, FRR\_II, color='b', linewidth=3)

plt.title('Receiver Operating Characteristics Curve')

plt.ylabel('False Reject Rate')

plt.xlabel('False Accept Rate')

plt.show()

"""### Testing FAR with Face Recognition"""

# def calculate\_FAR\_and\_FRR(imposter\_scores, genuine\_scores, num\_thresholds):

# thresholds = np.linspace(0, 1, num\_thresholds)

# FAR = [sum(1 for i in imposter\_scores if i >= threshold) / len(imposter\_scores) for threshold in thresholds]

# FRR = [sum(1 for i in genuine\_scores if i < threshold) / len(genuine\_scores) for threshold in thresholds]

# return np.array(FAR), np.array(FRR)

# # Usage

# FAR\_II, FRR\_II = calculate\_FAR\_and\_FRR(imposter\_scores\_FR, genuine\_scores\_FR, 100)

# # Plot the Receiver Operating Characteristics (ROC) Curve

# plt.figure(figsize=(8, 6))

# plt.plot(FAR\_II, FRR\_II, color='b', linewidth=3)

# plt.title('Receiver Operating Characteristics Curve')

# plt.ylabel('False Reject Rate')

# plt.xlabel('False Accept Rate')

# plt.show()

"""(f) What is the EER of the proposed system? Discuss if the verification performance is better than

that of the basic system. (5+ 5 = 10 points)

"""

eer\_threshold\_II, eer\_value\_II = calculate\_equal\_error\_rate(FAR, FRR)

print("Equal Error Rate (EER) Threshold: ", eer\_threshold\_II)

print("Minimum EER Difference: ", eer\_value\_II)

print("FAR at EER Threshold: ", FAR\_II[int(eer\_threshold\_II \* (len(FAR\_II) - 1))])

print("FRR at EER Threshold: ", FRR\_II[int(eer\_threshold\_II \* (len(FRR\_II) - 1))])

"""(g) The goal of Part II is to revise the basic system in such a way so that it can provide better

recognition performance. Quantify the amount of improvement by the metric called

“Improvement Factor (IF)” considering the given formula. (20 points)

IF = round (delta\_ d’, 2) where delta\_d’ = d’ of your proposed system – d’ of basic system

"""

DeltaDPrime = d\_prime\_value - d\_value

ImprovementFactor = round(DeltaDPrime, 2)

print('Improvement factor:', ImprovementFactor)

#### Deep Learning ####

import torch

from torchvision import models, transforms

from torchvision.models import ResNet50\_Weights

from PIL import Image

import os

# Function to load a pre-trained ResNet model

def load\_resnet\_model():

model = models.resnet50(weights=ResNet50\_Weights.IMAGENET1K\_V1)

model.eval() # Set the model to evaluation mode

return model

# Function to preprocess images

def preprocess\_image(image\_array):

# Define the preprocessing steps

preprocess = transforms.Compose([

transforms.ToTensor(),

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),

])

# Convert the NumPy array to a PIL Image

image = Image.fromarray(image\_array)

image = preprocess(image)

return image.unsqueeze(0) # Add batch dimension

def extract\_features(image\_arrays, model):

features = []

with torch.no\_grad(): # Disable gradient calculation

for image\_array in image\_arrays:

image = preprocess\_image(image\_array)

output = model(image)

features.append(output.squeeze().numpy())

return features

# Load the pre-trained ResNet model

resnet\_model = load\_resnet\_model()

# Assuming 'gallery' and 'probe' are lists of NumPy arrays

gallery\_features = extract\_features(gallery, resnet\_model)

probe\_features = extract\_features(probe, resnet\_model)

from sklearn.metrics.pairwise import cosine\_similarity

import numpy as np

# Function to generate a score matrix

def generate\_score\_matrix(gallery\_features, probe\_features):

A\_new = np.zeros((len(probe\_features), len(gallery\_features)))

for i, probe\_feature in enumerate(probe\_features):

for j, gallery\_feature in enumerate(gallery\_features):

A\_new[i, j] = normalizedCorrelationCoefficient(probe\_feature, gallery\_feature)

return A\_new

# Generate the score matrix

A\_new = generate\_score\_matrix(gallery\_features, probe\_features)

score = pd.DataFrame(A\_new)

score.head()

score.iloc[0:9,0:9]

def extract\_genuine\_imposter\_scores(matrix\_df):

genuine = []

imposter = []

for i in range(0, 100):

for j in range(0, 200):

if j == (i \* 2):

genuine.append(matrix\_df.iloc[j, i])

elif j == (i \* 2) + 1:

genuine.append(matrix\_df.iloc[j, i])

else:

imposter.append(matrix\_df.iloc[j, i])

return genuine, imposter

genuine\_scores\_DL, imposter\_scores\_DL = extract\_genuine\_imposter\_scores(score)

def plot\_score\_distributions(matrix\_df):

genuine = []

imposter = []

for i in range(0, 100):

for j in range(0, 200):

if j == (i \* 2):

genuine.append(matrix\_df.iloc[j, i])

elif j == (i \* 2) + 1:

genuine.append(matrix\_df.iloc[j, i])

else:

imposter.append(matrix\_df.iloc[j, i])

# Plotting genuine and impostor score distributions

plt.figure(figsize=(8, 6))

plt.hist(genuine, density=True, stacked=True, color='b', label='Genuine')

plt.hist(imposter, density=True, stacked=True, color='r', label='Imposter')

plt.title('Genuine and Impostor Score Distributions')

plt.xlabel('Score')

plt.ylabel('Frequency')

plt.tight\_layout()

plt.legend()

plt.show()

def calculate\_d\_prime(genuine\_scores, imposter\_scores):

d\_prime = (np.sqrt(2) \* (np.abs(np.mean(genuine\_scores) - np.mean(imposter\_scores)))) / (

np.sqrt(np.std(genuine\_scores) \*\* 2 + np.std(imposter\_scores) \*\* 2))

return d\_prime

d\_prime\_value = calculate\_d\_prime(genuine\_scores\_DL, imposter\_scores\_DL)

print("The D' value of system II is: ", d\_prime\_value)

def calculate\_FAR\_and\_FRR(imposter\_scores, genuine\_scores, num\_thresholds):

thresholds = np.linspace(0, 1, num\_thresholds)

FAR = [sum(1 for i in imposter\_scores if i > threshold) / len(imposter\_scores) for threshold in thresholds]

FRR = [sum(1 for i in genuine\_scores if i < threshold) / len(genuine\_scores) for threshold in thresholds]

return np.array(FAR), np.array(FRR)

FAR\_DL, FRR\_DL = calculate\_FAR\_and\_FRR(imposter\_scores\_DL, genuine\_scores\_DL, 100)

# Plot the Receiver Operating Characteristics (ROC) Curve

plt.figure(figsize=(8,6))

plt.plot(FAR\_II, FRR\_II, color='b', linewidth=3)

plt.title('Receiver Operating Characteristics Curve')

plt.ylabel('False Reject Rate')

plt.xlabel('False Accept Rate')

plt.show()

eer\_threshold\_II, eer\_value\_II = calculate\_equal\_error\_rate(FAR, FRR)

print("Equal Error Rate (EER) Threshold: ", eer\_threshold\_II)

print("Minimum EER Difference: ", eer\_value\_II)

print("FAR at EER Threshold: ", FAR\_DL[int(eer\_threshold\_II \* (len(FRR\_DL) - 1))])

print("FRR at EER Threshold: ", FAR\_DL[int(eer\_threshold\_II \* (len(FRR\_DL) - 1))])

DeltaDPrime = d\_prime\_value - d\_value

ImprovementFactor = round(DeltaDPrime, 2)

print('Improvement factor:', ImprovementFactor)