Analyzing a Light-weight Face Recognition System with Further Improvement

Caden Kroonenberg
Department of Electrical Engineering & Computer Science
University of Kansas
Lawrence, KS
cskroonenberg@ku.edu

Michael Talaga
Department of Electrical Engineering & Computer Science
University of Kansas
Lawrence, KS
mktalaga@ku.edu

Oluwanifemi Fadahunsi
Department of Electrical Engineering & Computer Science
University of Kansas
Lawrence, KS
nifemi-fadahunsi@ku.edu

Javante Ewing
Department of Electrical Engineering & Computer Science
University of Kansas
Lawrence, KS
javanteewing@ku.edu

Andrew Riachi
Department of Electrical Engineering & Computer Science
University of Kansas
Lawrence, KS
ariachi@ku.edu

***Abstract*—Facial recognition algorithms take images of faces as input and determine who they belong to. This technology has practical uses, for example, in biometric authentication. In this paper, we demonstrate a trivial facial recognition algorithm called System A based on binarization. We then develop a second algorithm, System B, which is also based on binarization. We finally demonstrate that System B is more accurate than System A.**

I. Introduction

Facial recognition algorithms take images of faces as input and determine who they belong to. This technology has practical uses, for example, in biometric authentication. In this paper, we demonstrate the performance of a trivial facial recognition algorithm called System A which consists of two operations: thresholding two images and calculating the Hamming distance between them. We then devise an improved facial recognition algorithm called System B. System B contains additional preprocessing before thresholding and uses the Euclidean distance measure. We demonstrate a significant improvement in the accuracy of System B over System A. Our work shows the potential for improving facial recognition algorithms based on binarization.

II. Background

Several terms used throughout the paper are now introduced.

A. Thresholding

Thresholding is a simple technique to convert a grayscale image to a binary image. Each pixel in the output image is assigned '1' if the corresponding input pixel is greater than some threshold; otherwise, it is assigned '0'. Equation (1) shows the formula for image binarization by thresholding.

$$I'(x,y) = \begin{cases} 1 \text{ if } I(x,y) > \text{threshold} \\ 0 \text{ otherwise} \end{cases} \quad (1)$$

B. 2D Filter

Filtering an image means convolving some matrix, called a kernel, with an image. The function applies an arbitrary linear filter to an image [1]. For the purposes of this project, the kernel in Figure 1 is used.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figure 1: 2D-Filter Kernel

C. Gaussian Blur

The Gaussian Blur is implemented by filtering an image using a Gaussian kernel [1]. The Gaussian kernel used for this project is shown in Figure 2.

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

Figure 2: Gaussian Kernel

D. Hamming Distance

The Hamming Distance between two equally sized binary images is defined as the number of pixels which differ between the two images [2].

E. Euclidean Distance

Euclidean distance is defined as the square root of the sum of squared differences between corresponding pixel values. Because the images will be binary, this is the same as the square root of the number of pixels which differ between the two images [2].

F. Score Matrix

For the purposes of this project, a facial recognition algorithm takes two images as input and outputs a "score" quantifying how similar the faces in each image are.

A score matrix S can be generated given an algorithm and a set of input images. To generate S, the input images are partitioned into a "probe" set and a "gallery" set. $S[i,j]$ is then the score produced by the algorithm comparing the ith image in the probe set to the jth image in the gallery set.

G. Decidability Index

The decidability index of a facial recognition algorithm quantifies how effectively the algorithm correctly identifies two faces as either matching or non-matching.

A "genuine" score is one that was generated by the algorithm comparing two matching faces. An "imposter" score is one that was generated by the algorithm comparing two non-matching faces. Therefore, given a probe set, a gallery set, and an algorithm's score matrix S, $S[i,j]$ is a genuine score if the ith probe image and the jth gallery image contain the same face. It is an imposter score if they contain different faces.
Given the genuine and imposter scores, the decidability index is calculated by (2).

$$d' = \frac{\sqrt{2}|\mu_1 - \mu_0|}{\sqrt{\sigma_1^2 + \sigma_0^2}} \tag{2}$$

In (2), μ_1 and σ_1 are the mean and standard deviation, respectively, of all the genuine scores in a score matrix. Likewise, μ_0 and σ_0 are the mean and standard deviation of all the imposter scores.

Intuitively, if the algorithm tends to output the same score for two matching faces as it does for two non-matching faces, then the mean genuine score will be close to the mean imposter score, and the decidability index will be close to zero. If the algorithm tends to output a much higher score for two matching faces than it does for two non-matching faces, then the mean genuine score will be much higher than the mean imposter score, and the decidability index will be large. This property makes the decidability index suitable for quantifying the effectiveness of a facial recognition algorithm.

H. Improvement Factor

The improvement factor is used to compare two facial recognition algorithms. The improvement factor of algorithm B over algorithm A is defined as the difference between B's decidability index and A's decidability index, shown formally in (3). A higher improvement factor value indicates a better performance by algorithm B over algorithm A.

$$\text{IF} = d'_B - d'_A \tag{3}$$

III. Methodology

A. Image Handling

The first step for both systems is reading in the provided images as single channel grayscale images. Figure 3 and Figure 4 show sample images from the gallery and probe sets, respectively. Reading in the images is implemented in the following code block [3]:

```
gallery_imgs = []
probe_imgs = []
for i in range(100):
    gallery_img_gray =
cv2.imread("data/gallery/subject{}_img1.pgm".format(i+1),
cv2.IMREAD_GRAYSCALE)
    probe_img_gray =
cv2.imread("data/probe/subject{}_img2.pgm".format(i+1),
cv2.IMREAD_GRAYSCALE)
    gallery_imgs.append(gallery_img_gray)
    probe_imgs.append(probe_img_gray)
```

Figure 3: Sample gallery image

Figure 4: Sample probe image

B. System A

In System A, images are binarized by thresholding with a threshold value of 128. The result of this process is 5 and Figure 6.

Figure 5: Sample gallery image after Binary Thresholding

Figure 6: Sample probe image after Binary Thresholding

Our implementation to calculate System A's score matrix is given in the following code block [3]:

```
def System_A_Score_Matrix(gallery_imgs, probe_imgs):
    for i, img in enumerate(gallery_imgs):
        _, gallery_imgs[i] =
cv2.threshold(img,128,255,cv2.THRESH_BINARY)
    for i, img in enumerate(probe_imgs):
        _, probe_imgs[i] =
cv2.threshold(img,128,255,cv2.THRESH_BINARY)

    A = np.empty((100,100))
    for i in range(100):
        for j in range(100):
            A[i, j] = dist.HammingDistance(probe_imgs[i], gallery_imgs[j])

    return A
```

C. System B

In System B, images are first preprocessed to improve the accuracy of the facial recognition system before undergoing binarization. Two of the preprocessing techniques were linear filters called Filter2D and Gaussian Blur. By using these filters, the noise in the images is reduced and a smoothing effect is applied. In between the application of the linear filters, the images were cropped. The bottom third of images were cropped off in order to remove the mouth, which tends to vary between probe and gallery images. The result of this pre-processing is shown in Figure 7 and Figure 8.

Figure 7: Sample gallery image after System B pre-processing

Figure 8: Sample probe image after System B pre-processing

For the binarization of the system, the same method as System A is used, i.e., "binarization by thresholding" but instead of having a threshold value of 128 a threshold value of 60 is used. The result of binarization is shown in Figures 9 and 10.

Figure 9: Sample probe image after Binary Thresholding

Figure 10: Sample probe image after Binary Thresholding

Our implementation to calculate System B's score matrix is given in the following code block [3]:

```
def System_B_Score_Matrix(gallery_imgs, probe_imgs):
    filt_2d_kernel = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
    for i, img in enumerate(gallery_imgs):
        gallery_imgs[i] = cv2.filter2D(gallery_imgs[i].copy(), ddepth= -1,
kernel= filt_2d_kernel)
        gallery_imgs[i] = gallery_imgs[i][0:33, :]
        gallery_imgs[i] = cv2.GaussianBlur(gallery_imgs[i].copy(), (3,3), 0)
    for i, img in enumerate(probe_imgs):
        probe_imgs[i] = cv2.filter2D(probe_imgs[i].copy(), ddepth= -1,
kernel= filt_2d_kernel)
        probe_imgs[i] = probe_imgs[i][0:33, :]
        probe_imgs[i] = cv2.GaussianBlur(probe_imgs[i].copy(), (3,3), 0)
    for i, img in enumerate(gallery_imgs):
        _, gallery_imgs[i] = cv2.threshold(img,60,255,cv2.THRESH_BINARY)
    for i, img in enumerate(probe_imgs):
```

```
    _, probe_imgs[i] = cv2.threshold(img,60,255,cv2.THRESH_BINARY)
A = np.empty((100,100))
for i in range(100):
    for j in range(100):
        A[i, j] = dist.EuclideanDistance(probe_imgs[i], gallery_imgs[j])
return A
```

IV. Results

The goal with our systems is to maximize System B's decidability index, thus increasing our improvement factor. The score matrix and decidability index is given for System A and System B, and the improvement factor of System B over System A is computed.

A. System A

System A's score matrix is visualized in Figure 11. A snippet of the first 10 values along both axes is visualized in Figure 12 and shown in plain text in Figure 13. The score matrix is visualized as a heat map with smaller distances as green and larger distances as red. In Figure 11, a green line is present along the diagonal, where images are of the same individual, indicating that the matching faces do in fact have a smaller Hamming Distance score. This is more evident in Figure 12 where it is possible to identify matching faces. The visuals display the contrast in values between genuine scores and the imposter scores.

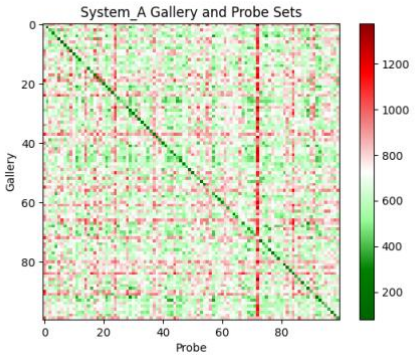

Figure 11: System A Score Matrix (Graph)

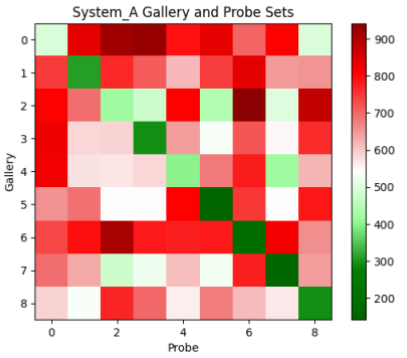

Figure 12: System A Score Matrix Snippet (Graph)

```
Part 1A: A[0:9, 0:9] snippet:
[[492. 837. 916. 926. 791. 833. 703. 808. 498.]
 [748. 309. 766. 712. 615. 743. 841. 648. 652.]
 [809. 694. 423. 479. 808. 442. 942. 501. 877.]
 [828. 585. 588. 292. 643. 535. 717. 550. 762.]
 [822. 573. 568. 584. 401. 681. 777. 418. 618.]
 [657. 690. 545. 545. 810. 142. 750. 539. 785.]
 [733. 792. 907. 781. 774. 780. 190. 819. 659.]
 [693. 630. 477. 523. 608. 524. 774. 159. 643.]
 [587. 534. 769. 701. 560. 678. 612. 565. 293.]]
```

Figure 13: System A Score Matrix Snippet (Plain Text)

System A obtained a decidability index of ~2.9104. This value will serve as a baseline for System B's performance.

Part 1c - Suppose another face recognition system called FaceRec yielded a decidability index of 3.5 on the same dataset. This decidability index implies better performance and will feature more genuine results. This means FaceRec would be better at differentiating the matching face from other faces so the margin between a match and other faces will be higher. By definition, the decidability index quantifies the separation between genuine and imposter scores, so a higher score indicates it is more easily able to accomplish the task of making the distinction.

B. System B

System B's score matrix is visualized as a heat map in Figure 14 and a snippet of the first 10 values along the two axes are shown in Figure 15. A plain text version is also shown in Figure 16. The heat maps indicate larger distances in red and smaller distances in green. In Figure 14, there is a green line along the diagonal of the heat map, which is red elsewhere. This indicates that the faces that match have a significantly smaller Euclidean Distance score. This disparity is shown in more detail in Figure 15. When comparing Figure 14 to Figure 11, it is obvious that there is much more contrast in color. The pixels are either green or red, with fewer lightly shaded boxes like those seen in Figure 11. There is no visible green in Figure 14 outside of the genuine score region (along the diagonal).

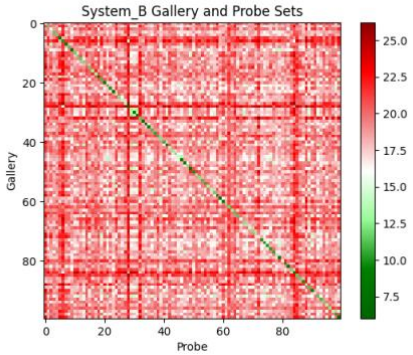

Figure 14: System B Score Matrix (Graph)

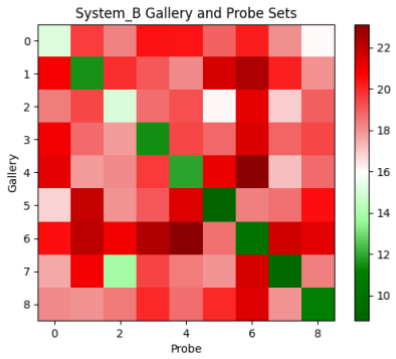

Figure 15: System B Score Matrix Snippet (Graph)

15.13274...	19.54482...	18.27566...	20.39607...	20.34698...	18.89444...	20.19900...	18.02775...	16.06237...
20.83266...	11.53256...	19.82422...	19.05255...	18.13835...	21.54065...	22.38302...	20.14944...	17.97220...
18.41195...	19.39071...	15.09966...	18.65475...	19.23538...	16.09347...	21.21320...	16.88194...	18.97366...
20.95232...	18.70828...	17.83255...	11.48912...	19.39071...	18.78829...	21.49418...	18.84144...	19.39071...
21.30727...	17.80449...	18.13835...	19.62141...	11.87434...	21.07130...	23.08679...	17.14642...	18.73499...
16.76305...	21.90890...	17.94435...	19.07878...	21.40093...	8.774964...	18.33030...	18.62793...	20.49390...
20.44504...	22.11334...	20.95232...	22.33830...	23.13006...	18.60107...	10.04987...	21.63330...	21.28379...
17.57839...	20.92844...	13.85640...	19.44222...	18.38477...	17.91647...	21.54065...	9.327379...	18.33030...
18.13835...	18	18.43908...	19.94993...	18.65475...	19.97498...	21.40093...	17.91647...	11.13552...

Figure 16: System B Score Matrix Snippet (Plain Text)

System B obtained a decidability index of ~3.8174. Comparing the two systems, we ended up with a positive improvement factor of 0.91. This indicates that System B performs better than System A.

One reason for System B's improved performance is that cropping the mouth out removed a major source of dissimilarity between matching faces. The other factor which improved performance is that the blurring and filtering removed fine details from the image, allowing the similar overall shapes in matching faces to be assigned a higher score.

V. Conclusion

Improvements were made when comparing System B to System A as the decidability score increased from 2.91 to 3.82. This can be attributed to pre-processing techniques and a threshold better suited for the lighting conditions of the images in the dataset. The thresholding used in System A is not able to easily capture the minute details seen in the images, and the lack of pre-processing leaves the images vulnerable to noise and does not take advantage of smoothening or sharpening.

VI. Contributions

Caden Kroonenberg: Implemented pipeline to read in gallery and probe image sets and pass them off to functions for computing the score matrices for Systems A and B. Implemented functions for calculating decidability index and System A's score matrix. Added Gaussian Blur and 2D-Filter pre-processing methods to System B. Contributed to Background, Methodology, Results, and References sections of the final report.

Oluwanifemi Fadahunsi: Implemented an initial build of System B adding preprocessing features like Gaussian Blur, Median Blur and 2D Filtering. This aspect of the code was then edited by Caden Kroonenberg. Worked on multiple feature extraction methods like SIFT and ORB but due to the lack of binarization to the keypoints and descriptors, code was scrapped and replaced by the thresholding created by Javante Ewing with a few tweaks. Contributed to the final report by describing the methodology and results of System B.

Javante Ewing: Implemented a function for calculating Hamming Distance. Implemented binarization by threshold method used in Systems A and B. Contributed to final report sections on Background and Methodology.

Andrew Riachi: Discovered that cropping out the mouth improved facial recognition accuracy. Added cropping into System B. Contributed to the Abstract, Introduction, and Background sections of the report.

Michael Talaga: Implemented distance functions in testing (Euclidean and City Block). Introduced LBP as a pre-processing method for System B. Added Gaussian Blur and Median Blur as options for pre-processing. Investigated adaptive thresholding and OTSU thresholding as methods for feature binarization. Developed code for "heat-map" plots. Wrote Results and Conclusion sections in report.

References

[1] "Image Filtering." *OpenCV*, https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1.

[2] "Matrix Distance." Matrix Distance, https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/matrdist.htm

[3] facial-recognition-eecs741, (2023), Github Repository, github.com/cskroonenberg/facial-recognition-eecs741

APPENDIX

GitHub repo: https://github.com/cskroonenberg/facial-recognition-eecs741

main.py:

```
import cv2
import math
import numpy as np
from system_a import System_A_Score_Matrix
from system_b import System_B_Score_Matrix
from plot_matrix import plot_matrix

def DecidabilityIndex(score_matrix):
    genuine_scores = np.array([])
    imposter_scores = np.array([])
    for i in range(100):
        genuine_scores = np.append(genuine_scores, score_matrix[i, i])
        imposter_scores = np.append(imposter_scores, score_matrix[i, :i])
        imposter_scores = np.append(imposter_scores, score_matrix[i, (i+1):])

    genuine_mean = np.mean(genuine_scores)
    imposter_mean = np.mean(imposter_scores)

    genuine_std_dev = np.std(genuine_scores)
    imposter_std_dev = np.std(imposter_scores)

    num = math.sqrt(2)*abs(genuine_mean - imposter_mean)
    denom = math.sqrt(math.pow(genuine_std_dev, 2) + math.pow(imposter_std_dev, 2))

    return num/denom

def main():
    # Read in images
    gallery_imgs = []
    probe_imgs = []
    for i in range(100):
        gallery_img_gray = cv2.imread("data/gallery/subject{}_img1.pgm".format(i+1), cv2.IMREAD_GRAYSCALE)
        probe_img_gray = cv2.imread("data/probe/subject{}_img2.pgm".format(i+1), cv2.IMREAD_GRAYSCALE)
        gallery_imgs.append(gallery_img_gray)
        probe_imgs.append(probe_img_gray)

    #cv2.imwrite('sample_gallery.png', gallery_imgs[1])
    #cv2.imwrite('sample_probe.png', probe_imgs[1])

    # Generate score matrix for first facial recognition method
    A = System_A_Score_Matrix(gallery_imgs.copy(), probe_imgs.copy())
    print("Part 1A: A[0:9, 0:9] snippet:")
    print(A[0:9, 0:9])

    #plot_matrix(A)
```

```
    #plot_matrix(A[0:9, 0:9])

    hamming_decidability_idx = DecidabilityIndex(A)
    print("Part 1B: Hamming distance decidability index: {}".format(hamming_decidability_idx))

    # Generate score matrix for second facial recognition method
    B = System_B_Score_Matrix(gallery_imgs.copy(), probe_imgs.copy())
    print("Part 2A: B[0:9, 0:9] snippet:")
    print(B[0:9, 0:9])

    #plot_matrix(B)
    #plot_matrix(B[0:9, 0:9])

    system_b_decidability_idx = DecidabilityIndex(B)
    print("Part 2B: System B decidability index: {}".format(system_b_decidability_idx))

    improvement_factor = round((system_b_decidability_idx - hamming_decidability_idx), 2)
    if improvement_factor < 0:
        score = 0
    elif improvement_factor == 0:
        score = 5
    elif improvement_factor >= 0.1 and improvement_factor < 0.4:
        score = 10
    elif improvement_factor >= 0.4 and improvement_factor < 0.9:
        score = 15
    elif improvement_factor >= 0.9 and improvement_factor < 1.5:
        score = 18
    else:
        score = 20
    print("Part 2C Improvement Factor (IF) = {}\tScore = {}".format(improvement_factor, score))

if __name__ == "__main__":
    main()
```

dist.py:

```
import numpy as np

def EuclideanDistance(img_a, img_b):
    # TODO: Calculate distance between images using some distance function
    #COMPLETE(Caden): Calculate total Euclidean distance
    distance = 0
    diff_arr = np.logical_xor(img_a, img_b)
    distance = diff_arr.sum()
    distance = distance**0.5
    return distance

def HammingDistance(img_a, img_b):
    # TODO: Calculate Hamming Distance
```

```
    # COMPLETE(Javante): calculate hamming distance
    return len((img_a != img_b).nonzero()[0])

def CityBlockDistance(img_a, img_b):
    # TODO: Calculate distance between images using some distance function
    #COMPLETE(Michael): Calculate City Block distance
    distance = 0
    for i in range(len(img_a)):
        distance += np.sum(np.abs(img_a[i] - img_b[i]))
    return distance

def ChessboardDistance(img_a, img_b):
    # TODO: Calculate distance between images using some distance function
    #Incomplete(Michael):Calculate Chessboard distance
    distance = 0
    for i in range(len(img_a)):
        distance += np.max(np.abs(img_a[i] - img_b[i]))
    return distance
```

plot_matrix.py:

```
import matplotlib.pyplot as plt
from  matplotlib.colors import LinearSegmentedColormap

def plot_matrix(score_matrix):
    cmap=LinearSegmentedColormap.from_list('rg',["darkgreen","green" , "palegreen", "w","lightcoral", "red", "darkred"], N=256)
    plt.xlabel("Probe")
    plt.ylabel("Gallery")
    plt.title("System_B Gallery and Probe Sets")
    plt.imshow(score_matrix, interpolation='none', cmap=cmap)
    plt.colorbar()
    plt.show()
```

system_a.py:

```
import cv2
import numpy as np
import dist

def System_A_Score_Matrix(gallery_imgs, probe_imgs):
    # Convert images to binary
    for i, img in enumerate(gallery_imgs):
        _, gallery_imgs[i] = cv2.threshold(img,128,255,cv2.THRESH_BINARY)
    for i, img in enumerate(probe_imgs):
        _, probe_imgs[i] = cv2.threshold(img,128,255,cv2.THRESH_BINARY)

    #cv2.imwrite('sysA_binarized_gallery.png', gallery_imgs[1])
    #cv2.imwrite('sysA_binarized_probe.png', probe_imgs[1])

    A = np.empty((100,100))
    for i in range(100):
```

```
    for j in range(100):
        A[i, j] = dist.HammingDistance(probe_imgs[i], gallery_imgs[j])

return A
```

system_b.py:

```
import cv2
import numpy as np
import dist

def System_B_Score_Matrix(gallery_imgs, probe_imgs):
    # cv2.imshow('original gallery', gallery_imgs[1])
    # cv2.imshow('original probe', probe_imgs[1])

    # Crop mouth
    filt_2d_kernel = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
    for i, img in enumerate(gallery_imgs):
        gallery_imgs[i] = cv2.filter2D(gallery_imgs[i].copy(), ddepth= -1, kernel= filt_2d_kernel)
        gallery_imgs[i] = gallery_imgs[i][0:33, :]
        gallery_imgs[i] = cv2.GaussianBlur(gallery_imgs[i].copy(), (3,3), 0)
    for i, img in enumerate(probe_imgs):
        probe_imgs[i] = cv2.filter2D(probe_imgs[i].copy(), ddepth= -1, kernel= filt_2d_kernel)
        probe_imgs[i] = probe_imgs[i][0:33, :]
        probe_imgs[i] = cv2.GaussianBlur(probe_imgs[i].copy(), (3,3), 0)


    #cv2.imwrite('sysB_modified_gallery.png', gallery_imgs[1])
    #cv2.imwrite('sysB_modified_probe.png', probe_imgs[1])

    # Convert images to binary
    for i, img in enumerate(gallery_imgs):
        _, gallery_imgs[i] = cv2.threshold(img,60,255,cv2.THRESH_BINARY)
    for i, img in enumerate(probe_imgs):
        _, probe_imgs[i] = cv2.threshold(img,60,255,cv2.THRESH_BINARY)

    #cv2.imwrite('sysB_binarized_gallery.png', gallery_imgs[1])
    #cv2.imwrite('sysB_binarized_probe.png', probe_imgs[1])

    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    # Calculate distance scores between gallery and probe images
    A = np.empty((100,100))
    for i in range(100):
        for j in range(100):
            A[i, j] = dist.EuclideanDistance(probe_imgs[i], gallery_imgs[j])

    return A
```