Face Detection and Recognition

Table Of Contents

- Face Detection
- Face Recognition
- ROC Curve

Submitted By: Team 15
Submitted To: Dr. Ahmed Badawi
Eng. Omar Dawah
Eng. Laila Abbas

Face Detection

Introduction:

We start by leveraging pre-trained models within OpenCV, specifically the Haar cascade classifier, which has been trained on thousands of images to recognize facial features. This classifier serves as the backbone of our system, enabling us to detect faces with accuracy and efficiency.

Our journey involves breaking down the face detection process into simple steps, from loading the classifier to drawing rectangles around detected faces. Each step contributes to the overall functionality of our system, transforming raw images into actionable data.

Throughout this report, we explain the logic behind each decision. By understanding concepts like scale factors and neighbor thresholds, we can optimize our system for different scenarios, whether it's identifying faces in a crowded room or analyzing images with varying lighting conditions.

Steps of Face Detection:

1. **Loading the Pre-trained Face Detector:**
    - The first step involves loading a pre-trained face detection model. In our implementation, we utilize the Haar cascade classifier provided by OpenCV. This classifier is trained on a large dataset of positive and negative images to recognize patterns corresponding to human faces.
2. **Detecting Faces in the Image:**
    - Once the face detector is loaded, we apply it to the input image. The detectMultiScale function is used to identify potential face regions within the image. This function operates by scanning the image at multiple scales and detecting objects that match the learned patterns.
3. **Drawing Rectangles Around Detected Faces:**
    - After detecting faces, we draw rectangles around each detected face region to visually indicate their location in the image. This step is crucial for visualization and further analysis of the detected faces.
4. **Displaying the Image with Detected Faces:**

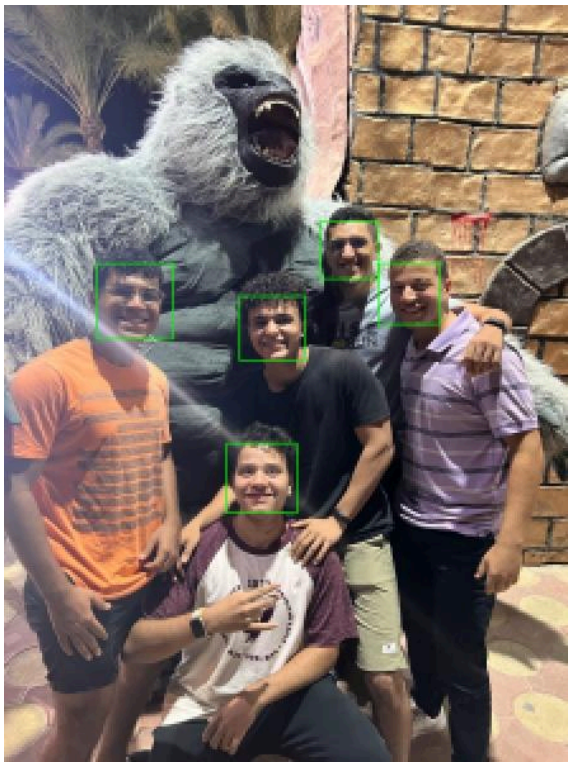
- Finally, we display the original image overlaid with the rectangles highlighting the detected faces. This provides a comprehensive view of the face detection results and facilitates further interpretation or processing of the image.

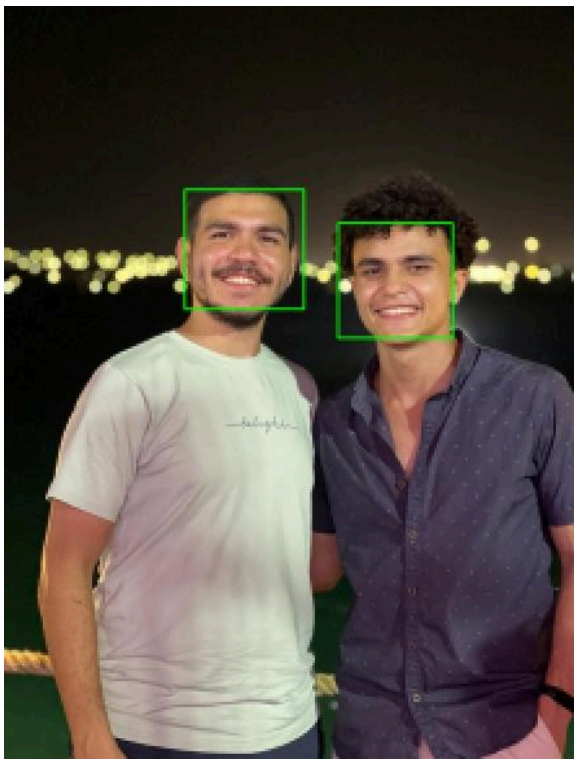
Overview of the Logic Behind the Library:

The face detection logic implemented in our system relies on the Haar cascade classifier provided by the OpenCV library. Here's an overview of the key aspects of this library and its functionality:

- **Haar Cascade Classifier:**
  - The Haar cascade classifier is a machine learning-based algorithm used for object detection, including face detection. It operates by analyzing the intensity patterns of pixels in an image to identify regions containing objects of interest.
- **detectMultiScale Function:**
  - This function is a fundamental component of the face detection process. It applies the Haar cascade classifier to the input image, scanning it at multiple scales to detect potential face regions. The function returns a list of rectangles representing the detected faces' positions and sizes.
- **Parameters:**
  - Several parameters, such as scaleFactor, minNeighbors, minSize, and maxSize, are provided to fine-tune the face detection process. These parameters control the sensitivity, accuracy, and computational efficiency of the detection algorithm.
- **Rectangular Drawing and Image Display:**
  - After detecting faces, the algorithm draws rectangles around each detected face region on the original image. This step allows users to visually inspect the detection results. Finally, the image with overlaid rectangles is displayed to provide a comprehensive view of the detected faces.

The Results:





Face Recognition

Introduction to Face Recognition using Eigenfaces:

Face recognition, a fundamental task in computer vision, has garnered significant attention due to its wide range of applications in security, surveillance, human-computer interaction, and personalization systems. Among the myriad of techniques developed for face recognition, Eigenfaces stands as one of the pioneering methods, offering a robust and computationally efficient solution.

Eigenfaces, a concept introduced by Matthew Turk and Alex Pentland in 1991, leverages Principal Component Analysis (PCA) to represent facial images in a lower-dimensional space. This approach transforms high-dimensional image data into a set of basis vectors, known as eigenfaces, which encapsulate the most significant features of the face images. By projecting new facial images onto this eigenface space, recognition becomes a matter of measuring the similarity between the eigenface representations.

The core idea behind Eigenfaces lies in the recognition of faces based on holistic features rather than local details. This property makes it resilient to variations in lighting conditions, facial expressions, and minor occlusions. Moreover, Eigenfaces require minimal preprocessing and can handle a large number of images efficiently, making them suitable for real-time applications.

Through this project, we aim to demonstrate the efficacy of Eigenfaces in real-world scenarios and contribute to the advancement of face recognition technologies, paving the way for innovative applications in various domains.

Overview:

In this project, we explore the application of Eigenfaces for face recognition, utilizing a dataset consisting of 5 individuals, each with 12 facial images. The dataset was divided into a training set, comprising 9 images per person, and a testing set, comprising 3 images per person.

The core methodology involved applying Principal Component Analysis (PCA) to the training images to extract the most discriminative features. By computing the eigenvectors corresponding to the largest eigenvalues, we obtained a set of basis vectors known as eigenfaces, which capture the essential facial characteristics shared across the dataset.

These eigenfaces formed a lower-dimensional space in which all facial images could be represented. Subsequently, we projected both the training and testing datasets onto this eigenface space. This transformation enabled us to effectively reduce the dimensionality of the data while preserving the essential facial information.

During the recognition phase, new facial images were projected onto the same eigenface space. By measuring the similarity between the projected test image and the training dataset, we assigned the test image to the closest matching individual based on a predefined threshold or distance metric.

Through this approach, we aimed to demonstrate the effectiveness of Eigenfaces in accurately recognizing faces across varying conditions such as lighting, facial expressions, and minor occlusions. The project contributes to the understanding and application of dimensionality reduction techniques in the field of computer vision, particularly in the domain of face recognition.

Main Steps:

1. **Data Preparation:**
    - Navigate to the directory containing all the training images.
    - Upload all the images and store them in a data structure, such as a matrix, where each column represents an image.
    - Each image is typically represented as a vector by flattening it into a one-dimensional array, with each element corresponding to a pixel value.

2. **Mean Image Calculation and Centering:**
    - Calculate the mean image by averaging pixel values across all images in the dataset. This results in a single mean image that represents the average facial appearance across the dataset.
    - Subtract the mean image from each individual image in the dataset. This process, known as centering, aligns all images around a common reference point, effectively removing variations in lighting and background across different images.
    - The centered images now reflect the differences or variations from the average facial appearance captured by the mean image. This step is crucial for reducing the impact of illumination changes and enhancing the discriminative features for face recognition.

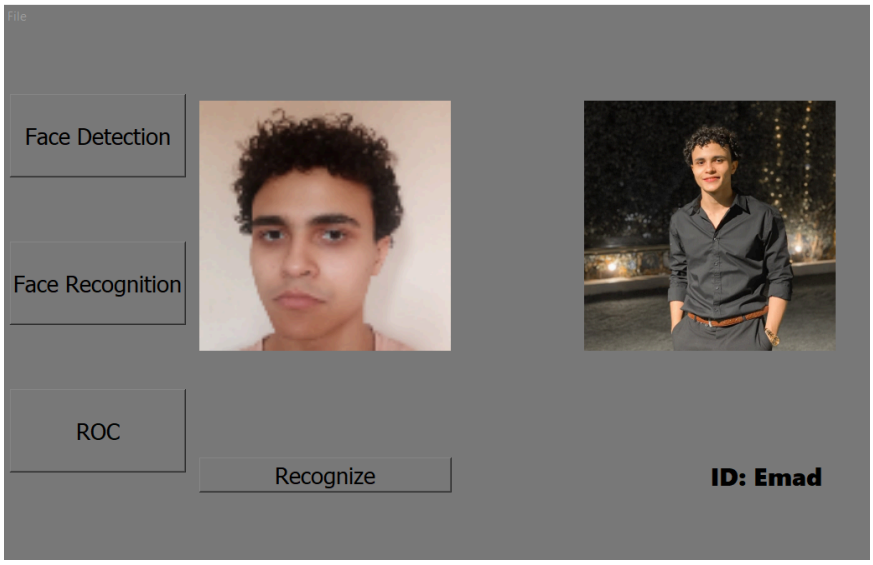
3. **Principal Component Analysis (PCA):**
    - Calculate the covariance matrix of the matrix of centered images, capturing the relationships between pixel values across all images.
    - Perform eigen decomposition on the covariance matrix to obtain eigenvalues and eigenvectors. These eigenvectors represent the directions of maximum variance in the dataset, while the eigenvalues indicate the amount of variance explained by each eigenvector.
    - Sort the eigenvectors based on their corresponding eigenvalues in descending order. This ensures that the most significant components, capturing the most variance in the data, are prioritized.
    - Empirically determine the number of eigenfaces to retain based on the dataset. In this project, 40 eigenfaces were chosen, reflecting a balance between computational efficiency and capturing essential facial variations present in the images. This selection was based on experimentation and analysis of the dataset, ensuring that it sufficiently represented the underlying facial structure and features.

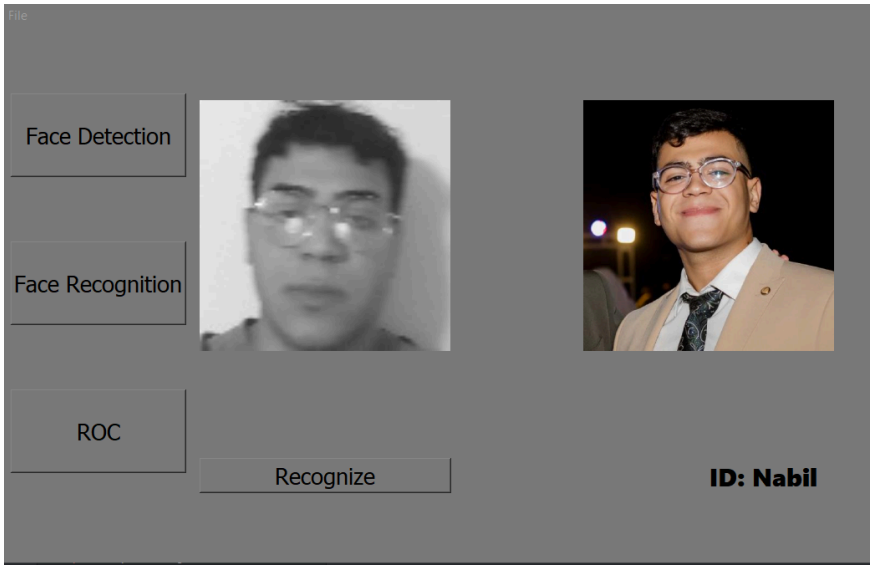
4. **Projection onto Eigenface Space:**
    - Utilize the selected eigenfaces, which form a new basis for representing facial images, to project all images in the training dataset onto this new space.
    - Each image in the training dataset is projected onto the eigenface space by computing the dot product between the image vector and each of the chosen eigenfaces.
    - The resulting projection coefficients represent the contribution of each eigenface to the representation of the corresponding facial image in the new space.
    - This projection process effectively transforms each facial image from its original high-dimensional pixel space into a lower-dimensional space defined by the chosen eigenfaces.
5. **Testing Phase - Recognition:**
    - To recognize a new facial image, apply the same projection process used during training to represent the image in the lower-dimensional space defined by the selected eigenfaces.
    - Compute the projection coefficients of the new image by taking the dot product with each of the chosen eigenfaces.
    - With the new image represented in the eigenface space, compare it with the representations of all images in the training dataset.
    - Calculate the Euclidean distance between the projection coefficients of the new image and those of each training image.
    - Assign the new image to the person whose training image yields the smallest Euclidean distance. If the distance exceeds a predefined threshold, classify the image as unknown.
    - This process effectively matches the new image to the closest representation in the training dataset, allowing for accurate classification of the individual depicted in the image or identification of unknown faces.

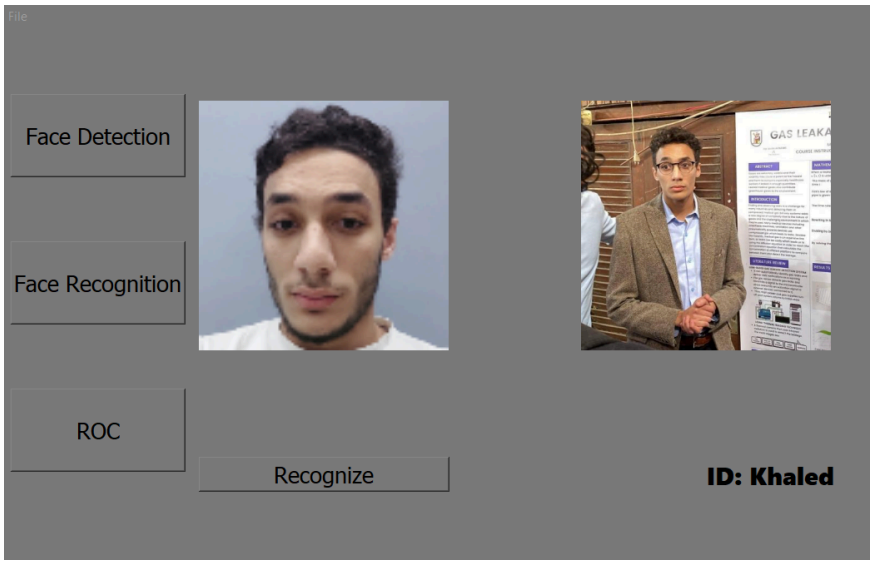
Conclusion:

In summary, this project successfully demonstrates the efficiency of Eigenfaces for face recognition using Principal Component Analysis (PCA). Through careful experimentation and analysis, we determined the optimal number of eigenfaces required for accurate recognition while maintaining computational efficiency.

The Results:


File
Face Detection
Face Recognition
ROC
Recognize
ID: Emad


File
Face Detection
Face Recognition
ROC
Recognize
ID: Nabil


File
Face Detection
Face Recognition
ROC
Recognize
ID: Khaled

ROC Curve

Introduction:

The Concept of ROC

The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classifier system as its discrimination threshold is varied. It plots two parameters:

- True Positive Rate (TPR, Sensitivity): The proportion of actual positives correctly identified by the classifier. It is calculated as:

$$TPR = \frac{TP}{TP+FN}$$

  where TP is true positives and FN is false negatives.

- False Positive Rate (FPR, 1 - Specificity): The proportion of actual negatives incorrectly classified as positives. It is calculated as:

$$FPR = \frac{FP}{TN+FP}$$

  where FP is false positives and TN is true negatives.

The Concept of AUC

Area Under the Curve (AUC) measures the entire two-dimensional area underneath the entire ROC curve.. It provides an aggregate measure of performance across all possible classification thresholds. The AUC value lies between 0 and 1:

- **AUC = 1:** Represents a perfect classifier. This classifier makes no mistakes and perfectly distinguishes between all positives and negatives. Every positive instance scores higher than every negative instance.
- **AUC > 0.5 and < 1:** Represents a classifier that generally performs well in distinguishing between the classes. The closer the AUC is to 1, the better it is at differentiating positives from negatives.
- **AUC = 0.5**: Represents a classifier that performs no better than random guessing. This scenario typically happens when the classifier's predictions are not correlated with the actual class labels.
- **AUC < 0.5:** Indicates a classifier that performs worse than random guessing. This suggests that the classifier is consistently wrong about its predictions. In practice, one might simply invert the predictions of such a classifier to improve performance.
- **AUC = 0:** Represents a perfectly incorrect classifier, which consistently misclassifies all instances. This theoretical scenario means the classifier always predicts a positive instance as negative and vice versa.

Performance Metrics Overview

In the evaluation of classification models, particularly binary classifiers, understanding the metrics of accuracy, precision, and specificity is essential for assessing the performance comprehensively. Each of these metrics provides insights into different aspects of a classifier's performance, and they are particularly useful in contexts where the balance between identifying positives and negatives is critical.

1-Accuracy

- **Definition:** Accuracy is the ratio of correctly predicted observations to the total observations. It gives a general measure of the model's overall correctness.
- **Formula:**

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{Total Observations}}$$

- **Interpretation:** A high accuracy rate indicates that the classifier performs well across both positive and negative instances. However, it might be misleading in unbalanced datasets where the majority class dominates the prediction. In such cases, even a naive model that predicts only the majority class can still achieve high accuracy.

2. Precision

- **Definition:** Precision is the ratio of correctly predicted positive observations to the total predicted positives. It is also known as Positive Predictive Value (PPV).
- **Formula:**

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

- **Interpretation:** Precision is a critical measure in scenarios where the cost of a false positive is high. For example, in email spam detection, a high precision means fewer regular emails are incorrectly marked as spam. A high precision indicates that when the model predicts an instance as positive, it is very likely to be correct. However, precision alone doesn't consider false negatives (instances that are positive but predicted as negative).

3. Specificity

- **Definition:** Specificity measures the proportion of actual negatives that are correctly identified as such. It is also known as the True Negative Rate (TNR).
- **Formula:**

$$\text{Specificity} = \frac{\text{True Negatives (TN)}}{\text{True Negatives (TN)} + \text{False Positives (FP)}}$$

- **Interpretation:** Specificity is especially important in fields like medical testing, where it's crucial not to falsely diagnose a healthy patient as sick. High specificity means the test is effective at identifying negatives. A high specificity rate indicates that the classifier is effective at ruling out the negative cases but similar to precision, it does not consider false negatives.

Algorithm Description of the 'plot_roc' Function

- The **plot_roc** function is designed to plot the Receiver Operating Characteristic (ROC) curve based on true labels and predicted labels derived from an adjustable threshold value set by the user.
- Algorithm:

1. Initialization and Setup

- **Threshold Acquisition:** The function starts by retrieving the current threshold value set by the user using a slider
- **Dialog Creation:** A dialog window is initialized where the ROC curve will be displayed. Also a Figure object is created to host the plot.

2. Data Preparation

- **True Labels:** A predefined array of true labels is defined, representing the actual classes of the instances.
- **Predicted Labels Generation:** Predicted labels are generated dynamically based on a distance metric (or similar measure) compared against the input threshold. This operation is looped for each data point (or feature set) represented by **self.roc_image_projected.shape[1]**.

3. ROC Computation Per Class

- **Classes Enumeration:** Unique classes are identified from the true labels. Storage Initialization: Dictionaries are initialized to store the False Positive Rate (FPR), True Positive Rate (TPR), and Area Under the Curve (AUC) for each class.
- **Metric Calculation:** For each class, the function creates binary representations of true and predicted labels. It computes the ROC curve and the AUC for these binary labels.Accuracy, precision, and specificity metrics are calculated.These metrics and ROC components (FPR and TPR) are stored for further processing.

4. Average ROC Curve Calculation

- **Unique FPR Aggregation:** All unique FPR values across classes are combined.
- **Mean TPR Calculation:** TPR values are interpolated and averaged at these FPR points across all classes, forming a mean ROC curve.

5. AUC and Additional Metrics Calculation

- **AUC Computation:** The AUC for the mean ROC curve is calculated.
- **Mean Metrics Computation:** The average values for accuracy, precision, and specificity are computed.

6. Plotting

- **ROC and Chance Curves:** The mean ROC curve and a 'chance' line (representing random guessing) are plotted.
- **Display:** The axes are labeled, and the title is set to reflect the current threshold. Metrics are added to the legend using a workaround of plotting an empty line to include non-graphical elements.

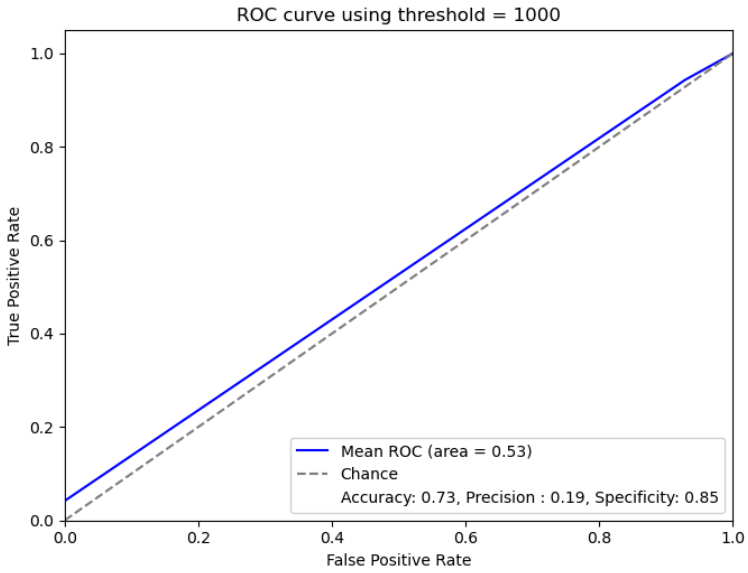
- Importance of the Threshold:

  The entire functionality and flexibility of the **plot_roc** function hinge on the **input_threshold** variable, which allows users to dynamically adjust how the classifier's predictions are determined based on a continuous or discrete metric. This adaptability is crucial for fine-tuning the classifier's performance and visually examining the impact of different thresholds on the classifier's sensitivity (TPR) and specificity (1 - FPR).

Results:

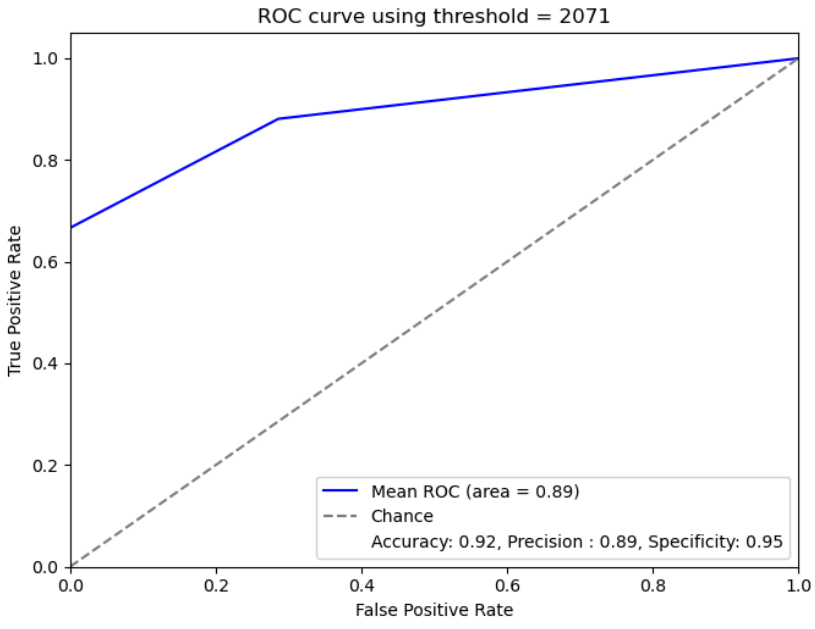
Threshold setting can significantly influence the classifier's performance in terms of its True Positive Rate (TPR), False Positive Rate (FPR), and the overall metrics such as accuracy, precision, and specificity. We are going to analyze different ROC curves at different thresholds:

1. ROC Curve at Threshold = 1000



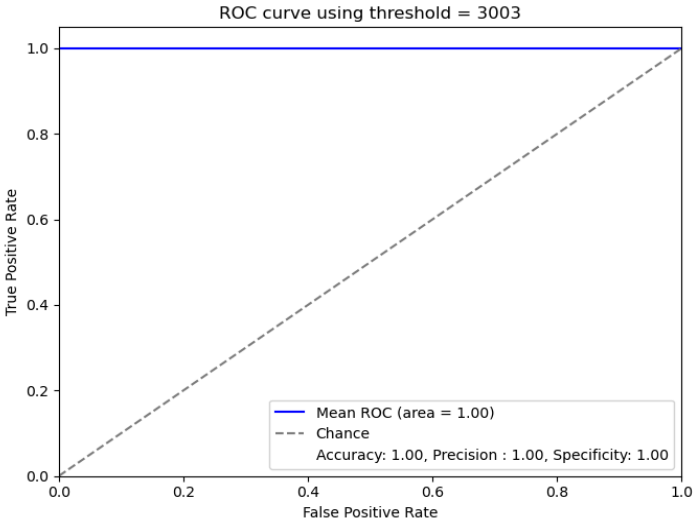
- AUC (Area Under Curve): 0.53
- Accuracy: 0.73 , Precision: 0.19 , Specificity: 0.85
- Observation: The AUC of 0.53 suggests that the classifier is barely performing better than random chance at this threshold. While the specificity is relatively high, indicating that the classifier is good at identifying true negatives, the precision is quite low, implying that a significant number of positive predictions are false.

2. ROC Curve at Threshold = 2071



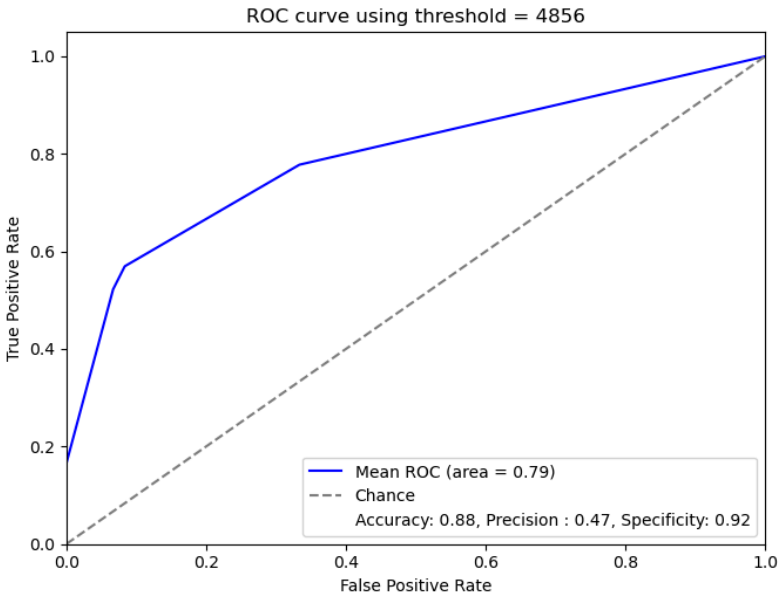
- AUC: 0.89
- Accuracy: 0.92, Precision: 0.89 , Specificity: 0.95
- Observations: This represents a significant improvement in all metrics. The AUC of 0.89 indicates a strong predictive performance. High accuracy, precision, and specificity suggest that the classifier is effective at identifying both positive and negative classes accurately, with few errors.

3. ROC Curve at Threshold = 3003



- AUC: 1.00
- Accuracy: 1.00 , Precision: 1.00,Specificity: 1.00
- Observations: This is an ideal scenario where the classifier achieves perfect prediction across all metrics. An AUC of 1.00 indicates the classifier perfectly discriminates between the positive and negative classes without any error.

4. ROC Curve at Threshold = 4856



- AUC: 0.79
- Accuracy: 0.88,Precision: 0.47,Specificity: 0.92
- Observation: Here, the classifier still performs decently with an AUC of 0.79, which suggests a good, though not perfect, capability to differentiate between classes. The high specificity indicates a low rate of false positives, but the precision is significantly lower, meaning that while the positive predictions are fewer, they're often incorrect. This threshold might be useful where it's critical to minimize false positives, but at the cost of missing several true positives.