Team 18

Face Recognition

Name	BN
Talal Mahmoud Emara	34
Maya Mohamed	52
Meram Mahmoud	68
Nouran Hani	75

Face Detection

We manually implemented face detection, using techniques such as color segmentation, edge detection, morphological processing, and geometric analysis. The aim is to detect human faces in images by extracting relevant features and identifying regions likely to contain faces.

---

1. Pipeline

The face detection follows a multi-stage pipeline:

1. **Grayscale Conversion**
2. **Skin Color Segmentation**
3. **Edge Detection**
4. **Region Masking and Refinement**
5. **Morphological Filtering**
6. **Contour Detection**
7. **Geometric Filtering**
8. **Facial Feature Verification (Eyes Detection)**

Each stage contributes to isolating facial regions while minimizing false positives due to background confusion or irrelevant objects.

---

2. Steps

2.1 Skin Color Segmentation

To isolate potential skin regions, the image is transformed into the **HSV (Hue, Saturation, Value)** color space, which separates chromatic content (hue and saturation) from intensity.

A range of HSV values associated with human skin tones is defined. A binary mask is generated by thresholding the HSV image using this range. Pixels falling within the defined hue and saturation bounds are retained as probable skin pixels, while others are discarded.

Adjusting the HSV threshold values significantly affects detection performance.

2.2 Edge Detection

The luminance-based grayscale image is then subjected to **Canny edge detection**, that:

- Smooths the image using a Gaussian filter to reduce noise.
- Computes gradient magnitude and direction.
- Applies non-maximum suppression to thin out edges.
- Uses double thresholding and edge tracking to retain strong and weak edge pixels.

This highlights boundaries in the image, particularly around facial features like the eyes, nose, and mouth.

2.3 Region Masking

The skin mask is combined with the inverted edge mask using a bitwise logical AND operation. This retains regions that are likely to be skin but **not dominated by sharp edges**, helping to isolate smooth skin surfaces typical of the human face while suppressing background textures and edges.

2.4 Morphological Filtering

To improve the quality of the skin mask and remove small artifacts or gaps, **morphological operations** are applied using an elliptical structuring element:

- **Closing**: Dilation followed by erosion; it fills small holes within the foreground regions.
- **Opening**: Erosion followed by dilation; it removes small noise regions.

These operations clean the binary mask and ensure that detected regions form contiguous, blob-like shapes typical of facial areas.

2.5 Contour Detection

Contours are extracted. These contours represent candidate regions for face detection.

Each contour is evaluated to compute:

- **Area**: Total number of pixels enclosed. Small regions are discarded as they are unlikely to represent faces so any area bellow 2000 pixels is discarded.

2.6 Geometric Filtering

Bounding rectangles are computed for each remaining contour. Their **aspect ratio** (width divided by height) and size are used as constraints:

- Acceptable aspect ratios typically range between **0.6 and 1.8**, reflecting the natural variability in face shapes (from round to slightly elongated).

These constraints help eliminate regions that geometrically do not resemble a human face, too narrow or too wide are discarded

---

2.7 Facial Feature Validation (Eye Detection)

To further verify if a region is indeed a face, the region of interest (ROI) is passed through an internal validation function that attempts to detect at least one eye so to prevent the program from deteceting a hand or any skin area as a face so we detect eyes to ensure it is a face.

The validation process includes:

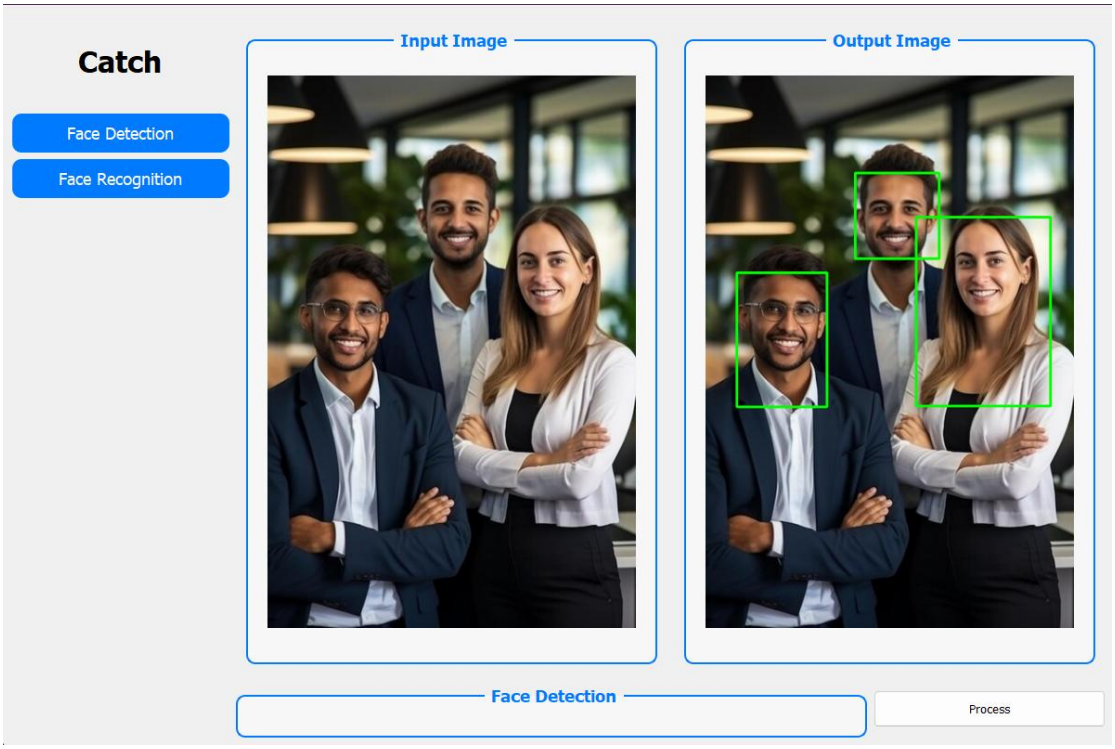
- **Histogram equalization**: Enhancing contrast in the ROI to make eye features more noticeable.
- **Thresholding**: Converting the image to binary by inverting pixel intensities, emphasizing dark regions typical of eyes.
- **Contour analysis**: Small, dark blobs (within a reasonable eye area from 50 toc 500) are considered potential eyes.

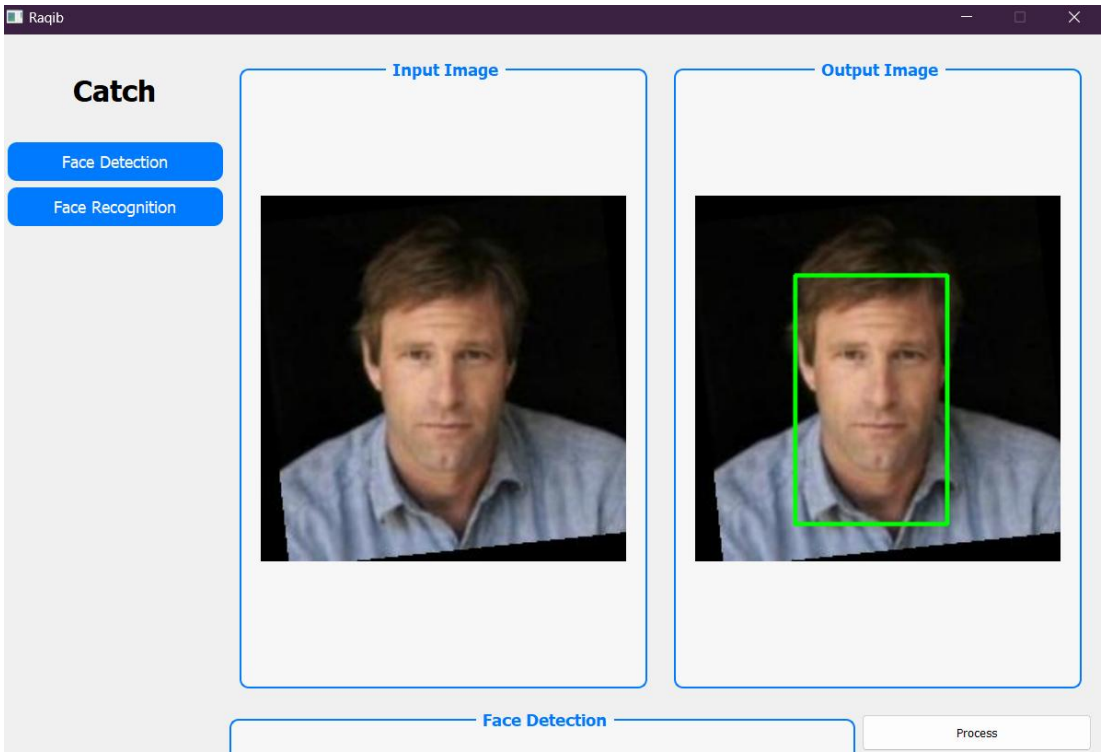
If at least one eye-like feature is detected, the region is accepted as a face.

---

3. Output

The result of the face detection process is a list of bounding boxes corresponding to regions that have passed all the above criteria. These are visualized by overlaying rectangles on the original image.





PCA

The pca function is designed to perform Principal Component Analysis (PCA) robustly, particularly suited for high-dimensional data like facial images. In face recognition, PCA is used to reduce the dimensionality of flattened face images while preserving the most important variations that distinguish one face from another. This technique enables the extraction of "Eigenfaces"—the principal components that represent common patterns across faces in the dataset. The function intelligently chooses between standard covariance computation and a more efficient "dual covariance trick" when the number of face images (samples) is smaller than the number of pixels (features), which is a common scenario. By projecting facial data onto the top n_components eigenvectors, the function allows each face to be represented compactly in a lower-dimensional space. Changing n_components directly affects recognition performance: fewer components may improve speed but lose detail, while more components retain more facial variance but increase computational cost. Thus, pca provides an effective and adaptable backbone for face recognition pipelines.

Model training

Image Preparation and Preprocessing

Before using images, they must pass through preprocessing stage, to enhance model performance, several preprocessing steps are applied:

- **Histogram equalization**: This improves the contrast of the luminance channel to standardize lighting conditions across images.
- **Data augmentation**: The system performs transformations such as horizontal flipping, brightness enhancement, rotation, and scaling. These augmentations help to increase the size and variability of the dataset, improving the robustness of the classifier.

Images that do not meet a minimum threshold of samples per class are filtered out to ensure adequate training data for each identity.

Dimensionality Reduction with PCA

Preprocessed images are then passed to the PCA function discussed earlier for dimensionality reduction, which is an important step to get more efficient results.

Classification with SVM

After transforming the images into PCA space, the system trains a Support Vector Machine (SVM) classifier with a linear kernel. The classifier learns to distinguish between different individuals based on their projected eigenface features.

To evaluate performance, the system uses 5-fold cross-validation and generates detailed classification reports, including accuracy and class-specific metrics. These metrics are helpful in assessing the generalization capability of the model.

Face Detection and Recognition

In the recognition phase, the system accepts a new test image and first detects the face region using a face detection module. The detected face is then preprocessed to match the training format.

The face is projected into PCA space using the same transformation derived during training. The projected feature vector is passed to the trained SVM classifier, which returns the predicted label corresponding to the closest match from the training data.

Model Persistence

To avoid retraining for every session, the system includes functionality to save and load trained models. All relevant components—including the SVM classifier, PCA transformation matrix, mean face vector, and label encoder—are serialized and stored on disk.

This allows the system to quickly reload a trained model and perform predictions in new environments without requiring access to the original training dataset.

Prediction Interface

A dedicated prediction interface enables users to perform recognition directly from an image file path. This function loads the image, performs detection and preprocessing, and returns the predicted label, assuming a trained model is available.

ROC

In the context of evaluating machine learning models, particularly in classification tasks, the Receiver Operating Characteristic (ROC) curve is a fundamental tool for assessing the performance of classifiers. It visualizes the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) at various threshold levels. ROC

analysis is especially useful for binary classification, but it can also be adapted for multi-class settings, as in this task, where the model classifies multiple classes.

Here outlines the procedure for performing ROC analysis in a multi-class classification problem using Support Vector Machines (SVM) with a One-vs-Rest approach. We break down the main functions involved and explain their roles in generating the ROC curve.

Methodology

1. **Model and Data Preparation**
    - The model used in this task is a linear Support Vector Machine (SVM) with the class_weight='balanced' parameter, ensuring that the model handles class imbalance appropriately. The model is trained on data that has undergone dimensionality reduction through Principal Component Analysis (PCA).
    - Labels are first encoded using a label encoder (LabelBinarizer), which transforms the categorical labels into a binary format for the classification task.
2. **Cross-validation Setup**
    - A Stratified K-Fold cross-validation strategy is applied to ensure that each fold maintains the class distribution, which is crucial when working with imbalanced datasets. The number of splits is set to 5, meaning the dataset is divided into five subsets, and the model is trained and tested on each fold in turn.
3. **ROC Analysis for Multi-class Classification**
   The process for ROC analysis in a multi-class setting involves the following steps:
    - **One-vs-Rest Strategy**: For multi-class classification, ROC analysis is adapted using the One-vs-Rest approach. In this approach, each class is treated as the "positive" class, and the other classes are combined as the "negative" class. The ROC curve is then computed for each individual class, treating it separately from the others.
    - **Decision Function**: After training the SVM model on each fold of the cross-validation, the decision function scores are computed for each test instance. These scores reflect the confidence the model has in its classification for each class. In a One-vs-Rest setup, the decision function returns a score for each class, and for each class, a binary decision score is extracted (1 if the instance belongs to the class, 0 otherwise).

- **Collecting Scores**: For each fold, the decision function scores and the corresponding true labels for the current class (binarized as 1 for the target class and 0 for all others) are collected and stored. This process accumulates the decision function scores across all folds for each class, providing the necessary data to compute the ROC curve.

4. **Computing the ROC Curve**

The **ROC curve** is computed for each class based on its accumulated scores and true labels:

- **Thresholding**: The decision scores are sorted in descending order to simulate how different threshold values will affect the classification. At each threshold, the predicted class labels are compared to the true labels to calculate the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).
- **True Positive Rate (TPR)**: This is the ratio of correctly identified positive samples (TP) to the total number of actual positive samples (TP + FN). It represents the sensitivity of the classifier for that class.
- **False Positive Rate (FPR)**: This is the ratio of incorrectly classified negative samples (FP) to the total number of actual negative samples (FP + TN). It represents the rate of false alarms for that class.
- These metrics are calculated for each threshold, and the TPR and FPR values are stored. The ROC curve is then plotted with FPR on the x-axis and TPR on the y-axis.

5. **Plotting the ROC Curve**

- **Plotting**: Once the TPR and FPR values are computed for each threshold, the ROC curve is plotted. A diagonal line representing random guessing (chance level) is added as a reference. The area under the ROC curve (AUC) can also be computed to quantify the model's overall performance, where a higher AUC indicates better classifier performance.
- **Visualization**: The ROC curve is plotted for each class individually. This visualization allows for an assessment of the classifier's performance per class, making it easier to identify if the model struggles with certain classes (e.g., those with high false positive rates).

Breaking Down the Functions

1. **compute_roc_curve**

This function is responsible for calculating the ROC curve from scratch. It takes the true labels (y_true) and the predicted decision scores (y_scores) as inputs. The function:

- Sorts the decision scores in descending order.
- Initializes variables to track true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).
- Iterates through each unique decision score (threshold), calculating the TPR and FPR at each threshold.
- Returns the FPR and TPR values, which are used to plot the ROC curve.

2. **plot_roc**

The plot_roc function takes the true labels and the decision scores as inputs, uses compute_roc_curve to get the FPR and TPR, and then visualizes the ROC curve. It also plots a reference diagonal line that represents random guessing. This function helps to visualize the classifier's performance on each class.

3. **get_decision_function_scores**

This function retrieves the decision function scores from the SVM model. These scores represent the confidence level of the model's predictions and are essential for generating the ROC curve. For multi-class classification, it returns a matrix of scores, with each column representing the decision scores for each class.