Face Detection

Parameters:

- img (numpy.ndarray): The input image in BGR format.
- sf (float): The scale factor for multi-scale detection.

Main Point of the Function:

The FaceDetection function performs face detection on the input image using Haar cascades. It detects faces of frontal view using one Haar cascade and side faces using another cascade. The function draws rectangles around detected faces and side faces on the image and returns the processed image along with the detection time and the number of faces found.

Description:

The FaceDetection function implements Haar cascade object detection for detecting faces in an input image. It follows the methodology proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features". The main points of the function's implementation are as follows:

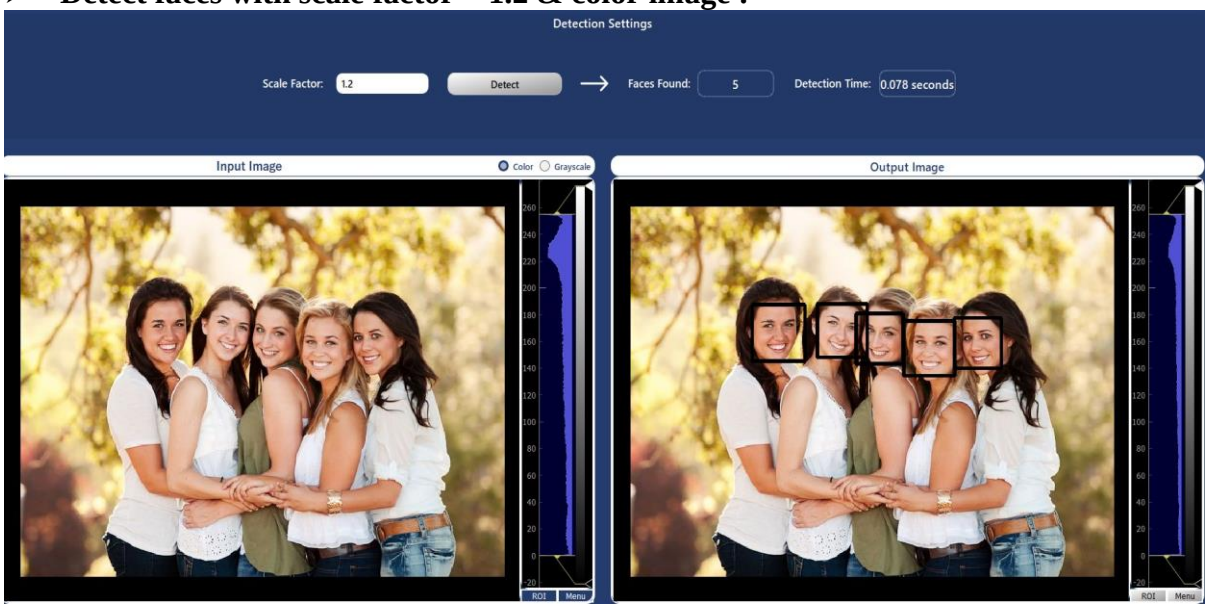
1. **Initialization:** The function initializes Haar cascade classifiers for detecting frontal and side faces using pre-trained XML files.
   - ✓ Observation: The use of pre-trained classifiers simplifies the implementation and leverages the extensive training data used to create these classifiers.
2. **Feature Extraction:** The function converts the input image to grayscale and applies Haar-like features to extract relevant features for face detection.
   - ✓ Observation: Haar-like features are efficient representations of facial characteristics and are used to differentiate between face and non-face regions.
3. **Face Detection:** The function applies the cascade classifier to detect faces in the grayscale image. It iterates through different stages of classifiers to classify each region of interest as a potential face.
   - ✓ Observation: The cascade of classifiers approach efficiently reduces the number of features evaluated per sub-window, leading to faster detection.
4. **Rectangle Drawing:** Upon detection of faces, the function draws rectangles around the detected faces on the input image.
   - ✓ Observation: Visualizing the detected faces provides a clear indication of the effectiveness of the face detection algorithm.
5. **Performance Analysis:** The function measures the time taken for face detection and returns it along with the number of faces found.
   - ✓ Observation: Analyzing the detection time and the number of faces found provides insights into the efficiency and accuracy of the face detection process.

Observations:

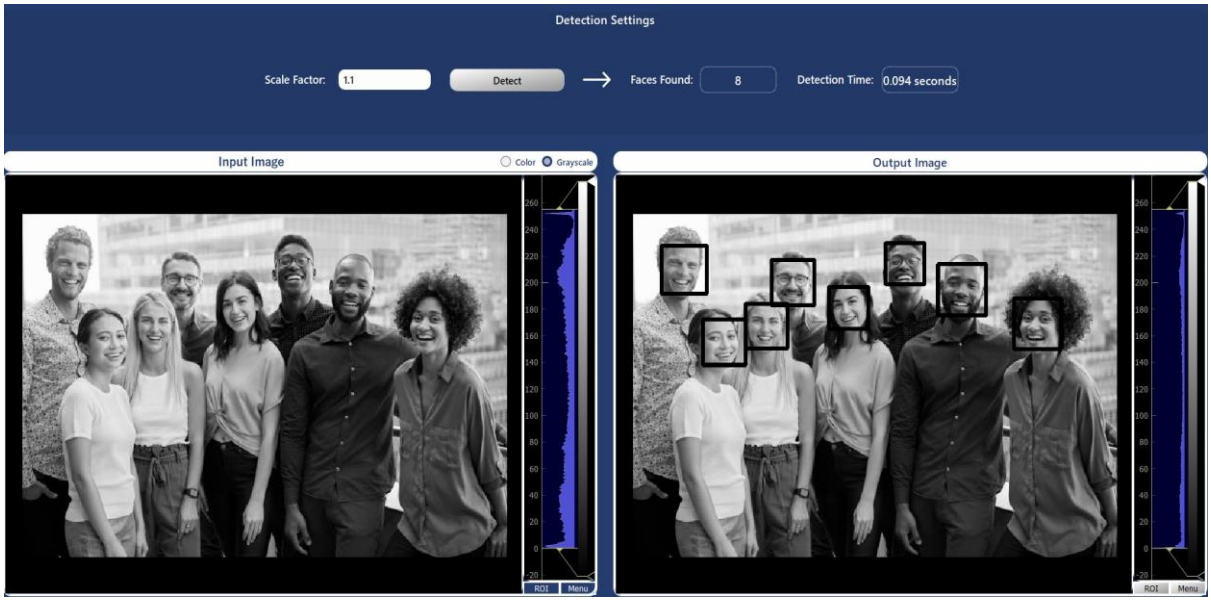
- The function relies on Haar cascade classifiers for face detection, which are pre-trained XML files. The accuracy and performance of face detection heavily depend on the quality and suitability of these classifiers.
- The scale factor parameter (sf) plays a crucial role in multi-scale face detection. Adjusting this parameter can affect the trade-off between detection speed and accuracy.
- The function handles the case when no frontal faces are detected by attempting to detect side faces. This enhances the robustness of face detection in various scenarios.
- The returned detection time provides insights into the computational efficiency of the face detection process. It can be influenced by factors such as image size, scale factor, and hardware capabilities.

Experiment:

- **Detect faces with scale factor = 1.2 & color image :**



- **Detect faces with scale factor = 1.1 & gray scale image :**



Face Recognition

[insert dataset name and extra info] The dataset contains 400 grayscale images of 40 subjects, 10 images each. It was divided into a 80:20 train to test ratio. Face Recognition achieved an accuracy of 86.25%.

1. face_recog_init:

Main Point:

This function sets the initial parameters required for face recognition. Initializing the array of training images, test images and test labels, setting the path for the dataset and calling the construct_data_matrix function that gets the train data and labels as well as the init_PCA function that calculates the eigenfaces that we will project the data on in later functions as well as the train_KNN function which prepares our KNN model to be used for prediction.

2. Init_PCA:

Main Point:

The init_PCA function performs Principal Component Analysis (PCA) on a set of images. PCA is a technique used to reduce the dimensionality of data while preserving most of its variance.

Steps:

1. Reshape Images: The function reshapes the images into 1D vectors and constructs a data matrix of shape (#images * #pixles => 112*92).
2. Compute Mean Image: Calculates the mean image vector by taking the mean of all pixel values across all images. This results a list of mean values of shape (#pixels * 1).
3. Subtract Mean Image: Subtracts the mean image from each image to center the data around zero.
4. Compute Covariance Matrix: Calculates the covariance matrix of the mean-centered data. This results a matrix of shape (#images * #images).
5. Compute Eigenvalues and Eigenvectors.
6. Sort Eigenvalues and Eigenvectors in descending order.

7. Compute the cumulative sum of eigenvalues and the ratio of each eigenvalue to the total sum.
8. Select the number of eigenvectors necessary to retain 90% of the variance, then select the corresponding top eigenvectors based on the determined number.
9. Project the mean-centered data onto the selected eigenvectors.
10. Compute the reduced data which is the result of further reducing the dimensionality of the data by representing each image using a smaller set of features derived from the PCA transformation matrix. This is done by multiplying the original data matrix with the transposed projected data matrix.

Observations:

- Unlike what we learned in class, covariance matrix is in shape of (#images * #images) as we need a computationally feasible method to determine eigenvectors by solving smaller matrix and taking linear combinations of the resulting vectors.
- In the final reduced data each row corresponds to one of the original images, and each column represents the contribution of each eigenvector (eigenface) to that image.

3. apply_PCA:

Main Point:

This function uses the eigenfaces in init_PCA and projects the data on them to produce the reduced_data of size (#samples, 111) instead of (#samples, 112*92).

Parameters:

- data (numpy 2D array): The data to be projected on the eigenfaces.

4. Predict_face:

Main Point:

This function is called when the predict button is pressed. It has the following tasks:

1. Takes the input image and its path, extracts its true label from the path.
2. Calls the recognize_face function on this image and receives the predicted class and the distance between the test image and its neighbors in the KNN model.
3. It sums these distances and compares it to an empirical threshold of 224000000 to determine if the image is of an outsider.
4. If it is not an outsider, it calls the choose_random_file function that gets another picture of the predicted class to allow the user to assess whether the input image looks similar to the predicted class and thus judge the model prediction.
5. It calls calc_accuracy function to calculate the accuracy of the model on the wholes test set.

5. recognise_face:

Parameters:

Test_img (numpy ndarray): The image we want to predict its class.

Returns:

Test_predicted_label (string): A string carrying a number from 1 to 40 representing the number of subjects (classes) in the dataset.

Total_dist (int): The sum of the distances between the image and its nearest neighbors.

Function:

This function uses the apply_PCA function to get the reduced data, it then sends it to the knn predict function and uses knn.neighbors() to get the distances between the image and its nearest neighbors, sums it and returns it to the calling function.

6. Calc_accuracy:

Returns:

acc (float): The accuracy of the model on the whole test set.

Function:

This function calls the create_test_datasets function to create the test image set and its labels. Then calls the recognize_face function on each test image and counts the correct prediction and divides it by the total number of images in the test set.