Assignment-3

CSL7360 - Computer Vision

Question-1: Eigen faces from scratch

Method-1

Procedure:

1.First for storing the images a function named get_filepaths is defined and is first initialized with an empty list called file_paths, which will be used to store the full file paths of all the files in the directory.

Fig.1 Input Images

2.Next loop is run to extract category labels from the filenames of image files in the training directory, and stores them in a list for use in classification models.

3. Then the images are flattened Flattened image array is then stored in the corresponding row of the training_images array.

4. After this I calculated the mean pixel value for each pixel by dividing the mean_face array by the total number of images in the training set. Finally, the mean_face array is flattened and reshaped into a 2D array of shape (height, width).

5.After finding mean image then normalization of each image in the training_images array by subtracting the mean face calculated. Then, for each image in training_images, it subtracts the mean face from the image using np. subtract, and stores the resulting normalized image in the corresponding row of normalised_training.

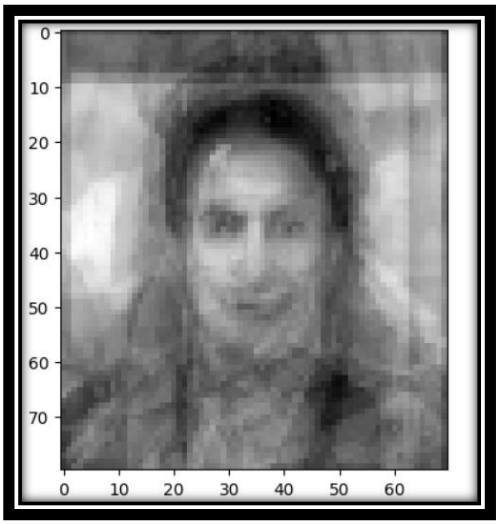

Fig.2 Mean Image

6.Then I computed the covariance matrix of the normalized training set and then computed the eigenvalues and eigenvectors of the covariance matrix, sorted them in descending order, and after normalization of the eigenvectors, and they were stored as eigen faces.

7.Then finally eigen faces were plotted as shown below

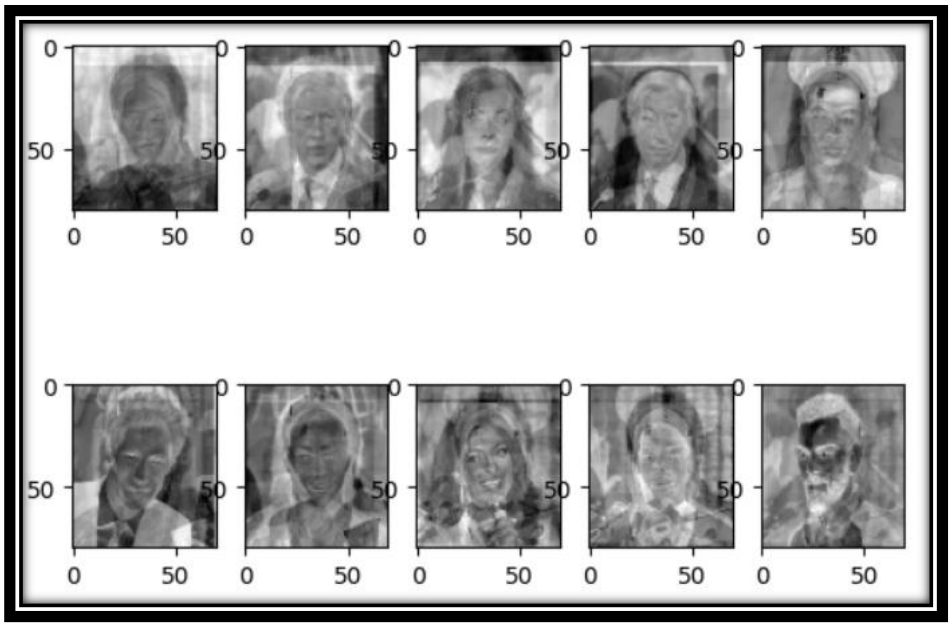

Fig.3 Eigen Faces

8.At last reconstruction of query image is done. Here I have read an unknown face image from a file, and flattened it into a vector. It then was normalized image was found the by subtracting image vector from the mean_face computed earlier. Next weights were computed. Then, I computed the Euclidean distance between the

unknown face's weight vector and the weight vectors of all the training faces using np. linalg.norm. Finally closest image was found and is played as shown below.

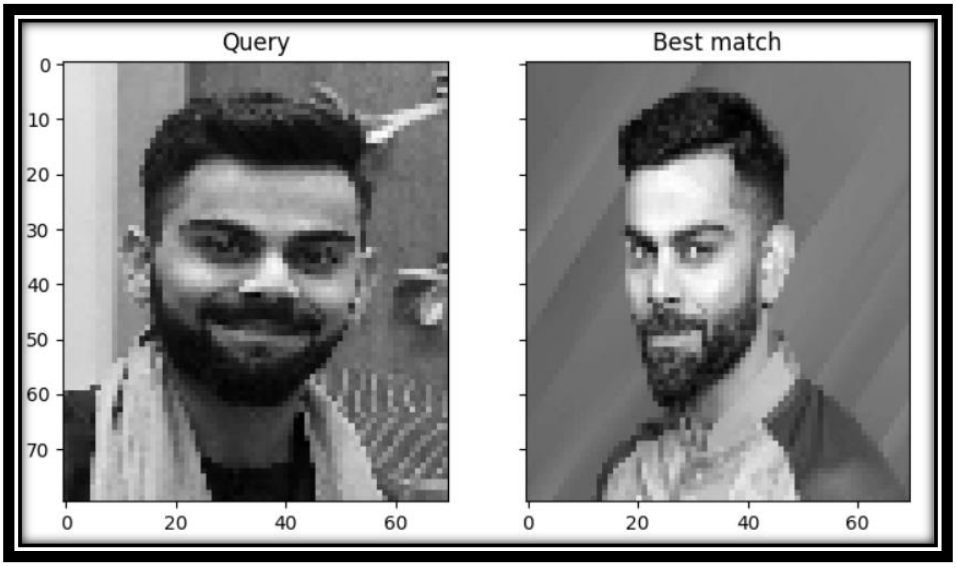

Fig.4 Query and Best match

METHOD-2

Procedure:

1.In this I have divided images in to three channels. Then I have computed the mean face for each of three groups of images (X_1, X_2, and X_3).

2.Then, I have subtracted the mean face of each group from every image in that group, effectively centring the data around the mean.

3. Then I have calculated the covariance matrix and eigenvalues/eigenvectors of a datasets. The mean of the dataset is subtracted from each data point before the covariance matrix is calculated. Then, the eigenvalues and eigenvectors, and then sorted the eigenvalues in ascending order.

4.Then I have selected the last 10 eigenvectors as the "component vectors" and the corresponding eigenvalues as the "component values". Then performed dimensionality reduction using principal component analysis (PCA). The component vectors can be used to project data onto a lower-dimensional subspace while preserving the most important information, as determined by the corresponding eigenvalues.

5.Then eigen faces were found and then displayed like

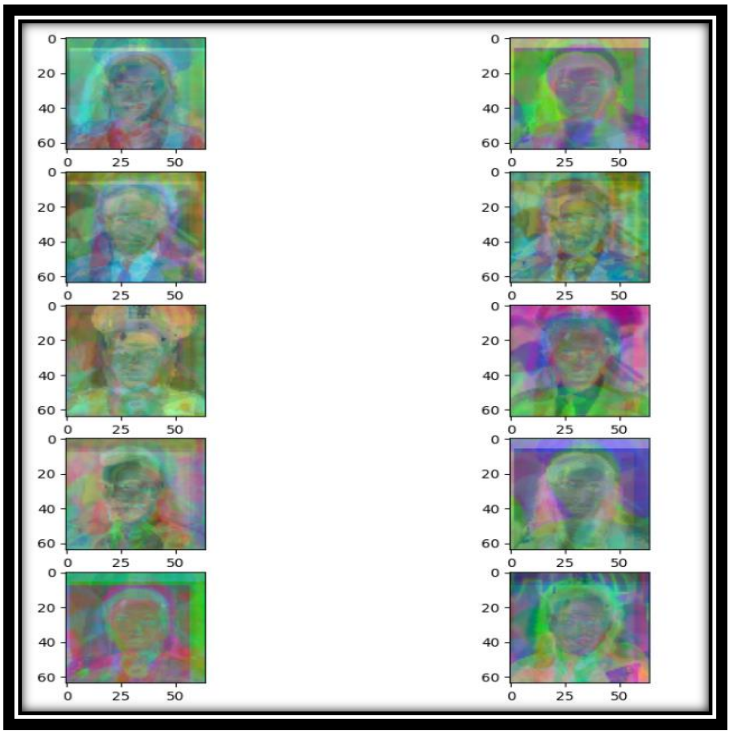

Fig.1 Eigen Faces

6. To reconstruct an original facial image using the k principal components, we first project the original image onto the k-dimensional subspace spanned by the top k (here k =5) eigenvectors. This is done by multiplying the original image vector by the transpose of the matrix containing the top k eigenvectors. The resulting projection coefficients represent the contribution of each principal component to the original image. We can then reconstruct the original image by reversing this process.

7.Here I have used k = {1, 50, 10, 100, 100} to reconstruct the original facial image using different numbers of principal components. For each value of k, the script projects the original image onto the k-dimensional subspace, reconstructs the image using the top k eigenvectors, and displays the reconstructed image in a grid of images.

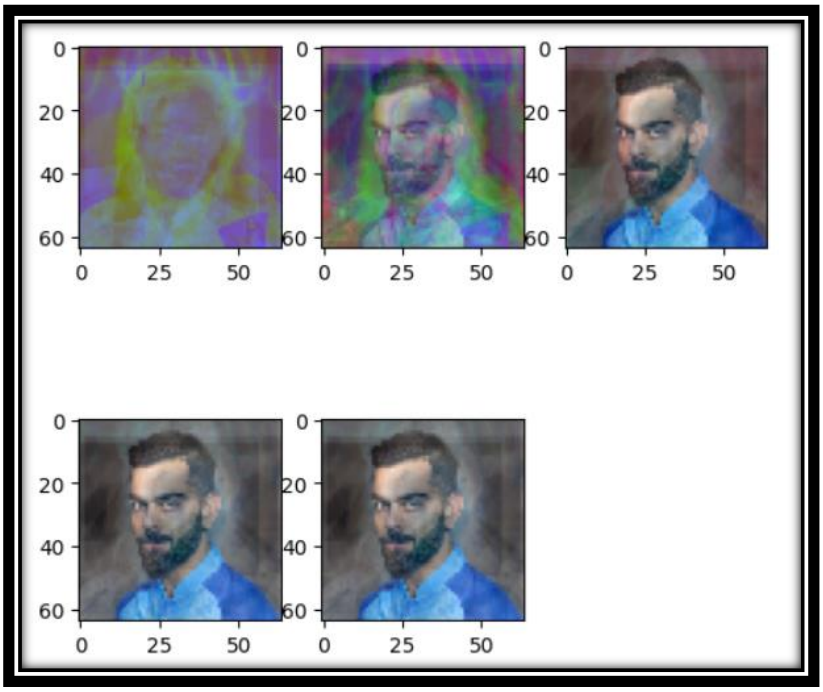

Fig.2 Top K faces

COLLAB LINK:

https://colab.research.google.com/drive/1XzPaY58kLxj5TV_eXY5eIK71Bcatbo2r?usp=share_link

OBSERVATION:

-In method 1 the reconstruction of query image was done based on Euclidean distance. Based on this the image was successfully retrieved.

-In method-2 I have observed that as the number of k increased the reconstruction of query image was also improved significantly.

Credits

https://github.com/vutsalsinghal/EigenFace

QUESTION-2: Visual Bow

Visual Bow (Bag-of-Words) is a popular method for image recognition and classification that is based on the idea of representing images as histograms of visual words. The technique is inspired by text-based Bow models, which count the occurrence of words in a document to represent it as a bag of words. In the case of image classification.

Visual Bow involves three main steps: feature extraction, feature quantization, and histogram generation. First, local features such as SIFT, SURF, or ORB are extracted from the images. Then, a clustering algorithm such as k-means is used to group similar features into a fixed number of clusters. Each cluster is considered as a visual word. Finally, the images are represented by a histogram of visual words where the count of each visual word is the number of local features in the image that belong to that cluster.

The advantage of the Visual Bow approach is that it provides a way to represent images using a fixed-size vector that can be used as input for machine learning algorithms such as SVM or random forests.

Procedure:

1.At first cifar dataset is downloaded and then preprocess_images function takes a set of input images and applies some pre-processing steps on them. Specifically, it converts the RGB images to grayscale by taking the mean over the colour channels, resizes each image to 32x32.

2.The pre-processed images are then stored in x_train_preprocessed and x_test_preprocessed, which correspond to the pre-processed training and test sets, respectively.

3. Then local features are extracted from pre-processed images using the SIFT detector. In this case, the SIFT algorithm is applied to each pre-processed image to extract the local features. The resulting features are stacked together to create a feature matrix for the training and test sets.

4. After that Minibatch K Means algorithm is used to cluster the local features extracted from the pre-processed training images. The algorithm is run with a specified number of clusters (n_clusters) set to 100, and a batch size of 1000.Then, a NearestNeighbors object is defined with n_neighbors set to 1000. If n_neighbors is greater than the number of features in the training set, n_neighbors is set to the

number of features in the training set. The Euclidean distance metric is used to compute the distances between the histograms.

6.Finally Testing is performed to retrieve the top 5 similar images. First After taking test image, I used k-means to cluster the local features into a fixed number of visual words. The MiniBatchKMeans function is used to speed up the clustering process by using mini-batches. After clustering, a histogram is computed for each image in the training set by counting the number of occurrences of each visual word in the image. The histogram is normalized to have unit length.

7.The retrieve_similar_images function takes a query histogram and a set of histograms for the training set images. It computes the Euclidean distance between the query histogram and each training set histogram, and returns the top-5 images with the smallest distances.

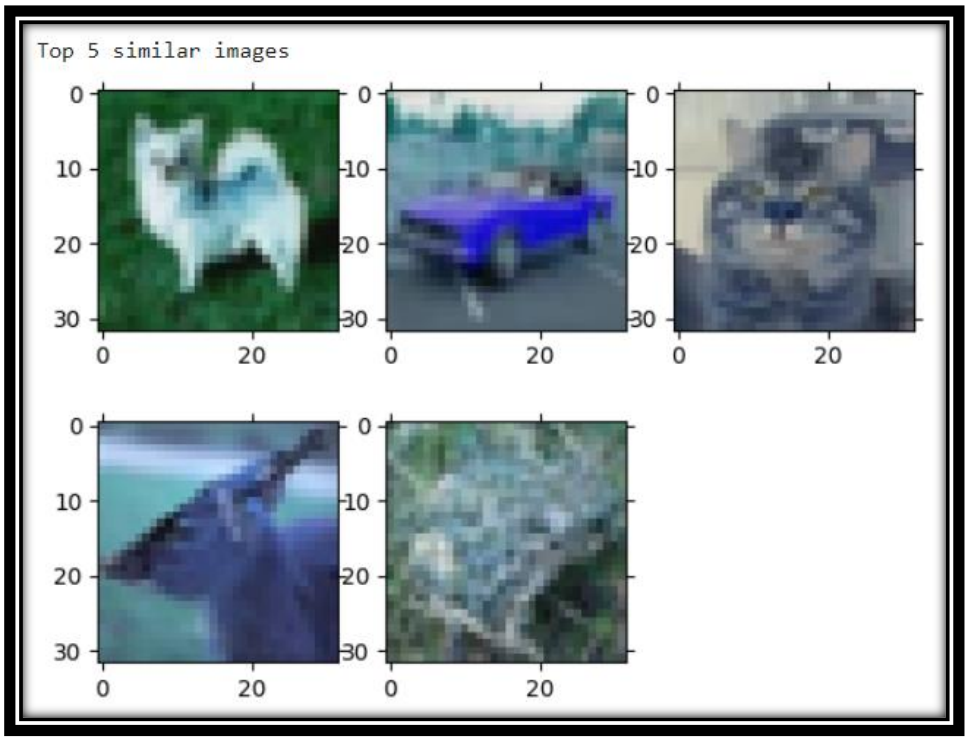

Fig.1 Top 5 Similar Images

8.Finally I have retrieved similar images to a given query image using a visual word histogram approach and evaluates the performance of the retrieval using precision-recall curve analysis.The retrieve_similar_images function takes a query image histogram and compares it with the histograms of a database of images to retrieve the most similar images using k-nearest neighbours (kNN).The precision_recall_curve function is used to compute the precision and recall values for different distance thresholds between the query image histogram and the database histograms. These values are stored in precisions and recalls.

9.The average_precision_score function is used to compute the area under the precision-recall curve, which is stored in average_precision. The average precision and recall are `0.10698634` and `0.3577889447`. The P-R curve is shown below

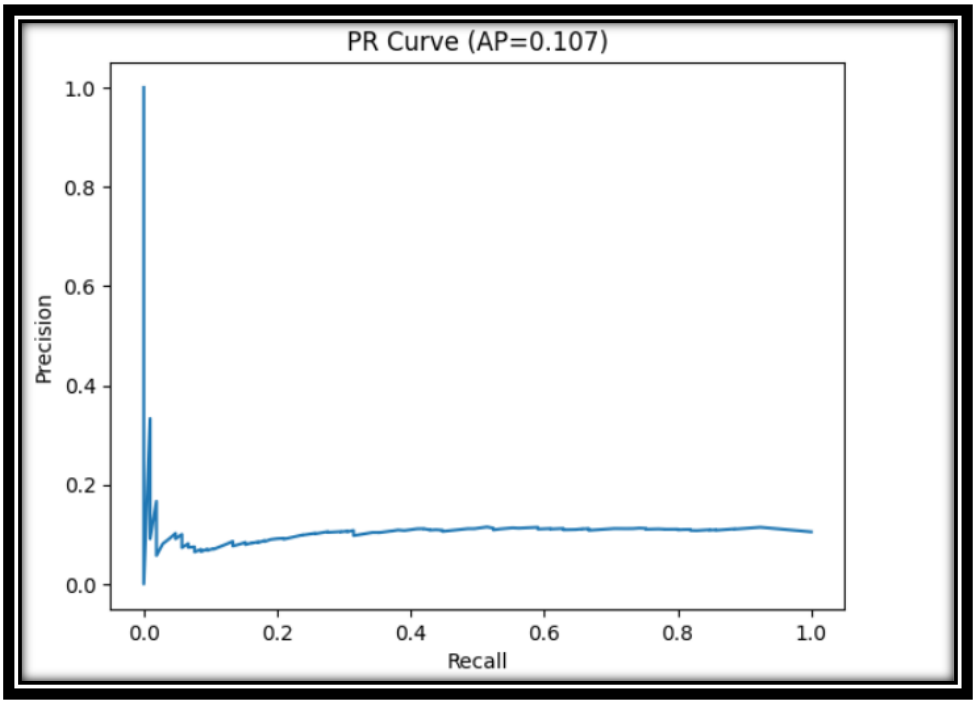

Fig.2 P-R Curve

COLLAB LINK:

https://colab.research.google.com/drive/1ztX85mXpsVlyL2aEsmVX6B7BlfJSrLnX?usp=share_link

OBSERVATION:

I have got low precision and recall because of following reasons

-**Insufficient number of training images:** The number of training images is small; the system was not be able to learn the visual features that are necessary for accurate retrieval.

-**Poor quality of training images:** The training images became poor quality after cropping. And They were not containing sufficient variation in terms of the objects or scenes depicted (e.g., Colour variation of deer and non-deer was almost similar), thus system may not be able to learn robust features.

QUESTION-3: Viola Jones's face detection

STEPS:

Viola-Jones face detection is a popular object detection algorithm used for detecting faces in images. The algorithm consists of the following steps:

1.Haar feature selection: In this step, a set of Haar-like features are selected based on their ability to distinguish between faces and non-faces. Haar features are rectangular features that consist of the difference of sums of pixel values in two adjacent rectangular regions.

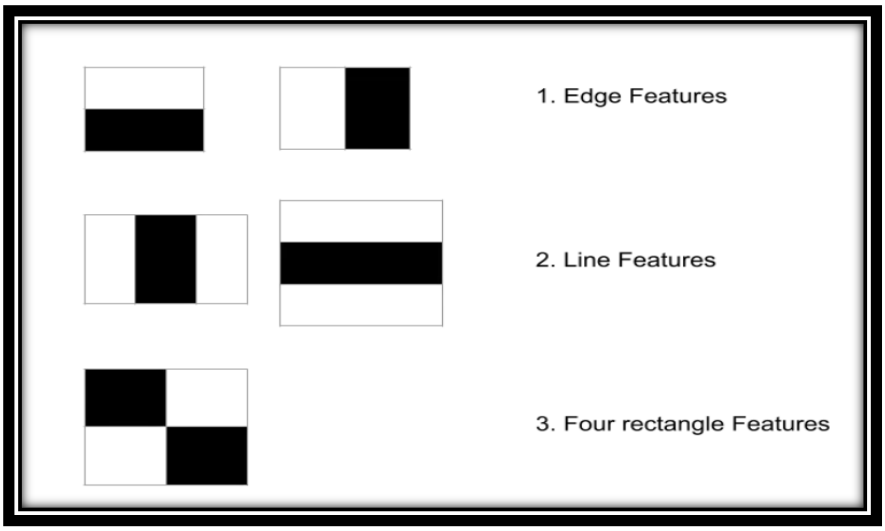

Fig.1 Haar Features

2.Integral Image calculation: Integral images are pre-calculated for the input image. Integral image at a pixel (x, y) is the sum of all pixels above and to the left of (x, y).

3.Adaboost training: In this step, Adaboost algorithm is applied to train a classifier using the selected Haar features. Adaboost is a machine learning algorithm that selects a subset of Haar features and combines them to form a strong classifier that can accurately distinguish between faces and non-faces.

In In the Adaboost algorithm, the first predictor is usually a weak learner. Each instance weight w(i) is initially set to 1/m. A first predictor is trained and its weighted error rate r1 is computed on the training set

$$r_j = \sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}$$

The predictor's weight j is then computed using the formulae given below. The more accurate the predictor is, the higher its weight will be. If it is just guessing randomly, then its weight will be close to zero. However, most often, it is wrong and its weight will be negative.

Next the instance weights are updated using the formula provided below in order to boost the misclassified instances

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & if\ \widehat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & if\ \widehat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

Then all predictors are normalized. Finally, a new predictor is trained using the updated weights, and the whole process is repeated until the desired number of predictors is reached which is specified by the user.

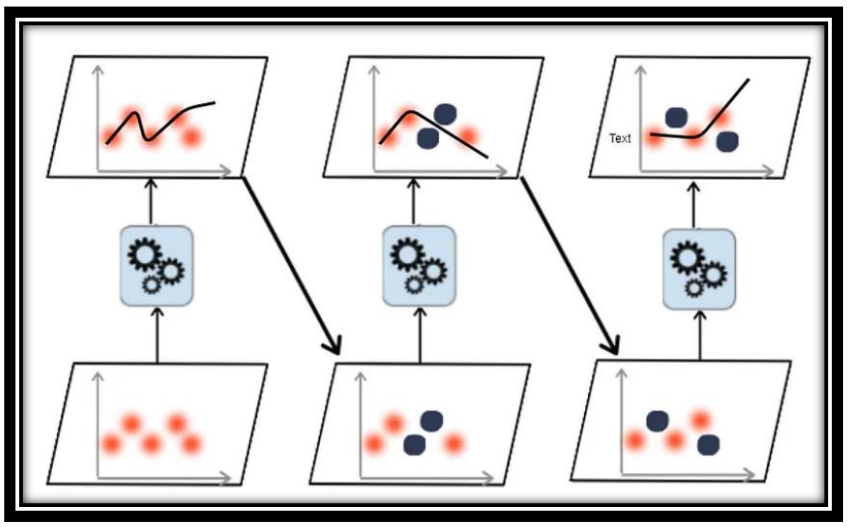

Fig.2 Ada Booster

4.Cascade classifier: The trained classifier is then used to detect faces in the input image. A cascade classifier is used to efficiently classify sub-windows of the input image. The cascade classifier consists of multiple stages, each of which consists of a set of weak classifiers. At each stage, a sub-window is classified as non-face if it fails any of the weak classifiers, and it is passed to the next stage if it passes all the weak classifiers.

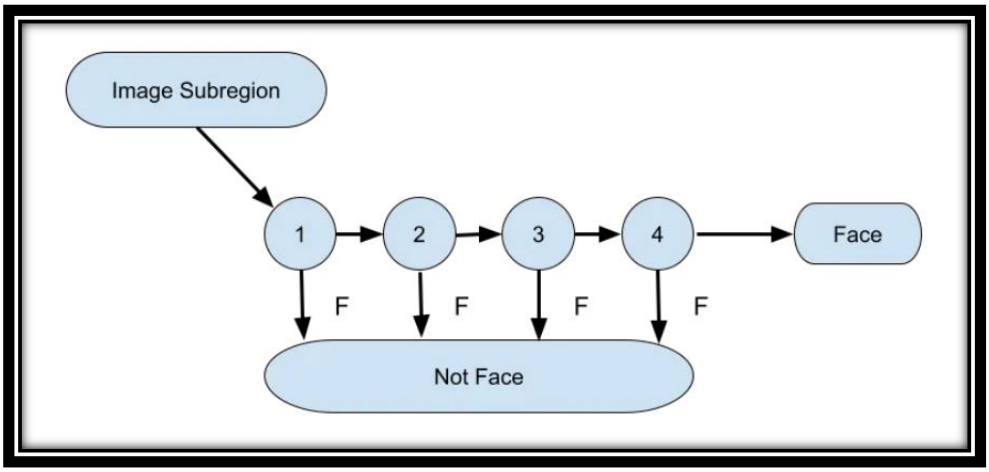

Fig.2 Cascade Classifier

5.Face Detection: Once the cascading classifiers are trained, they can be used to detect faces in new images. To do this, the input image is scanned using a sliding window technique, where a window of a fixed size is moved across the image. At each position of the window, the Haar-like features are computed, and the weak classifiers are evaluated. If a region of the image passes all the cascading classifiers, it is considered to contain a face. The size and position of the detected face can be estimated based on the position and size of the sliding window.

6.Non-Maximum Suppression: The final step involves removing overlapping detections and selecting the most likely face candidate. Non-maximum suppression is used to remove overlapping detections and select the face candidate with the highest confidence score.

That's a high-level overview of the Viola-Jones face detection algorithm. It is a powerful technique that has been used in many applications, including face recognition, emotion detection etc.

QUESTION-4: Sliding window object detection using HOG

HOG (Histogram of Oriented Gradients) is a feature extraction technique used in computer vision and image processing. HOG works by dividing the image into small cells and computing the gradient orientation and magnitude for each pixel in the cell. These gradient values are then used to create a histogram of gradient orientations over the cell. The cell histograms are then normalized and concatenated to form a feature vector that describes the local gradient structure of the image. The idea behind HOG is to describe the gradient of the image in a local region, which can be used to represent the local appearance and shape of an object

Method-1

Procedure:

1. First I have Loaded deer and non-deer images from their respective directories using os library.

2.Then I have Pre-processed each image patch (e.g., resize, normalize, grayscale) and computed HOG (Histogram of Oriented Gradients) features for each patch using skimage.feature.hog function.

3.I have Splitted the data into training and testing sets using train_test_split function from sklearn.model_selection.

4.Then I have trained a SVM (Support Vector Machine) classifier using the training set and the HOG features using sklearn.svm.LinearSVC.

5.Evaluation of the classifier's performance using the testing set is done and I printed the accuracy score using svm.

Testing:

1. First, the test image is loaded using cv2.imread and resized to 200x200 pixels. An empty list called detected_deer_boxes is initialized to store the detected deer bounding boxes. The trained SVM classifier is loaded using joblib.load.

2.Here sliding window approach is used to detect deer in the image.

3. The detected deer bounding boxes are stored in a list. After all the windows have been classified, non-maximum suppression is applied to remove overlapping bounding boxes using the cv2.groupRectangles function.

4.Finally, the detected bounding boxes are drawn on the test image using cv2.rectangle, and the image is displayed using cv2_imshow.Result is shown below.

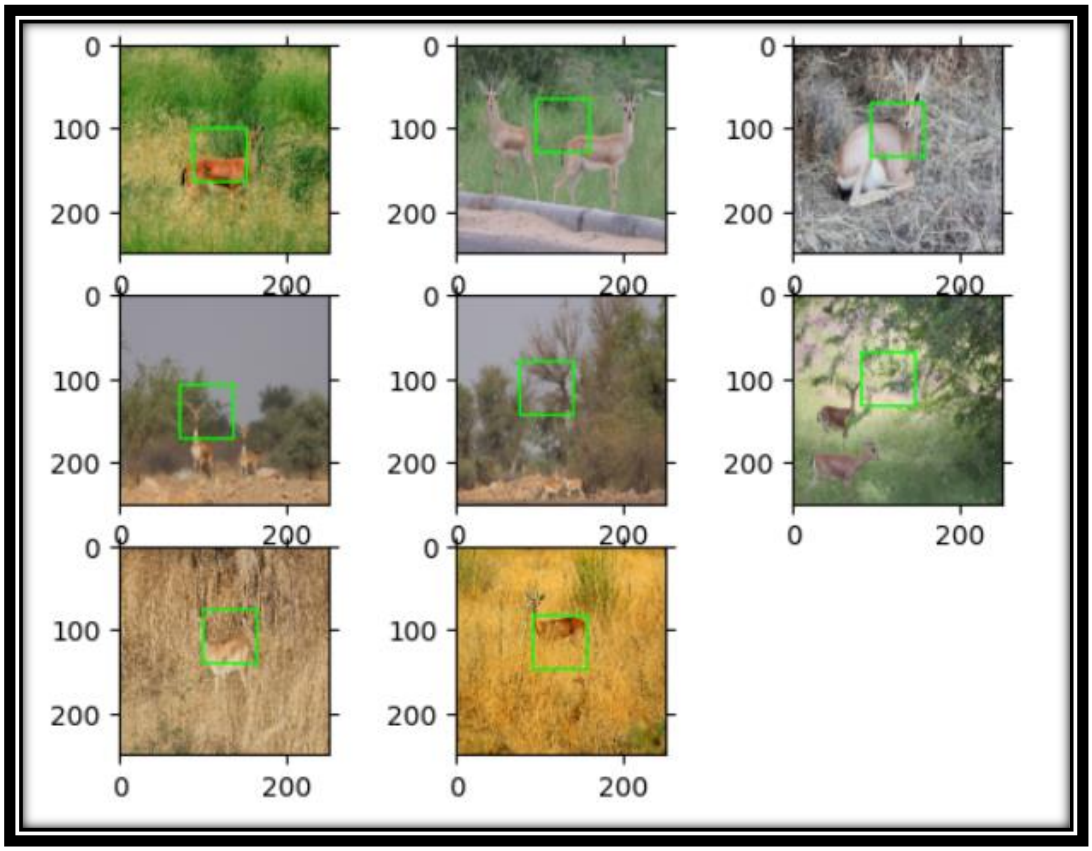

Fig.1 Resultant Images

Method-2

1. First I have Loaded deer and non-deer images from their respective directories using os library.

2. Then, the HOG parameters are specified, including the number of orientations, the size of the cells and blocks, and the normalization method.

3.Then I have Pre-processed each image patch (e.g., resize, normalize, grayscale) and computed HOG (Histogram of Oriented Gradients) features for each patch using skimage.feature.hog function.

3.I have Splitted the data into training and testing sets using train_test_split function from sklearn.model_selection.

4.Then I have trained a SVM (Support Vector Machine) classifier using the training set and the HOG features using sklearn.svm.LinearSVC.

5. The trained model is saved to a file using the joblib.dump function from the joblib module.

6.Evaluation of the classifier's performance using the testing set is done and I printed the accuracy score using svm.

Testing:

1. First, the test image is loaded using cv2.imread and resized to 200x200 pixels. An empty list called detected_deer_boxes is initialized to store the detected deer bounding boxes. The trained SVM classifier is loaded using joblib.load.

2.Here sliding window approach is used to detect deer in the image.

3. The detected deer bounding boxes are stored in a list. After all the windows have been classified, non-maximum suppression is applied to remove overlapping bounding boxes using the cv2.groupRectangles function.

4.Finally, the detected bounding boxes are drawn on the test image using cv2.rectangle, and the image is displayed using cv2_imshow.Result is shown below.

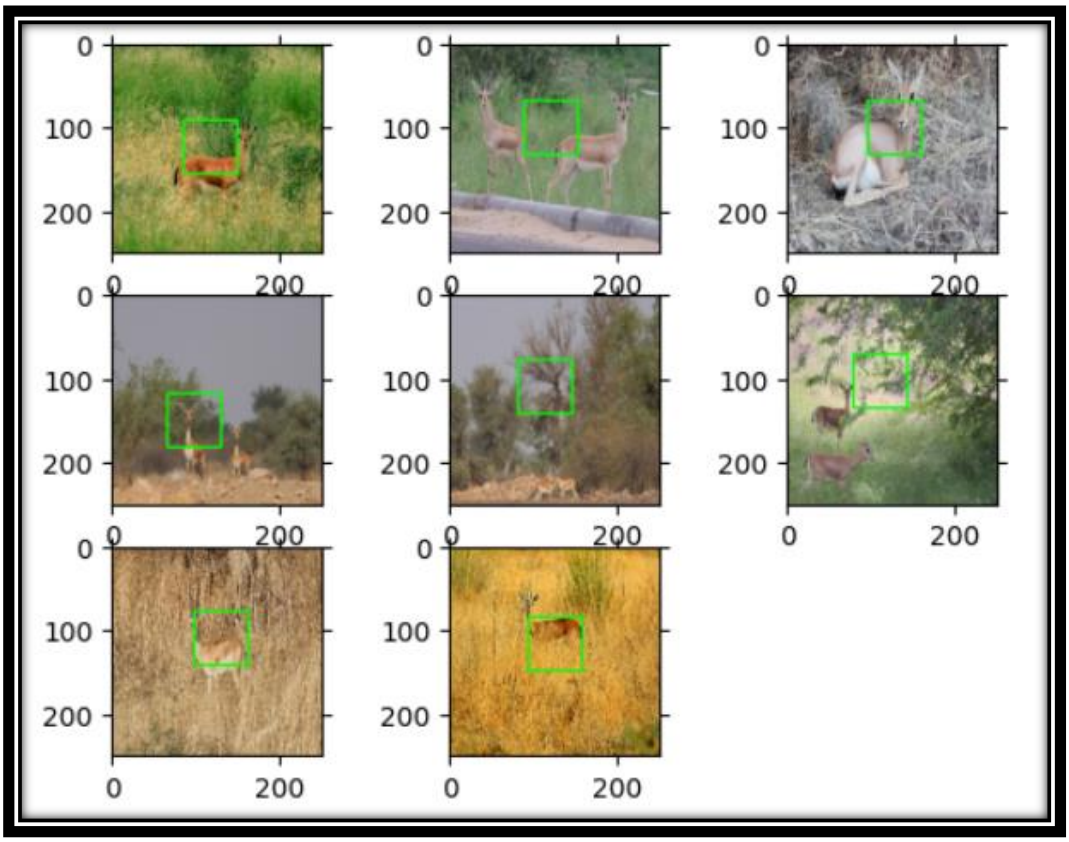

Fig.2 Resultant Images

COLLAB LINK:

https://colab.research.google.com/drive/1SZ9mp6Q0xtVQPEXcSqWi1BiplPxgqOoa?usp=share_link

OBSERVATION:

-In method 1 Accuracy of the model was 75%. In the testing phase it was not able to correctly classify all the test images because of hog of nondeer region was matching with hog of deer regions.

-In method 2 Accuracy of the model was 100%. In the testing phase it was not able to correctly classify all the test images because of inadequate amount of input data the model was overfitted. Therefore, it failed to predict the deer images.