COMPUTER VISION

Assignment 2A Report

M22MA003

Question 1 : Eigen faces from scratch: Use the subset of the LFW dataset provided with this assignment, include 1 face photograph of your favorite Indian sportsperson from the web to augment the dataset, and implement Eigen face recognition from scratch. You may use the PCA library, but other functionalities should be originally written. Show top-K Eigen's faces of the favorite Indian sportsperson you considered for different values of K. The report should also contain a detailed quantitative and qualitative analysis. (Use provided data as train set and a test set will be provided separately).

Solution 1 :-

1. Import the necessary libraries including PCA from sklearn.decomposition, cv2 for image processing, matplotlib for visualization, numpy for numerical operations, cv2_imshow for displaying images in Colab, os for file operations, and glob for file path manipulation.
2. Mount the Google Drive to access the input image files from the data.zip file using drive.mount().
3. Define the file paths for the input images, the test image, and the number of top eigenfaces to consider.
4. Define functions to read and preprocess the input training images (get_train_images()), compute the mean of the training images (get_mean_train()), plot the top eigenfaces (show_eigen_face()), preprocess the test image and compute the difference array (get_mean_test()), and plot the reconstructed images with varying numbers of eigenfaces (plot_ef()).
5. Load the training images using get_train_images() function and display the cricketer image and the mean image using cv2_imshow().
6. Compute the difference array between the training images and the mean image using the get_mean_train() function.
7. Compute the covariance matrix of the difference array using np.dot() and np.T for transpose.
8. Create an instance of PCA with the desired number of components (n_components=100) and fit it to the covariance matrix using obj_pca.fit().

9. Extract the top eigenfaces from the PCA components using slicing (obj_pca.components_[:pca_count]) and reshape them to the original image size using ef = top_ef.reshape((11, 100,100)).
10. Display a specific eigenface using plt.imshow().
11. Call the show_eigen_face() function to plot the top eigenfaces.
12. Preprocess the test image and compute the difference array using get_mean_test() function.
13. Call the plot_ef() function to plot the reconstructed images with varying numbers of eigenfaces.

Results:-

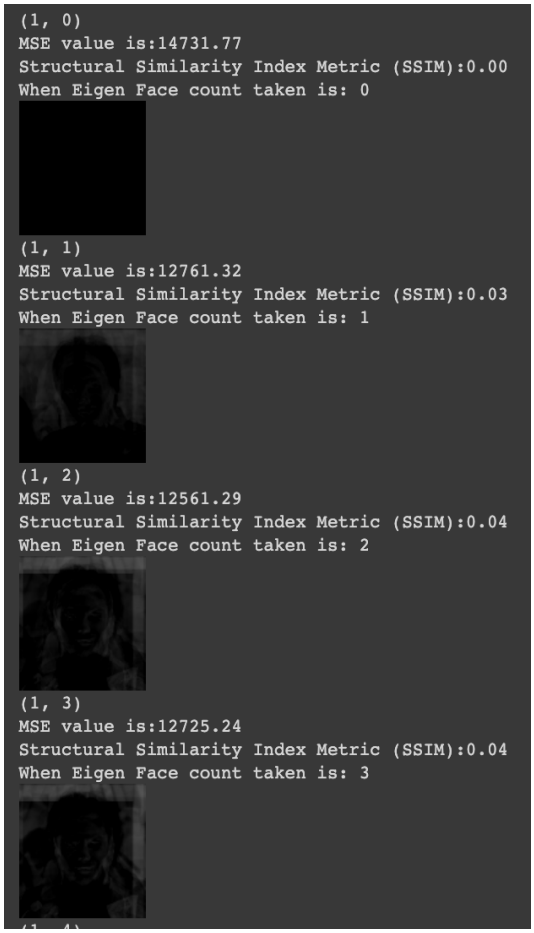
Cricketer Image:-

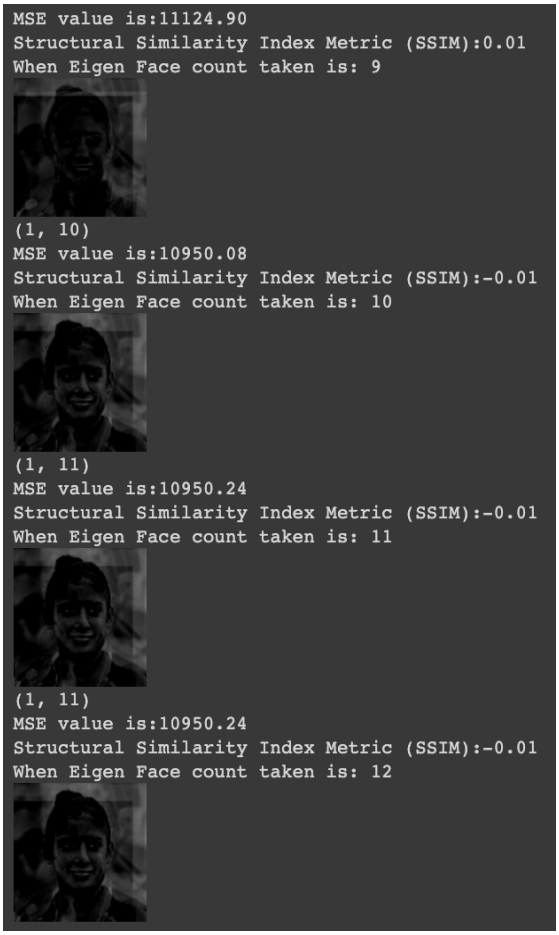
Mean Image of the input images:-

One of the reshaped Eigen's faces out of the eleven Eigen's faces created :-

Top 7 Eigenfaces

Some more examples:-





Observations : As the value of eigen's faces is increased, MSE is decreasing and furthermore, the picture is relatively clear. If the rebuilt face closely resembles the actual test face, it proves that eigenfaces are useful for facial recognition since they properly reflect facial traits. The quality of the eigenfaces in this instance, however, could not be adequate because of things like the test faces not being centered, faces being in different places in the photos, the player's face being tiny, or problems with image depth. These elements may have produced eigenfaces that were less ideal, which would have impacted the outcomes.

Colab Link Question1

Question 2: Visual BoW Develop an Image Search Engine for CIFAR-10 that takes the image as a query and retrieves top-5 similar images using Visual BoW. Report Precision, Recall, and AP. Draw the P-R curve. Write down each step of implementation in clear and precise terms with an appropriate illustration.

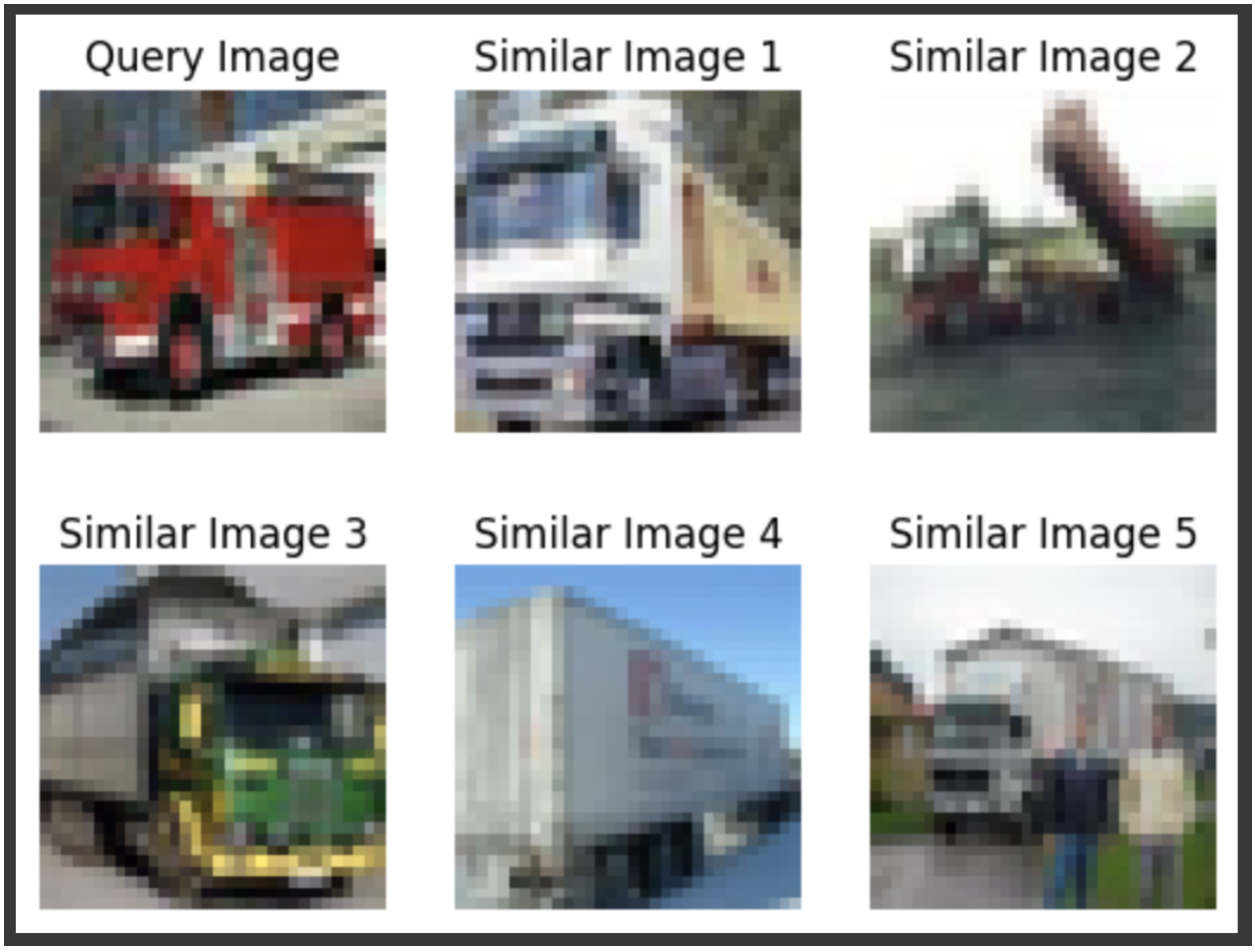
Solution 2:-

1. Imports necessary libraries, including OpenCV (cv2), Torch (torch), NumPy (np), and machine learning modules from scikit-learn (sklearn).
2. Defines a class-to-label dictionary for mapping class indices to class names in the CIFAR-10 dataset.
3. Defines a function extract_sift_features() that extracts Scale-Invariant Feature Transform (SIFT) features from the training images. SIFT is a popular local feature extraction technique that is used to detect and describe keypoints in images. The SIFT features are computed using OpenCV's cv2.SIFT_create() function, and are stored in a list of descriptor arrays.
4. Defines a function extract_sift_features_test() that extracts SIFT features from the test images, and uses a pre-trained k-means clustering model (kmeans) and a pre-trained Support Vector Machine (SVM) classifier (svm_cl) to predict the class labels of the test images. The SIFT features are first quantized to visual words using the k-means clustering model, and then a histogram of visual word occurrences is computed for each image. The histogram is used as a feature vector for the SVM classifier, which predicts the class labels of the test images.
5. Defines a function plot_graph() that plots a Precision-Recall curve for evaluating the performance of the BoW model.
6. Loads the CIFAR-10 dataset using Keras' cifar10.load_data() function, and preprocesses the images by flattening them and converting them to grayscale.
7. Extracts SIFT features from the training and test images using the extract_sift_features() function, and concatenates the descriptor arrays to obtain descriptors_train and descriptors_test.
8. Performs k-means clustering on the training image features (descriptors_train) to create a visual vocabulary of k visual words using scikit-learn's KMeans class.
9. Defines a function compute_bow_histogram() that computes the BoW histograms for a set of image features (image_features) using a pre-trained visual vocabulary (visual_vocabulary). The function first quantizes the image features to visual words using the k-means clustering model, and then computes a histogram of visual word occurrences.

10. Computes the BoW histograms for the training and test image features using the compute_bow_histogram() function, and stores the histograms in train_histograms and test_histograms, respectively.
11. Trains a Support Vector Machine (SVM) classifier on the training histograms using scikit-learn's SVC class, and stores the trained classifier in the variable svm_cl.
12. Evaluates the performance of the BoW model by computing precision, recall, and average precision scores, and plots a Precision-Recall curve using the plot_graph() function.

Results:-

Query Image Search Outcomes:-



All the similar images obtained are of same class as the queried image ,i.e., Truck.

Histogram of the Key descriptors of the Queried image :-

```
[[  0.  12.  13.   0.   0.   0.   0.   0.   1.  24.  62.  40.   1.   0.
    0.   0.  18.   3.   4.  17.   0.   0.   0.   2.   3.   0.   0.   0.
    0.   0.   0.   0.   2.  75.  93.   0.   0.   0.   1.   2.  58. 135.
  159. 121.   1.   0.   0.  13. 159. 116.  51.  45.   1.   0.   0.  20.
   60.  53.   7.   0.   0.   0.   0.   2.   7.  27.  19.   0.   0.   3.
   55.  40. 110.  47.  62.   3.   0.   0.  24. 159. 159.  88.  38.   1.
    0.   4.  19. 130.   9.  35.  20.   0.   0.   1.   1.   0.   1.   0.
    0.   0.   0.   1.  16.  11.  42.   0.   0.   0.   0.   0.  10. 116.
   18.   0.   1.   0.   0.   6.  57.  81.   0.   0.   1.   0.   0.   1.
    4.   0.]]
```

Comparing label of the prediction vs the ground truth label:-

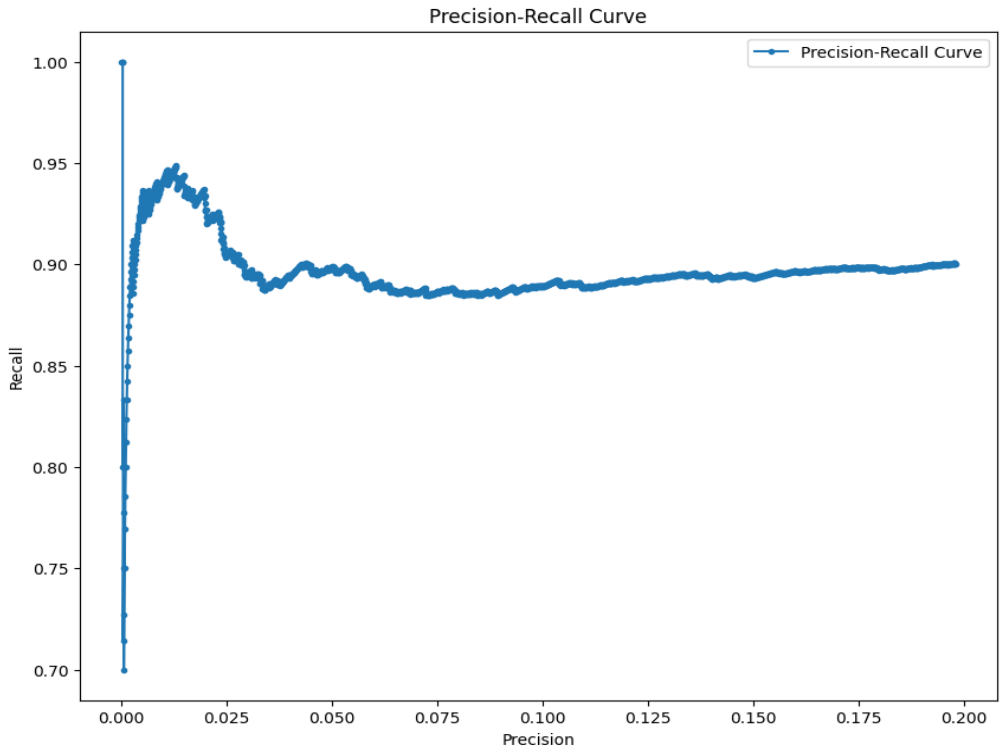
```
Label Predicted: truck
Actual class: truck
```

Test Data Results:-

```
Accuracy is 24.867320 | Precision is 0.971380
Recall is 0.971380 | F1_score is 36.922524 % | AP is 24.867320
```

Precision Recall Plot :-

We can observe that the **minimum recall is 0.88** and **average recall is 0.91**



[Colab Link Question2]

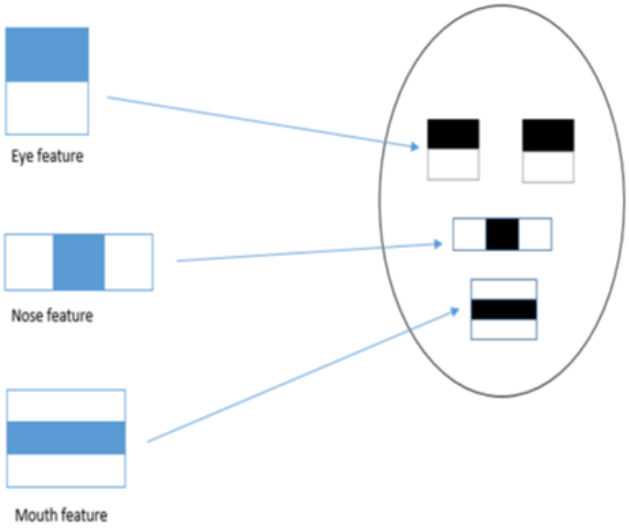
Question 3 : Viola Jones Face detection: Write down Viola Jones's face detection steps in detail.

Solution 3 :-

- **Haar-like feature extraction**

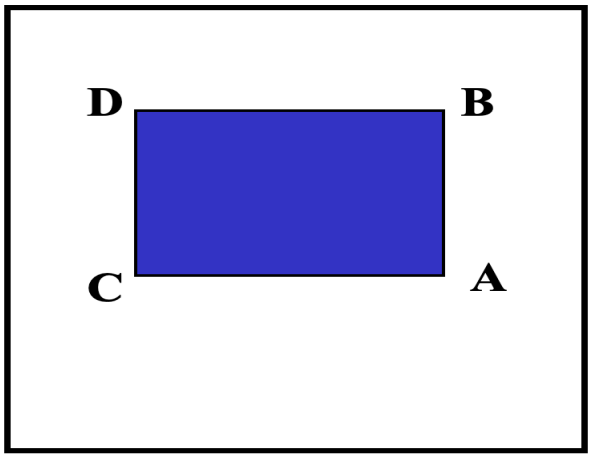
Haar-like features are used as the basic building blocks of the Haar cascade classifier, a machine learning algorithm used for object detection.

Haar-like features are designed to differentiate between areas of an image with different intensity levels, allowing them to capture local contrast and texture information.



The local look of face features like the eyes, nose, and mouth are represented by the algorithm using Haar-like characteristics. Simple rectangular filters that may be applied to the picture at various sizes and locations are known as **Haar-like features extraction.**

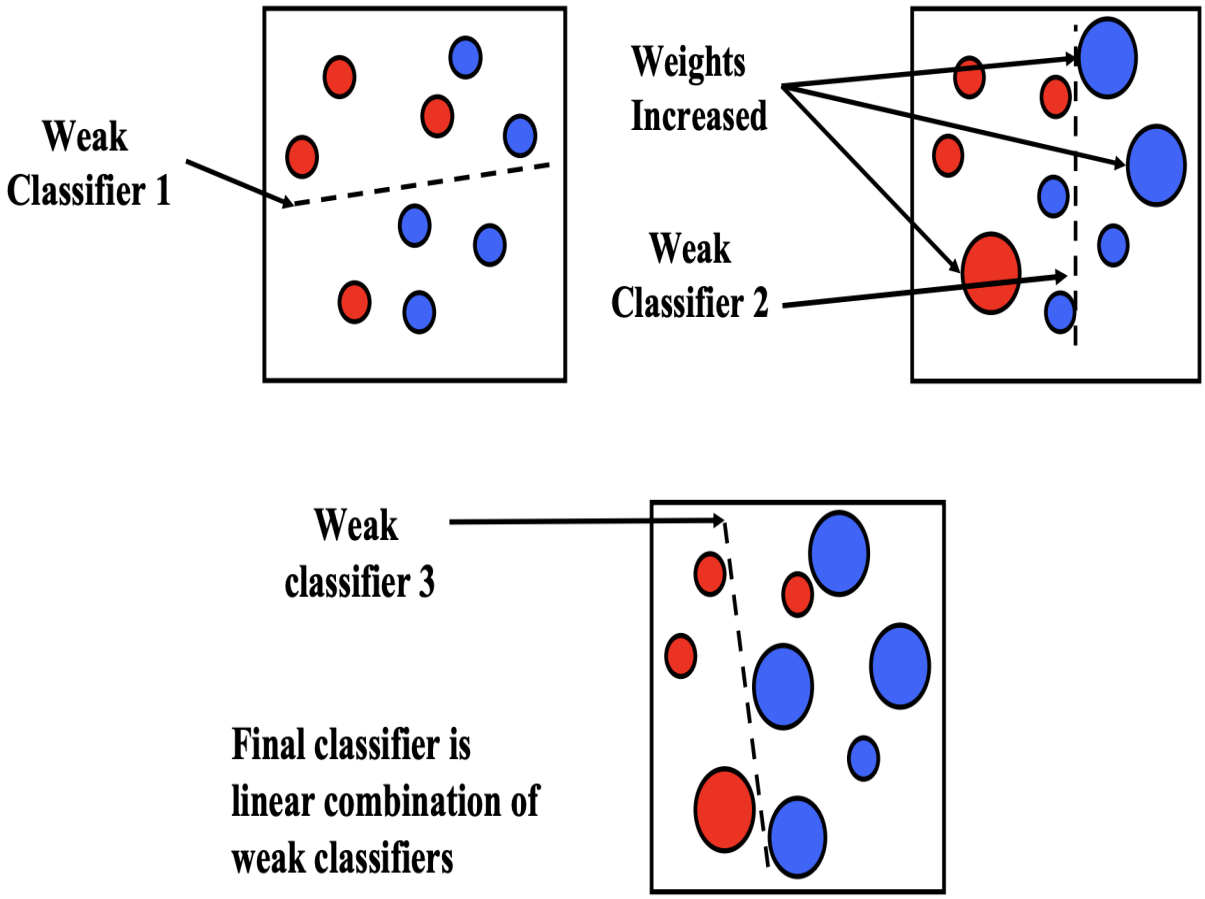
- **Calculating Integral Image**



An **integral image** is computed from the input picture to expedite the calculation of Haar-like characteristics. The computation of Haar-like features at any size and position in constant time is made possible by the integral picture, which depicts the sum of pixel intensities along rows and columns.

D — the Integral Image's at-position pixel intensity sum (x, y),
C — the Integral Image's pixel intensities at the place directly above, added together (x, y),
B — the total of the Integral Image's pixel intensities at the location directly to the left of (x, y)
A — the input image's input pixel's matching intensity

- **AdaBoost Training**



A powerful classifier for face identification is trained using a machine learning method called Adaboost. Adaboost chooses the ideal Haar-like feature and threshold repeatedly in order to

distinguish between positive (face) and negative (non-facial) data. To reduce classification error, the classifier has been trained.

Final Output: The Viola-Jones face detection method produces a series of rectangular bounding boxes as its final result, each of which represents a part of the face that was recognised in the input picture. For reasons of visualization or further processing, these bounding boxes can be employed.

Question 4: Sliding window object detection using HOG:. You are given a few deer train images with this assignment. Manually crop them to find tight bounding boxes for Deer and also obtain some non-deer image patches of different sizes and aspect ratios. Compute HOG features for deer and non- deer image patches and build an SVM classifier to classify deer vs non-deer. Now, implement a sliding window object detection to find out deer in the test images. Write down each step in the report. Also, objectively evaluate your detection performance.

Solution 4 :-

Deer detection using a Support Vector Machine (SVM) classifier and Histogram of Oriented Gradients (HOG) features:-

Deer photos are cropped manually and model is trained on the same images. The precision and f1 score is calculated on Test images containing both deer and non deer images.
Links : Cropped_Deer_Images Non-Deer Images Test_data_images_link

1. The necessary libraries are imported, including OpenCV (cv2), NumPy (np), scikit-learn (svm, cross_val_score), and other relevant libraries for image processing and evaluation.
2. Images are loaded from specified directories using OpenCV's imread function and stored in the variables test_data, deer_data, and ND_data. These represent the test images containing deer, the training images of deer, and the training images of non-deer.
3. A function called "findHOG" is defined to compute HOG features from an input image. HOG features are widely used for object detection in computer vision tasks.
4. The deer_hog_features and nondeer_hog_features arrays are concatenated along with their corresponding labels (1 for deer and 0 for non-deer) to create the feature matrix (X) and label vector (y) for training the SVM classifier. The data is shuffled to avoid any bias during training.
5. An SVM classifier with a linear kernel and probability estimation is trained using the cross_val_score function from scikit-learn. The cross-validation score is printed as the accuracy of the model.
6. HOG features are computed for the test images containing deer using the "findHOG" function, and the resulting features are stored in the testdeer_features array.
7. The SVM classifier is used to predict the labels (1 for deer) of the test images based on their HOG features. The precision score and F1-score of the model are calculated and printed.
8. Sliding window technique is used to detect deer in each test image. For each window, the HOG features are computed and fed into the trained SVM classifier for prediction. If the predicted label is deer and the predicted probability is above a threshold (p), the bounding box coordinates are stored.
9. The bounding box coordinates are used to draw bounding boxes around the detected deer objects in the test images using OpenCV's rectangle function.

Results:-

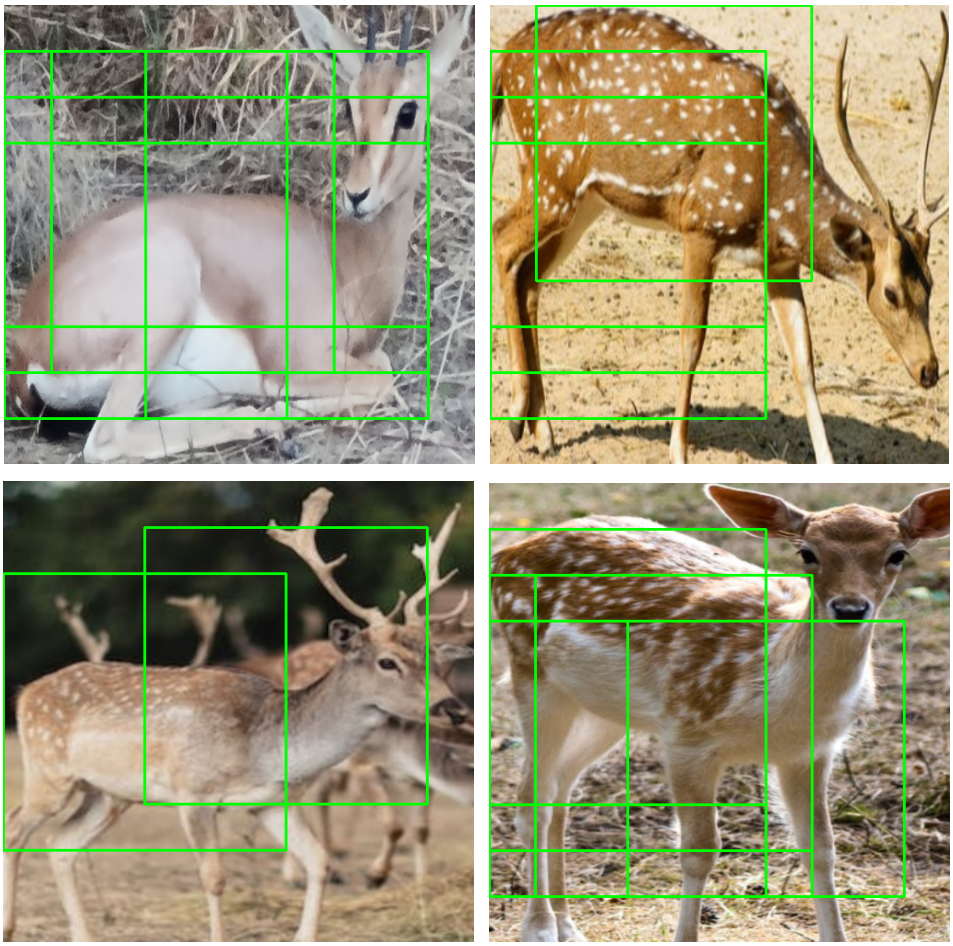
Training the SVM Model on Train data. Accuracy obtained is:

```
Accuracy:  0.76190
```

Result of other metrics on Test data :

```
precision_score of the model is:::::
1.0
f1_score  of the model is:::::
0.7586206896551725
```

Applying Sliding Window concept to detect the Deer in deer images and show no result in non-deer images:-



Result on non deer images :-

[Colab Notebook Question4]