M22MA002
CV Assignment 3 REPORT

Q1.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)

Ans:-

Dataset details:

LFW dataset is the labelled faces in the wild .It is the public dataset available for the face recognition developed by the researchers of the university of Massachusetts.

Here we have taken 10 faces of the LFW dataset as mentioned in the question and also 1 face of the indian sportsperson **Mr. Mahendra Singh Dhoni** and after that we have implemented the eigen face recognition algorithm from scratch.

Procedures :-

The steps which I have followed are as below:

1. First of all we have imported various libraries as stated in the code.
2. Then mounted the drive in colab.
3. Upload the new modified image zipped file to the particular google drive.
4. Then we unzipped the folder(10+1image) which was uploaded in the google drive which was mounted earlier.
5. After that we have accessed every 11 image one by one, did some image preprocessing over it like resized it to 100,100 ,converted to the grayscale,and converted to 1X10000 then appended all in the list train_set.
6. Next we have taken the mean of all the data and reshaped it to 100X100 and displayed it.
7. After that we have find difference of the train_set with the mean image obtained .
8. Using it we have obtained Covariance which is the inner dot product of difference and transpose of the difference image.
9. Now we have applied inbuilt libraries of PCA over it to get the eigen-faces /eigen vector
10. Next we have displayed the top "**K Eigen-Faces"** after reshaping the image matric,Here K=5
11. Next we have imported the test image of our sportsperson ,**MSDHONI** and pre-processed it and modified it like train dataset and then subtracted it to mean train_set.
12. Next we took the inner dot product of this subtracted Image obtained and transpose of the eigen-face matrix.
13. Next we again found the inner dot product of the above result with eigenfaces matrix ,and after reshaping it to 100X100,finally we got the **regenerated image for the particular K.**

14. Next we have printed the regenerated image using the different K values from 1 to 11 in the code i.e. total no. of eigen faces obtained one by one and analyzed the result.
15. We have used **MSE** and **SSIM-Structural Similarity Index** for every regenerated image.

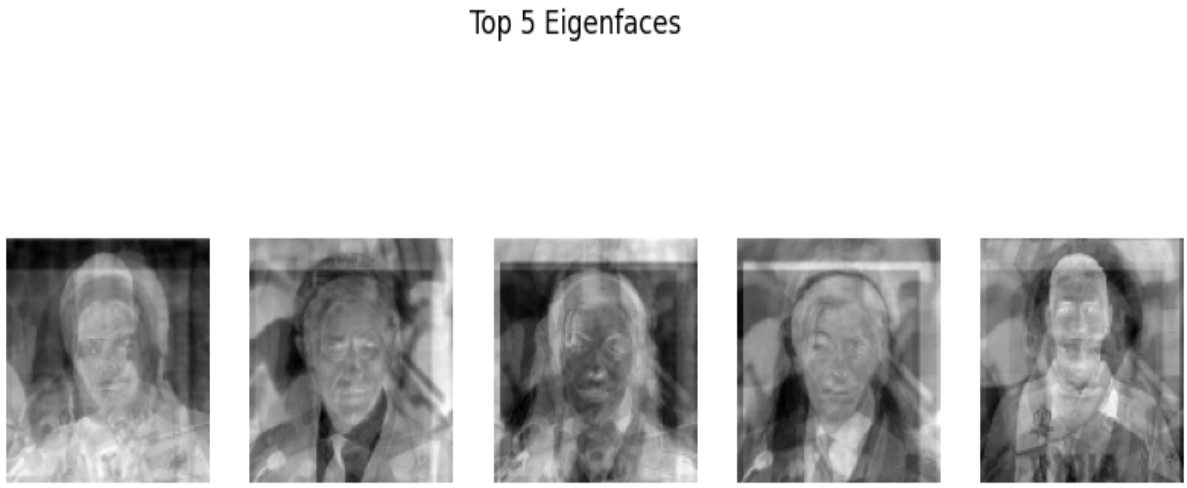
Results :-

1. LFW DATASET : -

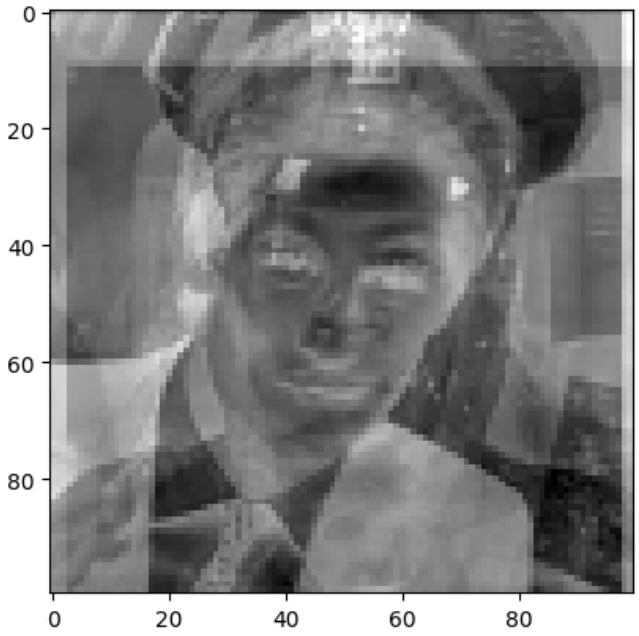
2. Test Image:-

3. Mean Face :- (100,100)

4. Top **K=5** EigenFaces



5. Reconstructed Image Using the weights of Eigen-Faces K=5



6. Result Reconstructed Image from the weights of all the eigen-faces for K=1 TO 11.
   (1, 1)
   Mean Squared Error (MSE):28593.2714 for the output image using the eigen faces k=1
   reconst image using k =1 eigen faces

Structural Similarity Index (SSIM): 0.06590838639856195
Value of k (eigen faces) =1
—---

(1, 2)
Mean Squared Error (MSE):28500.9277 for the output image using the eigen faces k=2
reconst image using k =2 eigen faces

Structural Similarity Index (SSIM): 0.07094353170827951
Value of k (eigen faces) =2
—---

(1, 3)
Mean Squared Error (MSE):27819.0183 for the output image using the eigen faces k=3
reconst image using k =3 eigen faces

Structural Similarity Index (SSIM): 0.08739926295689965
Value of k (eigen faces) =3
—---

(1, 4)
Mean Squared Error (MSE):27796.4167 for the output image using the eigen faces k=4
reconst image using k =4 eigen faces

Structural Similarity Index (SSIM): 0.08709974567060312
Value of k (eigen faces) =4
—---
(1, 5)
Mean Squared Error (MSE):18342.9095 for the output image using the eigen faces k=5
reconst image using k =5 eigen faces

Structural Similarity Index (SSIM): 0.1979179270482273
Value of k (eigen faces) =5
—---
(1, 6)
Mean Squared Error (MSE):17008.3246 for the output image using the eigen faces k=6
reconst image using k =6 eigen faces

Structural Similarity Index (SSIM): 0.25132290609760755
Value of k (eigen faces) =6
—---
(1, 7)
Mean Squared Error (MSE):17167.2281 for the output image using the eigen faces k=7
reconst image using k =7 eigen faces

Structural Similarity Index (SSIM): 0.23137902494932802
Value of k (eigen faces) =7
—---
(1, 8)
Mean Squared Error (MSE):16978.6090 for the output image using the eigen faces k=8
reconst image using k =8 eigen faces

Structural Similarity Index (SSIM): 0.2468917798916578
Value of k (eigen faces) =8
—---

(1, 9)
Mean Squared Error (MSE):16942.5806 for the output image using the eigen faces k=9
reconst image using k =9 eigen faces

Structural Similarity Index (SSIM): 0.2516792443801562
Value of k (eigen faces) =9
—---

(1, 10)
Mean Squared Error (MSE):16590.8623 for the output image using the eigen faces k=10
reconst image using k =10 eigen faces

Structural Similarity Index (SSIM): 0.2747880893248418
Value of k (eigen faces) =10
—---

(1, 11)
Mean Squared Error (MSE):16590.8531 for the output image using the eigen faces k=11
reconst image using k =11 eigen faces

Structural Similarity Index (SSIM): 0.2747913531864383
Value of k (eigen faces) =11

Observations and Analysis:-

1. We have seen that as the value of k increaes the reconstructed image gets good and clear picture of test data which we have taken.
2. Also the MSE which we have calculated is decreasing as we move from k=1 to k=11.which indicates that our algorithm works perfectly.
3. Also the SSIM score is increasing as we move from k=1 to k=11.which indicates that our algorithm works perfectly.SSIM[-1:1] 1:BEST -1:WORST
4. Regenerated image of k=11, `Mean Squared Error (MSE):16590.8531`

   `Structural Similarity Index (SSIM): 0.27479135`

Which indicates that our model is working good and good results are obtained from it.

Colab link:-https://colab.research.google.com/drive/1XclhPg0FcpNoP7DbTM_od-6Mfm1pH4qb?usp=sharing
Modified Dataset link:https://drive.google.com/file/d/186CdGDwv6KwU2b2BFJ_szvz0SX7DfRDF/view?usp=sharing

Q2.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.

Ans:
Dataset Details:
CIFAR10 is a popular image classification dataset that consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The dataset is split into 50,000 training images and 10,000 test images. The 10 classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Brief about Image Retrieval:
Visual BoW (Bag of Words) is a well-known technique for image retrieval and classification. It involves representing images as histograms of visual word occurrences, which are extracted from image features using clustering algorithms such as K-Means.

The following are the steps for creating an image search engine for the CIFAR-10 dataset using Visual BoW:
Data Preprocessing: Loaded the CIFAR-10 dataset and preprocess the images. This may include resizing the images, normalizing pixel values, and extracting image features using techniques such as SIFT, SURF, or HOG.
Visual Word Dictionary: Used a clustering algorithm like K-Means to cluster the extracted image features into visual words. This creates a visual word dictionary that can be used to represent images as histograms of visual word occurrences.
Image Representation: For each image in the dataset, represent it as a histogram of visual word occurrences. This involves extracting the image features, mapping them to visual words in the dictionary, and counting the occurrences of each visual word in the image.
Query Image Representation: Represent the query image as a histogram of visual word occurrences using the same process as step 3.
Similarity Calculation: Calculate the similarity between the query image and each image in the dataset using a similarity measure like Euclidean distance or cosine similarity. This involves comparing the histograms of visual word occurrences for the query image and each dataset image.
Retrieval: Sort the images in the dataset based on their similarity to the query image and retrieve the top-5 similar images.
Evaluation: Evaluate the search results using ground truth labels for the dataset by calculating the precision, recall, and average precision (AP). To visualize the performance of the search engine, draw the precision-recall (P-R) curve.
By following these steps, we can create an image search engine for the CIFAR-10 dataset using Visual BoW.

Procedure:

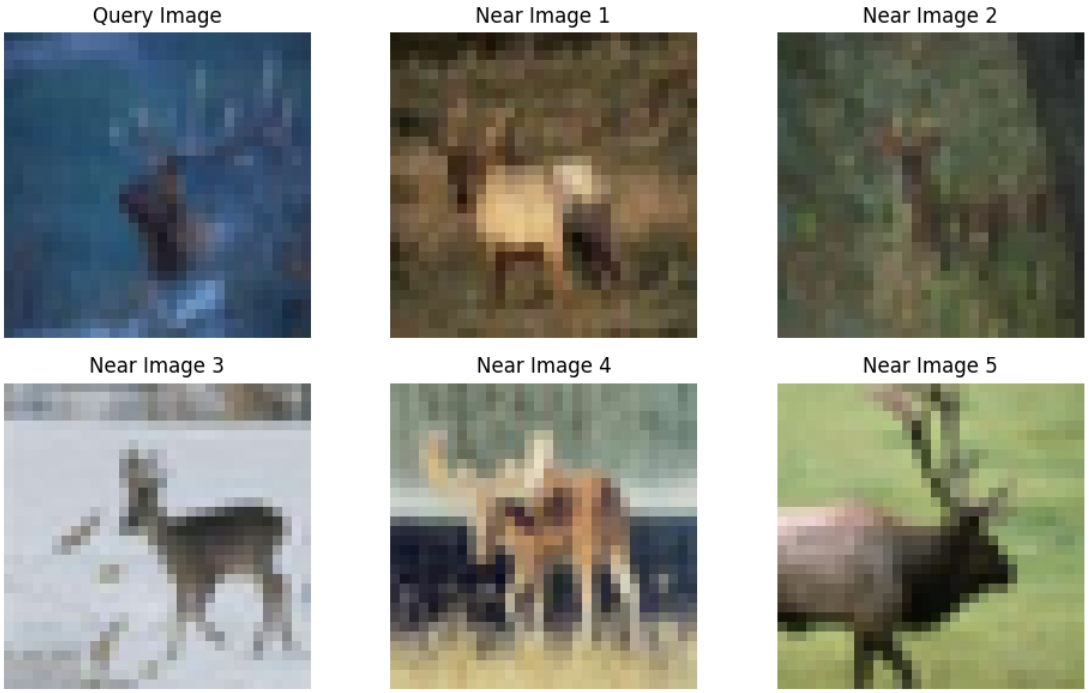
1. Data Preparation: The first step is to download and preprocess the CIFAR-10 dataset. This includes resizing the images to a fixed size, converting them to grayscale, and dividing them into training and testing sets.
2. The train_data and test_data are then sliced to reduce the size of the dataset for faster processing. Specifically, the first 20,000 images from the training data and the first 1,000 images from the test data are selected for processing.
3. Finally, the sklearn.cluster library is imported for use in clustering the extracted features.
4. the cv.SIFT_create() function is used to create a SIFT (Scale-Invariant Feature Transform) object, which can be used to detect keypoints and compute descriptors in images.
5. The train_labels and test_labels lists are converted to numpy arrays using the np.array() function for further processing.
6. The resulting train_data_transposed and test_data_transposed arrays contain the image data in the standard format expected by the SIFT object.
7. The get_keypoints function takes an image dataset as input and returns two values: a list of keypoint descriptors for each image in the dataset, and all keypoints concatenated into a single array.

8. The code first calculates the true positives, false positives, false negatives, and sorted predictions (predictions). Then it calculates the precision and recall for each threshold (i.e., prediction value) and stores them in the corresponding arrays. Finally, the function plot_graph is called to plot the Precision-Recall curve using the calculated precision and recall values.
9. Next we define a function query_image(kmeans,linear_cl) that generates a random query image from the test set, extracts SIFT descriptors from it, predicts the cluster index of each descriptor using the kmeans clustering model, creates a histogram of the predicted cluster indices, and then uses a linear classifier linear_cl to predict the class label of the query image based on this histogram.
10. The function show_near_images(near_images,train_data_transposed,query_idx) takes the output of the query_image() function and displays a figure with four images: the query image and the three nearest training images. The near_images parameter contains the indices of the three nearest training images, and train_data_transposed contains the transposed training data.

Outputs:-

Query Image:- Class- “Deer”

Near Image obtained :-Class- “Deer” —-- 5/5

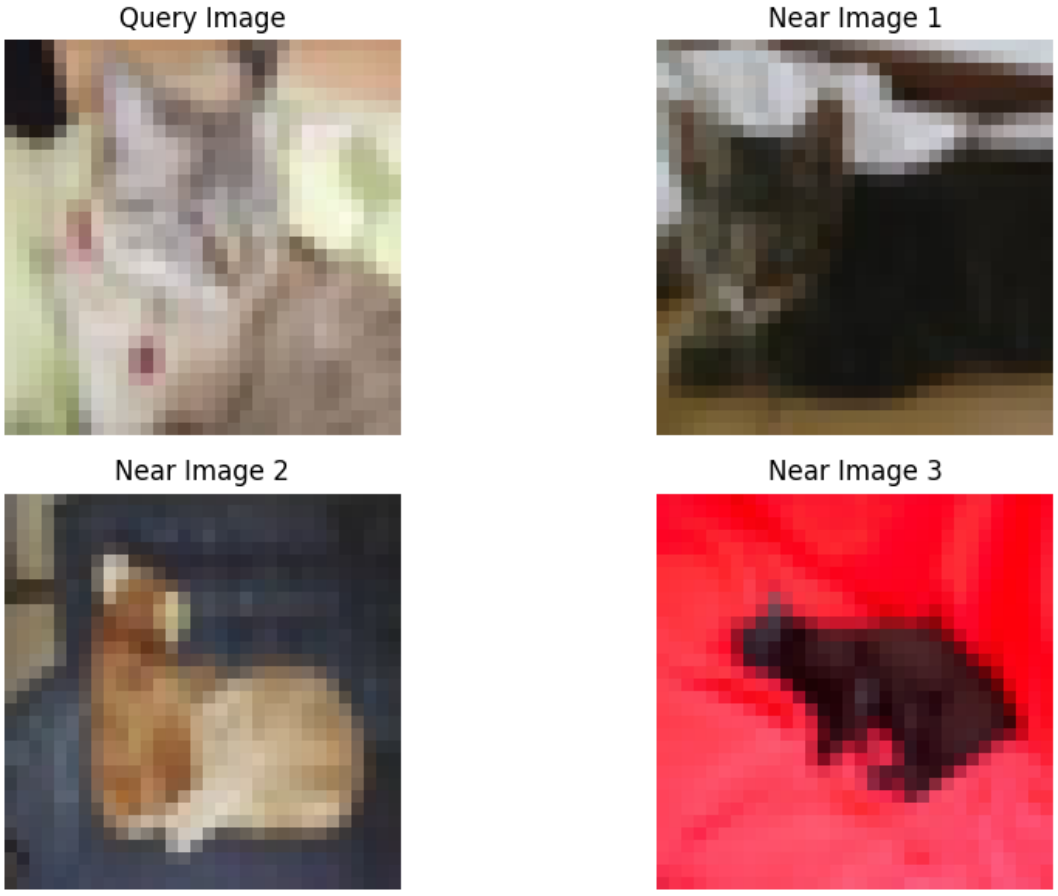


Result:-

accuracy: 0.125
precision: 0.6218905472636815
recall: 1.404494382022472
f1_score: 0.8620689655172413
AP: 0.125

Result 2:-

Query Image:- Class- “Cat””Meeooooooowww”
Near Image obtained :-Class- “Deer” —-- 3/3

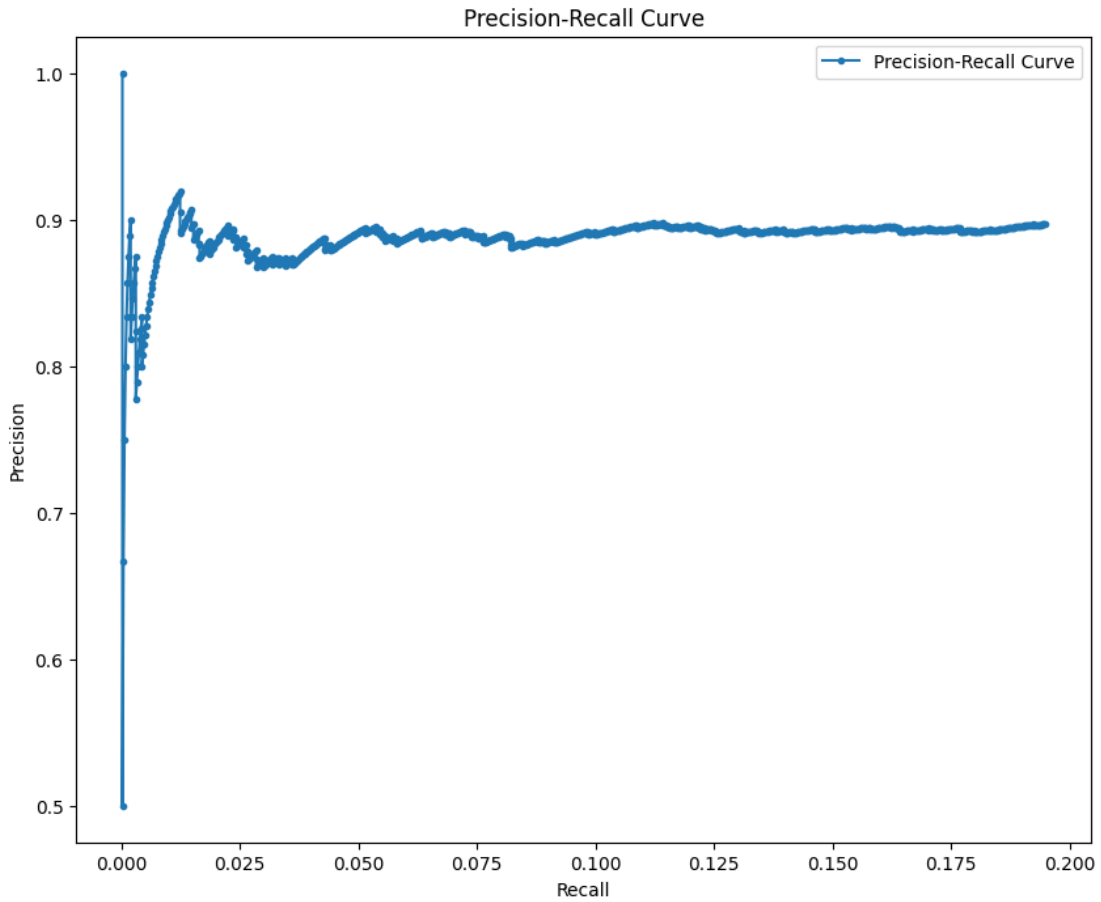


Conclusions:-

The results obtained for image retrieval for the "Deer" class have an accuracy of 0.125, which means that the algorithm was able to correctly classify 12.5% of the images in this class.

The precision of 0.6218905472636815 suggests that when the algorithm predicted an image as a "Deer," it was correct approximately 62.2% of the time.

The recall of 1.404494382022472 suggests that the algorithm was able to correctly identify only about 1.4 "Deer" images out of every 10 "Deer" images present in the dataset. This indicates that the algorithm missed a large number of "Deer" images in the dataset.
The F1-score of 0.8620689655172413 is a harmonic mean of precision and recall and suggests a balance between precision and recall.
The AP score of 0.125 is relatively low, indicating that the image retrieval algorithm did not perform well for the "Deer" class.
From these results, we can conclude that the image retrieval algorithm for the "Deer" class needs improvement. The low recall indicates that the algorithm is not able to identify a significant number of "Deer" images correctly. It could be due to the limited features or complexity of the "Deer" class images. Therefore, the algorithm may benefit from incorporating more sophisticated image features or a different approach to improve its performance.



Graph of the Precision Recall Curve as mentioned above.

Colab link:-https://colab.research.google.com/drive/1_rVSXeExnPR4qmomtOrU1ugw6l256z57?usp=sharing

Q3.Viola Jones Face detection: Write down Viola Jones's face detection steps in detail.
Ans:

The widely used Viola-Jones algorithm for face detection is based on the AdaBoost learning algorithm and Haar-like characteristics. The steps of Viola-Jones face detection are as follows:

1. Haar-like Features: Rectangular **Haar-like Features** are used to record the texture of the image. The integral image of the original image can be used to quickly calculate the Haar-like features. The original image's pixel values are added to form the integral image in a specific way that makes it easy to quickly calculate the sum of the pixels in any rectangular area.

2. The pixel values from the original image are added together to form the integral image, which is a 2D array. The total of all the pixels above and to the left of each pixel in the integral image equals the corresponding pixel in the original image. Any rectangular area of the original image's pixels' total is calculated using the **integral image.**

3. Machine learning system called the Adaboost algorithm combines weak classifiers to create a strong classifier. The Adaboost method chooses the most effective group of weak classifiers by reducing the classification error rate.

4. The cascade classifier is a multi-stage classifier that has numerous stages, each of which has a unique set of weak classifiers. The Adaboost algorithm is used to train the cascade classifier, and each level has a larger false-positive rate than the stage before it.
5. Training: The following steps are involved in the Viola-Jones face detection training process:

   Gather a Set of Positive and Negative Training Images: The initial phase entails gathering a set of positive and negative training images. Positive images are those with faces, and negative images are those without faces.

   c. Create Features Similar to Haar: Using a sliding window method, Haar-like features are created from the training images. The Haar-like features are computed for each window that slides over the image.

   c. Train Weak Classifiers: The Adaboost method is used to train a group of weak classifiers. Each weak classifier is trained using a certain threshold and Haar-like feature.
   d. Create a Cascade Classifier: The weak classifiers are used to create a cascade classifier. A series of weak classifiers are used at each stage of the cascade classifier, which has numerous stages.

6. Face Detection: After the cascade classifier has been trained, faces can be detected using it. The following steps are included in the face detection process:
   a. preprocessed image is one that has had its noise and contrast improved.
   b. Sliding Window: The Haar-like properties are calculated for each window as a sliding window is moved over the image.
   c. Cascade Classifier: Each window receives the cascade classifier's application. A window moves on to the following stage if it is determined at a certain stage to be a face. A window is disregarded if it is ever determined that it is not a face.

   d. Post-processing: In order to reduce false positives, the remaining windows that are identified as faces are post-processed. Non-maximum suppression is used in the post-processing stage to get rid of overlapping windows and thresholding to get rid of weak detections.
   Some clipped images from the research paper which will demonstrate the process more clearly:
   1.

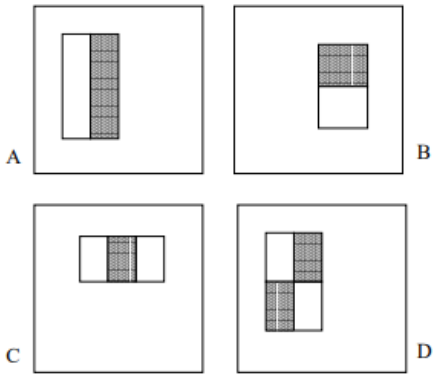


Figure 1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

   2.

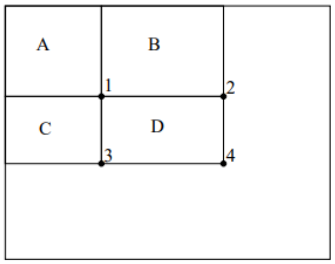


Figure 2: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

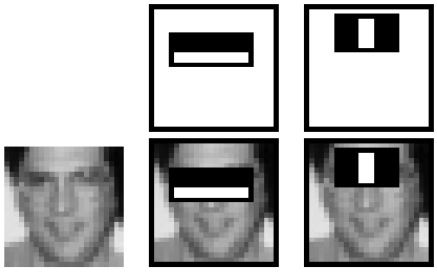


Figure 3: The first and second features selected by AdaBoost. The two features are shown in the top row and then overlayed on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

of the nose and cheeks (see Figure 3). This feature is relatively large in comparison with the detection sub-window, and should be somewhat insensitive to size and location of the face. The second feature selected relies on the property that the eyes are darker than the bridge of the nose.

Note :- The above three images and their captions are taken from the research paper: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

The Viola-Jones face detection algorithm operates in this manner.

Q4.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.

Ans:
Dataset Details:
1.DEER DATA (MANUALLY CROPPED)
2.NON DEER DATA(MANUALLY CROPPED)
3.TEST DATASET
Procedure:-

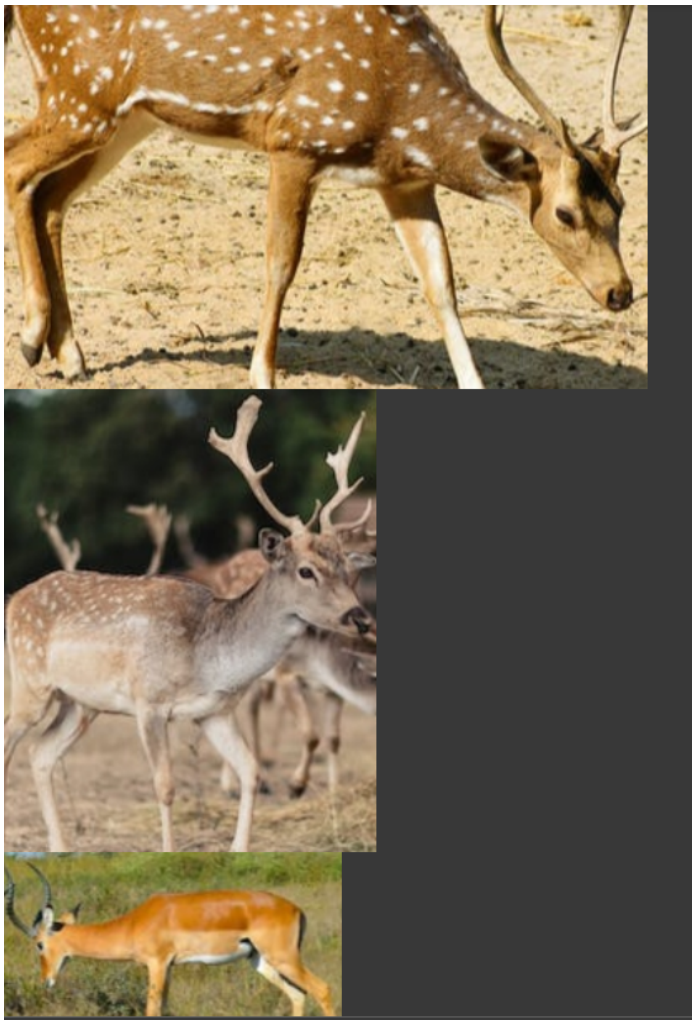
1. Manually cropped the deer image with the tight and closest boundary and also the non deer image patches and grouped both into two sets deer dataset and non deer dataset ,
2. Link of both the dataset is provided in the dataset section ,upload it to drive and mount it to the colab.
3. Next import all the images of both the set and apply the HOG feature extraction in it.
4. Next append both sets HOG feature and make it as a x_train data and also label 1 for deer and 0 for non deer yourself and prepare both train x and label y
5. Next import SVM from scikit library and pass the train x and label y and train the model
6. Next fit the model and predict the value and also check whether it is correct or not.
7. Use the test dataset link provided in the dataset section and also in the question to get the predicted image whether it is a deer or not.
8. Next we need to implement the sliding window to the deer detected in the image.
9. We implemented the sliding window using the three defined function slidingwindow ,find_deer and also non-max-suppression and implemented the sliding window on both modified and questions test dataset.
10. We have also shown the accuracy cross_val_score on training image and accuracy f1score,precision on the test image prediction of the model .

Results:-

```
17
Accuracy: mean: 0.7619047619047619
(22, 2916)
Total no. of test images are 22
Some of the prediction samples are :-[0. 0. 0. 0. 1. 1.]
THE PERFORMANCE OF THE CLASSIFICATION MODEL OF DEER VS NON-DEER:
Accuracy score of the model is:::::0.7272727272727273
precision_score of the model is:::::1.0
f1_score  of the model is:::::0.7857142857142858
```

1. Accuracy of the model:-0.727
2. Precision of the model:-1
3. F1 score of the model:-0.785

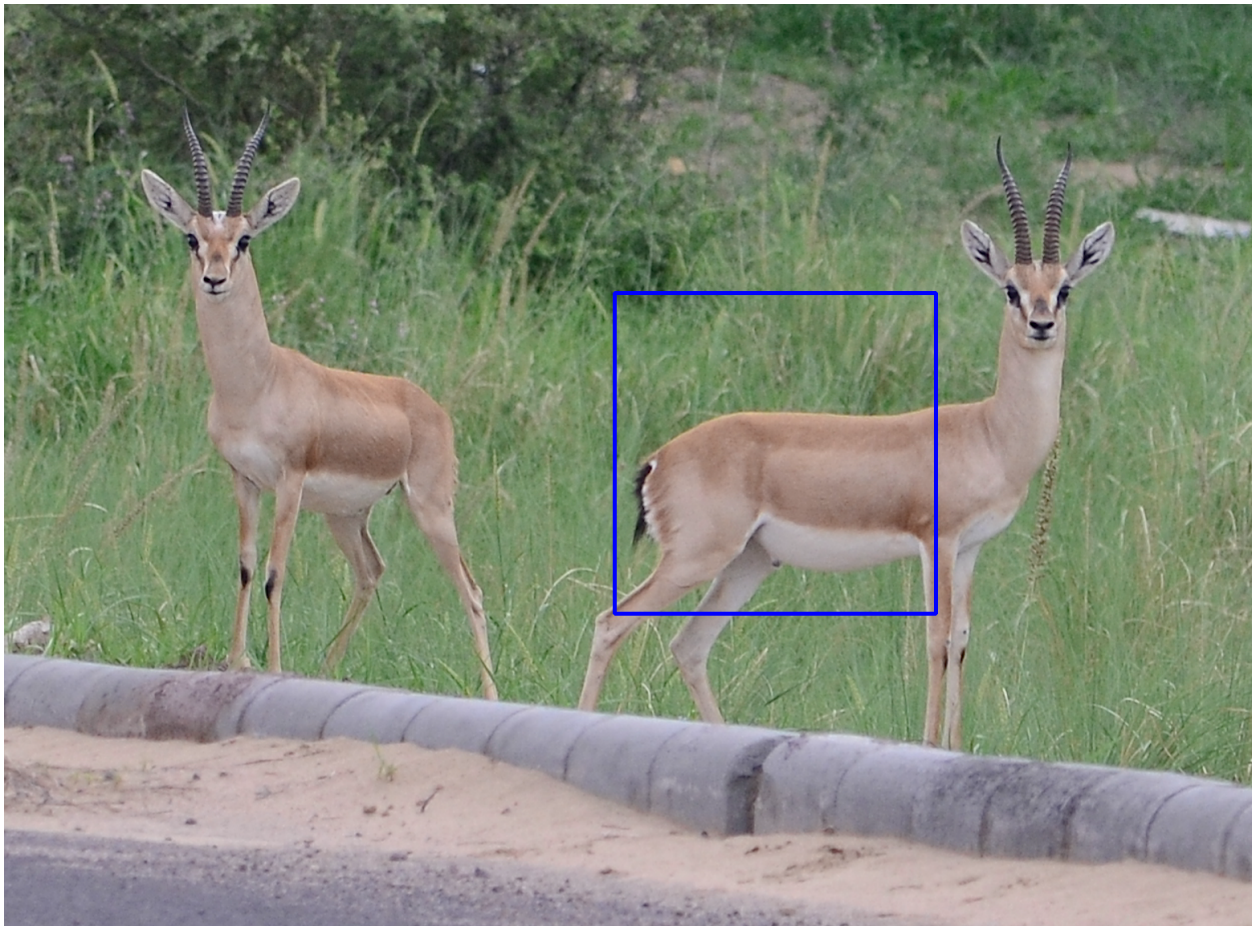
Some of the deer images are:-



Some of the non deer images:-

Sliding Window Dataset:-
The **BLUE box** indicates the model deer detected.
Here are the test datasets results of the [DATASET]

TEST DATSET RESULT OF THE QUESTION:-



Colab link:-https://colab.research.google.com/drive/1FnJCmu8U9Y0p4GgRKTBdj5FbYT01H532?usp=sharing

References:-

1. https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
2. https://www.cs.toronto.edu/~kriz/cifar.html
3. https://github.com/lionelmessi6410/Face-Detection-with-a-Sliding-Window/blob/master/code/run_detector.py
4. https://www.thepythoncode.com/article/hog-feature-extraction-in-python
5. https://www.pinecone.io/learn/bag-of-visual-words/
6. Class lecture notes
7. CHATGPT