Assignment 3

Computer Vision

Padhye Soham Satish (M22RM007)

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 in 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)

Colab link-

https://colab.research.google.com/drive/1jr30gUdUYTtYkM14x03TEsHB9oPheqLV#scrollTo=uj-8RkzQ7BMB

In this code I have appended Royal challengers Bengaluru IPL team player AB de Villiers for finding the eigenfaces and image recognition. I have used two methods, one with using PCA library and method 2 without using PCA . Explanation for is as follows.

Code explanation-

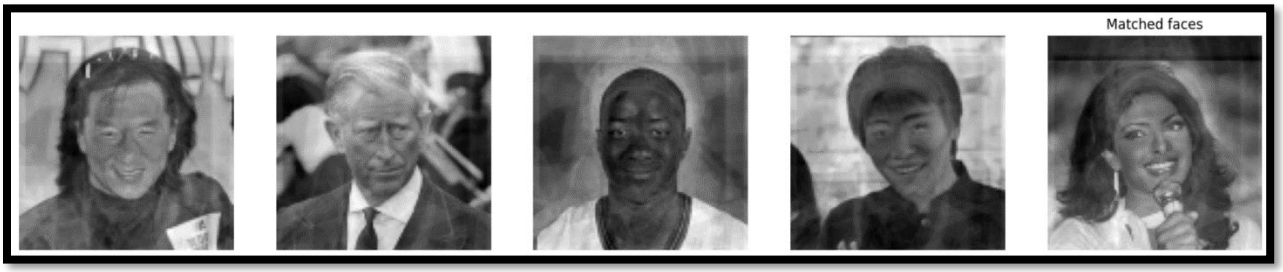
1. First mount the google drive where I have saved the images
2. Then set some common dimension for all images. As all images are of different sizes we should resize it such that all images will be of same size.
3. Define normalize image function for normalizing the image.
    a. Create an empty list to store normalized images
    b. Load the image from drive, resize the image to mention size then convert the image to grayscale as many of the function requires black and white image.
    c. Convert list of images into numpy array.
4. Preprocess the images.
    a. Flatten each image into vector and show the size of vector. Printing the size of vector is very important to match the dimensions of matrix while doing the dot product.
    b. Compute mean image
    c. Subtract the mean image from the flattened images
5. Use PCA inbuilt library to calculate the covariance matrix.
    a. Apply PCA to the Preprocessed images.

   b. For PCA we have to select the number of components. Selecting too few components may result in a significant loss of information, while selecting too many components may lead to overfitting and redundancy. Determining the optimal number of components in PCA often involves a trade-off between retaining enough information and keeping the dimensionality low. It is important to experiment with different numbers of components and evaluate the performance of the reduced dataset in a downstream task to determine the optimal size.
   c. Compute the covariance matrix and print Size of covariance matrix.
6. Find Eigenvalues and Eigenvectors
   a. Compute the eigenvectors and eigenvalues of the covariance matrix using SVD
   b. The eigenvectors are the columns of the matrix U
   c. The eigenvalues are the diagonal elements of the matrix S
   d. Sort the eigenvalues and eigenvectors in descending order because we have to find the best matching image.
   e. Calculate the weight. The weight of each image on the eigenfaces can be calculated using the transformation matrix obtained from the PCA model. The transformation matrix represents the projection of the original data onto the principal components, which are also known as the eigenfaces in the context of image analysis. The weights of each image can be calculated by multiplying the centered image matrix with the transpose of the transformation matrix.
7. Load the test image.
8. Compute the Euclidean distance between the test image and the training images and find best match
9. Preprocess the test image same as we have done for preprocessing of train images.
10. Find weight of test image
11. Find the Euclidian distance between test image and other training images
12. Get the indices of the closest matching images by sorting the image distances.
13. Create a figure object with 5 subplots
14. Loop through each subplot and display the closest matching image.
15. Resize matrix to get 2D image from flattened image.
16. Show the top 5 matching images.

Output of method of eigenface recognition using PCA-

Test Image

Matched images-



As we can see in the results the test image is not matching with the training image. The output shows top 5 best matches of test image, it is not matching, the reasons for not matching may be-

Observations and reasons-

1. Insufficient Training Data: Using only 11 images for training may not be sufficient to capture the variability in the face images. This may result in the eigenfaces not being able to adequately represent the test image, leading to a poor match.
2. The test image may have significant differences from the training images in terms of lighting, pose, expression, or occlusion. If the eigenfaces used for matching are not able to capture this variability, the test image may not match well with the training images.
3. The number of principal components selected for the eigenface representation can affect the quality of the match. If too few principal components are selected, the representation may not capture enough variability in the face images. On the other hand, if too many principal components are selected, the representation may be noisy and may not generalize well to new images.
4. Sensitivity to Alignment: The eigenface matching algorithm is sensitive to alignment errors. If the test image is not well aligned with the training images, the eigenfaces may not align well, resulting in a poor match.
5. The eigenface representation may not be robust to illumination variations, especially if the lighting conditions of the test image are significantly different from those of the training images.
6. Limited Variability in Training Images: Even if the number of training images is sufficient, if the images are very similar to each other in terms of lighting, pose, expression, or occlusion, the eigenfaces may not be able to capture the variability needed to match new faces.

So because of these reasons maybe the test face is not matching.

Now I have **not** used the PCA and matched the images. The test image matches perfectly with the given images. I have taken the help of original research paper of eigenfaces for Recognition. Link for the paper is given below-

https://ieeexplore.ieee.org/document/6793549/authors#authors

Explanation for the code-

1. Create a class named FaceRecognizer and define constructor and methods

2. Load the image from drive
3. Convert the list of images to a NumPy array
4. Reshape the grayscale images to a 2D array
5. Compute the mean image and show mean image
6. Subtract the mean image from the flattened images. New image the we will get are centered_images.
7. Compute covariance matrix of centered_images
8. Compute eigenvalues and eigenvectors for covariance matrix.
9. Sort the eigenvectors in descending order of eigenvalues for matching the test image with given images
10. Normalize the eigenvectors. This vector is also known as Eigenface
11. Select top k eigenfaces. K can take maximum value upto number of images in training set
12. Project all images on the eigenfaces.
13. Save mean image and centered images.
14. Load test image from drive.
15. Project the test image on the eigenfaces
16. Compute distances between the projected faces and the projected test image
17. Find the index of the closest match using argmin function.
18. The recognized face is the one that has the minimum distance from the test image.
19. Showing the matched images.

An important step in this approach is to calculate the covariance matrix and eigenvectors. If we take the big covariance matrix(A.AT) then the time required for computation is very large. So, we should use the (AT.A) as a covariance matrix and calculate the eigenvalues and eigenvectors of this matrix which is very easy. Reference for the calculation of Covariance matrix is originally given in the research paper-

$$C = \frac{1}{M}\sum_{n=1}^{M} \mathbf{\Phi}_n \mathbf{\Phi}_n^T \qquad (3)$$

$$= AA^T$$

where the matrix $A = [\mathbf{\Phi}_1\ \mathbf{\Phi}_2\ \ldots\ \mathbf{\Phi}_M]$. The matrix C, however, is N^2 by N^2, and determining the N^2 eigenvectors and eigenvalues is an intractable task for typical image sizes. We need a computationally feasible method to find these eigenvectors.

If the number of data points in the image space is less than the dimension of the space ($M < N^2$), there will be only $M - 1$, rather than N^2, meaningful eigenvectors. (The remaining eigenvectors will have associated eigenvalues of zero.) Fortunately we can solve for the N^2-dimensional eigenvectors in this case by first solving for the eigenvectors of an M by M matrix—e.g., solving a 16×16 matrix rather than a $16{,}384 \times 16{,}384$ matrix—

and then taking appropriate linear combinations of the face images $\mathbf{\Phi}_i$. Consider the eigenvectors $\mathbf{v}_i$ of A^TA such that

$$A^TA\mathbf{v}_i = \mu_i\mathbf{v}_i \tag{4}$$

Premultiplying both sides by A, we have

$$AA^TA\mathbf{v}_i = \mu_iA\mathbf{v}_i \tag{5}$$

from which we see that $A\mathbf{v}_i$ are the eigenvectors of $C = AA^T$.

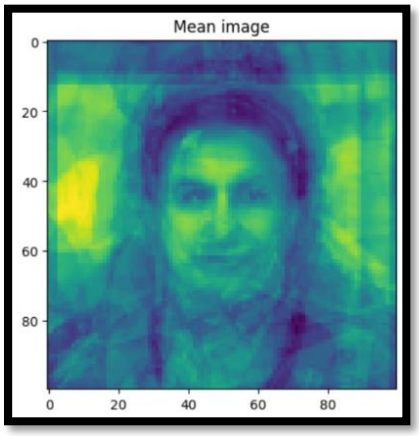
Following this analysis, we construct the M by M matrix $L = A^TA$, where $L_{mn} = \mathbf{\Phi}_m^T\mathbf{\Phi}_n$, and find the M eigenvectors, $\mathbf{v}_l$, of L. These vectors determine linear combinations of the M training set face images to form the eigenfaces $\mathbf{u}_l$.

$$\mathbf{u}_l = \sum_{k=1}^{M} \mathbf{v}_{lk}\mathbf{\Phi}_k, \qquad l = 1, \ldots, M \tag{6}$$

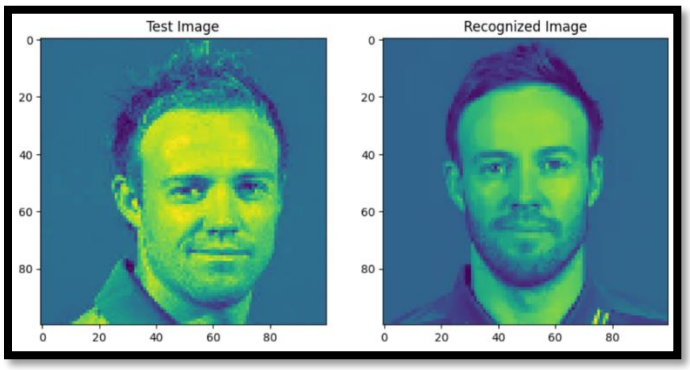
With this analysis the calculations are greatly reduced, from the order of the number of pixels in the images (N^2) to the order of the number of images in the training set (M). In practice, the training set of face images will be relatively small ($M \ll N^2$), and the calculations become quite manageable. The associated eigenvalues allow us to rank the eigenvectors according to their usefulness in characterizing the variation among the images. Figure 2

Create a class instance and see the results.

Output of mean image-



Output of matching images-



Observations-

1. Here we can see that the test image is matching perfectly, but if we change the image then there is a chance of mismatch. Two images should have face in same region of the image. Otherwise, it will not match.
2. There are some limitations of this approach which are
    a. Sensitivity to lighting conditions: Different lighting conditions can cause changes in the appearance of the face, leading to recognition errors.
    b. Limited capability to handle facial variations: The algorithm is not very effective in handling facial variations such as facial expressions, pose, and occlusions. This is because the algorithm relies on the assumption that the face is in a fixed position and orientation.
    c. Sensitive to noise: The algorithm is sensitive to noise in the images. This can lead to recognition errors when the images are corrupted by noise.
    d. Computationally expensive: The algorithm is computationally expensive, particularly when dealing with large datasets. This can limit its applicability in real-time applications.
    e.

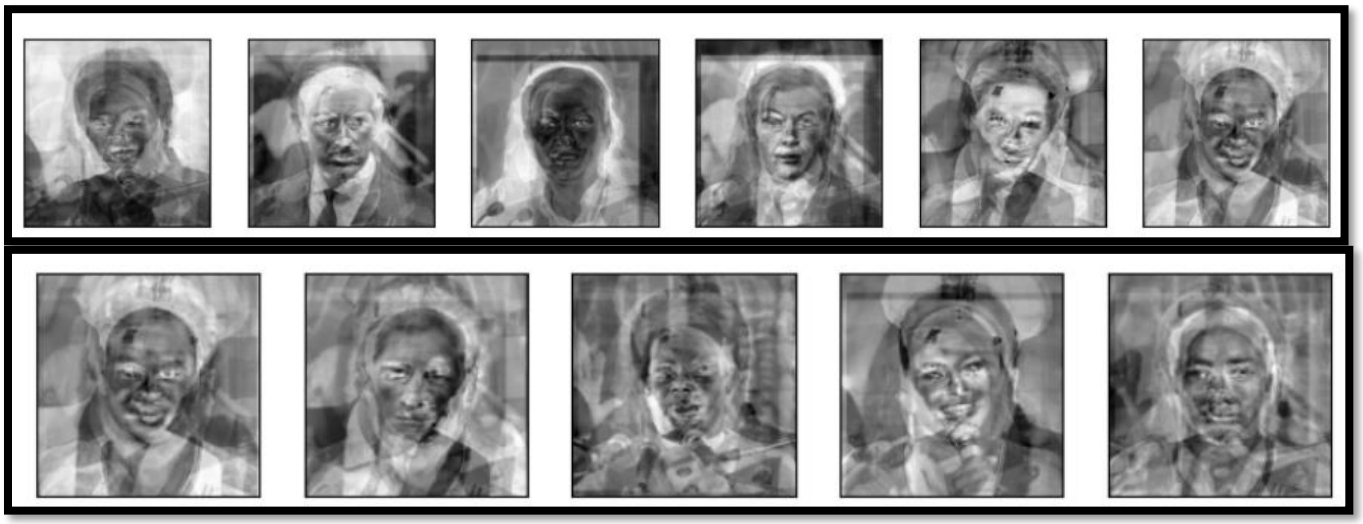
Showing top K eigenfaces of the images -

Explanation of code-

1. Define a class called Eigenfaces that contains several methods for loading and processing images, fitting a PCA model, and testing the model on a test image.
2. The init() method initializes the class with several parameters, like the number of principal components to use, the path to the training images, and the path to the test image.
3. The load_images() method loads the training images from the specified folder, resizes them to a standard size of 128x128, and stores them in an array.
4. The images are then flattened into vectors and standardized.
5. The fit() method creates a PCA model with the specified number of components and fits it to the standardized training data.
6. The test() method loads the test image, resizes it, and standardizes it. It then projects the test image onto the principal components using the transform() method of the PCA object. The reconstructed test image is obtained by inverse transforming the projected test image. Finally, the method displays the test image, along with the top eigenfaces.

Create the class instance to access various methods defined in the class.

If we want to show the top 5 eigenvectors, then put K=5 in the code.



Output of the code is top 10 eigenfaces of the given images is shown above

- Important points while writing code for eigenfaces is Matrix dimensions. We should check the matrix dimensions at every step so that we can get the idea of mathematical calculation that we are doing.
- We should reshape the matrices to match the order while doing dot product. Otherwise, it will throw an error.
- It is difficult to match any face images because of limitations mentioned earlier in the report.

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.

Colab Link-
https://colab.research.google.com/drive/1kQ1VELNe8hAYcB4XZgMiPKW_cj1otXD9#scrollTo=g1-EKHtpYL0c

In this question I'm using the TensorFlow library in Python to load and preprocess the CIFAR-10 dataset for use in a machine learning model.

1. Import the TensorFlow library
2. The code loads the CIFAR-10 dataset using the Keras API .It loads the train and test image data and their corresponding labels into four separate variables.
3. Do the preprocessing of the images-The train and test images are divided by 255.0 to scale their pixel values to the range between 0 and 1.
4. The data is now ready to be used for training a machine learning model.
5. Define the layers of the neural network. Use this network for extracting the features and doing multiclass classification. Steps in building the neural network are-
    a. The first layer is a 2D convolutional layer which has 32 filters of size (3, 3) I heve used the ReLU activation function which is very commonly used. The input shape is (32, 32, 3) this which means that the input is a color image with height and width of 32 pixels and three color channels (red, green, blue). The padding parameter is set to 'same', which indicates that the input image is zero-padded so that the output feature map has the same spatial dimensions as the input.
    b. The next layer is a max pooling layer which is having a pool size of (2, 2), which reduces the spatial dimensions of the output from the previous layer by a factor of 2.
    c. Then we will repeat same pattern for the next three layers, each with will have more filters and higher-level features. The output from the last convolutional layer has a spatial dimension of 4x4 and 256 feature maps. Don’t reduce the number of filters after increasing it.
    d. Then the next layer is a flatten layer that converts the output from the last convolutional layer into a 1D vector. By flattening the output of the convolutional layer into a 1D vector, we can give it to the dense layers, dense layer then learn and classify the features extracted by the convolutional layers.

e. Then the next layer is a fully connected dense layer with 512 units and the ReLU activation function.

f. Then the next layer is a dropout layer that randomly drops out 50% of the inputs during training to prevent overfitting.

g. Then the final layer is another fully connected dense layer with 10 units and the softmax activation function, which outputs a probability distribution over the 10 classes of the CIFAR-10 dataset. If we use ReLU activation in the last layer, the model will not produce valid probability distributions over the classes (in our case there are 10 classes), and it will not be able to make proper predictions.

6. Compile the model with adam optimizer .The loss function is set to 'sparse_categorical_crossentropy', which is a common loss function used for multiclass classification problems. The metrics parameter is set to 'accuracy', which means that the accuracy of the model will be calculated and displayed . As there are exactly 6000 images per class in the cifar10 dataset we can use the accuracy as performance matric here.

7. Train the model on cifar10 dataset with 10 epochs. Adam optimizer will try to minimize the loss function and evaluate the performance. Also save the model that is useful while making predictions on new images.

8. Use the above model to extract features for the different images from different class from the train and test dataset.

9. Use the k-means clustering algorithm to cluster the train features into 100 clusters. Cluster center contains different features clusters, For example cluster of tires, cluster of airplane wing, car headlight etc.

10. The cluster centers obtained from the KMeans algorithm are stored in variable.

11. Define a function to get histogram, which takes features from each image and vocabulary of BoW from KNN as inputs and returns the histogram of the features of each image in the dataset. In other words we are computing histogram of BoW for each image in the dataset .The function computes the Euclidean distance between each feature from the image and the cluster centers. Then assign each feature to the nearest cluster. For example, if there is a tire in the truck image then it will join the cluster of tires.

12. Define another function to retrieve similar images that takes a query image, train images and train histograms as inputs and returns the top-5 similar images to the query image based on the histogram of visual words.

    a. function first extracts the features from the query image using the pre-trained model and computes the histogram of visual words for the query image using the get_histogram() function.

    b. It then computes the Euclidean distance between the histogram of visual words of the query image and the histograms of visual words of all the train images. It then

sorts the train images based on their distance from the query image and returns the top-5 similar images.

c. Load a test image and resize it to (32, 32) as the model takes input images of size (32, 32, 3). We then call the function to find similar images. We plot the query image and similar images.

13. Lastly, we calculate the recall precision and P-R curve

a. Make the prediction on test data. The variable y_prob variable contains the predicted probabilities of each class for each test image.

b. Create the variable to store average precision of each class.

c. Create an array of zeros with the same shape as the test labels. This array will be used to create a one-vs-all binary classification problem.

d. y_test_one_vs_all(test_labels == I) = 1 - This line sets the elements of the y_test_one_vs_all array to 1 where the corresponding test label equals the current class i. This creates a binary classification problem where the current class is considered as positive and all other classes are considered as negative.

e. Calculate the average precision for the current class using the average_precision_score function.

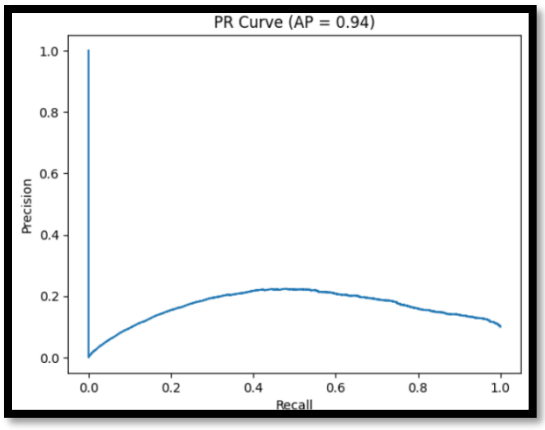
f. Append the calculated average precision for the current class to the created list.

g. Then calculate the mean average precision over all classes.

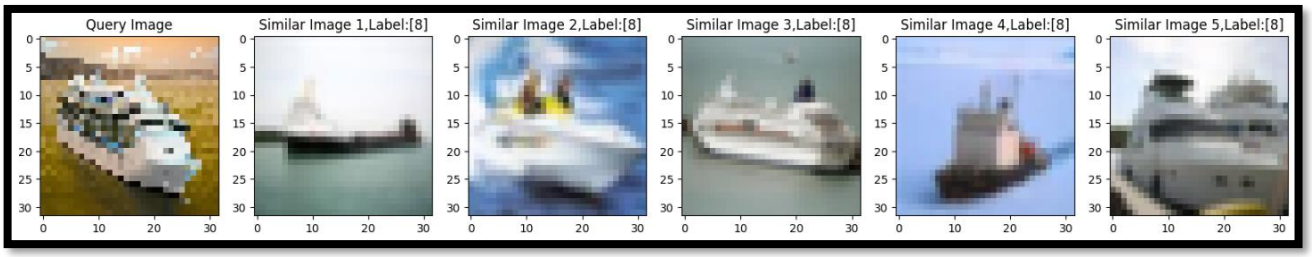
h. Calculate the precision, recall, and thresholds for the precision-recall curve for the current class.

i. Plot the precision-recall curve for the current class.

Output of PR curve-



Output of image search engine-



Observations- As you can see in the image, the test image is of ship, and it perfectly matches with the images in the dataset that corresponds to class label 8. Class 8 in the Cifar10 dataset has all ship images.

Observations and other modifications-

1. Adding more convolutional layers with more filters can allow the model to learn more complex and abstract features from the input images. The downsampling achieved by the MaxPooling2D layers helps to reduce the size of the feature maps and capture the most salient information, making the model more efficient and less prone to overfitting.
2. The changes to the layers in the model may improve its performance on the task , but this cannot be guaranteed without experimentation and evaluation.
3. In the Kmeans clustering function if we change the number of neighbors then it will affect the performance and computational complexity. A larger number of centroids may result in more accurate classification, but may also increase computational complexity. Small number of K will lead to small number of centroids and result in less accurate classification.

Question-3

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

The Viola–Jones object detection framework is a machine learning object detection framework.

The three main contribution of this algorithm are-

1. The first contribution is a new way to represent images called the "Integral Image". This helps to quickly compute features used by the object detector.
2. The second contribution is a learning algorithm called AdaBoost, which selects a small number of critical visual features from a larger set. This makes the classifiers extremely efficient.
3. The third contribution is a method called the "cascade" that combines increasingly complex classifiers. This allows the method to quickly discard background regions of an image and focus more computation on object-like regions. The cascade acts like an object-specific focus-of-attention mechanism.

Main feature of the algorithm is its ability to detect faces extremely rapidly.Fiest we calculate the Integral image. Once we have the integral image, any one of these Harr-like features can be computed at any scale

or location in constant time. This algorithm has less accuracy than CNN but it is still used in cases with limited computational power.

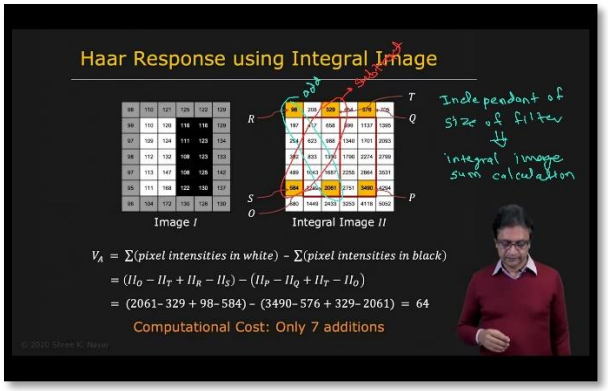
Viola Jones's face detection can only deal with the full view images means there is no occulusion,no head turning , no rotation of the head means simple portrait image of face can be detected.

Steps for Viola Jones's face detection-

1. Preprocessing: The input image is converted into grayscale as this algorithm works on black and white images and the contrast is normalized.
2. Calculate the Integral image-
    a. The integral image is calculated based on the original input image. The integral image at location (x, y) contains the sum of all pixels above and to the left of (x, y), including the pixel at (x, y) itself. Here are the steps to calculate the integral image.
    b. We can add padding for calculating the boundary pixel integral value or in the integral image the pixel value at the boundary is just same as input image.
    c. Calculate the remaining elements of the integral image by summing the pixel values of the input image along the horizontal and vertical directions. For each pixel at location (x, y), calculate the value of the corresponding element in the integral image as follows:
        i. Add the value of the pixel at (x, y) to the value of the element above it (at location (x, y-1)).
        ii. Add the value of the element to the left of it (at location (x-1, y)).
        iii. Subtract the value of the element to the upper-left of it (at location (x-1, y-1)) to avoid double counting.
        iv. Set the value of the corresponding element in the integral image to the result of the above calculations.
    d. Once all elements in the integral image have been calculated, it can be used for fast computation of Haar-like features in the Viola-Jones face detection algorithm
3. Haar-like Features Computation: Haar-like features are computed on the integral image. Haar-like features are simple rectangular patterns that capture variations in brightness of adjacent image regions.
    a. Let us consider a perceptron defined on two variables w and b. This takes image with fixed resolution as the input and returns the output as zero or one depending on the condition mentioned below-

$$f_{w,b}(I) = \begin{cases} 1, & \text{if } \sum_{x,y} w(x,y)I(x,y) + b > 0 \\ 0, & \text{else} \end{cases}$$

    b. Haar features are sufficiently complex to match features of typical human faces. For example: The eye region is darker than the upper-cheeks,the nose bridge region is brighter than the eyes.
    c. Haar-like features are simple rectangular patterns that capture variations in brightness of adjacent image regions.
    d. For each Haar-like feature type and size, the feature is placed at every possible position and scale in the input image. The integral image does fast computation of the sum of pixel intensities in any rectangular region by using only four values from the integral image. As shown-



e. This process is repeated for all possible positions and scales in the input image to generate a set of Haar-like features.

4. Apply a modified AdaBoost training algorithm to the set of all Haar feature classifiers of size (M,N), until a desired level of precision and recall is achieved. The modified AdaBoost algorithm will produce a sequence of Haar feature classifiers, denoted as f1, f2, ..., fk. The goal is to reach a certain level of precision and recall in the face detection task.
    a. If f1, f2, ..., fk are Haar feature classifiers and I is the image then compute f1(I),f2(I).....fk(I) subsequently. If at any point of calculation if F(I)=0, the algorithm immediately returns no face detected". If all classifiers return 1, then the algorithm returns "face detected".
    b. Although the evaluation of features is fast, the large number of features presents are difficult to calculate. For example, in a 24x24 pixel sub-window, there are 162336 potential features. Evaluating all of them for an image would be impractical. To address this issue, the Viola-Jones face detection algorithm utilizes a modified version of the AdaBoost learning algorithm. The modified algorithm selects the most effective features and trains classifiers based on them. AdaBoost combines multiple simple weak classifiers into a powerful strong classifier, which is a weighted linear combination of the weak classifiers.
    c. Learning algorithm - The Viola-Jones face detection algorithm takes a set of N positive and negative training images labeled as faces or not faces, and assigns weights to each image. The algorithm then applies a set of M features to each image, finding the optimal threshold and polarity that minimizes the weighted classification error. A weight is assigned to the best classifiers that are inversely proportional to the error rate. The weights for the next iteration are reduced for the images that were correctly classified. Finally, the algorithm sets the final classifier as a linear combination of the best classifiers.
5. To detect faces in an image, we use a method called cascade classifiers. It works by using a series of weak classifiers that each looks for a specific feature in a small portion of the image, called a sub-window. These weak classifiers are combined to form a strong classifier that can accurately detect faces.
    a. However, it's not efficient to apply all weak classifiers to all sub-windows, as most sub-windows do not contain a face. Instead, we use a cascade structure, where the weak classifiers are arranged in stages, and each stage only checks sub-windows that have not been rejected by the previous stage.
    b. The first stage has a simple classifier that can quickly reject sub-windows that are not faces, while keeping most of the potential face sub-windows through to the next stage.

The subsequent stages use more complex classifiers to reject more false positives and detect more faces.

c. To train the cascade, we need to set the acceptable false positive rate and the minimum detection rate for each stage, as well as the target overall false positive rate. We then use a set of positive examples (images with faces) and negative examples (images without faces) to train the weak classifiers for each stage.
d. By using a cascade of gradually more complex classifiers, we can achieve high accuracy in face detection while still being efficient enough to run in real-time applications
e. The cascade architecture used in face detection has an interesting feature: the performance of each classifier only depends on the performance of the previous classifier. This means that each classifier can have a high false positive rate, meaning it mistakenly detects non-face images as faces, as long as the overall cascade achieves a low false positive rate.
f. For example, if we want to achieve a false positive rate of 1 in a million, a 32-stage cascade can have classifiers with a false positive rate of 65% each and still achieve this overall false positive rate. However, each classifier needs to have a high detection rate, meaning it should correctly detect almost all actual faces in the image. To achieve a detection rate of 90%, each classifier needs to correctly detect about 99.7% of the faces in its sub-window. This means that even though each individual classifier may not be very accurate, when combined in a cascade, they can achieve high accuracy in detecting faces while still being efficient

6. Post-processing- Post-processing techniques help to improve the accuracy and reliability of the face detection algorithm by reducing false positives and false negatives and enabling more accurate tracking and analysis of detected faces. Here are some of the techniques used in post-processing:
    a. Non-maximum suppression (NMS): This technique is used to eliminate overlapping detections. When multiple detections of the same face are found in close proximity, only the detection with the highest confidence score is retained. This helps to reduce the number of false positives and improves the accuracy of the face detection results.
    b. Size filtering: This technique is used to eliminate detections that are too small or too large to be faces. Since faces are usually of a certain size range, detections that are significantly smaller or larger than this range are likely to be false positives. By applying a size filter, these detections can be eliminated and the accuracy of the face detection results can be improved.

Reference-
https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework#cite_note-3

Original research paper -Rapid Object Detection using a Boosted Cascade of Simple Features, Paul Viola, Michael Jones

Question 4-

You are given a few deer train images with this assignment. Manually crop them to find out 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.

Colab link-

https://colab.research.google.com/drive/12HLSyYMct--LiuYveDGgfymwEOkMAcOJ#scrollTo=-jUd6VnIghBA

Explanation of the code-

1. Import the necessary libraries and mount the drive.
2. Check the size of dataset for deer images and non deer images.
3. Define function for calculating the HOG features of deer images and non-deer images.
    a. Initialize empty lists to store the HOG features for the deer and non-deer images.
    b. Compute the HOG features for the deer image using the hog() function and append the resulting feature vector to the deer_features list. Detailed description of the function hog() is as follows.
        i. orientations: It denotes the number of equally spaced orientation histograms to use for the HOG descriptor.
        ii. pixels_per_cell: It denotes the size of the cells that will be used to compute the HOG descriptor. This parameter sets the width and height of each cell.
        iii. cells_per_block: It denotes the number of cells that will be combined to form each block.
        iv. visualize: It is a boolean flag that shows whether to return a visualization of the HOG descriptor or not. flag is set to True, the function will compute the HOG descriptor separately for each color channel and then concatenate the results.
    c. Compute the HOG features for the non-deer image using the hog() function.
    d. Create labels for the dataset by initializing two arrays, for deer and non deer labels, and set their values to 1 and 0, respectively. The length of these arrays corresponds to the number of deer and non-deer images in the dataset.
    e. Combine all deer features and non-deer features and similarly combine all labels and store it in two different variables. It is very important to store features of deer and non-deer in single variable.
    f. Return the combined features and and respective labels arrays.
4. Split the dataset into training and testing sets with 20% examples for testing.
5. Here I'm using 11 images for training and 13 images for testing that are manually cropped images.
6. Build an SVM classifier to classify the images into deer and non deer images using linear kernel
7. Train the SVM classifier by passing the training data and their labels.
8. Predict the labels for the test set and store it in the variable.

9. Calculate the accuracy of the classifier by considering the y testing and y prediction. We can use accuracy as performance measure because the dataset is balanced.
10. Then load the test image, change the color to gray color and resize it to the size equal to the training images. And finally convert it to numpy array.
11. Compute the hog features similarly to what we have calculated for training images.
12. Now pass these computed features of the test image to the classifier to classify it into deer vs non deer images.
    If the prediction is one, then it is deer otherwise there is no deer in the image

Output of the SVM classifier

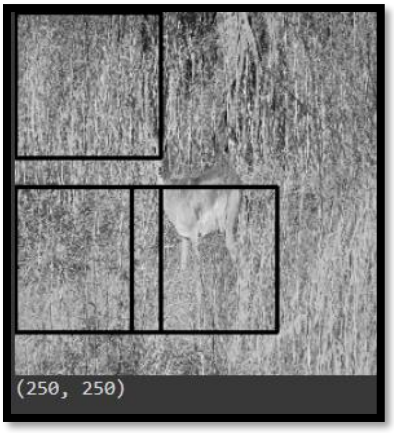
Now we are going to use sliding window technique to detect the deer in the image and draw the rectangle around the deer.

Steps for sliding window technique-

1. Load the test image, resize it to the size equal to the size of training images and concert to grey color.
2. Set the window size for searching the deer in the image. This is very important parameter in window matching algorithm. We have to set the window size by trial and error so that the deer in the image is perfectly captured.
3. Similarly select the steps value for the window sliding. Window will slide while moving in horizontal and vertical direction by this amount. There are other approaches also that are used for selecting window size. For example, We can analyze the size distribution of the objects in your training,Grid search etc.

4. Then loop over image patches(Window size) in both horizontal and vertical direction.
5. Extract image patch and compute the HoG features of the patch. Important step is again patch size should be resized to the training image size.
6. Classify patch as deer or non-deer using SVM classifier that we trained earlier.
7. Draw the bounding box if the prediction is deer.

Output of window search

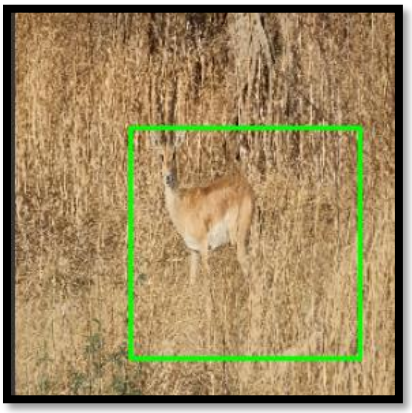


In the above image one can clearly see that there are multiple overlapping boxes around the deer. So here we can make the use of Non-Maximum suppression to remove the overlapping boxes by using some threshold value and keeping only different non overlapping boxes.

Steps for Non maximum suppression-

1. We already have the boxes around the deer that are predicted by SVM classifier.
2. Convert the bounding boxes to NumPy array for calculation
3. Then create the list of picked indexes
4. Sort the bounding boxes by their scores in descending order. Score is calculated by SVM prediction. If the classification is binary, then the score is one only. If the classification is multiclass classification, then the score can take the different values.
5. Loop over the indexes of the bounding boxes
6. Find the overlapping boxes so that we can eliminate them based on the threshold
7. Compute the width and height of the overlapping boxes which is important parameter for removing the box.
8. Then compute the overlap ratio between the overlapping boxes and the selected box
9. Then remove the overlapping boxes from the indexes list
10. Return only the bounding boxes that were picked
11. Finally draw the boxes on the image that are not overlapping in this we are ensuring there are nonoverlapping boxes.
12. Selecting the threshold value for the NMS is also very important. We can't choose higher value because it will draw many rectangles on the image.

Output after NMS -



In the above images the results are good. We can draw the rectangle perfectly around the deer. But in some cases, it is not able to give a good performance because we have a very limited number of images in the training set. So I decided to increase the data set by cropping more images for a given deer image. For example, I cropped whole deer, nose part of the deer, tail part, horns separately for the deer and added in the dataset.

Now I have 32 deer images in the training and 29 images of non-deer images.

Procedure for SVM classifier is same. Now selecting the different window size for each image and different threshold value to get the perfect bounding box around the deer.

Output of the window matching after NMS and increased dataset.

We can visually check the quality of the object detection. Else if we have the ground truth bounding box we can also check the precision, recall and f1 score.

Algorithm for the evaluation of the detection performance for window matching technique-

1. Load the ground truth image that has a bounding box. Also load the predicted bounding box coordinates
2. Extract ground truth bounding box coordinates.
3. Calculate the intersection between the predicted and ground truth bounding boxes using basic geometry formula.
4. Then calculate the union between the predicted and ground truth bounding boxes.
5. Calculate the Intersection over Union (IoU) between the predicted and ground truth bounding boxes by dividing the intersection by the union.
6. Calculate the precision, recall, and F1 score using inbuilt functions.

Observations and important points in the Window matching

1. Computationally expensive: In the sliding window detection we search over many potential locations and scales, which can be computationally expensive.
2. High false positives: This approach can produce false positives when the window is sliding on an area of the image that does not contain the deer. This can be seen in the above results.
3. Overlapping detection- Many times it can be observed that windows overlap over each other, and objects are not properly covered in the window.
4. It is very time consuming to set the size of the bounding box and set the threshold for NMS. There are sophisticated techniques that can be applied to reduce the time required for setting the window size, step size and threshold.
5. If we run the SVM model again then the images in the training and testing and training set are different as they get shuffled during the training process. So, the accuracy will be different. HoG features will be different. So, after each run, we need to set the window size and step size again. We can avoid this by setting the random_state parameter in the train test split but it is not advisable as model will not be trained in random fashion and it will not give good results on unseen data.

Thank You