Introduction to Computer Vision (ECSE 415)
Assignment 3: Classifier, Object Recognition

DEADLINE: October, 20th
Please submit your assignment solutions electronically via the **myCourses** assignment dropbox.

The submission should include a single Jupyter notebook. More details on the format of the submission can be found below. Submissions that do not follow the format will be penalized 10%.

The assignment will be graded out of a total of **100 points**. There are *50 points* for accurate analysis and description, *40 points* for bug-free and clean code, and *10 points* concerning the appropriate structure in writing your report with citations and references.

Each assignment will be graded according to defined rubrics that will be visible to students. Check out MyCourses, the "Rubrics" option on the navigation bar. You can use **OpenCV**, **Scikit-learn**, and **Numpy** library functions for all parts of the assignment except those stated otherwise. Students are expected to write their own code. (Academic integrity guidelines can be found **here**).

Assignments received late will be penalized by 10% per day.

Submission Instructions

1. Submit a single Jupyter notebook consisting of the solution of the entire assignment.
2. Comment your code appropriately.
3. Give references for all codes which are not written by you. (Ex. the code is taken from an online source or from tutorials)
4. Do not forget to run **Markdown** ('Text') cells.
5. Do not submit input/output images. Output images should be displayed in the Jupyter notebook itself.
6. Make sure that the submitted code is running without error. Add a **README** file if required.
7. If external libraries were used in your code please specify their name and version in the **README** file.
8. We are expecting you to make a path variable at the beginning of your codebase. This should point to your working local (or google drive) folder.
   Ex. If you are reading an image in the following format:

   ```
   img = cv2.imread ( '/content/drive/MyDrive/Assignment1/images/shapes.png' )
   ```

   Then you should convert it into the following:

   ```
   path = '/content/drive/MyDrive/Assignment1/images/'
   img = cv2.imread(path + 'shapes.png')
   ```

   Your path variable should be defined at the top of your Jupyter notebook. While grading, we are expecting that we just have to change the path variable once and it will allow us to run your solution smoothly. Specify, your path variable in the **README** file.
9. Answers to reasoning questions should be comprehensive but concise.

How to Access Data

In addition to this PDF file, you have been provided with a folder named "Data" that contains all the necessary datasets for your assignment.

For the first question, you should utilize the "HoG" folder within the "Data" directory. Inside the "HoG" folder, you will find two subfolders: "Train" and "Test," which respectively contain the training and testing sets of images.

For the subsequent part of the assignment, you will be working with another dataset, the details of which will be provided later. To access this dataset, navigate to the "Data/FaceRecognition" path. Inside this path, you will find two items:

1. A folder named "GeorgiaTechDataBase," which contains the imaging dataset.
2. A file named "mapping.csv," which contains the labels corresponding to each image in the dataset.

Folder Tree:

```
Data/

 HoG/
    Train/
       (Training set images)
       ...
    Test/
        (Testing set images)
        ...

 FaceRecognition/
     GeorgiaTechDataBase/
        (Imaging dataset)
        ...
     mapping.csv
```

1 Classification using HoG (25 points)

Imagine you've been hired by a company specializing in restoring and cataloging data from NHL teams. They've entrusted you with a collection of both old and new images depicting various hockey matches and scoreboards spanning several years. Your goal is to extract and rejuvenate historical game results. Upon examining these scoreboards, you notice the logos of each team, some of which are familiar, while others are old versions, closely resembling the current ones but not quite the same.

Recalling your Introduction to Computer Vision course from Fall 2023, you decide to construct a logo classification system using Histogram of Gradient (HoG) features. To expedite the process, you contemplate manually labeling some logos based on your knowledge. However, given the sheer volume of images at your disposal, manual classification would prove excessively time-consuming. Consequently, you opt to test your HoG classification method on a smaller subset of the data.

1.1 Build your classifier with training images

For this task, you have a set of training images; five 'Montreal Canadiens' logos and five 'Toronto Maple Leafs' logos. These can be found here: 'data/Q4/training' folder. Examples are found in Figures 1 and 2.

1. Resize the training images to 128 x 128.
2. Compute HoG features of size (32,32,8) for all training images.
3. Apply blocknorm in 4 x 4 cell neighborhoods.
4. Fit a nearest neighbor classifier with three neighbors. Use **KNeighborsClassifier** from sklearn library.

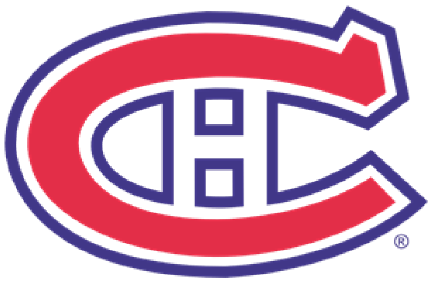

Figure 1: Training example for Montreal Canadiens Logo (source)

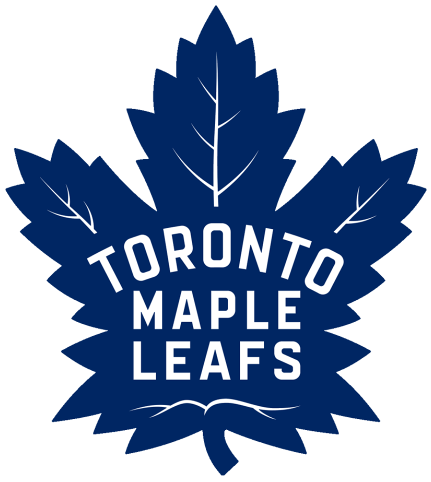

Figure 2: Training example for Toronto Maple Leafs.

1.2 Classify Test Images

For this part, You are given a set of test images for these two teams; two ‘Montreal Canadiens’ logos. These can be found here: ‘data/HoG/test’ folder. (Examples can be found below.) You are now asked:

1. Compute HoG features for all the test images.
2. Display HoG features for all the test images.
3. Classify the test images using the classifier you built earlier.
4. Does the classifier work on all the test images? Will HoG work if images of logos undergo random rotation? If yes, which property of HoG avails this feature? If not, discuss the sequence of computer vision technique(s) that can be used in order to regain the uniform orientation of the logos.

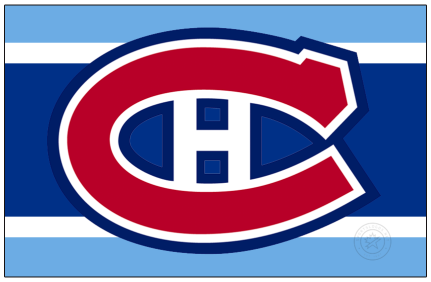

Figure 3: Testing example for Montreal Canadiens Logo.

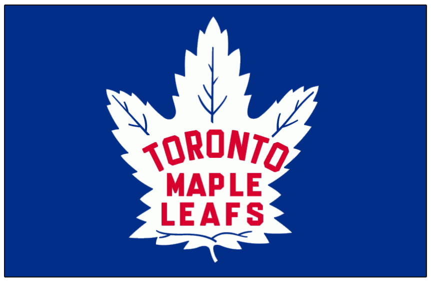

Figure 4: Testing example for Toronto Maple Leafs.

2 Task 2: Face Recognition System

2.1 Data Processing (10 points)

In this part of the assignment, you will be working with face images from the publicly available Georgia Tech Face Database, which is provided along with this assignment. This database comprises images of 50 different individuals, each represented by 15 color JPEG images. These images depict frontal and/or tilted faces with varying facial expressions, lighting conditions, and scales. Additionally, a CSV file is included with the dataset to associate image names with subject IDs, and the images themselves come in different resolutions. Please note that you can make use of either "NumPy" or "Pandas" Python libraries to read the CSV file.

Your task is to prepare this dataset for use in a face recognition system. To do this, you will need to convert all the images to grayscale and resize them to a resolution of 128x192 pixels.

Next, you'll need to randomly divide the dataset into training and test sets according to the following guidelines. Please note that every time you retrain your system, a new random selection should be made, unless you are debugging, in which case you can use a fixed random seed to ensure consistent train/test splits across runs.

Dataset Split:

- **Training Set:** Randomly select 80% of the total images provided. These will constitute your training set.
- **Test Set:** All remaining images of the dataset that were not included in the training set will be used for testing.

As an additional task, you are required to display a total of 10 random images from the training set, along with their respective names and labels. Furthermore, you should create histograms to visualize the frequency distribution of each image class (in this context, the subject ID or label) for both the training and testing sets.

2.2 Eigenface Representation (25 points)

Now that you have prepared your training dataset, it's time to create an eigenface representation through Principal Component Analysis (PCA). However, you are not allowed to use the built-in PCA functions in OpenCV or Scikit-Learn. Instead, you should implement the efficient Snapshot method for PCA, as covered in class (Lecture 8, Slide 55), using NumPy. (**The implementation is worth 15 points**)

Your tasks for this part of the assignment are as follows:

1. **Plot the Fraction of Total Variance vs. Number of Eigenvectors: (3 points)** Implement PCA using the Snapshot method and calculate the fraction of total variance explained by each eigenvector. Then, create a plot that shows how this fraction of total variance changes as you increase the number of eigenvectors used.
2. **Plot Normalized Variance (Eigenvalues) vs. Eigenvector Index: (2 points)** Plot the normalized variance (eigenvalues) against the eigenvector index used for computation. This plot will provide insights into the relative importance of each eigenvector.
3. **Discuss the Need for All Eigenvectors: (3 points)** Based on your analysis, discuss whether you need to retain all the eigenvectors to effectively represent the data. Consider factors such as the amount of variance explained and computational efficiency.
4. **Display the First 5 Eigenfaces: (2 points)** Visualize the first 5 eigenfaces that you obtained through PCA. These eigenfaces represent the principal components of the dataset and can be crucial for understanding the underlying structure of the face data.

By completing these tasks, you will gain a deeper understanding of the PCA process and the significance of each eigenvector in representing the data.

2.3 Face Recognition with Multiple Classifiers(40 points)

In this part, you will explore face recognition using three different classifiers: Linear Support Vector Machine (SVM), Random Forest, and k-Nearest Neighbors (KNN). Each of these classifiers has unique characteristics that can influence their performance in face recognition tasks.

Part A: Linear Support Vector Machine (SVM) (5 points)

1. Implement a Linear Support Vector Machine (SVM) classifier for face recognition using the eigenfaces as feature vectors. You can use libraries like Scikit-Learn to facilitate this task. Perform hyperparameter tuning on the regularization parameter for Linear SVM. Explain why you chose this specific value for your hyperparameters.
2. Train the Linear SVM classifier on your training dataset with eigenfaces as input features.
3. Evaluate the Linear SVM classifier's performance on your test dataset and report the recognition accuracy.

Part B: Random Forest Classifier (5 points)

1. Implement a Random Forest classifier for face recognition using the eigenface features. You can use libraries like Scikit-Learn to facilitate this task. Perform hyperparameter tuning such as the number of trees for Random Forest. Explain why you chose this specific value for your hyperparameters.
2. Train the Random Forest classifier on your training dataset with eigenfaces as input features. Experiment with different hyperparameters to optimize classification performance.
3. Evaluate the Random Forest classifier's performance on your test dataset and report the recognition accuracy.

Part C: k-Nearest Neighbors (KNN) Classifier (5 points)

1. Implement a k-Nearest Neighbors (KNN) classifier for face recognition using the eigenfaces as feature vectors. You should use a suitable distance metric (e.g., Euclidean distance) to measure similarity between eigenface representations. Perform hyperparameter tuning such as the value of K for KNN. Explain why you chose this specific value for your hyperparameters.
2. Train the KNN classifier on your training dataset with eigenfaces as input features. Experiment with different values of k to find an optimal setting.
3. Evaluate the KNN classifier's performance on your test dataset and report the recognition accuracy.

Part D: Comparison and Analysis (15 points)

1. Compare the recognition accuracies achieved by the Linear SVM, Random Forest, and KNN classifiers. Discuss the strengths and weaknesses of each classifier in the context of face recognition. You can convey your points through plots, tables, and comparisons.
2. Analyze any observed differences in performance. Consider factors such as computational efficiency, robustness to variations in lighting and pose, and sensitivity to hyperparameter settings.
3. Reflect on the potential use cases where one classifier might be preferred over the others for face recognition tasks.

Part E: Per-Class Accuracy and Histogram Analysis (10 points)

1. For one of the classifiers you've implemented (e.g., Linear SVM), calculate the per-class accuracy, also known as the reference accuracy. This involves measuring how well the classifier performs on each individual class (subject ID) in the dataset. [Reference]
2. Compare the per-class accuracy values against the histograms of the frequency of each image class for both the training dataset and the testing dataset.

3. Analyze whether there is any correlation or pattern between the per-class accuracy and the histograms. Do certain image classes consistently have higher or lower accuracy? Write your observations and insights regarding these correlations.