Computer Vision
Spring-2021
Assignment 4 - Face detection & Scene Recognition
Posted on: 18/03/2021
Due on: 23:59hrs, 02/04/2021
TAs - Gowri, Prajwal

Guidelines

1. Follow the specified repository structure. `src` folder will contain the Jupyter notebooks used for the assignment. `images` folder will contain any images used for the questions.
2. Commit your work regularly to avoid losing progress. Make sure you run your Jupyter notebook before committing, to save all outputs.
3. The report should contain description of the problem, algorithms and results. It should be written in markdown, in the notebook itself.
4. Make sure that the assignment that you submit is your own work. **Any breach of this rule could result in serious actions including an F grade in the course**.
5. The experiments and report writing takes time. Start your work early and do not wait till the deadline.
6. You are not allowed to use inbuilt functions that directly solve the tasks assigned. Confirm with TAs regarding whether some function can be used, when in doubt.

Questions

1. **Face Detection**

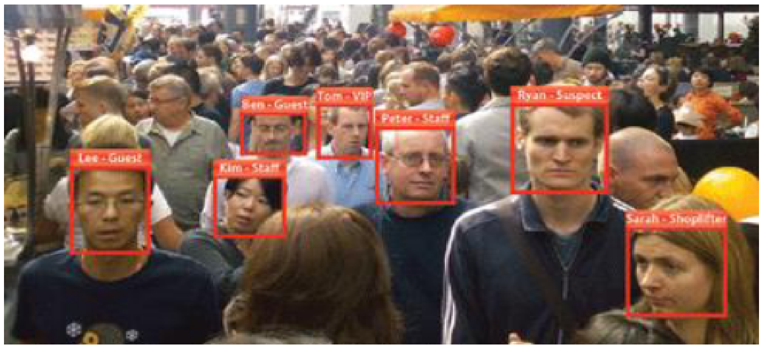


Figure 1: Face Detection

1. The task is detecting (marking out) faces in an image.
2. We will do this using a cascade architecture as in Viola-Jones algorithm, link to the paper here

3. It works on grayscale images and it has two phases training and detection. The Viola Jones algorithm has four main steps which you are expected to implement.
   (a) Selecting Haar-like features
   (b) Creating an integral image
   (c) Running AdaBoost training
   (d) Creating classifier cascades
4. **Selecting Haar-like features :** There are 3 types of Haar-like features that Viola and Jones identified in their research: Edge features, Line-features and Four-sided features. The value of a feature is calculated as a single number: the sum of pixel values in the black area minus the sum of pixel values in the white area.

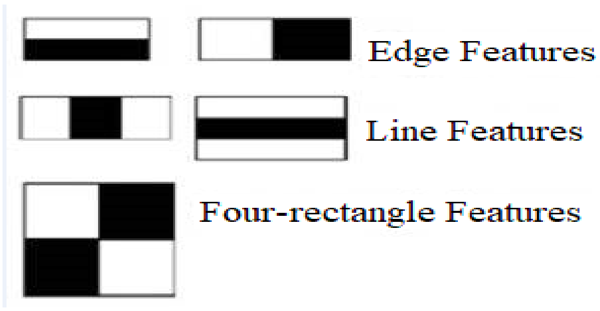
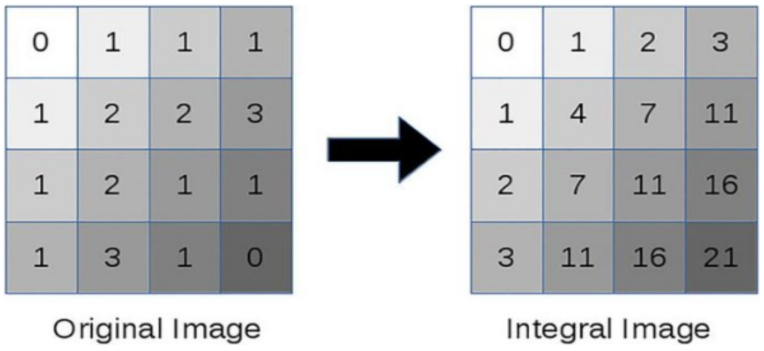


Figure 2: Harr features

5. **Creating an integral image :** In the previous section, we have seen that to calculate a value for each feature, we need to perform computations on all the pixels inside that particular feature. To avoid this we construct an integral image,where the value of each point is the sum of all pixels above and to the left, including the target pixel. This allows us to quickly calculate the value for a feature just by probing 4 corners of the required rectangular area.



0	1	1	1
1	2	2	3
1	2	1	1
1	3	1	0

Original Image

0	1	2	3
1	4	7	11
2	7	11	16
3	11	16	21

Integral Image

Figure 3: Integral Images

6. During training, the algorithm sets a minimum threshold to determine whether something can be classified as a feature or not. The algorithm shrinks the image to 24 x 24 and looks for the trained features within the image. For the algorithm to

detect faces we need to give it both facial images (one can use FDDB Dataset) and non-facial image data too. We can also use horizontal flip augmentation on images.

7. **Running AdaBoost training :** The number of features that are present in the 24×24 detector window is nearly 160,000, but only a few of these features are important to identify a face. We use AdaBoost to get an ensemble of weak features to get a strong classifier. Table 1 of the paper linked gives you the exact algorithm. Please refer to section 3.1 for more details.

8. [**BONUS**] **Creating classifier cascades :** This step is done to minimize the time taken by the algorithm in detection. We set up a cascaded system in which we divide the process of identifying a face into multiple stages. In the first stage, we have a classifier which is made up of our best features and see if it is present in the image within the subwindow. If it is not in the subwindow, then we don't even look at the subwindow, we just discard it. Then if it is present, we look at the second feature in the subwindow.
   Please note creating classifier cascades is a bonus part.

9. **Detection :** The Viola-Jones algorithm first detects the face on the gray-scale image and then finds the location on the colored image. We slide a window on the image and check if there is a face in it or not on a resized window of 24*24. One can change the size of windows and step size too.

2. **Scene Recognition using Bag of Visual Words (BOVW)**

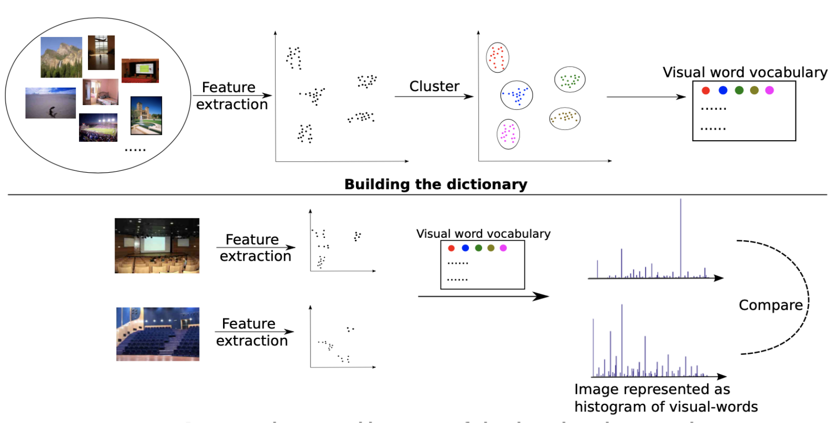


Figure 4: Bag of Visual Words

1. Implement a basic bag of visual words model by training and testing to classify images belonging to a subset of the SUN dataset into 8 classes. (**Note:** Use the folder SUN_data with the given train-test split and resize the images to 150x150 before performing BOVW).

2. Build a **vocabulary of visual words** by sampling local features from the training set and then clustering them with kmeans.

   (a) Compute dense-SIFT features on the images. (Implementing from scratch will be considered a bonus - densely sample with a certain step size)

(b) Represent the training and testing images as **histograms of visual words** by counting how many SIFT descriptors fall into each cluster in our visual word vocabulary. Use KMeans to do clustering and finding the nearest cluster centroid for each SIFT feature.

- [**BONUS**] Perform TF-IDF re-weighting on the histograms before normalization to enhance the importance of discriminative features and downweigh the uninformative features that occur in a lot of images.

(c) Normalize the histogram so that image size does not dramatically change the bag of feature magnitude.

(d) Experiment by varying parameters such as number of clusters, SIFT parameters, sampling density etc. for better performance.

3. Perform classification by training one-vs-all linear SVMs using sklearn. To get higher accuracy, tune the regularization parameter 'lambda' and report the final accuracy obtained.

4. Plot a confusion matrix without and with normalization to estimate the performance of the model. Display a few images that have been correctly and wrongly classified along with their predicted labels.

5. [**BONUS**] Implement BOVW with **spatial pyramid matching** (maximum 3 levels, L = 2) to retain spatial information by dividing the image into a small number of cells, and concatenating the histogram of each of these cells with a suitable weight to the histogram of the original image. Refer to section 3 of the paper *"Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories"* by Lazebnik et al. for more details.

   (a) Divide the image into $2^l \times 2^l$ cells where l is the layer number. Each cell is considered as a small image and we can count how often each visual word appears. This results in a histogram for every single cell in every layer.

   (b) Finally to represent the entire image, we concatenate all the histograms together after normalization by the total number of features in the image.

   (c) While concatenating all the histograms, set weights of 1/4, 1/4 and 1/2 for layer 0, 1 and 2 respectively for a 3 layered spatial pyramid as given in the reference paper.

   (d) After computing the histograms of the finest layer, save them so that the histograms of coarser layers can then be aggregated from finer ones. Normalize the histogram after aggregation.

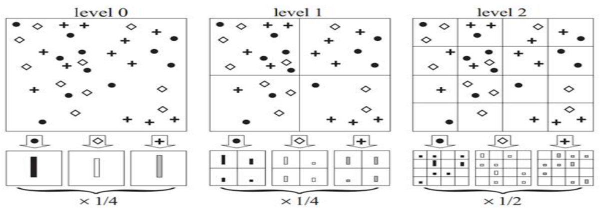


Figure 5: Spatial Pyramidal Matching for L=2