STATE UNIVERSITY OF NEW YORK AT BUFFALO

Programming Assignment 3

Pushpinder walia
UB id- pwalia
50318705
Nikhil Arora
UB id- nikhilar
50320282

May 16, 2020
Equal percentage of contribution was given by each member.

CONTENTS

1 INTRODUCTION

The purpose of this project is to implement the Viola-Jones algorithm to detect the faces in the wild from the given FDDB data set of faces. Face-detection is a handy tool used in various platforms ranging from the camera industry to the defense industry. Viola-Jones is a popular algorithm used by almost every industry as it is easy to implement and is much less costly than other methods available. This report explains in detail the data set, the algorithm we used, different approaches used to reach the goal, final results, possible improvements, and discussions.

2 VOILA JONES ALGORITHM

Viola-Jones algorithm is one of the most popular algorithms in the industry for face detection introduced by Paul Viola and Michael Jones in 2001. A few of the advantages of this model are that it is robust, which means it has a very high true positive rate. It can work in real-time also. This algorithm is used only for face-detection and not for face recognition.

2.1 INTEGRAL IMAGE

According to this algorithm, some parts of the face are darker than the other parts. Therefore, if we take a rectangle image matrices on each part of the face and subtract them, we will be able to tell which is a face and which is not a face. To make this calculation easier, Viola-Jones introduced the concept of an integral image. Here, we calculate the addition of every adjacent pixel in the taken rectangle and store it in a different matrix called an integral image matrix.

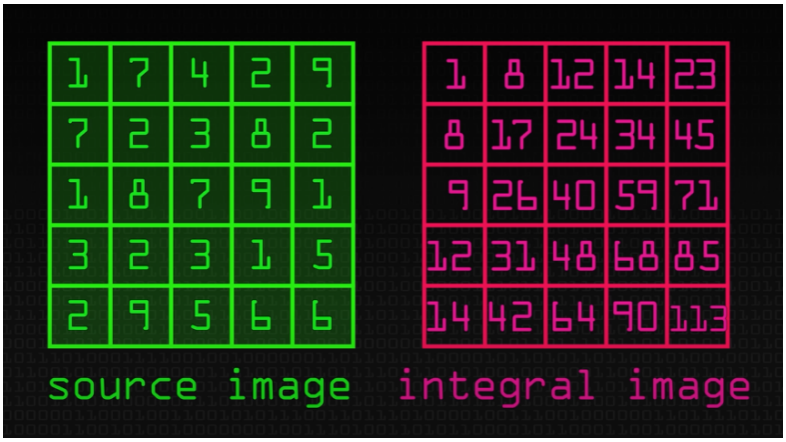

Figure 2.1: Integral Image

The integral image at location x, y contains the sum of the pixels above and to the left of x,y.

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

Figure 2.2: Integral Image Formula

2.2 Features

To extract the features from the images, we have to convert the face and non-face data set images to 24x24 images so that we can get the location of the features more accurately. To achieve that, we have to turn the ellipse features to the rectangle by getting the nearest square of the image shape.

This algorithm uses Haar features, which include three different rectangle features, namely, two rectangle features, three rectangle features, and four rectangle features. In the image

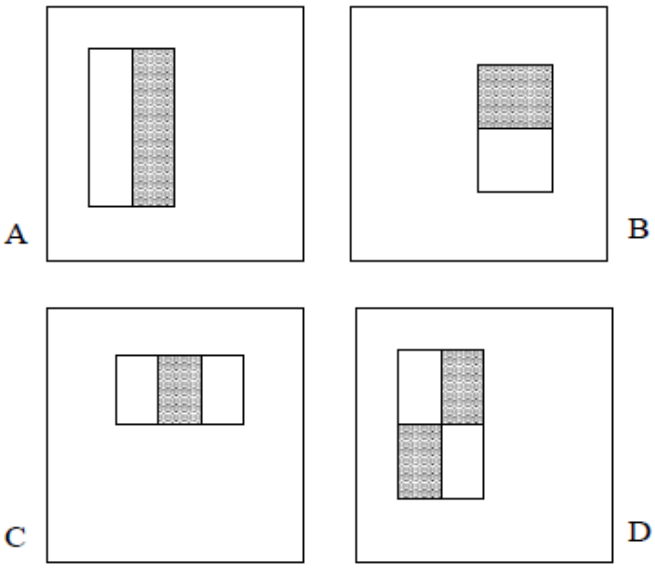

Figure 2.3: Different Rectangular Features

shown above, we can see that one part of the rectangle is darker, and the other is lighter. The lighter part of the rectangle is subtracted from the darker part, which gives us the features.

2.3 Adaboost

Given an image of 24x24 size, there will be approximately 160,000 rectangular features. Adaboost is used to select the small features and train the classifier. To achieve that, we im-

plemented a weak learning classifier algorithm that determines an optimal threshold. This algorithm selects the single rectangle feature which best separates the positives and negative examples to the approach. The approach for implementing the algorithm is shown below in the figure: Below is the example of one of the rectangular features selected on the face. The

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
     $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
     so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
     $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
     where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
  $$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
  where $\alpha_t = \log \frac{1}{\beta_t}$

Figure 2.4: Algorithm of AdaBoost

hypothesis is that the first feature region on the eyes is darker than the region of the nose. In the second feature, the range of eyes is darker than the bridge between the nose.

Figure 2.5: Examples of rectangular features on the face.

3 IMPLEMENTATION

1. To implement the algorithm, first, we need to extract the faces and non-faces data set. For faces data set, the FDDB data set available online, which has approximately 2800 images of 5700 faces, is used. For non-faces data set CBCL data set of non-faces is used. In a total of roughly 5000 images are used to train and test the model.
2. The images are converted to 24x24 and stored as .npz files after extracting the data set for both sets of images.

```
def face():

    final_faces = list()

    for i in range(1,11):

        filename = "FDDB-fold-%02d-ellipseList.txt" %i
        file_open = 'FDDB-folds/' + filename

        final_faces.extend(face_out(file_open))
    print(np.shape(final_faces))
    np.savez("face_images.npz", images=final_faces)

def non_face(folder):

    non_face = list()
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename))
        gry_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        non_face_patch = cv2.resize(gry_img, (24,24))
        non_face.append(non_face_patch)
    print(np.shape(non_face))
    np.savez("non_face_images.npz", images=non_face)
```

Figure 3.1: Code snippet

3. After we get the data set in the .npz files, the next step is to calculate the integral image by the method explained above.
4. We cannot get the location of the features directly from the image as every face's shape is different, so we have to convert the ellipse to a rectangle and get the location.
5. Once we get the rectangle feature's location, we were able to extract the haar features from the image.

3.1 THINGS WE TRIED TO IMPLEMENT BUT WERE NOT ABLE TO COMPLETE

After extracting the haar features, the next step is to implement the Cascade and AdaBoost, which we tried to apply but were unsuccessful. Even though this concludes our computer vision course, we still look forward to working to implement this algorithm completely to enhance our knowledge of the subject.

4 RESULTS AND DISCUSSIONS

In the output of the code, we can successfully see the location of the rectangle features and extracted haar features.
Future work, which includes implementing cascade and AdaBoost, is required, enabling us to detect the faces in the wild with high accuracy.

5 REFERENCES

- http://vis-www.cs.umass.edu/fddb/
- Rapid Object Detection using a Boosted Cascade of Simple Features - Paul Viola, Michael Jones
- http://cbcl.mit.edu/software-datasets/heisele/facerecognition-database.html