Assignment - 3 Faces Detection in the Wild

Nihar Patel UB-ID: 50318506
Manaswin Oddiraju UB-ID: 50320725
CSE 573: Computer Vision and Image Processing

1 Introduction

The goal of this project is to implement the Viola-Jones Face Algorithm [3].There are many classifiers used for image classification and face detection.

Viola-Jones algorithm uses an ensemble approach. It relies on various classifiers,which are limited to one feature each. These classifiers are weak classifiers.It means these individual classifiers are less accurate, generates more False Positives. But when you combine all those weak classifiers, it produces a strong classifier which detects the objects or face.

2 Implementation

Our Implementation of the algorithm consists of the following steps:

1. Training
    - Feature Classification using Haar Features
    - Computing feature values of the training set
    - Extracting best classifiers using adaboost and building the strong classifier
2. Detection
    - Scale given input image
    - Applying the strong classifier on sub-windows of the image

Below Image shows the overall stages of our implementation of the algorithm in the project. See Figure 1.

2.1 Feature Classification using Haar Features

Before extracting the Haar Features, we consolidated our training dataset with images consisting of faces and non faces . For training, we used a total of 7000 faces (3500 faces and 3500 vertically flipped faces), which were collected from the **Faces in the Wild** Data-set [1]. There were total 7000 non-faces data set which we created by randomly selecting a sub window which didn't contain

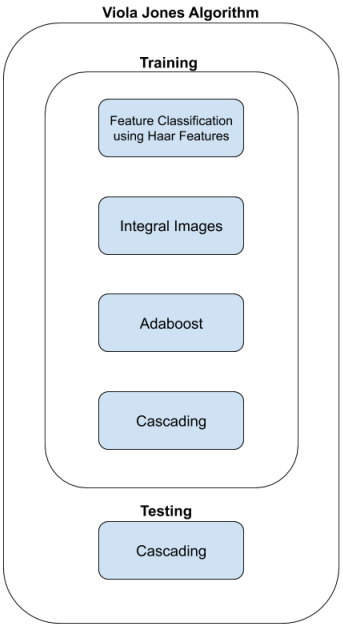

Figure 1: Viola Jones Algorithm Overview.

faces from the dataset . For testing, we used **256** test images as mentioned in the guidelines.
In our algorithm, as represented in the paper we created almost **90.64k features** while each pattern will go up till max size of the sub-windows that is **21x21**.Compared to the windows size of **24x24** used in the original paper Figure 2 shows the various types of Haar features considered in this project.The following snippet was used for feature extraction for one type of Haar features.

```
for x in range(0,img_width):
    for y in range(0,img_height):
        for w in range(1,img_width-x,2):
            for h in range(1,img_height-y):
                A=(x,y)
                B=(x+(w//2),y)
                C=(x+w,y)
                D=(x,y+h)
                E=(x+(w//2),y+h)
                F=(x+w,y+h)
                add = [E,E,A,C]
                sub = [F,D,B,B]
                add = np.asarray(add)
                sub = np.asarray(sub)
                sum_ = ((add),(sub))
                feature1.append(sum_)

features.extend(feature1)
print("Feature 1 created:",len(feature1))
```

```
return features
```

We pre-compute the points on the image corresponding to each feature.This pre processing helps avoid computing the feature points on every training image and also helps in vectorizing feature extraction from the integral image. The extracted points are saved into a binary file(.npy) for further use.

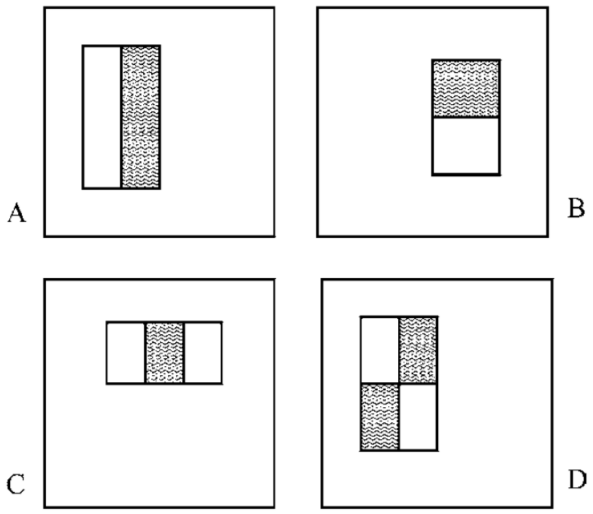

Figure 2: Haar Features different pattern

Since the algorithm requires iterating over all features, they must be computed very efficiently. To do this, Viola and Jones introduced the Integral Image.The Integral Image was first introduced to us in 1984, but was not properly introduced to the world of Computer Vision till 2001 by Viola and Jones with the Viola-Jones Object Detection Framework.The Integral Image is used as a quick and effective way of calculating the sum of values (pixel values) in a given image – or a rectangular subset of a grid (the given image).The output of the image we get by cumulative addition of intensities on subsequent pixels in both horizontal and vertical axis.
Below shows our code for the integral image:

```
def integral_image(img):
     sum_img = np.zeros(img.shape)
    ii_img = np.zeros(img.shape)
    ii_img[:,0] = img[:,0]
    for x in range(1,img.shape[1]):
        ii_img[:,x] = ii_img[:,x-1] + img[:,x]

    sum_img[0,:] = ii_img[0,:]
    for y in range(1,img.shape[0]):
        sum_img[y,:] = sum_img[y-1,:] + ii_img[y,:]
return sum_img
```

2.2 Extracting Feature Values

Once, we got all possible features of the size of the sub-window. We now apply all the images from the data set into all these 90640 features and extract the feature values. As mentioned in the previous section, we preprocessed the all possible features into *.npy* file hence, in this section we compute the features by imposing the feature points onto the integral image. As the feature extraction process is independent with respect to images, we were able to parallelize the feature extraction to run on theoretically any number of cores. We, ran the algorithm on the 4 cores available to us thereby quadrupling the speed of feature extraction.
The following flowchart provides an overview of the algorithm:

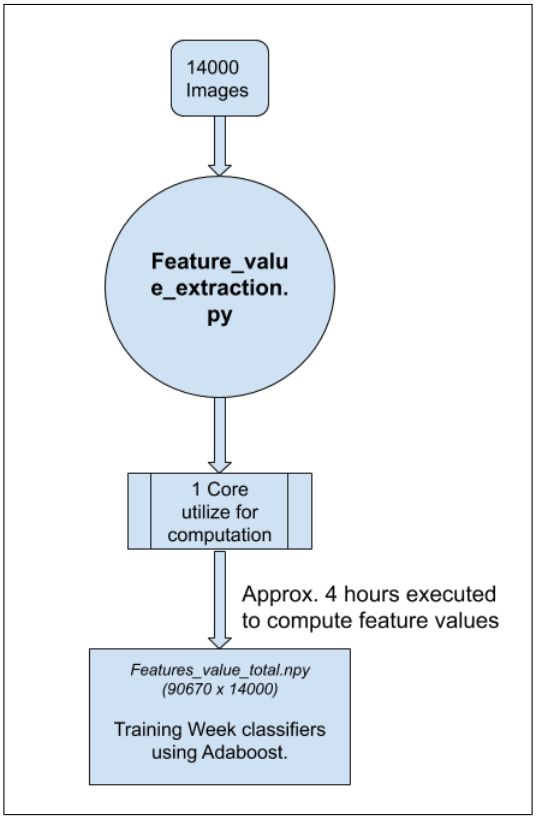

Figure 3: Serial Processing

Thus, we calculated the **90640 feature values** for all of **14000 Images**. These values were stored in a 2D numpy array for easy further processing. Here, each cell denotes the value of a

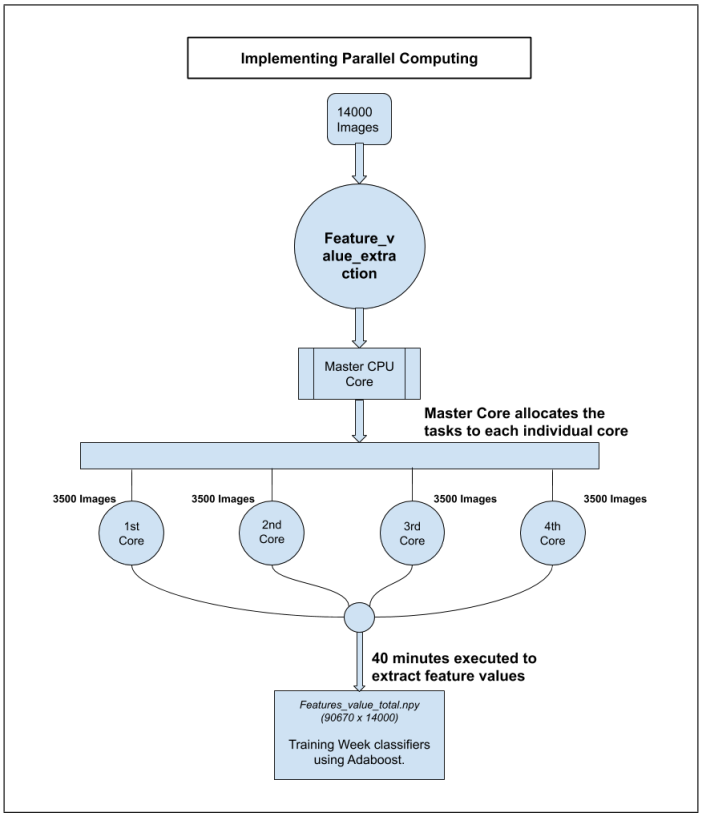

Figure 4: Parallel Processing

particular feature for a particular image. Since this step is also time-consuming. For speeding up the whole training phase, We saved this matrix into a numpy file *.npy* file. Below table shows the approx time consumed for computing feature values of all the images. Table 1 shows the computational time processed.

Sr.No.	Without MPI	With MPI
1	16 Hours	4 Hours

Table 1: Approx Computation time to calculate 90640 x 14000 Feature Values

2.3 Adaboost

Adaboost is the training scheme which provides us with the constituents of the final strong classifier. Each iteration of Adaboost gives a weak classifier to be included in the final strong classifer.We started the trianing process by having the minimum value of thresholds to be the average of feature values of images containing faces and non-faces. One thing to note is that this high memory requirement prevented us from parallelizing this part of the code. We acknowledge that the demands on the memory requirements could have been relaxed if we had used loops instead of vector operations, but the sacrifices in training speed were enormous so we chose to go ahead with this method.

The inputs for our training code are the feature values, ground truth of training data and the number of iterations we want to run the training for.
For our project, we trained for 6000 iterations which will give the same number of weak classifiers along with their α values. Below table shows the variations we tried:

Sr.No.	Iterations	Train Detection Accuracy
1	500	93.13%
2	1000	96.73%
3	6000	97.3%

Table 2: Training Accuracy for both face and non-face data.

While training, selecting best weak classifier for iteration took 10sec, thus for 500 best weak classifiers it took around 1.5 Hours to train with accuracy of 93.13%. With 6000 iterations, the algorithm took 16.6 Hours to train and achieved the accuracy of 97.3%

2.3.1 Drawbacks

Since the training part is much costly in computation [2] time(about 13 seconds per iteration with a training set of 14000 images), we used a virtual instance with 4-cores and 32GB memory on Google Cloud compute. Our initial trial took longer due to some unseen errors in the code, but we were able to debug the program and train the classifier.

2.4 Detection

The adaboost training will output a set of weak classifiers and the linear the combination of all these weak classifiers is defined as a strong classifier and is used to detect if a given face has an image or not. Below is the code given that we implemented in our project:

```
def faces(img,features,alpha,thresholds,polarity):
    fvals = feature_extraction(img,features)
    fvals = np.reshape(fvals,(len(fvals),1))
    fvals = np.multiply(fvals,polarity)
    weakclass = (fvals<thresholds)
    weakclass = np.reshape(weakclass,alpha.shape)
    test = np.multiply(weakclass,alpha)
    check = np.sum(test)>=0.5*np.sum(alpha)
    return check
```

The inputs for the function are sub-window of the image, all weak classifiers along with the alpha(α) values, thresholds(θ) and polarity(p) for all the train data-set. The output will be binary results saying If the image is a face or not.
For each image passed we scale the apply the sub-window sort of like a convolution window with a stride equal to 5% of the scaled image dimension. Then after obtaining all the matches we select the positives which have more matches(detected at multiple scales) in a local neighbourhood (10% of image width) and then output our results.This filtering is done to avoid multiple windows in the same neighbourhood.

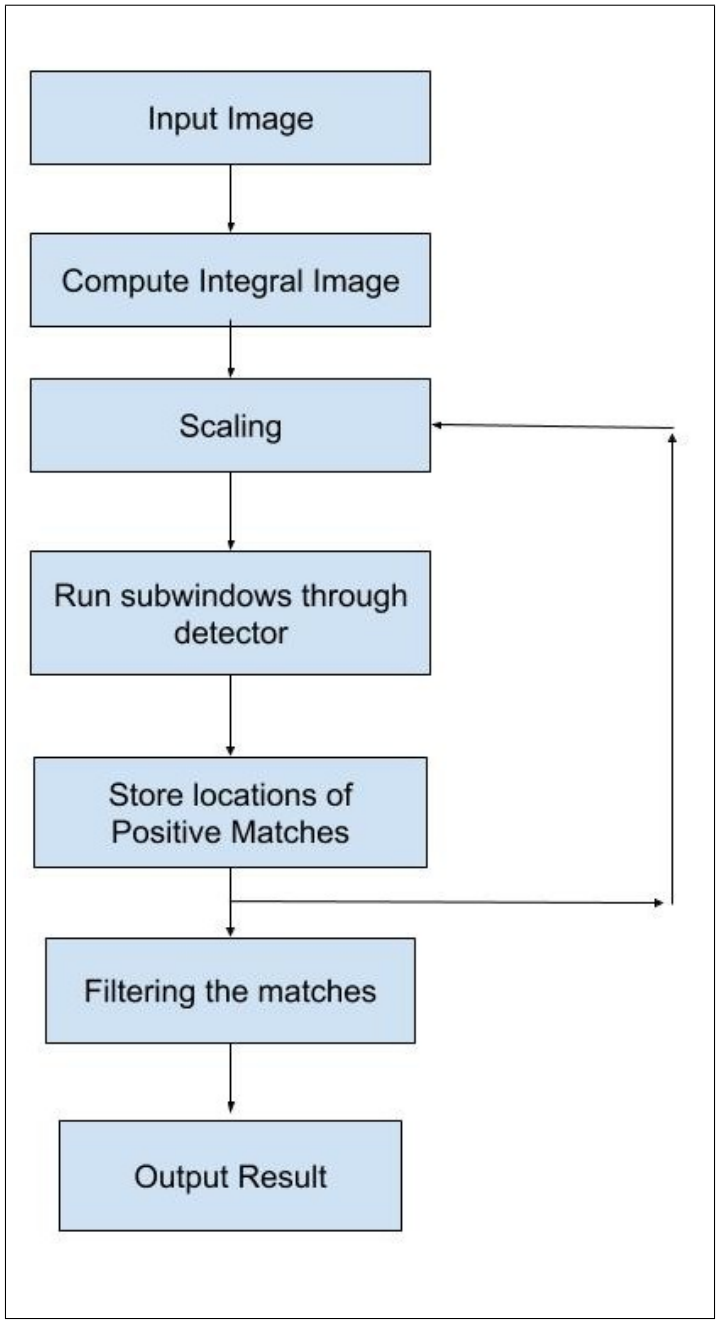

Figure 5: Detection Algorithm

3 Results

Below images are the output of our model:

Figure 6: Face Detected Images

4 Analysis

We found a few drawbacks of our classifier after testing. First thing being, our window size of 21x21 may have been too small. Thereby having to run the image through the detector more number of times at multiple scales compared to a window size of 24x24.
Normalization of the images before training would've produced better robustness to varying scene and lighting conditions and would've reduced the number of false positives.Being short on time is the only reason we are not implementing this as we have to retrain our model again. Implementing a cascade might have given better results again with respect to the number of false positives
Also the another error we figured out is that there were many faces which are not strictly a frontal face. As we trained with only frontal faces so our model can't detect those properly and as a result gives a lot of false positives in those images. Possible improvement is if we include the non-frontal faces in the detection, then there are chances of reduction in the false positives in the data-set. Also,our training non-face data lacked the diversity that was found in the backgrounds of the test images and retraining our classifier on a better data set could improve accuracy
The following table shows the amount of time we spent for each section of the project:

Sr.No.	Section	Time
1	Creating Faces and Non-Faces Data-set	5%
2	Feature Creation using Haar Features	10%
3	Feature Value Extraction	15%
4	Adaboost Training & Debugging	55%
5	Testing	15%
	Overall	100%

Table 3: Time consumed by project

5 Conclusion

We have implemented an approach for face detection using Viola and Jones algorithm. The approach we used to construct a feature values matrix using parallel programming is approximately 4 faster than approach mentioned in the paper. We also used Google Cloud Platform to maximize the performance of our project. While this not only helped us in working on the project but also it helped us to learn more about Cloud computing and flexibility on working on various cross-platform.As mentioned in the paper, experiments on such a large and complex data-set are difficult and time consuming.
The following shows our final implementation of the Viola-Jones algorithm [3]:
Equal contributions were made to the project by both team members and The following table shows our work allocations for the project:

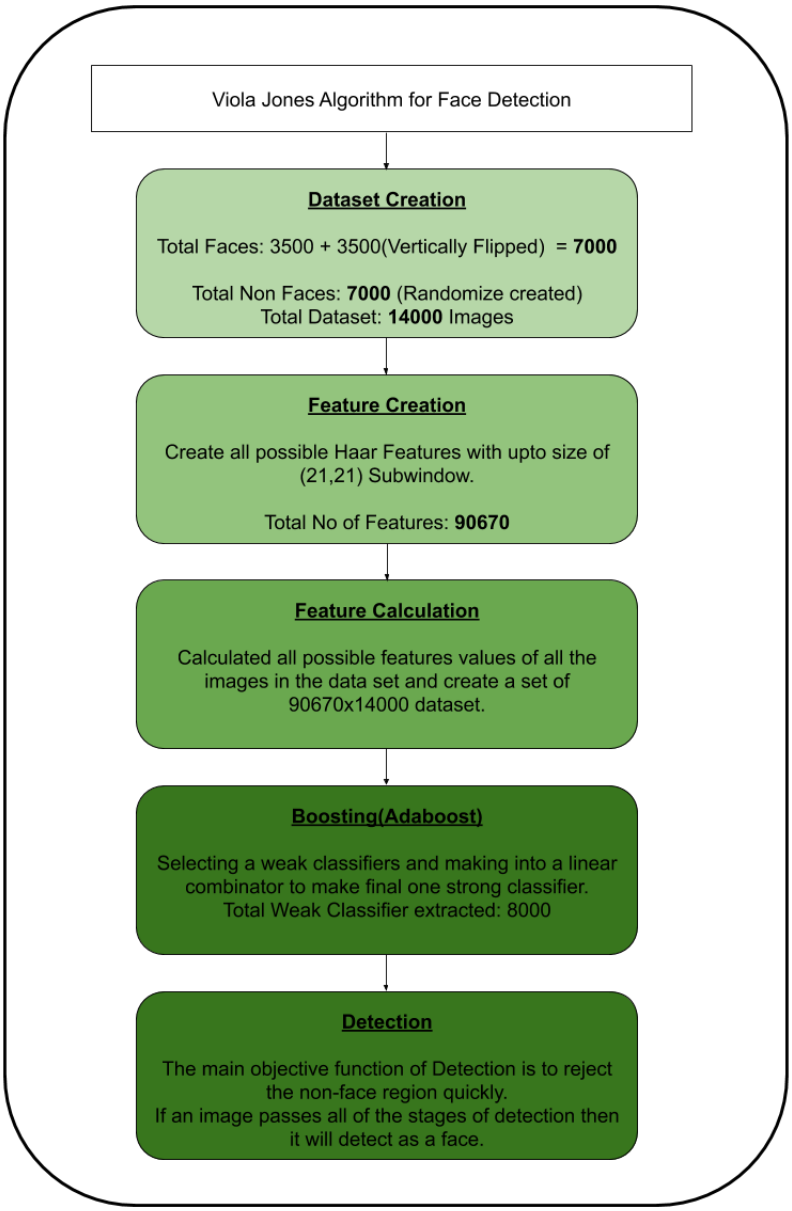

Figure 7: Flowchart of our assignment

Sr.No.	Section	Allocation
1	Creating Faces and Non-Faces Data-set	Manaswin & Nihar
2	Feature Creation using Haar Features	Nihar
3	Feature Value Extraction	Manaswin
4	Adaboost Training & Debugging	Manaswin & Nihar
5	Testing	Manaswin & Nihar
6	Report	Manaswin & Nihar

Table 4: Work Allocation

Acknowledgements

We thank the professor, David Doermann for giving us an opportunity to work on challenging projects. We would also like to thank all TA's for there constant support in solving the queries raised throughout the project and the course.

References

[1] Vidit Jain and Erik Learned-Miller. *FDDB: A Benchmark for Face Detection in Unconstrained Settings*. Tech. rep. UM-CS-2010-009. University of Massachusetts, Amherst, 2010.

[2] *Medium*. Jan. 2019. URL: `https://medium.com/datadriveninvestor/understanding-and-implementing-the-viola-jones-image-classification-algorithm-85621f7fe20b`.

[3] Paul Viola and Michael Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. Vol. 1. IEEE. 2001, pp. I–I.