Submitted by: SAI KRISHNA WUPADRASHTA
Unity id: swupadr

ECE 763 – COMPUTER VISION PROJECT 2 REPORT

TASK: Implementation of Adaptive Boosting for Face Detection

IMAGE DATA USED:

- I have used the FDDB(Face Detection and Data Set Benchmark) dataset for this project(extended from project 1), and have tried it out on the reference dataset provided by the instructor.
- n=1000 training images for face and non-face extracted.
- m=100 testing images for face and non-face extracted.
- I have set the resolution of the images extracted as 16x16. This can be changed to 20x20 and 60 x60 as well.
- Annotations were used to crop out face images while non-face images were randomly cropped from the background.

LAYOUT OF THE SOLUTION:

1. The face and the non-face images are used to extract a large number of positive ad negative images, where +ve are the face images and the -ve are the non-face images.
2. Values for Harr Features are computed , where these features correspond to a weak learner. The threshold between the face and non-face samples are determined. The Classification error for each weak learner is determined and the top 10 strong features before boosting are drawn.
3. The Adaboost Algorithm is then implemented, as given in the project specification.
4. Finally, the evaluation of the code built is done on the test dataset created, and the ROC curves, Histograms and Accuracy of the algorithm designed are seen.

ANALYSIS OF CODE MODULES (.py files) USED:

a) Main.py

This is the main master module where all the other modules are called for further actions. The main function is where the boosting classifier library is imported and the training epochs and the data to be trained is defined.

A positive data directory for face images and a negative face directory for non-face images is maintained here- both of which contain images of resolution 16x16. I am using 25 bins for boosting and parallelizing the process of boosting by implementing parallel computing – for which 8 cores are being used according to my code.

Carrying forward with the process of boosting, first the activations are saved cached for the weak classifiers , following which the strong classifier is generated, and the boosted data is concatenated into a new memory place. The time taken for implementation of all of the above is kept track of in this module. We observe the final result of the image before and after boosting in this module.

b) Img_processing.py

This module can be used to extract patches from image for all scales of the image, and then return the integrated images with the coordinates of the patches used, according to the viola jones algorithm. A vector of predictions (0/1) is returned after non-maximal suppression of the image, and the o/p is stored in the form of an array [x1,y1,x2,y2, prediction score]. After all this, the face that has to be detected within a bounded rectangle or box is taken care of this module, by considering the area of the bounding box-based on the integral image array obtained.

c) Utils.py

The Utils.py is the module where the image is taken as a 2-D array, and based on it a 3-D numpy arrat is generated for analysis. Generation of Haar Filters has been done here (4 types of filters-A,B,C,D). the images are then passed through these filters generated, and analysed.

d) Weak_Classifiers.py

The step where we generate the errors for the weak classifiers is done here. We have two separate functions to generate the real weak classifers and the adaboost weak classifiers.

e) Boosting_Classifer.py

The Boosting Classifier class is defined and the training activations are also calculated on another class here in this module. The weak classifiers are combined to form a strong classifier. After the training, predictions can be made by calling the self.sc (). The weak classifier errors are calculated in this function and the training results are cached to the self.viz for visualization of certain important metrics.

f) Visulaizer.py

This last module basically visualizes the various parameters for assessing the quality of the adaptive boosting performed so far. The ROCs upon thresholding the F(x), and the histograms for the strong classifier scores, the +ve scores and the -ve scores are also plotted.

Accuracy graphs are plotted considering the top 1000 weak classifiers and then the strong classifier error is computed in the end.

g) cross_check_with_built_in_VJ_features.py

This module consists of the Viola -Jones algorithm implementation using built-in module available within OpenCV. This was primarily used to compare the performance of the adaptive boosting model built for external test images.

Submitted by: SAI KRISHNA WUPADRASHTA
Unity id: swupadr

FINAL DETECTION RESULT :

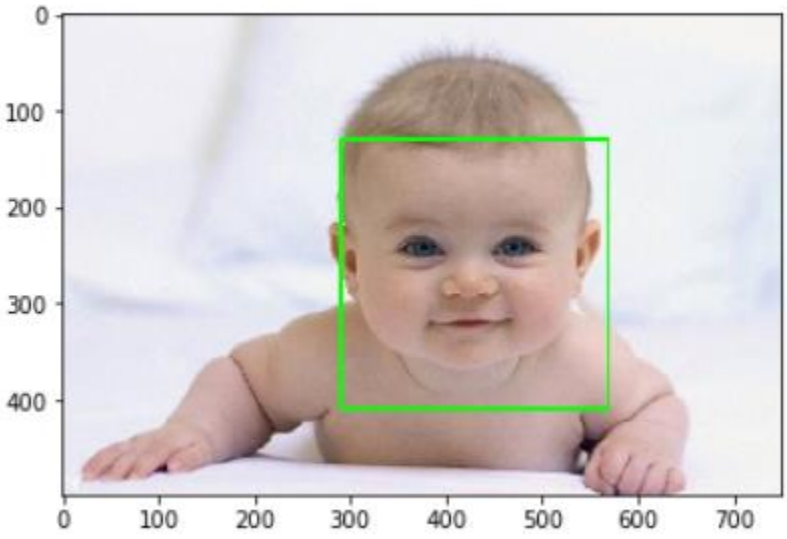
The following is the face detection observed for Face_1 and Face_2 images given for testing after training the adaboost model:

FACE 1 :

INPUT:

OUTPUT:

```
<matplotlib.image.AxesImage at 0x11f22bbe0>
```



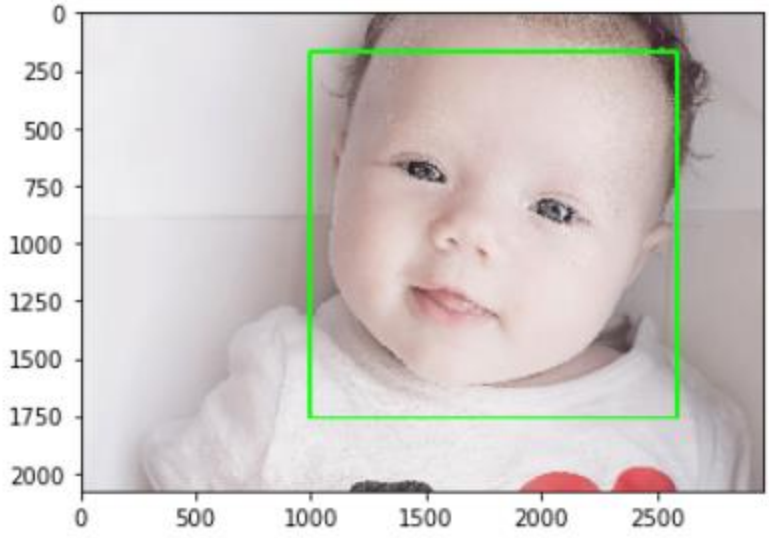
Submitted by: SAI KRISHNA WUPADRASHTA
Unity id: swupadr

FACE 2:

INPUT:

OUTPUT:

```
<matplotlib.image.AxesImage at 0x11f11a160>
```



Submitted by: SAI KRISHNA WUPADRASHTA
Unity id: swupadr

NOTE : The outputs for these testing images- have been inturn cross checked by using the last cross_check_with_built_in_VJ_features.py file- which is the .py file consisting of the Viola Jones algorithm implementation using built-in features. Almost the same output was observed, verifying the correctness of the implementation of adaptive boosting for face detection.

Some other important characteristics like the ROC plots, the histogram plots and the accuracy plots can be visualized by running the main.py file- as there are too many test cases to enlist all the o/p images here.

FINAL ACCURACY OBTAINED : Thus the final accuracy obtained for adaptive boosting for face detection using Haar Filters by implementing in the above mentioned fashion yielded a final accuracy of **92.5%**