ECEN 649 COURSE PROJECT

IMPLEMENTING VIOLA-JONES ALGORITHM

DILEEP KUMAR GUNDA

UIN: 627008899

PROJECT STATEMENT

Implementing Viola-Jones Algorithm for face detection.

INTRODUCTION

Viola-Jones is one of the most famous algorithms that is based on the original Ada-Boost thought. It has a great success in face detection before Deep Neural Network becomes the most popular one, and it is still used in all kinds of digital cameras.

DATASET

This project uses CMU Dataset and statistics of faces and non-faces in the dataset are given below.

	Train	Test
Faces	499	472
Non-faces	2000	2000

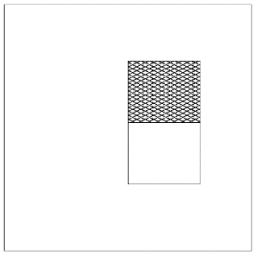
IMPLEMENTATION

The project is segregated into three sections and an additional bonus section. The first two sections discusses the implementation of Algorithm in steps and third section presents the modification to the algorithm to produce in lower False Positives or False Negatives depending on the use case of the application. Forth, Bonus section presents the implementation of advanced Cascading technique that not only reduces the detection time of algorithm but also accuracies.

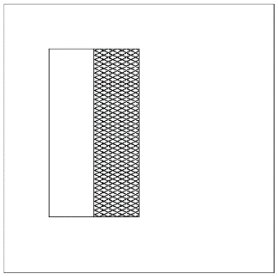
EXTRACT HAAR FEATURES

HAAR features are filters of different shapes and sized applied on the image. These filters are categorized into 5 types as mentioned below,

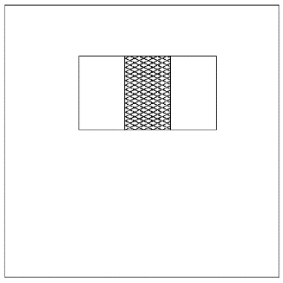
- Two Vertical – Two rectangles stacked on each other



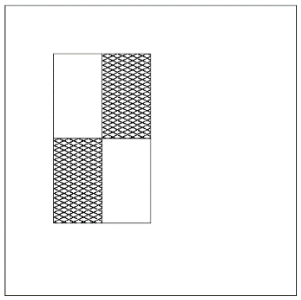
- Two Horizontal – Two rectangles next to each other



- Three Vertical – Three rectangles stacked on each other
  Similar to below, except image is rotated by 90.
- Three Horizontal – Three rectangles next to each other



- Four – Four rectangles



Logic to extract HAAR features is take a point (x,y). This is the top left corner of the rectangle section and begin iterating on width, height of each section. Here section refers to number of rectangular sections in each type of filter, like Two Vertical has two sections. All sections in a given filter are of equal dimensions. So given a point (x,y) and (width, height) one can easily construct a filter. Sample logic to construct 'Two Vertical' filter is given below. 'coords_x' and 'coords_y' store the coordinated of rectangular sections in the order top-left, top-right, bottom-left, bottom-right. As 'Two Vertical' filter has two such rectangular sections, there are total of 8 coordinates. Logic for all filters are mentioned in 'Feature definitions' under section 2.1 in jupyter notebook.

```
'''
2 rectangular sections, arranged vertically
A |+1
--
B |-1
value = A - B
'''
def feature_2v(x, y, width, height):
    coords_x = [x, x+width, x, x+width,
                x, x+width, x, x+width]
    coords_y = [y, y, y+height, y+height,
                y+height, y+height, y+2*height, y+2*height]
    coeffs = [1, -1, -1, 1,
              -1, 1, 1, -1]
    return Feature('Two Vertical', x, y, width, 2*height, coords_x, coords_y, coeffs)
```

Images in the train and test are of size 19x19, so exhaustively iterating over total grid (x,y) -> [0:19,0:19] and (width, height) -> [1:19,1:19] generates 63,960 features as tabulated below.

HAAR Filter type	Count
Two Vertical	**17,100**
Two Horizontal	**17,100**
Three Vertical	**10,830**
Three Horizontal	**10,830**
Four	**8,100**
TOTAL	**63,960**

BUILD YOUR ADABOOST DETECTOR

Adaboost algorithm generally selects an weak classifier for each round and aggregates the opinions from each of these weak classifier and produces final predictions. In this scenario each feature is a weak classifier. We have around 64K features for each sample and 2.5K training samples. Basically feature value is the given below and for a given sample it remains the same irrespective of the round of the Adaboost. To prevent duplicate computations in each round, features values are precomputed for both trainset and testset in the subsections 'Prepare trainset and Prepare testset' under section 2.2 in code. Feature values are efficiently calculated using Integral matrix technique. Code for Integral matrix technique is written in subsection 'Integral Matrix' under section 2.1

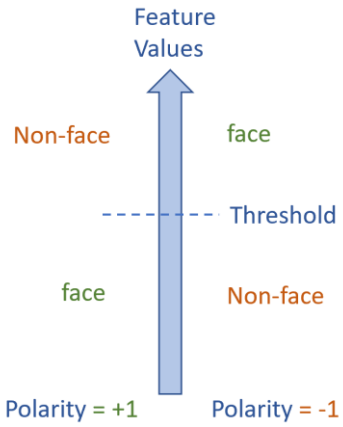
Feature value = Sum of pixels in gray rectangle section – Sum of pixels in white section

'Train' subsection of the code contains main train function that trains the detector taking in argument mainly train data, number of rounds to run. Train data is of size 2499x63960. Row corresponds to image samples and column corresponds to Haar features. There is an optional parameter skip_chance, which is probability to skip a feature. For example, this can be set to 0.25 to skip 25% of the features. This would make the algorithm run faster for testing purposes.

At every round 'train' method first finds thresholds and polarity for all the features and then picks the best classifier that gives least empirical error. This best classifier is the selected classifier of the round. After selecting the classifier at the end of the round the weights are updated according to the Adaboost algorithm. The algorithm increases the weights for misclassified samples and rest of the weights remains the same and later the weights are normalized. The classifier selected at each around are called weak classifiers and these are returned at the of T number of rounds.

Threshold & Polarity:

Each Feature value is a definite integer value and all feature value for all images spans a range. Threshold is fixing a value in this range as boundary for face and non-face. This implies that the feature with value less than threshold is classified as face and value more than this as non-face and viceversa. Below image depicts it.

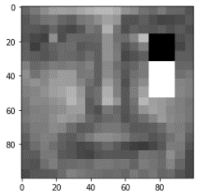


RESULTS

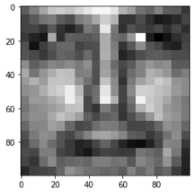
Adaboost detector for 1 round:

Top Feature (Only feature in this case):

		Train	Test
Type: **Two Vertical**			
Position: (14,3)	Accuracy:	**0.807**	**0.808**
Width: 2	False Positive Rate:	**0.181**	**0.096**
Height: 6		**0.017**	**0.095**
Threshold: -182	False Negative Rate:		
Polarity: 1			



Test Image with Filter

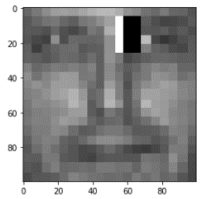


Original Test Image

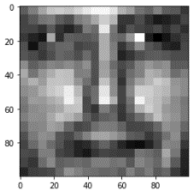
Adaboost detector for 3 rounds:

Top Feature :

		Train	Test
Type: **Two Horizontal**			
Position: (10,1)	Accuracy:	**0.875**	**0.842**
Width: 2	False Positive Rate:	**0.105**	**0.038**
Height: 3		**0.018**	**0.119**
Threshold: -45	False Negative Rate:		
Polarity: 1			



Test Image with Filter

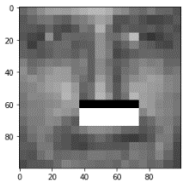


Original Test Image

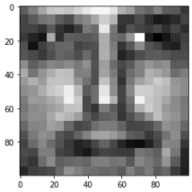
Adaboost detector for 5 rounds:

Top Feature :

Type: **Two Vertical**		Train	Test
Position: (7,11)	Accuracy:	**0.908**	**0.841**
Width: 6	False Positive	**0.076**	**0.016**
Height: 2	Rate:	**0.014**	**0.142**
Threshold: -48	False Negative		
Polarity: 1	Rate:		



Test Image with Filter

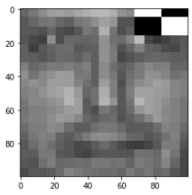


Original Test Image

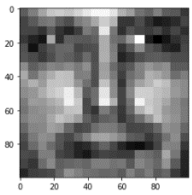
Adaboost detector for 10 rounds:

Top Feature (Only feature in this case):

Type: **Four**		Train	Test
Position: (13,0)	Accuracy:	**0.936**	**0.837**
Width: 6	False Positive	**0.051**	**0.014**
Height: 2	Rate:	**0.012**	**0.147**
Threshold: -53	False Negative		
Polarity: 1	Rate:		



Test Image with Filter



Original Test Image

It is observed that **train accuracy is consistently increasing by increasing T** and running Adaboost detector for more rounds and test accuracy increased comparatively with rounds but it is not consistent. The increase in accuracy can be attributed to detection of various prominent features. From the above results its clear that single round detector detected eye shade, and 3-round detector detected the bridge between the nose and right eye in addition to the eye shade. Each weak classifier detects more prominent features because as the the rounds increases, weak classifiers increases thereby increasing the prominent features detected by the classifier. As the prominent features increases, the detector get robust and detects the face with greater accuracy and it will even classify few false negatives correctly. This explains the decrease in the **consistent decrease in the False Negative Rate**.

ADJUST THE THRESHOLD

Till now Adaboost algorithm picks the best weak classifier and decides on classifier thresholds and polarity such that the empirical error is minimized. There can be use cases the tolerance for false positives and false negative change. For example security systems are more sensitive to false postitives i.e. they want to detect every face and wouldn't want to miss out on anyone. These systems has less tolerance for false positives and are flexible on false negatives. Next report discusses how to modify algorithm to make it less tolerant to False positive and False Negatives respectively.

Instead of selecting best weak classifiers at each round and, threshold and polarity for each feature, algorithm is modified to minimize error E, given as below.

E = lambda*FP + (1-lambda)*FN
FP is Sum of weights of all false positives and,
FN is Sum of weights of all false negatives

Higher values of lambda produces the algorithm more sensitive to false positives i.e. error is proportionately more when there are greater false positives than false negatives. Lower values of lambda would make the algorithm more sensitive to false negatives as it penalized when classifier detects more false negatives.

I ran an experiment for two values of lambda, 0.8 and 0.2. First value produced lower false positive rate and later value of lambda produced lower false negative rate than regular adaboost which minimizes empirical error.

RESULTS

Results on Train data for 5 round Adaboost:

Criterion	Total Accuracy	False Positives	False Negatives
Empirical Error	**90.8%**	**7.6%**	**1.4%**
False Positive	**92.11%**	**3.2%**	**4.6%**
False Negative	**86.95%**	**12.1%**	**0.9%**

Results on Test data for 5 round Adaboost:

Criterion	Total Accuracy	False Positives	False Negatives
Empirical Error	**84.1**	**1.6%**	**14.2%**
False Positive	**81.8%**	**0.7%**	**17.4%**
False Negative	**85.5%**	**2.8%**	**11.6%**

It is observed that False Positive Criterion produces lowest number of false positives and False Negative Criterion produces lowest number of false negatives on both train and test respectively. This is attributed because best classifier that minimized the respective criteria is selected at each round.

ADDITIONAL PART: BUILD THE CASCADE SYSTEM

Cascade system is a series of classifiers connected to each other such that input of next classifier is filtered based on the prediction labels of current classifier. All images which are correctly predicted as non-faces by the current classifier in the true set of non-faces are filtered out from the input before passing on to the next classifier. The final cascade detector classifies image as face if all the classifiers detects as the face and marks as non-face if one detects as non-face and it not passed to next classifiers for classifying. This technique brings in few advantages that are inferred from the results section below.

RESULTS

The below results are observed by running Cascade system for 30 rounds in sets of [5,5,10,10] and Regular Adaboost for 30 rounds.

Results on Train data:

	Cascade System	Regular Adaboost
Time to Train	**~17 min**	**~48 min**
Accuracy	**94.8%**	**93.8%**
False Positives	**0.0%**	**5.0%**
False Negatives	**5.1%**	**1.1%**

Number of Images discarded in Cascade System after each detector:

	Images Discarded
Set 1, T=5	**1794**
Set 2, T=5	**170**
Set 3, T=10	**35**
Set 4, T=10	**1**

Results suggests that Cascade system is 3 times faster than regular Adaboost and prodcues better results with greater accuracies. Fastness of the algorithm can be attributed to the fact that large number of non-faces are discarded in after each detector, thus reducing the input for next detector and increasing the speed of the algorithm. Removal of images classified as non-face explains the low false positives in Cascade System, because the system is left out with all face images. This accounts for increased accuracy too.

Code

The code can be found in Github repo, https://github.com/dileep-g/Face-Detection.

References

- https://medium.com/datadriveninvestor/understanding-and-implementing-the-viola-jones-image-classification-algorithm-85621f7fe20b
- https://medium.com/datadriveninvestor/understanding-and-implementing-viola-jones-part-two-97ae164ee60f
- https://medium.com/swlh/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999
- https://github.com/sunsided/viola-jones-adaboost/blob/master/viola-jones.ipynb