Study of Viola-Jones Algorithm for Face Detection

Niti Jain
nj305@tamu.edu
UIN: 727006290

Abstract

This work implements the famous Viola Jones Algorithm from scratch for face detection. Face detection is one of the challenging computer vision problem which involves detection and identification of faces in a given image. Some of the challenges in face detection include illumination, posture, occlusion etc. In this report, the Viola Jones algorithm is mainly divided into three major parts, namely Extracting Haar Features, Finding the best haar Features on training data, and Finally, using Ada-Boost to develop the face detector.

Index Terms—Haar, AdaBoost, threshold, cascade.

Github link - https://github.com/nitijain/Adaboost-Learner

Introduction

Face detection is one of the most challenging topics in Computer Vision. The goal of face detection task is to determine if the given image contains and any faces, and also its identification in the given image. The task is challenging for computers as we need to train the computers to understand how to detect if an image contains a face. Viola Jones Algorithm is the most coveted algorithm in the realm of face detection. Face detection is one of the most challenging topics in Computer Vision. The goal of face detection task is to determine if the given image contains and any faces, and also its identification in the given image. The task is challenging for computers as we need to train the computers to understand how to detect if an image contains a face. Viola Jones Algorithm is the most coveted algorithm in the realm of face detection.

In this report, we are going to cover the three major parts of its implementation, namely extracting Haar Features, finding the best Haar Features on training data, and finally, using Ada-Boost to develop the face detector. For the first part of the algorithm, we are going to introduce the concept of Integral Images that helps to calculate the Haar features quickly. The next part is finding the best haar feature that helps reduce the classification error on the training data and finally using a combination of these weak classifiers to generate a strong classifier. We are also going to discuss the concept of cascading that helps in efficient computation of the Viola Jones algorithm by discarding the non-face regions beforehand doing further computation.

To summarize, this work is contributing by creating a boosting algorithm for face detection. The major parts of the algorithm include extracting the Haar features, finding the best haar features and training the adaboost detection using the extracted haar features. The code is written in Python using object oriented designing principles in Jupyter Notebook and is uploaded on Github for anyone to

implement in real life scenario. Some of the machine learning packages used include are PIL, pickle, glob, sklearn, numpy, test.

Dataset Used

The dataset used is from Carnegie Mellon University (CMU) containing both face and non-face images. The images are uniformly sized. The training set contains 499 faces and 2000 non-faces whereas the test set contains 472 faces and 2000 non-faces. The faces images are the ones that contain faces and the non-faces are the background images that don't have any faces in it. The dataset also provides 15 group images for face detection. The representation of the dataset is summarized in Table 1.

TABLE I
NUMBER OF SAMPLES IN DATASET

# Dataset	Faces	Non-faces
Train Set	500	2000
Test Set	472	2000

Methodology

A. Image integral and extraction of Haar Features

The first step of the Viola-Jones face detection algorithm is converting the input image to integral image. The integral image helps in the fast and efficient computation of the sum of values of a subset of a rectangle. The integral image is computed at location (x,y) by summing of the pixels above and to the left of (x,y). Figure 1 represents a sample image and its corresponding integral image.

1	1	1
1	1	1
1	1	1

Input image

1	2	3
2	4	6
3	6	9

Integral image

Figure 1: Input Image and corresponding integral image

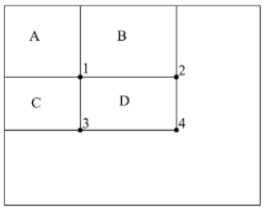

Figure 2: Using Integral images, to compute the area within a rectangle region.

Using these calculated integral images, we can easily find the value in any rectangle. For example, the value in the Figure 2 is basically the sum within D can be computed as (4+1) - (2+3). This is computationally efficient.

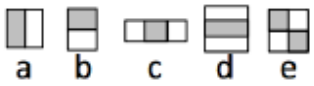

Figure 3: Different type of features

There are basically 5 basic types of Haar-like features as can be seen from the Figure 3.

1. Horizontal feature with two rectangles (Type 2)
2. Vertical feature with two rectangles (Type 1)
3. Vertical feature with three rectangles (Type 4)
4. Horizontal feature with three rectangles (Type 3)
5. Diagonal feature with four rectangles (Type 5)

The total number of haar-like features for 19*19 input image is :-

```
Feature Two Vertical Features with value: 17100
Feature Two Horizontal Features with value: 17100
Feature Three Vertical Features with value: 10830
Feature Three Horizontal Features with value: 10830
Feature Four Diagonal Features with value: 8100
Total Number of Features 63960
```

B. Build Your Adaboost Detector

Adaboost stands for Adaptive boosting. It aims to convert a set of weak classifiers to a strong and focusses on improving the classification error. An AdaBoost classifier is the weighted sum of many weak classifiers. Every weak classifier is a Haar-like feature with a threshold.

$$h(x, f, p, \theta) = 1 \; if \; pf(x) < p\theta$$

$$h(x, f, p, \theta) = 0 \; otherwise$$

where f denotes the feature value, θ is the threshold and p is the polarity indicating the direction of the inequality. The Algorithm used for adaptive boosting is summarized in Figure 4.

- 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, $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
  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 4: Adaptive Boosting Algorithm

On applying the above algorithm, we find that the performance on train data for different number of rounds is summarized in Table 1 and Figure 5

Dataset and Rounds	Accuracy	FPR	FNR
Train t=1	66.184	41.1	4.6
Train t=3	82.31	17.3	19.2
Train t=5	83.55	19.65	3.6
Train t=10	90.03	12.40	0.20

Table 1: Performance for various rounds on train dataset

On applying the above algorithm, we find that the performance on test data for different number of rounds is summarized in Table 2 and Figure 6

Dataset and Rounds	Accuracy	FPR	FNR
Test t=1	45.85	58.27	36.6
Test t=3	61.22	29.83	76.69
Test t=5	58.18	33.73	76.05
Test t=10	71.37	17.64	75.21

Table 2: Performance for various rounds on test dataset

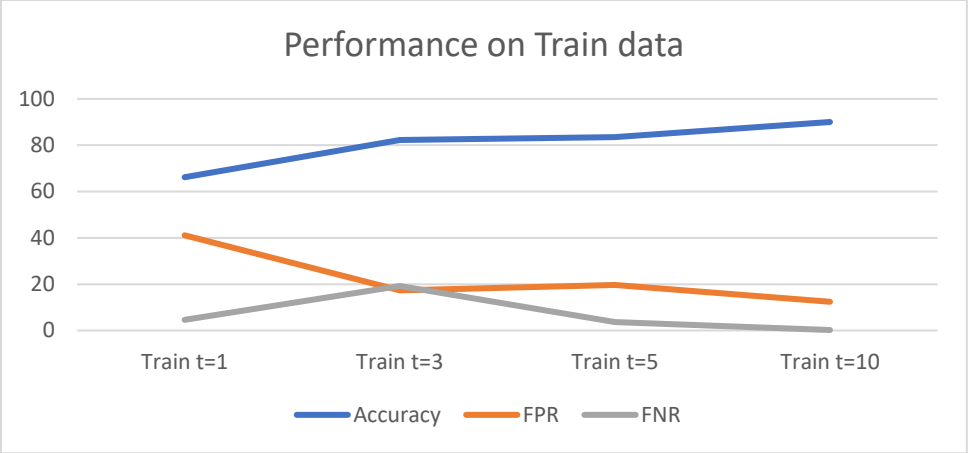

Figure 5: Performance comparison on train data

As we can see from above graph, the performance on the train set is continuously increasing with increasing the number of rounds in terms of Accuracy, FPR and FNR. This suggests our model is learning with an increasing number of rounds. At the end of 10 rounds, the performance on train set is 90.03% in terms of accuracy. We find the accuracy is increasing on the training data and False positive rate and false negative rate is decreasing with increasing number of rounds.

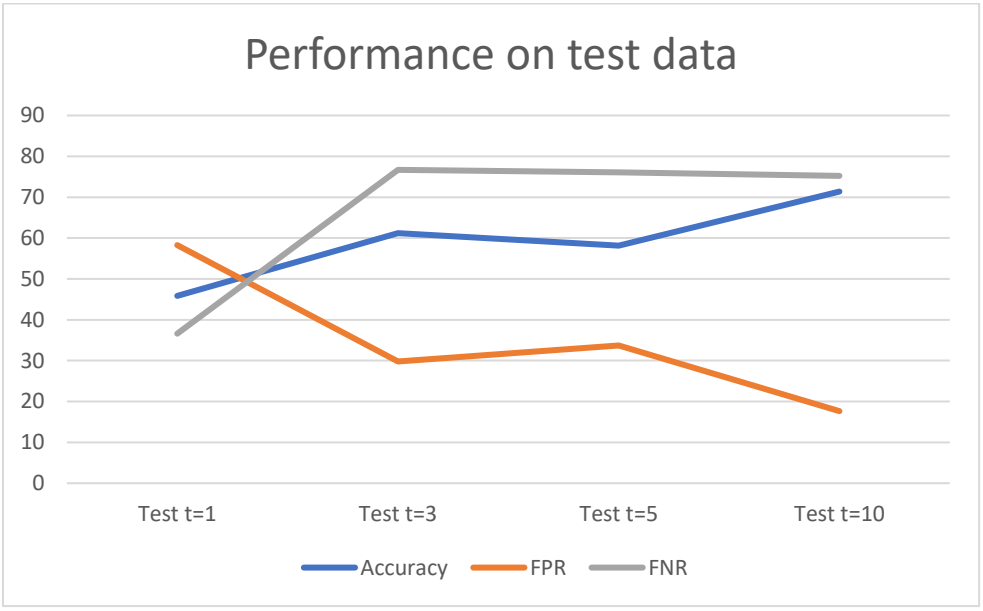

Figure 6: Performance comparison on test data

However, the performance on test set is not increasing with the number of rounds. This could be attributed to the fact that the training data is overfitting the model with increasing number of rounds. This means we have overfitted our model with the training data. The performance on test data is summarized in Figure 6 and Table 2.

Features for different rounds

Round - 1

Feature number 1:
Type: Two Horizontal
Position: (7,6)
Width: 12
Length: 10
Threshold: 0.00025
Training accuracy: 66.18%
The Two Horizontal feature is plotted below in Figure

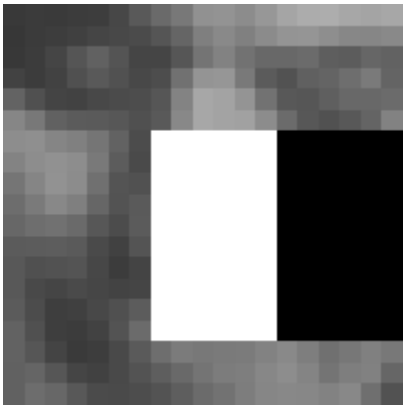

Figure 7: Feature from Round 1 plotted on test image

Round - 3

Type: Three Horizontal
Position: (5,2)
Width: 9
Length: 1
Threshold: 0.0015
Training accuracy: 82.31%

The Three Horizontal Features is plotted below in Figure 8

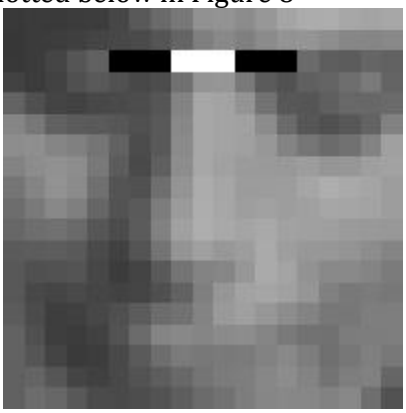

Figure 8: Feature from Round 3 plotted on test image

Round - 5
Type: Two Vertical Features
Position: (16,17)
Width: 1
Length: 2
Threshold: 5.518
Training accuracy: 83.55%

The Two Vertical Features is plotted below in Figure 9

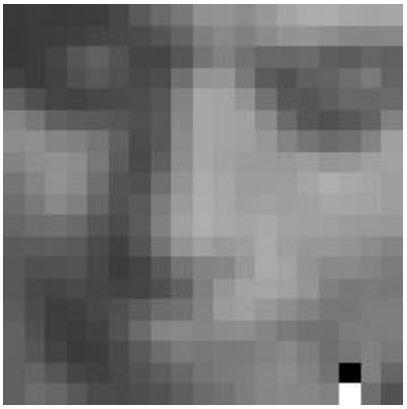

Figure 9: Feature from Round 5 plotted on test image

Round - 10
Type: Four Diagonal
Position: (8,3)
Width: 2
Length: 14
Threshold: 1.019
Training accuracy: 90.03%
The Four Diagonal feature is plotted below in Figure 10

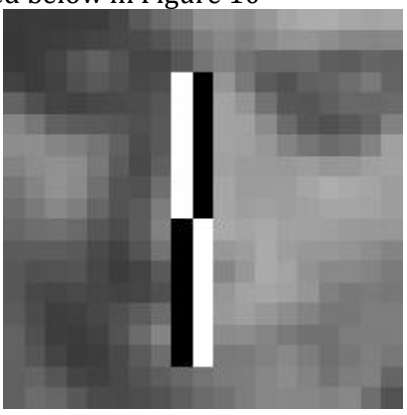

Figure 10: Feature from Round 10 plotted on test image

C. Adjust the threshold

To implement this part, we first calculate the false-positive (negative images classified as positive) and false-negative images (positive images classified as negative)

Following which, we calculate (FPR),

$$False\ Positive\ Rate = \frac{FP}{N} = \frac{FP}{FP + TN}$$

Where FP is the false positive, FN is false negative, and TN is true negative.

Similarly, we calculate False Negative Rate (FNR), which is

$$False\ Negative\ Rate = \frac{FN}{P} = \frac{FN}{TP + FN}$$

The formula used to minimize the error is:

$$Error\ = lambda * FPR + (1 - lambda) * FNR$$

Criterion	Accuracy	FPR	FNR
Train Empirical Error	**83.55**	19.65	3.6
Train False Positive	78.39	**7.5**	78.15
Train False Negative	58.38	52	**0**

Table 3: Performance comparison on Train for t=5

To adjust the threshold to minimize FPR and FNR, I have used the above error formula instead of calculating the empirical error. In my case I have taken the value of lambda =0.2 when I want to minimize FNR and lambda = 0.8 when I want to minimize FPR.

$$Error\ (FNR)\ = 0.2 * FPR + (1 - 0.2) * FNR$$

$$Error\ (FPR)\ = 0.8 * FPR + (1 - 0.8) * FNR$$

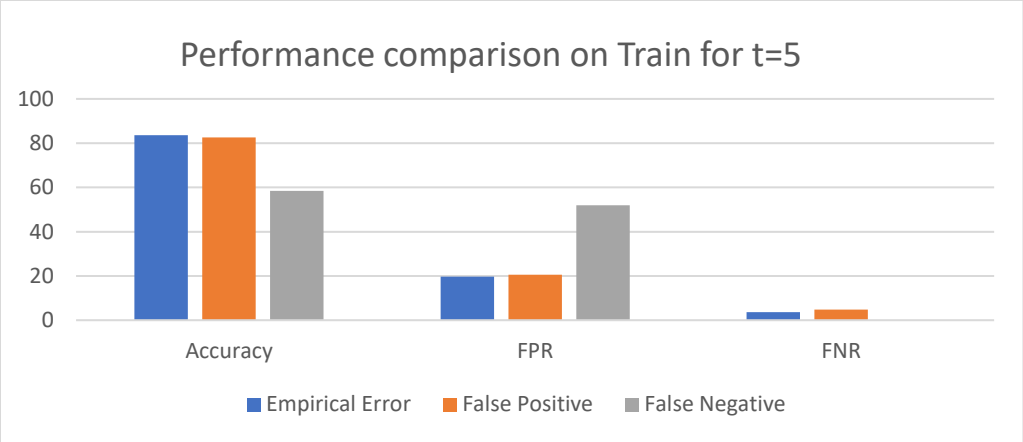

Figure 11: Performance comparison on Train for t=5

On training the model in a way to minimize the error as represented by above equation, the performance on training data can be summarized by Figure 11 and Table 3. As we see from the table, on using empirical error as the metric, the accuracy is the maximum and on using False positive as the metric, the corresponding training FPR is the lowest (7.5). Similarly, for False negative metric, its corresponding FNR is the least (0.0) compared to the empirical error and False positive metric. This suggests that when we use a metric, we are trying to minimize that particular metric and the other metrics get skewed.

Criterion	Accuracy	FPR	FNR
Test Empirical Error	58.18	33.73	76.05
Test False Positive	**82.85**	**0.4**	87.71
Test False Negative	45.04	61.91	**25.42**

Table 4: Performance comparison on test data for t=5

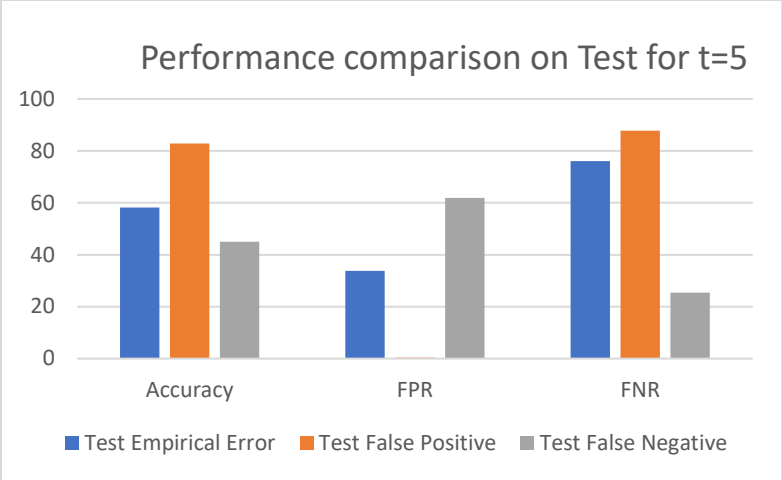

Figure 12: Performance comparison on test for t=5

The performance on test data is summarized in Figure 12 and Table 4. As we see from the table, on using False positive as the metric, the corresponding training FPR is the lowest (0.4) on test data. Similarly, for False negative metric, its corresponding FNR is the least (25.42) compared to the empirical error and False positive metric on the test data. This suggests that when we use a metric, we are trying to minimize that particular metric and the other metrics get skewed.

D. Bonus Point - Detecting in real-world photos:

To perform face detection on group images, I have followed the following steps:

1. **Preprocessing the image** – This includes converting image to the size (384, 288) and then converting the image to grayscale so that we can next extract the features.

2. **Feature extraction** – After the image is converted to grayscale, the integral image is calculated, and features (5 type of features) are extracted as in part A.

3. **Building Detector** – The detector is a cascade of multiple strong classifiers (in this case rounds 1,3,5,10,15,3025,50) developed in part B. On test time, the features extracted from every window is sent to detector to detect if there is a face or not.

4. **Window detection** – Every possible window of the resized image is then passed to the detector. The detector detects if the region has a face and if there is a face, the position of the window is stored.

5. **Box construction** – After the positions of the windows are identified in step 4, the next step is to construct a rectangle around the face. In the image attached, I have constructed the boxes white in color.

Figure 13 represents face detection on the test files. As we can see from the images, the aim was to remove the false positives.

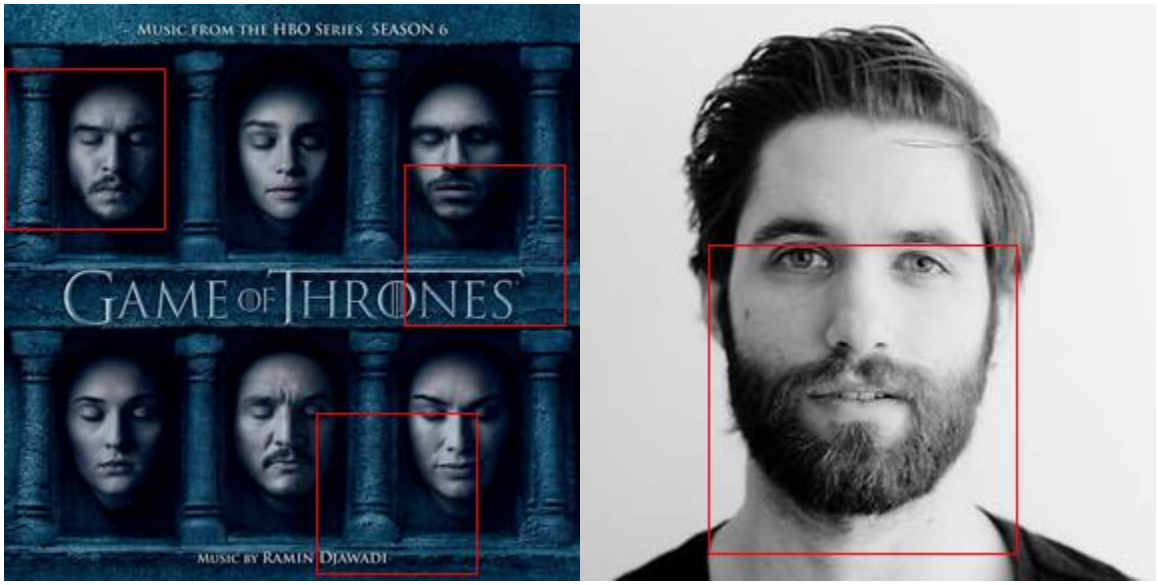

Figure 13: Face detection on test files