SMAI Final Project Report

four_20

Siddik Ayyappa

P Balaramakrishna Varma

Miryala Narayana Reddy

Haasa Garikapati

Statistical Methods in AI

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

December 7, 2022

1 Introduction

The Project includes the implementation of the Viola Jones Algortithm.In the domain of face detection the Viola Jones system yields detection rates comparable to the best previous systems. Viola Jones Algorithm is a very famous algorithm, for object detection at an extremely rapid rate. This work is distinguished by three key contributions.

a) Integral Image

b) Adaboost

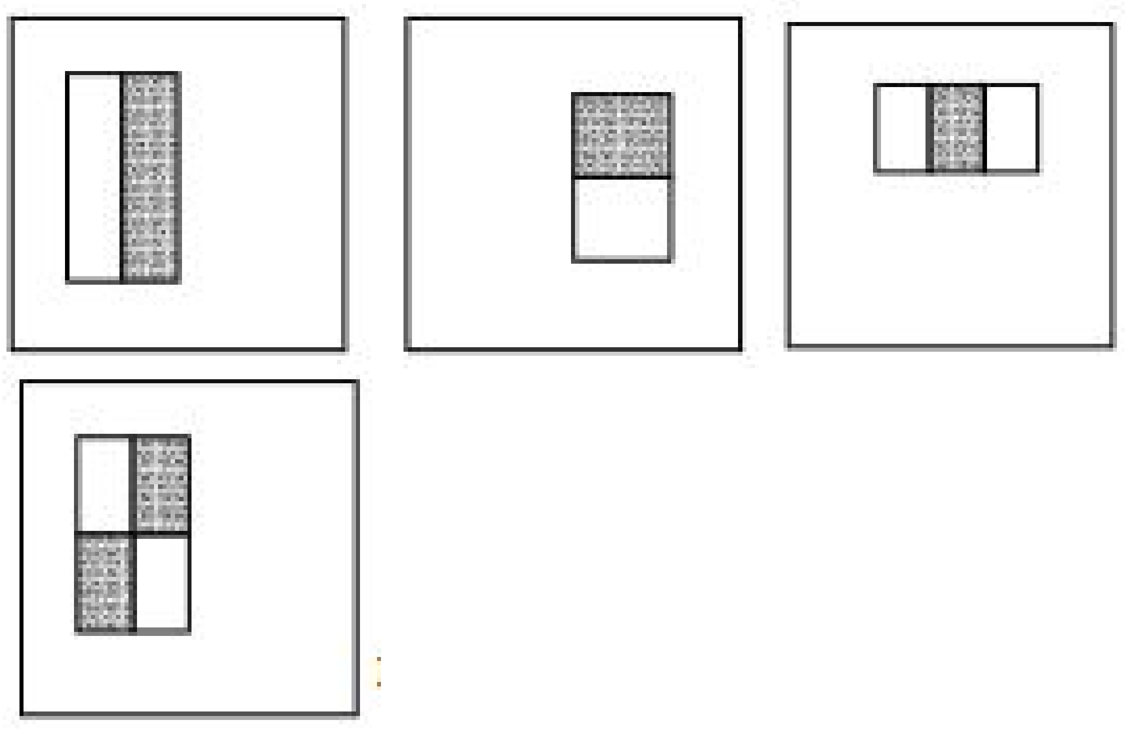
c) Cascade

2 Feature Extraction

2.1 Features used

The motivation of using feature extraction is that it encodes raw ad-hoc pixel information which is not easy to learn while training. Also it has been observed that feature based networks work faster.
We have used following Rectangle Feature features.

- **Two Rectangle Features:** Difference between sum of pixels within two rectangle regions of same shape and size and are adjacent either horizontally or vertically.
- **Three rectangle feature**: Sum of pixels of two outside rectangles is subtracted from centre rectangle's sum of pixels.
- **Four rectangle feature**: Difference between the diagonal pairs



2.2 Integral Image

Since the feature extraction for a Image takes so long and computationally heavy, this paper came up with new concept called Integral Image which computes the above features very efficiently with less number of array accesses. **Integral** is an intermediate representation of image that helps us in calculating the rectangle features very rapidly and efficiently.
Integral Image at x, y is defined as sum of x', y' for all $x' \leq x$ and $y' \leq y$

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

the following recurrence relation can be used to compute the integral image in one pass

$$s(x,y) = s(x, y-1) + i(x,y)$$

$$ii(x,y) = ii(x-1, y) + s(x,y)$$

with initial conditions $s(x,-1) = 0$ and $ii(-1,y) = 0$

2.3 Rectangular sum from integral images

We use this Integral Image to compute our Rectangular features using the following method. In the adjacent figure 1 we have integral images of all corners in A,B,C, D Rectangles. Sum of all pixels in D (rectangular sum) is

$$ii(x,y) + ii(x-1,y-1) - (ii(x-1,y) + ii(x,y-1))$$

which requires only 4 array references. So compute sum of pixels in a rectangle we need 4 references from integral image. Similarly, we see that Number of array references for

- Two rectangle feature is 6
- Three rectangle feature is 8
- Four rectangle feature is 9

This is used in computation of Rectangle features.

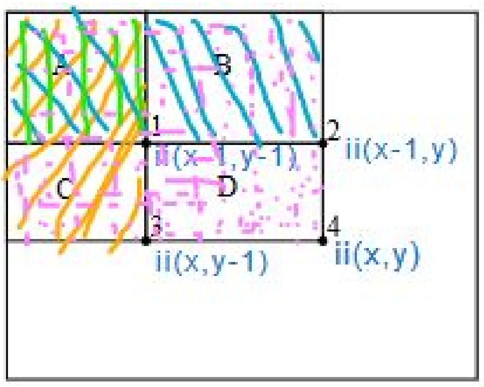


Figure 1: sum of pixels in rectangle D

2.4 Optimization

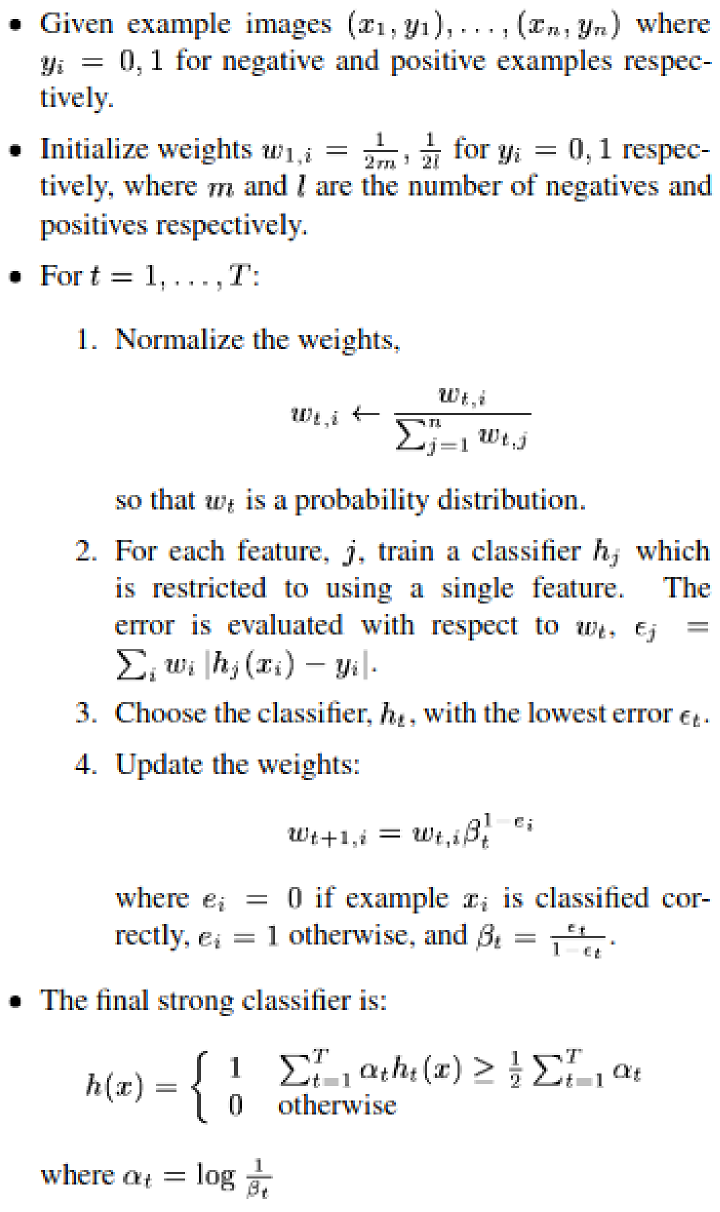
Even computation of Integral Image features is taking long time. So we tried to optimize using multiprocessing on multiple CPUs and got $50\% - 60\%$ reduction in the time taken for computation on lower number of images. But when we use large number of images the reduction seems quite significant as it takes advantage of multiprocessing.

3 Classifier

Now that we have obtained the features and are provided with labels, we shall do classification with them. There are various machine learning techniques to do classification.The focus of the paper has mostly been on Adaboost Technique. As provided in the paper, we know that there have been over 180K features, associated with each sub-window of the image. Even if the features have a low computation time, computing all of the 180K features and using every one of them for classification is a computationally demanding task. Hence, we decide to look for the best features, to classify on and then make an effective classifier out of them.

In order to find the best features, we use a weak learning algorithm, which classifies on the basis of a single feature. (Weak classifiers, with Adaboost).And then we select the best features in each round for multiple rounds. The final classifier uses these (best) features. The pseudocode of the training process is given in the next slide. The final (strong) classifier is an ensemble of many weak classifiers.

3.1 Pseudo code:



```
• Given example images $(x_1, y_1), \dots, (x_n, y_n)$ 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 $t = 1, \dots, T$:
    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 $\epsilon_t$.
    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 $\alpha_t = \log \frac{1}{\beta_t}$
```

3.2 Results:

The experiments in the paper have shown that using 200 features out of all the 180K+ features yields a 95% detection rate. (False Positive - 1/14084). Which is a good performance in 0.7 seconds.

For better performance, we could try including more features in the classifier. However, this would increase the computation time. The Training time, would be directly increased, since we have to do multiple rounds of boosting depending upon the number of features we would like to include. The testing time, would be directly increased, since the image would be scanned by multiple classifiers. Even if the performance numbers are impressive, they are not impressive enough for real-time tasks.

3.3 Interpretation

The features obtained by the AdaBoost Algorithm are interpretable and meaningful. They focus on various regions of the face, like the nose bridge, eyes-nose-cheek regions etc. In the experiments conducted, it was seen that the first feature selected by AdaBoost focuses on the eye-nose-cheek regions. The second feature focuses on the nose-bridge regions.

4 Cascading classifier

The key insight to this method is that smaller, and therefore more efficient, boosted classifiers can be constructed which reject many of the negative sub-windows while detecting almost all positive instances. The threshold of a boosted classifier can be adjusted so that the false negative rate is close to zero.

Simpler classifiers are used to reject the majority of subwindows before more complex classifiers are called upon to achieve low false positive rates. A positive result from the first classifier triggers the evaluation of a second classifier, a positive result from the second classifier triggers the evaluation of third classifier and so on until, we pass through all layers in this case we classify the input to have face or we are rejected by some layer i. In this case we classify the input to not contain face.

Each stage in the cascade is constructed by training classifiers using AdaBoost and then adjusting the threshold to minimize false negative. The early cascade stages should be very simple and should eliminate as many negatives as possible while the further stages are more and more complex and take more time for evaluation. The examples which make it through the first stage are "harder" than typical examples and so on.

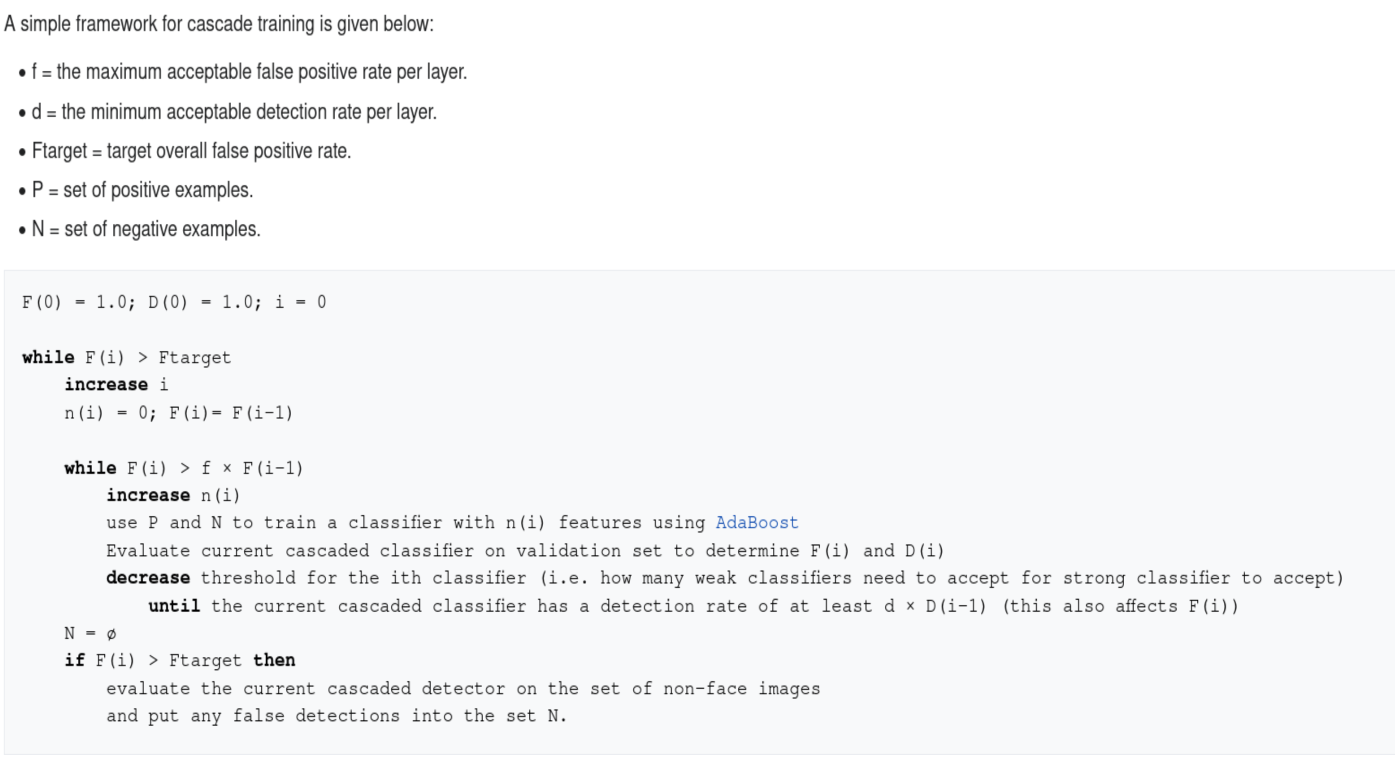
4.1 Cascade training process

cascade training process involves two types of tradeoffs.

- classifiers with more features will achieve higher detection rates and lower false positive rates.
- At the same time classifiers with more features require more time to compute.

The number of classifier stages, the number of features in each stage, and the threshold of each stage, are traded off in order to minimize the expected number of evaluated features. Each stage is trained by adding features using modified Adaboost until the target detection and false positives rates are met. Stages are added until the overall target for false positive and detection rate is met.

4.2 Pseudo code



A simple framework for cascade training is given below:

- f = the maximum acceptable false positive rate per layer.
- d = the minimum acceptable detection rate per layer.
- Ftarget = target overall false positive rate.
- P = set of positive examples.
- N = set of negative examples.

```
F(0) = 1.0; D(0) = 1.0; i = 0

while F(i) > Ftarget
    increase i
    n(i) = 0; F(i)= F(i-1)

    while F(i) > f × F(i-1)
        increase n(i)
        use P and N to train a classifier with n(i) features using AdaBoost
        Evaluate current cascaded classifier on validation set to determine F(i) and D(i)
        decrease threshold for the ith classifier (i.e. how many weak classifiers need to accept for strong classifier to accept)
            until the current cascaded classifier has a detection rate of at least d × D(i-1) (this also affects F(i))
    N = ∅
    if F(i) > Ftarget then
        evaluate the current cascaded detector on the set of non-face images
        and put any false detections into the set N.
```

5 Analysis

- All the results are gathered for 1000 training samples.
- We have trained a 5 layer cascade.
- The adaboost classifier performs similar to the results mentioned in the paper.

5.1 Feature Analysis

5.1.1 No of feature vectors vs size of the image

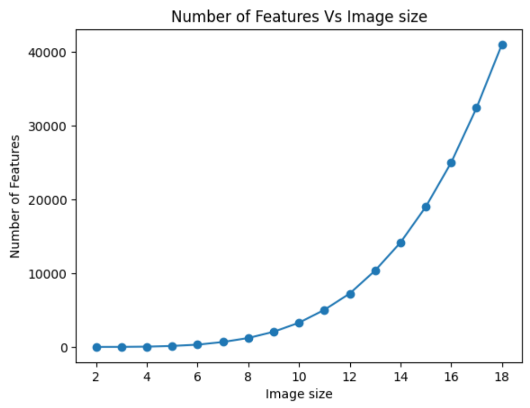


Figure 2: 5.1.1

The number of feature vectors have increased with the size of image. Thus we need to compute large number of image features for large size images as you can see 40,000 image features are computed for an image of size 19x19 itself. We plotted this graph to see how number of feature vectors increase with size of image. The smooth curve we got also verifies that there is no error in the feature vector calculation.

5.1.2 Time taken to compute feature vectors vs size of the image.

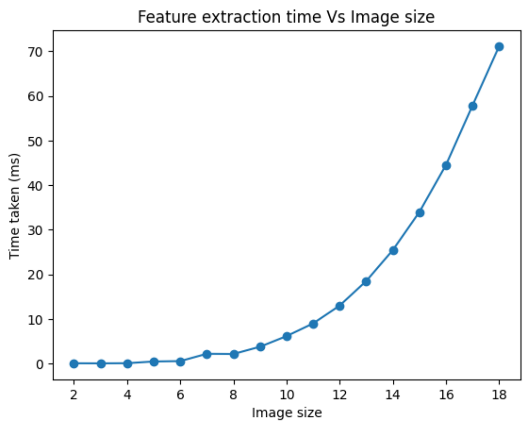


Figure 3: 5.1.2

This graph helps us to estimate how long it would take for feature extraction. Also we can see the similarity of graphs 5.1.1 and 5.1.2 and say that the time taken is proportional to number of features. Time taken to compute feature vectors increased with the image size because number of feature vectors increase with size of image and hence the time to to compute the then increased.

5.1.3 Time to compute i^{th} fixed vector for a fixed size of image.

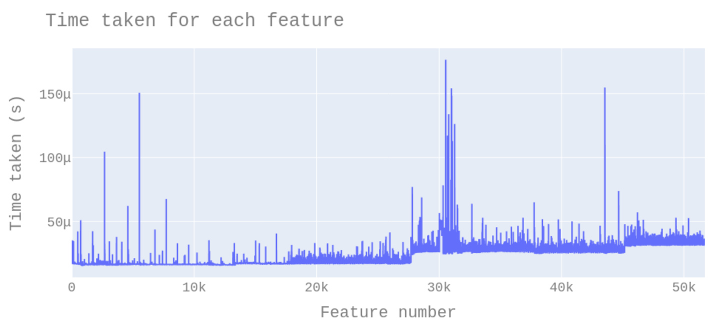


Figure 4: 5.1.3

This plot helped us to see which feature vectors are taking long time and which features are taking less time.From the plot we can see that most of the feature vectors have taken less than 50 μ seconds. If we avoid computation of the features which take long time and compute only feature vectors which take less time, we think it will give reasonably good results but we leave this experiment for future analysis.

5.1.4 Time to compute i^{th} feature vs size of the image.

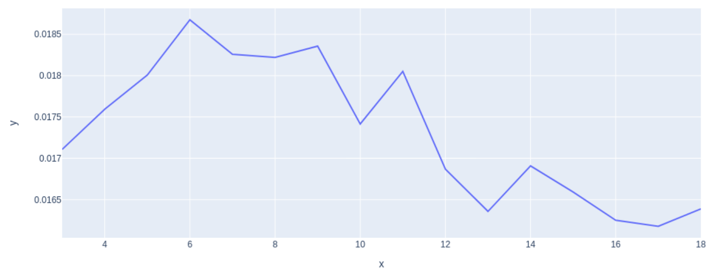


Figure 5: 5.1.4

Theoretically we may think that the ith feature should take same time but it doesn't practically as the following graph shows. The time taken for ith feature to be computed decreased as the size of image increased. The main reason for getting such a plot is the ith feature vector is different for each image size. To explain clearly, the smaller image sizes had bigger rectangular feature as ith feature vector and as size increased the ith feature vector sie decreased because more and more smaller feature vectors came up with image size. This caused the size of ith feature vector to go small and hence the time taken for the ith feature vector computation. If there there is any better explanation for this one I would sue welcome it.

5.2 Adaboost Analysis

5.2.1 Accuracy vs number of features used in the adaboost classifier.

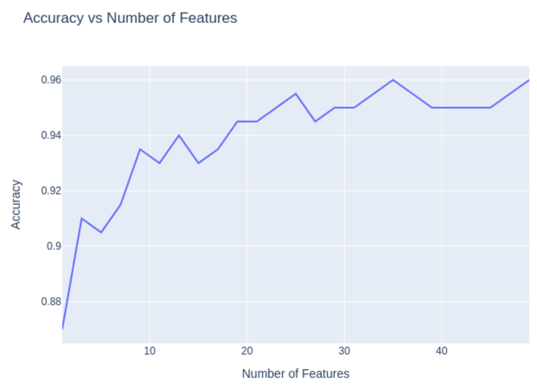


Figure 6: 5.2.1

Adaboost classifier uses decision trees of depth max 1 for each feature. Using more decision trees will give us more accuracy. It's like the coin tossing.. we toss coin more times we get closer and closer to probability 0.5 ...i.e., the accuracy. Hence the accuracy of Adaboost classifier increased with increase of number of features

5.2.2 Positve Detection rate vs number of features used in the adaboost classifier.

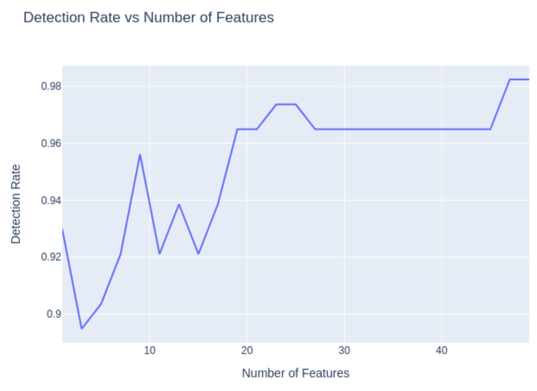


Figure 7: 5.2.2

The detection rate has to increase as the number of features are increased. The detection rate is around 98% when number of features 50 . This shows the percentage of correctly dectected faces with number of features.

5.2.3 False Positive rate vs number of features used in the adaboost classifier.

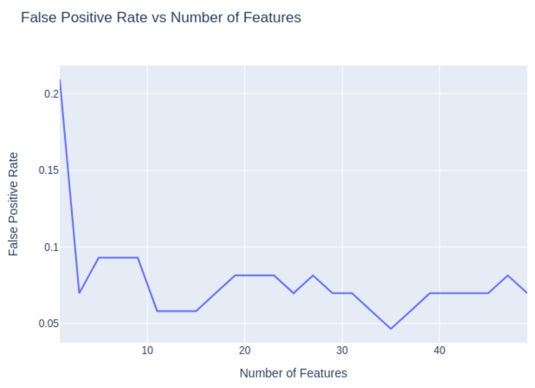


Figure 8: 5.2.3

We have some classifications where something is falsely detected as face. such false classifications have reduced as number of features are increased in the adaboost classifier as expected.

5.2.4 False Negative rate vs number of features used in the adaboost classifier

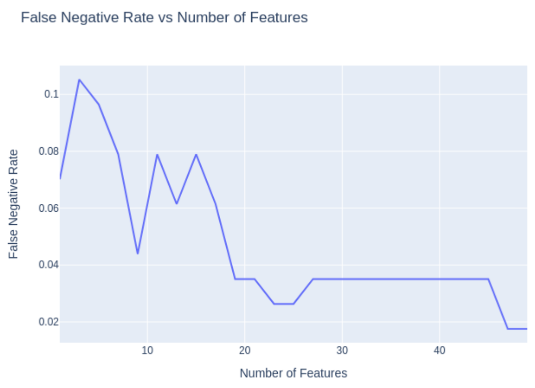


Figure 9: 5.2.4

These false Negatives too decrease just like false positives because more number of features in adaboost classifier decrease these false classifications using many weak classifiers (decision trees of depth 1)

5.2.5 Time taken to classify subwindows vs number of features used in adaboost classifier.

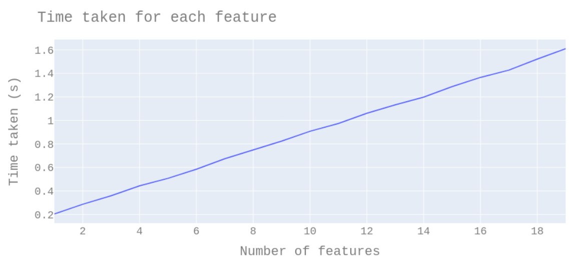


Figure 10: 5.2.5

The adaboost classifier classifies better with more number of features but the time taken to process each feature i.e., taking output of each decision tree and calculating the final result depends on number of features and is proportionally related. Hence the graph is linearly increasing with the number of features in the adaboost classifier.

5.3 cascade Analysis

5.3.1 number of training samples for each layer.

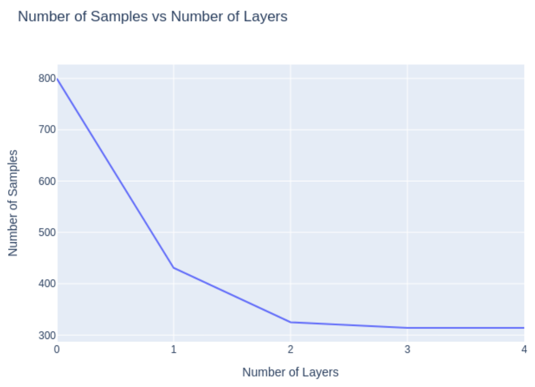


Figure 11: 5.3.1

As we go through the cascade layers many false face feature vectors are eliminated and only smaller number of features are passed to next adaboost cascade layer as input. This pattern is expected by the paper and the actual logic of the cascading.

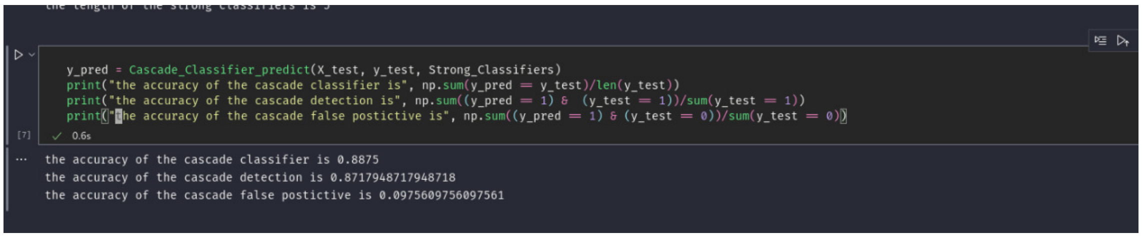
5.3.2 composition of training samples for each layer.

Layer Number	Number of Positive Samples	Number of Negative Samples
0	431	369
1	325	106
2	314	11
3	314	0
4	314	0

According to cascade logic the first adaboost classifier ouputs many negatives and pass out positive samples containing less number of negatives. These negatives are further filtered out by the next cascade layer which is a little more complex compard to first one. so it can filter out mostly all the majaor negative samples. The left over negatives are further eliminated in the successive layers .

This table shows the number of samples and negative samples at each layer. The negative samples are eliminated mostly in the first two layers and all by the time we reach last layer as expected.

5.3.3 Overall best accuracy results



```
y_pred = Cascade_Classifier_predict(X_test, y_test, Strong_Classifiers)
print("the accuracy of the cascade classifier is", np.sum(y_pred == y_test)/len(y_test))
print("the accuracy of the cascade detection is", np.sum((y_pred == 1) &  (y_test == 1))/sum(y_test == 1))
print("the accuracy of the cascade false postictive is", np.sum((y_pred == 1) & (y_test == 0))/sum(y_test == 0))
```

```
the accuracy of the cascade classifier is 0.8875
the accuracy of the cascade detection is 0.8717948717948718
the accuracy of the cascade false postictive is 0.0975609756097561
```

Figure 12: 5.3.3

We have achieved almost the same accuracy as the Viola-Jones paper did. our accuracy for cascade classifier is 88.75% for 87 false detections while Viola-Jones paper had 88.4% for 31 false dections and 92.1% for 78 false detections.

5.3.4 Time taken to classify vs no of subwindows to be classified

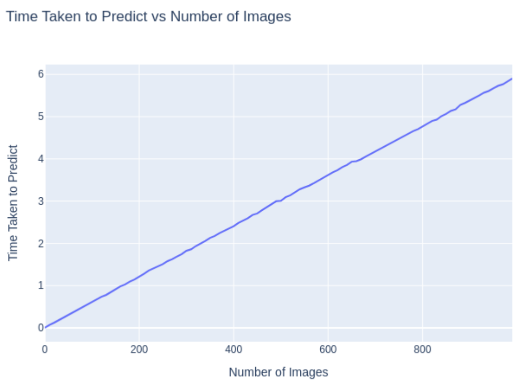


Figure 13: 5.2.7

Adaboost classifier takes constant time to classify each sub-window in the image. If we increase sub windows, time taken will increase linearly as observed in the graph.

6 Comparison of our implementation with that of the authors

6.1 Training Data

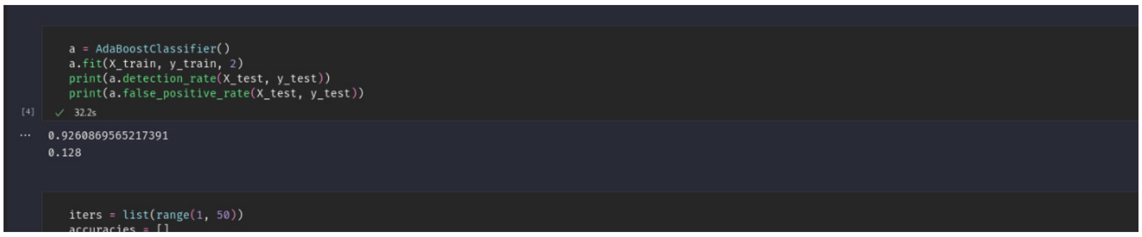
Author's trained their classifiers on 20000 images(10000 face and 10000 non-face) whereas we trained our classifiers on 1000 images(500 face and 500 non-face).

6.2 Feature Extraction

We use 19×19 size subwindows for fitting more samples in our RAM. Currently each sample takes 2MB of space. The authors have use 24×24 subwindow. There are no stats on the time taken for feature extraction given in the paper. We get about 1 micro second on intel core i5 10^{th}.

6.3 Adaboost Classifier

Measured against the testing set our classifier gave the following performance: Whereas the authors got 100%

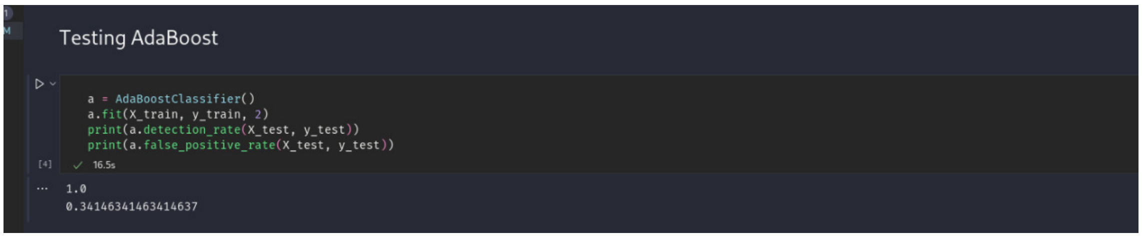


```
a = AdaBoostClassifier()
a.fit(X_train, y_train, 2)
print(a.detection_rate(X_test, y_test))
print(a.false_positive_rate(X_test, y_test))
[4]  ✓ 32.2s
...  0.9260869565217391
     0.128

iters = list(range(1, 50))
```

Figure 14: 6.3.1

detection rate with 40% false positive rate. It should be noted that with high detection rates false positive rates increase and both of these values are traded off b/w each other every time. Our results have lesser detection rates and lesser false positive rate compared to Author's.



Testing AdaBoost

```
a = AdaBoostClassifier()
a.fit(X_train, y_train, 2)
print(a.detection_rate(X_test, y_test))
print(a.false_positive_rate(X_test, y_test))
[4]  ✓ 16.5s
...  1.0
     0.34146341463414637
```

Figure 15: 6.3.2

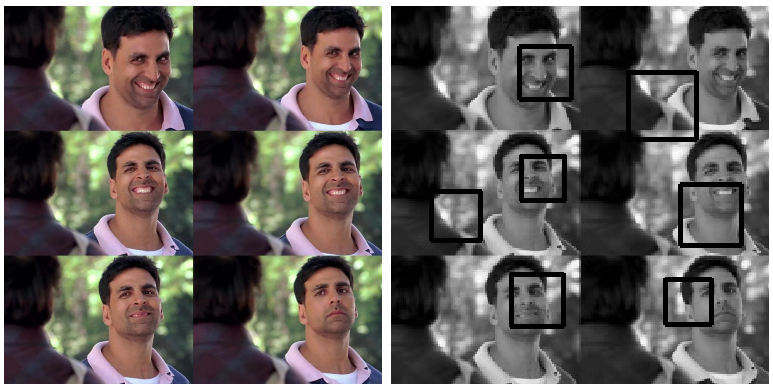
After multiple runs we got something very close to the Author's result.

6.4 Cascade Classifier

Author's Cascade classifier has 38 layers with more than 6000 features whereas our Cascade classifier has 5 layers with around 200 features. As we increase number of features the time taken for computation increases drastically which was not possible to train on our systems.

As we mentioned previously false positive rate decreases with increase in number of features, Since we have less number of features and layers our false positive rates could be way more when compared to the paper results.

6.5 Running the classifier on an Image



6.5.1 Time Wise

Using the same size of the image and other parameters like scale and stride our implementation takes $10sec$ while it takes $0.67sec$ for them. Our system runs $1.2GHz$ while there runs at $740MHz$ so our target is $0.03sec$. Authors have used cpp or c language for implementation as python was not popular back then and c, cpp was to go to language for performance. Python being interpreted language is many times slower than complied low level languages like c or cpp.

6.5.2 Performance

Refer to the *Cascade Classifier* comparison.