Introduction to Face Detection
A implementation which base on AdaBoost

ZIJIAN LIU
jasonleaster.github.io
XiangTan University
April 17, 2016

Abstract

We are going to introduce a method which will help us to do face detection. The idea of this system are arised by viola. In this paper, I also contribute some scheme for optimaization. Some face detection applications are based on nerual network but it may not be friendly to construct a nerual network. The detection system in this paper are useful and simple enough to be implemented. Data dirven application cost a lot of time, I use a concurrent framework to accelerate the training process. The time of extracting features from original images can be shorten obviously. The key point in this system is using AdaBoost to select and combine some weak classifier into a strong classifier. Finally, I demonstrate some test result of the our face detection system.

Keywords: Face Detection, AdaBoost, Haar Feature, Machine Learning

Version 1.2
All Right Reserved

Contents

1 Preparation **3**

2 Integral image **4**

3 Haar Features **5**

4 Weak Classifier **6**

5 AdaBoost **8**

6 Face Detection and optimaization **13**

6.1 Search and detection 13

6.2 Optimalization 14

7 Results **15**

8 Details of Implementation **18**

1 Preparation

All training images get from MIT and CMU library and all program written by Python. So you may have to install Python into your operating system. I just use basic Python library like numpy, matplotlib, os and pylab. We don't need OpenCV. We are doing the same great thing. I use five section to introduce this project. Integrated image, Haar features, AdaBoost and the test module.

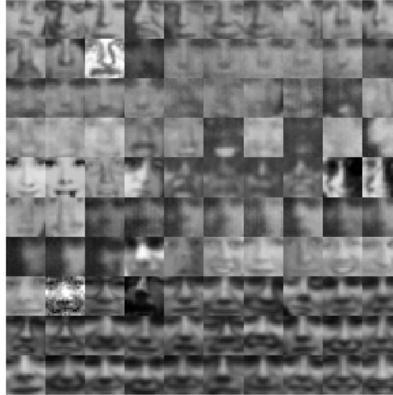


Figure 1: Samples from the training set

The most application of Machine Learning are data driven. So, you may have a good workstation or PC to run the system. Otherwise, it will cost your a lot of time to training the model.

To elimilate the influence of lights from environments, we should pre-process the images and normalize them.

$$I(x,y) = \frac{I^{ori}_{x,y} - \mu}{\sqrt{\frac{1}{M*N} \sum_{i=1}^{N} \sum_{j=1}^{M} (I^{ori}_{i,j} - \mu)^2}} \tag{1}$$

where μ is the mean value if the initial image which's size is M-row x N-col. In our training set, the size of image is 19x19 and all images are black and white. The training set contains about 2429 face images and 4548 non-face images.

The denominator of the equation 1 is the standard deviation of the pixel intensity values. I^{ori} means the original image in the training set. $I(x,y)$ represent the pixel at x-row, y-col in the image which is after processing.

2 Integral image

The first step of the Viola-Jones [**?**, **?**] face detection algorithm is to turn the input image into an integral image. It's easy to do this job something like integration.

$$ii = \iint i(x', y') \tag{2}$$

ii is the integral image and the i represent as the original image. Here is a demonstration in a normal image.

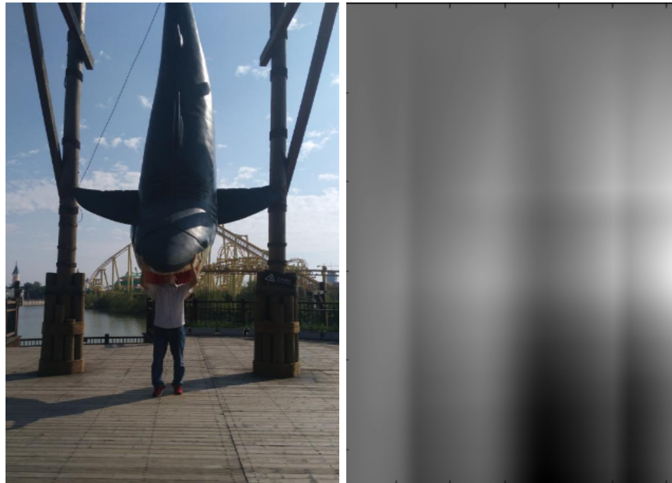


Figure 2: The left side is a normal image in our daily life. The right side is the integral image of the origianl left side image.

The following figure are one of the sample face and it's integral image.

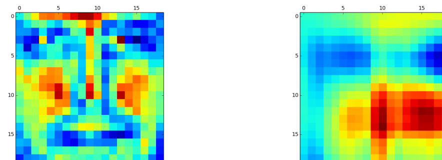


Figure 3: Image on the left is a training sample(face00001.pgm) which is show by Python matplotlib and the right side is the corresponding integral image. Here is a check point if you want to determine whether your implementation is right or wrong.

What does the integral image use for ? This allow for the calculation of the sum of all pixels inside any given rectangel using only for values – the boundary point of that rectangel.

3 Haar Features

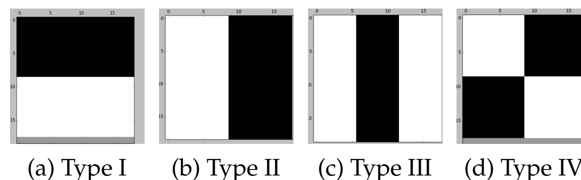
When we zoom the image we can find the image constructed with a lots of block(pixels). They like rectangles with different pattern. That's the essential of how the digital images are constructed. If you have the basical understanding of digital image, you must know that the digital image are constructed with a lots of pixels which's value come from 0 to 255.

The detection system are based on the value of simple features but not using the pixels directly. The reason given by Voila is that "features can act to encode ad-hoc domain knowlege that is difficult to learn using a finite quantity of training data."

The features used in the system are haar-liked features [?], which is similary with haar wavelets. There are other feature like LIF(Local Invariant Feature) [?] . But we didn't use it for the limitation of time.

From my perspective, it's a trick to raise demention of samples.

We used four different pattern in our implementation. They are shown in the folloing figure.



(a) Type I (b) Type II (c) Type III (d) Type IV

Figure 4: Four different types of Haar features.

Viola-Jones use 24*24 detetor window. Instead, I use 19*19 detector. Becasue my the size of my training set images are 19*19. Diffient size of window have different number of features. That's all right.
The value of each features are calculated by sum of pixelsa which lie within the white rectangles are subtracked from the sum of pixels in the grey rectangles.
In my implementation, features are represented as list like this "["II", x, y, w, h]". This means that the type of the selected feature is "II". It start point is (x, y). The width of the rectangle is w and the h is the height of the rectangle.
In Voila's and others work, they used all the features within the detector window, the number of that around 160,000. It's too large and cost a lot of time to compute 16,000 for every subwindow. In my implementation, I optimalized the feature generator which will drop out some feature and make the computation process more faster. I only used about 13,000 features to build our detection system.

4 Weak Classifier

Weak classifier[?] is a type of classifier which can not classify all samples correctly. But they also do good job on classification. A good weak classifier can classify samples with the correct rate over than 50%. Yes, it can be 99% or 51%. It just better than guess randomly.

Algorithm 1 Simple weak classifier

Input: A set of feature responses $\{f_j(x1), ..., f_j(xn)\}$ extracted by appliying the feature f_j to each training sample x_i and associatd labels $\{ y_1, ..., y_n \}$. A set of non-negative weights $\{ w_1, ..., w_n \}$

Output: θ is a threshold value. Attention! p $\in \{-1, +1\}$is a direction value. When the mean value of positive samples smaller than the mean value of negative samples, the direction value p is 1. otherwise, it's -1.

$$g(f_i; p; \theta) = \begin{cases} 1 \text{ if } pf_j(x) < p\theta \\ 0 \text{ if } otherwise \end{cases} \tag{3}$$

e is the error rate of the result of classification by this weak classifier g. **e must be smaller than 0.5**

Steps of algorithm

• Compute the weighted mean of the positive samples and negative samples.

$$\mu_P = \frac{\sum_{i=1}^{n} w_i f_j(x_i) y_i}{\sum_{i=1}^{n} w_i y_i}, \mu_N = \frac{\sum_{i=1}^{n} w_i f_j(x_i) y_i}{\sum_{i=1}^{n} w_i y_i} \tag{4}$$

• Set the threshold to $\theta = \frac{1}{2}(\mu_P + \mu_N)$.

• Compute the error associatd with the two possible values of the direction.

$$\varepsilon_{-1} = \sum_{i=1}^{n} w_i \left| y_i - g(f_i(x_i); -1; \theta) \right| \tag{5}$$

$$\varepsilon_{+1} = \sum_{i=1}^{n} w_i \left| y_i - g(f_i(x_i); +1; \theta) \right| \tag{6}$$

• Set $p^* = \underset{p \in \{-1,+1\}}{\text{argmin}} \ \varepsilon_{p^*}$

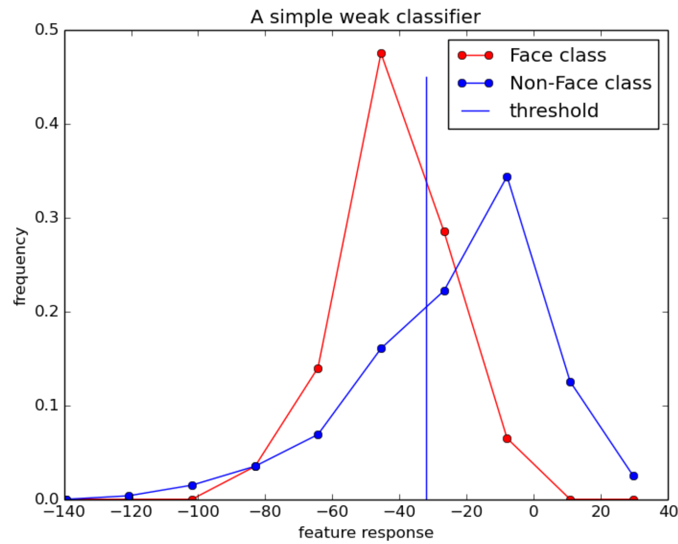


Figure 5: A simple weak classifier. The red curve is the histogram of face class and the blue curve is the histogram of Non-Face class.

The figure above there show what a good weak classifier look like. The more overlap between the two histogram and the more bad the result of classification by any possible threshold. It's important to know that the direction of classification depend on the error rate of two type of sample set.

More detail, I show a pieace of code in the detection system. Reader can understand what I have done more deeply.

```
for direction in [-1, 1]:
    errorRate = 0.
    for i in range(self.SampleNum):
        if self._Mat[d][i] *direction < threshold * direction:
            output[i] = LABEL_POSITIVE # positive label +1
        else:
            output[i] = LABEL_NEGATIVE # positive label -1

        if output[i] != self._Tag[i]:
            errorRate += self.weight[i]

    if errorRate < minErrRate:
        minErrRate     = errorRate
        bestDirection = direction

return minErrRate, threshold, bestDirection
```

5 AdaBoost

What's the next step? We have got the weak classifier. Let's use it to construct a strong classifier by boosting [?].

With window size of 19*19 pixels of the detector, there are huge number of features in the window. Every feature is a classifier. It also means that there are a lots of weak-classifier. There is a problem that which classifier we should use to do the job about classification.

There may have other solution like SVM, nerual network and other machine learning algorithm. In this paper, we try to solve this problem by AdaBoost which is a very useful and efficient algorithm to do classification.

Algorithm 2 AdaBoost

Input: Give sample $(x_1, y_1), ..., (x_n, y_n)$ where $y \in \{-1, +1\}$

Output: G is the strong classifier which produce by this algorithm.

Steps of algorithm

• Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively.

• For t = 1, ..., T

1. normalize the weight $w_{t,i} = \frac{w_{t,i}}{\sum_{i=1}^{n} w_{t,i}}$
2. select the best weak classifier with respect to the weighted error rate.
3. Define $g_t(x) = g(x, f_t, p_t, \theta_t)$ where $f_t, p_t, and\theta_t$ are the minimizers of ε_t
4. Update the weights

• The final strong classifier is:

$$G(x) = \begin{cases} 1 \text{ if } \sum_{t=1}^{T} \alpha_t g_t(x) >= \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 \text{ if } otherwise \end{cases} \tag{7}$$

Here is a demonstration that the features which are selected by AdaBoost. For presentation, there are 10 features in the following figure.

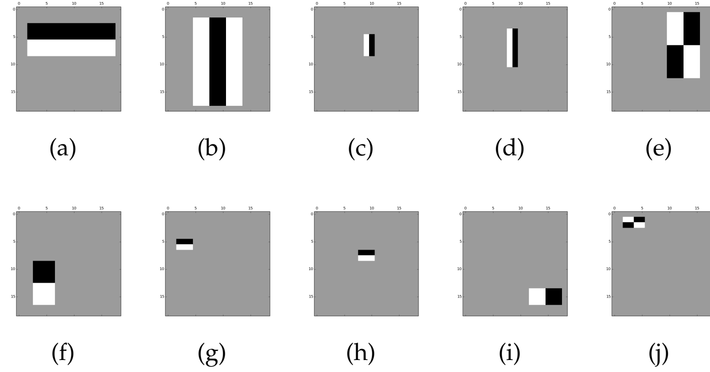
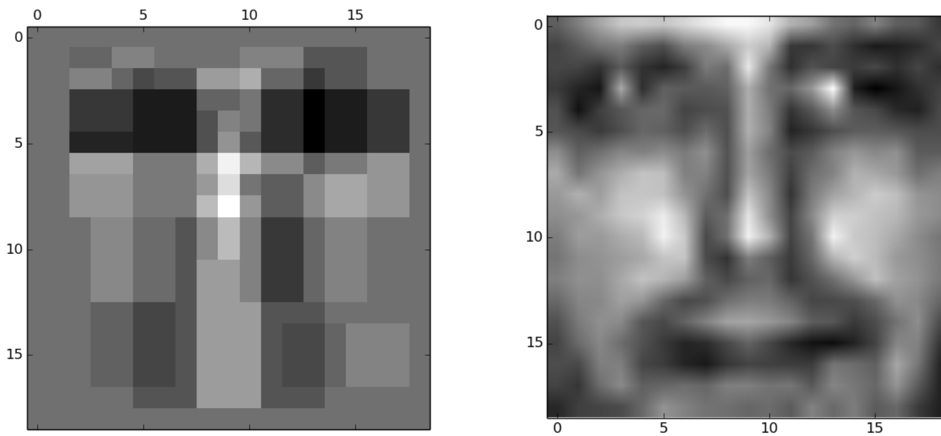


Figure 6: Ten features selected by AdaBoost



(a) boosted classifier by feature shown in figure 6

(b) A human face from training set

Figure 7: The final strong classifier with boosted 10 weak classifier and human face

Here, I compare the final classifier image with a training sample of human face. You will find that the more final classifier image look like to human face, the better final classifier are.

Why not use a human face image as a classifier directly?

My answer is "Do you forget what means overfitting? "

Look at table 1, I show a detail information about the final classifier. All the information are cached in local file */model/model.cache*

Table 1: Detail information about the final classifier

Figure	1	2	3	4	5	6	7
Feature Number	5287	13455	5797	6091	15360	1988	1165
α (Voting power)	1.9266	1.4872	0.9625	1.0217	0.9213	0.7948	0.6828
p (Direction)	+1	-1	+1	-1	+1	-1	+1
Figure	8	9	10				
Feature Number	1214	7937	14491				
α (Voting power)	0.7103	0.6336	0.6116				
p (Direction)	+1	+1	-1				

We have got the final classifier. We also know the threshold of each selected weak classifier. But what about the threshold of final classifier ? Here we should introduce some concepts which are used in the definition of the ROC-curve.

Table 2: A classifier predicts the class of a test example

Label	Predicted Class	True Class
True-Positive (tp)	Positive	Positive
False-Positive(fp)	Positive	Negative
True-Negative (tn)	Negative	Negative
False-Negative(fn)	Negative	Positive

Our target is to try our best to detect more face and avoid to mis-detect a non-face region as a face. That's something listening like quadratic programming. When we want to maximize the tpr(True-Postive Rate) and minimize the fpr(False-Negative Rate).

$$TruePostiveRate = tpr = \frac{n_{tp}}{n_{tp}+n_{fn}} \tag{8}$$

$$FalsePositiveRate = fpr = \frac{n_{fp}}{n_{tn}+n_{fp}} \tag{9}$$

n_{tp} is the number of True-Postive etc. It's not hard to understand that tpr and fpr will vary depending on the threshold applied to the final strong classifier. The ROC(Receiver operating characteristic) curve is a way to summarize this variation. It is a curve that plots fpr Vs tpr as the threshold varies from $-\infty$ to $+\infty$.

Don't forget we didn't have a good threshold for our final classifier. The ROC curve help us to choose a good threshold to do classification finally.

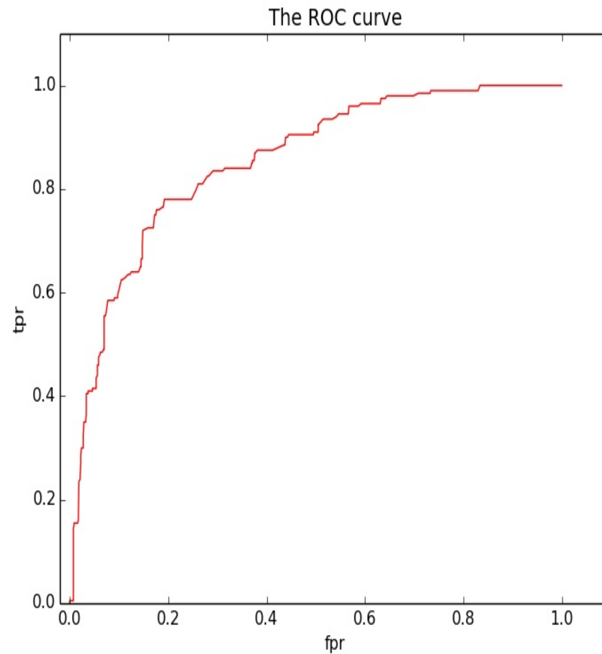


Figure 8: ROC cure computed from the images with 200 positive samples and 800 negative samples

With the limitation of my computer, I only used 1000 samples to training our model. If my face detection system could run on more powerful workstation, I think the result will be more beautiful.

Table 3: A classifier predicts the class of a test example

FinalThreshold	-15.0	-6.62	-2.1	0.4	5.7	10.0
tpr	1.0	0.99	0.81	0.615	0.005	0.00
fpr	1.0	0.8275	0.27	0.1025	0.0025	0.00

We extracted some classical data from total set and table 3 show us that how the tpr and fpr will vary when the final threshold is changing. In our experiment, the final threshold is 3.1

Here is could glance at the detail about how to compute the ROC Curve in our detection system.

```
def showROC(self):
    tprs, fprs = [], []
    best_tpr, best_fpr, best_th  = 0., 1., None
    for t in numpy.arange(AB_TH_MIN, AB_TH_MAX, 0.02):
        output = self.prediction(self._Mat, t)

        Num_tp, Num_fn, Num_tn, Num_fp = 0, 0, 0, 0
        for i in range(self.SamplesNum):
            if self._Tag[i] == LABEL_POSITIVE:
                if output[i] == LABEL_POSITIVE:
                    Num_tp += 1
                else:
                    Num_fn += 1
            else:
                if output[i] == LABEL_POSITIVE:
                    Num_fp += 1
                else:
                    Num_tn += 1

        tpr = Num_tp * 1./(Num_tp + Num_fn)
        fpr = Num_fp * 1./(Num_tn + Num_fp)

        if tpr >= best_tpr and fpr <= best_fpr:
            best_tpr, best_fpr, best_th  = tpr, fpr, t

        tprs.append(tpr)
        fprs.append(fpr)
```

6 Face Detection and optimaization

6.1 Search and detection

After we learnt a strong classifier. Now, we get the final part of the detection system. Normaly, we do search work and travel all sub-window as figure 9 shown. Every sub-window with size 19 x 19 may contain a face image, we may have to try every sub-window, or almost every.

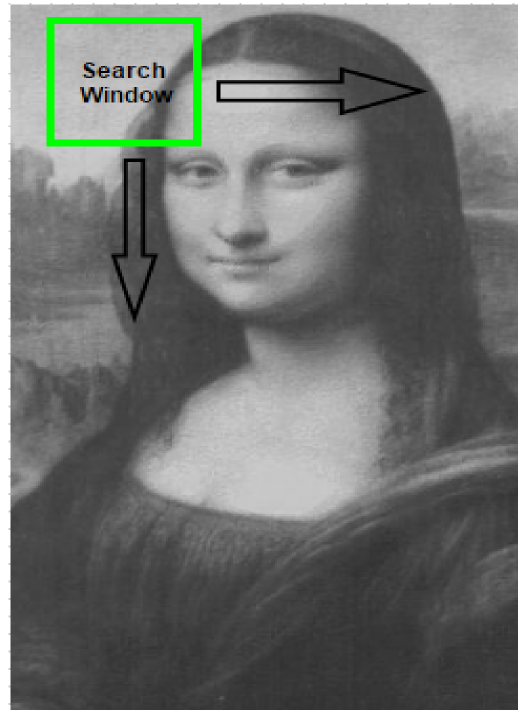


Figure 9: Mona Lisa

Becasue of the size of our detector is 19 x 19 and the training set image are also that size. It's very small for our daily image. It's nessesary to resize the original image into a smaller one.

In our implementation, I write a function scanImgFixedWin(image, scale) which take two arguments @image and @scale. The @image is a single channel image and @scale is in (0., 1.). This function return a list of sub-window which have a face which is predicted by the AdaBoost. For convenient, the returned sub-window represent as a tuple (x, y, w, h). (x, y) is the start point of the sub-window. w, h are the percentage of width and height of the original image.

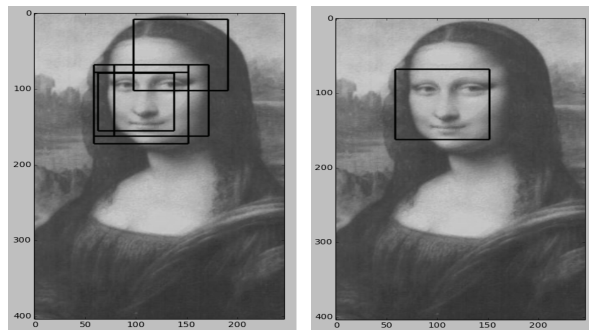
6.2 Optimalization

The figure below there shown a comparision between the detection result of the ordinary detection system and the image after optimalization. You can view some overlapped rectangel in the left figure below there. And there only one rectangle with the sinle woman who is in the image.

Algorithm 3 Deduce overlapped sub-window

Inout: a set of sub-windows which are predicted as containing a human face and a threshold to determine whether the sub-window should be reduced .
Output: set of deduced sub-windows.

for i from 1 to N **do**
  for for j from i to N **do**
    overlapArea = $S_i \cap S_j$
    totalArea = $S_i \cup S_j$
    overlapRate[i][j] = overlapArea / totalArea
  end for
end for
reduced = [1, ...,i, ..., N]
for i from 1 to N **do**
  for j from i+1 to N **do**
    if overlapRate[i][j] > overlap_threshold
    reduced[j] = reduced[i]
  end for
end for
return reduced



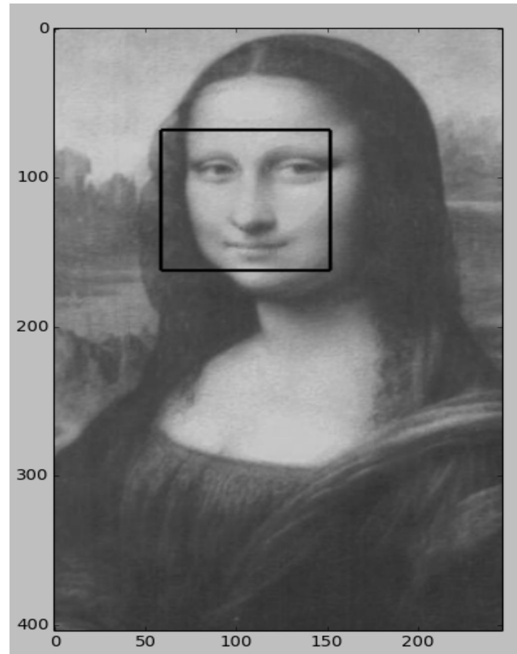
(a) Detection result of overlapped windows

(b) Detection result after optimalization

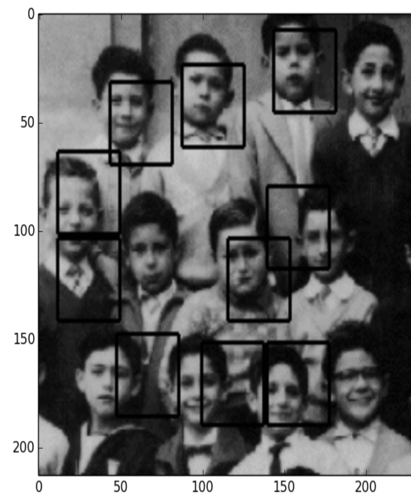
Figure 10: Comparision between two images. overlap_threshold = 0.1, scale range $\in (0.2, 0.35)$

7 Results

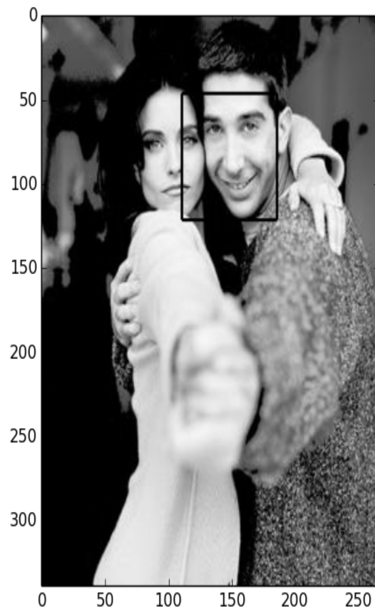
Some test on normal images are shown below there.



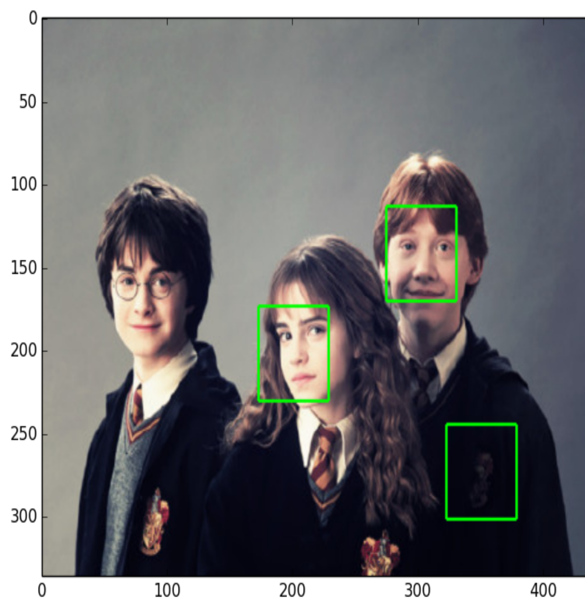
(a)



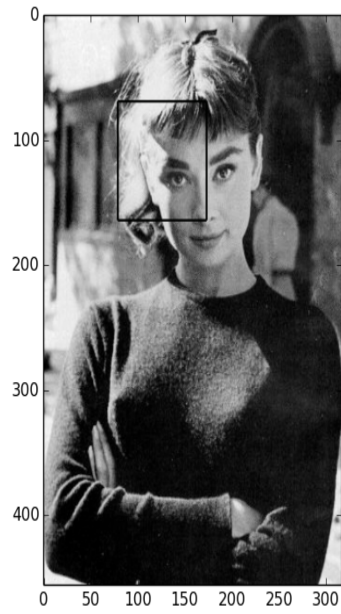
(b) scale = 0.4, Final_th = 0.3



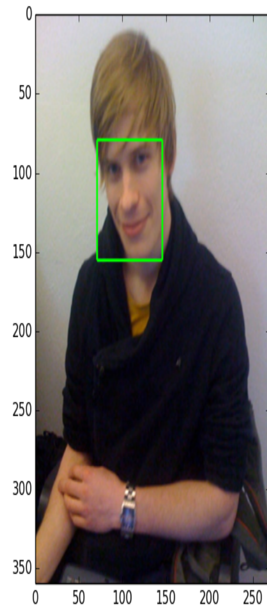
(c) scale = 0.25



(d) scale = 0.3, Final_th = 1.6, overlap_th = 0.1



(e) scale = 0.2, Final_th = 1.8, overlap_th = 0.1



(f) scale = 0.25, Final_th = 1.8, overlap_th = 0.1

(g) scale = 0.25, Final_th = 1.8, overlap_th = 0.1

8 Details of Implementation

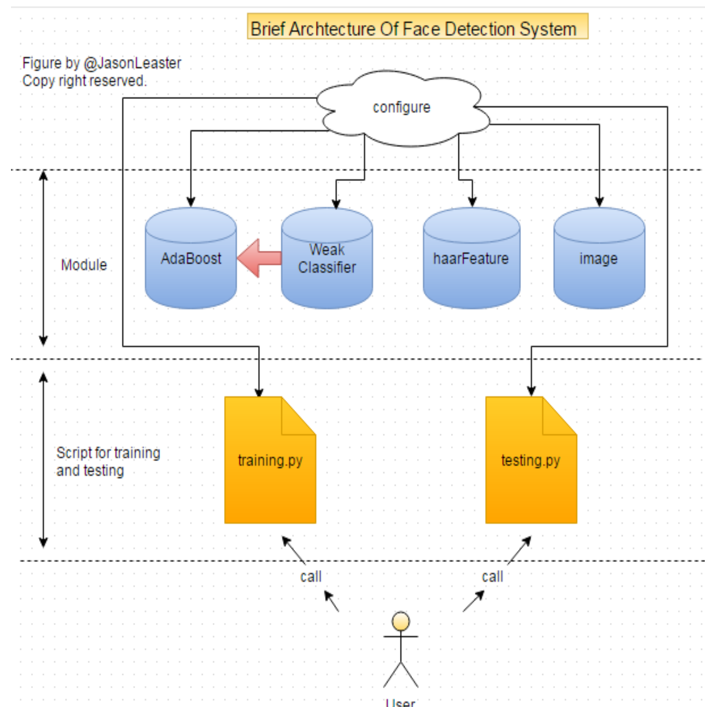


Figure 11: scale = 0.2, Final_th = 1.8, overlap_th = 0.1

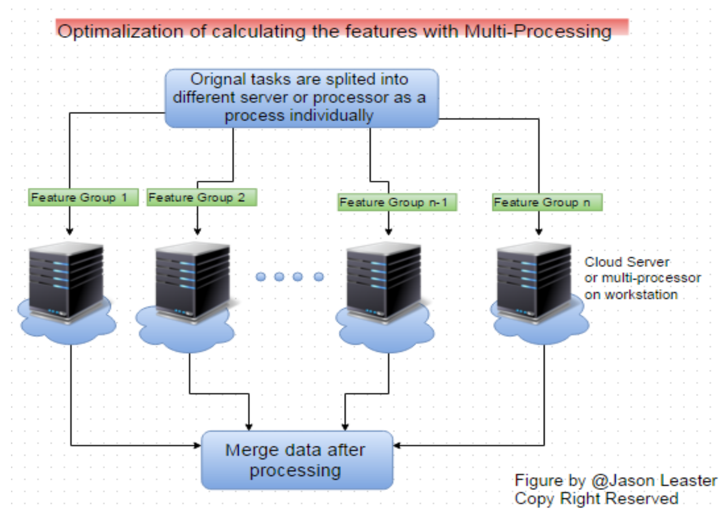


Figure 12: optimalization with Multi-Processing

Tell me and I forget, teach me and I may remember, involve me and I learn
– Benjamin Franklin