

# ECEN 649: Pattern Recognition

## AdaBoost for Face Recognition

Li-Hen Chen 928003907

### 1. Introduction

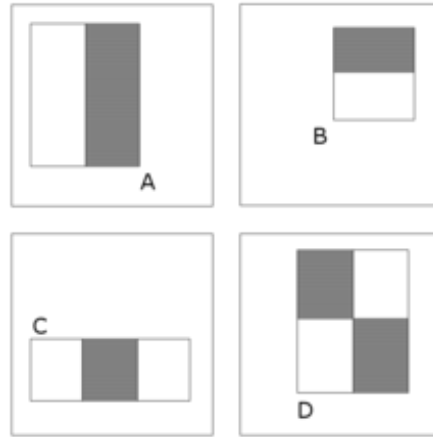
Developed by Paul Viola and Michael Jones back in 2001, the Viola-Jones Object Detection Framework can quickly and accurately detect objects in images and works particularly well with the human face (Viola & Jones, 2001). Despite its age, the framework is still a leading player in face detection along side many of its CNNs counter parts. The Viola-Jones Object Detection Framework combines the concepts of Haar-like Features, Integral Images, the AdaBoost Algorithm, and the Cascade Classifier to create a system for object detection that is fast and accurate. In order to implement the framework, we first need to understand each of these concepts individually and then figure out how they connect together to form the framework. The concepts and methodology are described in the part two. The homework question results are presented in the part three. **For the usage of this package, please refer to part six-Appendix.**

### 2. Methodology

#### 2.1 Haar Features

Human faces have some similar properties. These regularities may be extracted using Haar Features. A few properties are:

- The eye region is darker than the upper-cheeks.
- The nose bridge region is brighter than the eyes.
- Composition of properties forming matchable facial features:



The Viola Jones algorithm mainly uses the five kinds of feature such as two-horizontal, two-vertical, three-horizontal, three-vertical, four as shown in the above figure. Our dataset is of images that are  $19 \times 19$ . Take the two-vertical feature as an example, for the given width and height limitation of 19, apparently it can have the following possible rectangles:  $1 \times 2, 1 \times 4, 1 \times 6, \dots, 1 \times 18, 2 \times 2, 2 \times 4, \dots, 2 \times 18, 3 \times 2, \dots, 19 \times 18$ . In order to reduce the computation time, I have set the upper bound and the lower bound for the width and the height.

The implementation can be found in `haar.py` and `feature_type.py`.

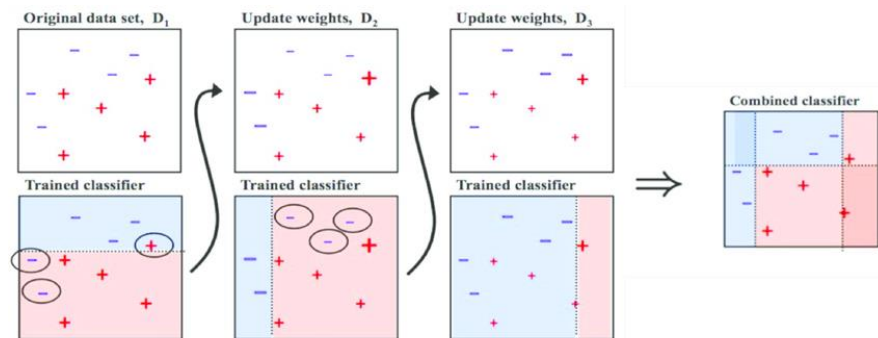
## 2.2 Integral Image

As we discussed above, there are extremely large number for a single image. Thus, we can adopt integral image to speed up the feature computation. Integral Image is used to speed up calculating the sum of a rectangle area in a 2D matrix as shown in the figure. We can easily calculate the sum of a rectangle area by using the integral image. The implementation to calculate integral image can be found in `utils.py`

Original	Integral	Original	Integral																																																																																																				
<table><tr><td>5</td><td>2</td><td>3</td><td>4</td><td>1</td></tr><tr><td>1</td><td>5</td><td>4</td><td>2</td><td>3</td></tr><tr><td>2</td><td>2</td><td>1</td><td>3</td><td>4</td></tr><tr><td>3</td><td>5</td><td>6</td><td>4</td><td>5</td></tr><tr><td>4</td><td>1</td><td>3</td><td>2</td><td>6</td></tr></table>	5	2	3	4	1	1	5	4	2	3	2	2	1	3	4	3	5	6	4	5	4	1	3	2	6	<table><tr><td>5</td><td>7</td><td>10</td><td>14</td><td>15</td></tr><tr><td>6</td><td>13</td><td>20</td><td>26</td><td>30</td></tr><tr><td>8</td><td>17</td><td>25</td><td>34</td><td>42</td></tr><tr><td>11</td><td>25</td><td>39</td><td>52</td><td>65</td></tr><tr><td>15</td><td>30</td><td>47</td><td>62</td><td>81</td></tr></table>	5	7	10	14	15	6	13	20	26	30	8	17	25	34	42	11	25	39	52	65	15	30	47	62	81	<table><tr><td>5</td><td>2</td><td>3</td><td>4</td><td>1</td></tr><tr><td>1</td><td>5</td><td>4</td><td>2</td><td>3</td></tr><tr><td>2</td><td>2</td><td>1</td><td>3</td><td>4</td></tr><tr><td>3</td><td>5</td><td>6</td><td>4</td><td>5</td></tr><tr><td>4</td><td>1</td><td>3</td><td>2</td><td>6</td></tr></table>	5	2	3	4	1	1	5	4	2	3	2	2	1	3	4	3	5	6	4	5	4	1	3	2	6	<table><tr><td>5</td><td>7</td><td>10</td><td>14</td><td>15</td></tr><tr><td>6</td><td>13</td><td>20</td><td>26</td><td>30</td></tr><tr><td>8</td><td>17</td><td>25</td><td>34</td><td>42</td></tr><tr><td>11</td><td>25</td><td>39</td><td>52</td><td>65</td></tr><tr><td>15</td><td>30</td><td>47</td><td>62</td><td>81</td></tr></table>	5	7	10	14	15	6	13	20	26	30	8	17	25	34	42	11	25	39	52	65	15	30	47	62	81
5	2	3	4	1																																																																																																			
1	5	4	2	3																																																																																																			
2	2	1	3	4																																																																																																			
3	5	6	4	5																																																																																																			
4	1	3	2	6																																																																																																			
5	7	10	14	15																																																																																																			
6	13	20	26	30																																																																																																			
8	17	25	34	42																																																																																																			
11	25	39	52	65																																																																																																			
15	30	47	62	81																																																																																																			
5	2	3	4	1																																																																																																			
1	5	4	2	3																																																																																																			
2	2	1	3	4																																																																																																			
3	5	6	4	5																																																																																																			
4	1	3	2	6																																																																																																			
5	7	10	14	15																																																																																																			
6	13	20	26	30																																																																																																			
8	17	25	34	42																																																																																																			
11	25	39	52	65																																																																																																			
15	30	47	62	81																																																																																																			
$5 + 2 + 3 + 1 + 5 + 4 = 20$		$5 + 4 + 2 + 2 + 1 + 3 = 17$																																																																																																					
		$34 - 14 - 8 + 5 = 17$																																																																																																					

## 2.3 Adaboost Training

Once we finish extracting the haar features for images, we can train the classifier by Adaboost. Adaboost is designed for classification problems and aims to convert a set of weak classifiers into a strong one which can be seen in the following figure. The Adaboost is implemented in the `adaboost.py` file which is a class called “AdaBoost” with a public function called `learn`.



## 2.4 Building Cascade System

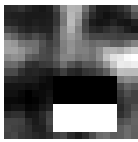
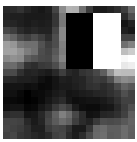
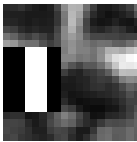
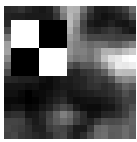
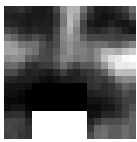
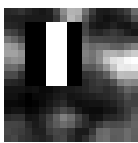
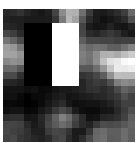
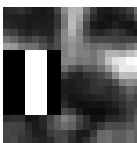

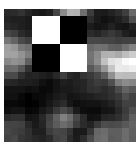
In cascading, each stage consists of a strong classifier. So all the features are grouped into several stages where each stage has certain number of features. The job of each stage is to determine whether a given sub-window is definitely not a face or may be a face. A given sub-window is immediately discarded as not a face if it fails in any of the stages. The implementation will be discussed in the section 3.4.2.

## 3. Results and Analysis

In this report, we have implemented and finished three basic problems and one bonus problem. The previous section described how whole the pipeline is implemented. The results of four problems including top 10 features selected by AdaBoost, the combined classifiers, ROC curve, statistical results and the improvement of performance.

### 3.1 Top 10 features selected by AdaBoost

The first question of homework 5 required us to show the top ten features selected by AdaBoost. In order to make the features self-explanatory, I visualized the top ten features by projecting these features to a testing face. One thing to notice is that I set the upper-bound and lower-bound for the sliding window in order to speed up the training process, some out-of-bound features would not be visualized in this case. The results show that the features we selected are important features on the human face such as mouth, nose, eyes.

Feature	1	2	3	4	5
					
Feature	6	7	8	9	10
					

### 3.2 Combined Classifiers

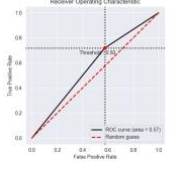
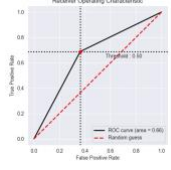
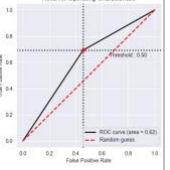
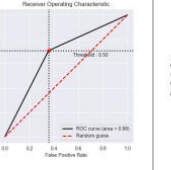
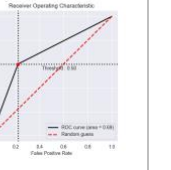
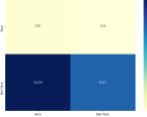
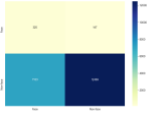
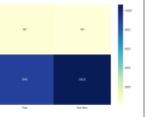
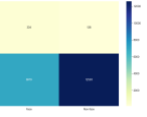

The second question of homework 5 required us to show the combined classifier after running 1, 3 5 and 10 boosting rounds. I listed the output of ten weak learners of first ten rounds including the threshold, weights...

Round	1	2	3	4	5
Best error	0.24	0.36	0.29	0.33	0.38
Feature weight	0.55	0.29	0.42	0.33	0.24

Round	6	7	8	9	10
Best error	0.34	0.33	0.38	0.35	0.40
Feature weight	0.31	0.34	0.24	0.29	0.20

### 3.3 ROC Curve and Statistical Results

The ROC curves of the combined classifiers after running 1, 3, 5 and 10 boosting rounds are plotted separately as the following table. Besides, some useful metrics are stated in the table too. We can tell from the table that the model performs best when running 20 boosting round.

Round	1	3	5	10	20
ROC Curve					
Confusion Matrix					
F1	0.056	0.082	0.067	0.085	0.111
Accuracy Faces	71.8%	68.8%	69.2%	70.7%	59.9%
Accuracy Non-Faces	42.4%	63.7%	54.33%	64.3%	77.8%

### 3.4 Performance Improvement (bonus)

In this section, I will discuss the ideas and the implementations of performance improvement in two parts.

#### 3.4.1 Efficiency Improvement

There are few methods I have tried to improve the efficiency of the training of Adaboost including restricting the feature size and the multiprocessing parallel computing.

As we discussed in the previous section, even for a low-resolution image, it still has huge amount of the Haas features. As a result, I set an upper-bound and lower-bound for the sliding window which reduce the computational time significantly. Before setting the restriction to height and width of the sliding window, it took almost two hours to train a single Adaboost model. After adding the restriction, the whole training process took only less than five minutes.

Since each Haas feature is independent, we can easily adopt the multiprocessing module provided by Python to speed up the Haas feature computation.

Multiprocessing is a package that supports spawning processes using an API similar

to the threading module. In the adaboost.py file, we import the multiprocessing library from naïve Python package and apply Pool class which is on the Haas feature calculation. Once we implemented multiprocessing, CPU with multiprocessor can be fully used by the Python script in the feature selection part.

### **3.4.2 Accuracy Improvement**

When it comes to improving the Adaboost accuracy, there are primarily three hyperparameters that you can tune to improve the performance of AdaBoost: The number of estimators, learning rate and maximum number of splits. As we can tell from the section 3.2 that increasing the number of estimators can improve the performance. We can tell from the results in the previous section, when we tried 20 estimators (20 rounds) we have better f1 score as well as roc curve.

Another idea is that Adaboost is the combination of multiple weak learners, it makes Adaboost a strong learner. I believe we can add more strong learners and utilize the bagging which is an ensemble method to vote the best prediction of the input. The implementation can be found in the main.py. The result showed the performance improved slightly but not much.

Besides, we can tell from the original paper that cascading would be another way to improve the accuracy.

## **4. Conclusion**

We have implemented the pipeline for Viola-Jones Face Detection algorithm including Haar-like Features, Integral Images, the AdaBoost Algorithm, and the Cascade Classifier. Besides, I also completed the bonus part describing in the efficiency and the accuracy separately. It turns out that we can achieve almost 70% accuracy with ten weak classifiers. I think the limitation of the algorithm is its time complexity regarding to the Haar features which leads to the model can only be trained on the low-resolution photos. Once the computational

## **5. Reference**

1. Viola, P., & Jones, M. J. (2004). Robust real-time face detection. *International journal of computer vision*, 57(2), 137-154.
2. Viola-Jones object detection framework. (25 October 2020). In *wikipedia*.  
[https://en.wikipedia.org/wiki/Viola%E2%80%93Jones\\_object\\_detection\\_framework](https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework)

## **6. Appendix**

### **6.1 Pipeline**

Please refer to README.md

Install the required python libraries for reading images, output visualization and scientific computing.

```
pip install -r requirements.txt
```

Run the main.py script to train the AdaBoost model, visualize the top ten selected features, draw the ROC curve, show the statistical results and the verification of performance improvement.

```
python3 main.py
```

### **6.2 File Description**

- `adaboost.py`: Class for adaboost training
- `haar.py`: Class for extracting Haar feature
- `feature_type.py`: Enum class to represent the different types of Haar feature
- `utils.py`: Utils including data visualization, summed-area table calculation, adaboost counting vote.

**6.3 Github:** <https://github.com/LeonChen66/AdaBoost-for-Face-Recognition>