ML Viola Jones Algorithm

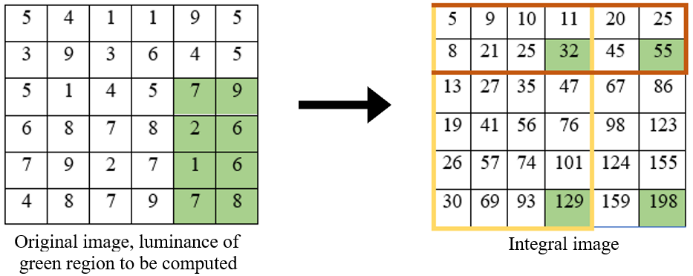
In 2001, Cambridge based researchers Paul Viola and Michael Jones introduced a highly effective machine learning (ML) approach to face detection (not face identification!). Classifiers are used with simple luminance-based features (FIG1) in a cascaded fashion, to ultimately determine whether a given image contains a face (positive) or not (negative). The method works on converted grayscale images, and its main hallmarks are:

* An image representation technique called the *Integral Image*. This makes it very quick to compare luminance values of different pixel regions.
* An *AdaBoost*-based learning algorithm which ranks different features by relevance, when given a set of positive and negative training samples. In this context, the samples are images of faces.
* “Cascading” classifiers of increasing computational demand, which are applied to every “sub-window”. A sub-window is a group of pixels, typically of size 24x24. The order in which classifiers are applied provides quick dismissal of non-face regions.

**Integral Image**

The Viola-Jones algorithm relies on the ability to compute luminance differences between pixel regions thousands of times. Despite its simple nature, performing such arithmetic becomes very slow as pixel groups grow larger. As a solution, the two researchers coined the *Integral Image* representation system. One pass is made over the whole original image from left to right, top to bottom. Each encountered pixel is replaced with the sum of all pixels **above and to the left** of it, as shown in figure X. This matrix is precomputed only **once** and stored in an intermediate form for quick referencing.

Given a grayscale image or sub-window, consider the green region in figure X for which we must determine the total luminance value (i.e., how bright, or dark the region is). Using the integral image, 4 reads (at the positions marked in green) are required to compute the region’s luminance value. Effectively, we are discarding the irrelevant brown and yellow regions, and then adding back their intersection which got removed twice.

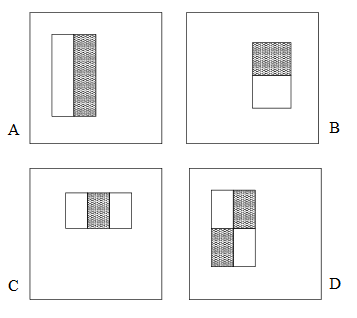


In general, computing the luminance value for rectangular regions of any size requires 4 reads when using the integral image. Without the integral image, this operation would have taken n\*m reads, where n is the number of rows, and m is the number of columns within the region. The computational overhead saved by the integral image is clear as *n* and *m* inevitably get

larger. Therefore, the time complexity is reduced from O(n^2) down to O(1). Viola-Jones propose 4 main features based on brightness differences, described in the next section. These are all combinations of rectangles and can thus be efficiently computed using the integral image similarly to figure 2.

**Feature Selection and *AdaBoost* Training**

Unlike deep learning, the Viola-Jones algorithm does not attempt to extract features itself. Instead, several Haar-like features are proposed as illustrated in figure X. These features are very simple rectangular shapes characterized by areas of contrasting luminance. Within any grayscale sub-window, these Haar-like features are present in vast amounts, more than there are pixels. Therefore, it is required to learn which **specific** Haar-like features map to **specific** facial features. Additionally, we wish to rank each relevant Haar-like feature in terms of its ability to differentiate between faces and non-faces, broadly to specifically. This is achieved using a slightly modified *AdaBoost* learning algorithm.



There are 3 different feature categories:

* *2-rectangle* features, such as *A* and *B* comprised of 2 horizontally or vertically adjacent rectangular regions. A *2-rectangle* feature’s value is the difference between summations of pixel values within the 2 rectangular regions.
* *3…*
* *4…*

With a quick look in the mirror, we can make some intuitive statements about brightness in the facial domain. For example, there is a contrast between the eyebrow and forehead regions. This can be illustrated using a variation on feature *B***.**



Another observation is that there is a contrast between the bridge of the nose and the sides of the nose.





Re. classifier, refer to vimeo bookmark… V&J’s Adaboost found the following…

1. **Your cheeks are generally brighter than your eyes!** The first feature is a 2-rectangle horizontal region that works out the difference between the area of the “eyes” and the area of the “cheeks”. These could be parts of faces or not at all (low threshold). If the feature fails, the image region is immediately discarded, and not let through to the next feature.
2. **Feature 2 tries to determine any brightness differences between nose and eyes**. Similar logic as above. If the region passes, move on to the next feature.

--------------------------------------------------NOTES-----------------------------------------------------

Based on a classifier which uses very simple features. VJ designed an efficient way of adding and subtracting (comparing) luminosity of image regions. Then, they trained a classifier to find the best features and the best order in which they can be applied for quick dismissal. It yields a good compromise between false-positives and speed. [In 2002, presented on 700MHz Pentium III], still used in cameras today. VJ can even be applied to other methods such as deep learning for FaceID, where face detection forms part of the process.

Unlike deep learning, VJ does not combine features (e.g., edges) into objects and sort them hierarchically. It makes quick decisions about what a face constitutes. Considering a grayscale image of a face: The eye is generally darker than the forehead.

VJ propose very simple rectangular features (of arbitrary pixel numbers), namely:

* 2-rectange feature. (Can be oriented both horizontally and vertically). One rectangle’s luminance value is subtracted from the other.
* 3-rectangle feature. Middle’s luminance subtracted from the outside rectangles’ luminance or vice versa.
* 4-rectangle feature. Find diagonal differences in luminance.

These features can be scaled and applied to different parts of an image. Even small images are very likely to have many combinations of these features. All features are applied to the image originally, and the most useful features for face-detection are learnt in a **training** **process**.

A problem: Performing arithmetic on large pixel groups is can be very slow.

Solution: **Integral image**:

When computing pixel group luminance differences 1000s of times over large image sections and different section combinations. With integral image, this arithmetic is precomputed (once!) and stored in an intermediate form. We do 1 pass over the image (left 🡪 right, top 🡪 bottom), and every new pixel is the sum of all pixels above and to the left. So integral image reduces number of reads/calculations. To get luminance value of any sized block, adjacent and overhead blocks are subtracted using integral image values. For example, the 4-rectangle feature takes ~9 reads using integral image.

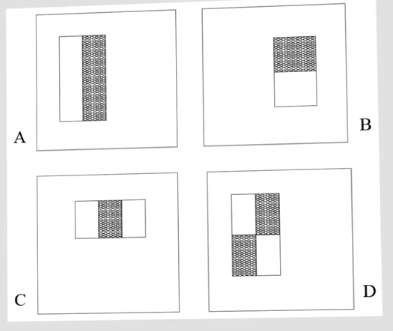
How does this act as a working face detector? VJ generally considers 24x24 pixel regions, with the ability to scale up or down. Consider a portrait. There are some features that do not make sense e.g., a vertical 2-rectangle feature applied to a whole face. This is because there are usually no significant brightness differences between 2 sides of a face. Now, for a 24x24 image, all 180,000 possible combinations of 2,3,4-rectangle features. For a given dataset of +ves and -ves, features best separating the +ves from the -ves are identified. In other words, “**Consider 10,000 portraits and 10,000 backgrounds. Which ONE feature best differentiates between faces and non-faces?**” Naturally, no single feature will correctly classify all subjects.

1. **Your cheeks are generally brighter than your eyes!** The first feature is a 2-rectangle horizontal region that works out the difference between the area of the “eyes” and the area of the “cheeks”. These could be parts of faces or not at all (low threshold). If the feature fails, the image region is immediately discarded, and not let through to the next feature.
2. **Feature 2 tries to determine any brightness differences between nose and eyes**. Similar logic as above. If the region passes, move on to the next feature.

**DEGENERATE DECISION TREE (BINARY TREE)**

The only time we need to consider ALL the features is when there is a substantial probability that the image contains a face! The features become more and more specific going forward (to ~6000). Therefore, for most of the image, the first feature(s) fails. If all stages are passed and the end is reached, VJ detects a face.

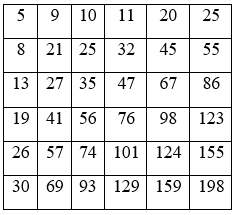
This approach is a ML approach. Put in some features and use ML to try and classify bits/whole of the image. VJ’s contribution was a quick way to calculate these features and use them to say, “There is a face in this block of image” **or** “There isn’t…”.

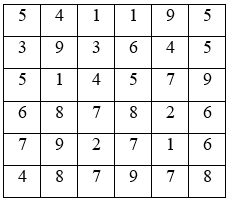


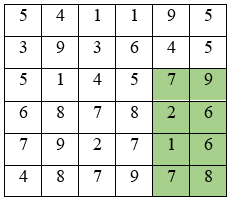
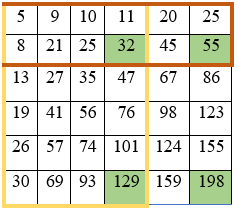
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 4 | 1 | 1 | 9 | 5 |
| 3 | 9 | 3 | 6 | 4 | 5 |
| 5 | 1 | 4 | 5 | 7 | 9 |
| 6 | 8 | 7 | 8 | 2 | 6 |
| 7 | 9 | 2 | 7 | 1 | 6 |
| 4 | 8 | 7 | 9 | 7 | 8 |

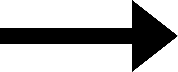
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 9 | 10 | 11 | 20 | 25 |
| 8 | 21 | 25 | 32 | 45 | 55 |
| 13 | 27 | 35 | 47 | 67 | 86 |
| 19 | 41 | 56 | 76 | 98 | 123 |
| 26 | 57 | 74 | 101 | 124 | 155 |
| 30 | 69 | 93 | 129 | 159 | 198 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 9 | 10 | 11 | 20 | 25 |
| 8 | 21 | 25 | 32 | 45 | 55 |
| 13 | 27 | 35 | 47 | 67 | 86 |
| 19 | 41 | 56 | 76 | 98 | 123 |
| 26 | 57 | 74 | 101 | 124 | 155 |
| 30 | 69 | 93 | 129 | 159 | 198 |







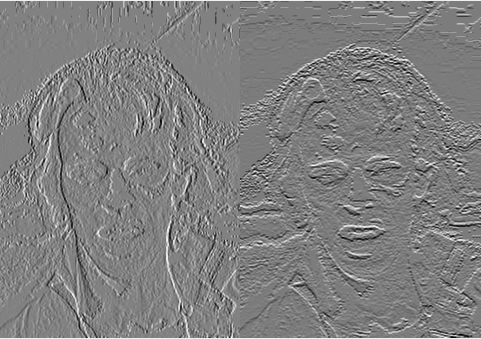


Integral image

Original image, luminance of

green region to be computed





**Task 1**

For this task, several Cascade classifiers were trained using the *OpenCV* framework. The goal is to present a frontal face detection artifact which achieves similar performance to *OpenCV*’s pre-trained alternatives. The artifact was developed into a simple terminal application using Python and the *opencv-python4.5.1.48* package. Since this task involves manual training, all training data (positive and negative) had to be manually sourced and evaluated prior to being used as official training data.

Grayscale positive images were sourced from the *LFWcrop* \cite{ <https://conradsanderson.id.au/lfwcrop/>}, a cropped version of the *Labeled Faces in the Wild (LFW)* dataset \cite{ http://vis-www.cs.umass.edu/lfw/ }. The dataset contains 13233 64x64 .PGM images of faces with minimal background interference, i.e., the face makes up most of the image. For this assignment, positive training data was limited to 2000 samples. Negative images were sourced from \cite{ <https://github.com/handaga/tutorial-haartraining/tree/master/data/negatives> }, which was compiled for the same purpose of training a group of Haar Cascades. The dataset contains 3019 640x480 .JPG images, of which none are/include faces. A good element of randomness is exhibited within the dataset (indoor and outdoor scenery, drawings and paintings of non-faces, circuit board schematics, text, graphs etc.). All negative images were resized to 128x128 in MALTAB using *imresize*. In total, the training data consists of 2000 positive and 3019 negative samples of size 64x64 and 128x128, respectively.

Before training, *OpenCV* requires a text file with entries as follows:

* path to positive image + number of positive subjects + rectangular boundaries (x1,y1) (x2,y2) for each positive subject

e.g., “*/positive/00174.png 1 0 0 128 128”.* The framework provides an executable **opencv\_annotation.exe** to generate this text file, however it involves manual onscreen selections for each image, which is impractical for 2000 images. Since we know all samples contain precisely 1 image in the same rectangular boundary (0,0), (128,128), it is most efficient to write a python function to populate the text file. For each image found within a “*positive”* directory, the function *make\_pos()* writes the string { image path + “1 0 0 128 128\n” } to a text file *pos.txt*.

**opencv\_annotation.exe** allows a user to manually select faces (or any subject) in each positive training image and writes their coordinates to a text file. Though this is very useful when creating custom datasets, the manual selection was not necessary since the faces occupy the entire image

OpenCV provides several useful executables which facilitate the training process.

, the manual selection was not needed because each positive sample contains **only** a face with no irrelevant regions. However, the text file it outputs is required, therefore a function *make\_pos()* was implemented to add a line

* **opencv\_annotation.exe**  - allows a user to manually select detection subjects from a positive dataset by drawing onscreen and writes the coordinates of all selected subjects to a text file. This is particularly useful when creating custom datasets. Because the chosen dataset contains only closeups of faces with minimal background, this program was not needed. However, the text file it outputs is required, therefore a function *make\_pos()* was implemented to add a line
* **opencv\_createsamples.exe** – creates