

University of Malta
Faculty of Information and Communication Technology
ICS2207 – Machine Learning Course Project

Coursework 2021-2022



Faculty of ICT
Scholastic year: 2021/22

Nathan Bonavia Zammit (198402L)

FACULTY OF INFORMATION AND
COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as “the unacknowledged use, as one’s own work, of work of another person, whether or not such work has been published” (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I / We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.


Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Nathan Bonavia Zammit

Student Name



Signature

Student Name

Signature

Student Name

Signature

Student Name

Signature

ICS2207

Course Code

Machine Learning Course Project

Title of work submitted

21/01/2022

Date

Contents

ICS2207 – Machine Learning Course Project.....1

How do Viola-Jones Object Detection/Haar Cascades work?4

Methodology – Artifact 17

Methodology – Artifact 210

Alternatives to Viola-Jones.....11

Statement of Completion.....12

References.....12

How do Viola-Jones Object Detection/Haar Cascades work?

Face Detection is a powerful tool with a vast range of applications. Whether it be our smartphones, laptops, security etc. They all benefit from the use of face detection as it helps the device to authenticate the identity of the user, and this occurs in real time.

All of this is thanks to Viola and Jones, who back in 2001 proposed the first ever Object Detection Framework for Real Time Face Detection in Video Footage. More commonly known as Haar Cascades, this algorithm uses edge or line detection features which were proposed by Viola and Jones in their research paper, “Rapid Object Detection using a Boosted Cascade of Simple Features”. The algorithm will be presented with positive images which consist of faces, and a lot of negative images which do not consist of any faces to train on.

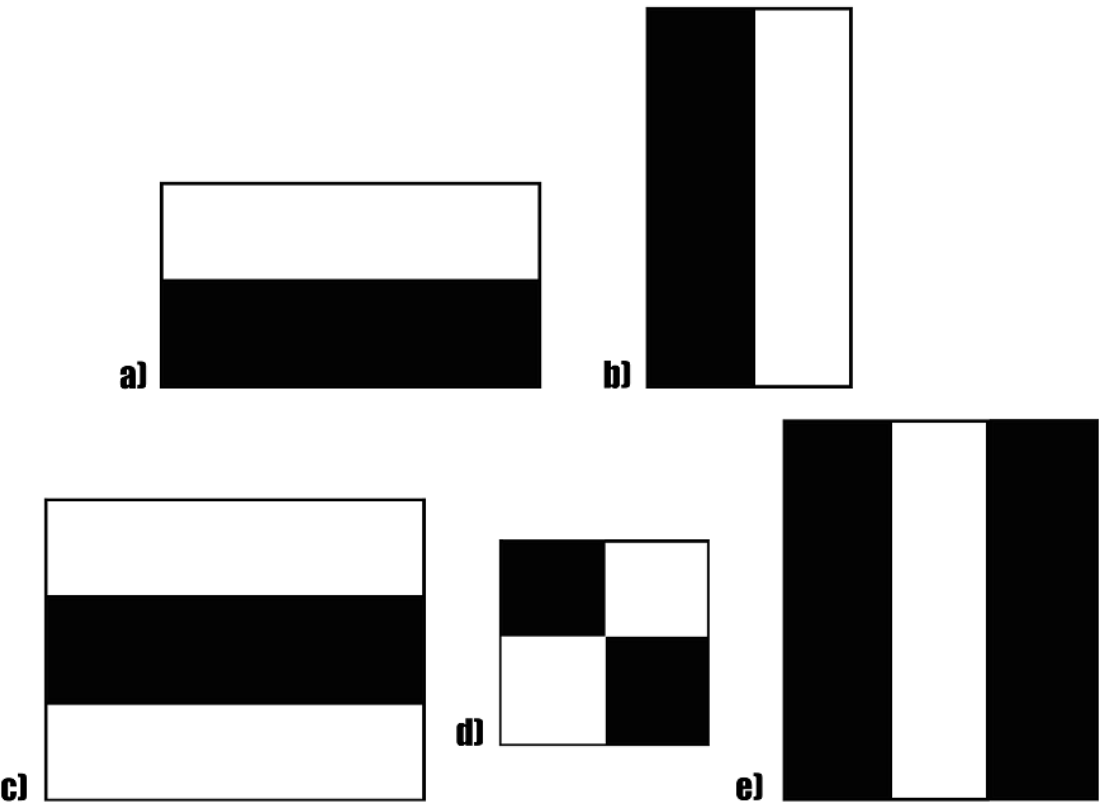


Fig: 1: A sample of Haar features used in the original paper

The Haar features which can be seen above were the first contribution to the research. The use of pixels in these rectangles are used to calculate the haar values. The darker areas have pixels with the value 1, whilst the lighter pixels have the value 0. Each pixel is responsible for finding out one particular feature in the image. Said features may include an edge, line or any structure in the image where there is a sudden change of intensities. For example, in the image below, the Haar feature is able to detect a vertical edge via the use of the darker pixels on the right, and the lighter pixels on the left.

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

Fig.2: A representation of a Haar feature with

The main objective is to find out the sum of all the image pixels. Those lying in the dark and light area respectively. After finding said values, we must find the difference. In the example that we used above, the haar value will be closer to 1 (if the dark and light sides were to swap sides or swap their values, it would be closer to 0). Therefore, we can say that an edge is detected if the haar value is closer to 1.

There are more than just the haar feature that we mentioned above. These will detect an edge in different directions or any other image structure. To detect an edge anywhere in an image, any haar feature would need to traverse the whole image. This would be done pixel by pixel throughout the entire image. Originally there were approximately 180,000 different features. A majority of said features wouldn't really work well and be very irrelevant to facial features since they'd be very random. As such, they needed a Feature Selection technique to find a subset of features from the massive set that they had. This technique needed to not only find the features which are performing better than the rest, but to eliminate the ones which are deemed as irrelevant. The technique which was used was called AdaBoost. The 180,000 features were applied to the images separately in order to create what are known as 'Weak Learners'. Some of these 'learners' managed to produce low error rates when it came to separate the positive images from the negative ones. The ones that didn't however were obviously not as efficient. Therefore, the ones with the low error rates were kept and as such they went from 180,000 features to only 6000.

These 6000 features will be used for cascading. They will run on the training images to detect if a facial feature is present or not. Let's say we have a standard window size of 24x24. The feature detection will be running in each of those windows which might be a but much. Therefore, the technique 'Attentional Cascade' comes into fruition. How it works is that not all 6000 features will be run on every single window. The features will be run on stages. If in the beginning stage, where the simpler features will be run, the features fail, then there is no need to continue running the other stages on said window. However, if the beginning stage succeeds, we move on to the latter stages, where features which are complex (look for the delicate details on the face) try to identify a face on the current window. This way a lot of processing time is saved since the irrelevant windows will not be processed for the majority of the stages.

In the Viola-Jones research they had a total of 38 stages for 6000 features. The number of features slowly increases as one goes through the stages. The use of the early stages with lesser and simpler features removed a large number of windows lacking any facial features and therefore reducing the false negative ratio. The later stages on the other hand, with more complex features, focused on reducing the error detection rate, hence achieving low false positive ratio.

Methodology – Artifact 1

For the first artifact we had to train a cascade classifier to be able to identify faces in images. First, two datasets were obtained. One named positive which contained images of people. And the second dataset named negative which contained random images, that more importantly did not have a single face in any of said images.

A function called generate_negative_description_file() was created which generated a text file called 'neg.txt'. Then the function looped through the negative dataset and took the names of each file and stored them in the text file in the format: 'negative/' + filename. The code of this function can be seen below:

```
def generate_negative_description_file():
    #open the outpur file for writing, this will overwrite all existing data in there
    with open('neg.txt', 'w') as f:
        #loop over all the filenames
        for filename in os.listdir('negative'):
            f.write('negative/' + filename + '\n')
```

Fig.3: The code for the function generate_negative_description_file()

For the 'pos.txt' file however following command was used:

```
PS C:\Users\natha\Desktop\Uni\2021-2022\1st_Semester\Machine_Learning> C:/Users/natha/Desktop/Uni/2021-2022/1st_Semester/opencv/build/x64/vc15/bin/opencv_annotation.exe
-annotations=pos.txt --images=positive/
```

The command above activates the command line program “opencv_annotation.exe”. This program is used to go through the positive dataset and draw boxes over the faces of the images in the positive dataset. This can be seen in the image below:



After going through every single image in the positive dataset, the text file 'pos.txt' is created in the same format as the 'neg.txt' file that was mentioned before. Using the 'pos.txt' text file we can create a positive vector file named 'pos.vec' using the command seen below:

```
PS C:\Users\natha\Desktop\Uni\2021-2022\1st_Semester\Machine_Learning> C:/Users/natha/Desktop/Uni/2021-2022/1st_Semester/opencv/build/x64/vc15/bin/o
pencv_createsamples.exe -info pos.txt -w 24 -h 24 -num 1000 -vec pos.vec
```

the 'pos.vec' file

The positive vector file is generated using the command line program called “opencv_createsamples.exe”. Then the width and height of the detection window size was inputted, in this case 24 pixels for each. When it came to choosing the width and height of the detection window size, more often than not the best results seemed to be obtained when using around 20 to 24 pixels. After the width and height of the detection window size, the number of vectors that would be created was inputted, which in this case was 1000. It was important for this number to be larger than

the total number of boxes were drawn on the positive images in order to have all boxes included, therefore 1000 was sufficient.

```
Info file name: pos.txt
Img file name: (NULL)
Vec file name: pos.vec
BG file name: (NULL)
Num: 1000
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 24
Height: 24
Max Scale: -1
RNG Seed: 12345
Create training samples from images collection...
pos.txt(274) : parse errorDone. Created 274 samples
```

Finally, the command line program called “opencv_traincascade.exe” was used to train the model.

```
PS C:\Users\natha\Desktop\Uni\2021-2022\1st_Semester\Machine_Learning> C:\Users\natha\Desktop\Uni\2021-2022\1st_Semester\opencv\build\x64\vc15\bin\opencv_traincascade.exe -data
cascade/ -vec pos.vec -bg neg.txt -w 24 -h 24 -numPos 200 -numNeg 100 -numStages 10
```

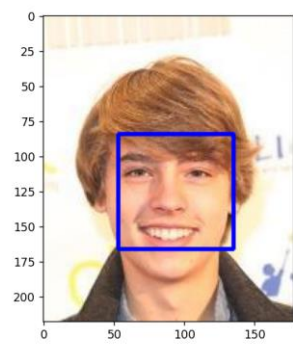
The location for where the trained stages will be stored is inputted, in this case a folder called ‘cascade’. Then of the vector file to use for the positive vector was inputted along with where to find all the negative images which in this case was done through the ‘neg.txt’ text file. Then the same width and height that was used previously for the window detection size was used and the number of positive and negative samples that will be used for the training is inputted. For the number of positive samples, it has to be less than the number of boxes drawn, so in this case 200 was used. For the negative samples this didn’t really apply as when taking the negatives, the trainer will simply take small snippets to create the negative images. The final input that was needed for the trainer was the number of stages of training. In the above image it can be seen that 10 stages were inputted, however after attempting this multiple times, the final number of stages used was 20 stages as this seemed the most efficient.

```
===== TRAINING 0-stage =====
<BEGIN
POS count : consumed 200 : 200
NEG count : acceptanceRatio 100 : 1
Precalculation time: 0.659
+-----+
| N | HR | FA |
+-----+
| 1 | 1 | 1 |
+-----+
| 2 | 1 | 1 |
+-----+
| 3 | 1 | 1 |
+-----+
| 4 | 1 | 0.67 |
+-----+
| 5 | 1 | 0.67 |
+-----+
| 6 | 1 | 0.69 |
+-----+
| 7 | 1 | 0.56 |
+-----+
| 8 | 1 | 0.61 |
+-----+
| 9 | 1 | 0.42 |
+-----+
END>
Training until now has taken 0 days 0 hours 0 minutes 6 seconds.
```

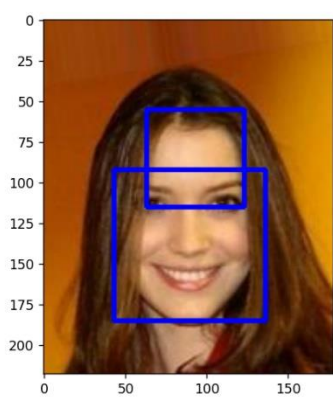
During the training after each stage is complete, a table output is displayed as seen above. HR stands for Hit Rate, FA stands for False Alarm, and N refers to the layer number. Layer 1 being the outermost layer, and layer 9 being the lowest layer with the highest number of finite details. At the bottom one can see that the time taken to complete the stage is also displayed.

What goes on during training is that the trainer will show the model a random image. This image will either be one of the boxes which we drew, or a random snippet from one of the negative images in the dataset. The model will try to predict if the image shown is one from the positive or negative dataset. The trainer will correct the model where necessary and the model will make adjustments to itself whether it was right or wrong.

When running the main program images of people should be displayed however with boxes around their faces as seen below:



However sometimes these results are not always as accurate:



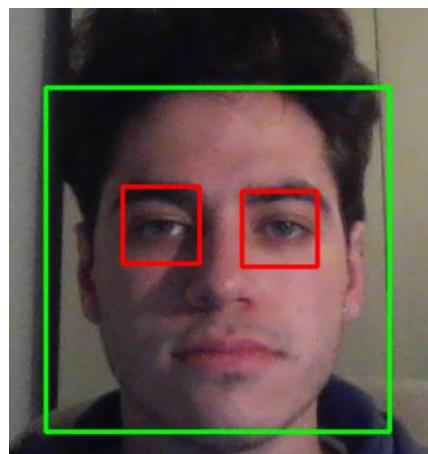
Possible Fixes or Improvements:

- If a larger positive dataset was used, then the results obtained would have been more accurate as it would have catered for different ethnic groups, ages, and so forth.
- If the boxes were drawn more accurately as this might have affected the classifier’s results.

Methodology – Artifact 2

The second artifact consisted of writing a program that made use of OpenCV's ready-made classifiers in order to detect eyes and faces.

The program will loop, taking in the input from the device's webcam. The input is then converted into grayscale via the use of the `cv2.cvtColor()` method. The image is then passed along to the classifier. The classifier uses 'haarcascade_frontalface_default.xml' for face detection and 'haarcascade_eye.xml' for eye detection. When a face is detected, a box is drawn in green around it, whilst when an eye is detected, red boxes are drawn around it as was required.



Alternatives to Viola-Jones

Convolutional Neural Networks (CNNs) can be used as an alternative to the Viola-Jones Algorithm. The difference between them is that CNNs work in non-uniform steps and as such the possibility of errors tends to increase.

CNNs are made of 4 layers:

- Convolution
- Pooling
- Rectified Linear UNITS (ReLU)
- Fully Connected Layers

Convolution involves the majority of mathematics in CNNs. During Convolution each pixel is multiplied by the value in the corresponding pixel in the image. The result is then divided by the total number of pixels in the image. Any matching pixel is given the value 1 and the others are given the value -1. This step is repeated over and over until it finds what matches the features it is looking for.

During the Pooling phase the image shrinks down to where the required features are kept in the image but the area which isn't required in the image is removed.

ReLU involves converting all the negative features from -1 to 0. This is done to simplify the mathematics which comes with this algorithm.

These 3 steps are repeated until the image has shrunk enough and the program can move to the final layer.

Fully Connected Layers looks at all the values assigned to the selected part of the image and references back to its training to see if the values correspond to a particular feature which is being looked for.

Differences:

The Viola-Jones algorithm suffers when trying to identify any faces which are not front facing and suffer from a lack of good-lighting. CNNs on the other hand are better at detecting faces in varying conditions.

CNNs however require more time consumption than Viola-Jones since there are multiple stages. This affects when this is implemented since it takes up more power consumption and memory on the device.

Statement of Completion

Item	Completed (Yes/No/Partial)
Artefact 1	Yes
Artefact 2	Yes
Artefact 2 – real time face/eye tracking	Yes
Experiments and Evaluation	Yes

References

[1] Behera, “Face Detection with Haar Cascade”
URL: <https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08>

[2] Training a Cascade Classifier - OpenCV Object Detection in Games #8
URL: <https://www.youtube.com/watch?v=XrCAvs9AePM&t=1215s>

[3] Haar Cascade Object Detection Face & Eye - OpenCV with Python for Image and Video Analysis 16
URL: <https://www.youtube.com/watch?v=88HdqNDQsEk>

[4], Enriquez, “Faster face detection using Convolutional Neural Networks & the Viola-Jones algorithm”
URL:
https://www.csustan.edu/sites/default/files/groups/University%20Honors%20Program/Journals/01_enriquez.pdf