Inha University

Face Detection using Viola Jones (VJ) Algorithm

Report

Jelle Ludovicus Petrus Goossens [12235213]

Ri Yevgeniy [12204557]

Digital Image Processing [202302-ISE4131-001]

Prof. Kakani Vijay

October 31, 2023

Abstract

To gain practical programming experience by putting the VJ algorithm into practice with OpenCV and detecting faces using Haar features. Utilization of Cascade Trainer GUI and open-source Kaggle datasets training of model took place. There are several conditions where model's performance varies depending on viewpoint variation, deformation, occlusion, illumination conditions, cluttered or textured background and intraclass variation. After several model training with diverse datasets and analysis there is conclusion, to increase performance of the model is crucial to increase number of datasets containing diverse and relevant images at the same time. As a result, the model can detect faces within images.

Introduction

The VJ algorithm, known for its efficiency and effectiveness in object detection, specifically employs Haar features to detect faces. The overarching goals of this study include the development of a real-time face detection system using the VJ algorithm and the subsequent performance evaluation of this detector. Through this laboratory experiment, we aim to gain valuable experience in programming and applying this algorithm while also assessing its capabilities in real-time facial detection scenarios. This report presents a comprehensive account of our endeavors and findings in pursuit of these objectives and goals.

Methods and Materials

In this lab experiment, our methodology involved the use of a Graphical User Interface (GUI) for the training of a Haar cascade classifier. To accomplish this, we employed face-related datasets obtained from publicly available sources, ensuring a diverse and representative range of facial images for training. Additionally, to enhance the performance of our classifier, we incorporated a self-made dataset created by our team.

Our team's self-made dataset was specifically designed to augment the training data. It includes images of all team members' faces captured under various conditions, encompassing different orientations, scales, and illumination conditions. This addition aimed to improve the classifier's ability to handle intra-class variation and adapt to diverse real-world scenarios.

Subsequently, the trained Haar cascade classifier was utilized for testing and evaluation. We employed this model to detect and analyze facial features, assessing its accuracy, sensitivity to orientation, and performance under varying lighting conditions. The combination of publicly available datasets and our self-made dataset was instrumental in training a robust classifier for our evaluation purposes.

Results

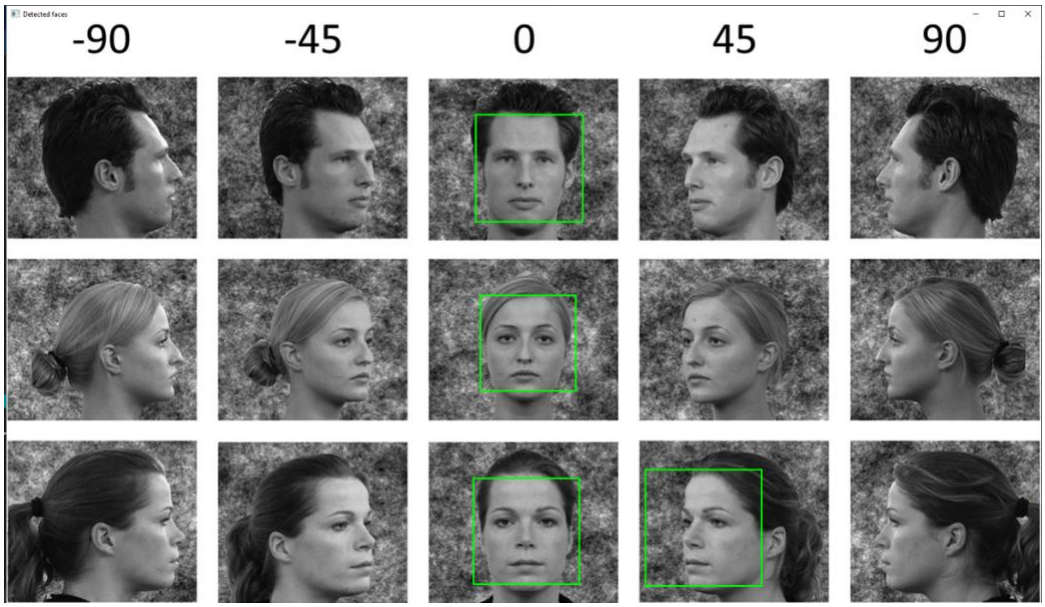
To know what to improve in our trained model we decide to compare results with open-source model of OpenCV.

1. Viewpoint variation

This challenge goes over the ability to handle different viewpoint/ angles of faces.

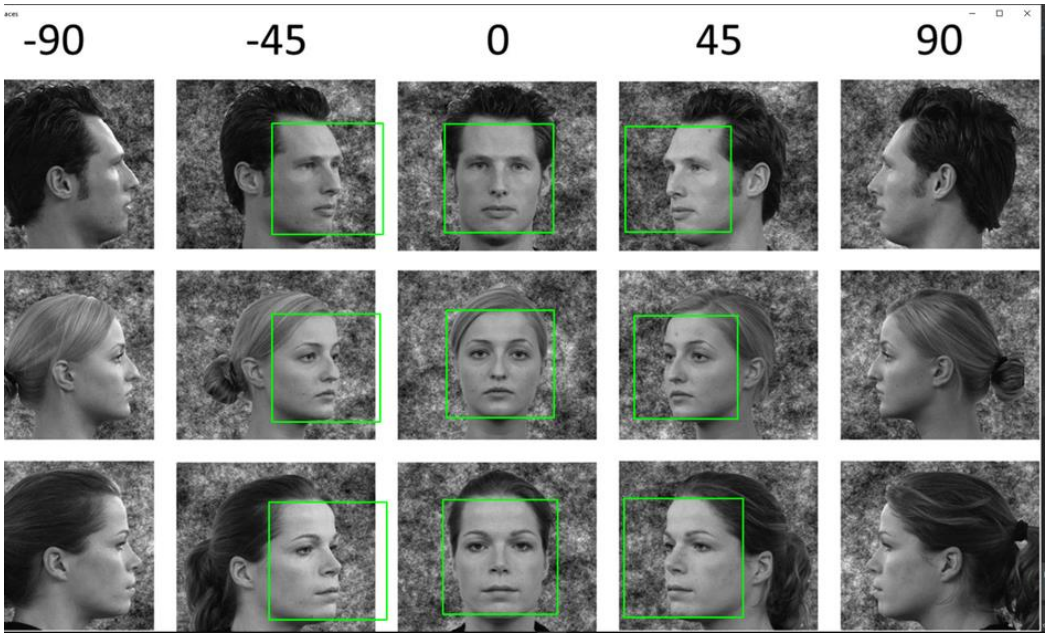
Our model

For this image the model struggles with angles of 45 or bigger. In the demo we show that the model can handle angles between 0 and 45.



Performance of Trained Model at Viewpoint Variation Example model

Example model



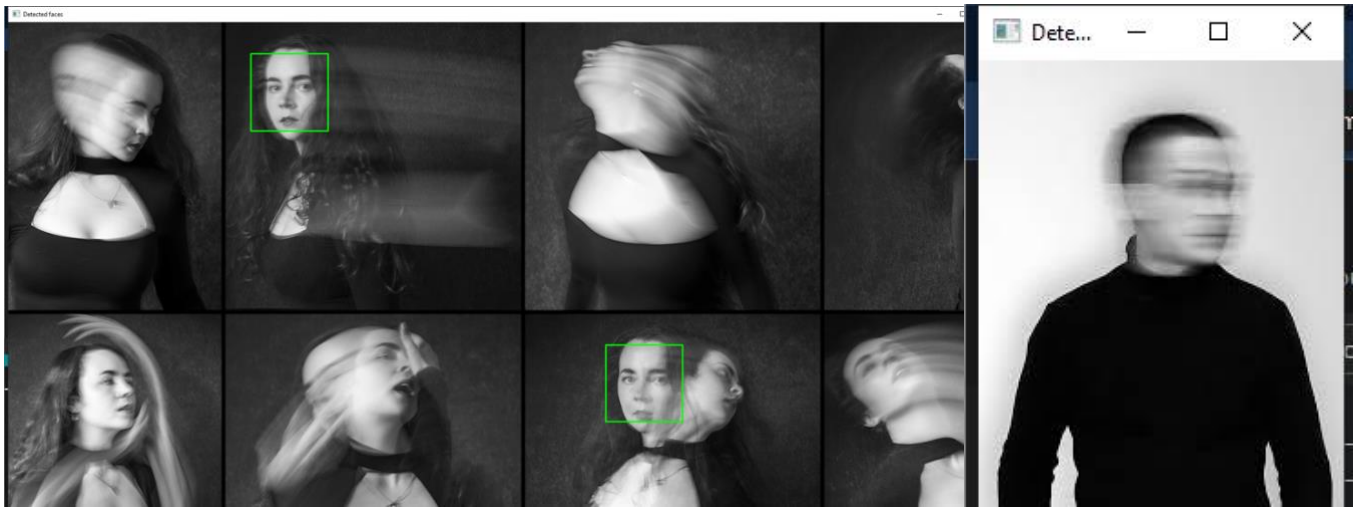
Performance of Reference Model at Viewpoint Variation

When dealing with large viewpoint variation, the VJ detector performs poorly. It works better when detecting items with more conventional orientations. The VJ detector depends on simple rectangular features and may miss items with extreme perspectives, making it difficult to recognize things from varied angles.

2. Deformation

This challenge goes over distorted faces:

Our model



Performance of Trained Model at Image Deformation

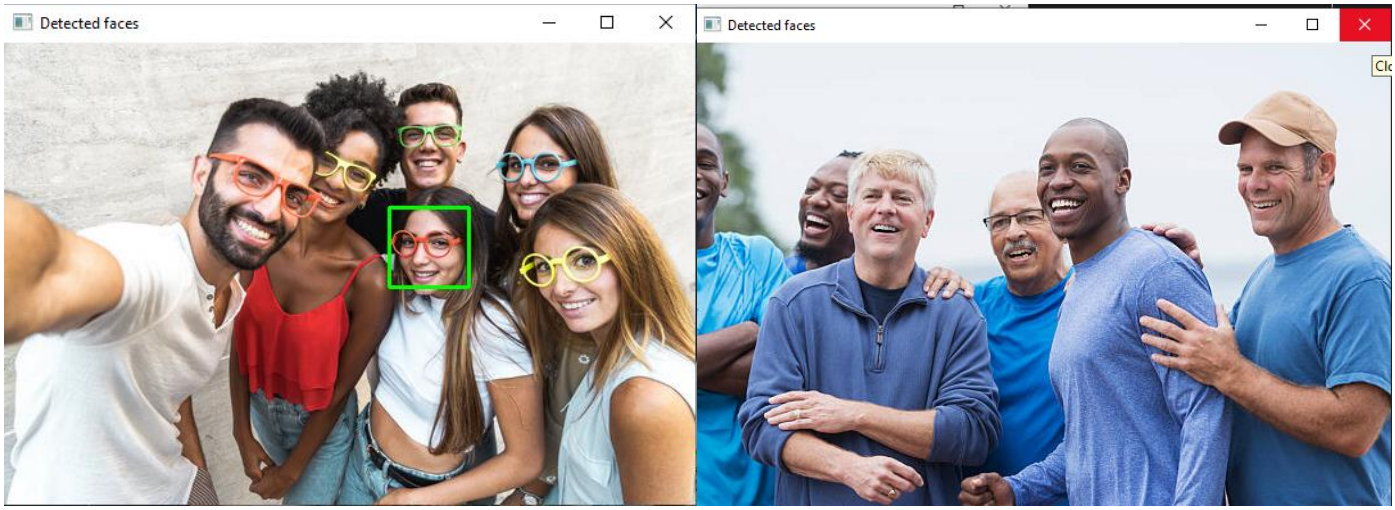
Example model

Performance of Reference Model at Image Deformation

The VJ detector experiences difficulties with deforming objects, much like viewpoint variation. It might not be able to accurately depict intricate and pliable changes in item shape. It can be challenging for the VJ detector to process objects with different deformations, such faces in the face detection scenario.

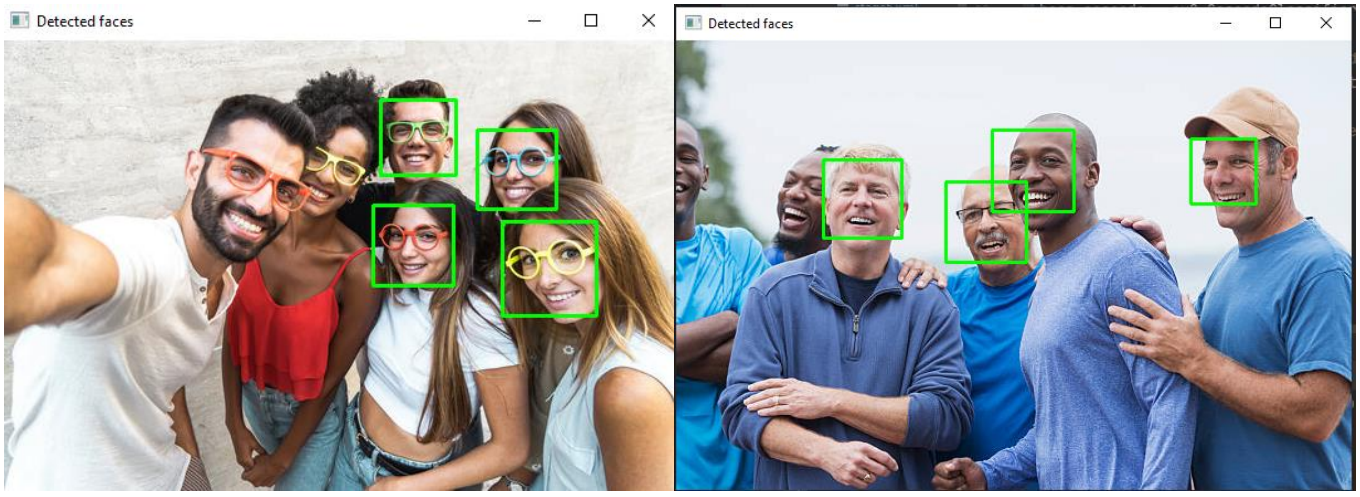
3. Occlusion

Our model



Performance of Trained Model at Image Occlusion

Example model



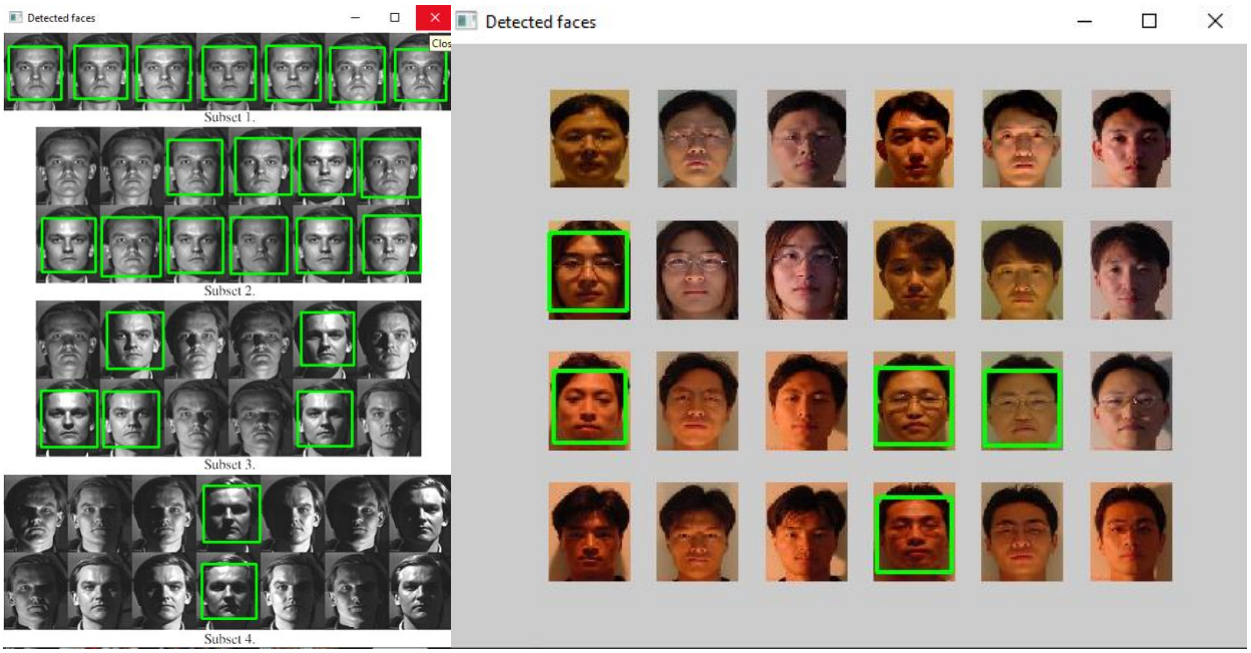
Performance of Reference Model at Image Occlusion

Although the VJ detector is not very resilient in situations of extreme occlusion, it can tolerate some occlusion. While objects that are partially obscured can still be identified, the accuracy of the detection can decline dramatically as the degree of occlusion rises.

4. Illumination conditions

Our model

Our model does work on different lighting conditions, it only when the lighting is equal on the whole face. It struggles when only half of the faces is lit up or covered in shadows.



Performance of Trained Model at Different Illumination Conditions

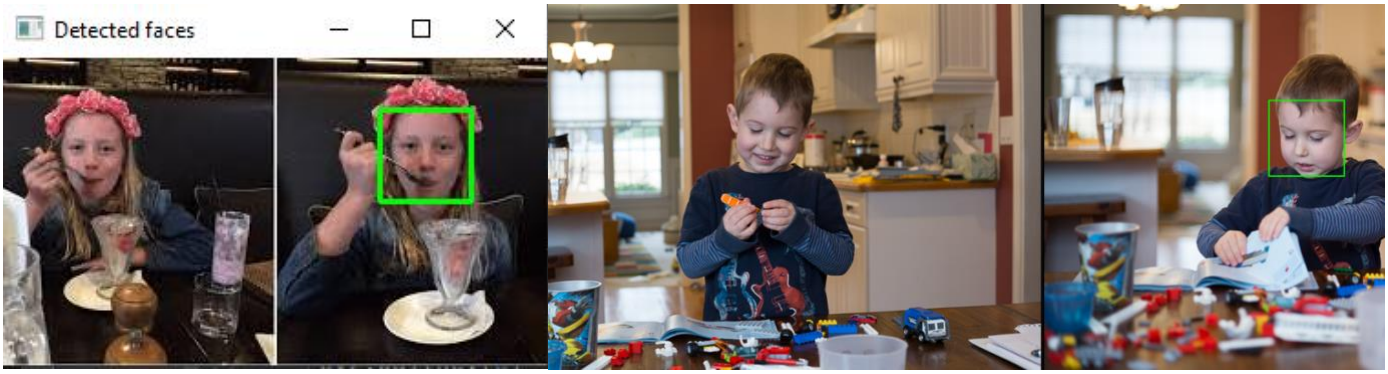
Example model

Performance of Reference Model at Different Illumination Conditions

Variations in illumination can cause the VJ detector to alter. It works best under well-regulated, steady illumination. Extreme or fluctuating lighting might cause the detector's accuracy to drop, which can result in missing or false-positive detections.

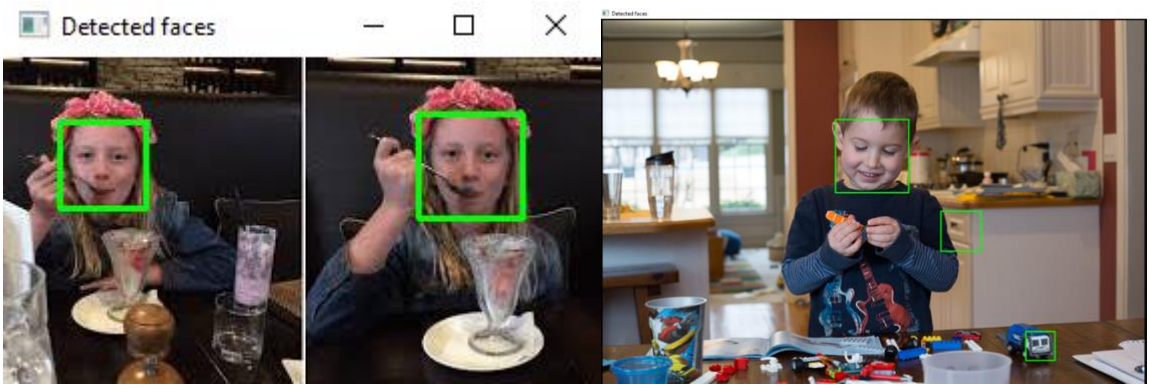
5. Cluttered or textured Background

Our model



Performance of Trained Model at Cluttered or textured Background

Example model



Performance of Reference Model at Cluttered or textured Background

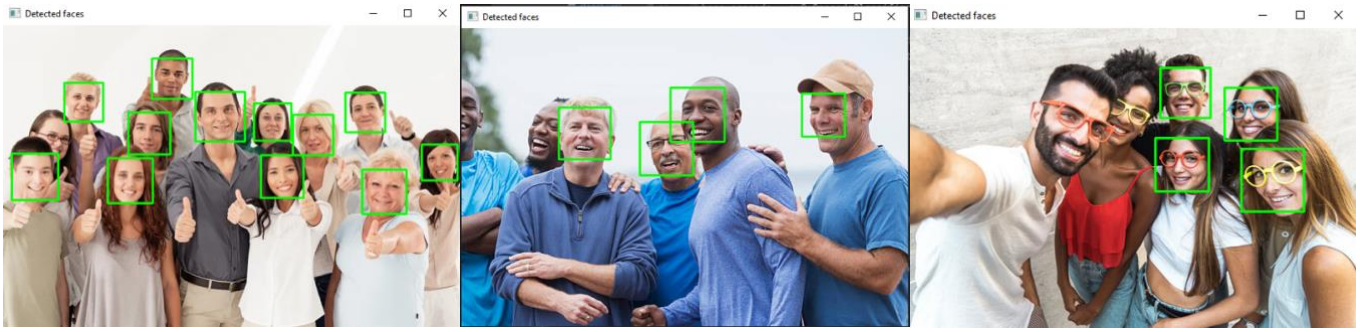
When objects are placed against textured or busy backdrops, the VJ detector may have trouble picking them out. When the object being recognized is comparable to similar patterns or textures in the backdrop, it may give false positives. In congested settings, this may result in less accuracy.

6. Intra-class variation

Our model

Performance of Trained Model at Intra-Class Variation

Example model



Performance of Reference Model at Intra-Class Variation

Intra-class variation, where objects of the same class have significant differences, can challenge the VJ detector. It may struggle to generalize well to objects with diverse appearances within the same category. Fine-tuning the detector for each subcategory or using more complex models might be necessary to handle this variation effectively.

Discussion

Iterations

As part of the research, we went through a couple of notable versions of the cascade classifier. This part describes the differences and why we changed something.

For every iteration we used the same dataset of negative images, the ratio for all versions is 1 positive to 5 negative images.

Iteration 0 Zoomed out faces

This version was trained on 150x150 size faces of celebrities. The positive amount om this dataset was 100.

When we tried this version, it did not detect any faces on the example images we tried.

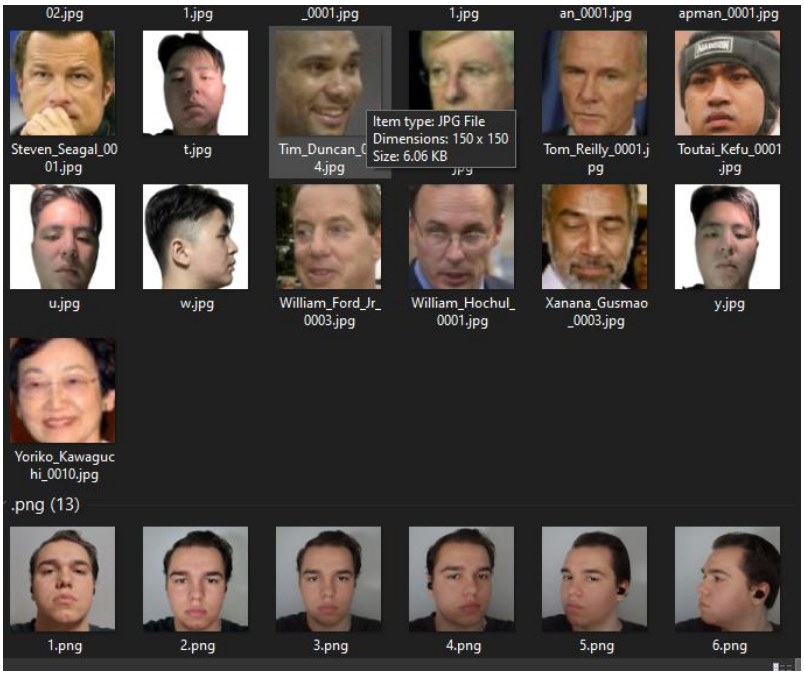
Iteration 1 zoomed-in

This version was trained with 100 pictures of 150x150 size, zoomed in on the faces of the celebrities.

This version was able to detect the “perfect” scenario faces in the examples. But struggled when the faces were tilted, less bright or angled. It also struggles when we are too close or too far away from the camera.

Iteration 2 Zoomed-in plus our faces

This version was trained with 200 pictures of 150x150 size. 30 of those are our own faces and the other 130 are faces from celebrities that are zoomed in on their faces.



This version worked much better when we stood far away from the camera and with different lighting conditions but really struggles with differences in angles.

Iteration 3 Zoomed-in plus our faces plus cherry picked for diverse angles

The final version we created had 30 pictures of us, 70 faces chosen at random from the zoomed in faces and 100 that were cheery picked. We selected specifically faces that were angled or covered in some way.
This version was much better at handling different angles but lost about 1 meter of range.

Face Detection

Problem 1: Minimum size

From the demo we saw that the minimum size is about “” pixels. This corresponds to about being 5-6 meters

Problem 2: Maximum size

In the demo we show that the maximum size is about 10 cm away from the camera.

Problem 3: Detection time

In the demo we show that the model is able to detect faces in real time and also detect multiple faces in real time.

Conclusion

In this report we showed the beginning to the creation of an acceptable model that can detect faces. We also showed why the model fails in certain challenges and show how it compares to the well-trained version from OpenCV themselves.

Room for improvement

To increase the accuracy of our model there are a couple of things we could do.

The first is to add more images with differing angles, this will ensure that the model is better able to handle angled and tilted faces.

The second is to add more images with different lighting conditions, this will better the ability to detect faces when only half the face is lit up or covered in shadows.

The third is to add more images of people from different races and gender, our dataset consisted of mostly white men. To ensure the model can handle woman and people from different races more diverse faces need to be used.

The fourth one is to add more positive images, this is done automatically when implementing the other 3.

When implementing the first 3 solutions, they should be intermixed as well. So, there should be more examples of for example: An Asian woman looking at an angle whilst only half her face is lit up.

References

Dataset of places without humans used as positives for training Haar cascade model
https://github.com/handaga/tutorial-haartraining/tree/master/data/negatives

Dataset of human faces used as positives for training Haar cascade model
https://www.kaggle.com/datasets/quadeer15sh/lfw-facial-recognition

Reference Haar Cascade model which was used for comparison
https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml.

Contribution

Both:

We both worked on finding the datasets that we used to train the model. We both filmed the and edited the demo.

Jelle:

I worked on creating the python project that we used for the demo and report. I created the snippets that can be seen in the report. Created/ ran the cascade GUI to create the cascade.xml files (Because Yevgeniy's laptop could not run the program). I put together the datasets from the one that we found. I edited the pictures of us to conform to the other ones. I wrote the conclusion and iteration part of this document.

Yevgeniy:

Since I had some experience working with image annotations in CVLab I gave advice on how model can behave and tried to correct minor bugs and missing parts while configuring cascade training GUI. I watched videos regarding training cascade and shared learned with Jelle to make our project more effective. I started writing report by giving it more academic structure with relevant chapters.