Face detection

Sasu-Simon Kinga

30434

Technical University of Cluj-Napoca

Table of Contents

Introduction

Context

The goal of this project is to design, implement and test face detection, more precisely a mouth, nose, and eyes detection program. As we all know face recognition is an exceedingly popular topic, with a lot of use cases in everyday lives.

Face detection is a computer technology that determines the locations and sizes of human faces in arbitrary (digital) images. It detects facial features such as eyes, eyebrows, nose, mouth, and jawline, and ignores anything else, such as buildings, trees, or bodies. Face detection can be regarded as a more general case of face localization. In face localization, the task is to find the locations and sizes of a known number of faces, usually one.

This application can be used by people who work in the field of face analysis. This program can help them to identify which parts of an image or video should be focused on to determine age, gender, and emotions using facial expressions. Face detection can also be used to auto-focus cameras or to count how many people have entered an area.

Specifications

Face detection applications use algorithms and machine learning to find human faces within larger images, which often incorporate other non-face objects such as landscapes, buildings and other human body parts like feet or hands. Face detection algorithms typically start by searching for human eyes (one of the easiest features to detect). The algorithm might then attempt to detect eyebrows, the mouth, nose, and the iris. Once the algorithm concludes that it has found a facial region, it applies additional tests to confirm that it has, in fact, detected a face. To help ensure accuracy, the algorithms need to be trained on large data sets incorporating hundreds of thousands of positive and negative images.

Detecting faces in pictures can be complicated due to the variability of factors such as pose, expression, position and orientation, skin color and pixel values, the presence of glasses or facial hair, and differences in camera gain, lighting conditions and image resolution. Recent years have brought advances in face detection using deep learning, which presents the advantage of significantly outperforming traditional computer vision methods.

As programming language for this project, I used python and one of its libraries, OpenCV, which is the most popular library for computer vision.

Objectives

The main objective of facial detection is to identify individuals, whether individually or collectively. The problem of face detection has been studied extensively. A wide spectrum of techniques have been used including color analysis, template matching, neural networks, support vector machines (SVM), maximal rejection classification and model-based detection. However, it is difficult to design algorithms that work for all illuminations, face colors, sizes and geometries, and image backgrounds. So, the objective of this project is to create

an application, which precisely detects the eyes, nose, and mouth. The algorithm will be a combination of various proposed methods along with some other features.

Bibliographic study

Facial detection is the process of identifying and locating human faces in digital images or video streams. One popular method for facial detection is the 68 landmark detection method, which involves detecting and annotating specific points of interest on the face, such as the corners of the mouth and the centers of the eyes. These landmarks are typically represented by a set of 68 (x, y) coordinates on the face. Dlib is a commonly used open-source library that can recognize these landmarks in an image. However, this system of 68 facial landmarks is limited for many purposes. Newer datasets and algorithms leverage a dense “face mesh” with over 468 3D face landmarks. This approach uses machine learning to infer the 3D facial surface from a single camera input, without a dedicated depth sensor. Google’s MediaPipe solution uses dense facial landmarks.

The 68 landmark detection method was introduced in the 2014 paper "One Millisecond Face Alignment with an Ensemble of Regression Trees". This method has become widely used in a variety of applications, including facial recognition, emotion detection, and head pose estimation. The indexes of the 68 coordinates can be visualized in *Figure1*.

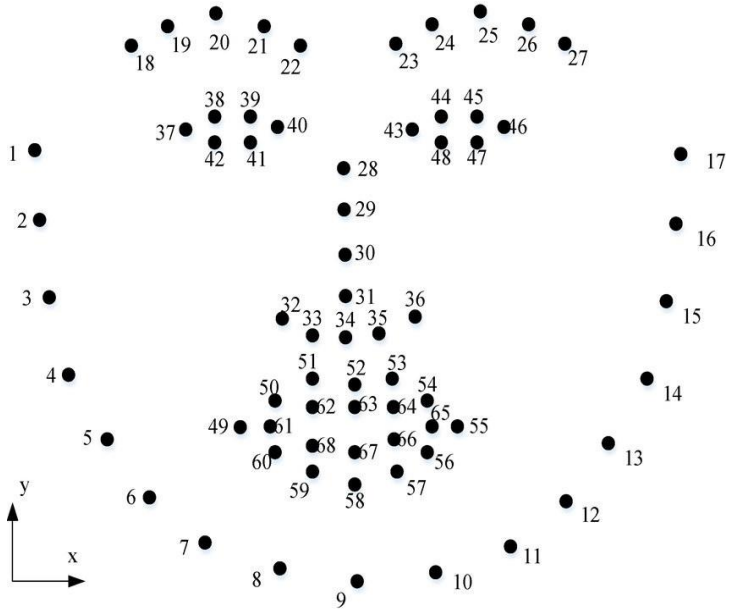


Figure 1: 68 facial landmarks

These annotations are part of the 68-point dataset on which the dlib facial landmark predictor was trained. It’s important to note that other flavors of facial landmark detectors exist, including the 194-point model that can be trained on the HELEN dataset. Regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data.

The landmark detection is carried out in two steps:

1. **Face detection.** Face detection locates a human face from the image and return a value in terms of the coordinates of the bounding rectangle of the detected face.
2. **Landmarks detection.** After detecting the face, we went through points inside the bounding rectangle for key points.

Depending on the facial appearance and facial shape patterns, Facial landmark detection algorithms are classified into three. Here, the facial appearance refers to the distinctive pixel intensity patterns around the facial landmarks or in the whole face region, while face shape patterns refer to the patterns of the face shapes as defined by the landmark locations and their spatial relationships.

The three major categories of landmark detection algorithms are:

- Holistic methods
- Constrained Local Model (CLM) methods
- Regression-based methods

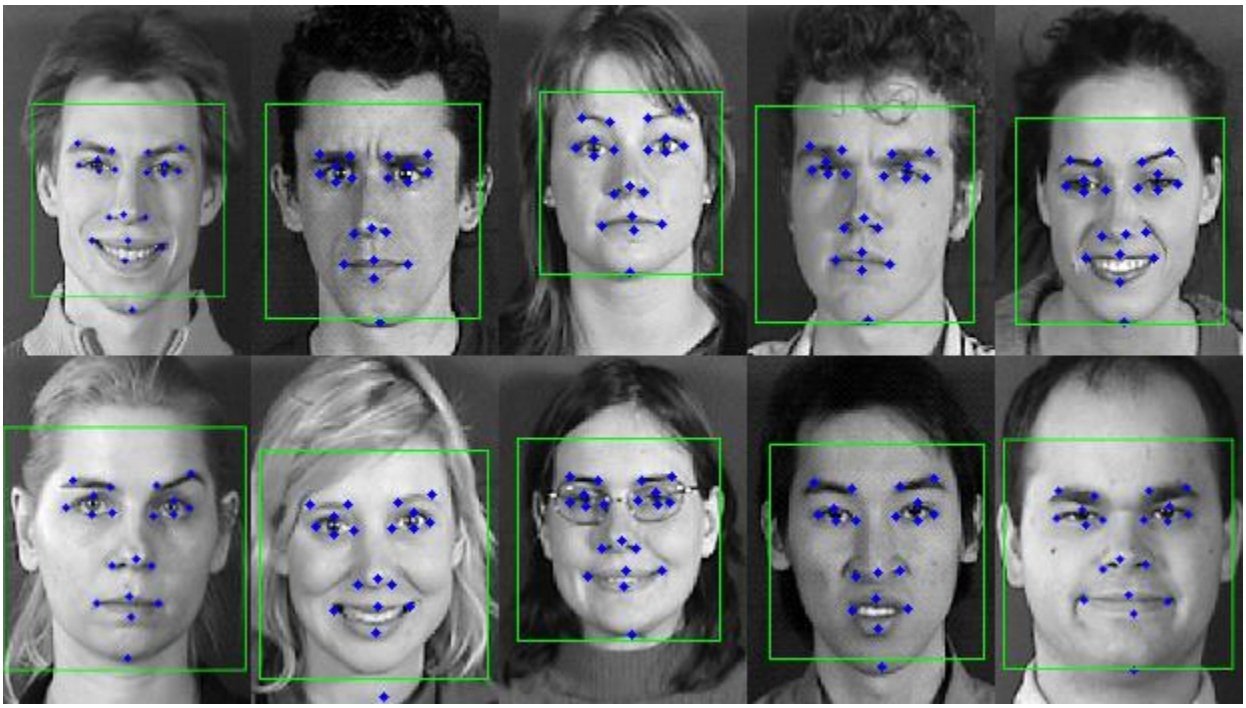
The holistic methods explicitly model the holistic facial appearance and global facial shape patterns.

CLMs rely on explicit local facial appearance and explicit global facial shape patterns.

The regression-based methods use holistic or local appearance information, and they may embed the global facial shape patterns implicitly for joint landmark detection.

Some recent methods combine deep learning models and global 3D shape models for landmark detection.

In the image below, we can notice that the program identifies each face and the primary landmarks, which include the eyebrows, eyes, nose, mouth, and jaw.



Dlib model

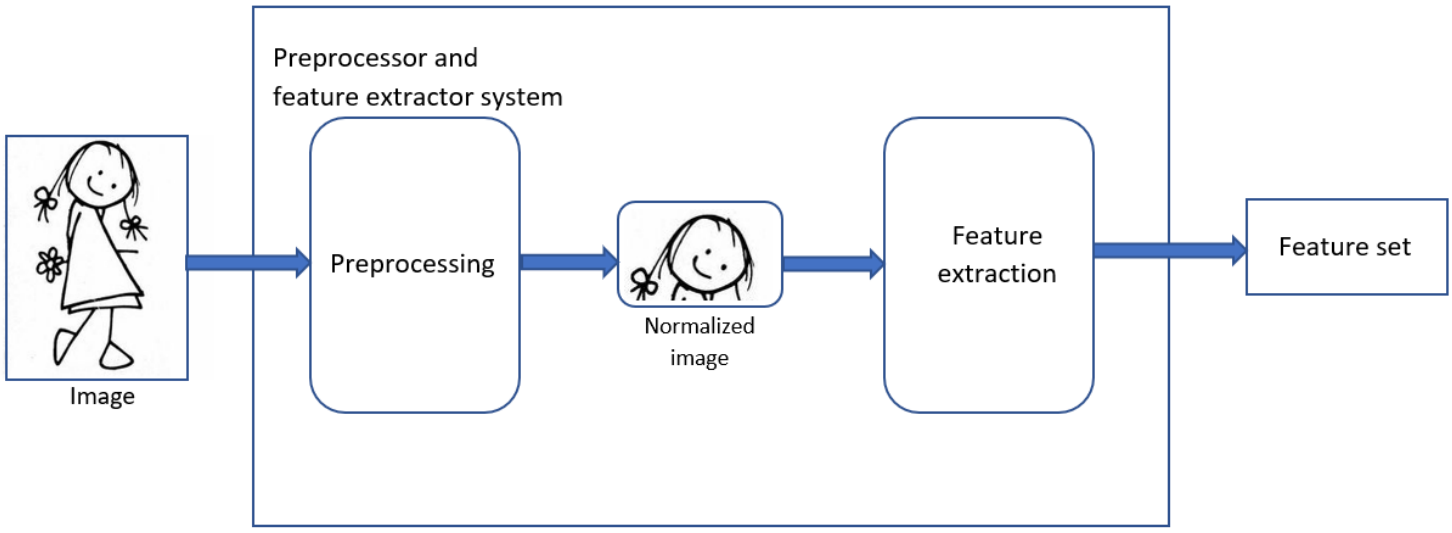
Dlib library includes face detection and landmark detection functions in it. Dlib face detection uses histogram-oriented methods (HOG), and landmark detection is based on Kazemi's model. It returns different 68 feature points from a face. Dlib includes a pre-built model for face landmark detection called *shape_predictor_68_facemarks.dat*. We are just accessing directly that model and creating an object.

The location of the facial parts are:

- Eyebrows: [18, 27]
- Eyes: [37, 48]
- Nose: [28, 36]
- Mouth: [49, 68]
- Jawline: [1, 17]
-

Design

A thorough survey has revealed that various methods and combinations of these methods can be applied in the development of a new face detection system. Among the many possible approaches, we have decided to use a combination of knowledge-based methods for the face detection part and a neural network approach for the face detection part. The main reason for this selection is their smooth applicability and reliability issues. Our face detection system approach is given in *Figure 3*.



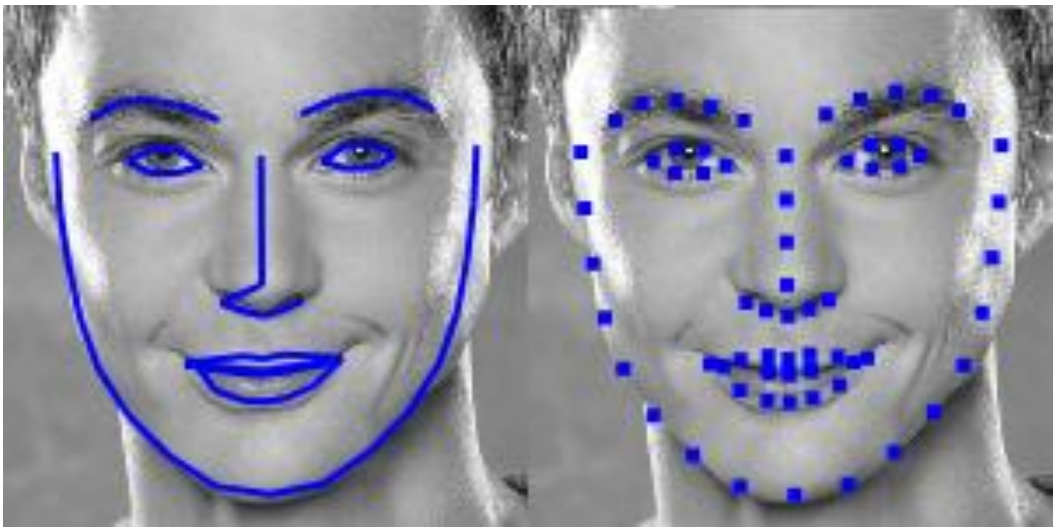
Input Part

The input part is a prerequisite for the face detection system. Image acquisition operation is performed in this part. Live captured images are converted to digital data for performing image-processing computations. These captured images are sent to a face detection algorithm. There are very few instructions on the composition of the set of input images, but one very important restriction is that each image in the set holds information about only one individual, and this individual is the same for all images. From here on, we will refer to this set as the 'input set' and the imaged individual as the 'subject'. On each run of the system, only one input set is presented.

Dlib

Dlib also features robust tools for object pose estimation, object tracking, face detection (classifying a perceived object as a face), and face recognition (identifying a perceived face).

Though Dlib is a cross-platform resource, many custom workflows involving facial capture and analysis (whether recognition or detection) use the OpenCV library of functions, operating in a Python environment, as in the image below.



Dlib offers two different functions for facial capture:

- **HoG + Linear SVM**

The Histogram of Oriented Gradients (HoG) + Linear Support Vector Machine (SVM) algorithm in Dlib offers very fast recognition of front-on faces but has limited capabilities in terms of recognizing face poses at acute angles (such as CCTV footage, or casual surveillance environments where the subject is not actively participating in the ID process).

It also supports passport-style profile faces, though with very little margin for error (faces pointing up or down, etc.). HoG + SVM is suitable for constrained situations

where the sensor can expect a direct and unobstructed view of the participant's face, such as ATM and mobile framework ID systems, as well as mobile traffic surveillance recognition systems, where cameras are able to obtain a straight profile shot of drivers.

Also, I used this method to implement my face detection application.

- **Max-Margin (MMOD) CNN face detector**

MMOD is a robust and reliable, GPU-accelerated face detector that leverages a convolutional neural network (CNN), and is far more capable of capturing faces at obscure angles and in challenging conditions, suiting it for casual surveillance and urban analysis.

MMOD is not a distinct alternative to HoG + Linear SVM, but rather can be applied to HoG itself, or to any bag-of-visual-word model, which treats discovered pixel groupings as explorable entities for potential labeling — including the identification of faces.

Implementation

Steps of the implementation:

1. In this first step, we are reading the input image using OpenCV.

```
img = cv2.imread(imagePath)
imageRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

2. To detect all the faces in an image I used the library's "shape_predictor_68_face_landmarks.dat" file, to predict the faces. For the prediction phase to take place, the frontal face detector method will be used to separate the faces from the rest of the image, so the detection is only focused on the faces, not on the entire picture. The shape predictor class will take the model (frontal face) and with the help of that, it will show all the faces that appear in the image.

```
faceLandmarkDetector = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
frontalFaceDetector = dlib.get_frontal_face_detector()
```

```
allFaces = frontalFaceDetector(imageRGB, 0)
allFacesLandmark = []
```

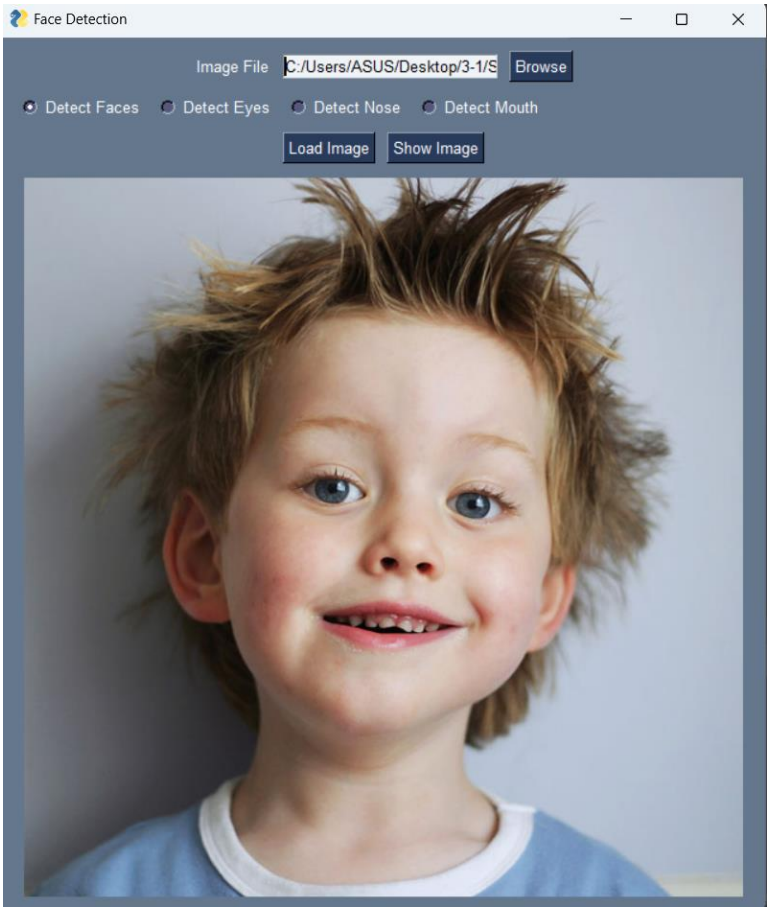
3. We already have the faces detected, now we want to detect the eyes, nose, and mouth on each face. To do this, we will iterate through each face and look for each feature, based on the 68 landmarks mentioned above. After the features are detected, we highlight them in the image.

```python
for face in allFaces:
    x1, y1 = face.left(), face.top()
    x2, y2 = face.right(), face.bottom()

    img = cv2.rectangle(img, (x1, y1), (x2, y2), (255, 255, 68), 1)
    faceRectangleDlib = dlib.rectangle(int(x1), int(y1), int(x2), int(y2))

    detectedLandmarks = faceLandmarkDetector(imageRGB, faceRectangleDlib)
    allFacesLandmark.append(detectedLandmarks)
```

4. To draw the outlines of each feature, we implemented a function connecting each landmark point for a given feature.

```python
def drawPoints(image, faceLandmarks, startpoint, endpoint, isClosed=False):
    points = []
    for i in range(startpoint, endpoint + 1):
        point = [faceLandmarks.part(i).x, faceLandmarks.part(i).y]
        points.append(point)

    points = np.array(points, dtype=np.int32)
    cv2.polylines(image, [points], isClosed, (253, 184, 55), thickness=2, lineType=cv2.LINE_8)
```

5. Lastly, we created an easy-to-use graphical user interface. The user can choose an image, upload it and choose the features that he or she wishes to detect.

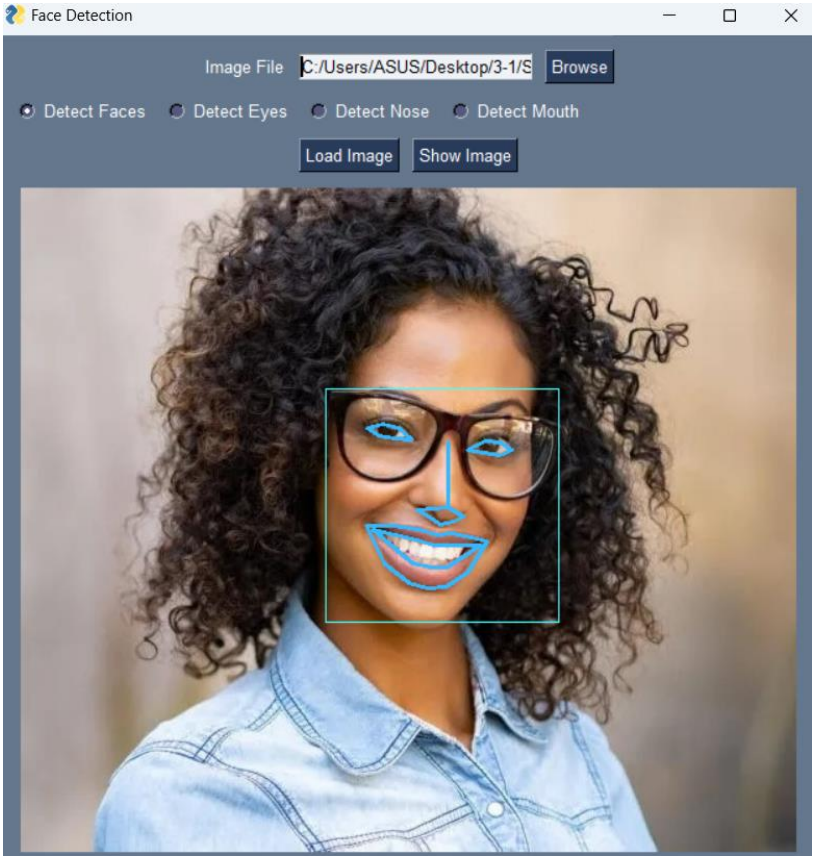


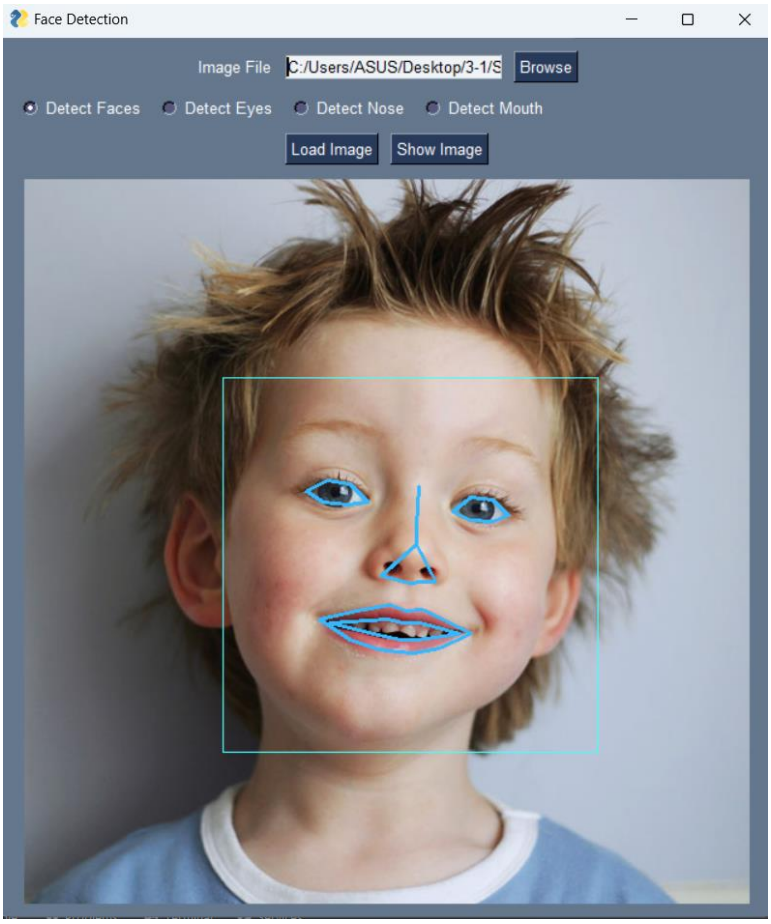
Testing

To test the application, I downloaded a dataset of images. Then tested the application with images with one face and with group photos.

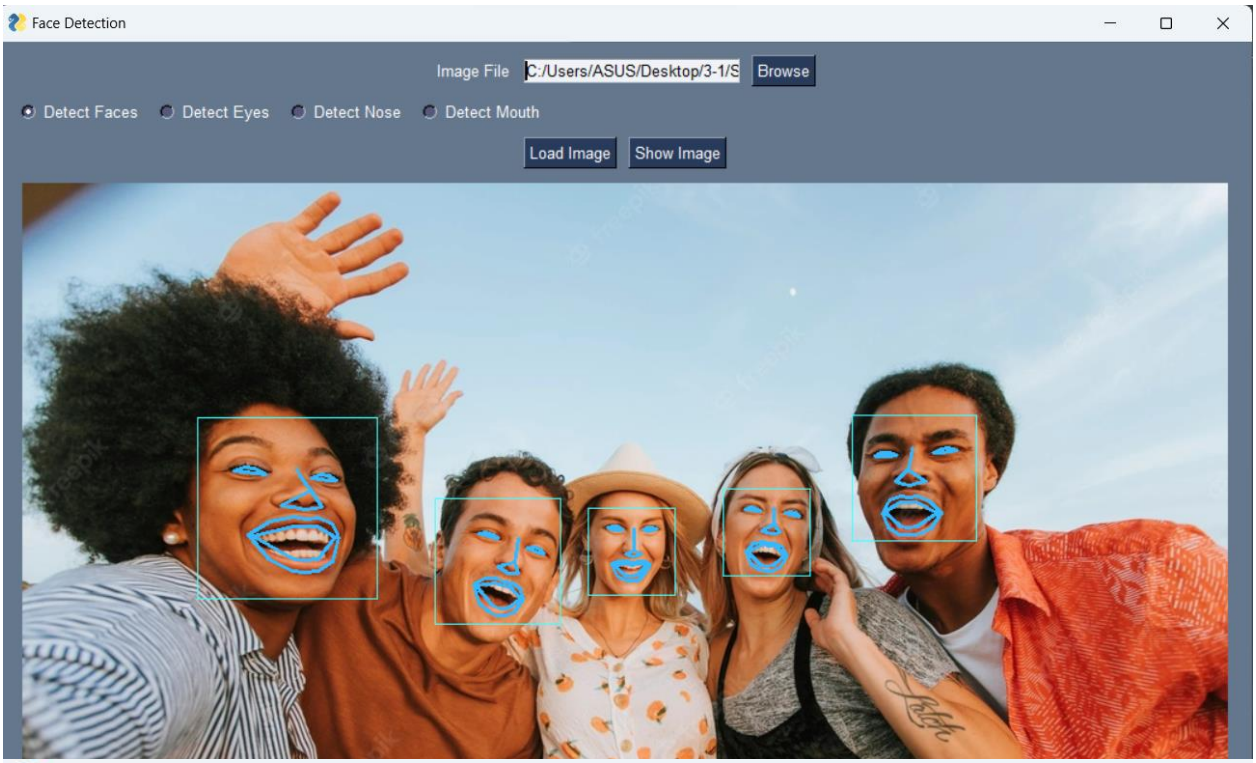
First, the detection of the face, with all three features (eyes, nose, and mouth):

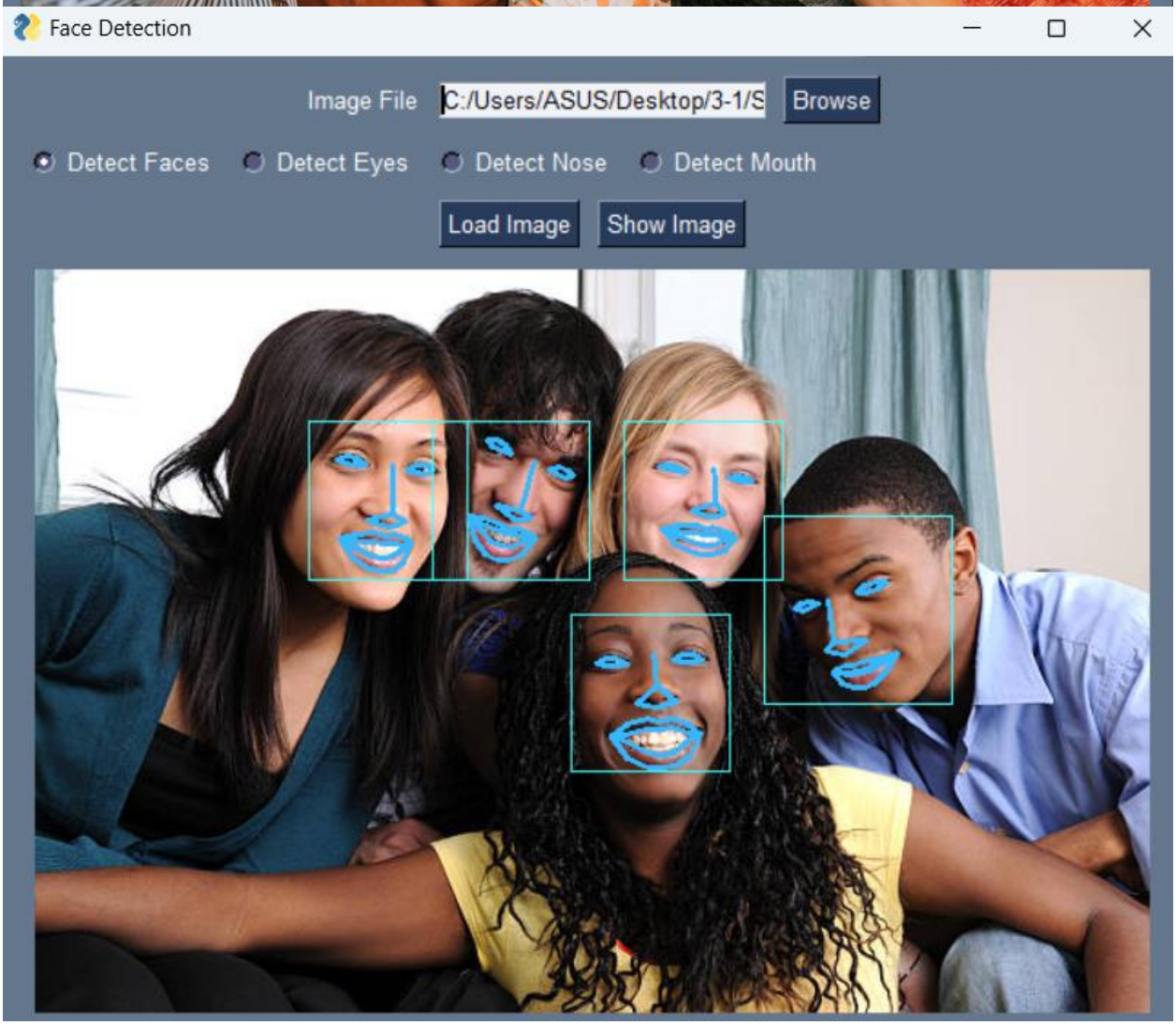
- Portrait





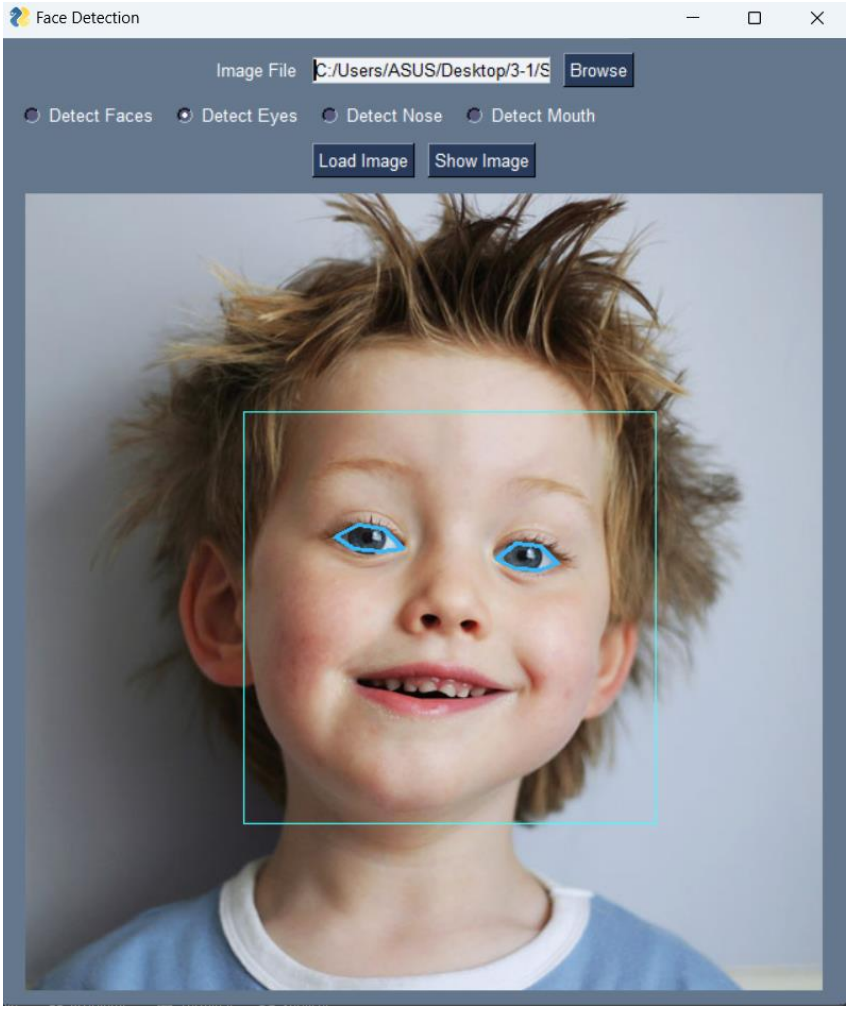
- Group photo

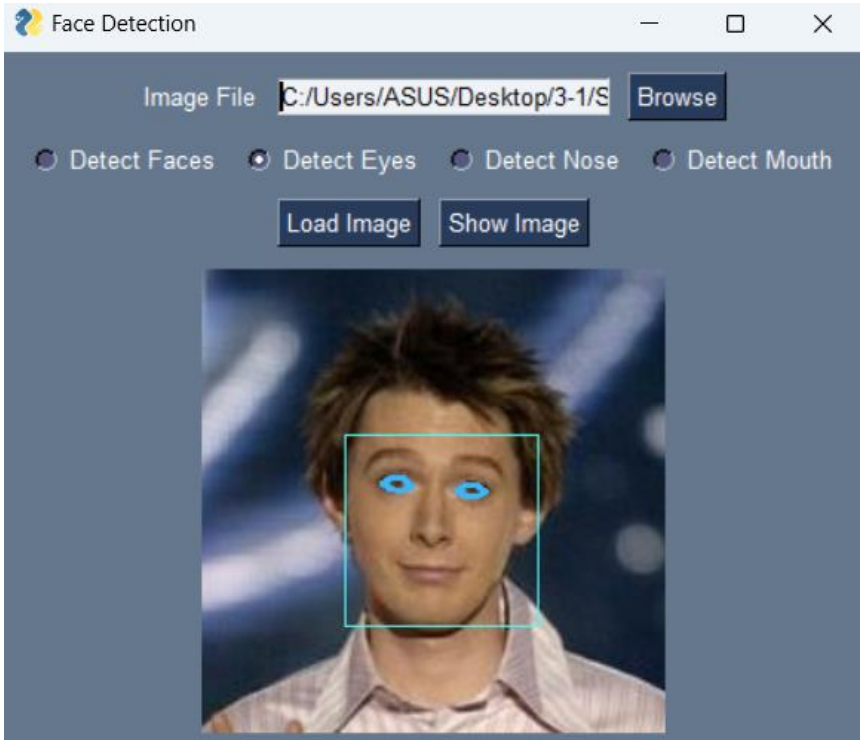

Face Detection
Image File
C:/Users/ASUS/Desktop/3-1/S
Browse
Detect Faces
Detect Eyes
Detect Nose
Detect Mouth
Load Image
Show Image


Face Detection
Image File
C:/Users/ASUS/Desktop/3-1/S
Browse
Detect Faces
Detect Eyes
Detect Nose
Detect Mouth
Load Image
Show Image

Detection of the eyes:

- Portrait

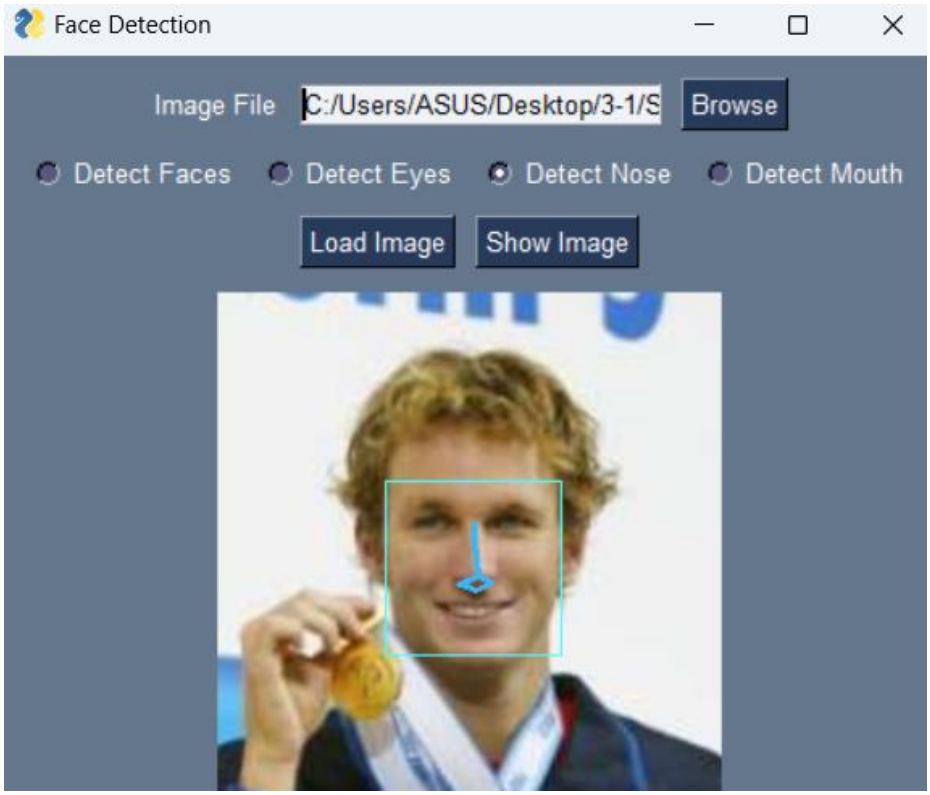




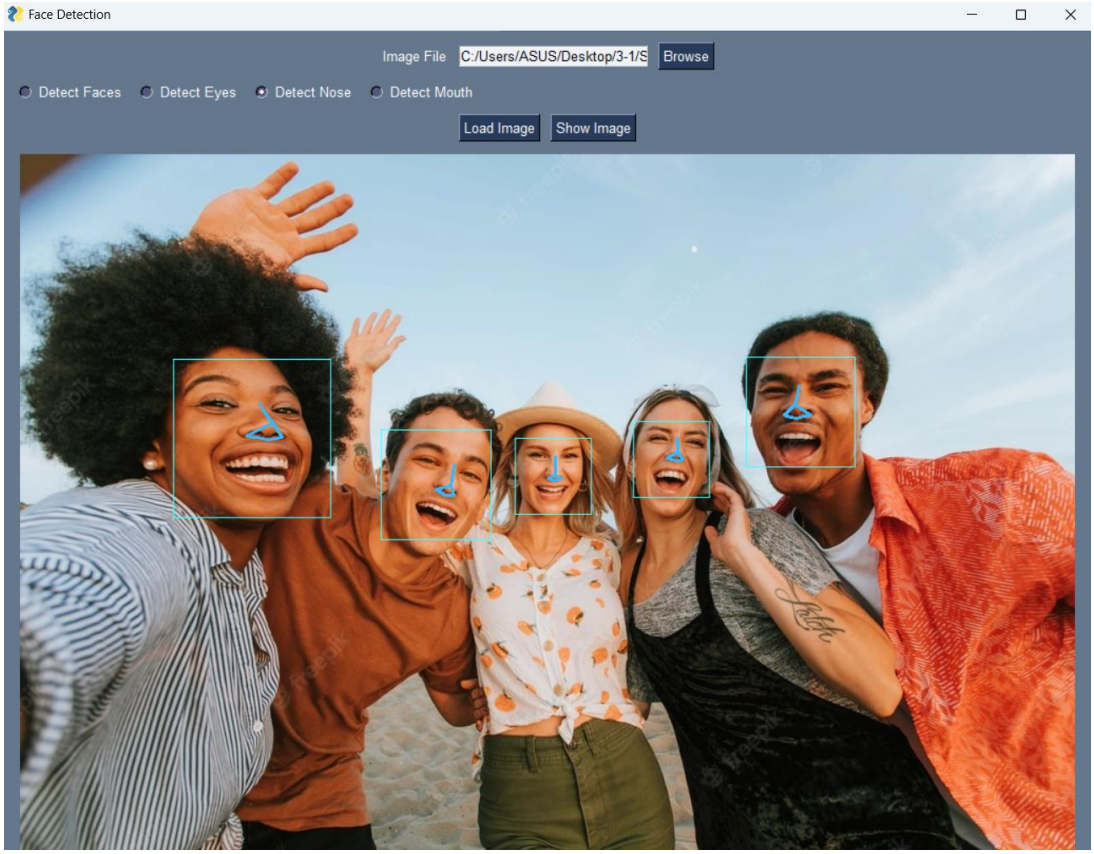
- Group photo

Detection of the nose:

- Portrait

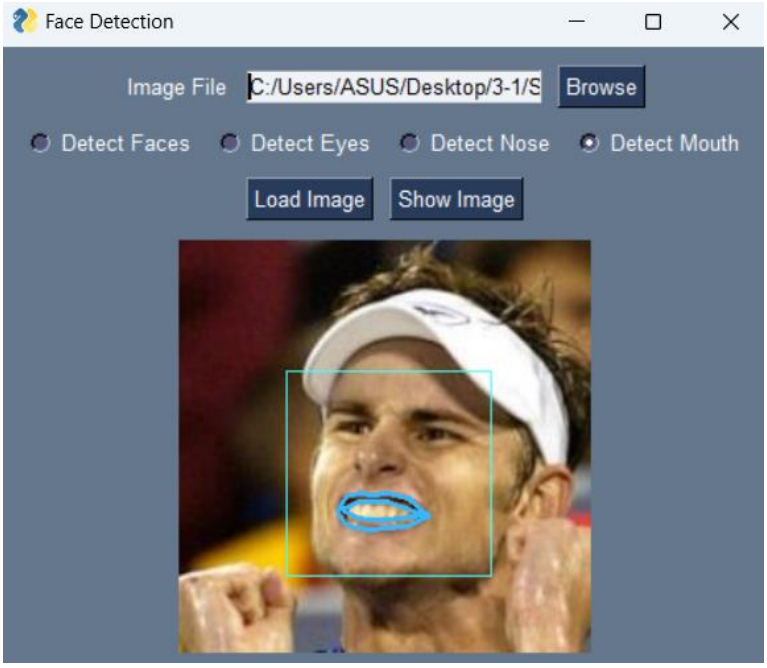


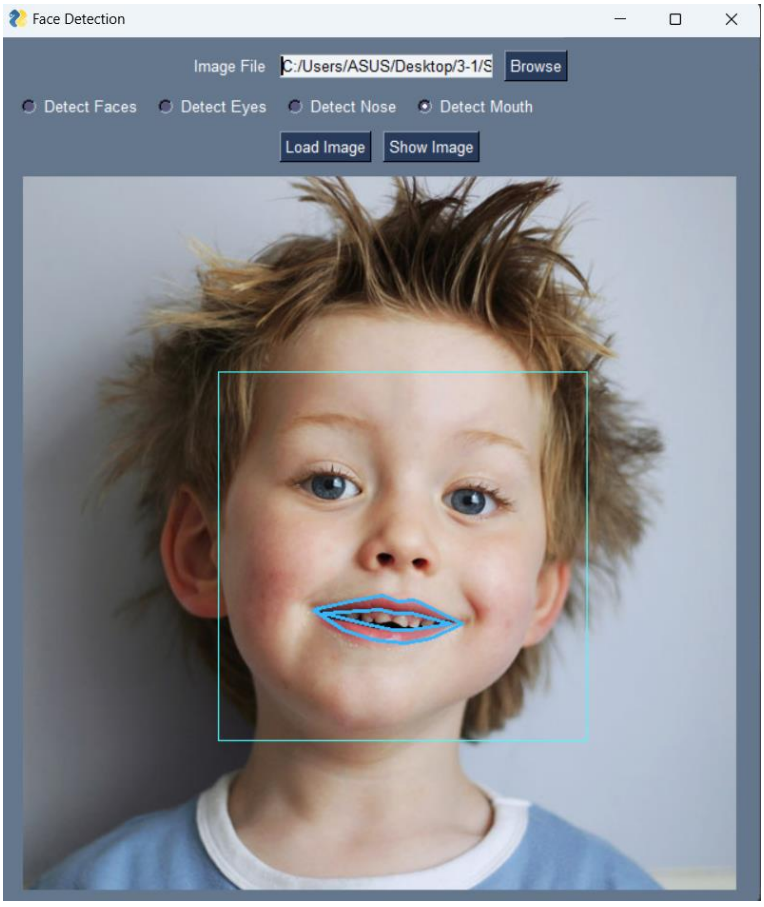
- Group photo



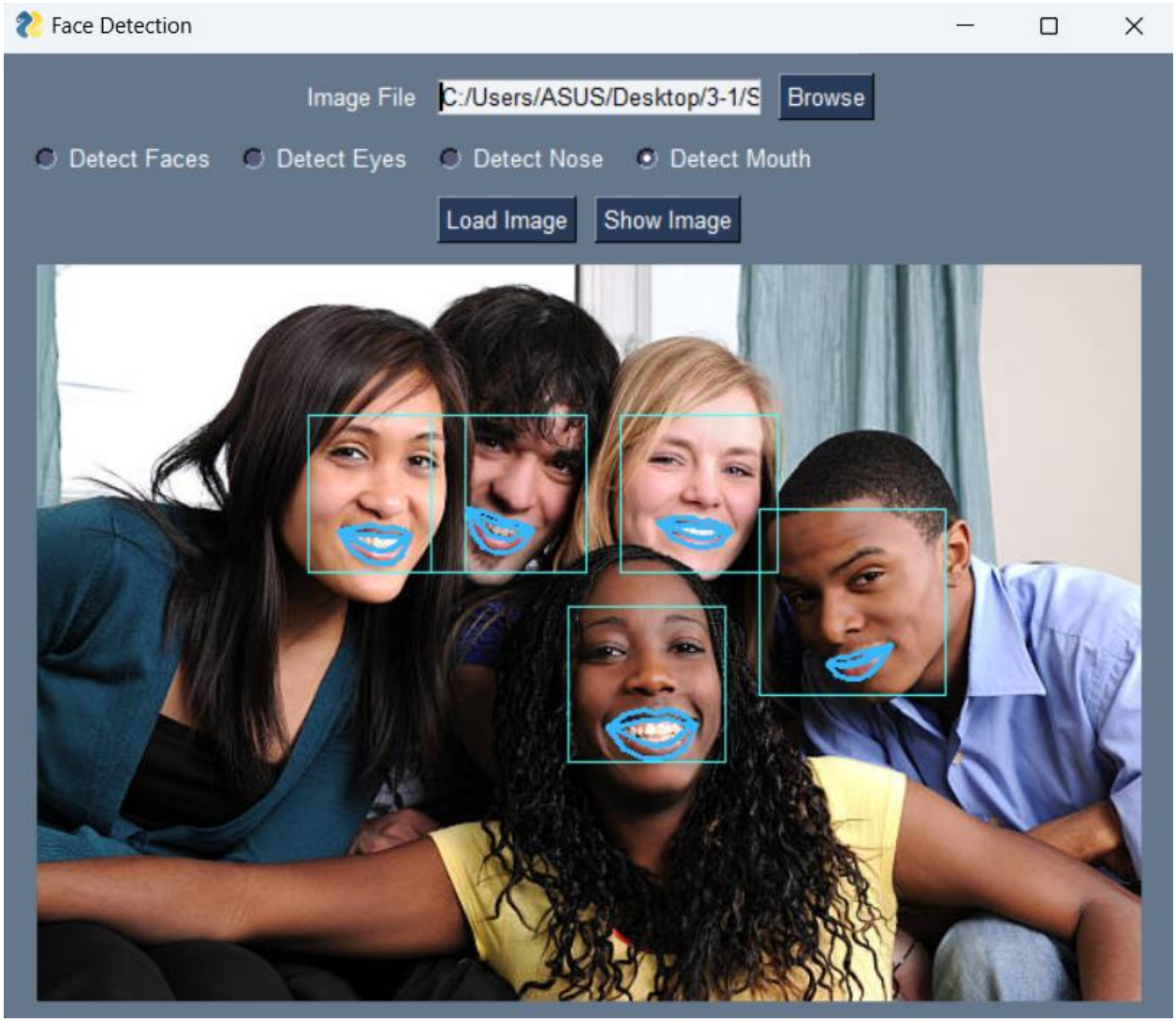
Detection of the mouth:

- Portrait





- Group photo



Conclusion

Face detection is a crucial aspect of computer vision and has numerous applications in various fields such as security, entertainment, and robotics. Dlib is a popular open-source library that provides state-of-the-art algorithms for facial detection and recognition.

The dlib library uses a combination of HOG (Histogram of Oriented Gradients) features and a linear SVM (Support Vector Machine) classifier to detect faces in images. The HOG features capture important information about the shape and appearance of the face, while the SVM classifier helps to distinguish faces from non-face regions.

One of the advantages of using dlib for face detection is its high accuracy and speed. It can detect faces in real time, even in crowded scenes with multiple faces. Dlib also has a small memory footprint, making it suitable for use in embedded systems and mobile devices. There are, however, a few limitations of dlib's face detection algorithm. It may struggle to detect faces in images with low resolution or low lighting conditions. It may also have difficulty detecting faces with extreme poses or facial expressions. To address these limitations, researchers have proposed various improvements to the dlib face detection algorithm. One approach is to use deep learning-based methods such as Convolutional Neural Networks (CNNs) to improve the accuracy of face detection. CNNs can learn complex features from the data, allowing them to better handle variations in lighting, pose, and facial expressions.

Further improvement ideas:

- Use a larger and more diverse dataset for training the face detection model. This can help the model generalize better and improve its accuracy on real-world images.
- Fine-tune the model on a specific task or dataset. For example, if you are working on a face detection project for a particular industry or domain, you can fine-tune the model on a dataset that is relevant to that domain.
- Use data augmentation techniques to increase the size and diversity of the training dataset. This can help the model generalize better and improve its robustness to variations in lighting, pose, and other factors.
- Experiment with different pre-processing techniques, such as histogram equalization or image resizing, to see if they improve the performance of the face detection model.
- Consider integrating other face analysis tasks, such as facial recognition or emotion detection, into your project to make it more comprehensive and useful.

In conclusion, dlib is a powerful tool for face detection that offers high accuracy and speed. However, there is still room for improvement, particularly in challenging scenarios such as low resolution and low lighting conditions. The use of deep learning and ensemble learning techniques can help to address these limitations and further improve the performance of dlib's face detection algorithm.

Bibliography

- https://www.techtarget.com/searchenterpriseai/definition/face-detection#:~:text=In%20face%20analysis%2C%20face%20detection,and%20emotions%20using%20facial%20expressions.
- https://mittalparveen652.medium.com/face-eye-and-mouth-detection-with-python-458f2a028674
- https://face-recognition.readthedocs.io/en/latest/readme.html
- https://towardsdatascience.com/step-by-step-face-recognition-code-implementation-from-scratch-in-python-cc95fa041120