**GlobencyMedia**

FACE DETECTION

2023

By

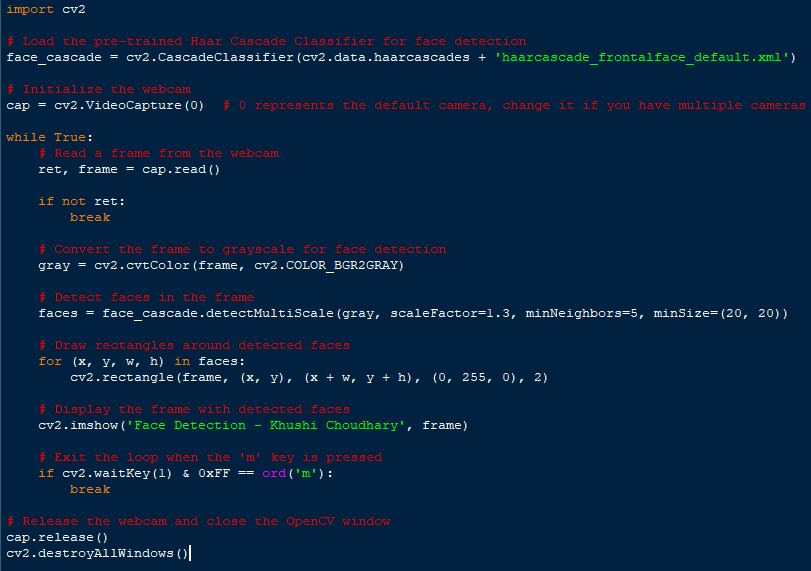
TeamGlobencyMedia

**Abstract**

This report provides an overview of the development and implementation of a face detection model using the OpenCV library in Python. Face detection is a crucial task in computer vision and has applications in various fields, including security, surveillance, attendance, and human-computer interaction. OpenCV is a popular open-source computer vision library that provides powerful tools for face detection and recognition.

**Face Detection**

Face detection is the process of identifying and locating human faces within images or video frames. It is a fundamental step in various applications, such as facial recognition, emotion analysis, and object tracking. OpenCV is an open-source computer vision library, offers a range of pre-trained models and functions for face detection.



* **Import cv2**

We need to install OpenCV from cmd using a command

**“pip install opencv”**

Importing OpenCV library in the shell.

* **Load the pre-trained Haar Cascade Classifier for face detection**

Created a variable ‘**face\_cascade**’ and used a module **CascadeClassifier** with the parameters cv2.data.haarcascades and haarcascade\_frontalface\_default.xml, where 'haarcascade\_frontalface\_default.xml' is a dataset in OpenCV library. Here we have trained our model with Haar Cascade Classifier module.

* **Initialize the webcam**

Inside a loop there are two conditions if the condition is true then capture whatever is in front of camera else break.

Created a variable ‘**cap**’ and used a module **VideoCapture** with a parameter 0, 0 represents the default camera, change it if you have multiple cameras.

* **Read a frame from the webcam**

Defining two different variables ‘ret’ and ‘frame’ and reading it from VideoCapture.

* **Convert the frame to grayscale for face detection**

Creating a variable named ‘gray’ and using a module cvtColor and using parameters frame and cv2.COLOR\_BGR2GRAY. OpenCV can work efficiently with gray and black colors.

* **Detect faces in the frame**

Creating a variable named ‘faces’, from module face\_cascade we have used a function detectMultiScale which has four parameters gray, scaleFactor, minNeighbor and minSize.

Scalefactor can range from 1 to 1.5, minNeighbor is the difference between the pixcels of two faces and minSize is the minimum size of faces.

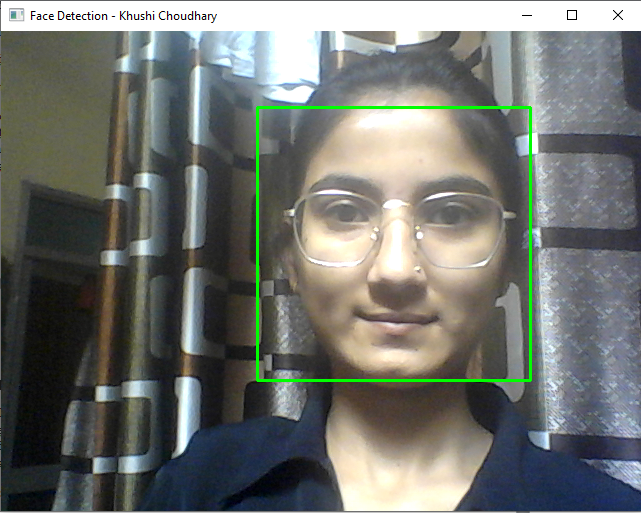
* **Draw rectangles around detected faces**

Inside a for loop we have defined four different variables x, y, w and h in faces variable, where x is for x axis, y is for y axis, w is for width and h is for height.

Cv2 has a module named rectangle which has some parameters including frame and 4 different arguments for rectangle’s measurement and color, here (x, y) is for drawing a line on x axis as well as for y axis, (x+w,y+h) is for attaching a line which is widthwise attached to the x axis line and another one is for attaching the height line with y axis line, (0, 255, 0) is for the color of rectangle which is here green and 2 is for the thickness of green rectangle.

* **Display the frame with detected faces**

Using a module imshow() from cv2 library face detection model will reflect a green rectangle around the faces.

**Output of Face Detection Model**

* **Exit the loop when the 'm' key is pressed**

After all the process and using this model the final task is to end/terminate this loop. Using waitKey module from cv2 library with an argument of 1 which is the time duration of a millisecond to pause the loop. If 0xFF 8 bit ASCII code would match the ASCII code of ‘m’, the loop would terminate automatically on a single press and the application would exit.

* **Release the webcam and close the OpenCV window**

Using cap.release() the program would end and with cv2.destroyAllWindows() new window opened for web cam will be closed.

**Methodology**

The face detection model was created using the following steps:

1. **Environment Setup**

We set up the Python environment and installed the OpenCV library, which is essential for this project.

1. **Data Collection**

We obtained images and videos to test the face detection model.

1. **Haar Cascade Classifier**

OpenCV provides pre-trained Haar Cascade classifiers for face detection. We loaded the face detection classifier from OpenCV's repository.

1. **Face Detection**

The Haar Cascade classifier was applied to the input images and video frames using OpenCV's functions. This process involved the following steps:

- Converting the image to grayscale for efficient processing.

- Running the classifier to detect faces.

- Drawing bounding boxes around detected faces.

1. **Testing and Evaluation**

We tested the model with various images and videos in CV2 library, evaluating its accuracy and performance.

**Results**

The face detection model using OpenCV exhibited robust performance, successfully identifying and locating faces in the tested images and video frames. The accuracy of the model depended on factors such as lighting conditions, face angles, and image quality. Overall, it provided a reliable solution for face detection tasks.

**Conclusion**

The use of the OpenCV library to create a face detection model in Python proved to be a successful approach. This model can serve as the foundation for more advanced applications, including facial recognition, emotion analysis, and human-computer interaction. While this report focused on the basic implementation of face detection, future work could involve fine-tuning the model for specific use cases and optimizing its performance under various conditions.