Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 018
Code Title:	Emerging Technologies in CpE 1 - Fundamentals of Computer Vision
1st Semester	AY 2023-2024

ACTIVITY 6	**Face Detection using OpenCV**
Name	Marquez, Keith Leigh Zhen R.
Section	CPE32S3
Date Performed:	02/19/2025
Date Submitted:	02/21/2025
Instructor:	Engr. Roman M. Richard

1. Objectives

This activity aims to allow students to perform face detection on still images and videos using Haar cascades.

2. Intended Learning Outcomes (ILOs)

After this activity, the students should be able to:

- Utilize OpenCV to detect faces in still images and videos.
- Demonstrate the use of Haar-like features for detection of other human features.

3. Procedures and Outputs

Contrary to initial assumptions, conducting face detection on a static image and a video stream shares a remarkable similarity. Essentially, the latter is merely a sequential rendition of the former: when detecting faces in videos, it essentially involves applying face detection to every individual frame obtained from the camera feed. Of course, video face detection introduces additional elements like tracking, which aren't relevant to static images. Nevertheless, it's valuable to recognize that the fundamental principles behind both processes remain consistent.

Performing face detection on still image

The first and most basic way to perform face detection is to load an image and detect faces in it. To make the result visually meaningful, we will draw rectangles around faces on the original image.

Before implementing the code below, check the contents of the `cv2.CascadeClassifier()` function. Provide an explanation of the function, its parameters before running the code below.

```
# Make sure that for this activity, you have downloaded the
# file indicated below from the resource linked in the instructional
materials
```

```
# in the module.

import cv2

picPath = r'C:\Users\Keith\Documents\Activity 6. Face Detection using
OpenCV\breaking_bad.png'
haarPath = r'C:\Users\Keith\Documents\Activity 6. Face Detection using
OpenCV\haarcascade_frontalface_default.xml'

def faceDetect(picPath):
  face_cascade = cv2.CascadeClassifier(haarPath)

  img = cv2.imread(picPath)
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  faces = face_cascade.detectMultiScale(gray, 1.3, 5)

  for (x, y, w, h) in faces:
    img = cv2.rectangle(img, (x, y), (x+w, y+h), (255,0,0), 2)

  cv2.namedWindow('my image', cv2.WINDOW_NORMAL)
  cv2.imshow('my image', img)
  cv2.resizeWindow('my image', 700, 600)
  cv2.waitKey()
  cv2.destroyAllWindows()

faceDetect(picPath)
```

Analysis:

- Based on your earlier analysis, where do you think the face detection works in the line of code above?
- Provide an analysis of the parameters of the `detectMultiScale` method.
- Change the color of the border of the detected faces to red.
- Are you able to make the borders thicker? Demonstrate.

Performing face detection on video

Step 1: Create a file called face_detection.py and include the following codes.

```
import cv2
```

Step 2: After this, we declare a method, `detect()`, which will perform face detection.

```
def detect():
  face_cascade =
cv2.CascadeClassifier('/content/haarcascade_frontalface_default.xml')
  eye_cascade = cv2.CascadeClassifier('/content/haarcascade_eye.xml')
  camera = cv2.VideoCapture(0)
```

Step 3: The first thing we need to do inside the detect() method is to load the Haar cascade files so that OpenCV can operate face detection. As we copied the cascade files in the local `cascades/` folder, we can use a relative path. Then, we open a VideoCapture object (the camera feed). The VideoCapture constructor takes a parameter, which indicates the camera to be used; zero indicates the first camera available.

```
  while (True):
    ret, frame = camera.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Step 4: Next up, we capture a frame. The read() method returns two values: a Boolean indicating the success of the frame read operation, and the frame itself. We capture the frame, and then we convert it to grayscale. This is a necessary operation, because face detection in OpenCV happens in the grayscale color space:

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

Step 5: Much like the single still image example, we call detectMultiScale on the grayscale version of the frame.

```
  for (x,y,w,h) in faces:
    img = cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray, 1.03,
    5, 0, (40,40))
```

Step 6: Here we have a further step compared to the still image example: we create a region of interest corresponding to the face rectangle, and within this rectangle, we operate "eye detection". This makes sense as you wouldn't want to go looking for eyes outside a face (well, for human beings at least!).

```
  for (ex,ey,ew,eh) in eyes:
    cv2.rectangle(img,(ex,ey),(ex+ew,ey+eh),
    (0,255,0),2)
```

Step 7: Again, we loop through the resulting eye tuples and draw green rectangles around them.

```

    cv2.imshow("camera", frame)
    if cv2.waitKey(1000 / 12) & 0xff == ord("q"):
      break

  camera.release()
  cv2.destroyAllWindows()

if __name__ == "__main__":
detect()
```

Provide the following:

- Screenshot of the output for the working code once you've put it all together.
- Summary of the steps you've performed along with observations.

```
import cv2

def detect():
    face_cascade = cv2.CascadeClassifier(r'C:\Users\Keith\Documents\
Activity 6. Face Detection using OpenCV\
haarcascade_frontalface_default.xml')
    eye_cascade = cv2.CascadeClassifier(r'C:\Users\Keith\Documents\
Activity 6. Face Detection using OpenCV\haarcascade_eye.xml')
    camera = cv2.VideoCapture(0)

    while True:
        ret, frame = camera.read()
        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)

        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0),
2)
            roi_gray = gray[y:y + h, x:x + w]
            eyes = eye_cascade.detectMultiScale(roi_gray, 1.03, 5,
minSize=(40, 40))

            for (ex, ey, ew, eh) in eyes:
                cv2.rectangle(frame, (x + ex, y + ey), (x + ex + ew, y
+ ey + eh), (0, 255, 0), 2)

        cv2.imshow("camera", frame)
        if cv2.waitKey(1000 // 12) & 0xff == ord("q"):
            break

    camera.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    detect()
```

4. Supplementary Activity

In your Cameo project, include real-time face detection using Haar cascade. Show screenshots of the working demonstration for this supplementary activity.

Additionally, implement similar steps to detect a smile using Haar cascades.

```
import cv2

def detect():
    face_cascade = cv2.CascadeClassifier(r'C:\Users\Keith\Documents\
Activity 6. Face Detection using OpenCV\
haarcascade_frontalface_default.xml')
    eye_cascade = cv2.CascadeClassifier(r'C:\Users\Keith\Documents\
Activity 6. Face Detection using OpenCV\haarcascade_eye.xml')
    smile_cascade = cv2.CascadeClassifier(r'C:\Users\Keith\Documents\
Activity 6. Face Detection using OpenCV\haarcascade_smile.xml')
    camera = cv2.VideoCapture(0)

    while True:
        ret, frame = camera.read()
        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)

        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0),
2)
            roi_gray = gray[y:y + h, x:x + w]
            eyes = eye_cascade.detectMultiScale(roi_gray, 1.03, 5,
minSize=(40, 40))
            smiles = smile_cascade.detectMultiScale(roi_gray, 1.8, 20)

            for (ex, ey, ew, eh) in eyes:
                cv2.rectangle(frame, (x + ex, y + ey), (x + ex + ew, y
+ ey + eh), (0, 255, 0), 2)

            if len(smiles) > 0:
                cv2.putText(frame, "Smiling (Sobrang Latina)", (x, y -
10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
                for (sx, sy, sw, sh) in smiles:
                    cv2.rectangle(frame, (x + sx, y + sy), (x + sx +
sw, y + sy + sh), (0, 255, 255), 2)
            else:
                cv2.putText(frame, "Not Smiling (awts gegege)", (x, y
- 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

        cv2.imshow("camera", frame)
        if cv2.waitKey(1000 // 12) & 0xff == ord("q"):
            break

    camera.release()
    cv2.destroyAllWindows()
```

```
if __name__ == "__main__":
    detect()

The Kernel crashed while executing code in the current cell or a
previous cell.

Please review the code in the cell(s) to identify a possible cause of
the failure.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for
more info.

View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further
details.
```

alt text

alt text

5. Summary, Conclusions and Lessons Learned

Through this activity, I learned that pre-trained models like Haar cascades provide a quick and easy way to detect faces, but their accuracy depends on external conditions. And in this project, I implemented real-time face, eye, and smile detection using OpenCV and Haar cascades. The program successfully identified facial features and indicated whether a person was smiling. While the detection worked well under good conditions, accuracy was affected by lighting, angles, and etc. I learned that right parameters improves performance, and real-time processing requires optimization for smooth operation. This activity also made me realize the limitations of Haar cascades and more accurate detection.