IMAGE PROCESSING

Face detection using cv2

HARMAN ABDUL WAHEED

Contents

Open-cv introduction

1. Introductions to Open-cv in python

- It is a library used for image processing tasks.
- We can access camera through it, capture videos, apply image processing techniques and filters and much more.

To download the library, open anaconda prompt and enter this command:

```
pip install opencv-python
```

After successful download, we can load the library in python by using this statement:

```
import cv2
```

2. Loading and showing images

To load an image, we give path of image this way:

```
cv2.imread('path')
```

For example, I have a cat image in local disk D, I will load my image this way:

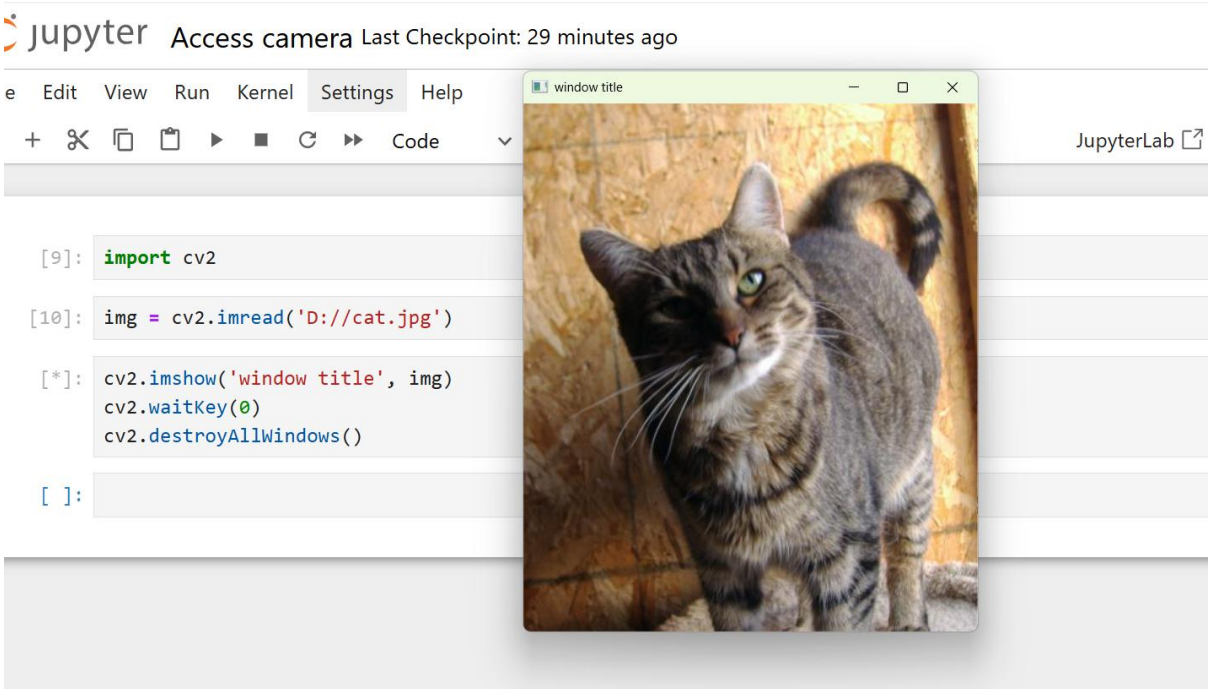
```
img = cv2.imread("D://cat.jpg")
```

To show this image, run these lines of code:

```
cv2.imshow('window title', img)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- A window for my image will appear with title ‘window title’.
- It will wait for us to press any key. As we press the key, it will stop the code.
- All cv2 windows will be destroyed.



2. Reading video feed from camera

Accessing the camera:

```
cap = cv2.VideoCapture(0)
```

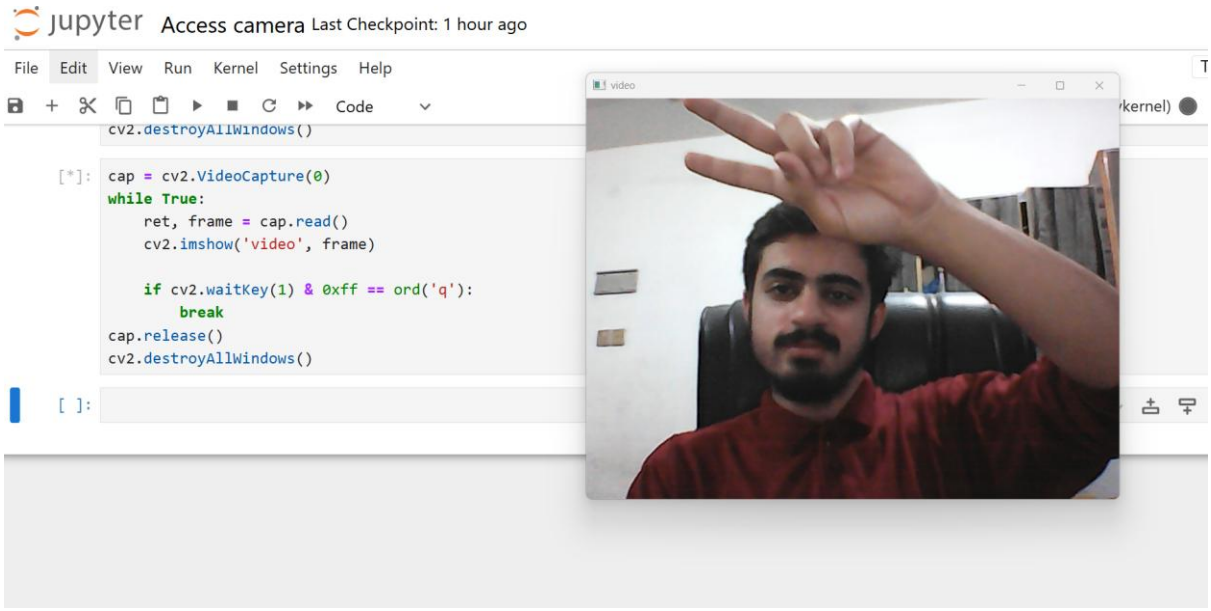
It will access the default camera of laptop computer

Reading and showing frames continuously:

```
While True:
      ret, frame = cap.read()
      cv2.imshow('video', frames)
```

```
if cv2.waitKey(1) & 0xff == ord('q'):
        break
```

- We have initiated an infinite while loop because we want to display the frames again and again (in form of video).
- `cap.read` gives two outputs: first one is True or False indicating whether the camera is working or not. Second one is the actual frame captured by camera at that moment.
- We have displayed the actual frame in windows named 'video'.
- As soon as someone press 'q', the loop will break.



Closing the camera and cv2 windows:

```
cap.release()
cv2.destroyAllWindows()
```

- Camera is set free.
- Window displaying the video is closed (all cv2 windows are closed)

Face Detection

1. Import the Required Library

```
import cv2
```

- `cv2` is OpenCV's library for image processing tasks.
- We use it to access the webcam, detect faces, and draw rectangles.

---

2. Load the Pre-trained Face Detection Model

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

- OpenCV provides pre-trained **Haar Cascade classifiers** for face detection.
- The `cv2.CascadeClassifier` loads the **Haar cascade XML file** for detecting frontal faces.
- `cv2.data.haarcascades` provides the path to OpenCV's built-in Haar cascade files.

---

3. Open the Webcam

```
cap = cv2.VideoCapture(0)
```

- `cv2.VideoCapture(0)` accesses the default webcam (`0` refers to the primary camera).
- If you have multiple cameras, you can use `1`, `2`, etc., instead of `0`.

---

4. Start an Infinite Loop to Read Frames

```
while True:
```

- Runs continuously to process video frames in real-time.

---

5. Capture Frames from the Webcam

```
ret, frame = cap.read()
if not ret:
    break  # Stop if the camera is not working
```

- `cap.read()` reads a frame from the webcam.
- `ret` is `True` if a frame is successfully captured; otherwise, it stops.

---

6. Convert the Frame to Grayscale

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

- The Haar cascade works better with **grayscale images**.
- `cv2.COLOR_BGR2GRAY` converts the frame from **color (BGR)** to **grayscale**.

---

7. Detect Faces in the Grayscale Image

```
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5, minSize=(30, 30))
```

- `detectMultiScale()` detects faces and returns a list of bounding box coordinates `(x, y, w, h)`.
- **Parameters**:
  - `gray`: Input grayscale image.
  - `scaleFactor=1.3`: Resizes image in steps to detect faces at different scales. Greater value: more accurate, but slow processing comparatively.
  - `minNeighbors=5`: how many detections for same face in (higher = more accuracy, but may miss some faces).
  - `minSize=(30, 30)`: Ignores objects smaller than `30x30` pixels.

---

8. Draw Rectangles Around Detected Faces

```
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)
```

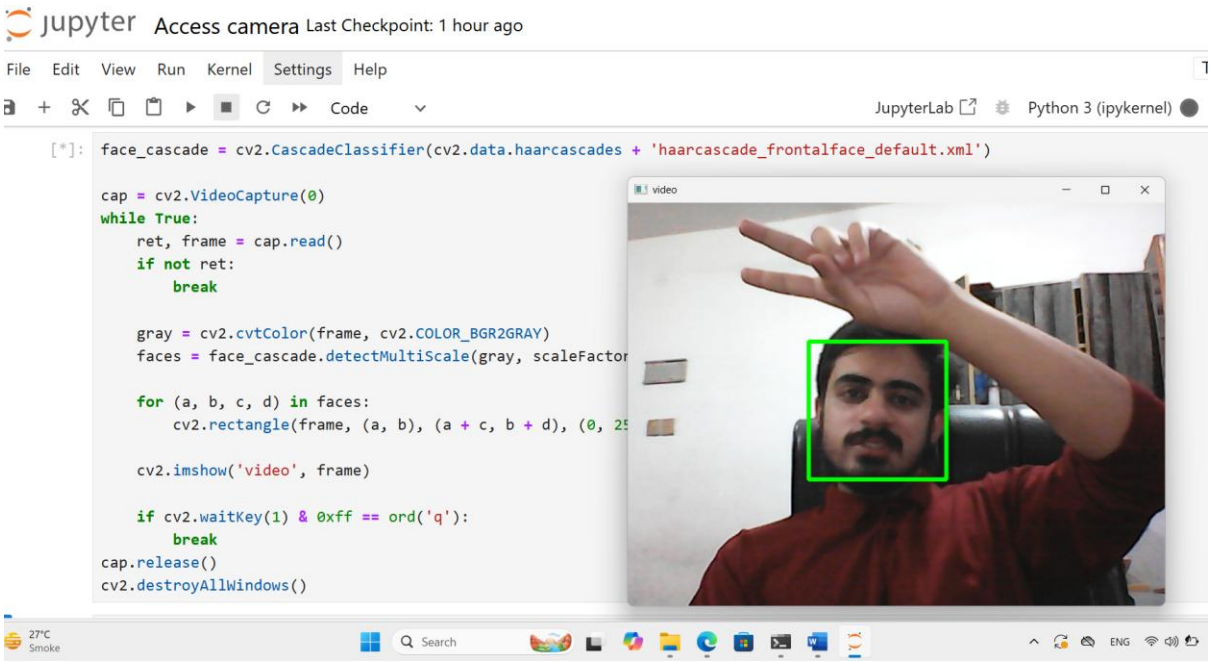
- Loops through all detected faces.
- `cv2.rectangle()` draws a green (`(0, 255, 0)`) rectangle **around each detected face**.
    - `(x, y)`: Top-left corner.
    - `(x + w, y + h)`: Bottom-right corner.
    - `3`: Thickness of the rectangle border.

---

9. Display the Frame with Detected Faces

```
cv2.imshow("Face Detection", frame)
```

- Displays the processed frame with rectangles around faces in a window titled **"Face Detection"**.



---

10. Exit the Loop When 'q' is Pressed

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

- `ord('q')`: If the **'q' key is pressed**, the loop breaks, and the program stops.

---

11. Release Resources and Close Windows

```
cap.release()
cv2.destroyAllWindows()
```

- `cap.release()`: Releases the webcam so other programs can use it.
- `cv2.destroyAllWindows()`: Closes all OpenCV windows.
