JUL 22ND, 2014 | COMMENTS

Face Detection in Python Using a Webcam

The following is a guest post by Shantnu Tiwari, who suffered at the hands of C/C++ for several years before he discovered Python, and it felt like a breath of fresh air.

He is now sharing his love (http://pythonforengineers.com/).

*This blog post is a follow-up to **Face Recognition in Python (https://realpython.com/blog/python/face-recognition-with-python/)**, so make sure you've gone through that first post.*

As mentioned in the first post, it's quite easy to move from detecting faces in images to detecting them in video via a webcam – which is exactly what we will detail in this post.

Before you ask any questions in the comments section:

1. Do not skip over the blog post and try to run the code. You must understand what the code does not only to run it properly but to troubleshoot it as well.
2. Make sure to use OpenCV v2.
3. You need a working webcam for this script to work properly.
4. Review the other comments/questions as your questions have probably already been addressed.

Thank you.

Pre-requisites:

1. OpenCV installed (see the previous blog post for details)
2. A working webcam

The Code

Let's dive straight into the code, taken from this repository (https://github.com/shantnu/Webcam-Face-Detect).

```
import cv2
import sys

cascPath = sys.argv[1]
faceCascade = cv2.CascadeClassifier(cascPath)

video_capture = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

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

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.cv.CV_HAAR_SCALE_IMAGE
    )

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Video', frame)

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

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

Now let's break it down...

```
1  import cv2
2  import sys
3
4  cascPath = sys.argv[1]
5  faceCascade = cv2.CascadeClassifier(cascPath)
```

This should be familiar to you. We are creating a face cascade, as we did in the image example.

```
1  video_capture = cv2.VideoCapture(0)
```

This line sets the video source to the default webcam, which OpenCV can easily capture.

> **NOTE**: You can also provide a filename here, and Python will read in the video file. However, you need to have ffmpeg (https://www.ffmpeg.org/) installed for that since OpenCV itself cannot decode compressed video. Ffmpeg acts as the front end for OpenCV, and, ideally, it should be compiled directly into OpenCV. This is not easy to do, especially on Windows.

```
1  while True:
2      # Capture frame-by-frame
3      ret, frame = video_capture.read()
```

Here, we capture the video. The `read()` function reads one frame from the video source, which in this example is the webcam. This returns:

1. The actual video frame read (one frame on each loop)
2. A return code

The return code tells us if we have run out of frames, which will happen if we are reading from a file. This doesn't matter when reading from the webcam, since we can record forever, so we will ignore it.

```
# Capture frame-by-frame
ret, frame = video_capture.read()

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

faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(30, 30),
    flags=cv2.cv.CV_HAAR_SCALE_IMAGE
)

# Draw a rectangle around the faces
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the resulting frame
cv2.imshow('Video', frame)
```

Again, this code should be familiar. We are merely searching for the face in our captured frame.

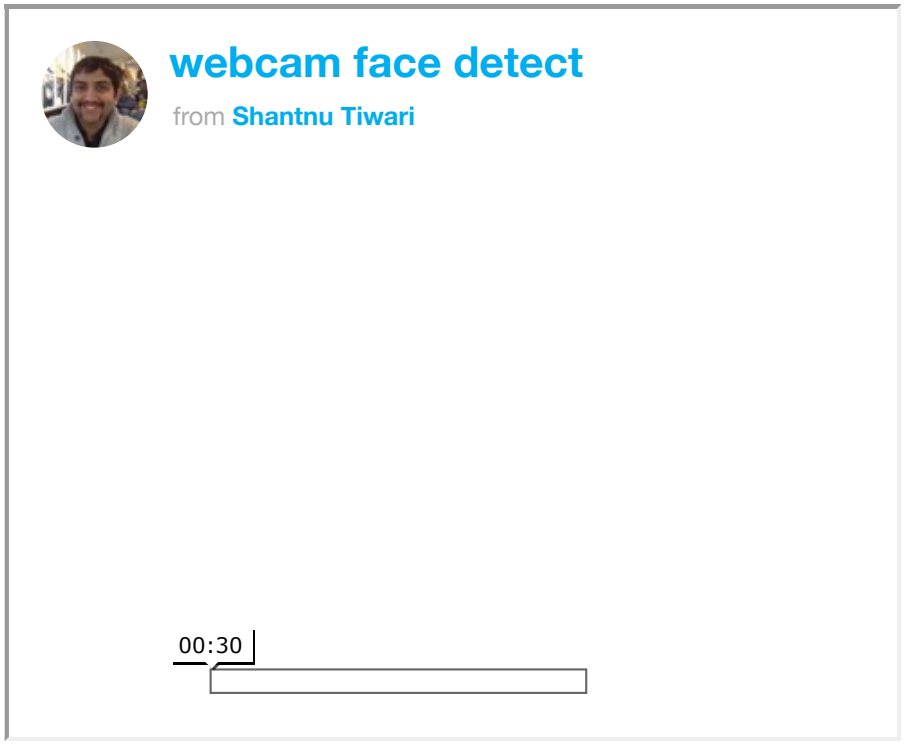
```
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

We wait for the ‘q’ key to be pressed. If it is, we exit the script.

```
# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

Here, we are just cleaning up.

Test!



So, that's me with my driver's license in my hand. And you can see that the algorithm tracks both the real me and the photo me. Note that when I move slowly, the algorithm can keep up. When I move my hand up to my face a bit faster, though, it gets confused and mistakes my wrist for a face.

Like I said in the last post, machine learning based algorithms are rarely 100% accurate. We aren't at the stage where Robocop driving his motorcycle at 100 mph can track criminals using low quality CCTV cameras... yet.

The code searches for the face frame by frame, so it will take a fair amount of processing power. For example, on my five year old laptop, it took almost 90% of the CPU.

Next Steps

Okay, so you know how to detect faces. But what if you want to detect your own object, like your car or your TV or your favorite toy?

OpenCV allows you to create your own cascades, but the process isn't well documented. Here is a blog (http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html) post that shows you how to train your own cascade to detect a banana.

If you want to take it one step further and recognize individual faces – perhaps to detect and recognize your face amongst many strangers – the task is surprisingly difficult. This is mainly due to the large amount of image pre-processing involved. But if you are willing to tackle the challenge, it is possible by using machine learning algorithms as described here (http://scikit-learn.sourceforge.net/0.6/auto_examples/applications/plot_face_recognition.html).

Want to know more?

This will be covered in much greater detail along with a number of computational science and machine learning topics and more in my upcoming course. The course is based on a highly successful Kickstarter (https://www.kickstarter.com/projects/513736598/python-for-science-and-engineering).

The Kickstarter is over, but you can still order the course at Python for Engineers (http://pythonforengineers.com/). Visit to find out more.

Also, post links to your videos below to get direct feedback from me. Comment if you have questions.

Oh – and next time we'll be tackling some motion detection. Stay tuned!

Cheers!

Posted by Real Python Jul 22nd, 2014 data science (/blog/categories/data-science/), opencv (/blog/categories/opencv/), python (/blog/categories/python/)

See an error in this post? Please submit a pull request on Github (https://github.com/realpython/realpython-blog).

« Django Migrations - A Primer (/blog/python/django-migrations-a-primer/)

Digging Deeper into Migrations » (/blog/python/digging-deeper-into-migrations/)

Comments