Programming a face detection tool in Python

Python face recognition is not as difficult as many people think. Below is a simple Python face recognition code for beginners.

Have you ever wondered how Snapchat and Messenger filters fit your face? Your smartphone uses magic to unlock with your face? No, you're simply witnessing facial recognition technology in action.

What is computer vision?

We are currently living in an age of AI revolution, marked by impressive advances in the field of deep learning. Throughout this year, we have witnessed amazing achievements of artificial intelligence applications that have surprised the whole world when they create realistic works of art, pass exams, and even , write Python code to create websites.

Computer vision is a deep learning application that is at the heart of this revolution. It allows computers to better understand input visual data such as images and video files. Typical examples of computer vision include facial recognition, facial recognition, human pose estimation, and obstacle detection.

In this article, let's learn how to make a Python facial recognition tool with TipsMake.com.com. This is one of the most popular programming languages ??today. It has many real-world applications, and facial recognition is just the most typical example.

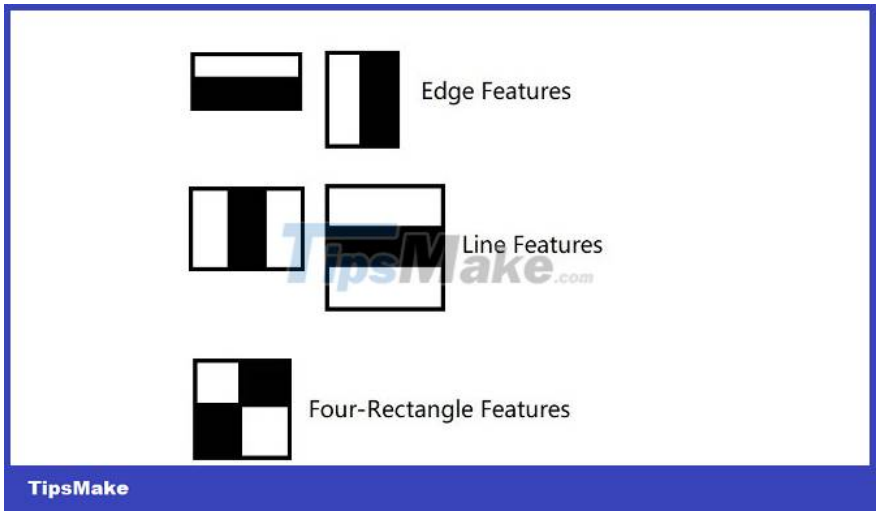
Facial recognition is an artificial intelligence technology that can identify human faces in digital photos or videos. In this article, TipsMake.com will learn with you how to build a real-time facial recognition tool in just under 25 lines of code with the legendary Haar Cascade algorithm.

What is Haar Cascade?

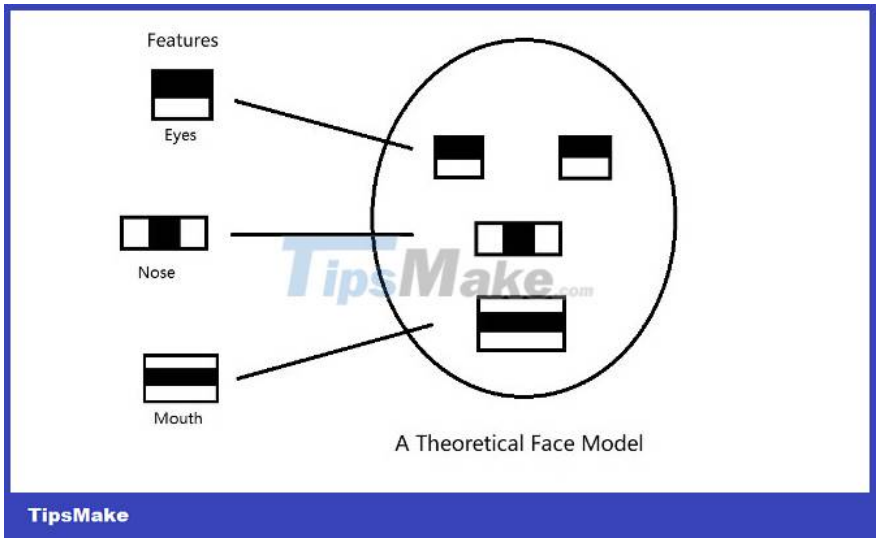
Haar Cascade is an object detection algorithm introduced by Paul Viola and Michael Jones to detect faces in photos or videos. A stratification function is trained using multiple negative and positive photos, which can later be used to identify any object or face. These pre-trained files are available in the OpenCV GitHub repo.

Using the sliding window approach, a fixed-sized window repeats the image from left to right, top to bottom. At each stage, the window stops and classifies whether the region contains a face or not.

OpenCV, a machine vision tool, works with a pre-trained Haar Cascade model to classify objects. Each phase tests five objects, two edge objects, two line objects, and one interwoven four-cell object.



When observed, the eye area appears darker than the cheek area while the nose area is brighter than the eye area. You can visualize the objects in the image below:



Using these objects and calculating the pixels, the algorithm identifies more than 100,000 data points. You can then further use the Adaboost algorithm to improve accuracy and remove irrelevant objects. Over many iterations, this approach minimizes the error rate and continues to adjust the object to achieve acceptable accuracy.

However, the sliding window technique stops if a particular test case fails and it is computationally expensive. To solve this problem, you can apply the concept of Cascade of Classifiers. Instead of applying all objects in a single window, this method groups and applies them in stages.

If the window fails at the first stage, the process will discard it, otherwise it will continue. This results in a significant reduction in the number of operations that must be performed and makes it possible to be used for real-time applications.

Face detection process

Here is the process you need to follow to build a facial recognition tool:

1. Download the Haar Cascade Frontal Face Algorithm
2. Initialize the camera
3. Read frames from camera
4. Convert images to grayscale
5. Get face coordinates
6. Draw a rectangle and place the appropriate message
7. Show output

What is OpenCV?

OpenCV is an open source machine vision and machine learning library. It has more than 2,500 algorithms that have been optimized for a variety of applications. This includes recognition, face/object detection, classification.

Many large companies such as Google, IBM and even Yahoo have used OpenCV in their applications. However, before starting you should also consider ensuring the safety of your private data.

To install OpenCV in Python, use the following command:

```
pip install opencv-python
```

Programming a face detection tool in Python

You need to perform the following steps to program a face detector in Python:

1. Download the Haar Cascade Frontal Face Default XML file and place it in the same location as your Python program.
2. Import OpenCV library:

```
# importing the required libraries import cv2
```

1. Save the Haar Cascade Frontal Face algorithm file for easy reference

```
# loading the haar case algorithm file into alg variable alg = "haarcascade_fron
```

1. Use the CascadeClassifier class to load an XML file into OpenCV:

```
# passing the algorithm to OpenCV haar_cascade = cv2.CascadeClassifier(alg)
```

1. Get video from camera. Pass the value 0 to the VideoCapture() function to use your main camera. If you have more cameras, you can use the next numbers like 1, 2. to designate the cameras.

```
# capturing the video feed from the camera cam = cv2.VideoCapture(0)
```

1. Set up an infinite loop to read each frame of the camera input. The read() function returns two parameters. The first value is a boolean to indicate the second success of the operation. The second parameter contains the actual framework you will be working with. Save this frame in the img variable.

```
while True: _, img = cam.read()
```

1. The default notification text setting is Face not detected. When a face is detected, update the value to that variable.

```
 text = "Face not detected"
```

1. Input from the real world is colorful but in BGR format. BGR stands for blue, green and red. This will cause the machine vision application to have a lot to handle. Therefore, to reduce the process volume, we use grayscale format.

```
 # convert each frame from BGR to Grayscale grayImg = cv2.cvtColor(img, cv2.COLO
```

Feed the frames and format conversion code, COLOR_BGR2GRAY, into cvtColor() to change each frame of the video from color to grayscale.

1. Use detectMultiScale() to detect faces. This method uses three parameters as input. First is the image source, grayImg. The second parameter is scaleFactor. It specifies how much you must reduce the image size at each image ratio. Use the default value 1.3 as the scaling factor. The higher the scaling factor, the faster the execution speed because fewer steps need to be performed. However, the possibility of missing faces is also higher. The third parameter is minNeighbors. This parameter specifies the number of neighbor functions each rectangle must have in order to retain it. Higher values ??have a lower chance of falsely detecting faces, but also a higher rate of missing ambiguous faces.

```
 # detect faces using Haar Cascade face = haar_cascade.detectMultiScale(grayImg,
```

1. When you detect a face, you will receive 4 coordinates. x represents x coordinate, y represents y coordinate, w represents width, and h represents height. Update the text message as Face Detected and draw a rectangle based on those coordinates. The color of the rectangle is green (in BGR format) and the rectangle border thickness is 2 pixels.

```
 # draw a rectangle around the face and update the text to Face Detected for (x,
```

1. Option to print text on the output panel. Display text on screen using captured frame as source, text captured in above text, font style of FONT_HERSHEY_SIMPLEX, font scale factor of 1, color blue, thickness 2 pixel and line type AA.

```
# display the text on the image print(text) image = cv2.putText(img, text, (50, 5
```

1. Displays a window with the title Face Detection and an image. Use the waitkey() method to display the window for 10 milliseconds and check for a key press. If the user presses the Esc key (ASCII Value 27), exit the loop.

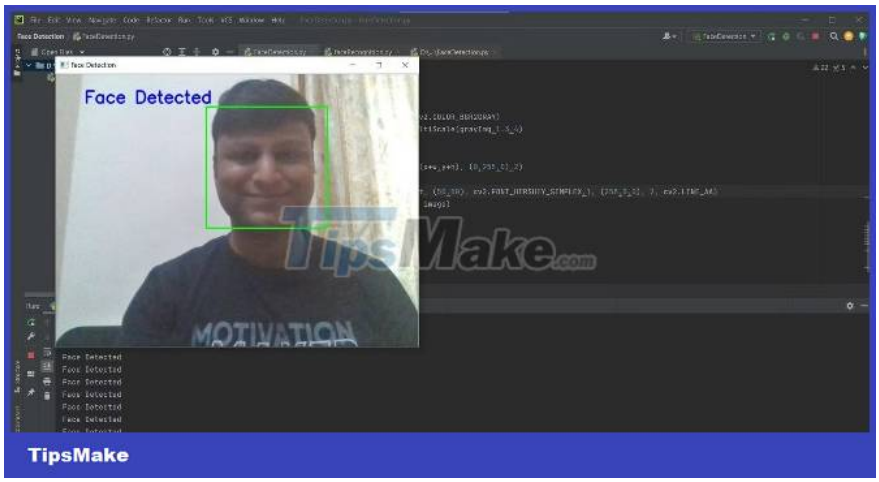
```
# display the output window and press escape key to exit cv2.imshow("Face Detecti
```

1. Finally, release the camera object from the Python program and close all windows.

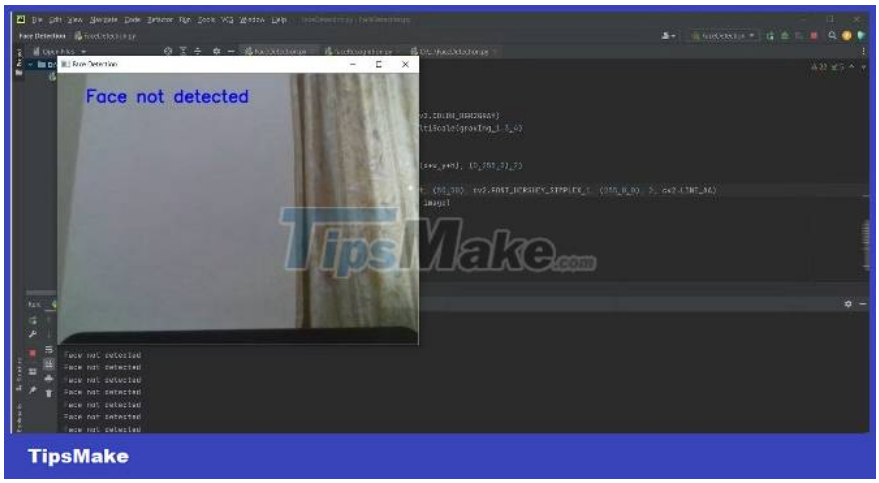
```
cam.release() cv2.destroyAllWindows()
```

Test run a facial recognition program programmed in Python

When the face is in frame, you'll see output like this:



When there is no face, you will receive a message like this:



Crop and save detected faces using Python

To crop and save a detected face in a photo, you can follow the steps below:

1. Import the required library. In most cases, Python libraries usually require OpenCV. So, make sure you have it installed.
2. Read the input image using cv2.imread() . Specify the full image path. Convert imported images to gray scale.
3. Initialize the Haar cascading classifier object face_cascade = cv2.CascadeClassifier() to detect faces. Convert the full path of the Haar Cascade file. You can use the haarcascade_frontalface_alt.xml file to detect faces in images.
4. Detect faces in imported images using face_cascade.detectMultiScale() . It returns the coordinates of the detected face in (x,y,w,h) format.

5. Loop through all detected faces. Find image[y:y+h, x:x+w] which is the cropped face and assign it a new variable, say face. Save the cropped face with cv2.imwrite() .
6. Optionally, show cropped faces for data visualization purposes.

Limitations of facial recognition using the Haar Cascade algorithm

While this algorithm is very lightweight, has a small sample size, and works fast, it has the following disadvantages:

1. In real video, the face needs to lie straight and within the camera's standard field of view. If located too far, too close or too tilted, the algorithm will fail to identify the objects.
2. This is an algorithm to determine the frontal face, so it cannot detect the side angle of the face.
3. The rate of false identification is very high. It often identifies an area as a face even though no face is present there.
4. Lighting conditions need to be optimized. The algorithm's accuracy will decrease significantly in excessive or dim lighting conditions.

Facial recognition is applied in many fields

Currently, facial recognition technology is being applied in many different fields. You can use it for unlocking your smartphone, home, vehicle, and airport authentication. Facial recognition is now also being used in surveillance camera systems, social media filters, and more. Automatic face tracking technology in the film industry.