Face Detection with Azure Face API

Dileep A

Table of Contents

Computer vision is an exciting and growing field. There are tons of interesting problems to solve! One of them is face detection: the ability of a computer to recognize that a photograph contains a human face and tell you where it is located. In this article, you'll learn about face detection with Azure Face API with Python.

To detect any object in an image, it is necessary to understand how images are represented inside a computer, and how that objects differs *visually* from any other object.

Once that is done, the process of scanning an image and looking for those visual cues needs to be automated and optimized. All these steps come together to form a fast and reliable computer vision algorithm.

In this Tutorial You'll learn:

- What is Azure Face API
- How Computers understand features in images
- How to quickly analyze many different features to reach a decision
- How to use a minimal python solution with Azure API for detecting Human Faces in Images.

What is Azure Face API?

The Azure Cognitive Services Face API provides algorithms that are used to detect, recognize, and analyze human faces in images. The ability to process human face information is important in many different software scenarios. Example scenarios are security, natural user interface, image content analysis and management, mobile apps, and robotics. The Face API provides several different functions. Each function is outlined in the following sections.

Face Detection

The Face API detects human faces in an image and returns the rectangle coordinates of their locations. Optionally, face detection can extract a series of face-related attributes. Examples are head pose, gender, age, emotion, facial hair, and glasses.

This article explains the concepts of face detection and face attribute data. Face detection is the action of locating human faces in an image and optionally returning different kinds of face-related data.

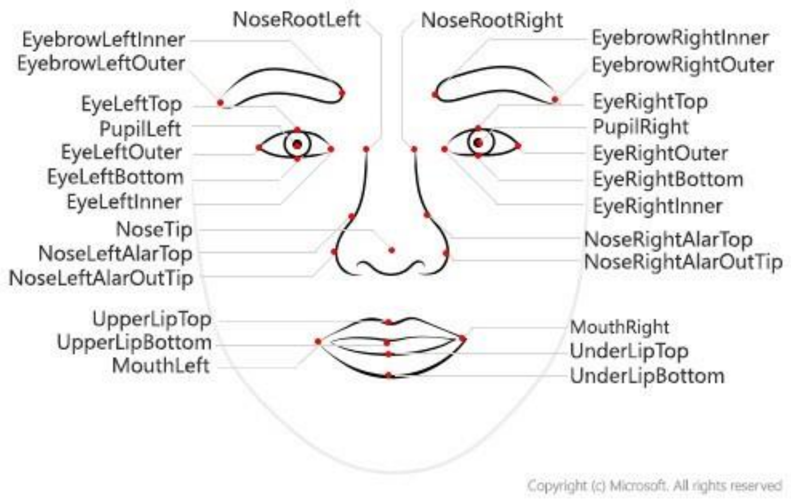
You use the **Face - Detect** operation to detect faces in an image. At a minimum, each detected face corresponds to a face Rectangle field in the response. This set of pixel coordinates for the left, top, width, and height mark the located face. Using these coordinates, you can get the location of the face and its size. In the API response, faces are listed in size order from largest to smallest.

Face ID

The face ID is a unique identifier string for each detected face in an image. You can request a face ID in your **Face - Detect** API call.

Face landmarks

Face landmarks are a set of easy-to-find points on a face, such as the pupils or the tip of the nose. By default, there are 27 predefined landmark points. The following figure shows all 27 points:

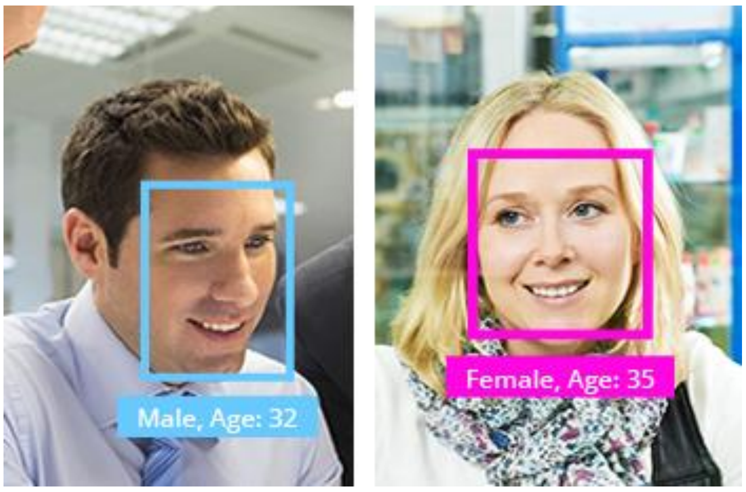


The Coordinates of points are returned in units of pixels.

Attributes

Attributes are a set of features that can optionally be detected by the Face - Detect API. The following attributes can be detected:

- **Age.** The estimated age in years of a face.
- **Blur.** The blurriness of the face in the image. This attribute returns a value between zero and one and an informal rating of low, medium, or high.
- **Emotion.** A list of emotions with their detection confidence for the given face. Confidence scores are normalized, and the scores across all emotions add up to one. The emotions returned are happiness, sadness, neutral, anger, contempt, disgust, surprise, and fear.



- **Exposure.** The exposure of the face in the image. This attribute returns a value between zero and one and an informal rating of under Exposure, good Exposure, or over Exposure.
- **Facial hair.** The estimated facial hair presence and the length for the given face.
- **Gender.** The estimated gender of the given face. Possible values are male, female, and genderless.
- **Glasses**. Whether the given face has eyeglasses. Possible values are No Glasses, Reading Glasses, Sunglasses, and Swimming Goggles.
- **Hair.** The hair type of the face. This attribute shows whether the hair is visible, whether baldness is detected, and what hair colors are detected.
- **Head pose**. The face's orientation in 3D space. This attribute is described by the pitch, roll, and yaw angles in degrees. The value ranges are -90 degrees to 90 degrees, -180 degrees to 180 degrees, and -90 degrees to 90 degrees, respectively. See the following diagram for angle mappings:
- **Makeup**. Whether the face has makeup. This attribute returns a Boolean value for eye Makeup and lip Makeup.
- **Noise.** The visual noise detected in the face image. This attribute returns a value between zero and one and an informal rating of low, medium, or high.
- **Occlusion.** Whether there are objects blocking parts of the face. This attribute returns a Boolean value for eye Occluded, forehead Occluded, and mouth Occluded.
- **Smile.** The smile expression of the given face. This value is between zero for no smile and one for a clear smile.

Face Verification

This article explains the concepts of the Verify, Find Similar, Group, and Identify face recognition operations and the underlying data structures. Broadly, recognition describes the work of comparing two different faces to determine if they're similar or belong to the same person.

Recognition-related data structures

The recognition operations use mainly the following data structures. These objects are stored in the cloud and can be referenced by their ID strings. ID strings are always unique within a subscription. Name fields can be duplicated.

NAME	DESCRIPTION
Detected Face	This single face representation is retrieved by the face detection operation. Its ID expires 24 hours after it's created.
Persisted Face	When Detected Face objects are added to a group, such as Face List or Person, they become Persisted Face objects. They can be retrieved at any time and don't expire.
Face List or Large Face List	This data structure is an assorted list of Persisted Face objects. A FaceList has a unique ID, a name string, and optionally a user data string.
Person	This data structure is a list of Persisted Face objects that belong to the same person. It has a unique ID, a name string, and optionally a user data string.
Person Group or Large Person Group	This data structure is an assorted list of Person objects. It has a unique ID, a name string, and optionally a user data string. A Person Group must be trained before it can be used in recognition operations.

Recognition operations

This section details how the four recognition operations use the data structures previously described. For a broad description of each recognition operation.

Verify

The **Verify** operation takes a face ID from Detected Face or Persisted Face and either another face ID or a Person object and determines whether they belong to the same person. If you pass in a Person object, you can optionally pass in a Person Group to which that Person belongs to improve performance.

Find Similar

The **Find Similar** operation takes a face ID from Detected Face or Persisted Face and either a Face List or an array of other face IDs. With a Face List, it returns a smaller Face List of faces that are like the given face. With an array of face IDs, it similarly returns a smaller array.

Group

The **Group** operation takes an array of assorted face IDs from Detected Face or Persisted Face and returns the same IDs grouped into several smaller arrays. Each "groups" array contains face IDs that appear similar. A single "messy Group" array contains face IDs for which no similarities were found.

Identify

The **Identify** operation takes one or several face IDs from Detected Face or Persisted Face and a Person Group and returns a list of Person objects that each face might belong to. Returned Person objects are wrapped as Candidate objects, which have a prediction confidence value.

How Do Computers “See” Images?

The smallest element of an image is called a **pixel**, or a picture element. It is basically a dot in the picture. An image contains multiple pixels arranged in rows and columns.

You will often see the number of rows and columns expressed as the image **resolution**. For example, an Ultra HD TV has the resolution of 3840x2160, meaning it is 3840 pixels wide and 2160 pixels high.

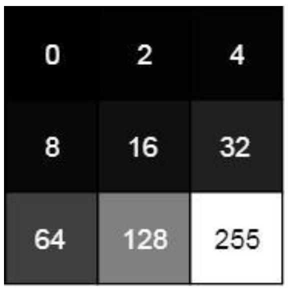
But a computer does not understand pixels as dots of color. It only understands numbers. To convert colors to numbers, the computer uses various color models.

In color images, pixels are often represented in the RGB color model. RGB stands for **R**ed **G**reen **B**lue. Each pixel is a mix of those three colors. RGB is great at modeling all the colors humans perceive by combining various amounts of red, green, and blue.

Since a computer only understand numbers, every pixel is represented by three numbers, corresponding to the amounts of red, green, and blue present in that pixel.

In grayscale (black and white) images, each pixel is a single number, representing the amount of light, or intensity, it carries. In many applications, the range of intensities is from 0 (black) to 255 (white). Everything between 0 and 255 is various shades of gray.

If each grayscale pixel is a number, an image is nothing more than a matrix (or table) of numbers:



0	2	4
8	16	32
64	128	255

Example 3x3 image with pixel values and colors

In color images, there are three such matrices representing the red, green, and blue channels.

Preparation:

Create a Python script to detect and frame faces in an image

Prerequisites

- A Face API subscription key. You can get a free trial subscription key from Try Cognitive Services. Or, follow the instructions in Create a Cognitive Services account to subscribe to the Face API service and get your key.
- Python 2.7+ or 3.5+
- pip tool

Detect Faces in an image

The script will detect faces by calling the **cognitive_face.detect** method, which wraps the Detect REST API and returns a list of faces.

```
    subscription_key = "f57b242734524a28b4177257a64e5b46" #DSFace API
    face_api_url = 'https://westcentralus.api.cognitive.microsoft.com/face/v1.0/detect'

      print("Call Config")

  return subscription_key, face_api_url
```

Face attributes

```
[{'faceId': '63916bf1-9c75-48c4-aae9-ef59f1f16271',
'faceRectangle': {'top': 77, 'left': 63, 'width': 211, 'height': 211},
'faceLandmarks': {'pupilLeft': {'x': 116.9, 'y': 134.8},
                 'pupilRight': {'x': 206.2, 'y': 127.6},
                 'noseTip': {'x': 170.4, 'y': 202.3},
                 'mouthLeft': {'x': 132.6, 'y': 233.6},
                 'mouthRight': {'x': 219.6, 'y': 222.3},
                 'eyebrowLeftOuter': {'x': 71.9, 'y': 123.9},
                 'eyebrowLeftInner': {'x': 143.1, 'y': 127.2},
                 'eyeLeftOuter': {'x': 99.5, 'y': 138.3},
                 'eyeLeftTop': {'x': 115.8, 'y': 130.8},
                 'eyeLeftBottom': {'x': 116.5, 'y': 142.7},
                 'eyeLeftInner': {'x': 131.9, 'y': 137.2},
                 'eyebrowRightInner': {'x': 178.4, 'y': 122.9},
                 'eyebrowRightOuter': {'x': 244.2, 'y': 111.4},
                    'eyeRightInner': {'x': 191.6, 'y': 131.8},
                    'eyeRightTop': {'x': 205.2, 'y': 122.4},
                    'eyeRightBottom': {'x': 207.2, 'y': 134.1},
                    'eyeRightOuter': {'x': 219.4, 'y': 127.9},
                    'noseRootLeft': {'x': 148.6, 'y': 141.4},
                    'noseRootRight': {'x': 172.5, 'y': 139.6},
                    'noseLeftAlarTop': {'x': 147.2, 'y': 180.5},
                    'noseRightAlarTop': {'x': 182.6, 'y': 177.1}, '
                    noseLeftAlarOutTip': {'x': 141.9, 'y': 200.2},
                    'noseRightAlarOutTip': {'x': 195.1, 'y': 192.1},
                    'upperLipTop': {'x': 176.2, 'y': 228.7},
                    'upperLipBottom': {'x': 177.1, 'y': 235.8},
                    'underLipTop': {'x': 178.9, 'y': 242.8},
                    'underLipBottom': {'x': 180.1, 'y': 254.9}}
        'faceAttributes': {'smile': 0.811, 'headPose': {'pitch': 0.0, 'roll': -7.2, 'yaw': -0.6},
        'gender': 'male',
        'age': 36.0,
        'glasses': 'NoGlasses',
        'emotion': {'anger': 0.0, 'contempt': 0.001, 'disgust': 0.0, 'fear': 0.0, 'happiness': 0.811,
'neutral': 0.188, 'sadness': 0.0, 'surprise': 0.0}}]
```

AzureFaceAPI_Basic.py

```
    import requests
    from PIL import Image
    import os
    import FaceAPIConfig as cnfg

    image_path = os.path.join('C:/Py/FaceAPIAzure/CapFrame.jpg')
    image_data = open(image_path, "rb")

    subscription_key, face_api_url = cnfg.config();

    headers = {'Content-Type': 'application/octet-stream',
        'Ocp-Apim-Subscription-Key': subscription_key}

    params = {
      'returnFaceId': 'true',
      'returnFaceLandmarks': 'true',
      'returnFaceAttributes': 'age,gender,headPose,smile,facialHair,glasses,emotion'
```

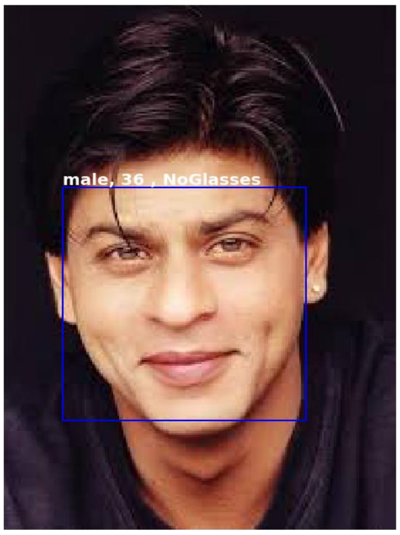
```
}

response = requests.post(face_api_url, params=params, headers=headers,
data=image_data)
response.raise_for_status()
faces = response.json()
print(faces)
```

AzureFaceAPI.py

```
import requests
import matplotlib.pyplot as plt
from PIL import Image
from matplotlib import patches
from io import BytesIO
import os
import FaceAPIConfig as cnfg

image_path = os.path.join('C:/Py/FaceAPIAzure/CapFrame.jpg')
image_data = open(image_path, "rb")

subscription_key, face_api_url = cnfg.config();

headers = {'Content-Type': 'application/octet-stream',
        'Ocp-Apim-Subscription-Key': subscription_key}

params = {
    'returnFaceId': 'true',
    'returnFaceLandmarks': 'true',
    'returnFaceAttributes': 'age,gender,headPose,smile,facialHair,glasses,emotion'
}

response = requests.post(face_api_url, params=params, headers=headers,
data=image_data)
response.raise_for_status()
faces = response.json()
print(faces)
```

```
#Display the image
image_orig = open(image_path, "rb").read()
image = Image.open(BytesIO(image_orig))

plt.figure(figsize=(12, 12))
ax = plt.imshow(image, alpha=1)
for face in faces:
    fr = face["faceRectangle"]
    fa = face["faceAttributes"]
    origin = (fr["left"], fr["top"])
    p = patches.Rectangle(
        origin, fr["width"], fr["height"], fill=False, linewidth=2, color='b')
    plt.text(origin[0], origin[1], "%s, %d , %s" % (fa["gender"], fa["age"], fa["glasses"]),
             fontsize=20, color='w', weight="bold", va="bottom")
    ax.axes.add_patch(p)

_ = plt.axis("off")
plt.show()
```

Result



Further Reading

Most current state-of-the-art methods for face detection and recognition use deep learning, which we will cover in a follow-up article.

Conclusion

Good work! You are now able to find faces in images.
In this tutorial, you have learned how to represent regions in an image with Azure Face API features. These features can be calculated very quickly using integral images.
These ideas apply to object detection in general and will help you solve many real-world challenges. Good luck!