Face API

Introduction

Microsoft Face API is a cloud-based service that provides the most advanced face algorithms. Face API has two main functions:

- face detection with attributes
- face recognition
    - Face Verification
    - Finding Similar Face
    - Face Grouping
    - Face Identification
    - Face Storage

Getting Started with Face API in Python Tutorial

Prerequisites

To use the tutorial, you will need to do the following:

- Install either Python 2.7+ or Python 3.5+.
- Install pip.
- Install the Python SDK for the Face API as follows:

Bash Command

```
pip install cognitive_face
```

- Obtain a subscription key for Microsoft Cognitive Services. You can use either your primary or your secondary key in this tutorial. (Note that to use any Face API, you must have a valid subscription key.)

Detect a Face in an Image

PythonCopy

```
import cognitive_face as CF

KEY = '<Subscription Key>'  # Replace with a valid subscription key (keeping the quotes in place).
CF.Key.set(KEY)

BASE_URL = 'https://westus.api.cognitive.microsoft.com/face/v1.0/'  # Replace with your regional Base URL
CF.BaseUrl.set(BASE_URL)

# You can use this example JPG or replace the URL below with your own URL to a JPEG image.
img_url = 'https://raw.githubusercontent.com/Microsoft/Cognitive-Face-Windows/master/Data/detection1.jpg'
faces = CF.face.detect(img_url)
print(faces)
```

Below is an example result. It's a `list` of detected faces. Each item in the list is a `dict` instance where `faceId` is a unique ID for the detected face and `faceRectangle` describes the position of the detected face. A face ID expires in 24 hours.

Python Input

```
[{u'faceId': u'68a0f8cf-9dba-4a25-afb3-f9cdf57cca51', u'faceRectangle': {u'width': 89, u'top': 66, u'height': 89, u'left': 446}}]
```

Draw rectangles around the faces

Using the json coordinates that you received from the previous command, you can draw rectangles on the image to visually represent each face. At the top of the file, add the following:

Python Input

```
import requests
from io import BytesIO
from PIL import Image, ImageDraw
```

Then, after `print(faces)`, include the following in your script:

Python Input

```
#Convert width height to a point in a rectangle
def getRectangle(faceDictionary):
    rect = faceDictionary['faceRectangle']
    left = rect['left']
    top = rect['top']
    bottom = left + rect['height']
    right = top + rect['width']
    return ((left, top), (bottom, right))

#Download the image from the url
response = requests.get(img_url)
img = Image.open(BytesIO(response.content))

#For each face returned use the face rectangle and draw a red box.
draw = ImageDraw.Draw(img)
for face in faces:
    draw.rectangle(getRectangle(face), outline='red')

#Display the image in the users default image browser.
img.show()
```

Further Exploration

To help you further explore the Face API, this tutorial provides a GUI sample. To run it, first install wxPython then run the commands below.

Bash Command

```
git clone https://github.com/Microsoft/Cognitive-Face-Python.git
cd Cognitive-Face-Python
python sample
```

Summary

In this tutorial, you have learned the basic process for using the Face API via invoking the Python SDK. For more information on API details, please refer to the How-To and API Reference.

Face detection using Cognitive Services

Prerequisites

You must have a Cognitive Services API account with **Face API**. The free trial is sufficient for this quickstart. You need the subscription key provided when you activate your free trial, or you may use a paid subscription key from your Azure dashboard.

Running the walkthrough

To continue with this walkthrough, replace `<Subscription Key>` with a valid subscription key.

Python Input

```
subscription_key = "<Subscription Key>"
assert subscription_key
```

Next, verify `face_api_url` and make sure it corresponds to the region you used when generating the subscription key. If you are using a trial key, you don't need to make any changes.

Python Input

```
face_api_url =
'https://westcentralus.api.cognitive.microsoft.com/face/v1.0/detect'
```

Here is the URL of the image. You can experiment with different images by changing `image_url` to point to a different image and rerunning this notebook.

Python Input

```
image_url = 'https://how-old.net/Images/faces2/main007.jpg'
```

The next few lines of code call into the Face API to detect the faces in the image. In this instance, the image is specified via a publicly visible URL. You can also pass an image directly as part of the request body. For more information, see the API reference.

Python Input

```
import requests
from IPython.display import HTML

headers = { 'Ocp-Apim-Subscription-Key': subscription_key }

params = {
    'returnFaceId': 'true',
    'returnFaceLandmarks': 'false',
    'returnFaceAttributes':
'age,gender,headPose,smile,facialHair,glasses,emotion,hair,makeup,occlusion,access
ories,blur,exposure,noise',
}

response = requests.post(face_api_url, params=params, headers=headers,
json={"url": image_url})
faces = response.json()
HTML("<font size='5'>Detected <font color='blue'>%d</font> faces in the
image</font>"%len(faces))
```

The expected result is

HTML Input

```
<font size='5'>Detected <font color='blue'>2</font> faces in the image</font>
```

Finally, the face information can be overlaid of the original image using the `matplotlib` library in Python.

Python Input

```
%matplotlib inline
import matplotlib.pyplot as plt

from PIL import Image
from matplotlib import patches
from io import BytesIO

response = requests.get(image_url)
image = Image.open(BytesIO(response.content))

plt.figure(figsize=(8,8))
ax = plt.imshow(image, alpha=0.6)
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')
    ax.axes.add_patch(p)
    plt.text(origin[0], origin[1], "%s, %d"%(fa["gender"].capitalize(),
fa["age"]), fontsize=20, weight="bold", va="bottom")
_ = plt.axis("off")
```

Here are more images that can be analyzed using the same technique. First, define a helper function, `annotate_image` to annotate an image given its URL by calling into the Face API.

Python Input

```
def annotate_image(image_url):
    response = requests.post(face_api_url, params=params, headers=headers,
json={"url": image_url})
    faces = response.json()

    image_file = BytesIO(requests.get(image_url).content)
    image = Image.open(image_file)

    plt.figure(figsize=(8,8))
    ax = plt.imshow(image, alpha=0.6)
    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')
        ax.axes.add_patch(p)
        plt.text(origin[0], origin[1], "%s, %d"%(fa["gender"].capitalize(),
fa["age"]), \
                 fontsize=20, weight="bold", va="bottom")
    plt.axis("off")
```

You can then call `annotate_image` on other images. A few examples samples are shown below.

Python Input

```
annotate_image("https://how-old.net/Images/faces2/main001.jpg")
```

Python Input

```
annotate_image("https://how-old.net/Images/faces2/main002.jpg")
```

Python Input

```
annotate_image("https://how-old.net/Images/faces2/main004.jpg")
```

API Reference

The Microsoft Face API is a cloud-based API that provides the most advanced algorithms for face detection and recognition. The reference document can be found in Microsoft Cognitive Services (formerly Project Oxford) Dev Portal.

Face APIs cover the following categories:

- LargePersonGroup Person Management APIs: Used to manage LargePersonGroup Person Faces for Identification.
- LargePersonGroup Management APIs: Used to manage a LargePersonGroup dataset for Identification.
- LargeFaceList Management APIs: Used to manage a LargeFaceList for Find Similar.
- PersonGroup Person Management APIs: Used to manage PersonGroup Person Faces for Identification.
- PersonGroup Management APIs: Used to manage a PersonGroup dataset for Identification.
- FaceList Management APIs: Used to manage a FaceList for Find Similar.

Face Algorithm APIs: Covers core functions such as Detection, Find Similar, Verification, Identification, and Group.