

프로젝트 명세서

Google Colab 기반의
Face Recognition 입문 PJT

목차

1. 프로젝트 개요.....	3
2. 기본과제	4
3. 심화과제	10
4. 과제 제출방법.....	18

1. 프로젝트 개요

본 프로젝트는 별도의 설치과정 없이 웹 브라우저 상에서 Python 코드를 작성하고 실행 할 수 있는 Google Colab 을 실습해 보고 Colab 상에서 얼굴 인식 등 몇 가지 재미있는 결과물을 만들어 보는 내용을 담고 있습니다.

본 프로젝트의 목표는 다음과 같습니다.

1. Google Colab 의 기본적인 사용법을 익혀봅니다.
2. Google Colab 을 통해 Python 코드를 GPU 가속을 적용해 실행해보고 Google Drive 와 연동해 수행결과를 읽고 쓸 줄 알게 됩니다.
3. OpenCV 라이브러리가 무엇인지 가볍게 살펴보고 Colab 상에서 간단한 OpenCV API 를 사용하여 이미지 처리를 하는 기본적인 방법을 경험 합니다.
4. 얼굴인식과 관련된 (가장 쉬운) 최상위 레벨의 패키지인 face_recognition 패키지를 사용하여 간단하게 얼굴인식 기능 개발을 경험해 봅니다.
5. 기존에 사용해 보지 못한 새로운 라이브러리 / 패키지를 레퍼런스 문서를 보고 적용하는 방법을 경험해 봅니다.

Python 을 한번도 사용해보지 않은 전공학생들은 다음의 링크를 참조하여 Java 와의 차이점을 빠르게 습득해보시기 바랍니다. 코드량이 많지 않으니 부담없이 시도해 보세요.

 <https://haloper.tistory.com/32>

2. 기본과제


2-1. Google Colab

Python 을 학습하는 과정에서 웹 브라우저 상에서 소스코드를 작성하고 한 줄씩 실행 및 디버깅을 한 후 문서로 저장을 할 수 있는 Jupyter Notebook 을 많이 사용하셨을 겁니다.

Colab 은 Jupyter Notebook 이 실행되는 PC 의 로컬 리소스를 사용하는 것과 달리 소스코드를 Google 의 클라우드 컴퓨팅 환경에서 제공되는 CPU / GPU / TPU 를 사용해 실행시킬 수 있고 소스코드나 데이터를 Google Drive 를 통해 불러오거나 저장할 수도 있는 개발 환경 입니다. Cloud 기반이므로 별도의 설치과정이 필요 없으며 딥러닝, M/L, 데이터 사이언스 분야에서 매우 널리 사용되고 있습니다.

Google Colab(oratory)

- For you Jupyter Notebook fans, this is an even better option!



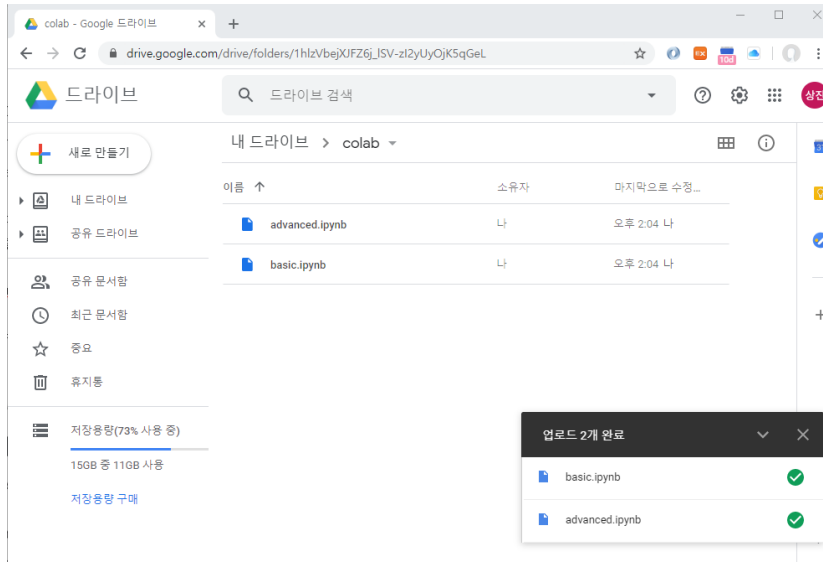
- 1) Hosted by Google
 - a) Extremely fast network speed
- 2) Access to GPU and TPU (Tensor Processing Unit)
 - a) Not something you can buy for your PC!
 - b) TF code works the same on all devices
- 3) Stored in Google Drive (the "cloud")
 - a) You'll never lose it, and it's easy to share
- 4) Many libraries for deep learning / machine learning / data science
 - a) More than I assumed there would be! (Theano, PyTorch)

다음의 링크를 참조하여 Colab 에 대해 학습해 보시기 바랍니다.

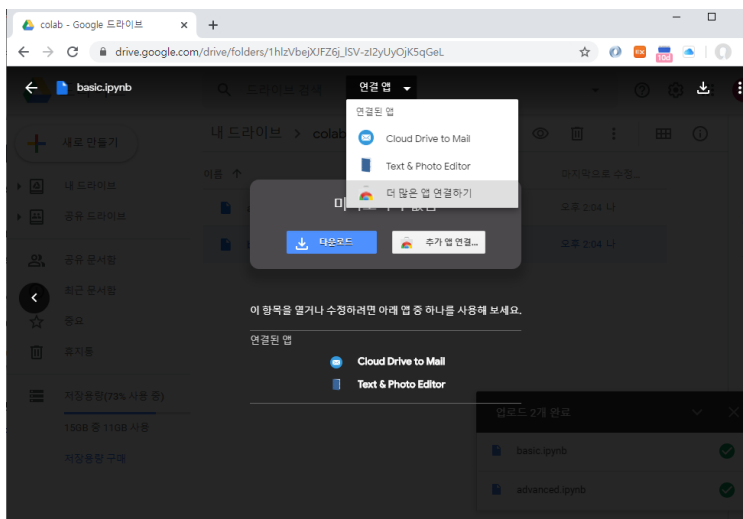
 <https://colab.research.google.com/>

2-2. Google Drive 의 ipynb 파일과 Colab 연결하기

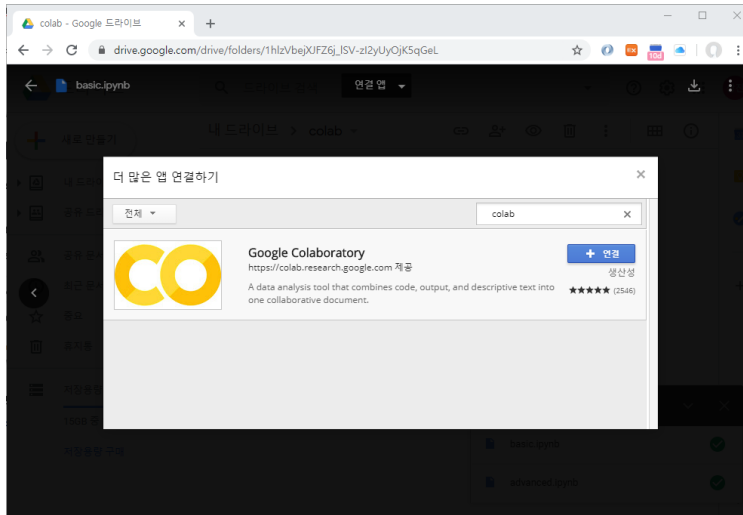
Google Drive 에 colab 이라는 디렉토리를 생성한 후, 전달받은 basic.ipynb 와 advanced.ipynb 파일을 업로드 합니다.



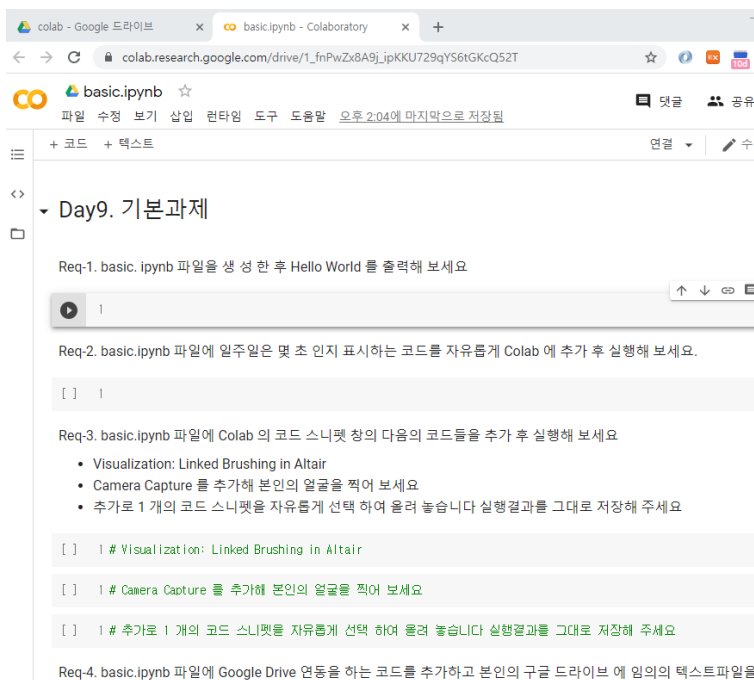
이후, 업로드 된 basic.ipynb 을 더블클릭 한 후 상단의 “연결 앱” 버튼을 누르면 다음과 같이 “더 많은 앱 연결하기” 를 선택 할 수 있습니다.



이후 다음과 같이 검색창에서 “colab” 을 검색하고 “연결” 버튼을 누르면 구글 드라이브의 ipynb 파일을 colab 과 연결 할 수 있습니다.




연결 완료후 ipynb 파일을 더블클릭하면 다음과 같이 Colab 으로 연동이 됩니다.



Req-1	제공된 basic.ipynb 파일상에서 Hello World 를 출력해 보세요.
-------	--

Req-2	basic.ipynb 파일에 일주일은 몇 초 인지 표시하는 코드를 자유롭게 Colab 에 추가 후 실행해 보세요.
-------	---

2-2. Google Colab - 코드 스니펫

Colab 에서는 재사용이 가능한 코드와 다양한 예제가 좌측의  모양의 아이콘을 클릭하면 나옵니다. 이를 코드 스니펫 창이라고 합니다. Req-3 을 수행해 봅니다.

Req-3	<p>basic.ipynb 파일에 Colab 의 코드 스니펫 창의 다음의 코드들을 추가 후 실행해 보세요.</p> <ul style="list-style-type: none"> - Visualization: Linked Brushing in Altair - Camera Capture 를 추가해 본인의 얼굴을 찍어 보세요. - 추가로 1 개의 코드 스니펫을 자유롭게 선택하여 올려 놓습니다. <p>실행결과를 그대로 저장해 주세요.</p>
-------	---


2-3. Google Colab - Google Drive 연동

Colab 에서는 간단한 인증과정을 거쳐 Google Drive 와 연동을 하여 데이터를 읽어오거나 연산결과를 저장 할 수 있습니다.

Req-4	basic.ipynb 파일에 Google Drive 연동을 하는 코드를 추가하고 본인의 구글 드라이브에 임의의 텍스트파일을 저장하는 코드를 작성해보세요. 코드 스니펫의 Mounting Google Drive in your VM 을 사용해도 무방합니다.
-------	--

2-4. Colab 에서 GPU / TPU 사용

Colab 에서는 Google 에서 제공하는 GPU 와 TPU 를 몇가지 제약사항이 있지만 무료로 사용해 코드를 실행 할 수 있습니다.


Req-5	<p>다음의 GPU 가속 예제코드를 실행해 보고 CPU 사용시 처리시간과 GPU 사용시 처리시간을 비교해 보고 마지막 블럭인 “Observe TensorFlow speedup on GPU relative to CPU”의 결과를 복사해서 basic.ipynb 의 Req-5 다음에 텍스트 형태로 붙여 주세요.</p> <p> https://colab.research.google.com/notebooks/gpu.ipynb</p>
-------	---

2-5. Colab 에서 OpenCV 라이브러리를 사용한 그래픽 처리

OpenCV 는 무료로 공개된 컴퓨터 비전 라이브러리로서 대부분의 OS 를 지원하며 C/C++ 프로그래밍 언어로 개발 되었으나 Python, Java 및 MATLAB 에 바인딩 된 인터페이스가 있어 Windows, Linux, Android 및 MacOS 를 지원합니다. 영상 관련 라이브러리로서 사실상 표준의 지위를 가지고 있다고 할 수 있으며 조금이라도 영상처리가 들어간다면 필수적으로 사용하게 되는 라이브러리 입니다.

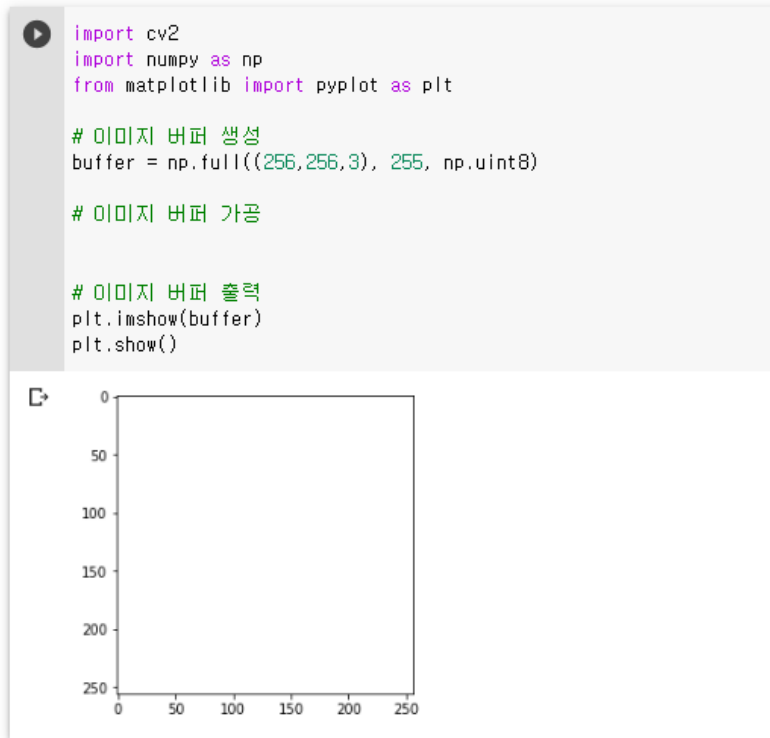
라이브러리에는 2500 개가 넘는 최적화 된 알고리즘이 있으며 여기에는 클래식 및 최신 컴퓨터 비전과 머신 러닝 알고리즘이 모두 포함됩니다. 이 알고리즘은 얼굴을 감지하고 인식하고, 물체를 식별하고, 비디오에서 사람의 행동을 분류하고, 카메라 움직임을 추적하고, 움직이는 물체를 추적하고, 물체의 3D 모델을 추출하고, 스테레오 카메라에서 3D 포인트 클라우드를 생성하고, 이미지를 연결하여 고해상도를 생성하는 데 사용될 수 있습니다.

본 명세서에서는 OpenCV 의 기능 중 단순히 인식한 얼굴에 사각형을 그리는 수준의 Drawing API 만 사용합니다. 따라서 잠시 언급하고 넘어가는 수준 입니다만, OpenCV 는 여러모로 활용도가 높은 라이브러리 이므로 어떤 기능들이 있는지 다음의 링크들을 참조하여 OpenCV 에 대해서 살펴보고 추후 프로젝트 진행시 활용해 보시기 바랍니다.

 <https://opencv.org/> - OpenCV 공식 사이트

 https://docs.opencv.org/master/d6/d00/tutorial_py_root.html - OpenCV 파이썬 튜토리얼

Colab에서는 OpenCV로 가공된 그래픽을 matplotlib를 사용해 이미지 버퍼 출력 부분을 바꿔 사용하는 방식으로 사용됩니다. 다음처럼 생성된 buffer에 cv2를 사용해 필요한 이미지 처리를 하신 후 plt.imshow() 함수와 plt.show()를 통해 간단히 화면에 출력 할 수 있습니다.



matplotlib는 위와 같이 좌표가 나오기 때문에 처리결과를 표시하기 좋습니다.

Req-6	basic.ipynb 파일에 matplotlib와 OpenCV의 Drawing API를 이용하여 화면에 여러 가지 도형을 그려주는 코드를 작성해 보세요. (난이도: 쉬움)
-------	---

3. 심화과제

우리가 요즘 사용하는 스마트폰에는 사용자 앨범의 사진들을 Deep Learning 기술을 사용해 탐색하여 사람의 얼굴인지 여부를 판독한 후, 사용자로부터 이름을 입력 받으며 약간의 학습과정을 거쳐 인물별로 자동으로 분류를 해주는 기능들이 들어가 있습니다.

이런 기능들을 구현하기 위해서는 가장 기본적으로 필요한 기술이 사진에서 얼굴을 찾아내 얼굴이 표시된 영역을 알아내야 하는데 이를 Face Detection 이라고 하고, 감지된 얼굴데이터를 기반으로 동일한 인물을 찾아내 인식하는 것을 Face Recognition 즉, 얼굴인식 이라고 합니다.

오늘 교육생 여러분들은 GPU 및 TPU 가속이 제공되는 Google Colab 상에서 Python 기반의 face_recognition 패키지를 사용하여 여러 가지 얼굴인식 모델을 경험해보고 이를 활용하여 입문수준의 간단한 과제를 수행해 보겠습니다.



이번 장에서는 여러 사람이 들어가 있는 사진에서 사람의 얼굴을 감지해서 사각형을 그리는 코드를 구현해 보고, 얼굴을 학습해서 새로운 사진이 들어왔을 때 학습된 데이터의 사진을 비교해서 누군지 알려주는 코드를 구현해 보겠습니다.

3-1. Face Detection

얼굴검지와 인식을 위해 우리는 Dlib 기반의 face-recognition 1.3.0 패키지를 사용합니다.

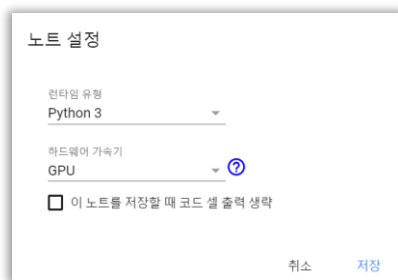
다음 URL 을 참조하여 어떤 패키지인지 확인을 합니다.

👉 <https://pypi.org/project/face-recognition/>

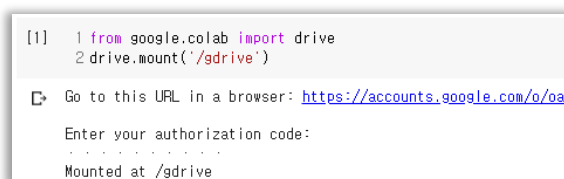
 https://github.com/ageitgey/face_recognition

Req-7	다음의 과정을 따라 advanced.ipynb 에 Face Detection 을 구현해 보세요.
-------	---

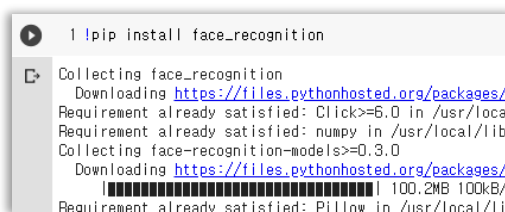
Google Colab 에 advanced.ipynb 파일을 생성한 후 런타임유형을 다음과 같이 Pyrhon3 / GPU 로 변경합니다.



다음과 같이 코드스니펫의 Google Drive 연동코드를 추가한 후 링크를 클릭하여 Authorization code 를 복사해서 붙여 넣어 줍니다.



다음처럼 pip 명령어를 사용하여 face_recognition package를 설치합니다.



여러 사람의 얼굴이 들어있는 사진 한 장을 준비해서 구글 드라이브에 올려놓습니다.
이미지파일이 업로드 된 Google Drive 의 경로명, 디렉터리명, 파일명을 설정해줍니다.

```
1 import cv2, os
2 import face_recognition as fr
3 from IPython.display import Image, display
4 from matplotlib import pyplot as plt

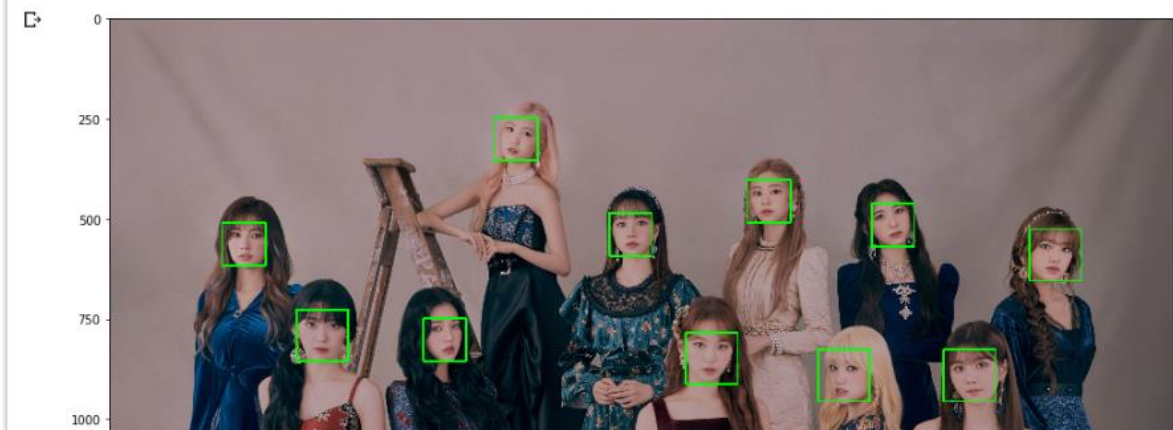
[40] 1 image_path = "/gdrive/My Drive/colab/gg01.jpg"
```

다음처럼 face_recognition 패키지의 기본 HOG (Histogram of Oriented Gradient) 모델을
사용해 얼굴감지를 하고 사각형을 그려 줍니다.

```
[48] 1 image = fr.load_image_file(image_path)
      2 face_locations = fr.face_locations(image)

[49] 1 for (top, right, bottom, left) in face_locations:
      2     cv2.rectangle(image, (left, top), (right, bottom), (0,255,0), 3)

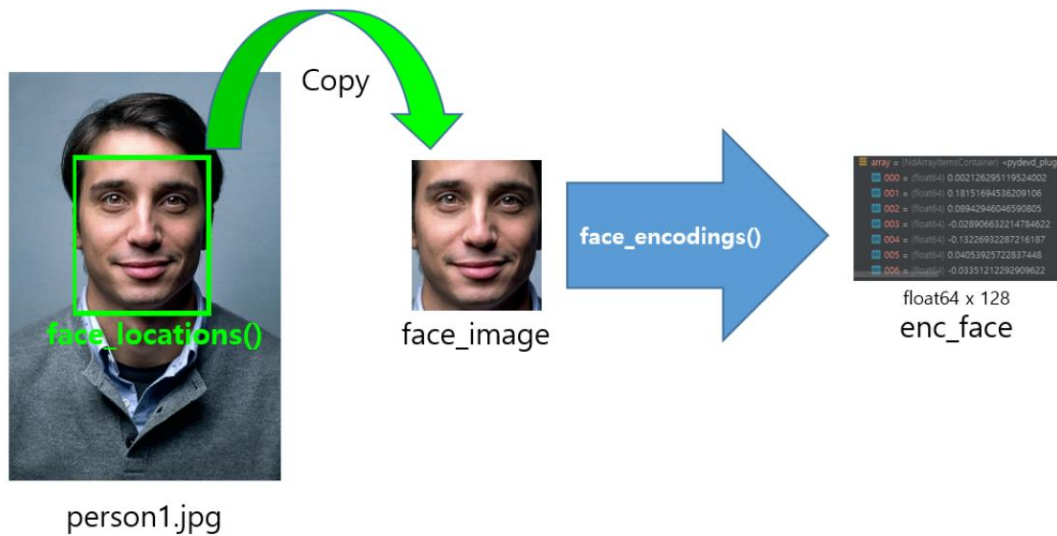
1 # 이미지 버퍼 출력
2 plt.rcParams["figure.figsize"] = (16,16)
3 plt.imshow(image)
4 plt.show()
```



입력한 사진에 위와 같이 초록색 사각형이 그려졌다면 face_recognition 패키지의
HOG 모델을 사용해 Face Detection 을 정상적으로 완료 했습니다.

3-2. Face Recognition

지난 과정을 통해 우리는 `face_locations()` 함수를 사용하여 사진에서 얼굴을 찾는 기능을 구현하였습니다. 이번에는 몇장의 인식된 얼굴영역에서 데이터를 추출하여 여러 사진들이 동일한 사람의 사진인지를 확인하는 방법을 알아 보겠습니다.





위와 같이 `face_recognition` 패키지에서는 인물사진에서 인식한 얼굴영역을 복사하여 라이브러리에서 사용하는 데이터구조로 encoding 을 한 후 이 encoding 된 데이터를 이용하여 동일한 여부를 확인 합니다.



즉, 2 장의 사진에서 encoding 된 2 개의 얼굴데이터를 `face_distance()`의 파라미터로

전달하여 두 사진간의 distance 를 얻어내고, distance 값이 0.6 이상이면 타인으로 볼 수 있고 0.6 미만이면 동일인으로 볼 수 있습니다. 좀 더 엄격한 기준으로 동일인 인식을 하고 싶으면 수치를 조금 낮춰 distance 0.5를 기준으로 로직을 작성하면 됩니다.

<p>Req-8</p>	<p>다음처럼 각기 다른 인물사진 3 장과 동일한 인물사진 2 장을 준비합니다.</p> <div style="display: flex; justify-content: space-around; align-items: flex-end;"> <div style="text-align: center;">  person1 </div> <div style="text-align: center;">  person2 </div> <div style="text-align: center;">  person3 </div> </div> <div style="display: flex; justify-content: space-around; align-items: flex-end; margin-top: 10px;"> <div style="text-align: center;">  person4 </div> <div style="text-align: center;">  unknown </div> </div> <p>person1 ~ person4 까지 4 명의 인물사진에서 얼굴을 감지한 후 해당영역을 face_encoding() 하여 보관합니다.</p> <p>이후 새로운 인물사진 unknown 을 입력 받아 기존 4 명의 얼굴 중 동일인을 찾는 코드를 다음을 참고해 advanced.ipynb 에 추가 후 실행해 보세요.</p>
--------------	--

4 장의 인물사진을 준비해서 구글 드라이브에 업로드 후 다음처럼 이미지파일을 읽어서 known_person_list 에 추가를 합니다.

```

1 plt.rcParams["figure.figsize"] = (1,1)
2
3 # 이미지 파일을 로드하여 known_person_list 리스트 생성
4 known_person_list = []
5 known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/person1.jpg"))
6 known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/person2.jpg"))
7 known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/person3.jpg"))
8 known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/person4.jpg"))
9

```

다음처럼 face_locations() 함수를 사용해 4 장의 사진에서 얼굴을 감지해 잘라내어서 known_face_list 에 저장합니다.

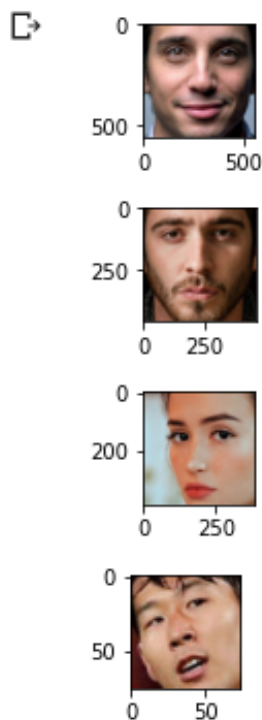
```
# 얼굴을 인식을 하여 감지된 부분을 잘라낸 다음 known_face_list에 저장
known_face_list = []
for person in known_person_list:

    # 얼굴좌표를 알아내서 잘라낸다
    top, right, bottom, left = fr.face_locations(person)[0]
    face_image = person[top:bottom, left:right]

    # known_face_list에 잘라낸 face_image를 저장
    known_face_list.append(face_image)
```

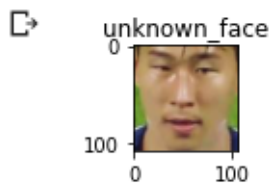
known_face_list 를 출력해보면 다음과 같이 얼굴들이 저장되어 있습니다.

```
20
21 # known_face_list에 저장된 얼굴들 출력
22 for face in known_face_list:
23     plt.imshow(face)
24     plt.show()
```



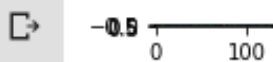
unknown.jpg 도 얼굴인식을 한 후, 다음처럼 잘라서 unknown_face 에 저장합니다.

```
[11] 1 # 기존리스트에 없는 새로운 파일을 열어서
      2 unknown_person = fr.load_image_file("/gdrive/My Drive/colab/unknown.jpg")
      3
      4 # 얼굴좌표를 알아내서 잘라낸다
      5 top, right, bottom, left = fr.face_locations(unknown_person)[0]
      6 unknown_face = unknown_person[top:bottom, left:right]
      7
      8 # unknown_face 이라는 타이틀을 붙여서 표시
      9 plt.title("unknown_face")
     10 plt.imshow(unknown_face)
     11 plt.show()
```



이후, face_encoding() 함수를 사용해 enc_unknown_face 에 얼굴영역을 인코딩해 저장합니다.

```
[ ] 1 # unknown_person_face를 인코딩
      2 enc_unknown_face = fr.face_encodings(unknown_face)
      3
      4 # 화면에 표시해보면 다음과 같다
      5 plt.imshow(enc_unknown_face)
      6 plt.show()
```



enc_unknown_face 에는 인코딩된 데이터가 저장되므로 plt.imshow(enc_unknown_face) 후 plt.show()를 해보면 이미지 Bitmap 데이터가 아니어서 이미지가 출력되지 않거나 오류가 발생한다. 이 과정은 이미지 데이터가 들어가 있지 않은것을 확인하는데 목적이 있으므로 이를 확인해 보고 해당부분을 주석처리 후 다음과정을 진행 합니다.

다음과 같이 face_distance() 함수를 사용해 known_face_list 에 저장된 기존 4 명의 얼굴과 새로 입력된 unknown_face 의 distance 를 구합니다.

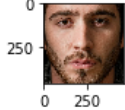
일반적으로 distance 가 0.6 이상이면 타인이라고 볼 수 있으나 다른 사람이라도 비슷한 사람은 0.5 대 distance 가 나오기도 합니다.

```
[ ] 1 # 등록된 얼굴리스트를 비교
    2 for face in known_face_list:
    3
    4 # 등록된 얼굴을 128-dimensional face 인코딩
    5 enc_known_face = fr.face_encodings(face)
    6
    7 # 등록된 얼굴과 새로운 얼굴의 distance를 얻기
    8 distance = fr.face_distance(enc_known_face, enc_unknown_face[0])
    9
    10 # distance 수치를 포함한 얼굴 출력
    11 plt.title("distance: " + str(distance))
    12 plt.imshow(face)
    13 plt.show()
```

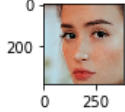
distance: [0.83117966]



distance: [0.83499745]



distance: [0.82744728]



distance: [0.36155637]



상기와 같은 경우 distance가 0.36155637이 나온 4번째 사진이 unknown과 동일인으로 추정됩니다.

오늘 수행한 과제를 바탕으로 Python 기반의 이미지 처리방법이나 얼굴 인식 등 혹은 Machine Learning, Deep Learning 등에 흥미가 생겨 해당 분야의 학습을 시작하게 되는 계기가 되시길 바랍니다.

4. 과제 제출방법

다음 제출방법을 잘 읽고 가이드에 따라 과제를 제출 하시기 바랍니다.

- 1) Local Git Repo 의 다음 2 개의 새 노트파일들을 명세서를 참고하여 Google Drive 에 업로드 합니다.
 - {PROJECT_ROOT}/self-project/con10/rep03/**basic.ipynb**
 - {PROJECT_ROOT}/self-project/con10/rep03/**advanced.ipynb**
- 2) 업로드 된 basic.ipynb 을 Colab 으로 열어서 Req-1 ~ 6 를 구현 후 실행 및 저장 합니다.
- 3) 업로드 된 advanced.ipynb 을 Colab 으로 열어서 Req-7 ~ 8 을 구현 후 실행 및 저장 합니다.
- 4) 실행 결과가 저장된 basic.ipynb 와 advanced.ipynb 파일을 다운로드 받아 다음의 Local Git Repo 에 덮어쓰기를 합니다.
 - {PROJECT_ROOT}/self-project/con10/rep03/**basic.ipynb**
 - {PROJECT_ROOT}/self-project/con10/rep03/**advanced.ipynb**
- 5) commit / push 하여 제출 합니다.
- 6) 제출경로
<https://lab.ssafy.com/s10-study/self-project/>의 “산출물 제출 가이드” .docx 참조