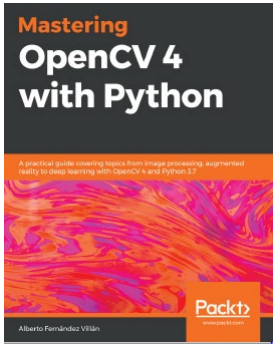
Face Recognition

-dlib 중심으로..(교재 11장)



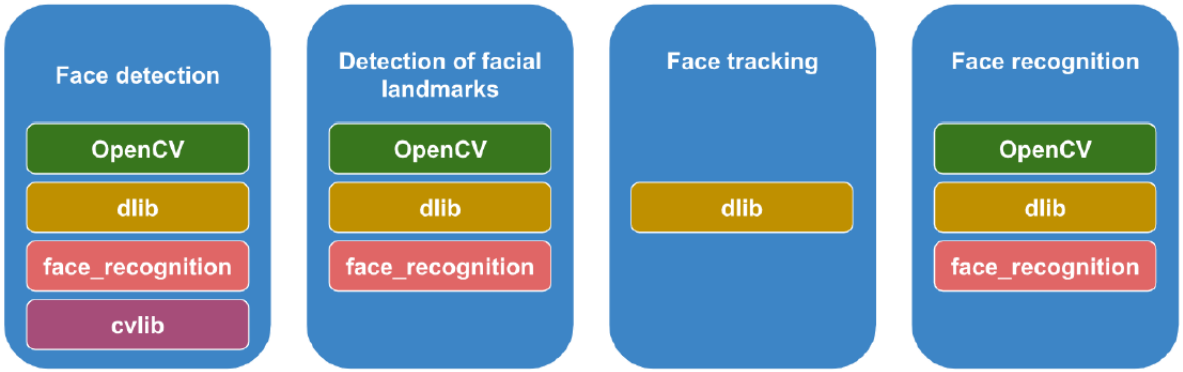
Mastering OpenCV 4 with Python, Alberto, Packt, Pub. 2019

2024년 1학기

서경대학교 김진헌

차례

1. 얼굴 영상 처리 기술의 장르



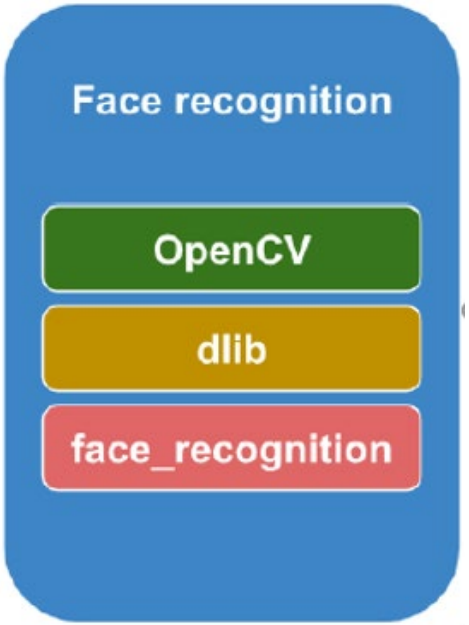
- Face detection
  - 얼굴의 위치와 크기 검출
- Detection of facial landmark
  - 얼굴의 주요 부위 위치/크기 검출: 눈, 입, 코, 뺨 등..
- Face tracking
  - 움직이는 얼굴의 위치와 크기 검출
- Face recognition
  - **Face identification(1: N): 등록된 얼굴 중에 누구인지를 맞추는 처리**
    - **응용 - 출입통제 시스템(Access Control)**
  - Face verification(1:1): 자신이라고 주장하는 사람이 맞는지 맞추는 처리 => ATM

다루고자 하는 기술

2. 얼굴 인식 함수

얼굴인식 기법 개요 링크

Pyramidsearch face recognition with OpenCV, Python and deep learning((2021 update)



- OpenCV
  - Eigenfaces
  - Fisherfaces
  - Local Binary Patterns Histograms(LBPH)
    - 조명조건을 고려한 설계로 실제 상황에서 3개의 접근 중 가장 우수. 새로운 얼굴을 등록하는 update() 메소드를 지원.

소개할 내용

- Dlib(documentation and API reference, github)
  - DNN을 이용하여 얼굴을 128개의 벡터로 수치화. 미지의 얼굴을 128 벡터와의 유클리디안 거리로 동일 인물을 판단.
- Face_recognition
  - dlib의 인코딩과 face 비교 함수를 사용한다.

~~2.1 Programming with OpenCV(생략)~~

- 3가지(Eigenfaces, Fisherfaces, LBPH) 기술 모두 공통으로 recognizer를 생성하는 것으로 시작한다.

```
face_recognizer = cv2.face.LBPHFaceRecognizer_create()
face_recognizer = cv2.face.EigenFaceRecognizer_create()
face_recognizer = cv2.face.FisherFaceRecognizer_create()
```

- 다음 2가지 메소드로 학습과 테스팅을 시행한다.
  - train() and predict()
  - face_recognizer.train(faces, labels)
    - LBPH의 경우: cv2.face_FaceRecognizer.update(src, labels)
      - 학습이 완료된 모델에 새로운 데이터를 기반으로 추가 학습
  - label, confidence = face_recognizer.predict(face)
- 학습된 모델을 저장하거나 로드하는 함수: write() and read() methods
  - cv2.face_FaceRecognizer.write(filename)
  - cv2.face_FaceRecognizer.read(filename)

2.2 Programming with dlib

1_encode_face_dlib.py

- Deep learning 기반으로 고품질 얼굴인식을 지원.
  - 실세계의 데이터베이스에 대해 99.38%의 정확도를 구현
- ResNet 34 뉴럴 모델을 기반으로 300만개의 얼굴에 대해 학습(21.4MB): 모델 다운로드
- 얼굴 특징을 나타내는 128 차원의 descriptor 생성. 유클리디언 거리가 0.6이하이면 같은 사람으로 간주한다. 모델 자체가 다른 사람과 구분되는 얼굴의 특징벡터를 출력하므로 사전에 얼굴 인식 모델을 학습할 필요가 없다.
- 학습과정 요약
  - Triplets 단위로 학습: 3개의 영상 단위로 학습. 그중 2개는 같은 사람. 128개의 벡터가 같은 사람에 대해서는 가까워지도록 하고 이 같은 사람의 벡터들이 다른 사람에 대해서는 각각 멀어지도록 학습시킨다.
  - 이러한 학습을 수천명의 사람에 대해서 수백만개의 영상에 대해 반복하여 학습시킨다.
  - 그 결과 생성되는 디스크립터는 다음 특징을 갖게 된다.

    - The generated 128D descriptors of two images of the same person are quite similar to each other.
    - The generated 128D descriptors of two images of different people are very different.

- 활용 방법
  - 1) 인식하고자 하는 인물들의 영상을 이용해 사전에 각 인물들의 descriptor들을 확보한다.
    - 미지의 인물에 대한 디스크립터를 구하여 이 벡터와 기존에 확보하고 있는 인물의 벡터와의 비교를 통해 어떤 인물인지 판단한다.
    - 판단 방법은 간단하게는 유클리디언 거리를 사용할 수 있다. ➔ SVM, KNN과의 비교. 설계자의 역량 판단과 역량
  - 2) 인식하고자 하는 인물에 대한 dlib 디스크립터를 구한 후 동일인인지 판별하고자하는 인물에 대한 디스크립터를 구해 두 디스크립터의 거리가 일정 기준이하이면 동일인으로 처리한다.

3. dlib identification 실험 결과

6 수행결과 먼저 보이기: 실험 세트 0

2_compare_faces_dlib.py

파일 이름과 미지 인물과의 유클리디언 거리

미지 인물 파일 이름 및 자기 자신과의 유클리디언 거리

Euclidean Distance 가 가장 멀다.

미지 인물 사진

요약
1) 각 얼굴에 대해 128개의 원소로 이루어진 dlib 1차원 face descriptor를 구한다.
2) 미지의 인물에 대해 나머지 알고 있는 인물에 대한 디스크립터들 간의 유클리디언 거리를 각각 구한다.
3) 0에 가까우면 동일인물 사진, 1에 가까우면 타인이다. 일반적으로 0.6이하면 동일 인물로 취급한다.

```
8) computed_distances_ordered, ordered_names
 = compare_faces_ordered(known_encodings, names, unknown_encoding)
name              =      jared_3      jared_1      jared_2        obama
ordered distance  =    0.3913190    0.3998328    0.4104154    0.9053702
The unknown person is identified as 'jared_3.jpg'.
```

유클리디언 거리가 커지는 순으로 정렬한 결과

4. dlib Identification 프로그램 흐름

7 프로그램의 흐름(1/5) 2_compare_faces_dlib.py

```
# landmark 검출기(shape predictor-검출기)를 다운로드하고 그 객체(callable object)를 생성한다.
pose_predictor_5_point = dlib.shape_predictor(model_path + "shape_predictor_5_face_landmarks.dat")
# face enconder를 다운로드하고, 그 객체(callable object)를 생성한다.
face_encoder = dlib.face_recognition_model_v1(model_path + "dlib_face_recognition_resnet_model_v1.dat")
# 얼굴 검출기 객체(callable object)를 생성한다.
detector = dlib.get_frontal_face_detector()     # dlib hog face detector을 사용하였음.
```

이 함수로 128개의 벡터로 이루어진 face descriptor들을 구하는 encoding 작업을 수행한다.

```
def face_encodings(face_image, number_of_times_to_upsample=1, num_jitters=1):
    # Returns the 128D dlib face descriptor for each face in the image
    gray = cv2.cvtColor(face_image, cv2.COLOR_RGB2GRAY)
    face_locations = detector(gray, number_of_times_to_upsample) # 그레이 영상으로 얼굴을 검출한다.

    # 주어진 얼굴의 위치로 5 지점의 landmark를 검출한다.
    raw_landmarks = [pose_predictor_5_point(face_image, face_location) for face_location in face_locations]
                    랜드마크 다섯 개 점이 인코딩하기 위한 정보로 활용된다.
    # Calculate the face encoding for every detected face using the detected landmarks for each one:
    return [np.array(face_encoder.compute_face_descriptor(face_image, raw_landmark_set, num_jitters)) for
            raw_landmark_set in raw_landmarks]
```

```
# face_encodings은 검출된 인물들의 descriptor 정보를 각 사람마다 ndarray 데이터로 만들어
# 리스트 자료형으로 반환한다.
```

프로그램의 흐름(2/5)


```
# 1) 실험 데이터 선택: 다음 4세트 중의 하나만 주석문을 해제하시오. 파일 확장자는 jpg를 가정한다.
# 선택된 사람이 위 4인 중에서 누구와 가장 가까운 것인가를 128차원 descriptor 정보로 결정한다.
data_path = "face_files/"    # 영상 파일의 위치
set0 = ["jared_1.jpg", "jared_2.jpg", "jared_3.jpg", "obama.jpg", "jared_4.jpg"]
set1 = ["jared_1.jpg", "jared_2.jpg", "jared_3.jpg", "obama.jpg", "obama2.jpg"]
set2 = ["obama2.jpg", "obama3.jpg", "obama4.jpg", "obama5.jpg", "obama6.jpg"]
set3 = ["obama2.jpg", "obama3.jpg", "obama5.jpg", "obama6.jpg", "obama4.jpg"]
data_set_list = [set0, set1, set2, set3]
set_num = 0
names = data_set_list[set_num]
print(f"사용된 영상 실험 세트={set_num}, 미지의 인물 파일명={names[4]}")
```

```
사용된 영상 실험 세트=0, 미지의 인물 파일명=jared_4.jpg
```

encodings: 신원을 아는 영상(4개)의 디스크립터
encoding_to_check: 신원을 모르는 미지의 영상(1개)의 디스크립터

```
# 2) 미리 누구인지 아는 사람들의 사진을 순서대로 읽어들인다.
# 맨 마지막 사진은 test 얼굴 영상이다.
# 이것과 나머지 사진들의 fcae descreptor 비교를 행한다.
known_image_1 = cv2.imread(data_path + names[0])
known_image_2 = cv2.imread(data_path + names[1])
known_image_3 = cv2.imread(data_path + names[2])
known_image_4 = cv2.imread(data_path + names[3])

# 3) 모르는 인물의 사진을 읽어들인다.
unknown_image = cv2.imread(data_path + names[4])
```

```
# 4) from BGR(OpenCV format) to RGB(dlib format):
known_image_1 = known_image_1[:, :, ::-1]
known_image_2 = known_image_2[:, :, ::-1]
known_image_3 = known_image_3[:, :, ::-1]
known_image_4 = known_image_4[:, :, ::-1]
unknown_image = unknown_image[:, :, ::-1]
```

프로그램의 흐름(3/5)


```
# 5) Create the encodings for both known images & unknown image:
known_image_1_encoding = face_encodings(known_image_1)[0]
known_image_2_encoding = face_encodings(known_image_2)[0]
known_image_3_encoding = face_encodings(known_image_3)[0]
known_image_4_encoding = face_encodings(known_image_4)[0]
known_encodings = [known_image_1_encoding, known_image_2_encoding,
                   known_image_3_encoding, known_image_4_encoding]
print(f"5) type(known_encodings)={type(known_encodings)}, len(known_encodings)={len(known_encodings)}")
unknown_encoding = face_encodings(unknown_image)[0]          # 여러 사람일 때는 index 번호를 바꿀 수 있다.
print(f"type(unknown_encoding)={type(unknown_encoding)}, unknown_encoding.shape={unknown_encoding.shape}")

all_encodings = known_encodings + [unknown_encoding]     # 모르는 인물의 인코딩까지 포함한 5인의 인코딩 생성
print(f"type(all_encodings)={type(all_encodings)}, len(all_encodings)={len(all_encodings)}")
```

face_encodings은 128개의 원소를 갖는 ndarray 데이터들을 검출된 얼굴 수에 따라 리스트 자료형으로 반환하는데 그중 0번째를 반환 받는다. 0번째 얼굴이 관심 얼굴인 것으로 가정한다. 4개의 영상 파일은 편의상 1개의 얼굴만이 존재하는 것을 사용하였다.

```
5) type(known_encodings)=<class 'list'>, len(known_encodings)=4
type(unknown_encoding)=<class 'numpy.ndarray'>, unknown_encoding.shape=(128,)
type(all_encodings)=<class 'list'>, len(all_encodings)=5
```

프로그램의 흐름(4/5)


```python
def compare_faces(encodings, encoding_to_check):
    # Returns the distances when comparing a list of face encodings against a candidate to check
    # 입력:
    #   encodings: 누군지 알고 있는 얼굴에 대해 dlib 함수로 적용하여 추출한 128차원의 face descriptor들의 list 자료
    #   encoding_to_check: 비교하고자 하는 신원 미상의 dlib face descriptor
    # 반환값:
    #   encodings list에 있는 여러 개의 encoding과 1개의 encoding_to_check의 인코딩 값을
    #   각 차원별로 뺀 유클리디언 거리를 리스트로 반환한다.
    #   작을 수록 encoding_to_check의 얼굴과 가깝다.
    return list(np.linalg.norm(encodings - encoding_to_check, axis=1))
    # linalg.norm 링크: 두 디스크립터간의 차이에 대한 norm을 계산한다.

    # If axis is an integer, it specifies the axis of x along which to compute the vector norms
```

encodings: 신원을 아는 영상(4개)의 디스크립터 리스트
encoding_to_check: 신원을 모르는 미지의 영상(1개)의 디스크립터

```python
# 6) Compare faces: 5개의 얼굴 디스크립터와 맨 마지막 디스크립터와의 유클리디언 거리를 계산한다.
# 유클리디언 거리가 0.6이하면 동일 인물로 본다.
# all_encodings[-1]은 미지의 얼굴의 인코딩(unknown_encoding)이다.
computed_distances = compare_faces(all_encodings, all_encodings[-1])
```

```
7) computed_distances = compare_faces(all_encodings, unknown_encoding)
name               =      jared_1      jared_2      jared_3        obama      jared_4
matching distance =    0.3998328    0.4104154    0.3913190    0.9053702    0.0000000
```

유클리디언 거리값을 작은 값부터 큰 값 순으로 sorting한 결과 반환하는 함수

```
def compare_faces_ordered(encodings, face_names, encoding_to_check):
    # 위와 같은 함수인데 반환할 때 norm의 값을 크기 순으로 소팅(작은 값부터..)하여 반환한다.
    # distances: 매칭값 순으로 작은 값부터 나열하여 반환한다. ...  작은 값이 가장 가까운 얼굴이다.
    # face_names: 매칭값에 따라 face_names 순서도 바꾸어 반환한다.
    distances = list(np.linalg.norm(encodings - encoding_to_check, axis=1))
    return zip(*sorted(zip(distances, face_names)))
```

```
# 8) Print obtained results: 매칭값 순으로 나열하여 반환한다. ... 작은 값부터
# 매칭값에 따라 names 순서도 바꾸어 반환한다.
computed_distances_ordered, ordered_names = compare_faces_ordered(known_encodings, names, unknown_encoding)
print(f"\nThe unknown person is identified as '{ordered_names[0]}'.")
```

```
8) computed_distances_ordered, ordered_names
 = compare_faces_ordered(known_encodings, names, unknown_encoding)
name              =      jared_3      jared_1      jared_2        obama
ordered distance  =    0.3913190    0.3998328    0.4104154    0.9053702
The unknown person is identified as 'jared_3.jpg'.
```

5. 개선된 버전

2차 과제의 주제로 생각해 볼만한....

12 각자의 코딩 능력 배양을 위해 소스 공개 안함 **2_compare_faces_dlib_ver2.py**

```
set1 = ["jared_1.jpg", "jared_3.jpg", "jared_4.jpg", "jared_5.jpg", "jared_6.jpg", "jared_2.jpg"]
data_set_list = [set0, set1, set2, set3]
set_num = 1
names = data_set_list[set_num]
print(f"사용된 영상 실험 세트={set_num}, 미지의 인물 파일명={names[-1]}")
```

```
사용된 영상 실험 세트=1, 미지의 인물 파일명=jared_2.jpg
```

단순화된 코드의 일부

```
images = dict()
encodings = []
for name in names:
    image = (cv2.imread(path + name))
    images[name] = image[:, :, ::-1]    # key와 value로 추가.
    encoding = face_encodings(image)
    encodings.append(encoding[0])   # encoding 결과를 list로 저장
```

```
Fig. 1) computed_distances = compare_faces(all_encodings, unknown_encoding)
name              =     jared_1     jared_3     jared_4     jared_5     jared_6     jared_2
matching distance =   0.4277265   0.4742946   0.4174237   0.4488114   0.4926418   0.0000000
The unknown(last) person: Euclidean distance between itselves =0.0
```

```
Fig. 2) computed_distances_ordered, ordered_names
 = compare_faces_ordered(known_encodings, names, unknown_encoding)
name              =     jared_2     jared_4     jared_1     jared_5     jared_3     jared_6
ordered distance  =   0.0000000   0.4174237   0.4277265   0.4488114   0.4742946   0.4926418
The unknown(first) person(jared_2.jpg) is close to 'jared_4.jpg'.
```

6. face_recognition 기반의 얼굴인식

compare_faces_fr.py

- face_recognition 모듈은 내부적으로는 dlib를 사용한다. 따라서 face descriptor도 동일하다.
- dlib 얼굴인식기는 모델 파일을 따로 제공해야 하는 불편함이 있는 반면에 face_recognition은 다음의 장점이 있다.
  - dlib 모델(face 5 landmark 검출, 얼굴 검출기)가 내장되어 있다.
  - 이런 디테일 과정이 모두 감추어져 있다. ⇨ 함수 운용 법이 무지 단순하다.
- 그러나 대신 아래의 단점을 각오해야 한다.
  - 얼굴 인식할 때 True 혹은 False로만 반환해 주어 성능 개선을 도모할 여지가 없다.
    - 더 옵션 기능이 있을 수 있겠지만, 추가로 확인해 보지는 않았음.

```
known_image_1 = face_recognition.load_image_file(path + "jared_1.jpg")
                              ▪ ▪ ▪
unknown_image = face_recognition.load_image_file(path + "jared_4.jpg")
```

영상 파일 읽기

```
known_image_1_encoding = face_recognition.face_encodings(known_image_1)[0]
known_image_2_encoding = face_recognition.face_encodings(known_image_2)[0]
known_image_3_encoding = face_recognition.face_encodings(known_image_3)[0]
known_image_4_encoding = face_recognition.face_encodings(known_image_4)[0]
known_encodings = [known_image_1_encoding, known_image_2_encoding,
                   known_image_3_encoding, known_image_4_encoding]
```

인코딩하여 알고 있는 인물의 디스크립터 리스트 만들기

```
unknown_encoding = face_recognition.face_encodings(unknown_image)[0]
```

미지 영상 디스크립터 만들기

```
results = face_recognition.compare_faces(known_encodings, unknown_encoding)
```

비교하기

```
print(results)
```

⇨ `[True, True, True, False]`

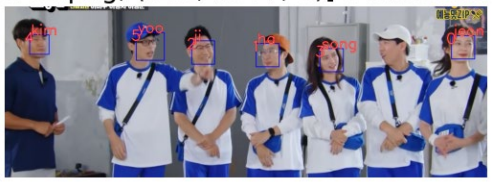
7. 응용 - 사진속의 러닝맨 멤버 찾기

2차 과제의 주제로 생각해 볼만한….

14 분류기는 SVM 사용-소스 제공 안함 rpt3_face_identify_SVM.py

[File: r8.png, (339, 1886, 3)] 6 faces found.

[File: r5.png, (663, 1867, 3)] 6 faces found.



[File: r9.png, (660, 1580, 3)] 2 faces found.

[File: r10.png, (675, 1565, 3)] 1 faces found.

[File: r6.png, (494, 1455, 3)] 4 faces found.

[File: r7.png, (415, 795, 3)] 2 faces found.

```
def svm_init(C=12.5, gamma=0.50625):
    """Creates empty model and assigns main parameters"""

    model = cv2.ml.SVM_create()
    model.setGamma(gamma)
    model.setC(C)
    model.setKernel(cv2.ml.SVM_RBF)
    model.setType(cv2.ml.SVM_C_SVC)
    model.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 100, 1e-6))

    return model
```

사용한 SVM 모델

학습용 각 개인의 얼굴 데이터 세트: data/face_db_running_man
테스트용 다수 인물 사진 데이터 세트: Data/test_images

8. 검토 - classifer의 선택

2차 과제의 주제로 생각해 볼만한….

- 얼굴 인식 과정에서 현재 제시된 내용의 성능을 개선할 사항을 제안해 보고, 소규모의 실험 데이터 비교로 밝혀 보자.
  - ~~얼굴 검출: dlib hog vs. open SSD~~
  - 분류기: Euclidean distance vs. KNN or SVM
- 대규모 표준 데이터 베이스를 사용한 검토
  - 분류기 성능이 시원치 않아 이렇게 큰 규모의 데이터는 불필요해 보여 다소 부정적. 더구나 남은 수업시간이 별로 없음.
  - 하지만 데이터 접근 경험을 쌓기 위해서 한 번 도전해 보는 것도 좋을 듯..
- Dlib face descriptor 128개 데이터의 군집성에 대한 검토
  - 같은 인물에 대한 군집성이 얼마나 강한지 검증해 보는 방법을 생각해 보았으면…
    - k-means clustering으로 자동으로 군집화를 시켰을 때 중앙값이 그 인물의 특징을 대변한다고 생각할 수 있는지? 표준 편차의 특징은 어떻게 나타나는지? 이것이 사실상 원리적으로 유클리디언으로 행한 식별화 작업과 다를 바가 없는 것인지?
- SVM이 식별자로 채택되었을 때 원리적으로 더 우수하거나 낮을 가능성을 설명할 근거가 있을까?