

<연구노트>

4월 26일 [1주차]

1. 개발주제 선정

안면 인식 도어락 , 신분증을 이용한 얼굴인식 , 동영상 사람 인식 , 표정 검출 프로그램

2 개발 동기

얼굴 인식, 머신 러닝 알고리즘을 적절하게 접목하며 최대한 활용할 수 있는 부분이라는 생각

보안과 관련된 부분에 두각을 나타낼 것이라고 생각 평소 보안이 가장 흔하게 나타나있는 도어락이 적절하다 봄

3 .개발 주제

안면 인식 도어락

- 사람들의 얼굴 저장 가능 , 일정 이상 유사도 확인시 도어락 개방
- 저장되지 않은 얼굴이면 자동으로 저장 후 , 같은 얼굴미라 판단하는 얼굴일 때 인식
- 저장 된 얼굴의 경우 자동으로 최신 얼굴을 갱신하여 업데이트
- 웹에서 범죄자 얼굴을 받아오거나 자주 방문하는 사람의 얼굴 저장 가능

5월 3일 [2주차]

<1 주차 회의 결과 : 안면 인식 보안 장치(도어락)을 사용하는 것으로 결정>

1. **개발 효용성**

안면 인식을 보안 장치에 사용하려는 연구는 많이 되어 왔고, 제품으로 만드는 경우도 많았음. 안면 인식과 딥러닝을 잘 활용할 수 있는 분야.

2. **개발 필요성**

도어락의 경우 비밀번호를 입력해야 한다는 특성이 있으며 보안 상의 이유로 범죄에 노출될 수 있음. 특히 보안이 중요한 시설의 경우에는 필요성이 큼.

3. **개발 차별성**

딥러닝을 이용하여 많은 사진 데이터가 없어도 보안 장치를 사용하면서 최신 데이터를 누적, 높은 정확성과 자동 업데이트 기능을 가지고 있음.

4. **얼굴 저장 기능**

저장되지 않은 얼굴의 경우 특징을 검출해 저장해 놓고 이 후 재방문 하였을 때 해당 인물이 기존에 방문했다는 것을 알려주고, 등록을 통해 해당 인물의 출입도 허락하는 기능.

5. **개발 난이도 검토**

도어락은 Face Detection과 Face Recognition (얼굴 감지 및 얼굴 인식) 알고리즘을 사용하는데, 이 두 알고리즘이 제대로 둘 다 작동해야만 좋은 결과를 얻어낼 수 있다.

Face Detection 알고리즘의 경우에는 수업 시간에 실습한 두 가지 방법이 있는데,

- 먼저 실습을 진행한, 5가지 점을 바탕으로 얼굴을 검출하여 Box를 입력하는 알고리즘
- 이후에 실습을 진행한, 68가지 점을 사용하여 얼굴의 유사도를 이끌어내는 알고리즘.

Detection의 경우에 68개를 사용하는 알고리즘까지 사용할 필요는 없어 보인다. 도어락을 사용할 시에 사람 한 명이 도어락 앞에 있다고 가정하기에, 5개의 점을 사용하는 알고리즘을 사용하고자 한다.

Recognition의 경우에는 68개를 사용하는 알고리즘이 필요하다. 저장된 데이터들과 비교하여 유사도를 뽑아내는 알고리즘을 활용, 특정 유사도 이상 시에 도어락이 열리는 식으로 구현 가능하다.

[유사도 분석]

- 두 눈 사이의 거리
- 이마에서 턱까지의 거리
- 코와 입 사이의 거리
- 광대뼈의 모양
- 입술, 귀, 턱의 윤곽

적절한 유사도의 수치는 개발을 하면서 측정하는 것이 바람직해 보인다.

위 내용을 토대로 개발 난이도는 어느 정도 어렵지만, 불가능한 수준은 아닐 것이라고 판단했다.

1. 얼굴 이미지를 저장해 특징을 추출
2. 영상에서 얼굴 인식
3. 인식된 얼굴을 저장한 특징과 유사도 분석

5월 10일 [3주차]

<2 주차 회의 결과 : 안면 인식 보안 장치(도어락) 제작 난이도, 효용성 등 검토, 적합한 것으로 결정.>

1. **Face Detection**

Face Detection 기능을 주로 사용하기 때문에, 수업시간에 학습한 Face Detection 기능을 이용하여 구현할 수 있는 방법 회의.

- 1.1 영상에서의 얼굴 인식

Face Detection 알고리즘을 사용하면, 얼굴이 인식되었을 때 가상의 네모 칸을 그려서 확대/축소 기능을 거친 이후에 데이터베이스에 얼굴을 모으는 것이 가능하다.

- 1.2 지난 주의 방법과의 차이

5가지의 점만을 이용하여 Face Detection을 진행하면 속도가 더 빠를 것이라는 것은 사실이나, Dataset이 하나 더 필요하게 되는 셈이다. 따라서, 68개의 점을 이용하여 Face Detection과 Face Alignment 둘 다 진행하되, 너무 오래 걸린다고 생각이 된다면 추후에 Dataset을 추가하는 것 만으로도 수정이 편리하게 가능하므로 현 시점에서는 68개의 점을 가진 Dataset 하나만 가지고 먼저 개발을 시작한다.

68개의 점을 가진 Dataset은 수업시간에 Google Colab을 사용할 때 받았던 Dataset을 참고하여 사용하기로 한다.

2. **Face Alignment**

기본적으로 얼굴의 유사도를 측정할 때는 Face Alignment를 이용하여 얼굴을 정렬한 뒤에 Face Recognition을 사용하여 유사도를 측정한다. 도어락의 경우에 정면을 보더라도 정렬이 필요할 수 있기에 Face Alignment 또한 구현에 포함시킨다.

1. **Face Recognition**

Face Detection을 구성하면서 가져온 Dataset을 이용하여 Face Recognition을 같이 진행한다. 원래 사용하고 있었던 Dataset인 만큼 큰 무리 없이 코드를 구성할 수 있을 것임.

- vscode 사용하여 구현

- 1. 웹캠을 이용하여 인식 하려는 이미지를 확인하고 저장

- 2. 이미지의 얼굴 부분을 인식하여 얼굴만 존재하는 이미지를 구현
 - 3. dlib을 활용하여 얼굴의 랜드마크를 검출하고 검출된 랜드마크를 비교하여 유사도를 구하는 방법 : 실습 시간에 활용한 shape_predictor_68_face_landmark.dat을 활용하여 코드를 구현
 - 3-1. dlib 설치에 어려움이 있어 python 환경에서 쉽게 사용할 수 있다는 장점이 있는 mediapipe를 활용한 코드를 구현
 - 4. mediapipe의 Facemesh를 활용하여 얼굴의 좌표를 이미지 좌표로 변환하여 저장하고 저장된 이미지와 유사도를 분석
- dataset은 저희의 얼굴 사진과 kaggle의 이미지를 사용
- colab에서 진행을 하다가 vscode로 넘어와서 라이브러리 설치
- 라이브러리 설치가 오래 걸려서 colab에서 진행한 코드와 vscode에서 진행한 코드를 연결하는 과정을 하지 못했습니다.
- 다음주 계획
 - 라이브러리를 설치하여 코드를 확인
 - 웹캠의 영상에서 얼굴이 존재하는 것을 확인했을 때 이미지를 저장하는 것을 구현
 - 유사도 확인
- 개발 과제

```
import cv2

cap = cv2.VideoCapture(0) # 카메라(웹캠) 연결
if cap.isOpened() :
    while True:
        ret, frame = cap.read() # 카메라 프레임 읽기
        if ret:
            cv2.imshow('camera',frame) # 프레임 화면에 표시
            if cv2.waitKey(1) != -1: # 아무 키나 누르면
                cv2.imwrite(f'photo{i}.jpg', frame) # 프레임을 'photo{i}.jpg'에 저장
                i += 1
                break
            else:
                print('no frame!')
                break
        else:
            print('no camera!')
cap.release()
cv2.destroyAllWindows()
```

5월 17일 [4주차]

<3 주차 회의 : 라이브러리 설치에 어려움, 웹캠에서 바로 얼굴을 인식하기로 함, vscode에서 진행하기로 결정>

- Dlib 설치 완료. 경로에 한글이 있었던 부분이 문제.
- 웹캠에서 바로 얼굴을 인식하고, 얼굴이 나온 부분만 확대하여 이미지로 저장

[코드]

- 웹캠에서 얼굴을 인식하여 사각형을 표시
- 얼굴이 인식 되었다면 얼굴이 인식된 사각형의 이미지를 저장

```
def videocapface():
    capture = cv2.VideoCapture(0)
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

    while True:
        ret, frame = capture.read()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3, minSize=(100, 100))

        if len(faces):
            for x, y, w, h in faces:
                face_img = frame[y:y+h, x:x+w]
                cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 255), 2, cv2.LINE_4)
                cv2.imwrite('face_photo.jpg', face_img)
                break
            cv2.imshow("original", frame)
            if cv2.waitKey(1) == ord('q'):
                break

    capture.release()
    cv2.destroyAllWindows()
```

- 사람을 등록하면서 저장한 이미지에서 특징점을 구하고 .jpg 이미지로 저장
- 주어진 사진에서 얼굴을 인식하여 얼굴 디스크립터를 추출한 후, 표본 사진들과의 유사도를 비교, 저장된 이미지와의 유사도가 일정 범위 안에 들어오면 인식 완료
- 2명 이상 인식시 1명을 대표로 인식.

```

face_detector = dlib.get_frontal_face_detector()
shape_predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
face_rec_model = dlib.face_recognition_model_v1('dlib_face_recognition_resnet_model_v1.dat')

sample_images = []
sample_files = ['sample1.jpg', 'sample2.jpg', 'sample3.jpg']

for sample_file in sample_files:
    sample_image = cv2.imread(sample_file)
    gray_sample = cv2.cvtColor(sample_image, cv2.COLOR_BGR2GRAY)

    faces = face_detector(gray_sample)
    if len(faces) > 0:
        face = faces[0]
        landmarks = shape_predictor(gray_sample, face)
        face_descriptor = face_rec_model.compute_face_descriptor(sample_image, landmarks)
        face_descriptor = np.array(face_descriptor)
        sample_images.append((sample_file, face_descriptor))

query_image_file = 'query_image.jpg'
query_image = cv2.imread(query_image_file)
gray_query = cv2.cvtColor(query_image, cv2.COLOR_BGR2GRAY)

query_faces = face_detector(gray_query)
if len(query_faces) > 0:
    query_face = query_faces[0]
    query_landmarks = shape_predictor(gray_query, query_face)
    query_descriptor = face_rec_model.compute_face_descriptor(query_image, query_landmarks)
    query_descriptor = np.array(query_descriptor)

    similarities = []
    for sample_file, sample_descriptor in sample_images:
        similarity = np.linalg.norm(sample_descriptor - query_descriptor)
        similarities.append((sample_file, similarity))

    similarities.sort(key=lambda x: x[1])
    for sample_file, similarity in similarities:
        print("Sample:", sample_file)
        print("Similarity:", similarity)
else:
    print("No faces found in the query image.")

```

<추후 추가 사항> // 변경 가능

- ** 옆에서 찍은 사진의 인식률이 떨어짐 ⇒ Face Alignment 필요.
- 통과 유사도는 평균 75%, 최대 80% 이상으로 설정.
- 저장되지 않은 얼굴의 경우 평균은 생각하지 않음.
- 다른 얼굴 인식시 보통 40~50% 정도의 유사도 출력.
- 얼굴 저장 시 10장의 사진을 찍어 저장.
- 저장하는 사진 파일의 최대 갯수는 사람당 최대 200개로 설정.
- 저장되지 않은 사람의 파일 최대 갯수는 고려하지 않음.(삭제 X)
- 추가로 사람의 얼굴을 저장할 시 저장되지 않은 사람들의 파일을 전부 탐색해서, 유사도 75% 이상의 사진을 전부 가져와 저장.

5월 24일 [5주차]

05월 24일 회의 (5 주차)

<4 주차 진행 상황 : 라이브러리 설치 모두 완료. 웹캠에서 얼굴 검출해서 저장하는 기능 구현.>

- 지난 주 피드백 (Dataset 학습 방법을 통한 유사도 판단)은, 다음 주 (저장되지 않은 얼굴 구분) 알고리즘 구현 후 적용 예정.
- 지난 주 유사도 검출 프로그램 업그레이드 ⇒ 저장된 사진들의 유사도 평균 검출하는 함수 작성.
- 평균 유사도가 65% 이상일 경우 Door Opened 출력 / 이외의 경우 Door Closed 출력

[평균값 검출 알고리즘]

```
import cv2
```

```
import dlib
```

```
import numpy as np
```

```
import os
```

```
face_detector = dlib.get_frontal_face_detector()
```

```
shape_predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
```

```
face_rec_model =
```

```
dlib.face_recognition_model_v1('dlib_face_recognition_resnet_model_v1.dat')
```

```
sample_images = []
```

```
sample_folder = 'sample_images_ji/'
```

```
sample_files = os.listdir(sample_folder)[:200]
```

```
for sample_file in sample_files:
```

```
    sample_image = cv2.imread(os.path.join(sample_folder, sample_file))
```

```
    gray_sample = cv2.cvtColor(sample_image, cv2.COLOR_BGR2GRAY)
```

```
    faces = face_detector(gray_sample)
```

```
    if len(faces) > 0:
```

```
        face = faces[0]
```

```
        landmarks = shape_predictor(gray_sample, face)
```

```
        face_descriptor = face_rec_model.compute_face_descriptor(sample_image,  
landmarks)
```

```
        face_descriptor = np.array(face_descriptor)
```

```
        sample_images.append((sample_file, face_descriptor))
```

```

query_image_file = 'query_image.jpg'
query_image = cv2.imread(query_image_file)
gray_query = cv2.cvtColor(query_image, cv2.COLOR_BGR2GRAY)

query_faces = face_detector(gray_query)
if len(query_faces) > 0:
    query_face = query_faces[0]
    query_landmarks = shape_predictor(gray_query, query_face)
    query_descriptor = face_rec_model.compute_face_descriptor(query_image,
query_landmarks)
    query_descriptor = np.array(query_descriptor)

    similarities = []
    for sample_file, sample_descriptor in sample_images:
        similarity = np.linalg.norm(sample_descriptor - query_descriptor)
        similarities.append((sample_file, similarity))

    similarities.sort(key=lambda x: x[1])

    total_similarity = 0.0
    for sample_file, similarity in similarities:
        total_similarity += (1 - similarity)

    average_similarity = total_similarity / len(similarities)
    print("Average Similarity:", average_similarity)
else:
    print("No faces found in the query image.")

```

- 본인이라고 생각할 수 있는 유사도 (65%) 이상 사진들의 평균값 측정해서 화면에 표시.

- 예시 사진을 미리 1개씩 Images 폴더에 저장하고, samples 폴더를 탐색해서 그 인물이 누구인지 판별.

- 판별 후, Images 폴더 안에 새로운 디렉토리를 생성해서 폴더 안쪽에 경로를 집고 사진을 저장.

- File count 해서 디렉토리에 200개 이상의 파일이 있는 경우...

- 1안)

// glob.glob(folder_path + '/*') 함수로 파일 개수 파악.


```
// **os.p0ath.getctime()**으로 생성 시간 측정

// 생성 시간이 제일 이른 사진 삭제

- 2안)

// 파일 이름을 (사람 이름)_i로 저장.

// 저장할 때마다 라이브러리 파일 개수를 파악해서 i로 사용.

// i가 200이 넘어가면 i = 0 파일을 삭제하고, 1~199 파일은 i값을 하나씩 줄여서 다시
저장.

// 새로운 파일은 i = 199로 저장.
```

****< Face Alignment>**

[Deepface Face Alignment]

```
from deepface import DeepFa
import os
```

```
sample_images_folder = 'sample_images'
sample_faces_folder = 'sample_faces'
os.makedirs(sample_faces_folder, exist_ok=True)
```

```
img_path = os.path.join(sample_images_folder, 'sample0.jpg')
```

```
if os.path.exists(img_path):
    face_objs = DeepFace.extract_faces(img_path=img_path)
```

```
    for i, face_obj in enumerate(face_objs):
        img = face_obj["face"]
        face_file = os.path.join(sample_faces_folder, f'sample0_{i}.jpg')
        cv2.imwrite(face_file, img)
```

```
else:
    print("Image not found: sample0.jpg")
```

얼굴 중 눈 부분의 각도를 측정해 수평을 이루도록 사진을 회전시키는 알고리즘 사용.

눈을 감고 있는 경우에는 작동하지 않음.

눈을 감고 있는 사진은 삭제.

// 얼굴 정렬 도중 저장되지 않는 경우가 발생 ⇒ 화질이 좋지 않거나, 눈을 감고 있거나 등등.. 정렬되지 않음.

⇒ 새로운 알고리즘 / Dataset 필요

6월 1일 [6주차]

06월 01일 회의 (6주차)

<지난 주차 진행 사항>

- 이전 주차에 미리 저장해둔 폴더의 이미지들과 유사도를 비교하는 코드 작성
- 유사도 검출을 통해 65% 이상의 값인 경우 문을 여는 코드 작성

<이번 주 진행 사항>

- 얼굴 정렬 알고리즘을 구성하고 이를 활용
- 얼굴 정렬을 사용하여도 정확도를 개선하는데 큰 도움이 되지 않음
- 시간이 너무 오래 걸리는 단점을 해결하기 위해 비교하는 이미지의 수를 줄임 (200 → 50)
- 3~5개의 이미지를 활용하여 빠르게 판별하고 이를 만족하면 50개의 이미지와 비교
- 이를 활용한 코드 작성

[얼굴 이미지 저장](<https://www.notion.so/95c263c22b65456999cafa13ab3231c9?pvs=21>)

```
```python
def videocapface():
 global i
 capture = cv2.VideoCapture(0)
 face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

 while True:
 ret, frame = capture.read()
 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

 faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3,
 minSize=(200, 200))
```

```

 if len(faces):
 for x, y, w, h in faces:
 # 확장할 영역 계산
 extension = int(w * 0.2) # 얼굴 폭의 10%만큼 확장
 x_start = max(0, x - extension)
 y_start = max(0, y - extension)
 x_end = min(frame.shape[1], x + w + extension)
 y_end = min(frame.shape[0], y + h + extension)

 face_img = frame[y_start:y_end, x_start:x_end]
 # cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 255), 2,
cv2.LINE_4)

 # 이미지 저장
 cv2.imwrite('query_image.jpg', face_img)

 i += 1
 break
 cv2.imshow("original", frame)
 if cv2.waitKey(1) == ord('q'):
 break

capture.release()
cv2.destroyAllWindows()
...

```

[유사도 검출](<https://www.notion.so/3d038c289e2d46e0ac6687ac45713314?pvs=21>)

```

```python
def calculate_average_similarity(sample_folder, query_image_file):
    face_detector = dlib.get_frontal_face_detector()
    shape_predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
    face_rec_model =
dlib.face_recognition_model_v1('dlib_face_recognition_resnet_model_v1.dat')

    sample_images = []
    sample_files = os.listdir(sample_folder)[:50]

    for sample_file in sample_files:
        sample_image = cv2.imread(os.path.join(sample_folder, sample_file))
        gray_sample = cv2.cvtColor(sample_image, cv2.COLOR_BGR2GRAY)

```

```

        faces = face_detector(gray_sample)
        if len(faces) > 0:
            face = faces[0]
            landmarks = shape_predictor(gray_sample, face)
            face_descriptor = face_rec_model.compute_face_descriptor(sample_image,
landmarks)
            face_descriptor = np.array(face_descriptor)
            sample_images.append((sample_file, face_descriptor))

    query_image = cv2.imread(query_image_file)
    gray_query = cv2.cvtColor(query_image, cv2.COLOR_BGR2GRAY)

    query_faces = face_detector(gray_query)
    if len(query_faces) > 0:
        query_face = query_faces[0]
        query_landmarks = shape_predictor(gray_query, query_face)
        query_descriptor = face_rec_model.compute_face_descriptor(query_image,
query_landmarks)
        query_descriptor = np.array(query_descriptor)

    similarities = []
    for sample_file, sample_descriptor in sample_images:
        similarity = np.linalg.norm(sample_descriptor - query_descriptor)
        similarities.append((sample_file, similarity))

    similarities.sort(key=lambda x: x[1])

    total_similarity = 0.0
    for sample_file, similarity in similarities:
        total_similarity += (1 - similarity)

    average_similarity = total_similarity / len(similarities)
    return average_similarity
else:
    print("No faces found in the query image.")
...

```

[실행 코드](<https://www.notion.so/a97259db354f45a4bed6feb4a34cd36f?pvs=21>)

```python

```

def print_average_similarity():
 global i, j
 p = 0
 while p == 0:
 videocapface()
 for i in range(4):
 sample_folder = f'{i}/'
 query_image_file = 'query_image.jpg'
 average_similarity = calculate_average_similarity(sample_folder,
query_image_file)
 # print(average_similarity)
 if average_similarity > 0.5:
 average_similarity = calculate_average_similarity_50(sample_folder,
query_image_file)
 # print(round(average_similarity * 100, 2))
 if average_similarity > 0.7:
 print(f"Door Opened, {i}")
 print(round(average_similarity * 100, 2))
 sample_file = f'sample{j}.jpg'
 shutil.copy(query_image_file, sample_file)
 j += 1
 if j > 200:
 j = 0
 p = 1
 break
 ...

```

```python

```

def print_average_similarity():
    global i, j, p
    while p == 0:
        # videocapface()
        p = 0
        for i in range(4):
            sample_folder = f'{i}/'
            query_image_file = 'query_image.jpg'
            average_similarity = calculate_average_similarity(sample_folder,
query_image_file)
            if average_similarity > 0.5:
                average_similarity = calculate_average_similarity_50(sample_folder,
query_image_file)

```

```

if average_similarity > 0.7:
    print(f"Door Opened, {i}")
    print(round(average_similarity * 100, 2))
    sample_file = f'sample{j}.jpg'
    shutil.copy(query_image_file, sample_file)
    j += 1
    if j > 200:
        j = 0
    p = 1

```

```

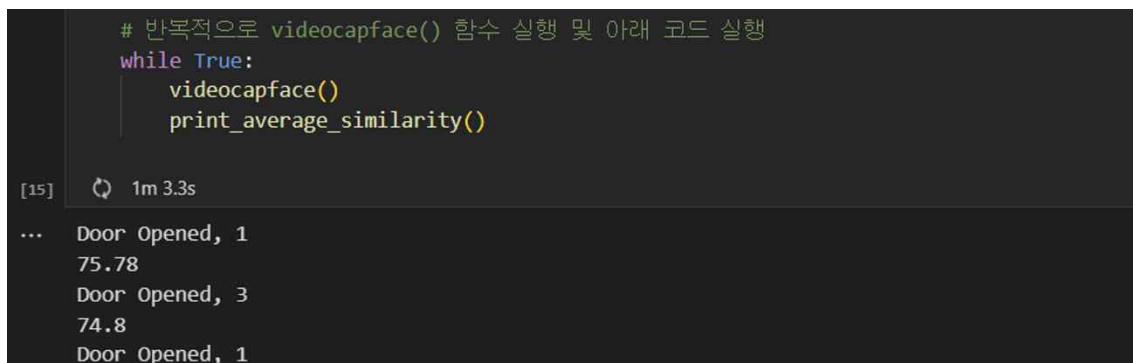
time.sleep(3) # 10초간 멈춤

```

```

# 반복적으로 videocapface() 함수 실행 및 아래 코드 실행
while True:
    videocapface()
    print_average_similarity()
...

```



```

# 반복적으로 videocapface() 함수 실행 및 아래 코드 실행
while True:
    videocapface()
    print_average_similarity()

```

[15] 1m 3.3s

```

... Door Opened, 1
    75.78
    Door Opened, 3
    74.8
    Door Opened, 1

```

<발생한 문제점>

- 계속해서 코드를 진행시키기 때문에 낭비되는 상황이 많음
- 새로운 얼굴을 받아왔을 때 저장하고 관리하는 것이 불편
- 현재 진행 중인 상황을 확인하기 어려움

<해결 방안>

- 새로운 코드를 활용
- 웹캠을 띄운 상황에서 얼굴을 인식 할 때마다 그 얼굴의 유사도를 확인하여 어떤 얼굴인지 확인
- 확인된 얼굴이 무엇인지 얼굴 위 사각형과 함께 문자로 출력
- face_recognition 라이브러리를 사용

[웹캠 내에서 얼굴 인식 및

저장](<https://www.notion.so/4368a53009fd492eaf6ed08fc2dcdd5?pvs=21>)

```
```python
import cv2
import os
import face_recognition

웹캠에서 얼굴 인식 및 비교0
known_dir = 'known' # 이미 알려진 얼굴 이미지가 있는 디렉토리
unknown_dir = 'unknown'
threshold = 0.4 # 임계값

'known' 디렉토리에서 알려진 얼굴 이미지를 불러옵니다.
known_faces = []
known_names = []
for file_name in os.listdir(known_dir):
 if file_name.endswith('.jpg') or file_name.endswith('.jpeg') or
file_name.endswith('.png'):
 image = face_recognition.load_image_file(os.path.join(known_dir, file_name))
 face_encoding = face_recognition.face_encodings(image)
 if len(face_encoding) > 0:
 known_faces.append(face_encoding[0])
 name = os.path.splitext(file_name)[0] # 파일 이름에서 확장자를 제거하여
이름으로 사용합니다.
 known_names.append(name)

video_capture = cv2.VideoCapture(0)

i = 1
j = 1
max_save_count = 10 # 최대 저장 횟수
while True:
 ret, frame = video_capture.read()
 if not ret:
 break

 # 프레임에서 얼굴 인식
 face_locations = face_recognition.face_locations(frame)
 face_encodings = face_recognition.face_encodings(frame, face_locations)
```

```

중복 인식 여부 확인을 위한 변수
duplicate_detected = False

for idx, face_encoding in enumerate(face_encodings):
 if len(face_encoding) == 0:
 continue

 # 저장된 얼굴과의 유사도 계산
 matches = face_recognition.compare_faces(known_faces, face_encoding,
tolerance=threshold)
 name = "Unknown"

 if True in matches:
 matched_index = matches.index(True)
 name = known_names[matched_index]
 cv2.rectangle(frame, (face_locations[idx][3], face_locations[idx][0]),
(face_locations[idx][1], face_locations[idx][2]), (0, 255, 0), 2)
 cv2.putText(frame, name, (face_locations[idx][3] + 6,
face_locations[idx][2] - 6), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
 duplicate_detected = True
 elif not duplicate_detected:
 cv2.rectangle(frame, (face_locations[idx][3], face_locations[idx][0]),
(face_locations[idx][1], face_locations[idx][2]), (0, 0, 255), 2)
 cv2.putText(frame, f'Unknown_{i}', (face_locations[idx][3] + 6,
face_locations[idx][2] - 6), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
 # unknown 디렉토리에 얼굴 저장
 elif j <= max_save_count:
 cv2.imwrite(f'unknown/unknown_{i}_{j}.jpg',
frame[face_locations[idx][0]:face_locations[idx][2],
face_locations[idx][3]:face_locations[idx][1]])

 if duplicate_detected:
 # 중복 인식된 경우 해당 i 값에 해당하는 사진들을 전부 삭제
 for file_name in os.listdir(unknown_dir):
 if file_name.startswith(f'unknown_{i}_'):
 os.remove(os.path.join(unknown_dir, file_name))

프레임 출력
cv2.imshow('Video', frame)

'q' 키를 누르면 종료

```



```
 if cv2.waitKey(1) & 0xFF == ord('q'):
 break

video_capture.release()
cv2.destroyAllWindows()
````
```

<다음 주 진행 사항>

- 기존 코드와 새로운 코드를 합쳐 코드를 작성
- 웹캠을 띄운 상황에서 얼굴을 인식하고, 분석하여 화면에 출력
- 인식한 결과는 저장된 이미지와 저장되지 않은 이미지로 분류하여 저장
- 코드를 완성한 경우, 직접 학습하는 방법으로 만들기