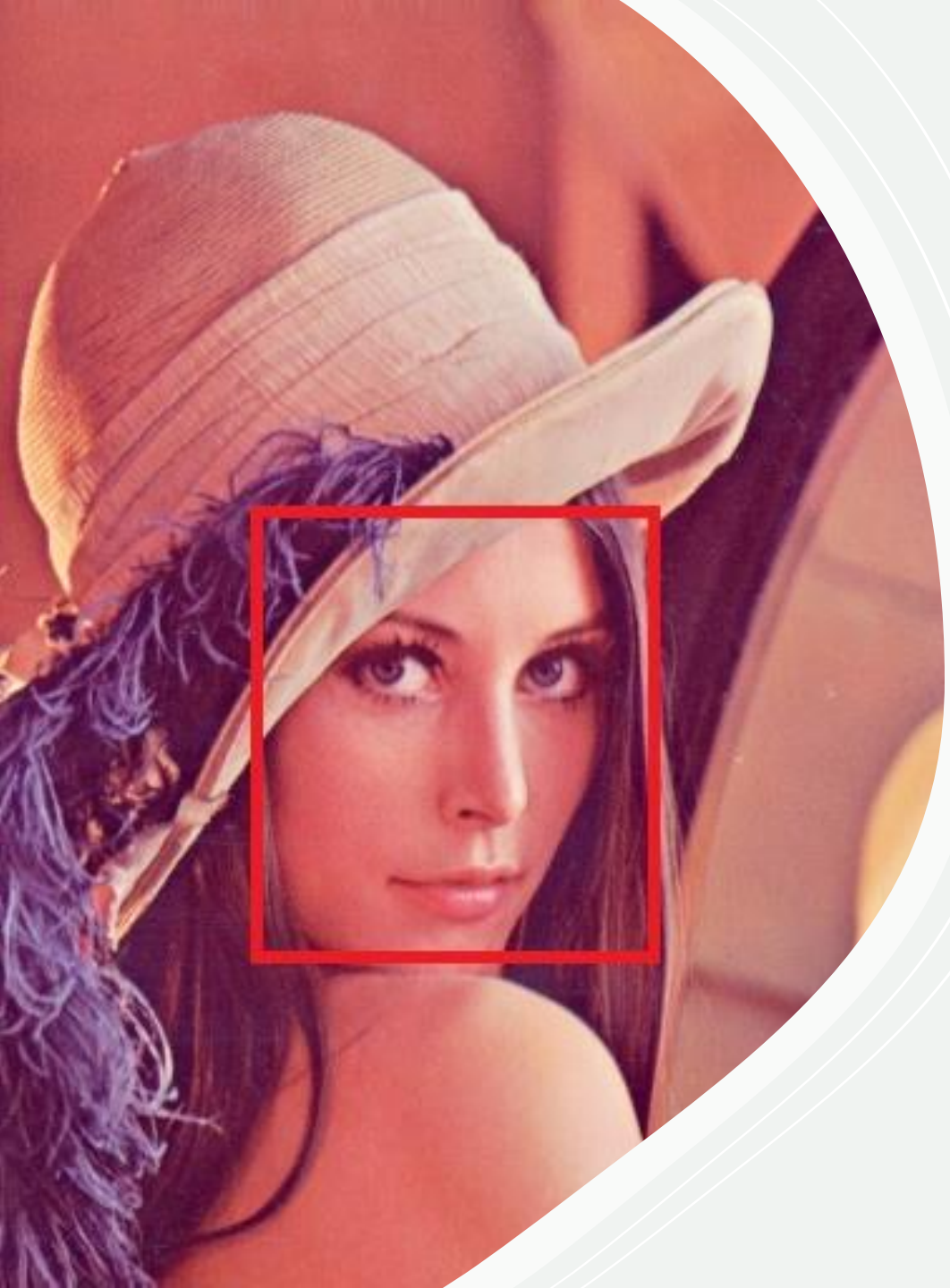
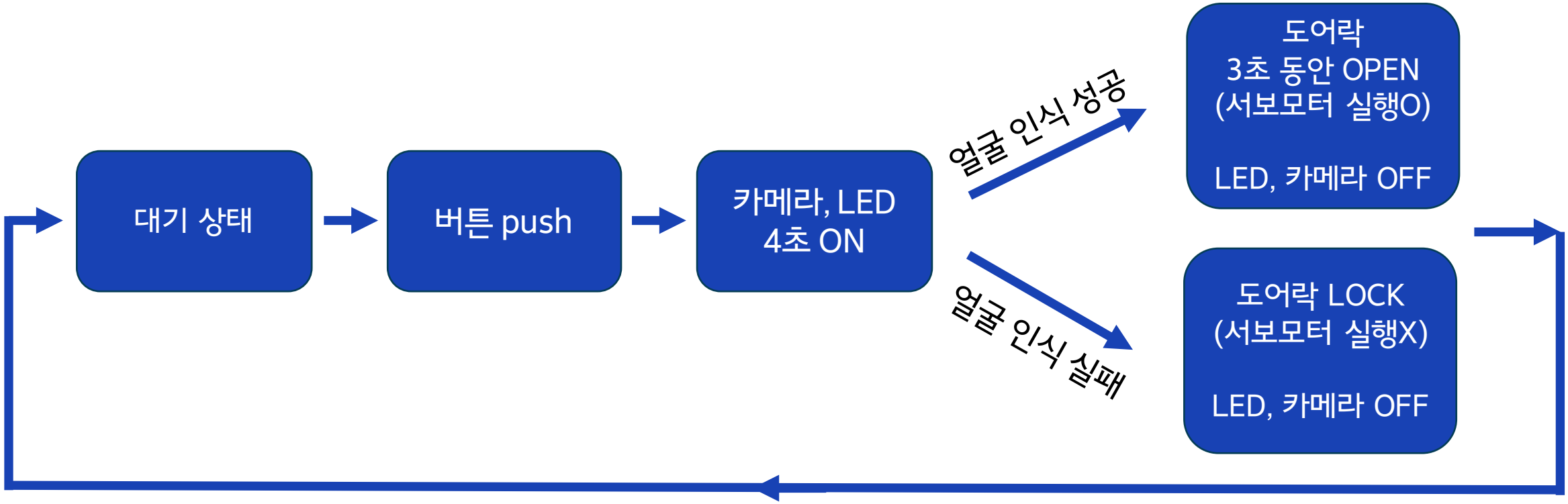


얼굴 인식 도어락

2019097138 장원

2019023654 전상우

개요-동작 설명


대기 상태
버튼 push
카메라, LED
4초 ON
얼굴 인식 성공
얼굴 인식 실패
도어락
3초 동안 OPEN
(서보모터 실행O)
LED, 카메라 OFF
도어락 LOCK
(서보모터 실행X)
LED, 카메라 OFF

개요-코드 핵심 파트 설명

```python
import face_recognition
import cv2
import numpy as np
import RPi.GPIO as GPIO
import time
```

사용 라이브러리 및 사용 목적

face_recognition: 얼굴 인식

Opencv: 이미지 처리 및 비디오 캡처

Numpy: 배열 및 행렬 연산

Rpi.GPIO: 라즈베리파이 GPIO핀을 사용

Time: 시간 측정

```python
# 샘플 이미지를 로드하고 얼굴을 인식하는 방법을 학습
jeon_image = face_recognition.load_image_file("/home/SIUUU/.venv/door_lock/jeonsang/jeon.jpg")
jeon_face_encoding = face_recognition.face_encodings(jeon_image)[0]

known_faces = [
    jeon_face_encoding
]
```

face_recognition.load_image_file: 이미지 파일 및 폴더를 읽어옴

face_recognition.face_encodings: 얼굴 위치에서 각 얼굴의 특징 벡터를 인코딩

개요-핵심 파트 설명

```python
# 비디오 처리 함수
def process_video(video_path):
    frame_number = 0
    frame_skip = 1
    recognition_start_time = None
    start_time = time.time()  # 비디오 처리가 시작된 시간

    # 입력 동영상 파일 열기
    input_movie = cv2.VideoCapture(video_path)
    length = int(input_movie.get(cv2.CAP_PROP_FRAME_COUNT))

    recognized = False

    while input_movie.isOpened():
        # 프레임 건너뛰기
        if frame_number % frame_skip != 0:
            frame_number += 1
            input_movie.grab()  # 프레임을 읽고 처리를 건너뜀
            continue

        # 비디오의 단일 프레임을 가져오기
        ret, frame = input_movie.read()
        frame_number += 1

        # 입력 동영상 파일이 끝나면 종료
        if not ret:
            break
```

frame_skip: 비디오 프레임을 건너 뛰어 모든 프레임을 보지 않도록 한다

cv2.VideoCapture: 비디오 파일을 열어 프레임을 읽기 위해 사용

개요-코드 핵심 파트 설명

```python
# 얼굴 인식 처리하기 위한 프레임 크기 조정
small_frame = cv2.resize(frame, (0, 0), fx=0.2, fy=0.2) 1
rgb_small_frame = np.ascontiguousarray(small_frame[:, :, ::-1])

# 현재 비디오 프레임에서 모든 얼굴 위치와 얼굴 인코딩 찾기
face_locations = face_recognition.face_locations(rgb_small_frame, model="hog") 2
face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations) 3

face_names = []
for face_encoding in face_encodings: 4
    matches = face_recognition.compare_faces(known_faces, face_encoding, tolerance=0.50)
    distances = face_recognition.face_distance(known_faces, face_encoding) 5
    name = None

    if len(distances) > 0:
        best_match_index = np.argmin(distances)
        if matches[best_match_index]:
            name = names[best_match_index]
            probability = 1 - distances[best_match_index]  # 일치 확률

    face_names.append(name)
    if name:
        recognized = True
```

1. 프레임의 크기를 조절하여 얼굴 인식의 속도 조절

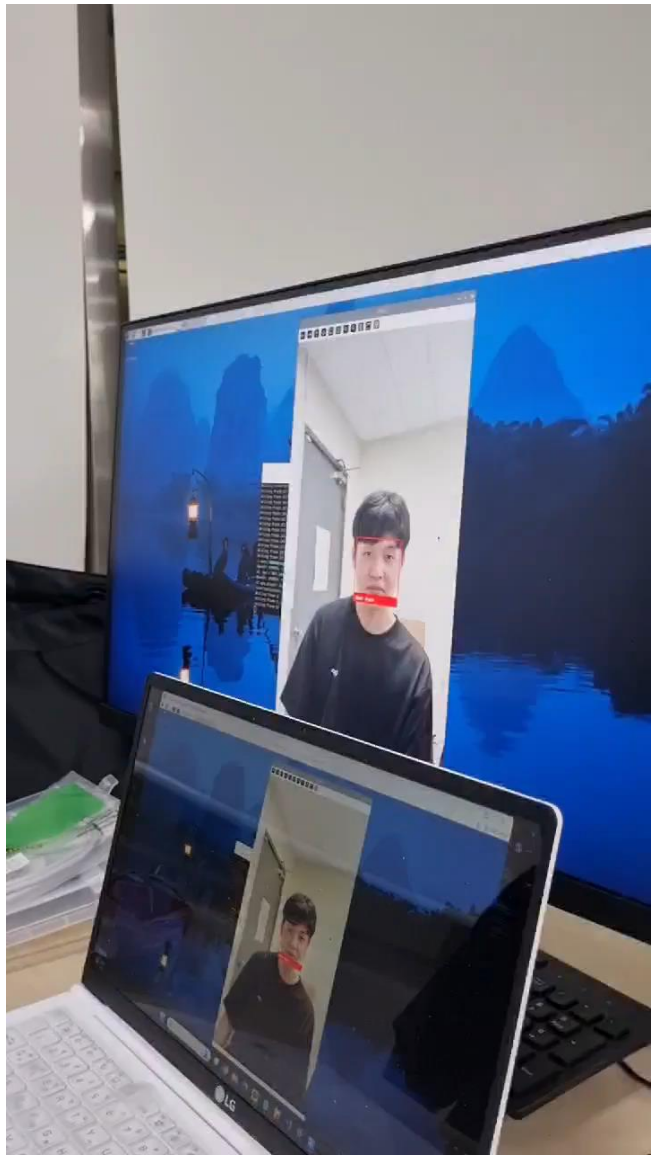
2. 현재 프레임에서 얼굴의 위치를 찾음

3. 얼굴 위치에서 각 얼굴의 특징 벡터를 인코딩

4. 현재 프레임의 얼굴 특징 벡터와 알려진 얼굴들의 특징 벡터를 비교하여 일치 여부를 판단

5. 현재 프레임의 얼굴 특징 벡터와 알려진 얼굴들의 특징 벡터 간의 거리를 계산하여 유사도를 평가하기 위해 사용

동영상 데모 [초기 모델]



초기 모델 문제점

1. 불러온 동영상이 너무 느리다.

2. 얼굴 인식률이 낮다(50퍼센트 초반).

목표: 최적화 하기! 무엇을?

1. 동영상을 빠르고 부드럽게 인식해야한다.

2. 얼굴 인식률을 최대화 한다.

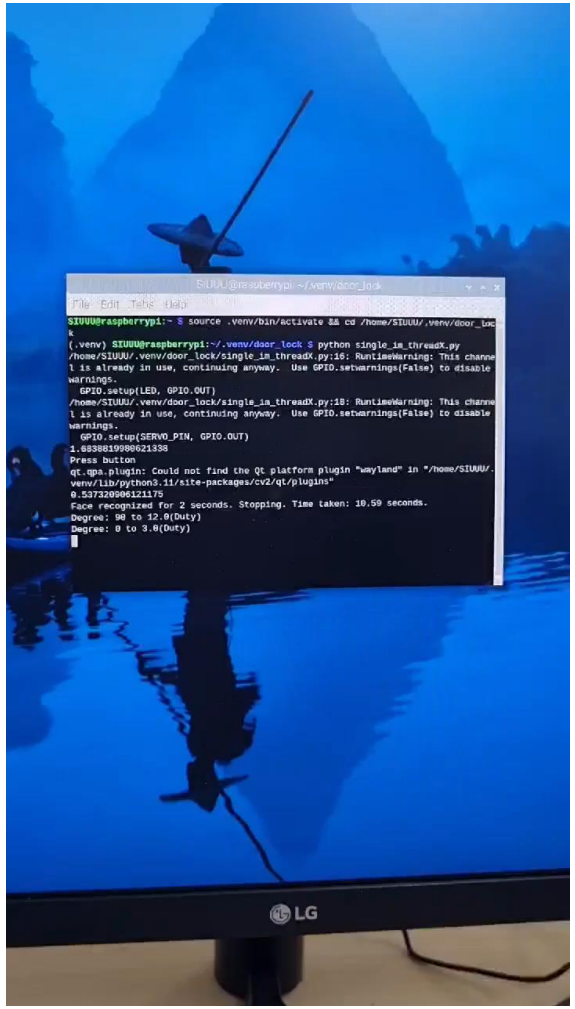
목표: 최적화 하기! 어떻게?

1. Resize, frame_skip 등 파라미터 조절

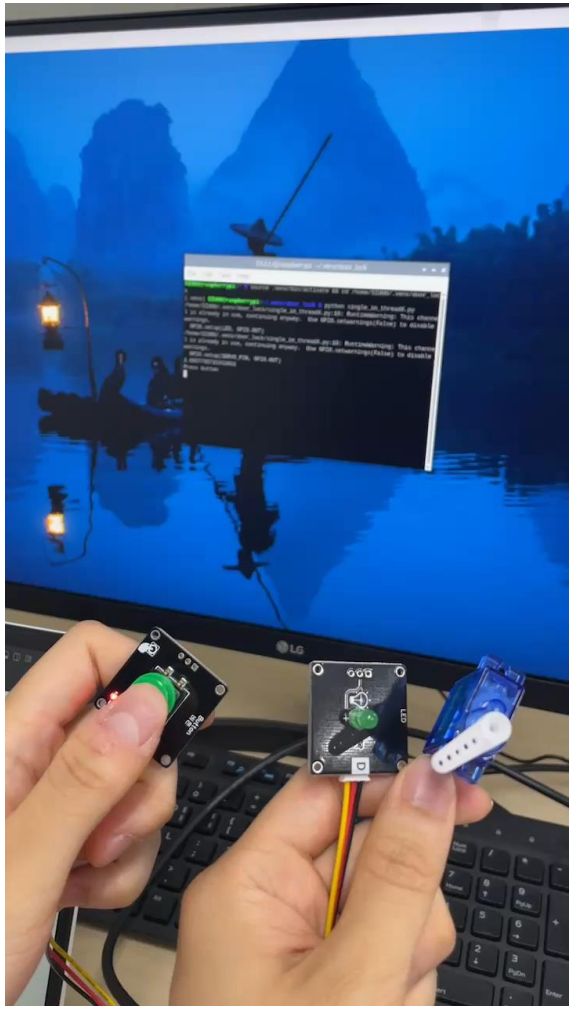
2. 라즈베리파이 스펙에 적합한 모델 찾기

3. Data Augmentation

목표: 최적화 하기! 어떻게? [Frame_skip 조절]



frame_skip 1일 때



frame_skip 6일 때

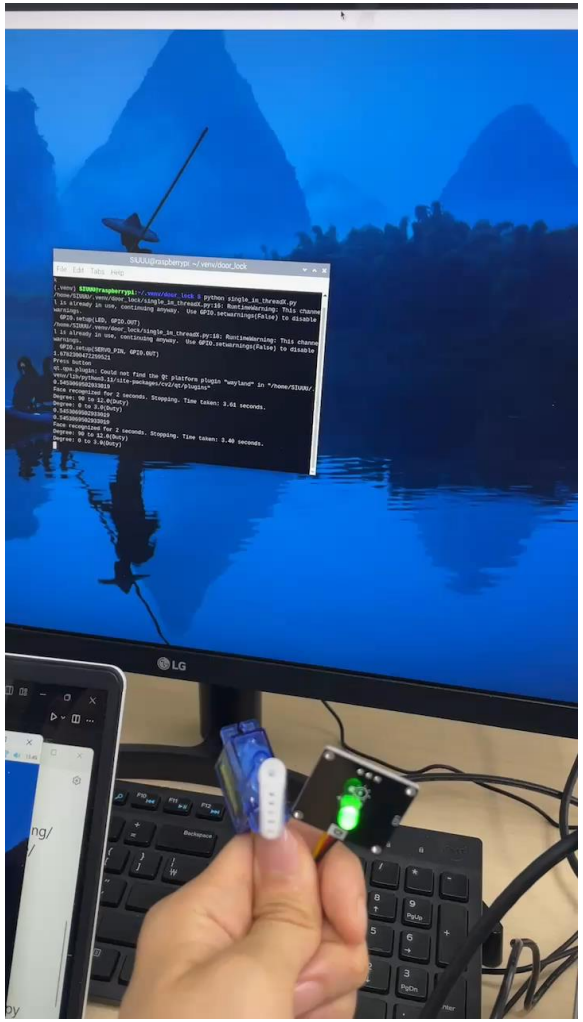
[결론]

Frame_skip이 1일 때 얼굴 인식에 걸리는 시간: 6.6초

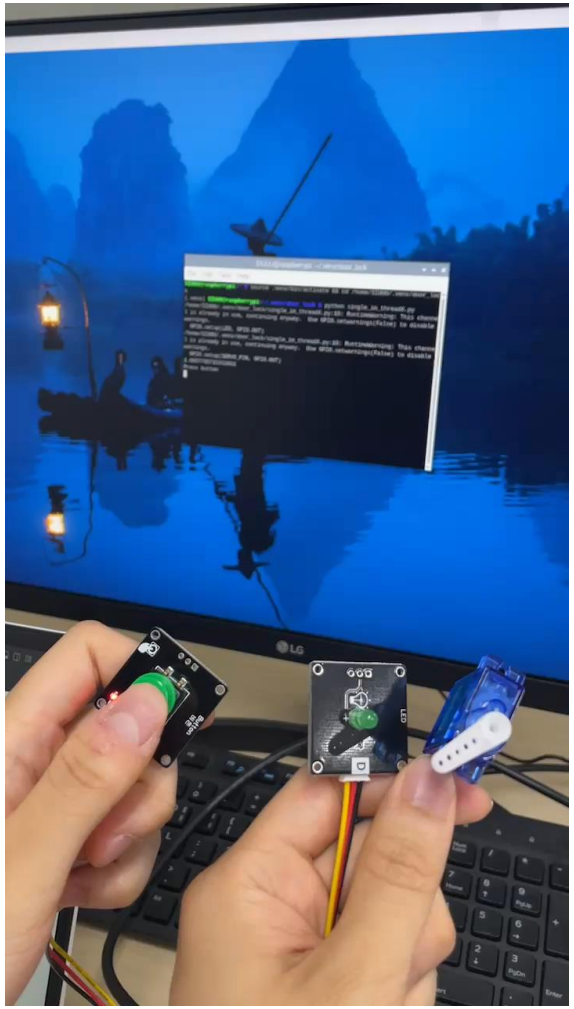
Frame_skip이 6일 때 얼굴 인식에 걸리는 시간: 2.8초

Frame_skip이 6보다 큰 경우 얼굴을 인식할 수 있는 프레임이 너무 적어진다. 영상 속도도 너무 빠르다.

목표: 최적화 하기! 어떻게? [cv2.resize조절]



resize 0.1일 때



Resize 0.2일 때

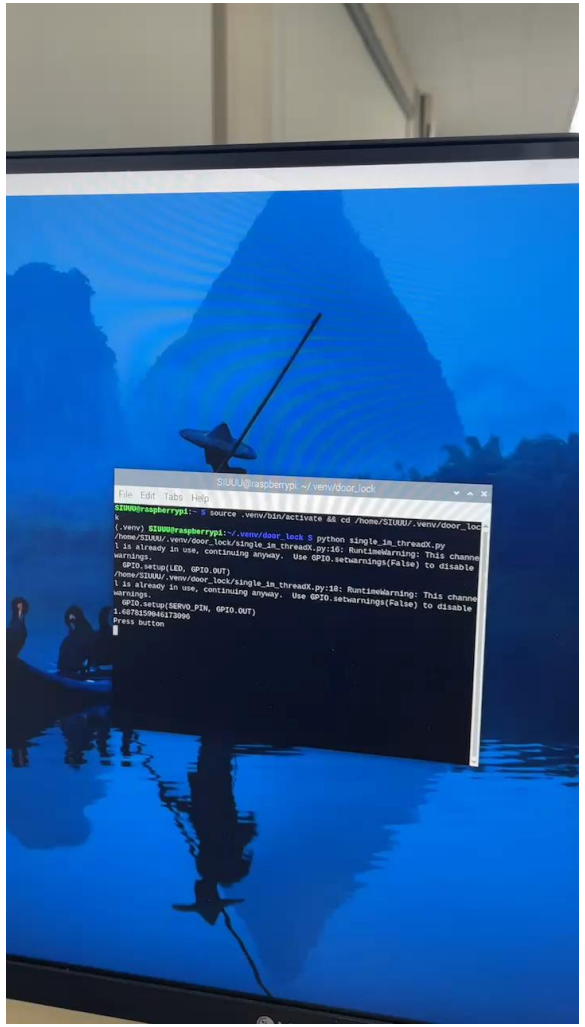
[결론]

Resize가 0.1일 때 얼굴 인식을 잘 못한다.

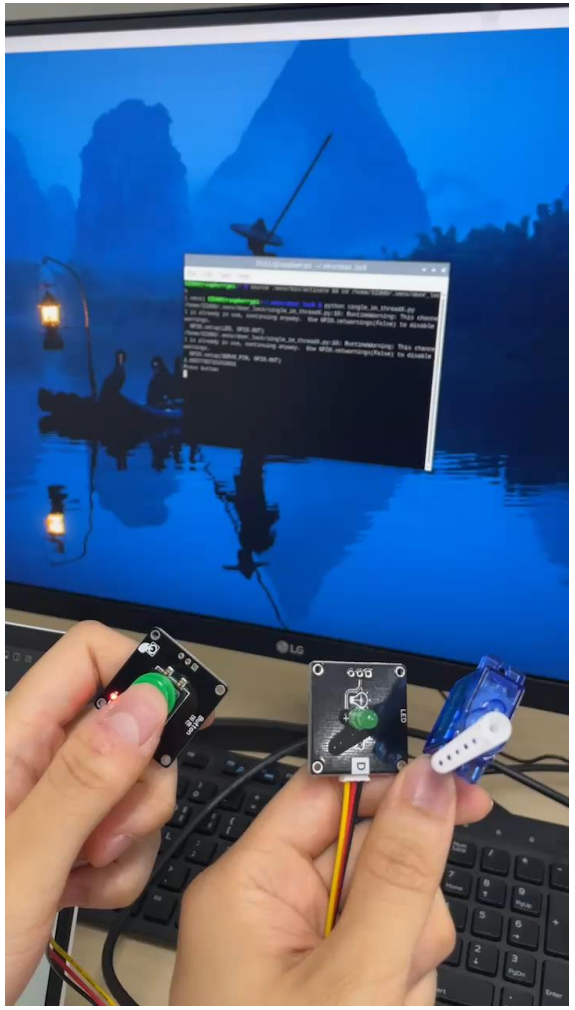
Resize가 0.2일 때 인식할 수 있는 프레임이 늘어난다.

Resize가 0.5보다 큰 경우 연산 처리량이 많아져서 영상이 과도하게 느려졌다.

목표: 최적화 하기! 어떻게? [CNN모델 vs HOG모델]



'CNN'일 때



'HOG'일 때

특징/사양	라즈베리파이 4B
SoC	Broadcom Quad Core BCM2711 Cortex-A72 @1.5GHz
GPU	500Mhz VideoCore VI

[결론]

CNN일 경우: GPU연산에 최적화된 모델
>>라즈베리파이4 GPU로는 부적합

HOG일 경우: CPU연산에 최적화 된 모델
>>라즈베리파이4는 쿼드코어 CPU 탑재

목표: 최적화 하기! 어떻게?

Data augmentation 대표적인 5가지 기법 적용

```python
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # 랜덤하게 좌우 반전
    transforms.RandomResizedCrop(256),  # 랜덤하게 잘라서 크기 조절
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),  # 랜덤하게 색감 조절
    transforms.RandomRotation(20),  # 랜덤하게 회전
    transforms.RandomAdjustSharpness(sharpness_factor=2),  # 랜덤하게 선명도 조절
])
```

목표: 최적화 하기! 어떻게?

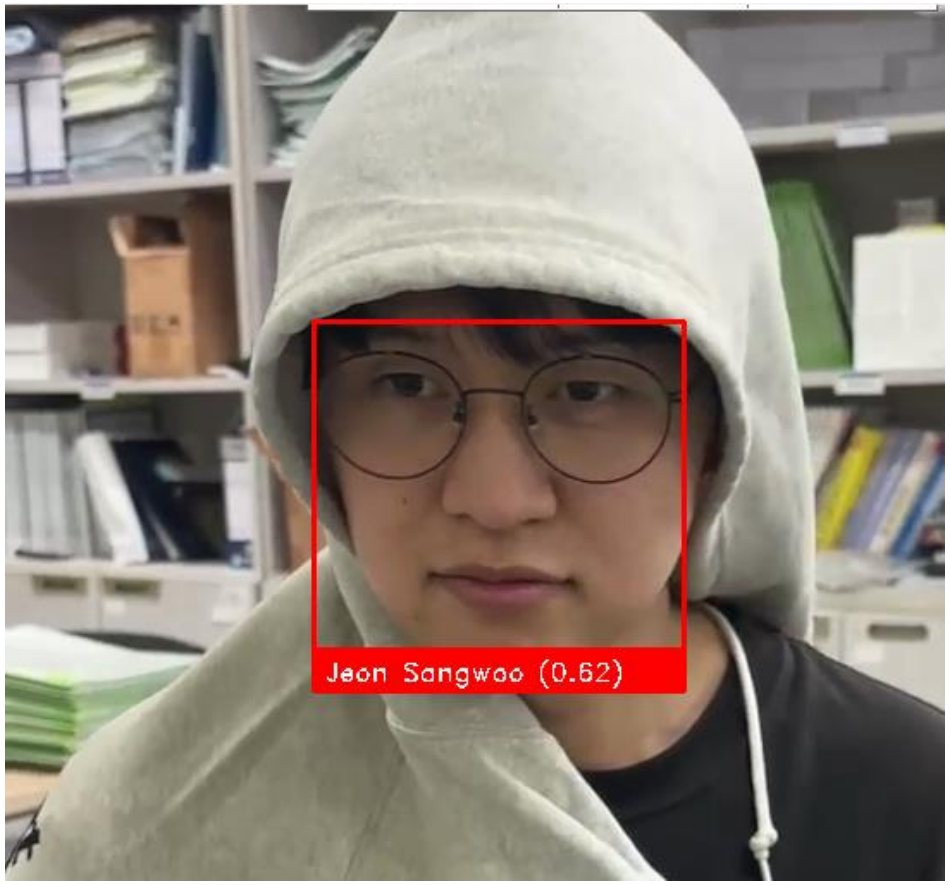
단일 사진(1장) vs 복수 사진+Data augmentation(42장)

63% 매칭

65.5% 매칭

목표: 최적화 하기! 어떻게?

단일 사진(1장) vs 복수 사진+Data augmentation(42장)

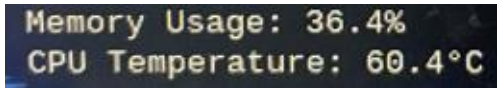
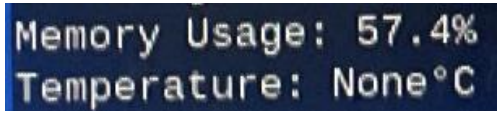


62% 매칭

63.8% 매칭

목표: 최적화 하기! 어떻게?

단일 사진(1장) vs Data augmentation(42장)

	인식률	메모리 사용량	로딩 속도(학습 시간)
단일 사진			
복수 사진+D.A			

- Memory management
  - 메모리 사용은 전력 소비에 상당한 영향을 미칠 수 있으며, 특히 메모리를 자주 읽고 쓰는 시스템에서는 그 영향이 더 커질 수 있음
  - 저전력 메모리장치, 메모리 압축, 전력 차단, 메모리 사용을 최적화 하는 소프트웨어 설계

결론: 적절한 인식률, 메모리 사용량, 로딩 속도에 대한 타협점을 찾았다.
>>단일 사진에 대한 Data augmentation(6장)을 만들어서 학습시켰다.

결론

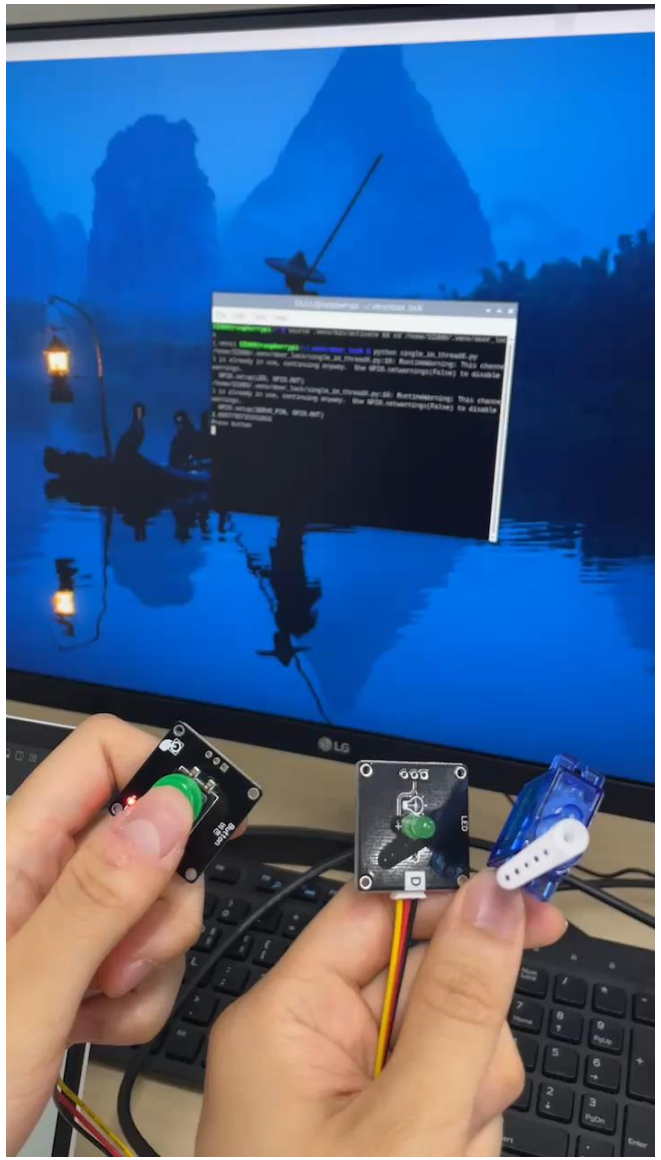
1. Frame_skip은 6이 얼굴 인식에 걸리는 시간이 적절했다.

2. Resize는 0.2가 얼굴 인식에 적절했다.

3. 라즈베리파이에는 HOG 모델이 적절했다.

4. 단일 사진에 대한 Data augmentation을 적용해 총 6장을 학습시켰다.

최종 모델 동영상 데모 및 평가



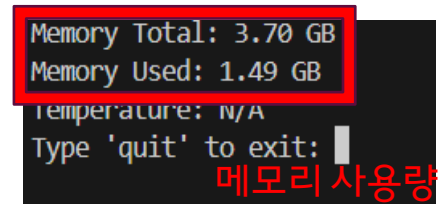
최고 인식률: 62% < **62.6%** < 63.8%

최저 인식률: 50% < **50.8%** < 51.2%

로딩 속도: 1.7초 < **41.1초** < 200초

메모리 사용량: 36% < **40.27%** < 57%

최초 얼굴 인식 속도: 2.8초 < **3.6초** < 8.6초



고찰

1. 실시간 영상으로 프로젝트를 진행하고 싶었으나 카메라 연결에 어려움을 겪어 동영상으로 진행한 점이 아쉬웠습니다.

2. 쓰레딩 라이브러리로 동작을 가속화하고 싶었지만 시간 및 능력 부족으로 인해 시도하다 멈춘 것이 아쉬웠습니다.

3. 사진을 많이 넣어도 생각보다 인식률이 낮았습니다.

4. 여러가지 라이브러리를 라즈베리파이에서 구현에 성공했을 때 성취감을 느꼈습니다.

참고자료

https://github.com/ageitgey/face_recognition/blob/master/README_Korean.md

https://ukayzm.github.io/python-face-recognition/#%EC%86%8C%EC%8A%A4%EC%BD%94%EB%93%9C-%EC%84%A4%EB%AA%85

감사합니다!