

[dlib 라이브러리 ResNet 모델을 이용한 Face-recognition 프로그램]

- 얼굴 인식을 이용한 특정 인물 탐지 및 피대상자 비식별 조치 -

연구 기간: 2021.12.01 ~ 2021.12.31

| | | |
|-------|---|--|
| 작성 | (한국소프트웨어산업협회-실시간 빅데이터분석 및 시각인지 시스템 개발자 양성과정) | BLUR_able 블러블 윤다영 김다솜 김다영 원석민 |
| 버전 정보 | 버전 1.0 2021 년 12 월 17 일 | |

< 목 차 >

| | |
|--------------------------------|----|
| 1. 연구 개요 | 4 |
| 1.1 주제선정배경..... | 4 |
| 1.2 연구 목표 | 5 |
| 1.3 연구 절차 | 5 |
| 2. 모델 비교분석 | 6 |
| 3. 기본 설정 | 17 |
| 4. 코드 구현 | 18 |
| 4.1 객체 정의 | 18 |
| 4.2 얼굴 탐지 함수 정의 | 18 |
| 4.3 랜드마크 거리 계산 | 19 |
| 4.4 이미지 학습 및 배열 변환(DB 생성)..... | 19 |
| 4.5 특정얼굴 탐지 및 비식별조치..... | 22 |
| 4.6 실시간 웹캠 이용..... | 24 |
| 5. 결론 및 한계점 | 27 |
| 6. 참고 문헌 | 29 |
| 7. 별첨[표 및 그림] | 30 |

1. 연구 개요

1.1 주제선정배경

정보통신 기술 인프라가 개선되고 스마트폰과 같은 기기 성능이 고도화되면서 유튜브 등의 플랫폼을 통한 1인 제작 동영상 콘텐츠가 흔해졌으나 초상권 보호 조치는 미흡한 문제가 지속적으로 발생하고 있다. 최근 이슈로 떠오른 교사 브이로그 콘텐츠에서의 학생 얼굴 노출로 인한 개인 신상 정보 악용 범죄에 대한 우려가 심각해지고 있는 바이다.

대법원 판례[2006.10.13 선고 2004 다 16280]¹에 따르면 사람은 누구나 자신의 얼굴 기타 사회통념상 특정인임을 식별할 수 있는 신체적 특징에 관하여 함부로 촬영 또는 그림묘사되거나 공표되지 아니하며 영리적으로 이용당하지 않을 권리를 가지는데, 이러한 초상권은 우리 헌법 제 10 조 제 1 문에 의하여 헌법적으로 보장되는 권리이다. 초상권을 침해했을 때 민사상 불법행위로 간주되며 손해배상청구 대상이 될 수 있다.

1인 미디어 확대의 흐름 속에서 야기된 초상권 관련 사회적문제를 해소하는 기술에 대한 필요성이 대두됨에 따라 본 연구를 진행하고자 한다.

¹ 대법원 2006.10.13 선고 2004 다 16280 판결[위자료][공 2006.11.15(262), 1897]

1.2 연구 목표

상기의 문제를 해결하기 위해 얼굴을 식별하여 학습된 특정 인물의 얼굴을 제외한 얼굴들에 대하여 비식별 조치(모자이크) 처리 함으로써 초상권을 보호하고 콘텐츠 제작자의 사용 편의를 제고시키는 기술을 제안한다. 이 연구의 핵심은 실생활 콘텐츠(사진 및 비디오)에서 인물의 초상권을 보호하기 위한 주요 기술인 얼굴 식별 기술의 정확도를 높이기 위한 부분이다. 얼굴 인식에 특화된 딥러닝 알고리즘과 관련된 선행 연구의 다양한 모델을 비교분석하고, 이를 바탕으로 얼굴 식별의 정확도를 높이기 위하여 모델을 선정하며 실시간 비디오에 적용하기 위하여 알고리즘을 추가·변경한다. 궁극적으로 본 연구는 대중이 접근하기 용이한 초상권 침해 방지 수단을 제공하는데에 그 목적이 있다.

1.3 연구 절차

본 연구는 아래와 같은 절차로 전개되었다.

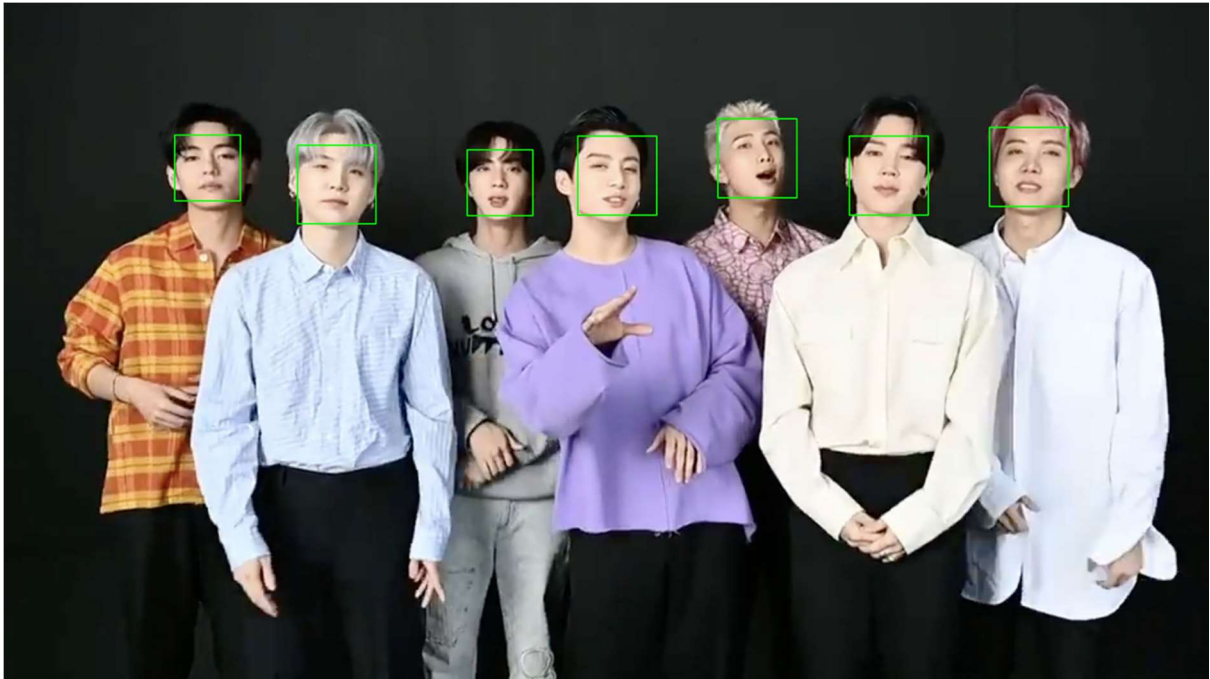
1. 학습된 얼굴인식 모델(라이브러리) 및 참고자료 탐색
2. 탐색된 정보를 바탕으로 인식 및 분류할 대상 설정
3. 데이터베이스에 인식하고자 하는 얼굴 랜드마크 저장
4. 사진 혹은 동영상(실시간 웹캠)에서 얼굴을 탐지하고 랜드마크 추출
5. 데이터베이스에 저장된 랜드마크와 새로 추출된 랜드마크의 간의 거리를 측정
6. 유클리드 거리 값(0.45 이하) 혹은 코사인 유사도(0.4 이하)를 계산하여 저장된 얼굴로 인식 해당 구간 초과 시 피대상자로 인식
7. 피대상자를 비식별(모자이크) 처리

2. 모델 비교분석

다양한 얼굴 탐지 모델 중 7 개 모델에 대해 비교 분석한 결과, 모든 모델에서 7 개의 얼굴에 대한 정면 사진의 경우 인식률 100% 였으며, 정면이 아닌 얼굴이 있는 사진의 경우 사용된 모델에 따라 인식률에 차이를 보였다.

| 모델 | 정면사진 인식 결과 (명, %) | test 사진 인식 결과 (명, %) | 비고 |
|------------------------|-------------------------|----------------------------|--------------------------------|
| 1 Harr_Cascade | 7 (100%) | 5 (71.4%) | 기타분류기 적용: 2 (28.6%) 3 (43%) |
| 2 dlib_Hog | 7 (100%) | 5 (71.4%) | 측면 2 명 제외 모두 인식 |
| 3 dlib_ResNet | 7 (100%) | 5 (71.4%) | 측면 2 명 제외 모두 인식 |
| 4 DNN_caffemodel | 7 (100%) | 0(0%) | 전원 인식 불가 |
| 5 DNN_deepface | 7 (100%) | 7 (100%) | 개별 인식 불가 |
| 6 DNN_Face_recognition | 5 (71.4%) | 5 (71.4%) | 학습데이터 오류: 다른 얼굴을 같은 얼굴로 오분류 |
| 7 cvlib | 7 (100%) | 0(0%) | 전원인식불가 |

[그림 1] 모든 모델의 성능 테스트에 사용한 정면 사진(7 개 얼굴)과 테스트 결과



[그림 2] 모델 성능 테스트에 사용한 측면 얼굴을 포함한 사진(7 개의 얼굴)



2.1 cascadeClassifier

cascadeClassifier 클래스는 OpenCV 에서 가장 많이 사용되는 알고리즘 중 하나로 사전에 학습된 데이터를 불러오고 특징분류기를 이용하여 객체를 검출 할 수 있다. 위치나 크기에 상관없이 얼굴을 감지 할 수 있고 속도가 빨라 실시간 비디오 스트림 감지에도 사용할 수 있다는 장점이 있는 반면 정면 이미지에 효과가 크고 오탐지 확률이 높다는 단점이 있다.

2.1.1 detectMultiScale

OpenCV cascadeClassifier 의 핵심이 되는 함수라 할 수 있으며 input 이미지에서 크기가 다른 object 를 detect 하고 이를 rect 리스트로 반환한다. 어떤 object 를 detect 할 것인지는 cascadeClassifier 초기화 부분에서 정한다. 보통 얼굴 인식을 하기 위해서 face 관련 xml 을 이용하고 그 외에 눈, 입, 부분 신체 감지 등의 분류기를 사용할 수 있다.


```
face_cascade = cv2.CascadeClassifier('haarcascade_frontface.xml')
img = cv2.imread('bts_test.jpeg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]

cv2.imshow('img', img)
```

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
#eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
profile_cascade = cv2.CascadeClassifier('haarcascade_profileface.xml')
```

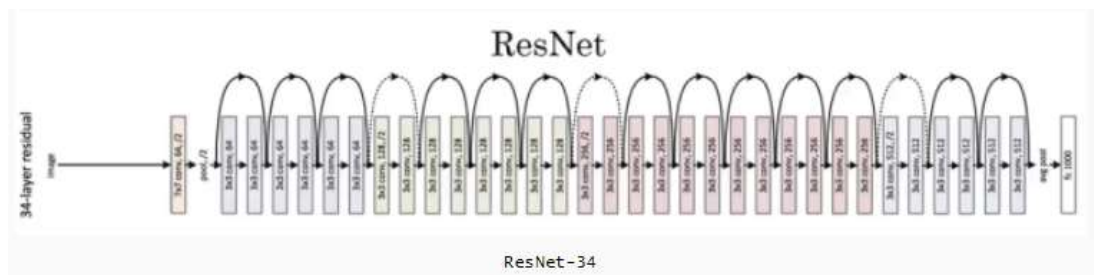

[표 1] haar 특징분류기 종류에 따른 얼굴 감지 성능 비교

| 분류기 | Test 결과 | 인식결과 (명, %) |
|---|--|----------------|
| haarcascade _frontface.xml |  | 2 명 (28.6%) |
| haarcascade _frontalface _default.xml |  | 3 명 (43.0%) |
| haarcascade _frontalface _default.xml & haarcascade _profileface. xml |  | 5 명 (71.4%) |

2.2 dlib

dlib 라이브러리는 OpenCV와 유사하게 이미지 프로세싱에 폭넓게 사용되고 있다. 주로 얼굴 탐지, 정렬 모듈의 사용 빈도가 높고 나아가 실시간 사용이 가능한 얼굴인식 모듈도 제공한다. C++로 작성되었으나 python 인터페이스에서도 구동 가능하다. dlib은 ResNet-34 model에서 몇몇 레이어를 제거하고 29개의 합성곱 레이어로 구성된 신경망을 재구성했다. 150x150x3 크기의 입력을 받고 128차원 벡터로 얼굴 이미지를 표현한다. 다양한 데이터셋으로 훈련한 결과 dlib은 사람의 정확도(97.53%)보다 높은 99.35%의 정확도 구현한다.

[그림 3] ResNet-34




2.2.1 HOG detector

HOG(Histogram of Oriented Gradients) 특성을 사용하여 얼굴 검출

```
hogFaceDetector = dlib.get_frontal_face_detector()
faces = hogFaceDetector(gray,1)
```

[표 2] HOG detector 를 이용한 얼굴 감지 성능

| 모델 | Test 결과 | 인식결과 (명, %) |
|-----|--|----------------|
| HOG |  | 5 명 (71.4%) |

2.2.2 ResNet

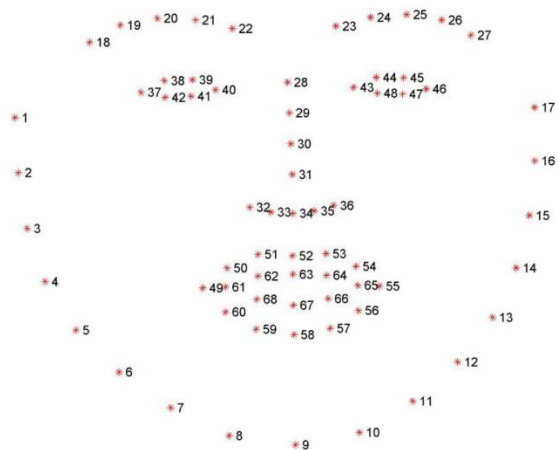
학습된 68 개의 랜드마크를 활용하여 그 거리를 계산하고 임계치 이하일 때 학습된 얼굴로 인식

```


detector = dlib.get_frontal_face_detector()
sp = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
facerec = dlib.face_recognition_model_v1('dlib_face_recognition_resnet_model_v1.dat')

```

[그림 4] 68 개 랜드마크 추출 결과



[표 3] ResNet 을 이용한 얼굴 감지 성능

| 모델 | Test 결과 | 인식결과 (명, %) |
|--------|--|----------------|
| ResNet |  | 5 명 (71.4%) |

2.3 DNN

다수의 입력 데이터의 가중합을 계산하여 하나의 출력값을 찾는다. Gradient descent, error backpropagation 등이 알고리즘을 사용하여 가중치와 편향을 결정해간다. OpenCV 3.3 이후 딥러닝 기반 얼굴 감지기 모듈이 포함되었으며, 사전 학습된 딥러닝 얼굴 감지기 모델은 빠르고 정확한 편이다.

2.3.1 Caffe

DNN 의 Caffe 모델은 아키텍처 정의 파일과 실제 레이어 가중치 파일을 불러와 얼굴을 감지한다. 이미지를 blob 객체로 준비한 후 setInput() 함수로 이미지를 모델에 입력하고 forward() 함수를 이용해 순방향 네트워크를 실행하고 얼굴 감지 결과를 얻는다.

```
net = cv2.dnn.readNetFromCaffe("deploy.prototxt", "res10_300x300_ssd_iter_140000.caffemodel")
```

```
image = cv2.imread('bts_test.jpg')
(h,w) = image.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(image,(300,300)),1.0,(300,300),(104.0, 177.0, 123.0))
```

```
net.setInput(blob)
detections = net.forward()
```

[표 4] caffe 모델을 이용한 얼굴 감지 성능

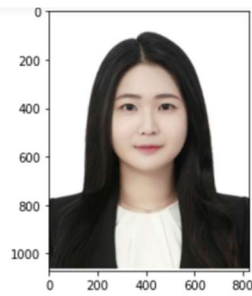
| 모델 | Test 결과 | 인식결과 (명, %) |
|-------|---|----------------|
| caffe |  | 0 명 (0%) |

2.3.2 DeepFace

```
from deepface import DeepFace
import cv2
import matplotlib.pyplot as plt
def verify(img1_path,img2_path):
    img1= cv2.imread(img1_path)
    img2= cv2.imread(img2_path)

    plt.imshow(img1[:,:,:-1])
    plt.show()
    plt.imshow(img2[:,:,:-1])
    plt.show()
    output = DeepFace.verify(img1_path,img2_path)
    print(output)
    verification = output['verified']
    if verification:
        print('They are same')
    else:
        print('The are not same')
verify('bts_test.jpeg', 'bts_1.jpeg')

{'verified': True, 'distance': 0.2830423765223782, 'max_threshold_to_verify': 0.4, 'model': 'VGG-Face', 'similarity_m
etric': 'cosine'}
They are same
```



```
{'verified': True, 'distance': 0.10025245456194487, 'max_threshold_to_verify': 0.4, 'model': 'VGG-Face', 'similarity_
metric': 'cosine'}
They are same
```





```
{'verified': True, 'distance': 0.2830423765223782, 'max_threshold_to_verify': 0.4, 'model': 'VGG-Face', 'similarity_m
etric': 'cosine'}
They are same
```

한 개의 얼굴 사진을 테스트 하여 유사도를 판별하였으나 대부분 높은 정확도를 유지하였다. 7 개의 얼굴 사진의 유사도를 판별하기 위해 단체사진으로 테스트 한 결과, 개별 7 개 얼굴을 분리하여 인식하는 것이 아니라, 사진을 전체를 인식하고 유사도를 계산하여 같은 사람임을 판단한다.

2.3.3 face_recognition

[표 5] face-recognition 을 이용한 얼굴 감지 성능

| 모델 | face_recognition |
|-------|---|
| 학습데이터 |  |
| 결과 |  |

7 개의 얼굴에 대하여 학습 데이터 각 1000 장, 총 7000 장 학습 시 얼굴 탐지 및 인물 식별 결과 정확도가 100%였다. 완전한 측면사진은 앞선 알고리즘과 같이 검출이 불가하였으나 개별 얼굴은 검출에 성공했다. 1 대 1 검출은 인식률이 좋은 편이나 인식하는 얼굴이 많아질수록 인식률이 현저하게 떨어진다.

Min_confidence 를 0.5 에서 0.3 으로 줄여나가며 테스트 한 결과 0.4 일 때 신뢰도와 인식률이 평이하게 나타났다. 학습 데이터를 이름으로 읽어들이어 분류하였으나 오분류 되는 문제점이 발생했다.

2.4 cvlib

사용하기 간편한 컴퓨터비전 오픈소스이며 쉽고 빠른 연구를 가능하게 하는 것에 초점을 두고 개발되었다. cvlib 은 딥러닝 Keras 에서 많은 영향을 받았고 사전 학습된 caffemodel 을 바탕으로 한다. detect_face() 함수로 바운딩 박스와 confidence 가 표시된 얼굴 감지가 가능하다.

```
import cvlib as cv
faces, confidences = cv.detect_face(image)
```

[표 6] cvlib 을 이용한 얼굴 감지 성능

| 모델 | Test 결과 | 인식결과 (명, %) |
|-------|--|----------------|
| cvlib |  | 0 명 (0%) |

3. 기본 설정

다음의 코드는 Anaconda Prompt 에서 실행 한다.

```
pip install opencv-contrib-python
```

openCV 주요 모듈 및 추가 모듈 설치

```
pip install cmake
```

```
Collecting cmake
  Downloading cmake-3.22.1-py2.py3-none-win_amd64.whl (38.1 MB)
Installing collected packages: cmake
Successfully installed cmake-3.22.1
```

빌드 자동화를 위한 CMake(Cross Platform Make) 설치

```
pip install dlib
```

```
Installing collected packages: dlib
Successfully installed dlib-19.19.0
```

이미지 처리를 위하여 C++로 작성된 범용 크로스 플랫폼 라이브러리 dlib 설치

- * Cmake – dlib 순차 설치
- * Visual Studio 설치시 Windows 용 C++ CMake 도구로 설치
- * 에러 발생 시 파이썬 버전 확인 후 해당 dlib wheel(.whl) 파일 다이렉트 설치
(cmd 실행 후 해당 경로에서 pip install dlib-버전.whl)

4. 코드 구현

코드 구현에 필요한 각종 라이브러리 및 모듈을 import 한다.

```
import dlib, cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import ImageFont, ImageDraw, Image
import tensorflow.keras
from tensorflow.keras import backend as K
```

4.1 객체정의

dlib 라이브러리 내장 함수로 객체를 정의한다.

```
detector = dlib.get_frontal_face_detector()
sp = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
facerec = dlib.face_recognition_model_v1('dlib_face_recognition_resnet_model_v1.dat')
```

4.2 얼굴탐지 함수 정의

```
def find_faces(img):
    dets = detector(img, 1)

    if len(dets) == 0:
        return np.empty(0), np.empty(0), np.empty(0)

    rects, shapes = [], []
    shapes_np = np.zeros((len(dets), 68, 2), dtype= int)
    for k, d in enumerate(dets):
```

```

rect = ((d.left(), d.top()), (d.right(), d.bottom()))
rects.append(rect)

shape = sp(img, d)

# dlib 형태를 numpy array 로 변환
for i in range(0, 68):
    shapes_np[k][i] = (shape.part(i).x, shape.part(i).y)

shapes.append(shape)

return rects, shapes, shapes_np

```

4.3 랜드마크 거리 계산

```

def encode_faces(img, shapes):
    face_descriptors = []
    for shape in shapes:
        face_descriptor = facerec.compute_face_descriptor(img, shape)
        face_descriptors.append(np.array(face_descriptor))

    return np.array(face_descriptors)

```

4.4 이미지 학습 및 학습된 이미지 배열로 변환(DB 생성)

```

img_paths = {
    '정국 1': '정국 1.jpeg',
    '정국 2' : '정국 2.jpeg',
    '정국 3': '정국_측면.jpeg'
}

```

```

descs = []

for name, img_path in img_paths.items():
    img = cv2.imread(img_path)
    _, img_shapes, _ = find_faces(img)
    descs.append([name, encode_faces(img, img_shapes)[0]])

np.save('images/descs.npy', descs)
print(descs)

```

```

[['정국 1', array([-6.42565116e-02,  1.67656913e-02,  1.09558143e-02, -1.03970215e-01,
 -6.32530674e-02, -4.79219817e-02, -1.03603899e-01, -1.01171561e-01,
  1.23213857e-01, -1.25224471e-01,  2.15959981e-01, -9.87264812e-02,
 -1.82047710e-01, -5.88646606e-02, -7.53460005e-02,  1.89463228e-01,
 -1.64266288e-01, -1.61002964e-01, -5.99844828e-02,  6.47784770e-03,
  1.00731090e-01,  4.02793065e-02, -4.91798334e-02,  8.15239325e-02,
 -1.18247569e-01, -2.68381476e-01, -1.00011528e-01, -4.03784588e-02,
 -2.52770521e-02, -7.82238990e-02, -3.13604102e-02,  2.59612501e-03,
 -1.88055992e-01, -1.06380507e-02,  8.15455839e-02,  1.40466124e-01,
  1.00518167e-02, -1.36288464e-01,  1.54881939e-01,  8.51102918e-03,
 -2.77306587e-01,  1.10800654e-01,  1.04618669e-01,  2.02128023e-01,
  1.85929865e-01, -1.66696906e-02, -4.04499620e-02, -2.23387659e-01,
  1.42773464e-01, -1.84091687e-01,  3.10075544e-02,  1.68163657e-01,
  1.56045770e-02,  6.02949150e-02,  1.93467289e-02, -1.06840484e-01,
  4.55908775e-02,  1.67973116e-01, -1.35017231e-01, -3.89719643e-02,
  9.42798406e-02, -3.17824408e-02, -1.32597052e-02, -1.38711810e-01,
  1.85645789e-01,  9.08341855e-02, -1.40306547e-01, -2.59748459e-01,
  8.65694433e-02, -1.63061500e-01, -1.11514315e-01,  1.33251920e-01,
 -1.44703194e-01, -1.62621886e-01, -2.95003414e-01,  1.33166015e-02,
  3.86625081e-01,  1.11073643e-01, -1.85457170e-01,  1.35781467e-01,
 -5.85894957e-02,  8.21418315e-03,  1.42083585e-01,  1.52244315e-01,
  5.81468046e-02,  4.06562090e-02, -7.16772750e-02,  7.90639222e-03,
  2.69650400e-01, -2.49183849e-02, -6.74192421e-03,  1.96166903e-01,
 -9.97232273e-03,  8.37102085e-02,  3.61586772e-02,  1.14262104e-04,
 -7.98895359e-02, -4.05825675e-04, -1.57400042e-01, -5.25964722e-02,
 -8.63356888e-03,  6.42675236e-02, -1.76072121e-03,  1.25857919e-01,
 -1.16175212e-01,  1.11205459e-01,  1.56629086e-02, -2.32450664e-04,
  3.59529257e-02,  5.92060387e-03, -5.01541197e-02, -4.44667637e-02,

```

```

1.06367469e-01, -2.51424611e-01, 1.91114977e-01, 8.53386149e-02,
1.56009316e-01, 5.46762384e-02, 1.59718946e-01, 8.24508816e-02,
-3.27838585e-02, -1.70516074e-02, -2.35824764e-01, 3.43621485e-02,
7.34194741e-02, -1.44532919e-02, 1.21290371e-01, -3.47564369e-02)], ['정국 2', array([-
0.07148495, 0.11425725, 0.06253581, -0.10690188, -0.09941487,
-0.08501741, -0.08171534, -0.09785117, 0.15395965, -0.09816284,
0.28578439, -0.12025108, -0.14225687, 0.00103408, -0.04254451,
0.17682259, -0.20171794, -0.14509757, -0.10039496, 0.09454692,
0.09140532, 0.0425202, -0.00928, 0.04867757, -0.11941779,
-0.34667844, -0.10253531, -0.01515473, -0.00964222, -0.07452801,
0.01766761, 0.02648807, -0.20726913, -0.01487788, 0.08887869,
0.10004792, 0.03476758, -0.10910901, 0.15653667, -0.03172307,
-0.31890425, 0.04772951, 0.06850426, 0.2178935, 0.21539043,
0.04904386, -0.01468089, -0.19102232, 0.13831848, -0.1695049,
0.04905201, 0.16501343, -0.01624366, 0.05554654, 0.01558506,
-0.11547321, 0.01915479, 0.21469846, -0.12860695, -0.06081405,
0.10098044, -0.00914313, 0.04478461, -0.13670263, 0.17670645,
0.07506244, -0.12618577, -0.23807874, 0.11022157, -0.12883976,
-0.11276143, 0.08346882, -0.18198623, -0.18010211, -0.32779458,
-0.04953894, 0.41420767, 0.0877009, -0.19818208, 0.05527357,
-0.08007935, 0.01013639, 0.09048544, 0.10858564, 0.04874714,
0.00115237, -0.13378422, -0.0095827, 0.33006114, -0.0551983,
-0.00983064, 0.20516858, -0.05294115, 0.00546874, -0.03217182,
0.01145869, -0.09259132, 0.04516383, -0.16321123, -0.05892272,
-0.03176463, 0.08076277, -0.0675854, 0.20436424, -0.19584787,
0.13845791, 0.0281677, 0.03868381, 0.00786148, 0.03414303,
-0.02718683, -0.08903971, 0.1152713, -0.19662911, 0.20098925,
0.13569504, 0.1140054, 0.09009734, 0.10209613, 0.08633729,
-0.00853087, 0.00664987, -0.23421128, -0.02433902, 0.04516517,
0.04317452, 0.09251611, 0.00759599)]), ['정국 3', array([-0.06465794, 0.07587674,
0.04304142, -0.09773387, -0.0910074,
-0.02283255, -0.07404677, -0.04131995, 0.14635617, -0.12019812,
0.21983413, -0.06007144, -0.16006263, 0.00179871, 0.02427702,
0.16741379, -0.1765222, -0.1808178, -0.05848839, -0.01700538,
0.07125233, 0.06121811, -0.04297529, 0.09349945, -0.1879084,
-0.26629624, -0.07168238, -0.06181313, -0.04251898, -0.07686955,
-0.01606777, 0.00654933, -0.19864313, -0.00980709, 0.08131678,
0.11969268, 0.03722405, -0.05916904, 0.15356454, 0.00295817,
-0.26991218, 0.11713493, 0.11191545, 0.21193935, 0.18453062,
0.06424341, -0.02949638, -0.1746103, 0.15324995, -0.16346253,
0.0544078, 0.13812797, 0.04196931, 0.06873585, 0.02267804,

```

```
-0.15200266, 0.04131022, 0.14335527, -0.1355488, -0.03035641,
0.0514878, -0.00768455, 0.03539322, -0.10610972, 0.22885239,
0.04406478, -0.11645851, -0.21563715, 0.0755468, -0.16208956,
-0.16368195, 0.12706642, -0.1433925, -0.1498881, -0.32708928,
0.01767375, 0.35251647, 0.10468224, -0.22896048, 0.11454249,
0.01376326, -0.00500709, 0.09225172, 0.17327435, -0.01007811,
0.05955774, -0.08130816, 0.0022281, 0.29813033, 0.00118864,
0.00270817, 0.20937794, 0.01842932, 0.08245087, 0.04343946,
0.01540203, -0.09611444, -0.02460172, -0.17805649, -0.07498424,
-0.02292819, 0.08590406, -0.05375417, 0.1670799, -0.20324311,
0.15582117, -0.03045036, -0.03401581, -0.06325556, 0.07577176,
-0.01143683, -0.05183654, 0.10129884, -0.23451641, 0.14088093,
0.13845435, 0.09070759, 0.08343071, 0.1145141, 0.06200586,
-0.03706247, 0.04207982, -0.24202126, -0.01178316, 0.10293601,
-0.05432028, 0.14627019, 0.01571271]]]]
```

[표 7] 학습 이미지 3 장

| | | |
|---|---|---|
|  |  |  |
| 정국 1(정면) | 정국 2(정면) | 정국 3(측면) |

4.5 이미지에서 특정 얼굴 탐지 및 피대상자비식별조치(모자이크처리)

```
img = cv2.imread('bts_test.jpeg')

rects, shapes, _ = find_faces(img) # 얼굴 찾기
descriptors = encode_faces(img, shapes) # 인코딩
```

```

for i, desc in enumerate(descriptors):
    x = rects[i][0][0] # 얼굴 X 좌표
    y = rects[i][0][1] # 얼굴 Y 좌표
    w = rects[i][1][1]-rects[i][0][1] # 얼굴 너비
    h = rects[i][1][0]-rects[i][0][0] # 얼굴 높이

    # 추출된 랜드마크와 데이터베이스의 랜드마크들 중 제일 짧은
    # 거리를 찾는 부분
    descsl = sorted(descs, key=lambda x: np.linalg.norm([desc]
- x[1]))
    dist = np.linalg.norm([desc] - descsl[0][1], axis=1)

    if dist < 0.45: # 그 거리가 0.45 보다 작다면 그 사람으로 판단
Threshold 값은 실험적으로 동양인의 얼굴은 이 값이 0.4 ~ 0.45 정도의 값이 적당
        name = descsl[0][0]
    else:
        # 0.45 보다 크다면 모르는 사람으로 판단 ->
        # 모자이크 처리
        mosaic_img = cv2.resize(img[y:y+h, x:x+w], dsize=(0,
0), fx=0.04, fy=0.04) # 축소
        mosaic_img = cv2.resize(mosaic_img, (w, h),
interpolation=cv2.INTER_AREA) # 확대
        #img[y:y+h, x:x+w] = mosaic_img # 인식된 얼굴 영역
        # 모자이크 처리

        cv2.rectangle(img, (x, y), (x+w, y+h), (0,255,0), 2) # 얼굴
영역 박스
        cv2.putText(img, str(dist)[1:6], (x+5,y+h-
5),cv2.FONT_HERSHEY_PLAIN, 2, (0,0,255), 4) # 사진에 거리 출력

        img = Image.fromarray(img)
        draw = ImageDraw.Draw(img)
        draw.text((x+5,y-50), name,
font=ImageFont.truetype("./batang.ttc", 60),
fill=(255,255,255))
        img = np.array(img)

        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

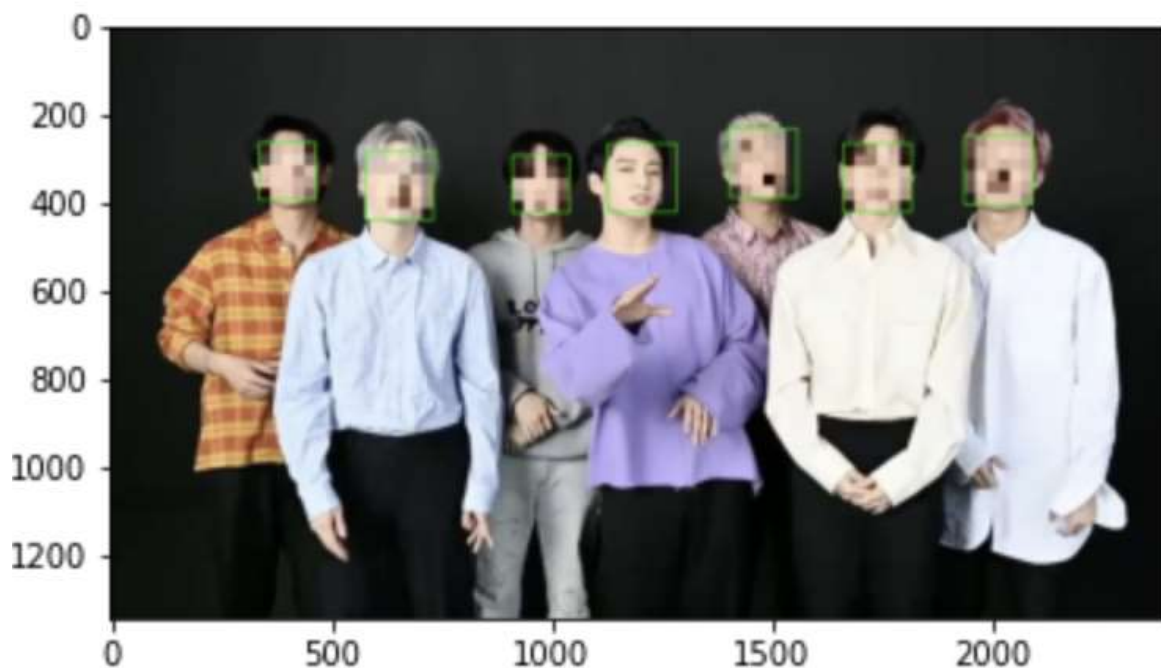
```

```

cv2.imshow('camera', img)
k = cv2.waitKey(10) & 0xff # 'ESC' 키 누르면 종료
if k == 27:
    break
cv2.destroyAllWindows()
plt.imshow(img)

```

[그림 5] 이미지 비식별 조치 결과



4.6 실시간 웹캠을 이용한 얼굴 인식 및 비식별조치

```

cam = cv2.VideoCapture(0) # 노트북 웹캠 사용
cam.set(3, 640) # 너비
cam.set(4, 480) # 높이
font = cv2.FONT_HERSHEY_SIMPLEX # 출력 시 사용폰트

while True:

    ret, img = cam.read()

```



```

img = cv2.flip(img, 1) # 좌우 대칭
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rects, shapes, _ = find_faces(img) # 얼굴 찾기
descriptors = encode_faces(img, shapes) # 인코딩

for i, desc in enumerate(descriptors):
    x = rects[i][0][0] # 얼굴 X 좌표
    y = rects[i][0][1] # 얼굴 Y 좌표
    w = rects[i][1][1]-rects[i][0][1] # 얼굴 너비
    h = rects[i][1][0]-rects[i][0][0] # 얼굴 높이

    # 추출된 랜드마크와 DB의 랜드마크 중 제일 짧은 거리를 찾는 부분
    descsl = sorted(descsl, key=lambda x: np.linalg.norm([desc] -
x[1]))
    dist = np.linalg.norm([desc] - descsl[0][1], axis=1)

    if dist < 0.45: # 그 거리가 0.45 보다 작다면 그 사람으로 판단
        name = descsl[0][0]
    else:          # 0.45 보다 크다면 모르는 사람으로 판단 ->
모자이크 처리
        name = "누구?"
        mosaic_img = cv2.resize(img[y:y+h, x:x+w], dsize=(0, 0),
fx=0.04, fy=0.04) # 축소
        mosaic_img = cv2.resize(mosaic_img, (w, h),
interpolation=cv2.INTER_AREA) # 확대
        img[y:y+h, x:x+w] = mosaic_img # 인식 얼굴 영역 모자이크 처리
        cv2.rectangle(img, (x, y), (x+w, y+h), (0,255,0), 2) # 얼굴 박스
        cv2.putText(img, str(dist)[1:6], (x+5,y+h-5),
cv2.FONT_HERSHEY_PLAIN, 2, (0,0,255), 4) # 사진에 거리 출력

img = Image.fromarray(img)
draw = ImageDraw.Draw(img)
draw.text((x + 5,y - 50), name,
font=ImageFont.truetype("./batang.ttc", 60), fill=(255,255,255))
img = np.array(img)

cv2.imshow('camera', img)

```

```
k = cv2.waitKey(10) & 0xff # 'ESC' 키 누르면 종료
if k == 27:
    break

cam.release()
cv2.destroyAllWindows()
```

[그림 6] 실시간 웹캠 인식 결과-학습된 특정인물 외 비식별 조치



5. 결론 및 한계점

본 연구는 얼굴 인식을 이용한 특정 인물 탐지 및 피대상자 비식별 조치를 통하여 초상권 침해 방지 효과와 초상권에 대한 인식 개선과 콘텐츠 제작자의 사용 편의를 제고 시키고자 진행되었다. 먼저 딥러닝 기반 얼굴 검출, 얼굴 랜드마크 검출 및 얼굴 인식 관련 다양한 모델을 살펴보았다. 8 개의 모델을 비교 분석하여 얼굴을 인식한 결과 측면 얼굴, 배경색에 따라 인식률이 현저하게 달라지는 것을 확인할 수 있었다. 특히, dlib 모델을 활용한 실험에서는 다수의 사진을 학습시켜 검증해보았을 때 오히려 정확도 및 인식률이 떨어지는 결과값을 얻기도 하였다. 또한 눈과 코가 어느 정도 인식이 되느냐에 따라 측면 얼굴이 인식률이 달라지는 실험 결과를 볼 수 있었다.

딥러닝 기반 물체 인식과 얼굴 인식의 경우 CNN 기반 네트워크가 좋은 성능을 보여 최근 많은 방식들이 제안되어 왔고 고용량의 공개 얼굴 Database 가 공개되고 있어서 연구 단계에서 실 제품 에 적용될 수 있는 알고리즘들이 나오고 있다. 딥러닝 및 CNN 알고리즘을 활용한 물체인식과 얼굴인식의 경우 그 성능이 빠르게 발전하고 있고, 상용화 될 수 있는 모델들이 다수 존재한다. 또한 일반적인 CNN 기반 네트워크가 Residual Network 와 같이 변화가 있을 때 이를 바로 얼굴 인식 쪽에 적용하여 성능 개선과 한계점들을 확인해 보고 있는 실정이다. 나아가 단순히 물체 인식용 CNN 을 얼굴에 사용하는 것이 아닌 얼굴 인식에 특화된 네트워크나 방법론에 대한 추가적인 연구가 필요하다.

다수의 얼굴 인식 관련 선행 연구를 비교 분석하여 보다 나은 성능의 모델을 확인하였으나 이에 비해 획기적으로 성능이 향상된 모델을 개발하는 데에는 아쉬운 부분이 있다. 향후 얼굴 인식 모델의 시스템 고도화를 통해 여러 분야에서 응용이 가능할 것이다. 예를 들어 사진이나 영상에 등장하는 군중속에서 범죄자를 식별할 수 있는 탐지 및 추적 기능, 나아가 얼굴인식 뿐 아니라 자동차 표지판, 도로, 가게 간판 등의 비식별조치화에 응용 가능성을 기대해볼 수 있다. 이러한 점에 대해서는 추후의 과제들에서 다루어지기를 기대한다.

6. 참고 문헌

<단행본>

- [1] 허진경, 『케라스를 이용한 인공지능망 딥러닝 알고리즘 구현』, BOOKK(2021).
- [2] 프랑소와솔레, 『케라스 창시자에게 배우는 딥러닝』, 박해선, 길벗(2021).
- [3] 오승환, 『파이썬 딥러닝 머신러닝 입문』, 정보문화사(2021).
- [4] 박해선, 『혼자 공부하는 머신러닝+딥러닝』, 한빛미디어(2021).
- [5] 오승환, 『파이썬 딥러닝 머신러닝 입문』, 정보문화사(2021).
- [6] 허진경, 『파이썬 OpenCV 를 이용한 영상처리』, BOOKK(2021).
- [7] 황선규, 『openCV 로 배우는 컴퓨터 비전과 머신러닝』, 길벗(2021).

<논문>

- [1] K. Zhang, Z. Zhang, Z. Li and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," IEEE Signal Processing Letters vol. 23, no. 10(2016), pp.1499-1503.
- [2] 황원준, "딥러닝 기반 얼굴 인식 최신 기술 동향", 전자공학회지 제 46 권(2019), pp.15-22.
- [3] 문형진, 김계희, "사용자 인증을 위한 딥러닝 기반 얼굴인식 기술 동향", 산업융합연구 제 17 권 제 3 호(2019), pp.23-29.
- [4] 김태미, 닐평푸, 김형원 "비디오 콘텐츠의 프라이버시 보호를 위한 CNN 기반 얼굴 추적 및 재식별 기술", 한국정보통신학회논문지 제 25 권 제 1 호(2021), pp.63-68.

<기사 및 웹게시물>

- [1] 정우태, "초상권침해에 무감각...신고건수 4 년새 급증", 영남일보, 2019 년 10 월 23 일, <https://m.yeongnam.com/view.php?key=20191023.010010708370001>
- [2] 김준영, "유튜브 영상에 내 얼굴이...보호 못 받는 SNS 초상권", 세계일보, 2019 년 6 월 22 일, <https://www.segye.com/newsView/20190620516449>
- [3] 정재욱, "유튜브 영상에 내 얼굴이...초상권 침해인가요?", 주간조선, 2021 년 7 월 28 일, <http://weekly.chosun.com/client/news/viw.asp?ctcd=C02&nNewsNumb=002668100010>
- [4] HyunDong Lee(easttwave), 2021 년 4 월 8 일, <https://velog.io/@easttwave/Deep-Learning-%EC%96%BC%EA%B5%B4-%EC%9D%B8%EC%8B%9D-%EB%AA%A8%EB%8D%B8-%EB%B9%84%EA%B5%90-%EC%A1%B0%EC%82%AC>
- [5] Adrian Rosebrock, 2019 년 12 월 16 일, <https://www.pyimagesearch.com/2019/12/16/training-a-custom-dlib-shape-predictor/>

7. 별첨[표 및 그림]

7.1 표 차례

| | |
|---|----|
| [표 1] haar 특징분류기 종류에 따른 얼굴 감지 성능 비교 | 9 |
| [표 2] HOG detector 를 이용한 얼굴 감지 성능 | 11 |
| [표 3] ResNet 을 이용한 얼굴 감지 성능 | 12 |
| [표 4] caffe 모델을 이용한 얼굴 감지 성능 | 13 |
| [표 5] face-recognition 을 이용한 얼굴 감지 성능 | 15 |
| [표 6] cvlib 을 이용한 얼굴 감지 성능 | 16 |
| [표 7] 학습 이미지 3 장 | 22 |

7.2 그림 차례

| | |
|--|----|
| [그림 1] 모든 모델의 성능 테스트에 사용한 정면 사진(7 개 얼굴)과 테스트 결과..... | 7 |
| [그림 2] 모델 성능 테스트에 사용한 측면 얼굴을 포함한 사진(7 개의 얼굴) | 7 |
| [그림 3] ResNet-34 | 10 |
| [그림 4] 68 개 랜드마크 추출 결과 | 11 |
| [그림 5] 이미지 비식별 조치 결과 | 24 |
| [그림 6] 실시간 웹캠 인식 결과-학습된 특정인물 외 비식별 조치 | 26 |