2020학년도 학사학위논문 (종합설계보고서)
지도교수 유 성 욱

라즈베리파이를 활용한
공공시설 올바른 마스크 착용 안내 시스템

중앙대학교 전자전기공학부
박 종 현 (20143730)
문 예 인 (20163346)
2020. 11.

목 차

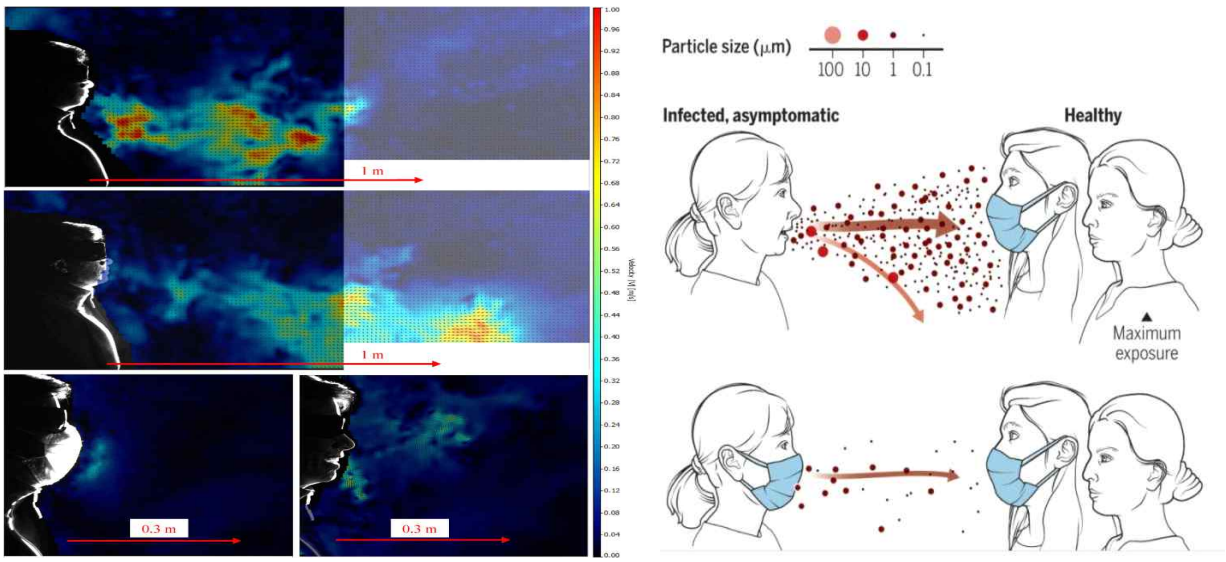
Abstract : 올바른 마스크 착용을 통해 COVID-19로부터 특히 사람이 밀집되는 공공시설 등에서의 전염 위험을 낮추기 위한 시스템을 만들고자 한다. 실시간으로 입과 코를 모두 가려 마스크를 착용했는지 확인하고 착용 상태에 따른 안내 음성 송출을 위해, 경량 영상 특징 추출 기술인 Mobilenet-v2와 라즈베리파이 카메라 모듈을 활용하였다. 또한 효율적인 학습을 위해 Face Recognition을 기반으로 하여 데이터를 추가적으로 생성하였고 각 데이터들을 마스크 착용 상태에 따라 세 가지로 분류하여 학습시켜 98%이상의 정확도를 얻을 수 있었다. 실제 시설에서 적용한다면 마스크를 올바르지 않게 착용한 인원을 통제하고 바이러스 감염 예방수칙의 기본인 마스크 착용에 대한 경각심을 환기시켜주는 역할을 하게 될 것 이다.

Ⅰ. 서 론

1. 연구목적

COVID-19 팬데믹에 가장 화두가 된 것은 마스크이다. 사회적 거리두기 2단계 이상부터 실내 전체 마스크 착용을 의무화하였으나, 마스크로 코 밑에 걸치거나 턱으로 내려 착용한 이른바 '턱스크'의 경우가 많이 발견된다. 그 예로 공항철도 열차 내 마스크 착용 점검 결과 5일간 484건의 적발 사례(하루 평균 97건)이 나왔으며, 이는 마스크를 소지하지 않은 것 뿐 아니라 위와 같은 경우를 포함한 결과이다.

Christian 외 1인은 비말 감염에 있어 마스크를 통한 예방 효과에 대한 연구[1]에서, 마스크를 착용하지 않고
대화를 하는 것이 입부터 코까지 마스크로 가린 채 기침을 하는 것보다 더 많은 비말을 확산시킨다는 것을 보여준다.



<그림1 마스크 착용 유무에 따른 비말확산 가능성>

[2]또한 Yixuan J Hou 외 18인은 COVID-19 감염 경로에 대한 연구에서 비강(코)이 코로나 바이러스에 가작 취약한 부위임을 확인했다. 더불어 코에는 다른 호흡기보다 감염에 취약한 세포가 많기때문에 감염자가 코로 내쉬는 날숨에는 입으로 내뱉는 숨보다 바이러스 입자가 더 많을 수 있다고 지적했다.

이처럼 '턱마스크'를 포함해 코를 내놓은 채 마스크를 쓰는 것은 자신은 물론 잠재적으로 타인에게 피해를 줄 수 있는 행동이다. 하지만 이러한 착용 실태를 수시로 점검하는 것은 많은 제약이 따른다. 이를 위해 특히 사람이 밀집되어 있는 공공장소 등에서 마스크를 올바르게 착용하지 않은 경우 알림음을 통해 안내를 돕는 시스템을 제안하고자 한다.

1.2. 연구 방법

라즈베리 파이 카메라 모듈을 활용해 스트리밍으로 영상을 입력 받아 Convolutional Network로 사람들의 얼굴을 인식하고 바르게 마스크를 착용하였는지 확인한다. 다양한 객체 검출 기술 중 임베디드에서도 실시간으로 작동할 수 있도록 경량화 된 네트워크인 Mobilene-v2을 활용한다.

학습을 위한 데이터는 Flickr-Faces-HQ Dataset(FFHQ)을 사용한다. FFHQ는 비영리 목적으로 라이센스가 허가된 얼굴 데이터셋이며 70,000장의 사진 중 3000 여장을 학습에 활용했다. Landmark detection을 통해 데이터셋 일부에 원하는 수준으로 마스크를 씌우고, 원본 사진과 함께 총 세 가지(with_mask, without_mask, wrong_mask) 레이블로 학습을 진행하였다.

Ⅱ. 학습데이터 생성

1. 데이터 확장의 필요성

사람의 안면에 마스크를 착용 하거나 착용하지 않는 경우 이외에 올바르게 착용하지 않은, 소위 턱마스크, 코마스크라고 불리우는 방법으로 착용하는 사례가 많다. 인식하고자 하는 대상을 정확하게 인식시키도록 하기 위해서는 양질의 데이터를 올바르게 전처리 하여 적절하게 카테고리를 분류하고 데이터를 학습시키는 것이 바람직 하지만, 개개인이 항상 원하는 데이터셋만을 수집하는 것이 쉽지만은 않다. 따라서 데이터를 직접 생성하고 트레이닝 함으로써 분류성능이 뛰어난 모델을 만들고자 하였고 해당 프로젝트에서는 원본 이미지에서 마스크 이미지를 합성하는 방식으로 원본 이미지로부터 데이터를 확장하여 사용하고자 한다.

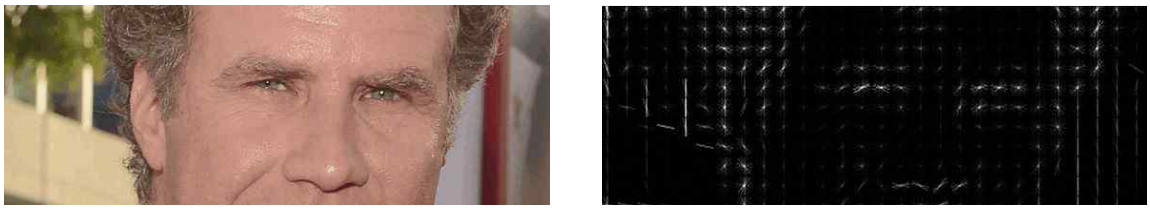
2. 얼굴인식 원리 이용

(1)사진합성 방식

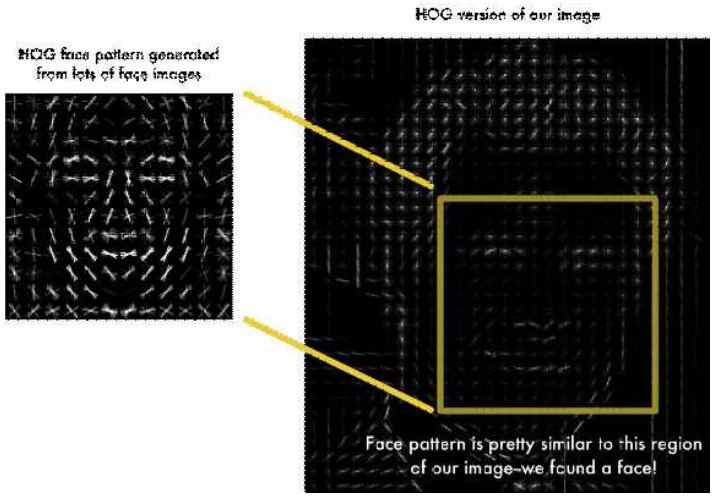
원본이미지에 마스크를 합성하는 방식을 이용하려면 우선 사람 안면의 각도나 이미지의 기울기, 사람 개개인의 얼굴 특징을 고려하여 자연스럽게 합성이 되어야 한다는 전제가 필요하다. 그렇지 않으면 같은 카테고리에 속하는 데이터에서 공통된 특징을 찾아내는데 문제가 생길 것 이고 이는 최종적으로 올바른 학습 모델을 생성하지 못하는 결과를 가져올 것이다. 즉 해당 프로젝트에서 목표로 하는 기능의 성능이 저하된다는 것이다. 따라서 이 방식을 사용하기 위해서 얼굴인식 모듈인 python의 dlib Library와 이를 효율적으로 이용할 수 있도록 구축한 face_recognition Library를 이용하여 얼굴의 특징점(face landmarks)을 추출하고 특정 추출점을 기준으로 마스크이미지(.png) 파일을 일부 변형시켜 원본이미지에 합성하여 새로운 데이터를 생성한다.

(2)Face Recognition/ Face Landmark estimation

얼굴인식 알고리즘을 구현하기 위해서는 우선 영상(사진)에서 사람얼굴을 찾는 과정이 선행되어야 한다. 해당 프로젝트에서는 원본 데이터셋의 모든 파일들은 한명의 사람이 크롭되어있지만, 알고리즘상으로는 얼굴을 찾는과정이 필수이다. 여기서 dlib Library는 Hog(Histogram of Oriented Gradients)솔루션으로 사람의 얼굴을 찾는 방식을 제공한다. [3]Hog solution 은 원본사진을 흑백 스케일으로 변환하고, 16x16 픽셀 단위로 묶어 밝기 값 Gradient를 표현하고 이를 훈련된 hog model과 비교하여 얼굴인지 아닌지를 판별하는 방법이다.

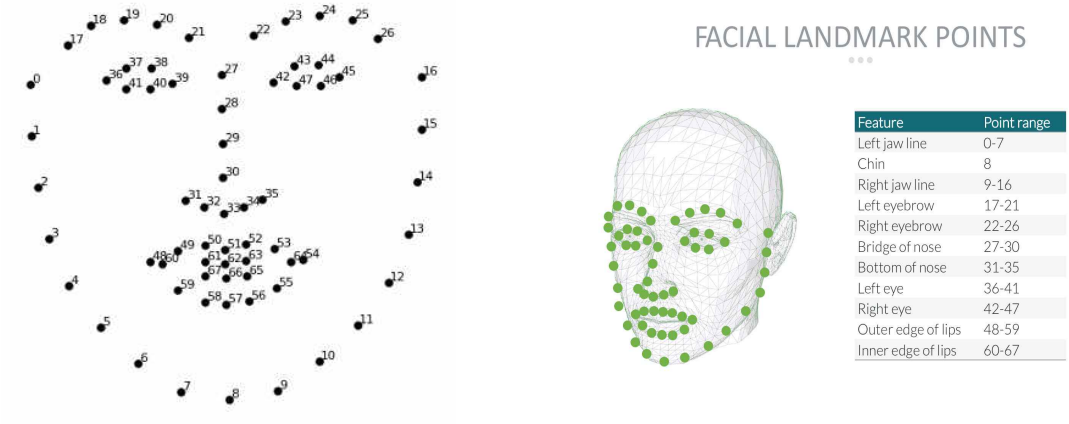


<그림2. Hog Solution을 이용해 추출한 밝기값 변화 Gradient>



<그림3. 훈련된 모델과 Hog version image의 비교 및 얼굴인식>

이 방법으로 얼굴을 찾았다면 그 다음으로 얼굴의 특징점을 찾아내는 것(face landmark estimation) 이다. 여러 솔루션이 많이 있지만 dlib에서는 2014년에 Vahid Kazemi와 Josephine Sullivan이 발명 한 접근 방법을 사용하여 68개의 특징점(턱의 상단, 눈 바깥의 가장자리, 눈썹 안쪽의 가장자리, 등)을 추출해 낸다. 이렇게 추출해 낸 몇가지 포인트를 마스크를 합성하기 위한 기준점으로 사용하게 될 것이다.



<그림4. 68개의 얼굴 부위별 특징점(Landmark)>

3. 데이터 생성 코드

원본 데이터를 확장하여 새로운 카테고리(with_mask, wrong_mask)를 만들어 내기 위해서는 폴더내의 모든 파일들에 마스크 이미지를 합성하는 알고리즘을 적용해야 하므로 FaceMasker라는 class를 저장하는 mask.py 그리고 폴더에 루프를 돌며 모든 파일에 해당 알고리즘을 적용하는 loop_through_folder.py을 이용하였다.

(1)mask.py

```python
import ...


def create_mask(image_path):
    pic_path = image_path
    #마스크 경로
    mask_path = "C:/Users/JongHyeon/Desktop/maskon-final/images/undernose.png"
    show = False
    model = "hog"
    FaceMasker(pic_path, mask_path, show, model).mask()
```

필요한 모듈들을 import 하고 create_mask라는 함수를 정의한다. 이 함수에서 합성할 이미지가 저장되어 있는 디렉터리 지정해주고 사용할 모델을 hog로 설정한다. 그리고 FaceMasker클래스의 mask 매서드를 실행하도록 한다.

```python
class FaceMasker:
    KEY_FACIAL_FEATURES = ('nose_bridge', 'chin')

    def __init__(self, face_path, mask_path, show=False, model='hog'):
        self.face_path = face_path
        self.mask_path = mask_path
        self.show = show
        self.model = model
        self._face_img: ImageFile = None
        self._mask_img: ImageFile = None

    def mask(self):
        import face_recognition

        face_image_np = face_recognition.load_image_file(self.face_path)
        face_locations = face_recognition.face_locations(face_image_np, model=self.model)
        face_landmarks = face_recognition.face_landmarks(face_image_np, face_locations)
        self._face_img = Image.fromarray(face_image_np)
        self._mask_img = Image.open(self.mask_path)

        found_face = False
        for face_landmark in face_landmarks:
            # check whether facial features meet requirement
            skip = False
            for facial_feature in self.KEY_FACIAL_FEATURES:
                if facial_feature not in face_landmark:
                    skip = True
                    break
            if skip:
                continue

            # mask face
            found_face = True
            self._mask_face(face_landmark)

        if found_face:
            if self.show:
                self._face_img.show()

            # save
            self._save()
        else:
            print('Found no face.')

    def _mask_face(self, face_landmark: dict):...

    def _save(self):
        path_splits = os.path.splitext(self.face_path)
        new_face_path = path_splits[0] + '-with-mask' + path_splits[1]
        self._face_img.save(new_face_path)
        print(f'Save to {new_face_path}')

    @staticmethod
    def get_distance_from_point_to_line(point, line_point1, line_point2):
        distance = np.abs((line_point2[1] - line_point1[1]) * point[0] +
                          (line_point1[0] - line_point2[0]) * point[1] +
                          (line_point2[0] - line_point1[0]) * line_point1[1] +
                          (line_point1[1] - line_point2[1]) * line_point1[0]) / \
                   np.sqrt((line_point2[1] - line_point1[1]) * (line_point2[1] - line_point1[1]) +
                           (line_point1[0] - line_point2[0]) * (line_point1[0] - line_point2[0]))
        return int(distance)


if __name__ == '__main__':
    create_mask(image_path)
```

FaceMasker라는 클래스를 선언하고 필요한 파라미터들을 받도록 한다. Key_Facial_feature는 nose bridge, chin(콧대, 턱)이고 mask 매서드에서 얼굴인식을 이용할 수 있도록 face_recognition Library를 임포트 하고 이미지에서 얼굴의 위치, 랜드마크를 저장할 array를 정의한다. for문 이하를 해석하면 Key_Facial_feature인 콧대와 턱을 모두 찾을 때까지 반복문을 실행하고 모두 찾은 경우 얼굴을 찾았다고 입력받고(Found_face = True) 마스크의 위치 결정 및 모양을 변형하는 매서드(_mask_face)를 실행한다. 그리고 Found_face = True상태이므로 이미지를 저장한다.
특징점 간의 절대거리를 계산하는 정적 매소드를 포함시켜 _mask_face 매서드에서 이용하도록 하였다.

```
61      def _mask_face(self, face_landmark: dict):
62          nose_bridge = face_landmark['nose_bridge']
63          nose_point = nose_bridge[len(nose_bridge) * 1 // 4]
64          nose_v = np.array(nose_point)
65
66          chin = face_landmark['chin']
67          chin_len = len(chin)
68          chin_bottom_point = chin[chin_len // 2]
69          chin_bottom_v = np.array(chin_bottom_point)
70          chin_left_point = chin[chin_len // 8]
71          chin_right_point = chin[chin_len * 7 // 8]
72
73          # split mask and resize
74          width = self._mask_img.width
75          height = self._mask_img.height
76          width_ratio = 1.2
77          new_height = int(np.linalg.norm(nose_v - chin_bottom_v))
78
79          # left
80          mask_left_img = self._mask_img.crop((0, 0, width // 2, height))
81          mask_left_width = self.get_distance_from_point_to_line(chin_left_point, nose_point, chin_bottom_point)
82          mask_left_width = int(mask_left_width * width_ratio)
83          mask_left_img = mask_left_img.resize((mask_left_width, new_height))
84
85          # right
86          mask_right_img = self._mask_img.crop((width // 2, 0, width, height))
87          mask_right_width = self.get_distance_from_point_to_line(chin_right_point, nose_point, chin_bottom_point)
88          mask_right_width = int(mask_right_width * width_ratio)
89          mask_right_img = mask_right_img.resize((mask_right_width, new_height))
90
91          # merge mask
92          size = (mask_left_img.width + mask_right_img.width, new_height)
93          mask_img = Image.new('RGBA', size)
94          mask_img.paste(mask_left_img, (0, 0), mask_left_img)
95          mask_img.paste(mask_right_img, (mask_left_img.width, 0), mask_right_img)
96
97          # rotate mask
98          angle = np.arctan2(chin_bottom_point[1] - nose_point[1], chin_bottom_point[0] - nose_point[0])
99          rotated_mask_img = mask_img.rotate(angle, expand=True)
100
101         # calculate mask location
102         center_x = (nose_point[0] + chin_bottom_point[0]) // 2
103         center_y = (nose_point[1] + chin_bottom_point[1]) // 2
104
105         offset = mask_img.width // 2 - mask_left_img.width
106         radian = angle * np.pi / 180
107         box_x = center_x + int(offset * np.cos(radian)) - rotated_mask_img.width // 2
108         box_y = center_y + int(offset * np.sin(radian)) - rotated_mask_img.height // 2
109
110         # add mask
111         self._face_img.paste(mask_img, (box_x, box_y), mask_img)
```

위 코드는 빨간색 박스처리 한 _mask_face 매서드를 자세히 나타낸 것이다.

(1)loop_through_folder.py

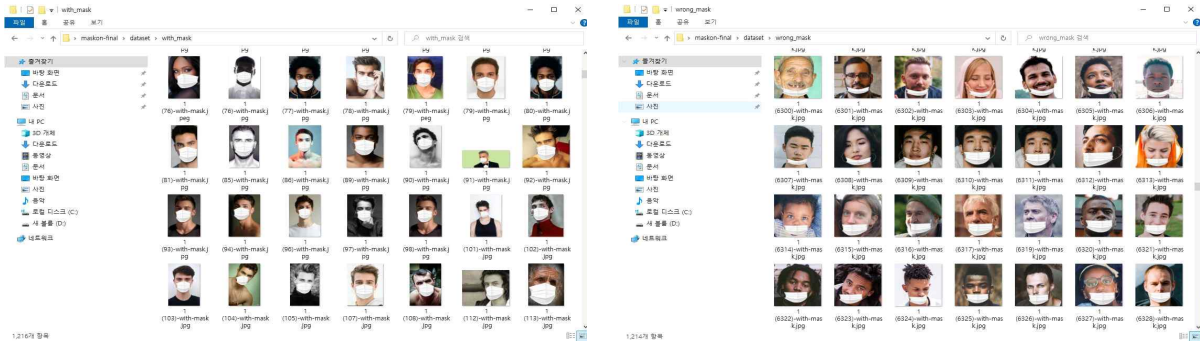
```python
import cv2
import os
from mask import create_mask

#마스크 안낀 얼굴 데이터셋
folder_path = "C:/Users/JongHyeon/Desktop/without_mask"
#dist_path = "/home/preeth/Downloads"

#c = 0
images = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))]
for i in range(len(images)):
    print("the path of the image is", images[i])
    #image = cv2.imread(images[i])
    #c = c + 1
    create_mask(images[i])
```

해당 코드는 mask.py를 임포트하여 원본이미지 데이터셋 디렉터리 내에 있는 모든 이미지를 루프를 돌며 create_mask 매서드를 실행하는 코드이다. 이를 통해 원하는 데이터 셋을 생성 할 수 있는 것이다.

4. Labeling

loop_through_folder.py를 실행하여 다음 그림과 같은 데이터를 생성하였고 마스크 착용여부 및 방법을 식별하기 위한 목적에 맞도록 다음과 같이 분류하여 새로운 데이터 셋을 완성하였다. 새롭게 생성된 데이터셋을 포함하여 세 가지 분류로 Labling 하였다.

without_mask, with_mask. wrong_mask 각각 2483, 2925, 1217개의 이미지로 구성하였다.



<그림5. 새롭게 생성된 with_mask, wrong_mask dataset>

<그림6 합성 가능한 마스크 이미지(.png)파일>

위와 같이 여러 마스크 이미지를 가지고 있으면 다른 데이터를 생성가능하며 많은 경우의 수를 대비 할 수록 생성할 모델의 정확도와 실제 테스트에서 인식률을 상승시킬 수 있을 것이라 예상한다. 이번 프로젝트는 white_mask를 합성시킨 데이터 셋을 with_mask로 지정하였으며, wrong_mask일 경우 코 위로 개방되도록 하는 마스크 이미지와 턱 위로 개방되도록 합성하는 마스크 이미지를 선택하여 적절한 수로 섞어서 데이터셋을 구성하였다.

Ⅲ. 데이터 트레이닝

1. 전이학습(Transfer Learning)

[4]전이학습의 의미를 풀이하자면 학습데이터가 부족한 분야의 모델구축을 위해 기존에 존재하는 데이터가 풍부한 분야에서 훈련된 모델을 재사용 하는 기법이라고 할 수 있다. 이는 특정한 Task(Classification, Detection, Segmentation 등)에 대하여 학습된 딥러닝 모델을, 다른 Task로 Transfer(전이)하여 해당 모델을 사후적으로 학습하는 개념을 포괄한다. 이를 통해 특정 분야에서 학습된 신경망의 일부 능력을 유사하거나 전혀 새로운 분야에서 사용되는 신경망의 학습에 이용이 가능하다. 따라서 기존의 깊고 복잡한 신경망을 통해 방대한 데이터를 학습시킨 모델을 사용하여 새로운 모델을 만들시 복잡한 구조의 CNN을 처음부터 만들지 않으며 비교적 소량의 데이터를 이용하기 때문에 개발시간을 훨씬 단축할 수 있고 예측도를 높이기에도 수월하다는 장점이 있다. 일반적으로 VGG, ResNet, Mobilenet, gooGleNet등 사전에 학습이 완료된 모델(Pre-Trained Model)을 가지고 우리가 원하는 학습을 위해 미세 조정 즉, 작은변화를 주어 학습을 진행한다. 신경망의 이러한 재학습 과정을 세부 조정(fine-tuning)이라 부르고 이 과정에서 자신의 데이터를 일부 추가하여 보틀넥피처를 추출하거나, pre-trained model을 그대로 이용하고 affine layer(Fully connected layer)를 추가하거나 , 일부 layer를 목적에 맞도록 조정하여 트레이닝 할 수 있다.

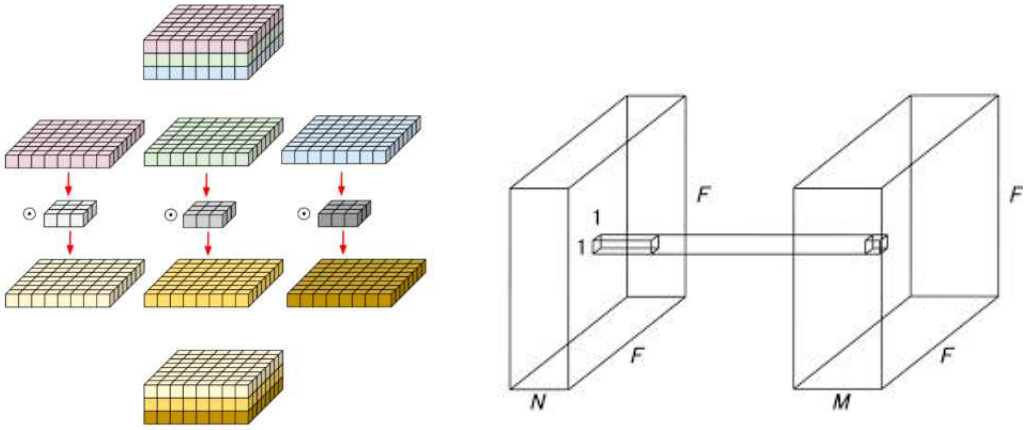
이번 프로젝트에서는 Mobilenest-v2 모델구조를 베이스 모델로 하고 마스크를 착용하거나, 잘못 착용하거나, 착용하지 않았는지를 판별하도록 해당 목적에 맞는 데이터 셋을 추가하여 Top 부분의 레이어를 수정 한 뒤(분류기(classfier)추가) 데이터셋의 특징을 추출하는 새로운 모델을 만들 것이다.

2. Mobilenet-v2

[5]Mobilenet 네트워크 모델은 구글에서 발표하였으며 모바일 디바이스에 특화하여 디자인 및 설계되었다. 모바일 기기에 특화되었다는 사실에서 짐작할 수 잇겠지만 이 신경망은 다른 환경보다 비교적 메모리자원이 한정되어있는 모바일 기기에서의 구동을 지원하기 위해 만들어 졌기 때문에 다른 cnn네트워크에 비해 계산비용을 획기적으로 줄이며 성능 또한 뒤떨어지지 않는 것이 특징이다. 프로젝트에서 이 네트워크를 통해 모델을 생성기로 한 이유는 다음과 같다.

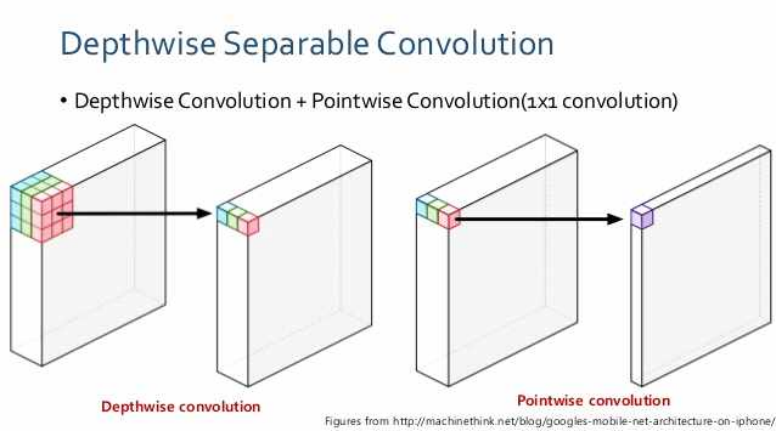
- 다양한 제품군의 마스크가 이미 존재하고 만들어지기 때문에 정확도 향상을 위한 수시적인 모델업데이트를 지원하기 위해 빠르게 학습 가능한 전이학습 네트워크 필요
- 해당 프로젝트의 목표는 스트리밍 서비스를 지원하는 것이기 때문에 영상을 빠르게 인식하고 판별하는 것이 중요함. 이상적으로는 실시간으로 처리될 정도의 뉴럴네트워크를 사용하는 것이 바람직하다. 따라서 Mobilenet-v2 네트워크를 통해 전이학습 시킨 모델을 이용함으로써 프로젝트의 목표를 부분적으로 실현하는데 도움이 됨.

[6],[7]Mobilenet-v2에서는 Channel-reduction, Depthwise separable convolution, remove fc layer 와 같은 방법을 통해 네트워크를 경량화 하며 특히 계산비용을 줄이기 위해 mobilenetv2 네트워크에서 사용하는 핵심 아이디어는 Depthwise separable convolution이다. 이 방법은 Depthwise convolution을 먼저 수행한 후 Pointwise convolution을 수행한다.



<그림7 Depthwise convolution(좌) 와 Pointwise convolution(우)>

Depthwise convolution의 기본적인 개념은 위 그림처럼 H*W*C의 conv output을 C단위로 분리하여 각각 conv filter을 적용하여 output을 만들고 그 결과를 다시 합치는 것이다 이를 통해 conv filter가 훨씬 적은 파라미터를 가지고서 동일한 크기의 아웃풋을 낼 수 있다.
Pointwise convolution는 흔히 1x1 Conv라고 불리는 필터이다. 주로 기존의 matrix의 결과를 논리적으로 다시 shuffle해서 뽑아내는 것을 목적으로 한다. 위 방법을 통해 총 channel수를 줄이거나 늘리는 목적으로 사용된다.



<그림 8 Depthwise separable convolution>

Depthwise convolution을 먼저 수행한 후 Pointwise convolution을 수행한다. 이를 통해서 3x3의 필터를 통해 conv 연산도 진행하고, 서로 다른 channel들의 정보도 공유하면서 동시에 파라미터 수도 줄일 수 있다. 이 과정을 통해서만 일반적인 Conv 대비 8-9배 연산량이 감소하게 된다.(kernel size = 3 기준)
Conv layer를 지나면 다층구조의 bottleneck layer를 통해 차원은 줄이되 manifold 상의 중요한 정보들은 최대한 그대로 유지하도록 한다.

3. 데이터 트레이닝 코드

(1) 코드분석(train_mask_detector)

```python
# USAGE
# python train_mask_detector.py --dataset dataset

# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
# 마스크 낀 이미지, 안낀 이미지 모아놓은 폴더의 경로
ap.add_argument("-d", "--dataset", required=True,
    help="path to input dataset")
#데이터 플롯 생성 디렉터리(파일이름 지정까지)
ap.add_argument("-p", "--plot", type=str, default="plot.png",
    help="path to output loss/accuracy plot")
#.model파일 저장경로(파일이름 지정까지)
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path to output face mask detector model")
args = vars(ap.parse_args())
```

필수적인 패키지들을 불어온다. 텐서플로우, 케라스의 Mobilenetv2 패키지를 이용하게 될 것임을 알 수 있고 텐서플로우 프레임워크 기반의 인공신경망을 통해 모델을 생성할 것이다. 또한 데이터 전처리 과정에서 사이킷런 패키지를 이용할 것이다. 그리고 해당 파일을 실행할 때 인자값을 받도록 argparse Library를 이용한다. 즉 (directory)/python -d (dataset dir) -m (save model dir) -p (save plot dir) 형태로 실행할 것이고 불러올 데이터셋 위치, 모델 저장위치,플롯 저장위치를 입력해준다.

```python
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []

# loop over the image paths
for imagePath in imagePaths:
    # extract the class label from the filename
    label = imagePath.split(os.path.sep)[-2]

    # load the input image (224x224) and preprocess it
    image = load_img(imagePath, target_size=(224, 224))
    image = img_to_array(image)
    image = preprocess_input(image)

    # update the data and labels lists, respectively
    data.append(image)
    labels.append(label)

# convert the data and labels to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
```

학습의 hyper parameter값(학습률, Epoch, 배치사이즈)을 지정해준다. 그리고 인자로 입력받은 데이터셋 디렉터리 하위의 이미지 파일의 경로를 리스트로 지정하고(imagePaths) 데이터와 레이블의 리스트를 초기화한다. 반복문을 통해 imagePaths 의 상위 디렉터리([-2])이름을 label 리스트에 저장하고 (with_mask, without_mask, wrong_mask) 이미지 파일을 전처리 하여(image to array) data 리스트에 저장한다. 그리고 다중 클래스의 소프트맥스 회귀분석을 위해 lable list의 값들을 one-hot-encoding 해줌으로써 모든 데이터 전처리를 완료하게 된다.

```
# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, stratify=labels, random_state=42)

# construct the training image generator for data augmentation
aug = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")

# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(3, activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
```

사이킷런 패키지의 train_test_split을 통해 train data/ test data(validation set)를 분리한다. train data를 100% 학습시키게 될 경우 overfitting 현상에 의해 성능이 잘 나오지 않기 때문에 데이터를 분리시켜 학습 중간에 test data셋으로 모델을 평가하는 것이다. 위 코드는 20% 데이터를 test data로 지정하고 데이터를 섞어주고(기본값이 True), labels 각각의 train/test 비율을 유지하도록 한다.

케라스의 ImageDataGenerator를 통해 해당 parameter에 부합하도록 데이터들을 augmentation 해준다. 이 과정은 데이터를 회전, 확대 좌우 이동 등을 거친 데이터를 추가해 줌으로써 데이터를 늘려주고 인식률을 개선시킬 수 있다.

전이학습을 위해 케라스의 Mobilenetv2 패키지를 이용한다. imagenet의 학습된 weight를 이용하고 top 부분에 마스크 착용 여부를 식별하는 분류기 layer를 쌓기 위해 include_top 값을 False로 지정한다. headModel(분류기)는 baseModel의 output을 input으로 받아 pooling layer, fully-connected layer, drop out layer를 통과하고 최종적으로 활성화 함수 softmax를 거치는 신경망으로 만들어 진다.

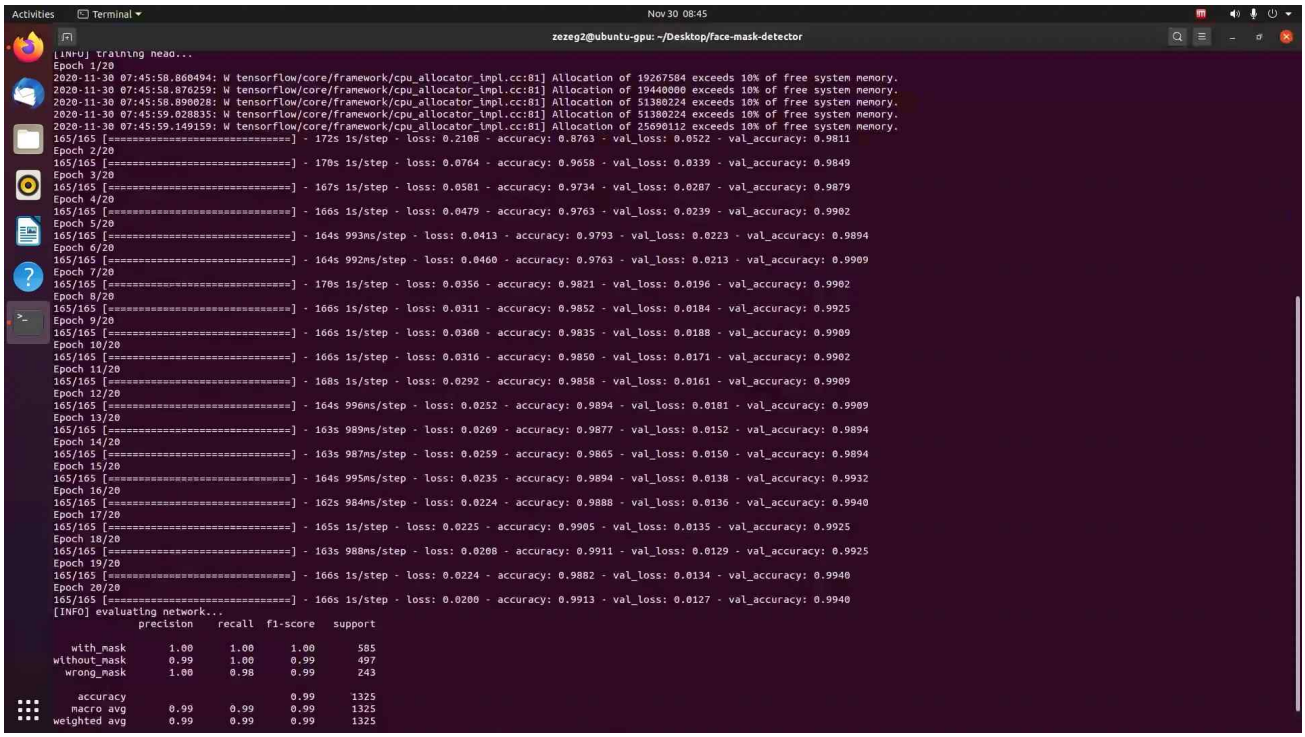
```python
for layer in baseModel.layers:
    layer.trainable = False

# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

# train the head of the network
print("[INFO] training head...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)

# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
    target_names=lb.classes_))

# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save(args["model"], save_format="h5")
```

baseModel의 layer들을 loop하며 layer.trainable = False로 지정 해줌으로 baseModel의 parameter값을 고정시켜준다. 즉 headModel의 layer만 트레이닝 하도록 하는 것이다.
optimizer 함수를 지정된 parameter 값을 가지도록 설정해주고 model.compile을 통해 model의 학습과정을 설정해준다. 이 때 데이터를 one-hot encoding 해주었고 softmax함수를 이용한 다중회귀분석임으로 cross-entropy loss를 이용한다.
최종적으로 model.fit 함수를 통해서 데이터를 학습시킨다. augmentation 데이터를 batch 마다 weights를 업데이트 하고 총 20회의 EPOCH 동안 트레이닝을 진행한다.
데이터 트레이닝이 끝나면 모델을 평가하고 모델을 생성한다.

```python
# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

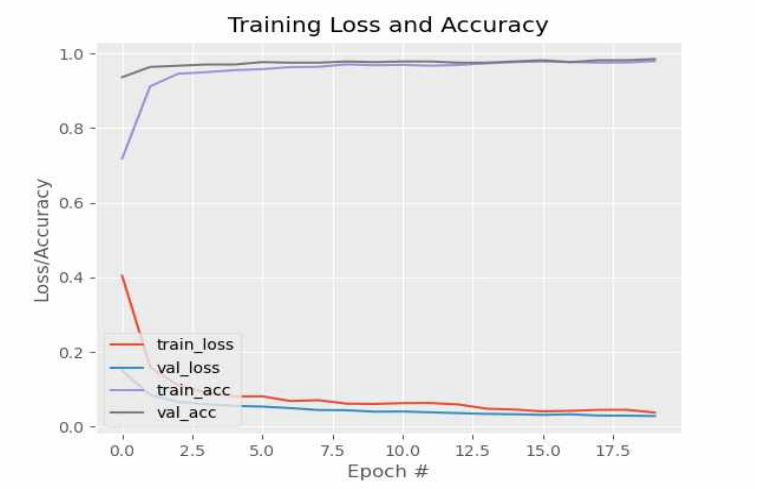
plot 데이터를 저장하기 위한 코드이다. 그래프에 표시하는 데이터는 각각 train loss/ validation(test) loss 그리고 train accuracy/ validation(test) accuracy이다.

(2) 트레이닝 결과 및 모델생성



<그림9 트레이닝 결과>

트레이닝을 마치고 트레이닝 데이터는 평균적으로 99% 이상의 정확도를 보여준다. 다만 wrong_mask class일 경우 두 가지 경우의 수를 대비하는 데이터셋을 준비하였기 때문에 정확도가 98%로 다소 떨어지는 경향을 보인다.



<그림10 생성된 plot 결과>

플롯을 분석해보면 데이터셋을 train data/ test data로 나눴기 때문에 학습 중간에 모델을 평가하며 트레이닝을 진행하였기 때문에 validation(test) accuracy가 train accuracy와 비슷하게 수렴함을 알 수 있다.

Ⅳ. 라즈베리파이를 이용한 스트리밍

1. 라즈베리파이 사용의 의미

라즈베리파이는 RAM과 CPU가 있고 무선네트워크(WI-FI) 모듈 및 랜포트를 통해 네트워크에 접속이 가능하며 기존의 컴퓨터와 같이 다양한 입출력장치 및 센서를 연결하는 포트가 있는 초소형 컴퓨터이다. 보통 리눅스 기반의 라즈비안 OS를 사용한다. 그렇게 때문에 원하는 목적에 따라 운영체제가 필요한 환경에 설치되어 각종 입출력 장치를 이용하여 기능을 수행하도록 조작할 수 있다.

해당 프로젝트에서는 실시간으로 영상을 전송하는 기능을 수행해야하기 때문에 라즈베리파이에 카메라 모듈을 연결해서 사용한다. 라즈베리파이에서 입력받은 영상정보는 python streaming Library MJPG를 이용하여 실시간 영상을 내부IP서버에 송출한다. 무선 및 유선네트워크를 통해 main computer가 이 스트리밍 서버에 접속하여 송출된 영상을 입력받아 얼굴인식 및 마스크 인식 알고리즘을 실행하게 된다. 해당 프로젝트의 이상적인 목표는 마스크 착용을 안내하는 기능뿐만 아니라 다수 기기의 스트리밍 정보를 main computer에서 관측하고 결과물을 저장할 수 있는 서비스를 만드는 것이다. 즉 여러 장소에 라즈베리파이를 설치하여 마스크 착용을 안내할 수 있을 뿐만 아니라 각 라즈베리 파이의 네트워크에 접속하여 중앙관리 및 제어, 관측이 가능한 형태의 서비스를 구축할 수 있는 가능성 및 방법을 열어놓는 것이다. 프로젝트에서는 방식을 다소 간소화하여 한 대의 라즈베리파이를 이용하고 main computer에서 마스크 착용여부를 판별하는 알고리즘을 구현하였다.

<그림 11 프로젝트의 이상적인 서비스형태>

2. MJPG 라이브러리

[8]MJPG streamer는 라즈베리파이의 카메라모듈을 이용하여 jpg촬영을 연속적으로 수행해서 영상처럼 보이도록 하는 솔루션이며 MJPG streamer를 설치함으로써 특정포트로 접근할 수 있는 HTTP server를 생성할 수 있다. 이 페이지를 통해 라즈베리파이 카메라의 실시간 촬영 영상에 접근할 수 있게 된다. 해당 패키지는 main computer가 아닌 라즈베리파이 기기에 컴파일 해서 실행해야 하며 기본적인 실행명령어는 mjpg_streamer -i "input_raspicam.so " -o "output_http.so -w ./www/"이고 여러 가지 실행옵션을 추가할 수 있다.

WebCam 옵션

-d	video device to open (your camera)
-r	the resolution of the video device, can be one of the following strings: QSIF QCIF CGA QVGA CIF VGA SVGA XGA SXGA or a custom value like: 640×480
-f	frames per second
-y	enable YUYV format and disable MJPEG mode
-q	JPEG compression quality in percent (activates YUYV format, disables MJPEG)
-m	drop frames smaller then this limit, useful if the webcam produces small-sized garbage frames may happen under low light conditions
-n	do not initalize dynctrls of Linux-UVC driver
-l	switch the LED "on", "off", let it "blink" or leave it up to the driver using the value "auto"

HTTP 출력 옵션

-w	folder that contains webpages in flat hierarchy (no subfolders)
-p	TCP port for this HTTP server
-c	ask for "username:password" on connect
-n	disable execution of commands

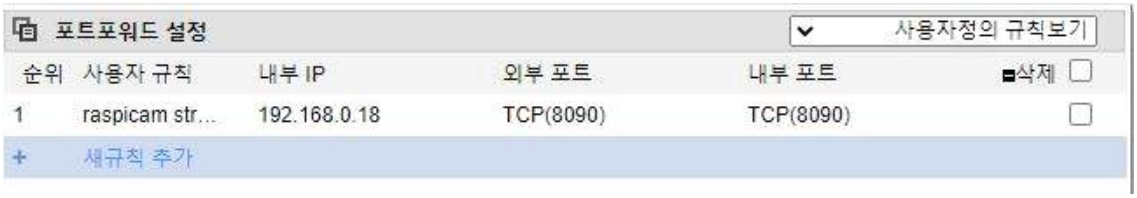
3. 카메라모듈 제원 및 세부설정

Product Name - Raspberry Pi Camera Module[OPEN-RBP-004]
Image Sensor - Sony IMX 219 PQ CMOS image sensor in a fixed-focus module.
Resolution 8-megapixel , Still picture resolution 3280 x 2464
Max image transfer rate 1080p: 30fps (encode and decode), 720p: 60fps
Connection to Raspberry Pi 15-pin ribbon cable, to the dedicated 15-pin MIPI CSI-2

프로젝트에서 스트리밍을 위해 사용한 코드는 아래와 같다.
mjpg_streamer -i "input_raspicam.so -x 1280 -y 720 -fps 10 -vf -hf " -o "output_http.so -p 8090 -w /usr/local/share/mjpg-streamer/www/"
즉 이 코드를 통해 input으로 raspicam 모듈을 사용하여 해상도 1280*720, 10fps인 영상을 상하, 좌우 반전하여 8090포트로 접근하는 스트리밍 서버를 생성하는 것이다.

4. 네트워크 접속

라즈베리파이에서 생성한 스트리밍 서버에 접속하기 위해서는 두가지 방법이 있다. 첫 번째로 네트워크를 공유하는 것이다. 즉 main computer가 라즈베리파이와 같은 인터넷 망, 내부망으로 구성되어있을 때 라즈베리파이의 내부IP와 접속포트(내부포트)8090을 입력하여(http://192.168.0.18:8090/?action=stream) 접속할 수 있다.
두 번째 방법으로는 DDNS 설정과 포트포워드를 통해서 외부망에서 라즈베리파이 네트워크에 접속할 수 있다. 이럴 경우 네트워크 관리자 권한으로써 설정을 필요로 한다.



<그림 12 포트 포워드 규칙설정>

위와 같이 설정하였을 경우 외부에서 http://DDNS.iptime.org:8090 입력하였을 때 DDNS주소를 통해 네트워크(공유기)에 접속하고 외부포트 8090를 입력해주면 내부IP의 8090 포트(http://192.168.0.18:8090/)에 접속할 수 있게 되는 것이다 즉 http://DDNS.iptime.org:8090 /?action = stream 으로 접속하여 라즈베리파이 스트리밍 서버로 접속하는 것이다.
main computer가 내부망에 포함되어있던, 외부망에서 접속을 하던 라즈베리파이가 생성한 서버의 주소를 opencv의 VideoCapture 의 인자값에 전달하면 스트리밍 영상을 입력받아 영상처리가 가능해진다. 이 원리를 이용해 궁극적으로 프로젝트의 목적을 실현하는 것이다.

Ⅴ. Mask detection

1. 사용 모델

해당 프로젝트에서는 두 가지의 모델을 사용한다. 첫 번째로는 영상에서 사람의 얼굴을 인식해서 찾아내는 모델 (res10_300x300_ssd_iter_140000), 그리고 두 번째는 찾아낸 얼굴에서 마스크 착용법을 분류하는 모델, 즉 **Ⅲ. 데이터 트레이닝**에서 트레이닝 시킨 모델(test.model)을 사용한다.

2. Mask detection 코드

```
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.models import load_model
import numpy as np
from pygame import mixer
import cv2
import datetime

mixer.init()
sound_nm = mixer.Sound('nomask_case.mp3')
sound_wm = mixer.Sound('wrmask_case.mp3')
prev, switch = 0, 0

now = datetime.datetime.now()
now = now.strftime('%Y-%m-%d-%H-%M')

facenet = cv2.dnn.readNet('models/deploy.prototxt', 'models/res10_300x300_ssd_iter_140000.caffemodel')
model = load_model('models'
                   '/test3.model')
```

케라스 패키지에서 mobilenet_v2 모델을 이용하기 위한 데이터 전처리를 해주는 preprocess_input, 그리고 모델을 로드하는 load_model을 임포트 한다. 그리고 opencv를 로드 한다.
그리고 opencv의 dnn모듈을 사용해서 face detection 모델(res10_300x300_ssd_iter_140000)을 로드하고 케라스 모듈을 이용해서 mask classification 모델(test.model)을 로드 한다.
마지막으로 라즈베리파이 스트리밍 서버 주소를 인자로 받아 VIdeoCapture 객체를 정의한다.

```
while cap.isOpened():
    ret, img = cap.read()
    if not ret:
        break

    h, w = img.shape[:2]

    blob = cv2.dnn.blobFromImage(img, scalefactor=1., size=(300, 300), mean=(104., 177., 123.))
    facenet.setInput(blob)
    dets = facenet.forward()

    result_img = img.copy()

    for i in range(dets.shape[2]):
        confidence = dets[0, 0, i, 2]
        if confidence < 0.5:
            continue

        x1 = int(dets[0, 0, i, 3] * w)
        y1 = int(dets[0, 0, i, 4] * h)
        x2 = int(dets[0, 0, i, 5] * w)
        y2 = int(dets[0, 0, i, 6] * h)

        face = img[y1:y2, x1:x2]

        try:
            face_input = cv2.resize(face, dsize=(224, 224))
            face_input = cv2.cvtColor(face_input, cv2.COLOR_BGR2RGB)
            face_input = preprocess_input(face_input)
            face_input = np.expand_dims(face_input, axis=0)
        except:
            pass

        mask, nomask, wrmask= model.predict(face_input).squeeze()
```

영상이 재생되고 프레임을 정상적으로 읽으면 루프가 지속된다. 이미지를 cap으로부터 불러오고 높이와 너비를 저장한다. dnn모듈이 사용 가능한 형태로 이미지를 변형하여 이 값을 facenet의 인풋으로 설정하고 결과를 추론하여 dets에 저장한다. 여러 개의 얼굴을 detect할 수 있기 때문에 하위에 for문 루프를 넣어 threshold(=0.5) 이하의 detection은 exception하고 나머지는 얼굴을 둘러싸는 bounding-box를 구해얼굴만 잘라내 face에 저장한다.
이후 face를 mobilenet-v2 모델이 사용가능한 형태로 전처리 해준다.(resize, BGR to RGB, preprocess_input, expand_dim)
케라스의 model.predict 메서드를 통해 각 레이블(mask, nomask, wrmask)의 predict값을 구해준다.

```
        if max(mask, nomask, wrmask) == mask:
            color = (0, 255, 0)
            label = 'Mask %d%%' % (mask * 100)
            switch = 0
        elif max(mask, nomask, wrmask) == nomask:
            color = (0, 0, 255)
            label = 'No Mask %d%%' % (nomask * 100)
            switch = 1
        else:
            if abs(wrmask - mask) < 0.2:
                color = (0, 255, 0)
                label = 'Mask %d%%' % (mask * 100)
            if abs(wrmask - nomask) < 0.2:
                color = (0, 0, 255)
                label = 'No Mask %d%%' % (nomask * 100)
                switch = 1
            else:
                color = (255, 0, 0)
                label = 'Wrong Mask %d%%' % (wrmask * 100)
                switch = 2
        if not mixer.get_busy():
            if prev < switch:
                if prev + 1 == switch:
                    sound_nm.play()
                if prev + 2 == switch:
                    sound_wm.play()
        prev = switch

        cv2.rectangle(result_img, pt1=(x1, y1), pt2=(x2, y2), thickness=2, color=color, lineType=cv2.LINE_AA)
        cv2.putText(result_img, text=label, org=(x1, y1 - 10), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8,
                    color=color, thickness=2, lineType=cv2.LINE_AA)

    out.write(result_img)
    cv2.imshow('result', result_img)
    if cv2.waitKey(40) == ord('q'):
        break
```

각 레이블의 predict값을 조건문과 .max 함수를 이용해서 predict값이 가장 큰 레이블의 값을 영상에서 bounding-box와 함께 출력한다. 이 때 wrong_mask 이미지가 마스크를 착용하거나 착용하지 않은 이미지 양쪽의 특성을 모두 공유하기 때문에 잘못 인식하는 경우가 있어 wrmask 값이 가장 크더라도 그 값이 나머지 레이블의 predict값과 어느 정도 차이(0.2 = 20%) 이하라면 wrong_mask를 출력하지 않도록 알고리즘을 설계하였다. 간략한 마스크 착용 음성안내 알고리즘을 추가하고 영상이 끝나거나 'q' key를 누르면 종료하도록 한다.

3. 테스트

(1) 영상파일에서의 테스트결과

<그림13 영상파일 detection 테스트 결과>

(2) 스트리밍 테스트 결과

<그림14 스트리밍 detection 테스트 결과>

영상파일을 테스트 한 결과는 위 그림13과 같다. 세가지 레이블의 predict 값에 따라 얼굴영역에 bouding box 처리하여 detection 결과를 나타낸다. 다만 얼굴의 다양한 각도에 따른 데이터의 부족 혹은 다른 이류로 인해서 얼굴 측면의 모습에서는 얼굴을 인식하지 못하는 결과를 보여주었다. 오른쪽 그림에서는 코 밑에까지 마스크를 걸친 모습을 wrong_mask로 인식하고 코까지 완전하게 덮었을 경우 올바르게 마스크를 착용했다고 인식하는 것을 볼 수 있다. 스트리밍 서버에 접속하여 테스트 한 결과는 그림 14와 같다. 마찬가지로 분류한 세 가지 predict 모두 인식할 수 있으며, 세 번째 그림에서와 같이 마스크를 정확하게 착용하지 않은 경우 두 가지 데이터셋으로 트레이닝 시킨 결과로 턱에 걸치거나, 코 밑에 걸친 경우 모두 잘못된 착용으로 올바르게 인식하고 있다.

Ⅵ. 보완할점

1. 마스크 착용 인식률 및 정확도 개선

이 프로젝트에서 가장 우려하던 문제점이 마스크를 올바르게 착용하지 않았을 때 인식률이 저조할 수 있다는 문제였다. 위에도 언급한바가 있듯이 마스크를 턱이나 코밑에 걸쳐서 착용했을 경우 마스크를 착용하지 않은 맨얼굴, 그리고 얼굴에 마스크를 정확하게 착용한 얼굴의 특징 양쪽 모두 어느 정도 공유하기 때문이다. 필터를 통해 특징의 유무를 찾아내는 deep neural network의 특성상 중간 값이라는 특성을 정확하게 잡아내기란 한계가 있을 것이다. 그리고 테스트 결과도 정확도의 신뢰성에 대한 문제가 없지 않았다. 얼굴인식 및 특징을 인식하기 쉽도록 가까운 정면의 얼굴에서 테스트 할 경우 우수한 성능을 보여주었지만, 멀리 있거나, 다른 각도의 얼굴일 경우 인식을 하지 못하거나 얼굴을 인식하더라도 착용법을 잘못 분류하는 결과를 보여주었다.

이러한 문제를 해결하기 위해서는 다각도의 원본 얼굴 데이터셋을 준비하는 것이다. 그러고 나서 마스크를 씌우는 알고리즘을 개선하여(더 다양한 특징점을 이용) 더 정확한 얼굴 위치에 마스크를 합성시킬 수 있도록 함으로써 더 실제와 같은 데이터를 생성하는 것이다.

2. 네트워크 보안문제

이 프로젝트는 IP camera방식으로 영상을 이용하기 때문에 라즈베리파이 기기의 내부IP를 알아낼 수 있으면 실시간 영상에 접속할 수 있다. 즉 방화벽, 포트포워드 규칙을 해킹하는 등의 방법으로 내부네트워크에 접속하기만 한다면 실시간 영상이 유출된다는 취약점을 가지고 있다. 최소한의 수단으로 SSID를 감추고 승인된 기기에서만 접근을 허용하도록 하여 어느 정도의 네트워크 보안문제를 해결할 수 있다.

Ⅶ 결 론

이 프로젝트를 통해 실현하고자 한 목적은 공공시설에서 마스크 착용안내뿐만 아니라 시설 내 기기의 확장이 용이하도록 설계를 하는 것이다. 이러한 점에서 라즈베리파이와 카메라모듈을 이용하여 IP카메라로써 사용하고 이를 main computer에서 관리하는 구조를 이용함으로써 실현하고자 하는 목적에 부합하도록 설계했다고 평가할 수 있다.

얼굴인식 및 마스크 착용법 식별을 위해서는 기존에 부족한 데이터를 보충할 필요가 있었다. 이 과정에서 크롤링을 통한 데이터 수집에 대해서도 고민했지만, 규칙성이 떨어지는 등 여러 가지 방면에서 데이터의 질이 떨어져서 학습을 위한 데이터로써는 적합하지 않다고 판단하게 되어 데이터를 생성하는 방향으로 프로젝트를 진행하였다. 파이썬 얼굴인식 라이브러리를 이용해 마스크 이미지를 원본 이미지에 합성하는 방식으로 약 5000여장의 새로운 이미지 데이터를 획득하였고 이렇게 생성된 데이터를 구분하고자 하는 기준에 따라 레이블별로 나누고 트레이닝 하였다. 각 레이블별로 train data, test data모두 평균적으로 99% 이상의 정확도를 보였으므로 양질의 데이터를 획득하여 모델을 생성하는 과정은 성공적이라고 평가할 수 있다.

그러나 이 모델을 이용해서 스트리밍 영상정보를 받아 실제상황에서 테스트 하였을 때는 트레이닝 과정에서 수치로 나타났던 99%의 정확도에 못 미치는 결과를 보여줬다. 그 이유는 실제 상황에서는 연속적인 값으로 존재하는 데이터를 트레이닝을 위해 세 가지로만 분류해야하는 필연적인 과정에서 중간 값이라는 특성에서 인식률이 떨어지는 것이라고 생각한다. 또한 데이터셋 자체에서 정면인 얼굴을 제외하고 다른 각도에서의 데이터가 부족한 것도 하나의 원인이라고 생각한다.

프로젝트의 전체 과정에서 다소 미흡하거나 간과했던 부분이 있을 수도 있겠지만, 공공시설에서 마스크 착용을 안내해줌과 동시에 main computer관리자를 통해 한 번 더 전체관리가 가능한 시스템의 패러다임을 제시한 것에 의의가 있음을 알리는 바이다.

참고 문헌

[1]Christian J. Kähler외 1인, 2020, Fundamental protective mechanisms of face masks against droplet infections

[2]Yixuan J Hou 외 18인, 2020, SARS-CoV-2 Reverse Genetics Reveals a Variable Infection Gradient in the Respiratory Tract

[3] Google , "face_recognition hog“, Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning(hog 관련 자료)
https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78

[4] Google , "transfer learning and fine tuning", Transfer Learning(전이 학습)
https://choice-life.tistory.com/m/40

[5] Mark Sandler외 4명, 2018, MobileNetV2: Inverted Residuals and Linear Bottlenecks

[6]Google , “movilenetv2 논문해석”, MobileNets - Efficient Convolutional Neural Networks for Mobile Vision Applications
https://gaussian37.github.io/dl-concept-mobilenet/

[7]Google , "Depthwise Separable Convolution", Depthwise Separable Convolution 설명 및 pytorch 구현
https://wingnim.tistory.com/104

[8]Google , "mjpg 패키지설치 및 사용법" [라즈베리파이 중급](13) 웹캠 스트림 서버
https://echo.tistory.com/264