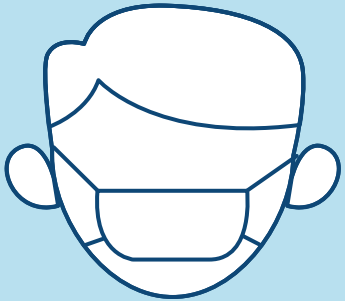
마스크 착용 여부 판별기

발표일 : 2022.1.7

2ternals

팀장 : 손예린
팀원 : 권희정, 김창현, 이우섭, 이해성, 최정수

TABLE OF CONTENTS



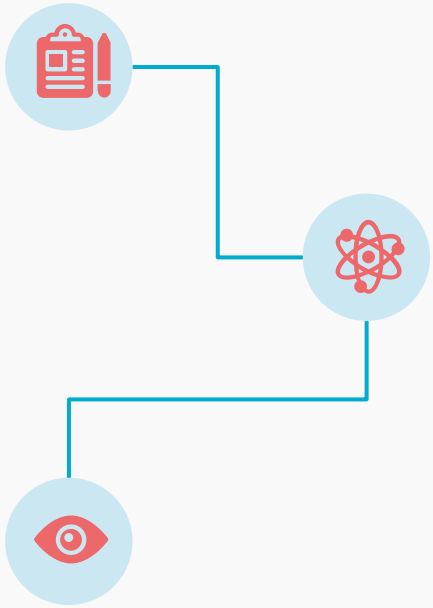
INTRODUCTION

무인상점 및 출입이 자유로운
공공시설 이용자의 얼굴인식을
통해 마스크 착용 여부를
판별하는 프로그램

PROCESS...

Data

마스크 착용 데이터
미착용 얼굴 데이터



AI Model

마스크 착용 여부 판별
CNN 이미지 분류 모델
설계 및 훈련

Results

영상 내 얼굴 영역 검출 후
마스크 착용 여부 판별 및 알림

주요 라이브러리

02
OpenCV
웹캠을 이용한 영상 촬영
얼굴 박스, 마스크 착용확률 표시

01
Tensorflow
딥러닝 모델 구축

03
Cvlib
얼굴 인식

04
Numpy
이미지 연산

06
Matplotlib
수치 도식화

05
Pygame
알림음

Roles and Responsibilities

이미지 수집	권희정, 김창현, 최정수
모델 설계	손예린, 이우섭, 이해성
하이퍼 파라미터 튜닝	손예린, 김창현
발표	권희정

Development Schedule

1/3

주제선정

마스크 착용, 미착용 분류기

1/4

데이터 수집

마스크 착용, 미착용 이미지 수집

1/5

개발

이미지 분류기 생성,
딥러닝 모델 구축

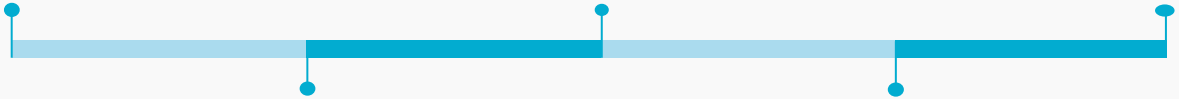
1/6

하이퍼 파라미터 튜닝

활성함수, 손실함수, 은닉층 개수 조절 및
정확도 측정

1/7

발표



데이터 전처리

```
path_dir1 = './nomask/'
path_dir2 = './mask/'

file_list1 = os.listdir(path_dir1) # path에 존재하는 파일 목록 가져오기
file_list2 = os.listdir(path_dir2)

file_list1_num = len(file_list1)
file_list2_num = len(file_list2)

file_num = file_list1_num + file_list2_num
```

```
num = 0;
all_img = np.float32(np.zeros((file_num, 224, 224, 3))) #224 x 244 x (BGR)
all_label = np.float64(np.zeros((file_num, 1)))
```

```
for img_name in file_list1: #nomask
    img_path = path_dir1+img_name
    img = load_img(img_path, target_size=(224, 224))

    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    all_img[num, :, :, :] = x

    all_label[num] = 0 # nomask
    num = num + 1
```

Nomask, 0

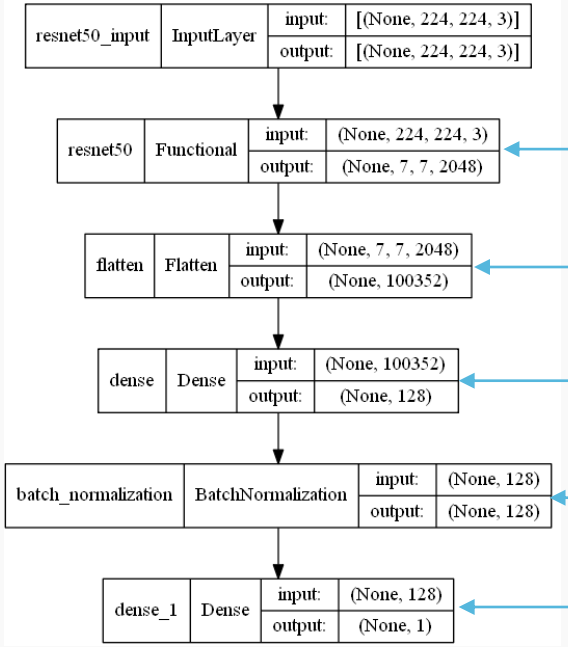
```
for img_name in file_list2:#mask
    img_path = path_dir2+img_name
    img = load_img(img_path, target_size=(224, 224))

    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    all_img[num, :, :, :] = x

    all_label[num] = 1 # mask
    num = num + 1
```

mask, 1

```
num_train = int(np.round(all_label.shape[0]*0.8))
num_test = int(np.round(all_label.shape[0]*0.2))

train_img = all_img[0:num_train, :, :, :]
test_img = all_img[num_train:, :, :, :]

train_label = all_label[0:num_train]
test_label = all_label[num_train:]
```

train, test 분할
8:2

Model ...




```
base_model = ResNet50(input_shape=IMG_SHAPE, weights='imagenet', include_top=False)
base_model.trainable = False
```

Transfer learning

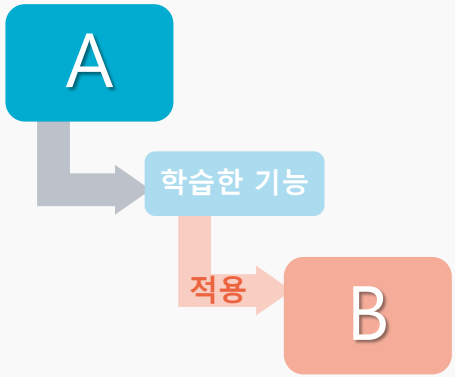
```
flatten_layer = Flatten()
dense_layer1 = Dense(128, activation='relu')
bn_layer1 = BatchNormalization()
dense_layer2 = Dense(1, activation=tf.nn.sigmoid)
```

```
model = Sequential([
        base_model,
        flatten_layer,
        dense_layer1,
        bn_layer1,
        dense_layer2,
        ])
```

```
base_learning_rate = 0.001
model.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Transfer learning

한 가지 문제에 대해 학습한 기능을 가져와 비슷한 새로운 문제에 활용하는 것

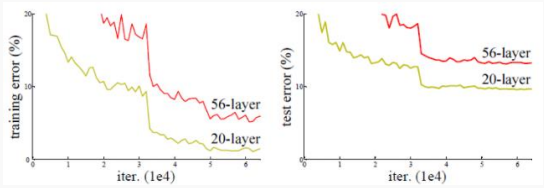


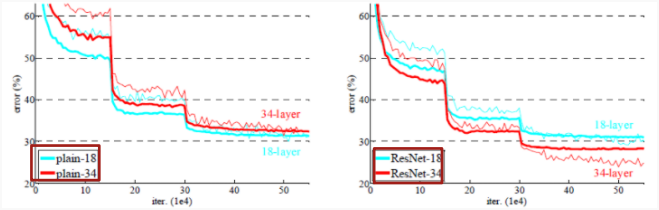
장점

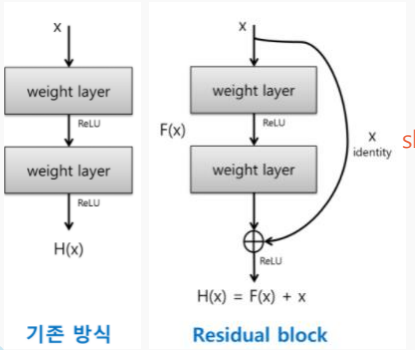
- 적은 데이터에 효과적
- 빠른 학습속도
- 높은 정확도

마스크 착용 이미지가 부족하여 학습이 충분히 이루어지지 않는 상황 방지 및 정확도 향상을 위해 사용

ResNet50





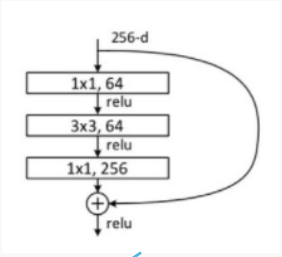


shortcut

- Residual Networks
- 기울기 소실 해결(shortcut, 입력값이 출력에 더해짐으로써)
- 깊은망 ≠ 좋은성능
- 깊은 layer모델, ResNet 활용시 에러 감소

ResNet50

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 [3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



ResidualBlock(in_channels, middle_channels, out_channels)	ResNet50 상에서		코드 상에서
적용 순서	Input 채널 개수	Output 채널 개수	
1, conv(1x1, middle_channels)	in_channels	middle_channels	conv1x1(in_channels, middle_channels)
2, conv(3x3, middle_channels)	middle_channels	middle_channels	conv3x3(middle_channels, middle_channels)
3, conv (1x1, out_channels)	middle_channels	out_channels	conv1x1(middle_channels, out_channels)

RESULTS

Jupyter Notebook (CPU)

```
Epoch 1/10
178/178 [==============================] - 240s 1s/step - loss: 0.1440 - accuracy: 0.9527 - val_loss: 0.0402 - val_accuracy: 0.9944
Epoch 2/10
178/178 [==============================] - 286s 2s/step - loss: 0.0240 - accuracy: 0.9972 - val_loss: 0.0214 - val_accuracy: 0.9929
Epoch 3/10
178/178 [==============================] - 291s 2s/step - loss: 0.0208 - accuracy: 0.9961 - val_loss: 0.0192 - val_accuracy: 0.9944
Epoch 4/10
178/178 [==============================] - 265s 1s/step - loss: 0.0139 - accuracy: 0.9982 - val_loss: 0.0405 - val_accuracy: 0.9901
Epoch 5/10
178/178 [==============================] - 271s 2s/step - loss: 0.0122 - accuracy: 0.9982 - val_loss: 0.0178 - val_accuracy: 0.9944
Epoch 6/10
178/178 [==============================] - 275s 2s/step - loss: 0.0103 - accuracy: 0.9979 - val_loss: 0.0147 - val_accuracy: 0.9944
Epoch 7/10
178/178 [==============================] - 275s 2s/step - loss: 0.0064 - accuracy: 0.9982 - val_loss: 0.0212 - val_accuracy: 0.9944
Epoch 8/10
178/178 [==============================] - 309s 2s/step - loss: 0.0098 - accuracy: 0.9996 - val_loss: 0.0178 - val_accuracy: 0.9944
Epoch 9/10
178/178 [==============================] - 294s 2s/step - loss: 0.0054 - accuracy: 0.9989 - val_loss: 0.0203 - val_accuracy: 0.9944
Epoch 10/10
178/178 [==============================] - 296s 2s/step - loss: 0.0124 - accuracy: 0.9968 - val_loss: 0.0170 - val_accuracy: 0.9958
학습시간: 0:46:42.463345
Saved model to disk
```

46min

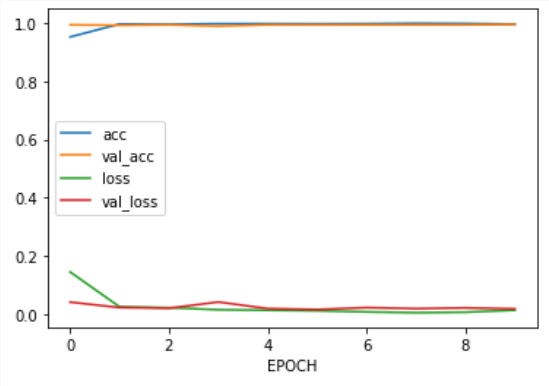
Colab (GPU)

```
Epoch 1/10
178/178 [==============================] - 42s 169ms/step - loss: 0.1259 - accuracy: 0.9658 - val_loss: 0.0297 - val_accuracy: 0.9929
Epoch 2/10
178/178 [==============================] - 28s 159ms/step - loss: 0.0320 - accuracy: 0.9926 - val_loss: 0.0330 - val_accuracy: 0.9944
Epoch 3/10
178/178 [==============================] - 28s 160ms/step - loss: 0.0231 - accuracy: 0.9954 - val_loss: 0.0422 - val_accuracy: 0.9944
Epoch 4/10
178/178 [==============================] - 28s 159ms/step - loss: 0.0127 - accuracy: 0.9965 - val_loss: 0.0259 - val_accuracy: 0.9915
Epoch 5/10
178/178 [==============================] - 28s 159ms/step - loss: 0.0090 - accuracy: 0.9986 - val_loss: 0.0240 - val_accuracy: 0.9929
Epoch 6/10
178/178 [==============================] - 33s 187ms/step - loss: 0.0240 - accuracy: 0.9947 - val_loss: 0.0134 - val_accuracy: 0.9986
Epoch 7/10
178/178 [==============================] - 28s 159ms/step - loss: 0.0145 - accuracy: 0.9958 - val_loss: 0.0116 - val_accuracy: 0.9958
Epoch 8/10
178/178 [==============================] - 28s 159ms/step - loss: 0.0076 - accuracy: 0.9982 - val_loss: 0.0126 - val_accuracy: 0.9958
Epoch 9/10
178/178 [==============================] - 28s 158ms/step - loss: 0.0106 - accuracy: 0.9986 - val_loss: 0.0094 - val_accuracy: 0.9958
Epoch 10/10
178/178 [==============================] - 28s 159ms/step - loss: 0.0030 - accuracy: 0.9996 - val_loss: 0.0146 - val_accuracy: 0.9944
학습시간: 0:05:26.866280
Saved model to disk
```

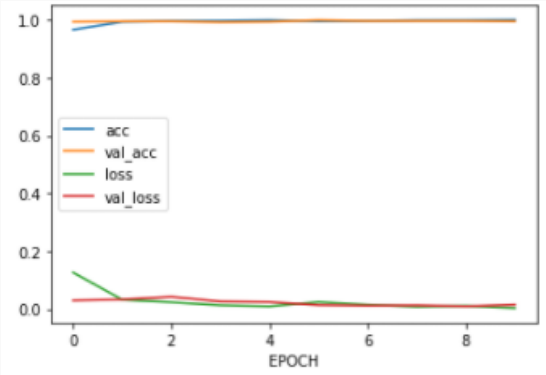
5min

RESULTS

Jupyter Notebook (CPU)

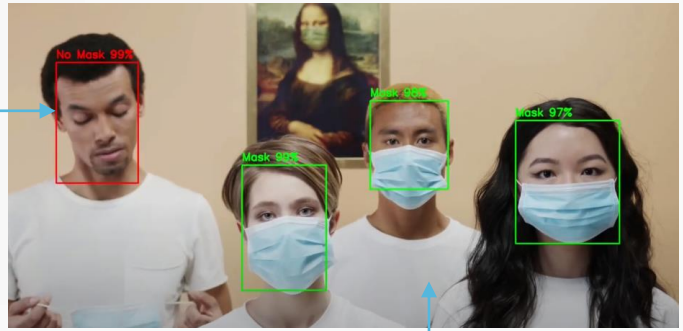


Colab (GPU)



Face Detection

```
for idx, f in enumerate(face):

    (startX, startY) = f[0], f[1]
    (endX, endY) = f[2], f[3]

    if 0 <= startX <= frame.shape[1] and 0 <= endX <= frame.shape[1] and 0 <= startY <= frame.shape[0] and 0 <= endY <= frame.shape[0]:

        face_region = frame[startY:endY, startX:endX]
        face_region1 = cv2.resize(face_region, (224, 224), interpolation = cv2.INTER_AREA)

        x = img_to_array(face_region1)
        x = np.expand_dims(x, axis=0)
        x = preprocess_input(x)

        prediction = model.predict(x)

        if prediction < 0.5: # 마스크 미착용으로 판별되면,
            cv2.rectangle(frame, (startX,startY), (endX,endY), (0,0,255), 2)
            Y = startY - 10 if startY - 10 > 10 else startY + 10
            text = "No Mask ({:.2f}%)".format((1 - prediction[0][0])*100)
            cv2.putText(frame, text, (startX,Y), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)

        else: # 마스크 착용으로 판별되면
            cv2.rectangle(frame, (startX,startY), (endX,endY), (0,255,0), 2)
            Y = startY - 10 if startY - 10 > 10 else startY + 10
            text = "Mask ({:.2f}%)".format(prediction[0][0]*100)
            cv2.putText(frame, text, (startX,Y), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0), 2)
```



Thread

```
if(condition == 1):
    t = threading.Thread(target=maskonplay, args=(5, soundplay))
    t.start()

def soundplay():
    mixer.init()
    sound = mixer.Sound('maskon.wav')
    if not webcam.isOpened():
        mixer.stop()
    if webcam.isOpened():
        sound.play()
```

solution

Error

Webcam 종료 후에도 알림음 지속

Val_count

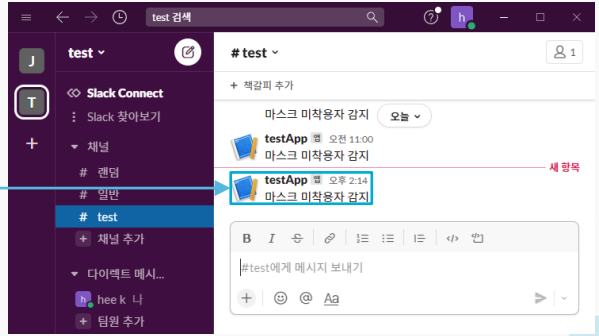
```
mixer.init()
sound = mixer.Sound('maskon.wav')

alarm += 1
count += 1
if alarm == 15:  # 알림을 재생
    sound.play()
    alarm = 0
if count == 50:  # 메시지 전송
    send_message('마스크 미착용자 감지')
    count = 0
```

Frame기준

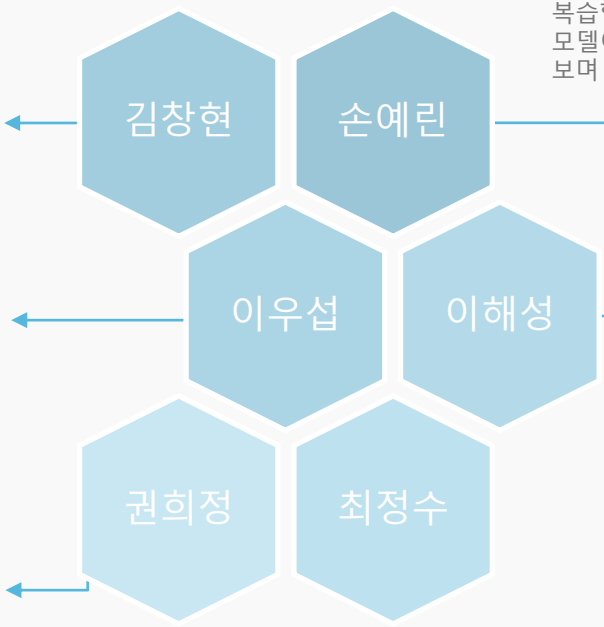
Send Message(feat. Slack) ...

```python
def send_message(msg):
    url='https://hooks.slack.com/services/WebhookURLs'
    data = {'text':msg}
    resp = requests.post(url=url, json=data)
    return resp
```

```python
alarm += 1
count += 1
if alarm == 15:  # 알림을 재생
    sound.play()
    alarm = 0
if count == 50:  # 메시지 전송
    send_message('마스크 미착용자 감지')
    count = 0
```



Review



구글링을 통해 많은 것을 배우고, 모델링 뿐만 아니라 여러가지 기능을 추가하는 방법도 배우는 시간이었다.

수업시간에 배운 딥러닝 이미지 분류 전체 과정을 복습할 수 있는 좋은 기회였다.
모델이 한번의 학습만으로도 뛰어난 성능을 내는 것을 보며 인공지능의 유용성을 실감했다.

딥러닝 프로젝트를 처음 해보는 기간이었는데 수업 시간에 배운 것 말고도 알아야 할 게 많다는 것을 알았고 기억에 남을 수 있도록 많이 만들어 봐야 한다고 생각이 들었다.

딥러닝 이미지 학습에 관하여 복습할 수 있는 좋은 프로젝트였다.
백그라운드에서 함수를 실행할 수 있는 thread에 대해 찾아 공부하는 의미있는 시간이었다.
하지만 thread로 실행한 함수를 정지 및 재실행 등 핸들링을 하려 했지만 구현을 하지 못한 아쉬움이 남는다.

딥러닝 이미지 분류에 대해 복습하며 새로운 것들을 많이 알게 되는 좋은 기회였다.

Thank You