개인 프로젝트 최종 발표

마스크 데이터셋을 활용한 이미지 분류

김태우

CONTENTS

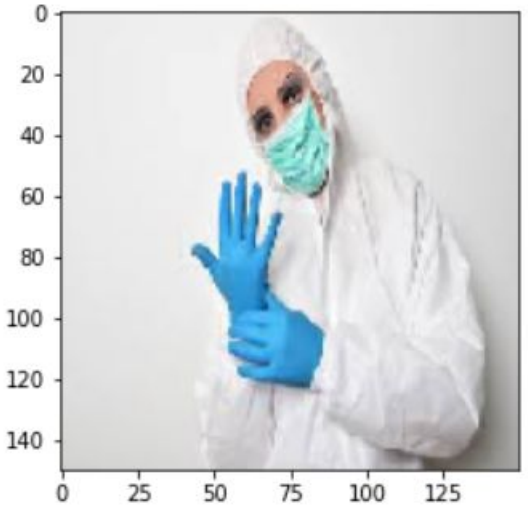
01 구현 여부

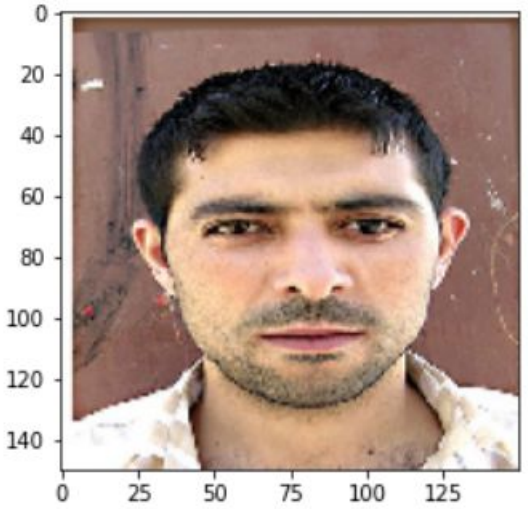
- 모델 구현

1 (마스크 미착용)

입력 : 마스크 착용한 얼굴 이미지와 착용하지 않은 얼굴 jpg 이미지 파일

출력 :이진분류 (0, 1)






```
[[0.00749988]]
[[0.99992466]]
[[0]]
[[1]]
```

02 성능 평가

- 기존 모델과 성능 향상 비교

기존 모델 측정 값

```
[[0.10941345]]
[[0.44961792]]
```

최종 모델 측정 값

```
[[0.4041057]]
[[0.79764885]]
```

특징 : 0.34 차이 -> 약 0.4 차이로 0.06 상승

```
from keras.preprocessing.image import image
import numpy as np

#아시아인 마스크
img_path = '/content/drive/MyDrive/코랩 딥러닝/딥러닝 데이터/Mask/Test1.jpg'
img_path2 = '/content/drive/MyDrive/코랩 딥러닝/딥러닝 데이터/Mask/Test2.jpg'

# img_path = '/content/drive/MyDrive/코랩 딥러닝/딥러닝 데이터/Mask/data_set_cla
# img_path2 = '/content/drive/MyDrive/코랩 딥러닝/딥러닝 데이터/Mask/data_set_cl


img = image.load_img(img_path,target_size =(150,150))

img2 = image.load_img(img_path2,target_size =(150,150))
x = image.img_to_array(img)
x2 = image.img_to_array(img2)
x /= 255.
x2 /= 255.

x = np.expand_dims(x ,axis = 0)
x2 = np.expand_dims(x2 ,axis = 0)
np.shape(x)

plt.imshow(x[0])
plt.show()
plt.imshow(x2[0])
plt.show()

ypred = model2.predict(x)
print(ypred)
ypred = model2.predict(x2)
print(ypred)

ypred = model2.predict_classes(x)
print(ypred)
ypred = model2.predict_classes(x2)
print(ypred)
```

02 성능 평가

- 약 310개의 테스트 이미지에 대한 정확도 비교

기존 모델 :0.6903

```
11/11 [==============================] - 1s 68ms/step - loss: 0.8411 - acc: 0.6903 - mae: 0.3429
테스트 데이터 점수 :  [0.8410696983337402, 0.6903225779533386, 0.342909574508667]
```

```
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=30,
        class_mode='binary')
```

최종 모델 :0.8258

```
print("테스트 데이터 점수 : ",model.evaluate(test_generator,batch_size=1000))
```

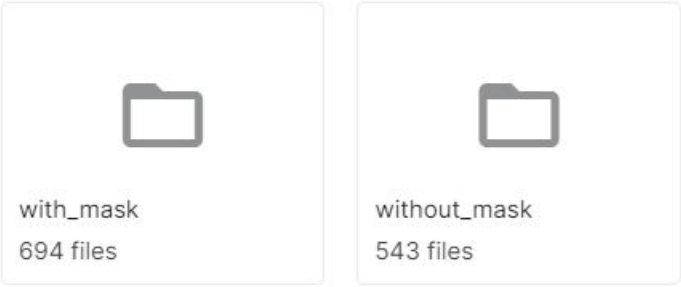
```
11/11 [==============================] - 1s 76ms/step - loss: 0.8527 - precision: 0.8611 - recall: 0.7850
VGG 16 테스트 데이터 점수 :  [0.8526999950408936, 0.8610698580741882, 0.7850282788276672, 95.5454559326171
```

03 성능 향상 방법

- 기존 대비 개선된 점

모델 교체 → 이미 학습된 모델 VGG 16모델 사용

부족한 이미지 데이터 수 → 이미지 제너레이터 조절



모델 과대적합 → 유닛 수, 에포크, 층의 수
하이퍼파라미터 조절,손실
그래프 참고

03 성능 향상 방법

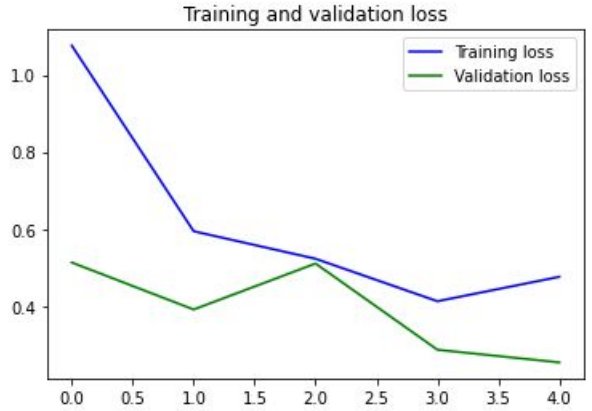
모델 교체

```
from keras import models
from keras import layers

model2 = models.Sequential()
model2.add(conv_base)
model2.add(layers.Flatten())
model2.add(layers.Dense(128, activation='relu'))
model2.add(layers.Dense(1, activation='sigmoid'))

input_shape = (None, 150, 150, 3)
model2.build(input_shape)

model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               1048704
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 15,763,521
Trainable params: 15,763,521
Non-trainable params: 0
_________________________________________________________________
```

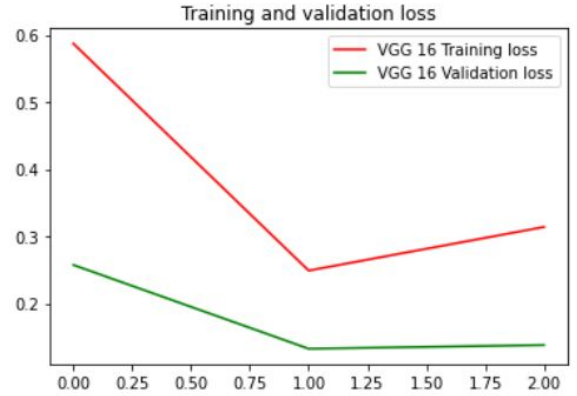
이미 학습된 VGG16 모델 활용

기존 모델 정확도, 손실




```
11/11 [==============================] - 1s 81ms/step - loss: 0.6267 - acc: 0.6935 - mae: 0.3557
테스트 데이터 점수 :  [0.626672625541687, 0.6935483813285828, 0.3556556105613708]
```

VGG16 활용 모델 정확도, 손실

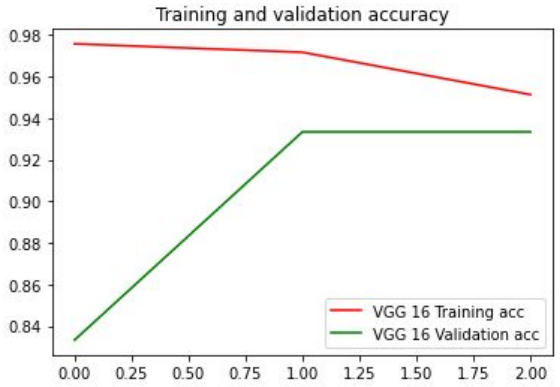


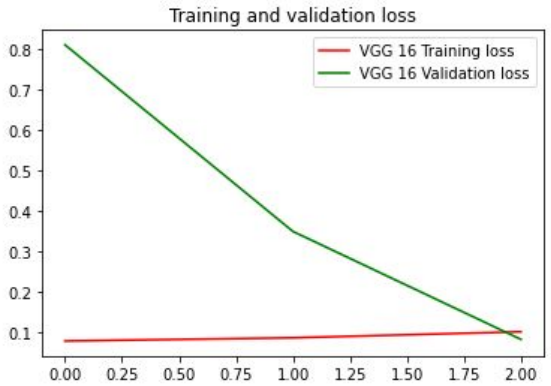

```
11/11 [==============================] - 1s 76ms/step - loss: 0.8527 - precision: 0.8611 - recall: 0.7850
VGG 16 테스트 데이터 점수 :  [0.8526999950408936, 0.8610698580741882, 0.7850282788276672, 95.5454559326171
```

03 성능 향상 방법

최적의 이미지 제너레이터 값

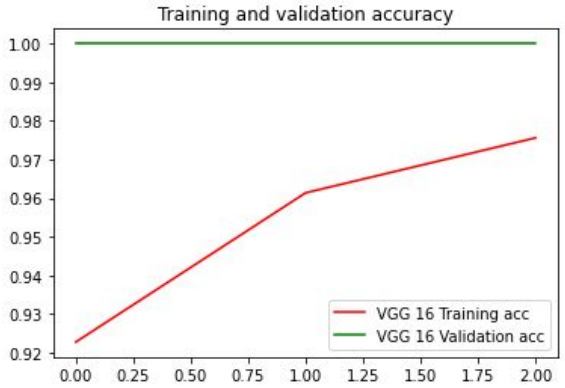
배치 사이즈 30

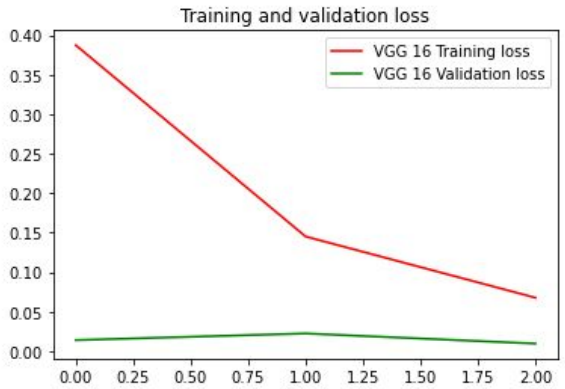




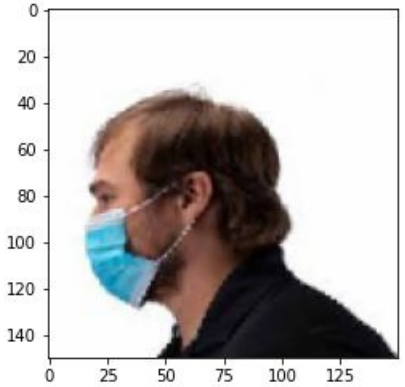

```
11/11 [==============================] - 1s 77ms/step - loss: 1.3616 - acc: 0.7355
VGG 16 테스트 데이터 점수 :  [1.3616122007369995, 0.7354838848114014, 0.26684415340
```

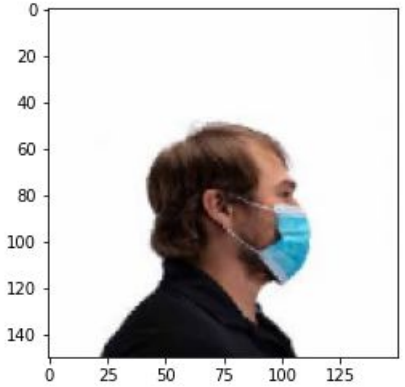
배치 사이즈 50

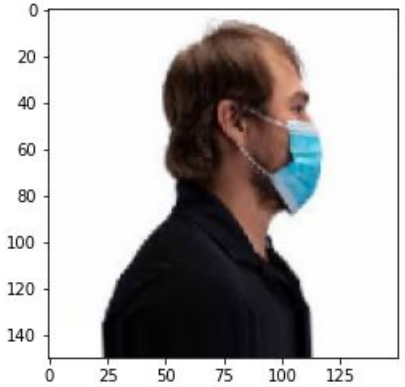





```
11/11 [==============================] - 1s 79ms/step - loss: 0.8828 - acc: 0.8258
VGG 16 테스트 데이터 점수 :  [0.8827643394470215, 0.8258064389228821, 0.17912672460
```



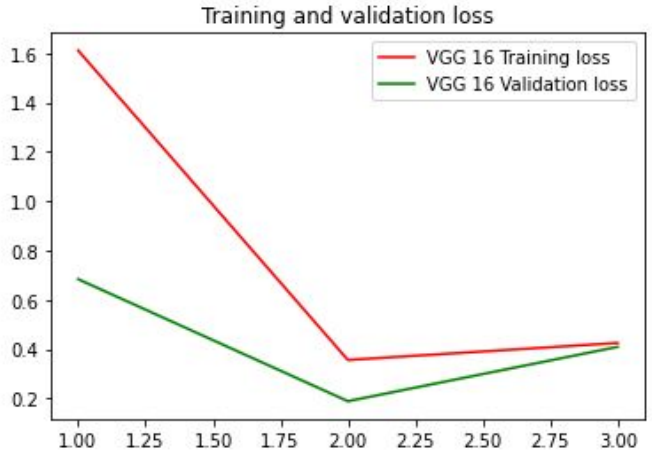




03 성능 향상 방법

하이퍼파라미터 조절

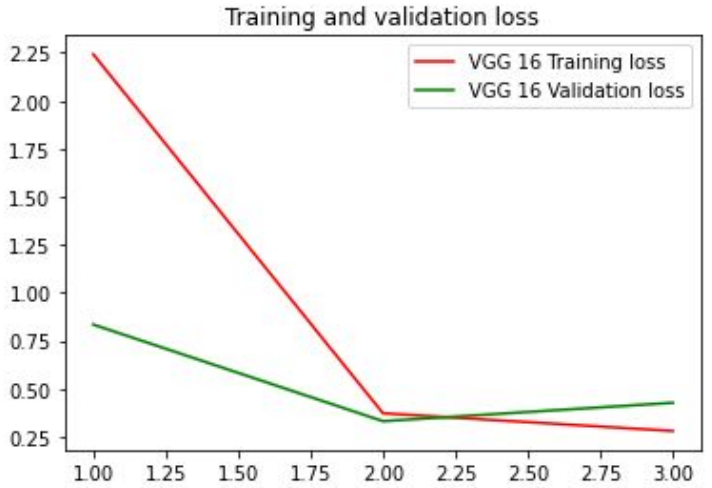
```
model2.add(layers.Dense(128, activation='relu'))
```




```
11/11 [==============================] - 1s 75ms/step - loss: 1.3608 - acc: 0.6581
VGG 16 테스트 데이터 점수 :  [1.360796332359314, 0.6580645442008972, 0.349529623985
```

정확도 : 0.6581

```
model2.add(layers.Dense(256, activation='relu'))
```



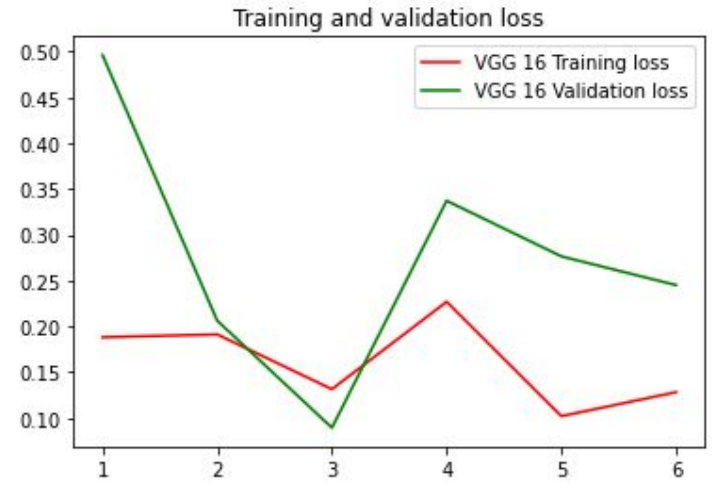

```
11/11 [==============================] - 1s 80ms/step - loss: 0.7336 - acc: 0.7387
VGG 16 테스트 데이터 점수 :  [0.7336379289627075, 0.7387096881866455, 0.29757234454
```

정확도 : 0.7387

03 성능 향상 방법

하이퍼파라미터 조절

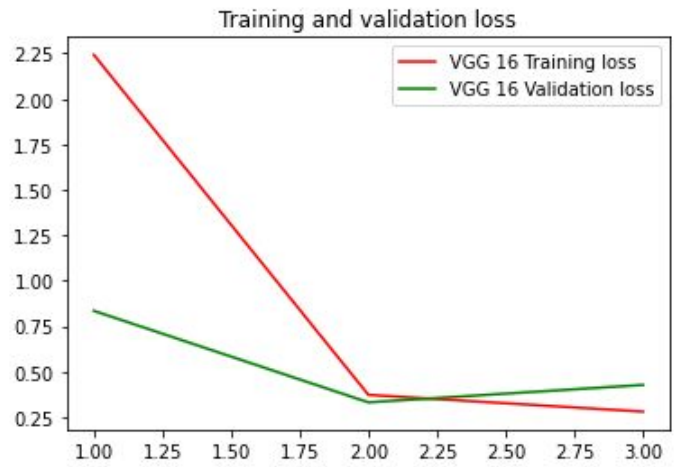
에포크 6




```
11/11 [==============================] - 1s 77ms/step - loss: 2.4002 - acc: 0.6323
VGG 16 테스트 데이터 점수 :  [2.4001567363739014, 0.6322580575942993, 0.35842406749
```

테스트 데이터 정확도 : 0.6323

에포크 3




```
11/11 [==============================] - 1s 80ms/step - loss: 0.7336 - acc: 0.7387
VGG 16 테스트 데이터 점수 :  [0.7336379289627075, 0.7387096881866455, 0.29757234454
```

테스트 데이터 정확도 : 0.7387

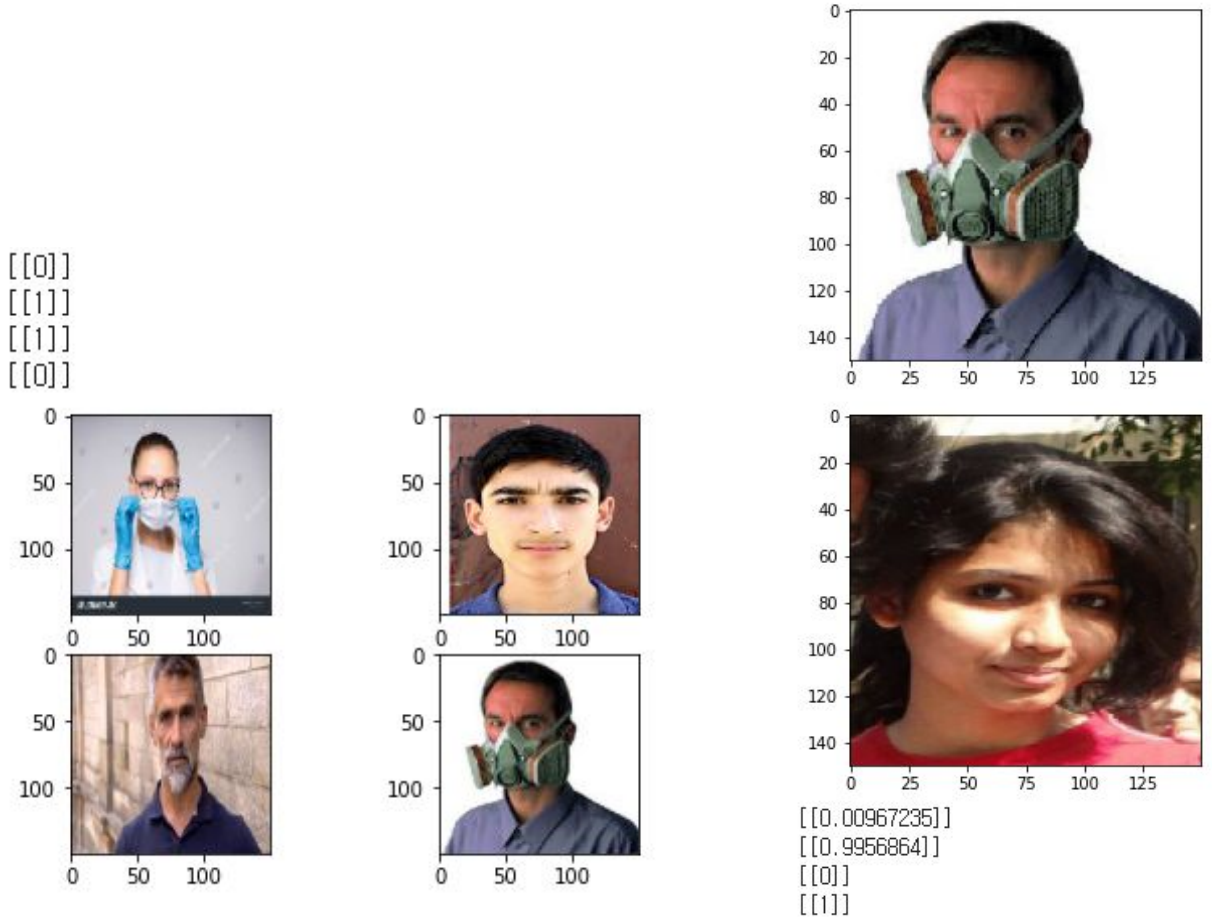
04 결론

프로젝트 진행 중 아쉬운 점

Hyperopt 와 같은 하이퍼 파라미터 자동 튜닝 도구 미사용

프로젝트 진행 중 배운 점

적은 데이터 셋에서 학습 방법



감사합니다