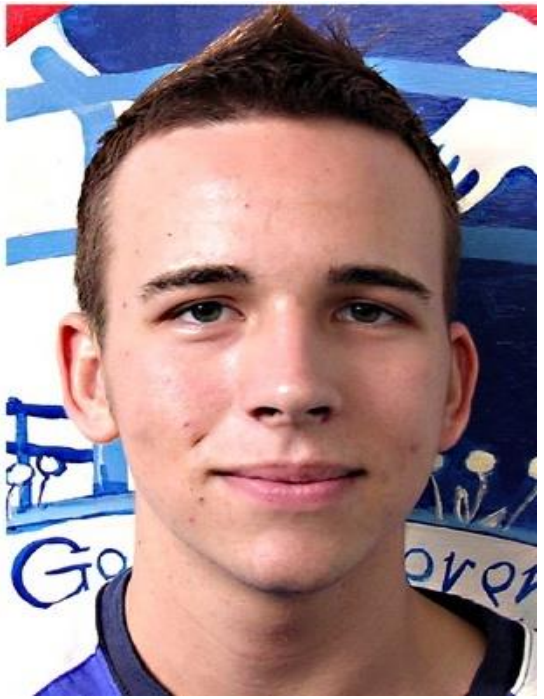








코로나 19대응 마스크 사용여부 판단 프로젝트

조장 : 노현석

조원 : 안병준, 유상윤, 이은진, 민승준

Contents

1. 각 팀원들의 소스 분석 내용을 취합
2. 소스 분석이 제대로 안되는 부분 분석 수행
3. 데이터 증식 전 / 후 비교 수행
4. RESNET 모델 적용 후 테스트 수행 중

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator #강사님이 말씀하진 이미지 제너레이터

train_data=ImageDataGenerator(rescale=1.0/255, zoom_range=0.2, shear_range=0.2) # train data 크기 조정
train_generator = train_data.flow_from_directory(directory=traindir,target_size=(64,64),class_mode='binary',batch_size=32) #어떤식으로 작용하는지는 모르겠습니다
test_data=ImageDataGenerator(rescale=1.0/255) #test data 크기 조정
valid_generator = test_data.flow_from_directory(directory=validdir,target_size=(64,64),class_mode='binary',batch_size=32) #어떤식으로 작용하는지는 모르겠습니다
test_generator = test_data.flow_from_directory(directory=testdir,target_size=(64,64),class_mode='binary',batch_size=32,shuffle=False) #어떤식으로 작용하는지는 모르겠

print(valid_generator.class_indices)
```

```
from tensorflow.keras import layers, Sequential
from keras.applications.vgg19 import VGG19 # VGG19 모델 적용
from keras.applications.vgg19 import preprocess_input

# 사전 교육된 기본 모델 생성
vgg19 = VGG19(weights='imagenet',include_top=False,input_shape=(64,64,3))

for layer in vgg19.layers: # <=이건 잘 모르겠다
    layer.trainable = False


vmodel = Sequential() # 모형 작성
vmodel.add(vgg19) # VGG19 기반 모델 추가
vmodel.add(layers.Flatten()) # 평평하게 만들어 Denser 레이어로 전환을 하기 위함.
vmodel.add(layers.Dense(1,activation='sigmoid')) # conv 후에 activation function 을 sigmoid 로 한다.
vmodel.summary()
# shape와 param 가 왜 이렇게 나오는지 분석 필요함.

vmodel.compile(optimizer='adam',loss='binary_crossentropy', metrics='accuracy')
```

[26] #2021-08-25 14:33분 <== 여기서부터 소스 분석을 수행해야합니다.
!pwd

```
/content/drive/My Drive/PART1_ai_project_20210824_0830
```

+ 코드 + 텍스트

[27] # 아래 부분은 트레인, 벨리데이션, 테스트에 대한 이미지를 가져올 경로를 정해주는 것이죠
traindir='/content/drive/My Drive/PART1_ai_project_20210824_0830/01_Images/Train'
validdir='/content/drive/My Drive/PART1_ai_project_20210824_0830/01_Images/Validation'
testdir='/content/drive/My Drive/PART1_ai_project_20210824_0830/01_Images/Test'

[28] !pwd

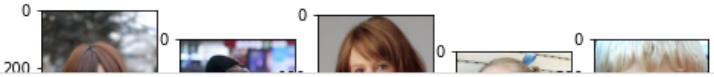
```
/content/drive/My Drive/PART1_ai_project_20210824_0830
```

[29] path='/content/drive/My Drive/PART1_ai_project_20210824_0830/01_Images/All' #전체 이미지가 들어있는 패스를 설정합니다.
#import cv2 # 이거 왜 임포트하죠? 티에서 쓰는게 없는거같은데? - 없어도 되네요?
import random # 값을 랜덤으로 생성하는 함수를 임포트합니다.
import matplotlib.pyplot as plt # matplotlib 라이브러리에 대해서 pyplot 패키지는 단순한 코드로 그래프를 생성할수 있음.
import matplotlib.image as mpimg # matplotlib 라이브러리에 대해서 image 패키지 이미지를 처리할수 있는 패키지

#plt.figure(figsize=(20,20))
cm = 1/2.54 # 센티미터 추가함.
plt.figure(figsize=(20*cm,20*cm)) # 20 센티미터로 하였음.

figure라는 그림단위, 이안에 한개 혹은 여러개의 plot을 관리하도록 함.
figure안에 들어가는 plot 하나를 subplot이라고 부릅니다.
그림(figure)의 크기. (가로,세로) 인치 단위 20, 20
의견 : 저그림이 해당범위내에서 보여줄수있는 최대비율의 사진인거같아요

for i in range(5): # 5개의 사진만 보여줄려고 한다.
file=random.choice(os.listdir(path)) # 설정된 패스에서 1개 파일만 선택을 한다.
img_path=os.path.join(path,file) # 해당 파일이 있는 경로를 조인한다.
image=mpimg.imread(img_path) # 해당 파일을 읽습니다.
ax=plt.subplot(1,5,i+1) # figure 안에 5개의 subplot 중에 처음에는 0+1 해서 1번째꺼에 넣는다.
plt.imshow(image) # 해당 이미지를 보여주게 됩니다.



```
from tensorflow.keras.preprocessing.image import ImageDataGenerator # 이미지에 대한 변환생성을 위해서 케라스의 ImageDataGenerator를 임포트 합니다.

#지금 부터는 해당 이미지의 사이즈를 동일하게 맞추는 작업 즉 train_data, test_data 전처리 작업을 수행하게 됩니다.

#[문의] train_data=ImageDataGenerator(rescale=1.0/255, zoom_range=0.2, shear_range=0.2) #어떤식으로 작용하는지는 모르겠습니다
# ImageDataGenerator
  #rescale=1.0/255
     #rescale: 원본 영상은 0-255의 RGB 계수로 구성되는데, 이 같은 입력값은 모델을 효과적으로 학습시키기에 너무 높습니다
     # (통상적인 learning rate를 사용할 경우). 그래서 이를 1/255로 스케일링하여 0-1 범위로 변환시켜줍니다.
     # 이는 다른 전처리 과정에 앞서 가장 먼저 적용됩니다.
  #zoom_range=0.2 # 20% 확대
  #shear_range=0.2
     # 블로그 예시에서는 45.0으로 보여주었는데 이는 오른쪽으로 45도 인거 같다고 생각이 됨(뇌피셜)
     # (https://ichi.pro/ko/keras-mich-tensorflowleul-sayonghan-imiji-deiteo-jeungdae-tamsaeg-184813206747204)
     #전단 변형은 이미지의 모양을 기울입니다. 이것은 전단 변환에서 하나의 축을 고정하고 전단 각도라는 특정 각도로 이미지를 늘린다는 점에서 회전과 다릅니다.
     #이것은 회전에서 보이지 않는 일종의 '늘이기'를 이미지에 만듭니다. shear_range경사 각도를도 단위로 지정합니다
     #shear_range: Float. Shear Intensity (Shear angle in counter-clockwise direction as radians)


# 위와 같이 수행하면
#train_data에 타입이 하나 만들어지게 됩니다.
# type(train_data) => keras.preprocessing.image.ImageDataGenerator

# ImageDataGenerator는 와플 기계이며 우리가 가지고 있는게 이미지에 해당 합니다.


# 이미지를 불러올 때 폴더명에 맞춰 자동으로 labelling 해준다.('WithMask' : 0 ,'WithoutMask' : 1) 이미지 사이즈는 64 * 64, 배치 사이즈는 32
train_generator = train_data.flow_from_directory(directory=traindir,target_size=(64,64),class_mode='binary',batch_size=32)

# flow_from_directory : 이미지 로드 및 이미지 증식을 할 수 있습니다
# directory=traindir <= 디렉토리 경로
# target_size=(64,64) <= 변환할 크기는 가로 64, 세로 64  단위는 픽셀
# class_mode='binary' <== 마스크 쓰고 안쓰고 2개의 바이너리 클래스로 나타낸다는거죠
# batch_size=32
   #batch_size는 한번 flow_from_directory가 실행 될 때 생성할 이미지 수입니다.
   #즉, 100개의 데이터가 있을때 batch_size가 10이면 flow_from_directory를 10번실행해야 가지고 있는 100개의 데이터를 불러낼수가 있습니다.

# train_data 라는 ImageDataGenerator의 오브젝트에 메소드 중에서 flow_from_directory가 있는데
# flow_from_directory는 디렉토리에 있는걸 가져와서 train_data를 적용해서 반환해주는거죠
# 즉 100개가 있으면 100번 포문 돌아서 적용하는것처럼.
```

https://ichi.pro/ko/keras-mich-tensorflowleul-sayonghan-imiji-deiteo-jeungdae-tamsaeg-184813206747204

5. 전단 강도

```
1  data_generator = ImageDataGenerator(shear_range=45.0)
2  plot(data_generator)
```

shear.py hosted with ❤ by GitHub view raw

전단 변형은 이미지의 모양을 기울입니다. 이것은 전단 변환에서 하나의 축을 고정하고 전단 각도라는 특정 각도로 이미지를 늘린다는 점에서 회전과 다릅니다. 이것은 회전에서 보이지 않는 일종의 '늘이기'를 이미지에 만듭니다. `shear_range` 경사 각도를도 단위로 지정합니다.

```
test_data=ImageDataGenerator(rescale=1.0/255)

# valid generator 도 train generator와 마찬가지로 진행
valid_generator = test_data.flow_from_directory(directory=validdir,target_size=(64,64),class_mode='binary',batch_size=32)

# test 데이터도 train generator 에서 진행한 방식과 동일하고 추가적으로 rescaling을 진행
test_generator = test_data.flow_from_directory(directory=testdir,target_size=(64,64),class_mode='binary',batch_size=32,shuffle=False)

#shuffle=False 석지 않는다. 디폴트는 True
#test는 왜 석으면 안될까요?
 #shuffle=False를 True로 해주고 결과를 한번 확인해보시기 바랍니다. [실습]

print(valid_generator.class_indices)
```

```
Found 600 images belonging to 2 classes.
Found 306 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
{'WithMask': 0, 'WithoutMask': 1}
```

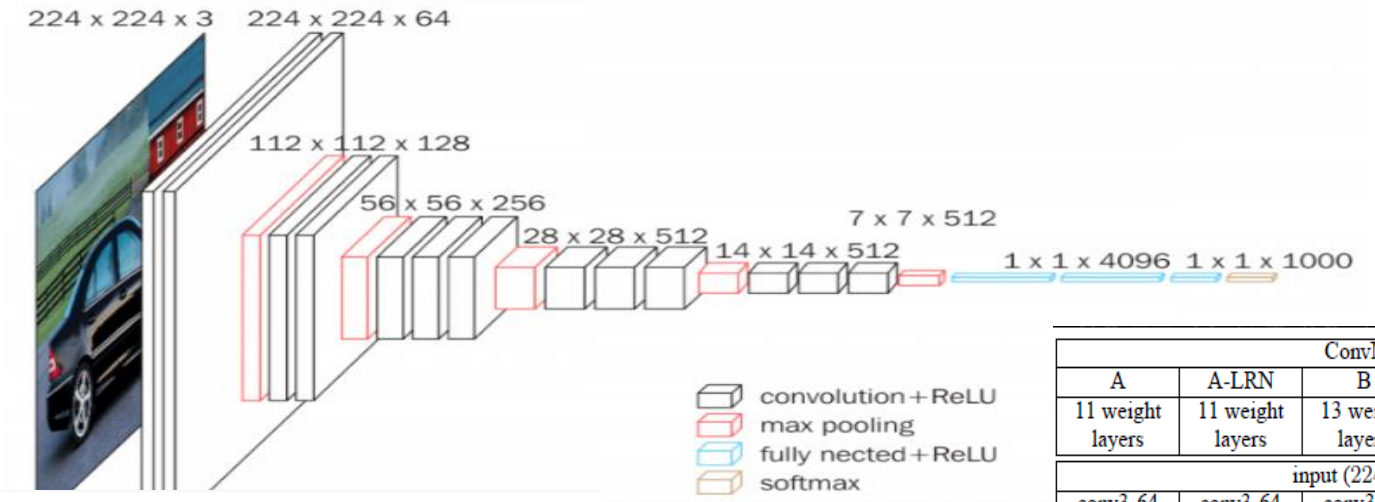
```
[33] # 데이터 증식 적용해봄
     from tensorflow.keras.preprocessing.image import ImageDataGenerator


     # 데이터 증식 수행
     train_data=ImageDataGenerator(rescale=1.0/255,
                                   zoom_range=0.2, # 20% 확대
                                   shear_range=0.2, # 이것은 회전에서 보이지 않는 일종의 '늘이기'를 이미지에 만듭니다
                                    rotation_range=40,  # 40도 회전
                                   width_shift_range=0.2, # 0.2만큼 옆으로 shift
                                   height_shift_range=0.2, # 0.1만큼 위로 shift
                                   horizontal_flip=True) # 인풋을 무작위로 가로로 뒤집습니다.

     train_generator = train_data.flow_from_directory(directory=traindir,target_size=(64,64),class_mode='binary',batch_size=32)

     test_data=ImageDataGenerator(rescale=1.0/255, rotation_range=40, width_shift_range=0.2, height_shift_range=0.2, horizontal_flip=True )

     valid_generator = test_data.flow_from_directory(directory=validdir,target_size=(64,64),class_mode='binary',batch_size=32)
     test_generator = test_data.flow_from_directory(directory=testdir,target_size=(64,64),class_mode='binary',batch_size=32,shuffle=False)

     print(valid_generator.class_indices)
```

```
Found 600 images belonging to 2 classes.
Found 306 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
{'WithMask': 0, 'WithoutMask': 1}
```

```
# 데이터 증식이란거는 우리가 기존 생각하고 있던 데이터의 수가 늘어나는 것이 아니라 여기서는 이미지의 형태를 다르게 해서 딥러닝할때 다양성을 늘리는것으로 보임.
# [민승준] 원본코드 kaggle쪽에서도 제가 건드려봤는데 Found값은 어떻게 제너레이트 하더라도 계속동일하네요
```

+ 코드 + 텍스트

```
[43] from tensorflow.keras import layers, Sequential
     # layers는 기존에 수작업으로 하던 부분들에 대해서 레이어 부분을 클래스화 시킨 부분입니다.
     # 케라스 코어에서 보면
     # Sequential 모델은 각 레이어에 정확히 하나의 입력 텐서와 하나의 출력 텐서가 있는 일반 레이어 스택에 적합합니다.

     from keras.applications.vgg19 import VGG19 # VGG19 모델 적용
     # VGG-19는 19개 계층으로 구성된 컨벌루션 신경망입니다

     from keras.applications.vgg19 import preprocess_input # VGG19 모델에 데이터 입력받는 프로세서 임포트


     vgg19 = VGG19(weights='imagenet',include_top=False,input_shape=(64,64,3))
       # VGG19 모델을 생성합니다.
       # weights: None으로 지정시 랜덤 초기화,
                 #'imagenet'로 지정시  (pre-training on ImageNet), 선행학습(pre-training) or 사전훈련(pre-training) or 전처리과정(pre-training)
                 #이라고도 하는데, 이는 Multi Layered Perceptron (MLP)에서 Weight와 Bias를 잘 초기화 시키는 방법이다.
                 #이러한 Pre-training을 통해서 효과적으로 layer를 쌓아서 여러개의 hidden layer도 효율적으로 훈련 할 수 있다.
       # include_top :
         # include_top은 상단의 F.C 레이어를 포함할 것인이 아닌지 결정하는 파라미터이다.
         # 나는 classifier를 새로 만들었기 때문에 include_top을 False로 주었다.
       # input_shape : (64,64,3) 는 초기들어가는 이미지가 위에서 이미지 제러레이터를 했을때 64*64 픽셀로 만들었고 RGB이기 때문에 3으로 들어가야하는데
                       # 이부분은 조원들하고 상의를 해봐야겠다.<================================================
                       # optional shape tuple, only to be specified if include_top is False (otherwise the input shape has to be (224, 224, 3)
                       #(with channels_last data format) or (3, 224, 224) (with channels_first data format).
                       #It should have exactly 3 input channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value.

     vgg19.summary() ## 22개 레이어가 존재하고 끝에 FC(flatten_2 와 dense_2)가 없는 모델이다.

     for layer in vgg19.layers: ## 22개 레이어를 꺼내서
         layer.trainable = False ## 속성을 변경합니다. 훈련하지 않도록

     vmodel = Sequential() # 모형 작성
     vmodel.add(vgg19) # VGG19 기반 모델 추가
     vmodel.add(layers.Flatten()) # 평평하게 만들어 Denser 레이어로 전환을 하기 위함.
     vmodel.add(layers.Dense(1,activation='sigmoid')) # conv 후에 activation function 을 sigmoid 로 한다.
```

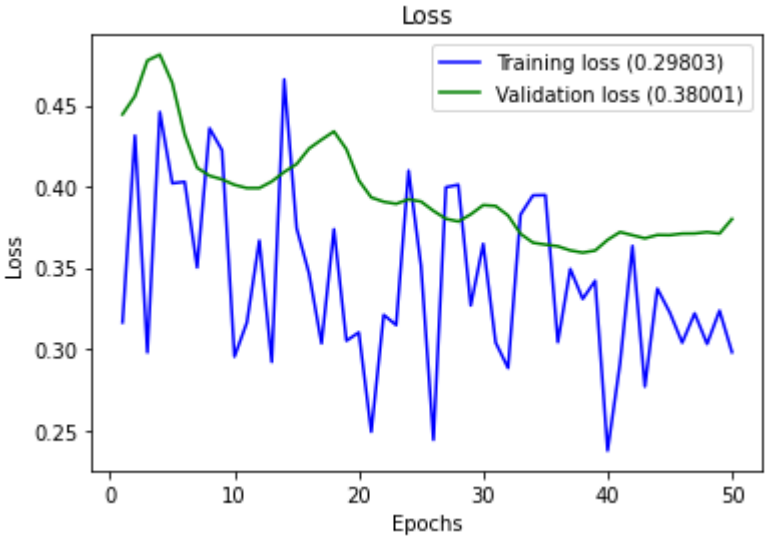


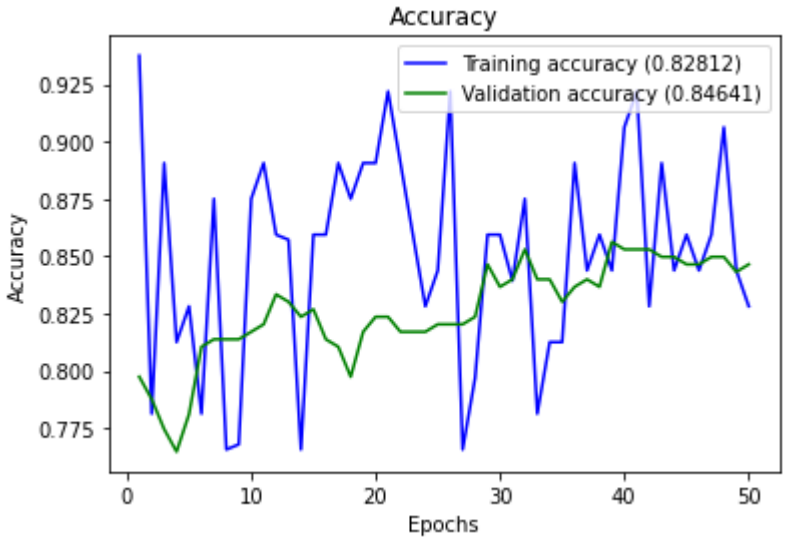
Model: "vgg19"

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 64, 64, 3)]	0
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 32, 32, 128)	73856
block2_conv2 (Conv2D)	(None, 32, 32, 128)	147584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
block3_conv1 (Conv2D)	(None, 16, 16, 256)	295168
block3_conv2 (Conv2D)	(None, 16, 16, 256)	590080
block3_conv3 (Conv2D)	(None, 16, 16, 256)	590080
block3_conv4 (Conv2D)	(None, 16, 16, 256)	590080
block3_pool (MaxPooling2D)	(None, 8, 8, 256)	0
block4_conv1 (Conv2D)	(None, 8, 8, 512)	1180160
block4_conv2 (Conv2D)	(None, 8, 8, 512)	2359808

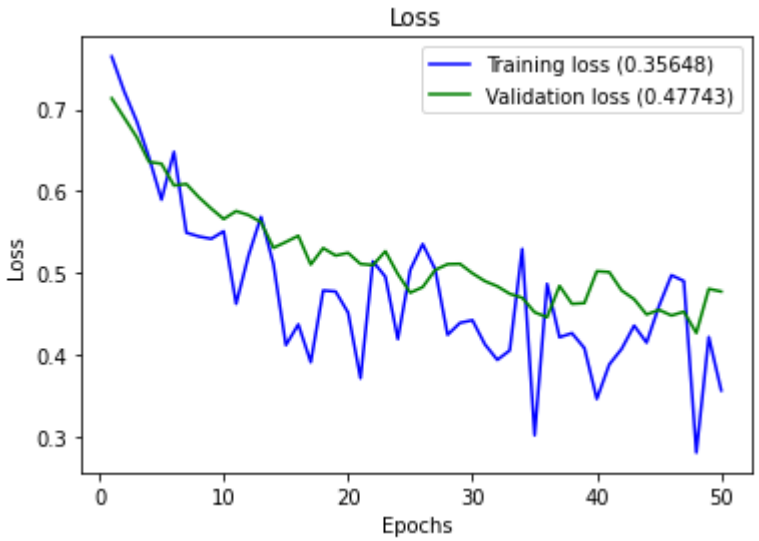
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

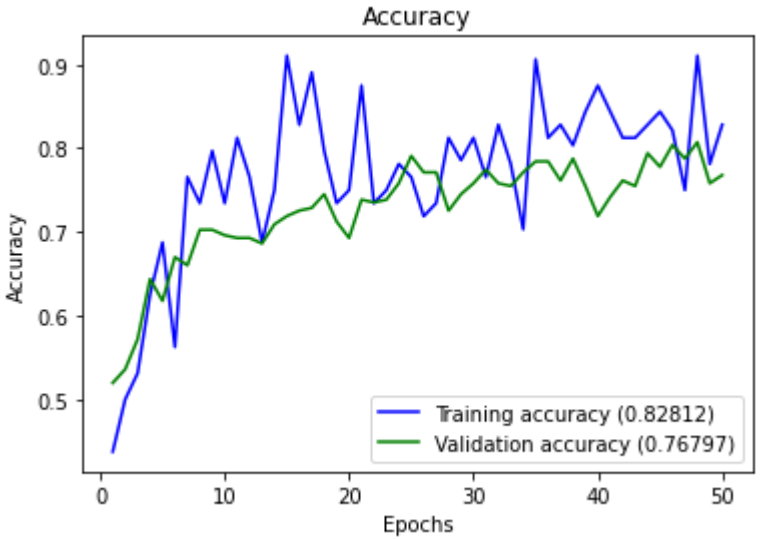
데이터 증식전





데이터 증식후





데이터증식전

1	이미지 사이즈 변경에 따른 결과 확인
2	DensNet 적용 결과 확인
3	ResNet 적용 결과 확인
4	ResNet V2 적용 결과 확인
5	VGG19, DensNet, ResNet, ResNet V2 정확도 비교
6	VGG19, DensNet, ResNet, ResNet V2 Loss 비교
7	ResNext 적용 결과 확인

20210825 강사님 말씀

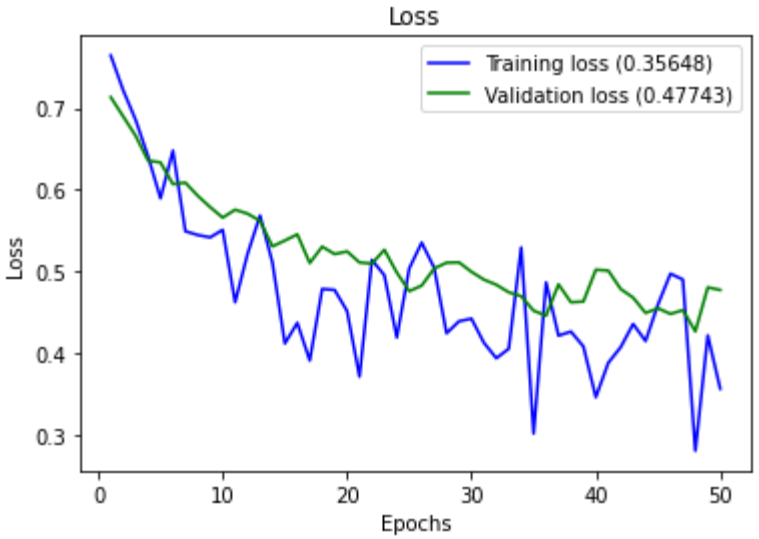
plot의 사이즈가 고정되지 않음
아래와 같이 수행하면 고정된 화면을 보여줌
- fig, axes = plt.subplots(3,4, figsize=(20,20))

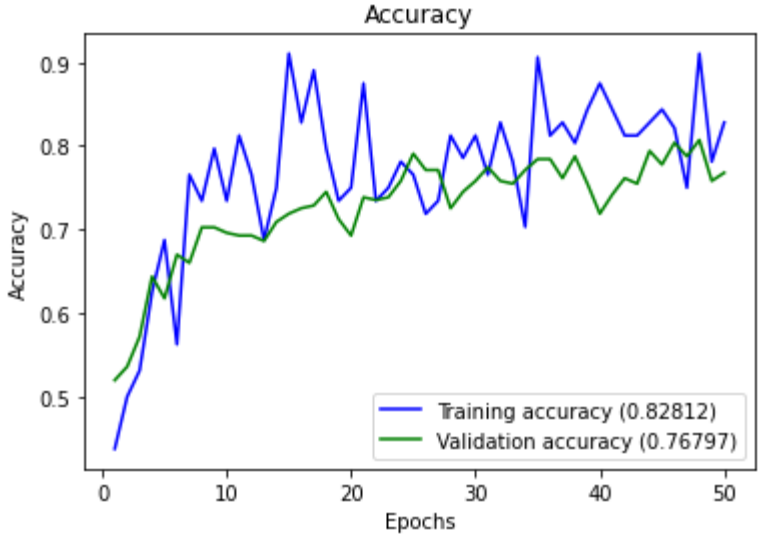
shear_range = 45.0 문의
- 해당 값은 45도가 맞음
- -45도 ~ +45 사이에 값으로 비틀겠다.

이미지 사이즈 너무 작음
- 기존 64*64로 분별력이 너무 작음.
- 이미지 사이즈를 높여 분별력을 높이는게 좋음
- 256*256 정도로 높여서 테스트 필요

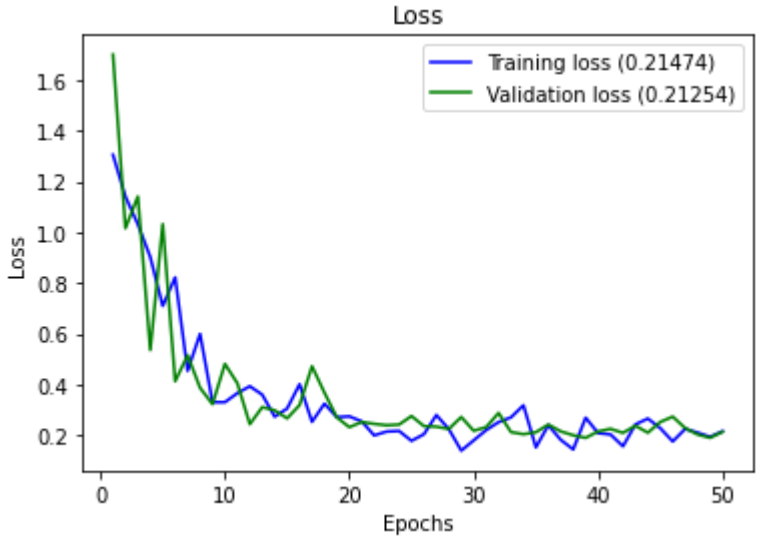
최종보고서에는 사회기여도가 포함되었으면 좋겠음.

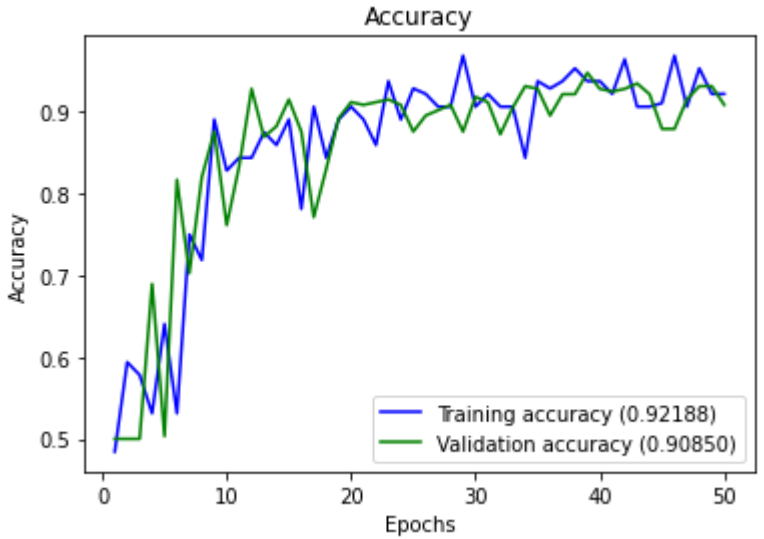
데이터 증식 + 이미지 (64*64) + VGG19



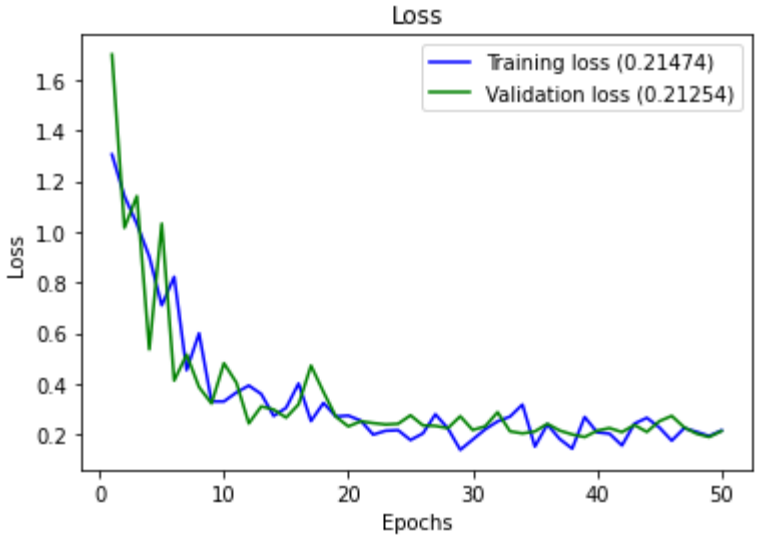


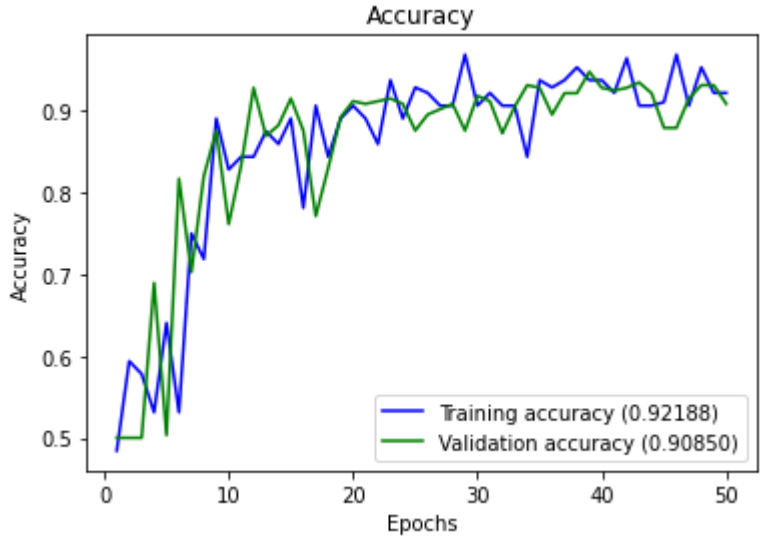
데이터 증식 + **이미지 (256*256)** + VGG19



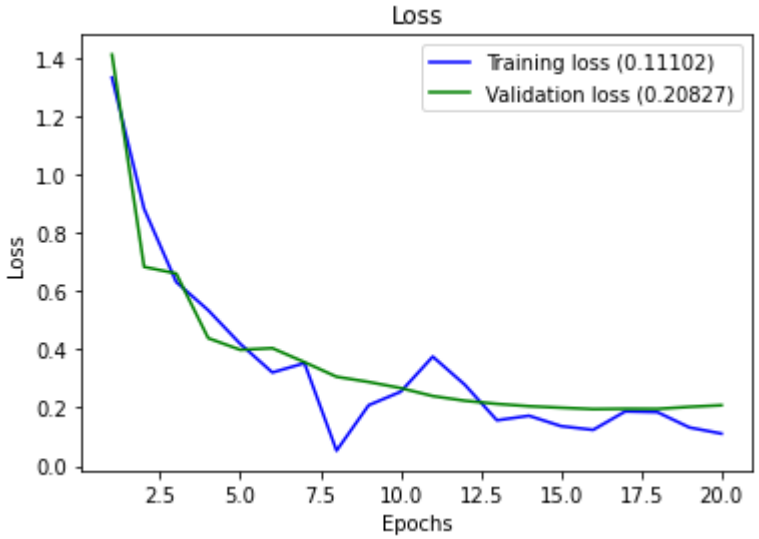


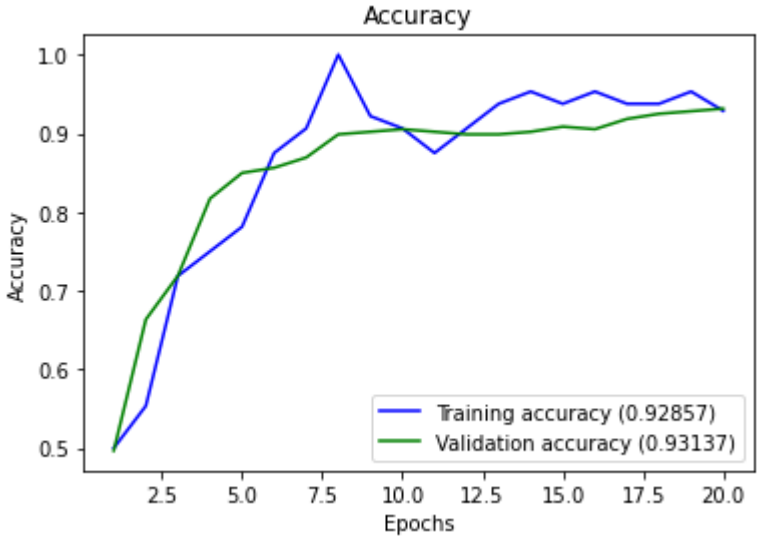
데이터 증식 + 이미지 (256*256) + VGG19





데이터 증식 + 이미지 (256*256) + **DenseNet (201)**





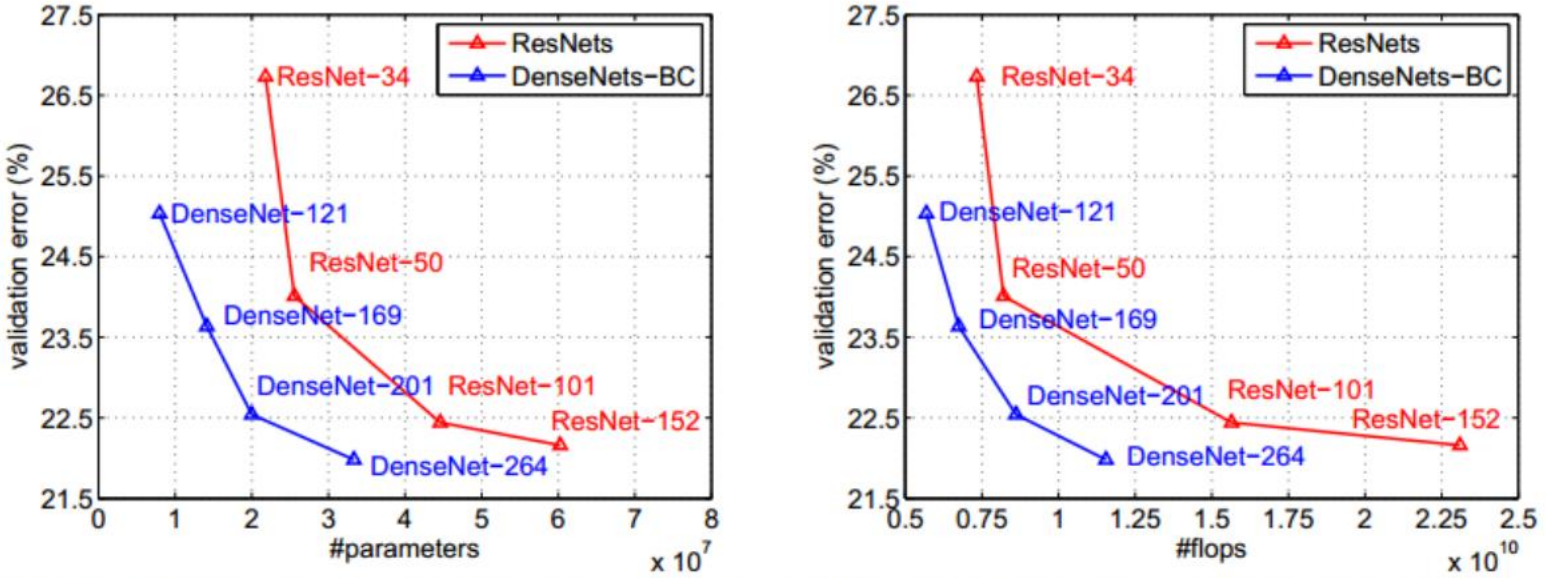
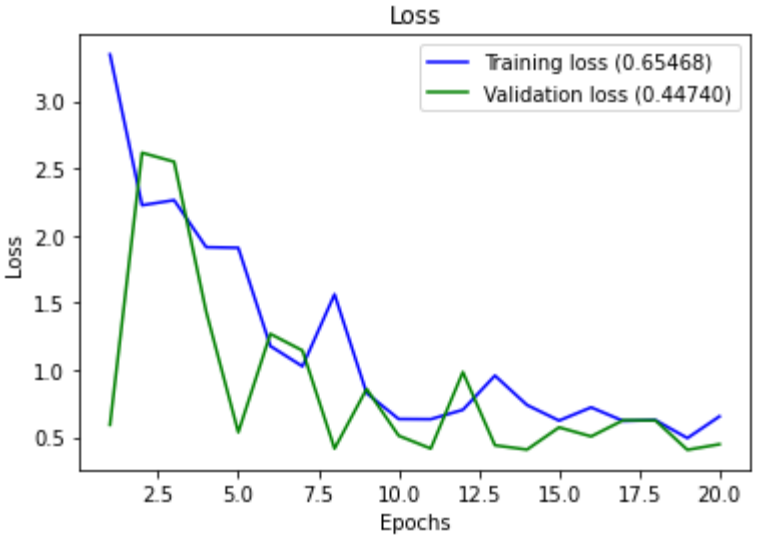
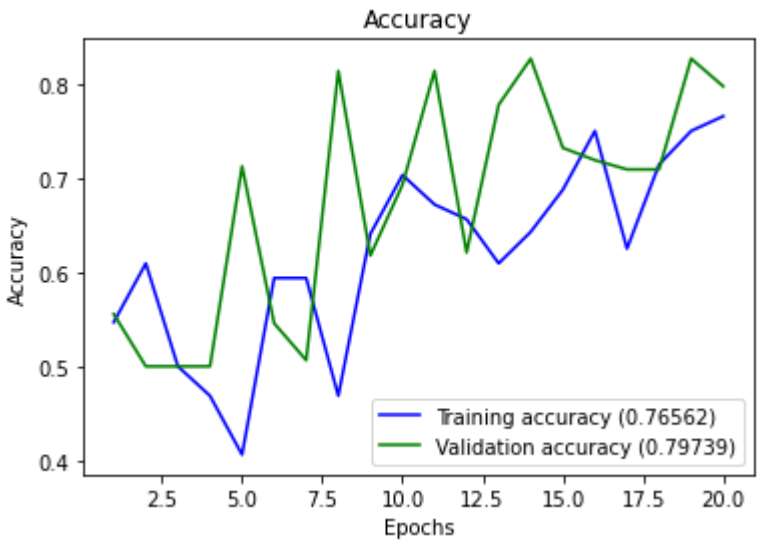


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

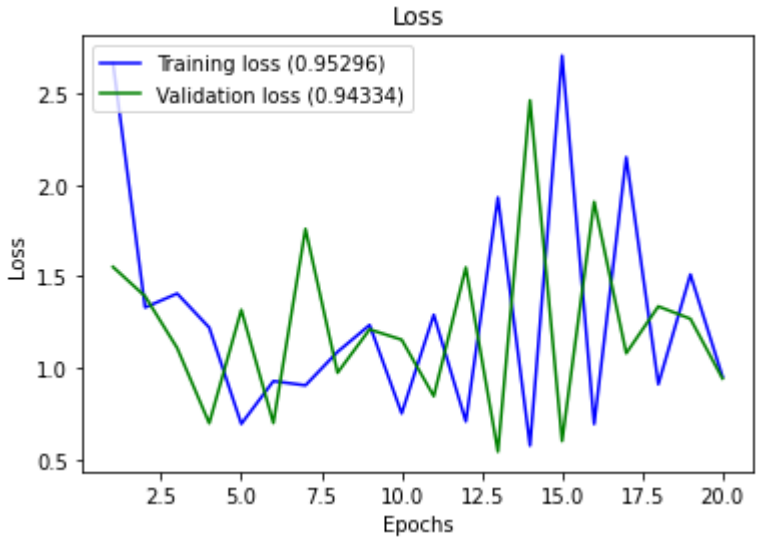
DenseNet 264는 케라스에서 제공하지 않으므로 제공하는 레이어중에 에러율이 적은 DenseNet 201 사용했습니다

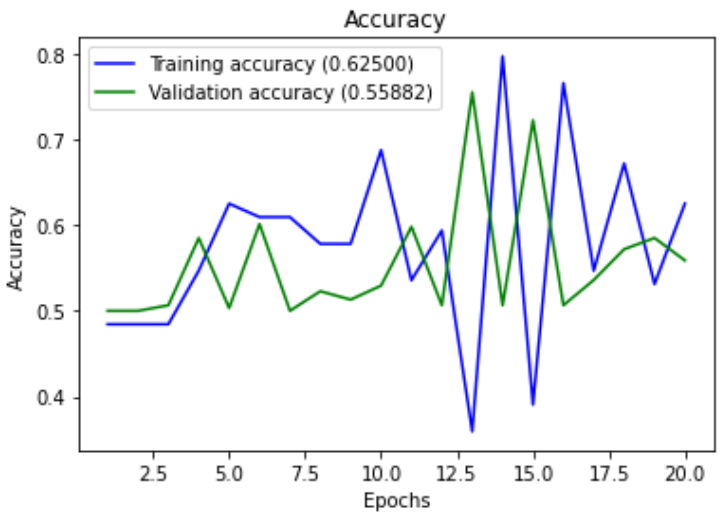
데이터 증식 + 이미지 (256*256) + **ResNet (50)**



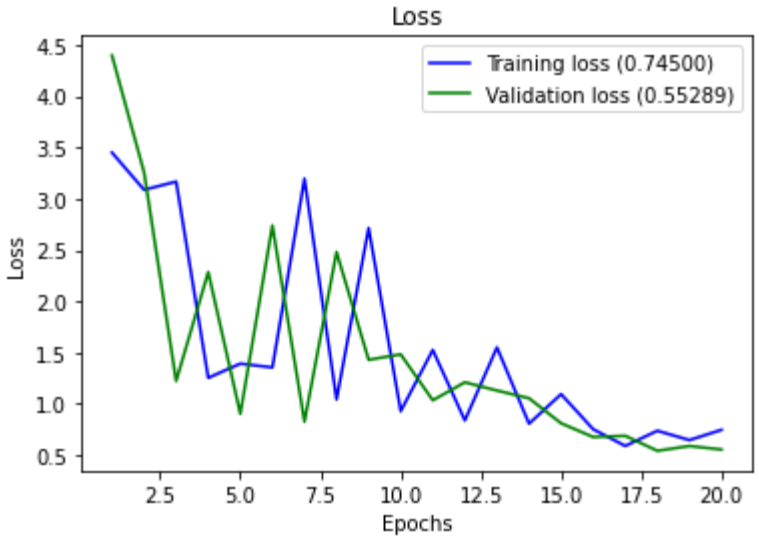


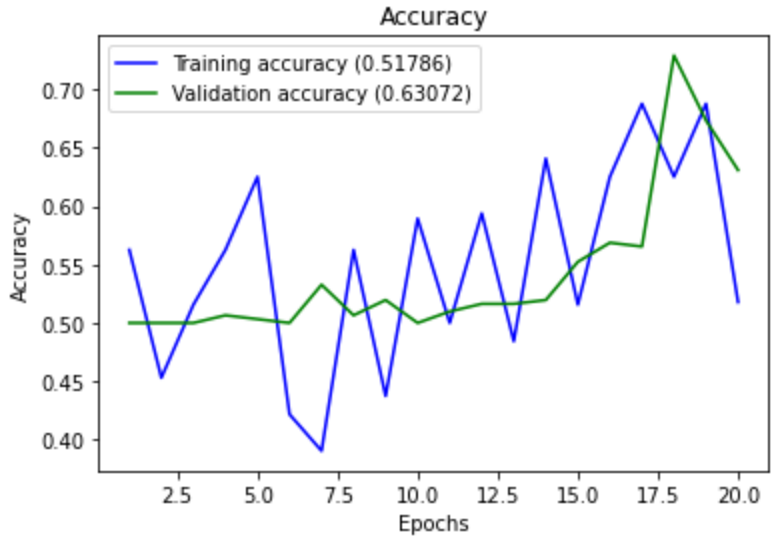
데이터 증식 + 이미지 (256*256) + **ResNet (101)**



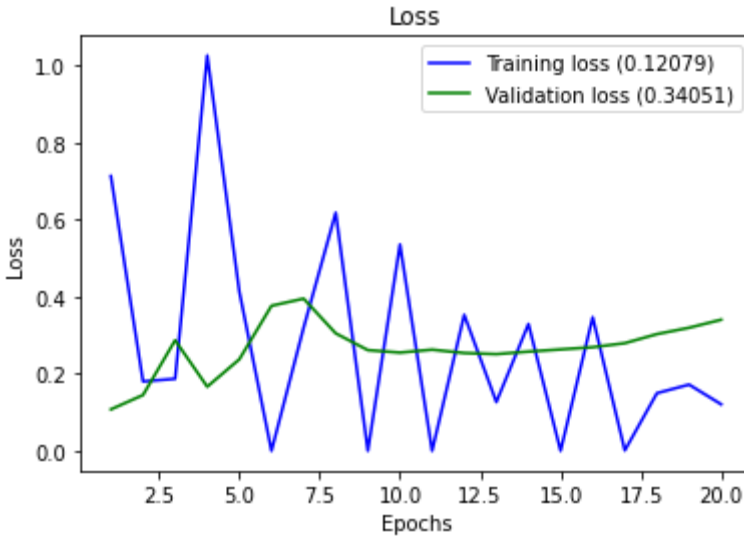


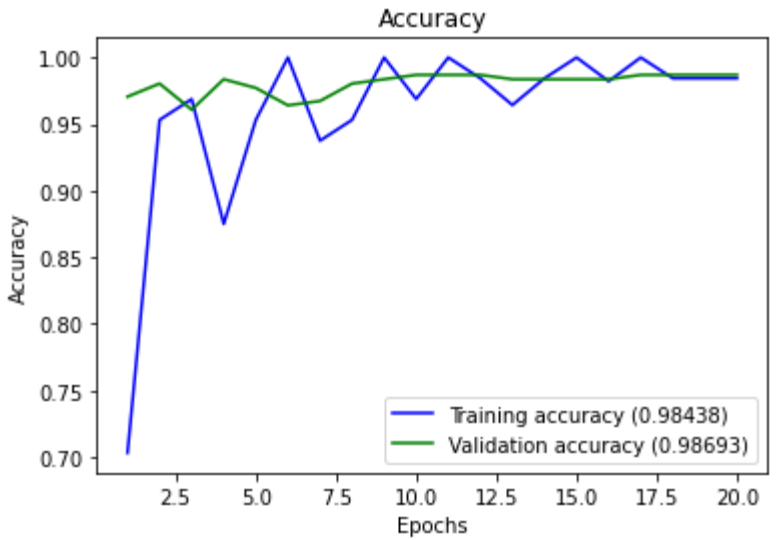
데이터 증식 + 이미지 (256*256) + **ResNet (152)**



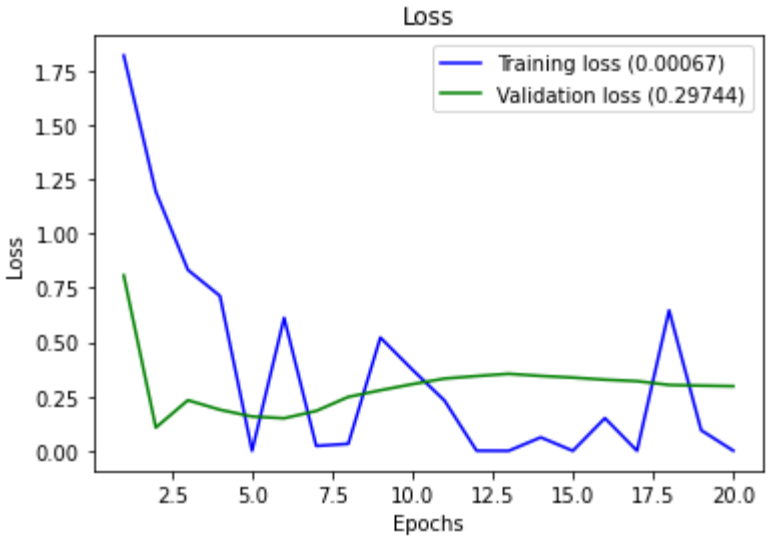


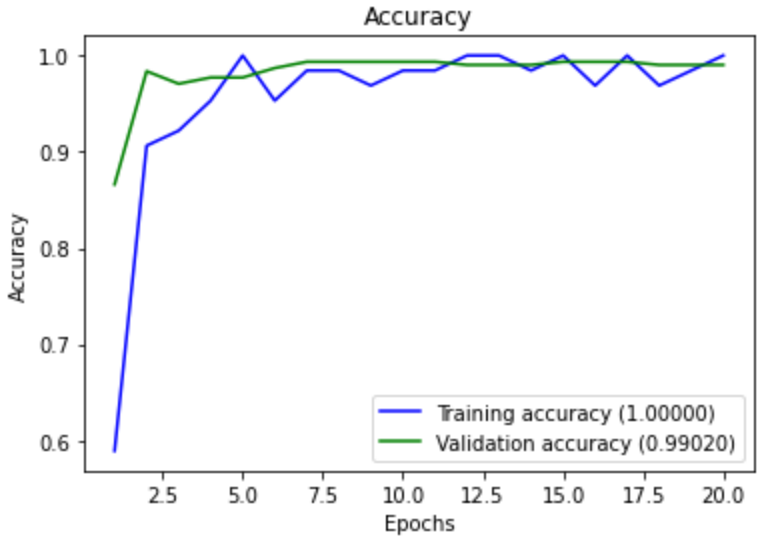
데이터 증식 + 이미지 (256*256) + **ResNetv2 (50)**





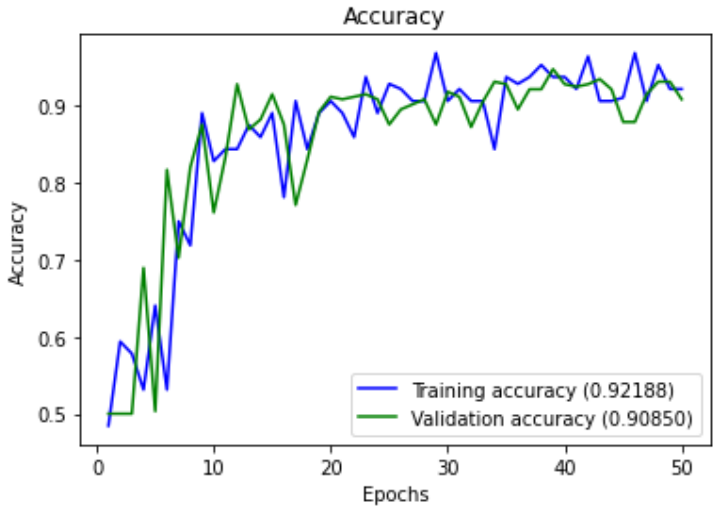
데이터 증식 + 이미지 (256*256) + **ResNetv2 (101)**



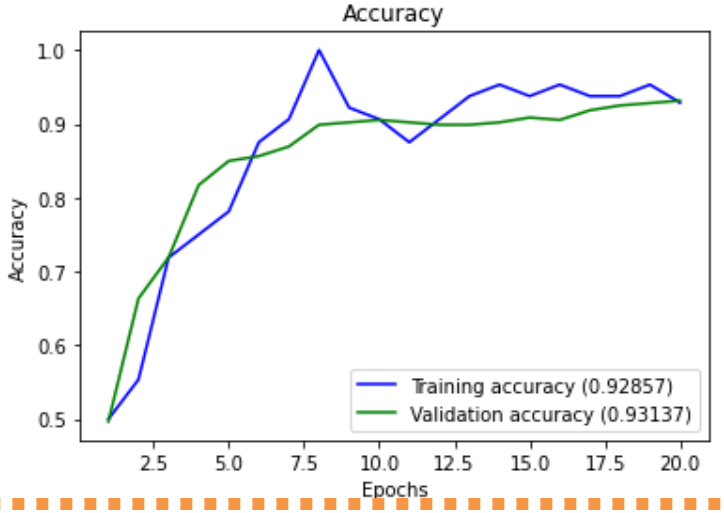


데이터 증식 + 이미지 (256*256) + **ResNetv2 (152)**

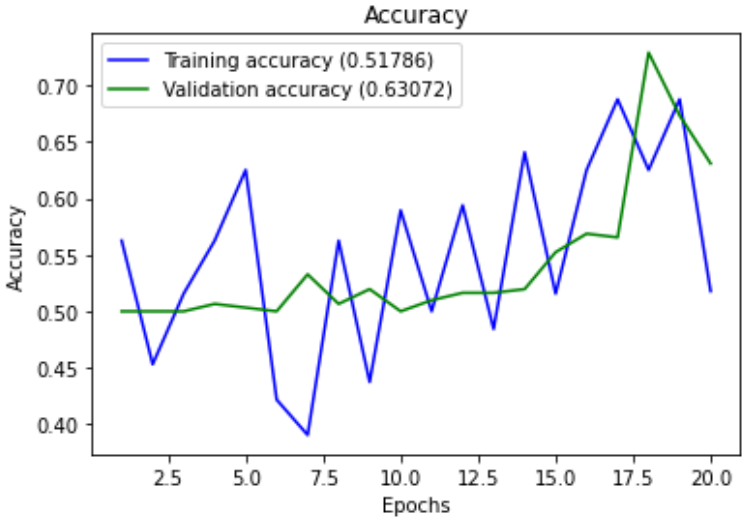
데이터증식+이미지(256*256)+**VGG19**



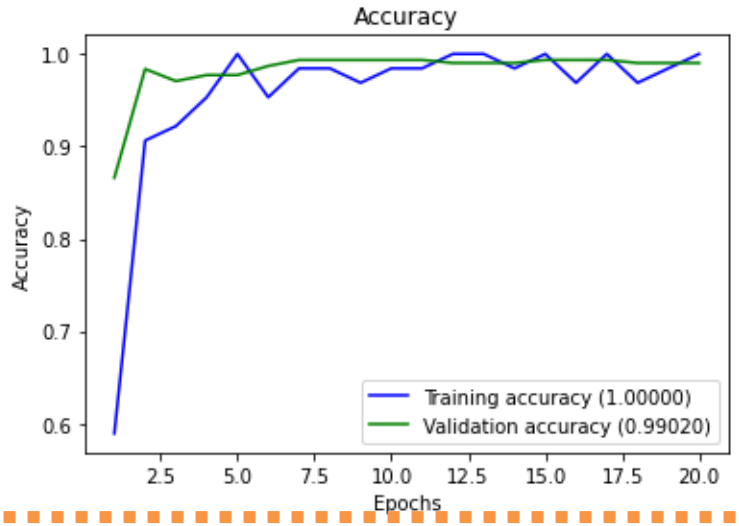
데이터증식+이미지(256*256)+**DenseNet(201)**



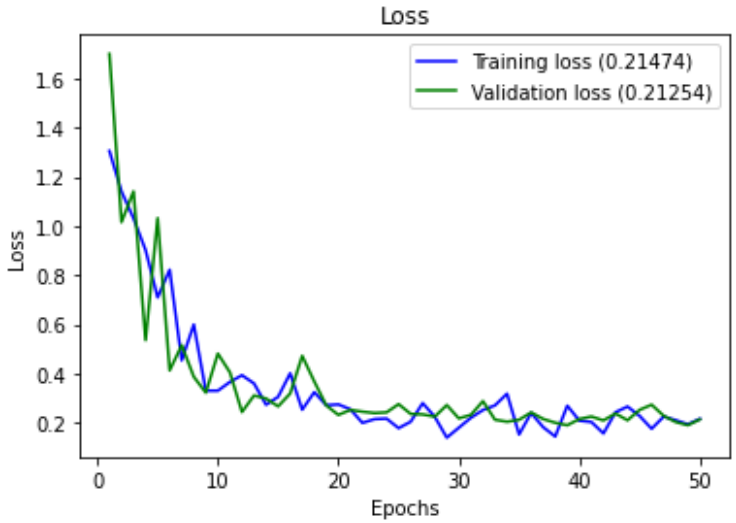
데이터증식+이미지(256*256)+**ResNet(152)**



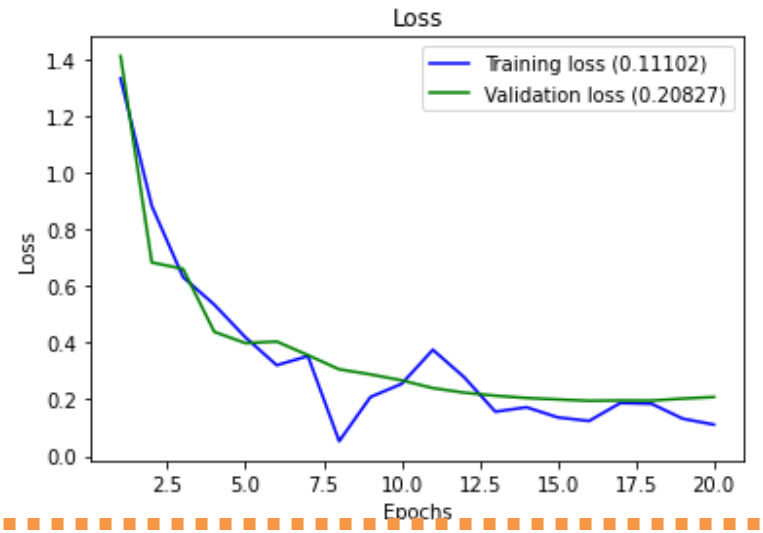
데이터증식+이미지(256*256)+**ResNetv2(101)**



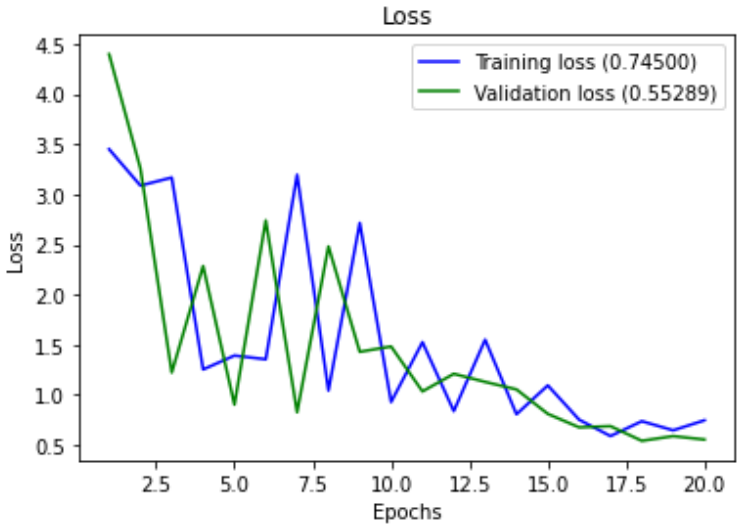
데이터증식+이미지(256*256)+**VGG19**



데이터증식+이미지(256*256)+**DenseNet(201)**



데이터증식+이미지(256*256)+**ResNet(152)**



데이터증식+이미지(256*256)+**ResNetv2(101)**

데이터증식+이미지(256*256)+**ResNext(50)**

```
Epoch 1/20
2/2 [==============================] - 196s 146s/step - loss: 2.4835 - accuracy: 0.4219 - val_loss: 0.6940 - val_accuracy: 0.5000
Epoch 2/20
2/2 [==============================] - 186s 139s/step - loss: 0.9075 - accuracy: 0.5000 - val_loss: 0.6942 - val_accuracy: 0.5000
Epoch 3/20
2/2 [==============================] - 180s 134s/step - loss: 0.8884 - accuracy: 0.5469 - val_loss: 0.6947 - val_accuracy: 0.5000
Epoch 4/20
2/2 [==============================] - 181s 139s/step - loss: 0.6683 - accuracy: 0.5312 - val_loss: 0.6938 - val_accuracy: 0.5000
Epoch 5/20
2/2 [==============================] - 185s 140s/step - loss: 0.6541 - accuracy: 0.4219 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6/20
2/2 [==============================] - 186s 141s/step - loss: 0.6265 - accuracy: 0.5781 - val_loss: 0.7107 - val_accuracy: 0.5000
Epoch 7/20
2/2 [==============================] - 185s 139s/step - loss: 0.7731 - accuracy: 0.5469 - val_loss: 0.6928 - val_accuracy: 0.5000
Epoch 8/20
2/2 [==============================] - 187s 142s/step - loss: 0.4411 - accuracy: 0.5781 - val_loss: 0.7184 - val_accuracy: 0.5000
Epoch 9/20
2/2 [==============================] - 185s 140s/step - loss: 0.5228 - accuracy: 0.5938 - val_loss: 0.7606 - val_accuracy: 0.5000
Epoch 10/20
2/2 [==============================] - 173s 131s/step - loss: 0.5874 - accuracy: 0.5156 - val_loss: 0.7184 - val_accuracy: 0.5000
Epoch 11/20
2/2 [==============================] - 178s 136s/step - loss: 0.5924 - accuracy: 0.5000 - val_loss: 0.6967 - val_accuracy: 0.5000
Epoch 12/20
2/2 [==============================] - 184s 139s/step - loss: 0.4924 - accuracy: 0.5312 - val_loss: 0.6939 - val_accuracy: 0.5000
Epoch 13/20
2/2 [==============================] - 183s 139s/step - loss: 0.6429 - accuracy: 0.4688 - val_loss: 0.7710 - val_accuracy: 0.5000
Epoch 14/20
2/2 [==============================] - 184s 139s/step - loss: 0.5279 - accuracy: 0.5000 - val_loss: 0.7931 - val_accuracy: 0.5000
Epoch 15/20
2/2 [==============================] - 183s 138s/step - loss: 0.5875 - accuracy: 0.5156 - val_loss: 0.7611 - val_accuracy: 0.5000
Epoch 16/20
2/2 [==============================] - 182s 137s/step - loss: 0.4442 - accuracy: 0.3906 - val_loss: 0.7021 - val_accuracy: 0.5000
Epoch 17/20
2/2 [==============================] - 162s 120s/step - loss: 0.5807 - accuracy: 0.4286 - val_loss: 0.6856 - val_accuracy: 0.5000
Epoch 18/20
2/2 [==============================] - 173s 131s/step - loss: 0.3577 - accuracy: 0.5156 - val_loss: 0.7245 - val_accuracy: 0.5000
Epoch 19/20
2/2 [==============================] - 174s 132s/step - loss: 0.4188 - accuracy: 0.4375 - val_loss: 0.6988 - val_accuracy: 0.5000
Epoch 20/20
2/2 [==============================] - 173s 131s/step - loss: 0.4263 - accuracy: 0.4219 - val_loss: 0.6820 - val_accuracy: 0.5000
```

FIT 수행
steps_per_epoch = len(train_generator)//**19**

```
Epoch 1/20
1/1 [==============================] - 14s 14s/step - loss: 0.1988 - accuracy: 0.9062 - val_loss: 0.2318 - val_accuracy: 0.9020
Epoch 2/20
1/1 [==============================] - 13s 13s/step - loss: 0.2251 - accuracy: 0.9375 - val_loss: 0.2610 - val_accuracy: 0.9085
Epoch 3/20
1/1 [==============================] - 13s 13s/step - loss: 0.1577 - accuracy: 0.9375 - val_loss: 0.2717 - val_accuracy: 0.8889
Epoch 4/20
1/1 [==============================] - 13s 13s/step - loss: 0.2985 - accuracy: 0.9062 - val_loss: 0.2703 - val_accuracy: 0.8922
Epoch 5/20
1/1 [==============================] - 13s 13s/step - loss: 0.2613 - accuracy: 0.9062 - val_loss: 0.2693 - val_accuracy: 0.8791
Epoch 6/20
1/1 [==============================] - 13s 13s/step - loss: 0.3803 - accuracy: 0.8438 - val_loss: 0.2369 - val_accuracy: 0.9052
Epoch 7/20
1/1 [==============================] - 13s 13s/step - loss: 0.1550 - accuracy: 0.9062 - val_loss: 0.2504 - val_accuracy: 0.8889
Epoch 8/20
1/1 [==============================] - 13s 13s/step - loss: 0.1265 - accuracy: 0.9375 - val_loss: 0.2453 - val_accuracy: 0.8889
Epoch 9/20
1/1 [==============================] - 13s 13s/step - loss: 0.1662 - accuracy: 0.9375 - val_loss: 0.2142 - val_accuracy: 0.9314
Epoch 10/20
1/1 [==============================] - 12s 12s/step - loss: 0.2106 - accuracy: 0.9375 - val_loss: 0.2225 - val_accuracy: 0.9052
Epoch 11/20
1/1 [==============================] - 12s 12s/step - loss: 0.1922 - accuracy: 0.9062 - val_loss: 0.2045 - val_accuracy: 0.9281
Epoch 12/20
1/1 [==============================] - 12s 12s/step - loss: 0.3028 - accuracy: 0.8125 - val_loss: 0.2221 - val_accuracy: 0.9052
Epoch 13/20
1/1 [==============================] - 12s 12s/step - loss: 0.1331 - accuracy: 0.9688 - val_loss: 0.2285 - val_accuracy: 0.9248
Epoch 14/20
1/1 [==============================] - 12s 12s/step - loss: 0.2057 - accuracy: 0.8750 - val_loss: 0.2414 - val_accuracy: 0.9052
Epoch 15/20
1/1 [==============================] - 12s 12s/step - loss: 0.2599 - accuracy: 0.9375 - val_loss: 0.2345 - val_accuracy: 0.9085
Epoch 16/20
1/1 [==============================] - 12s 12s/step - loss: 0.2005 - accuracy: 0.9062 - val_loss: 0.2166 - val_accuracy: 0.9281
Epoch 17/20
1/1 [==============================] - 12s 12s/step - loss: 0.2674 - accuracy: 0.8438 - val_loss: 0.2221 - val_accuracy: 0.9183
Epoch 18/20
1/1 [==============================] - 12s 12s/step - loss: 0.2270 - accuracy: 0.9375 - val_loss: 0.2829 - val_accuracy: 0.8922
Epoch 19/20
1/1 [==============================] - 12s 12s/step - loss: 0.2472 - accuracy: 0.8750 - val_loss: 0.3223 - val_accuracy: 0.8529
Epoch 20/20
1/1 [==============================] - 12s 12s/step - loss: 0.2044 - accuracy: 0.9375 - val_loss: 0.3709 - val_accuracy: 0.8268
```

FIT 수행
steps_per_epoch = len(train_generator)//**8**

```
Epoch 1/20
2/2 [==============================] - 15s 14s/step - loss: 0.3996 - accuracy: 0.7969 - val_loss: 0.3158 - val_accuracy: 0.8791
Epoch 2/20
2/2 [==============================] - 13s 12s/step - loss: 0.2944 - accuracy: 0.8906 - val_loss: 0.5870 - val_accuracy: 0.6993
Epoch 3/20
2/2 [==============================] - 13s 12s/step - loss: 0.5616 - accuracy: 0.6607 - val_loss: 0.5042 - val_accuracy: 0.7549
Epoch 4/20
2/2 [==============================] - 14s 12s/step - loss: 0.3605 - accuracy: 0.7812 - val_loss: 0.3056 - val_accuracy: 0.9020
Epoch 5/20
2/2 [==============================] - 13s 12s/step - loss: 0.4835 - accuracy: 0.7812 - val_loss: 0.3598 - val_accuracy: 0.8366
Epoch 6/20
2/2 [==============================] - 13s 12s/step - loss: 0.3066 - accuracy: 0.9375 - val_loss: 0.3517 - val_accuracy: 0.8464
Epoch 7/20
2/2 [==============================] - 13s 12s/step - loss: 0.4364 - accuracy: 0.8281 - val_loss: 0.2736 - val_accuracy: 0.8987
Epoch 8/20
2/2 [==============================] - 13s 12s/step - loss: 0.3016 - accuracy: 0.8906 - val_loss: 0.2783 - val_accuracy: 0.8889
Epoch 9/20
2/2 [==============================] - 13s 12s/step - loss: 0.3299 - accuracy: 0.8750 - val_loss: 0.3653 - val_accuracy: 0.8497
Epoch 10/20
2/2 [==============================] - 13s 12s/step - loss: 0.4133 - accuracy: 0.8281 - val_loss: 0.2661 - val_accuracy: 0.8889
Epoch 11/20
2/2 [==============================] - 14s 12s/step - loss: 0.2265 - accuracy: 0.9062 - val_loss: 0.2252 - val_accuracy: 0.9216
Epoch 12/20
2/2 [==============================] - 13s 12s/step - loss: 0.3376 - accuracy: 0.8125 - val_loss: 0.2451 - val_accuracy: 0.9150
Epoch 13/20
2/2 [==============================] - 14s 12s/step - loss: 0.2210 - accuracy: 0.9375 - val_loss: 0.2309 - val_accuracy: 0.9085
Epoch 14/20
2/2 [==============================] - 14s 13s/step - loss: 0.2387 - accuracy: 0.9219 - val_loss: 0.2661 - val_accuracy: 0.8987
Epoch 15/20
2/2 [==============================] - 13s 12s/step - loss: 0.1285 - accuracy: 0.9688 - val_loss: 0.2969 - val_accuracy: 0.8660
Epoch 16/20
2/2 [==============================] - 14s 13s/step - loss: 0.2543 - accuracy: 0.9062 - val_loss: 0.3116 - val_accuracy: 0.8693
Epoch 17/20
2/2 [==============================] - 13s 12s/step - loss: 0.3058 - accuracy: 0.9062 - val_loss: 0.2307 - val_accuracy: 0.9118
Epoch 18/20
2/2 [==============================] - 13s 12s/step - loss: 0.3519 - accuracy: 0.8906 - val_loss: 0.2079 - val_accuracy: 0.9346
Epoch 19/20
2/2 [==============================] - 13s 12s/step - loss: 0.1993 - accuracy: 0.9375 - val_loss: 0.2347 - val_accuracy: 0.9085
Epoch 20/20
2/2 [==============================] - 13s 12s/step - loss: 0.3090 - accuracy: 0.8906 - val_loss: 0.2414 - val_accuracy: 0.8987
```

steps_per_epoch에 따른 결과가 매우 차이가 많이 남
steps_per_epoch = len(train_generator)//8 = val_accuracy: **0.9346**
steps_per_epoch = len(train_generator)//19 = val_accuracy: **0.7288**

다시 한번 돌려보니 //19 에서도 0.93이 나오게 됨
steps_per_epoch=len(train_generator) 로 돌려보니 최대 val_accuracy: 0.9641

```
# 데이터 증식
train_data=ImageDataGenerator(rescale=1.0/255, zoom_range=0.2,
                              shear_range=0.2, rotation_range=40,
                              width_shift_range=0.2, height_shift_range=0.2,
                              horizontal_flip=True)

train_generator = train_data.flow_from_directory(directory=traindir,target_siz
e=(256,256),class_mode='binary',batch_size=32)

# 모델 fit 수행
history = vmodel.fit(train_generator
                     ,steps_per_epoch=len(train_generator)//8
                     ,epochs=20
                      ,validation_data=valid_generator
                      ,batch_size=128)
```

제러레이터 할때 배치 사이즈와
fit을 수행할때 steps_per_epoch에 대해서 머리속에서 그림으로 표현이 잘안되서 설명 한번 부탁드립니다.
둘간에 적절한 상관관계가 있는거 같은데요

1	Train 데이터 생성 Suffle = TRUE
2	런닝메이트 조절 결과
3	VGG19, DensNet, ResNet, ResNet V2 정확도 비교
4	VGG19, DensNet, ResNet, ResNet V2 Loss 비교
5	NasNet Large 적용 결과 확인
6	모델 비교 및 선택

20210826 강사님 말씀

모델선정 후 추가 사용방법
1번째 : 앙상블 또는 스택킹
2번째 : 메타러닝

모델선정 방법
- 상황에 맞는 비용 대비 결정

모델선정 후 튜닝 가능 방법
- Fine-Tuning (미세조정)
- 클래시 파이어(식별자)
- 유닛수, 레이어수, 런닝메이트
- L2, drop out

좋은 모델 2개를 앙상블 하면 좋을 꺼 같다.

그래프의 등락이 너무 심함.
- 런닝메이트 감소
- 런닝메이트 플라토 메소드

ResNext 모델의 val_acc가 고정임.
- 더미모델로 학습이 안된것 같음.
- 파인튜닝 안하고 트랜스 러닝만 하고 학습이 안된거 같음.

steps_per_epoch 값에 따라 정확도가 차이가 많이남.
- train 생성시 suffle이 true가 아님
- steps_per_epoch 적절하게 주지않으면 일부분만 학습을할수 있음.
- 이미지수가 적으면 영향도가 큼
- 전체학습데이터를 볼려면 20이므로 steps_per_epoch 를 20 + suffle=True 수행하면 영향도가 적을것임.

20210826 강사님 말씀

steps_per_epoch 값에 따라 정확도가 차이가 많이남.
- train 생성시 suffle이 true가 아님
- steps_per_epoch 적절하게 주지않으면 일부분만 학습을할수 있음.
- 이미지수가 적으면 영향도가 큼
- 전체학습데이터를 볼려면 20이므로 steps_per_epoch 를 20 + suffle=True 수행하면 영향도가 적을것임.

```
train_generator = train_data.flow_from_directory(directory=traindir,target_size=(331,331),class_mode='binary',batch_size=32)
```

flow_from_directory

```
flow_from_directory(directory, target_size=(256, 256), color_mode='rgb', classes=None, class_mode='cate
```

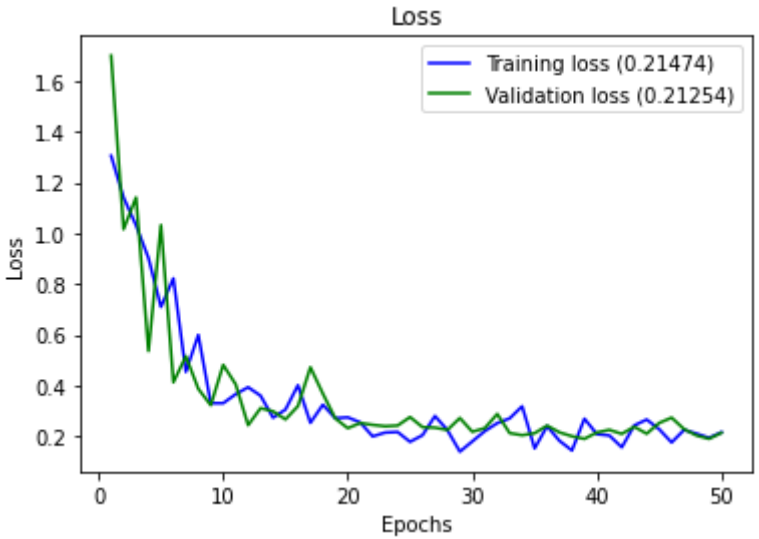
디렉토리에의 경로를 전달받아 증강된 데이터의 배치를 생성합니다.

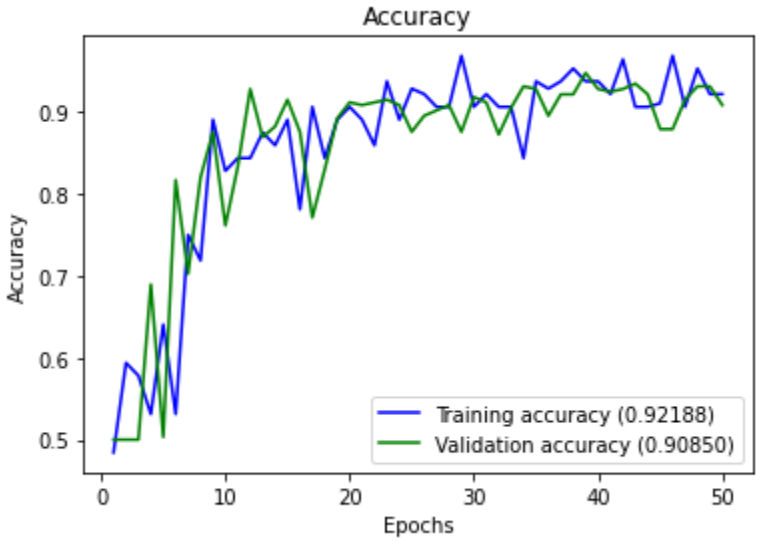
인수

- **directory**: string, 표적 디렉토리에의 경로. 반드시 한 클래스 당 하나의 하위 디렉토리가 있어야 합니다. 각 하위 디렉토리 내에 위치한 어떤 PNG, JPG, BMP, PPM 혹은 TIF 이미지도 생성자에 포함됩니다. 세부사항은 이 스크립트 를 참조하십시오.
- **target_size**: 정수 튜플 `(높이, 넓이)`, 디폴트 값: `(256, 256)`. 모든 이미지의 크기를 재조정할 치수.
- **color_mode**: "grayscale", "rbg", "rgba" 중 하나. 디폴트 값: "rgb". 변환될 이미지가 1개, 3개, 혹은 4개의 채널을 가질지 여부.
- **classes**: 클래스 하위 디렉토리의 선택적 리스트 (예. `['dogs', 'cats']`). 디폴트 값: None. 특별히 값을 지정하지 않으면, 각 하위 디렉토리를 각기 다른 클래스로 대하는 방식으로 클래스의 리스트가 `directory` 내 하위 디렉토리의 이름/구조에서 자동으로 유추됩니다 (그리고 라벨 색인에 대응되는 클래스의 순서는 영숫자 순서를 따릅니다). `class_indices` 속성을 통해서 클래스 이름과 클래스 색인 간 매핑을 담은 딕셔너리를 얻을 수 있습니다.
- **class_mode**: "categorical", "binary", "sparse", "input", 혹은 None 중 하나. 디폴트 값: "categorical". 반환될 라벨 배열의 종류를 결정합니다:
  - "categorical"은 2D형태의 원-핫 인코딩된 라벨입니다,
  - "binary"는 1D 형태의 이진 라벨입니다, "sparse"는 1D 형태의 정수 라벨입니다,
  - "input"은 인풋 이미지와 동일한 이미지입니다 (주로 자동 인코더와 함께 사용합니다).
  - None의 경우, 어떤 라벨도 반환되지 않습니다 (생성자가 이미지 데이터의 배치만 만들기 때문에, `model.predict_generator()`, `model.evaluate_generator()` 등과 함께 사용하는 것이 유용합니다). class_mode가 None일 경우, 제대로 작동하려면 데이터가 `directory` 내 하위 디렉토리에 위치해야 한다는 점을 유의하십시오.
- **batch_size**: 데이터 배치의 크기 (디폴트 값: 32).
- **shuffle**: 데이터를 뒤섞을지 여부 (디폴트 값: 참)
- **seed**: 데이터 셔플링과 변형에 사용할 선택적 난수 시드.

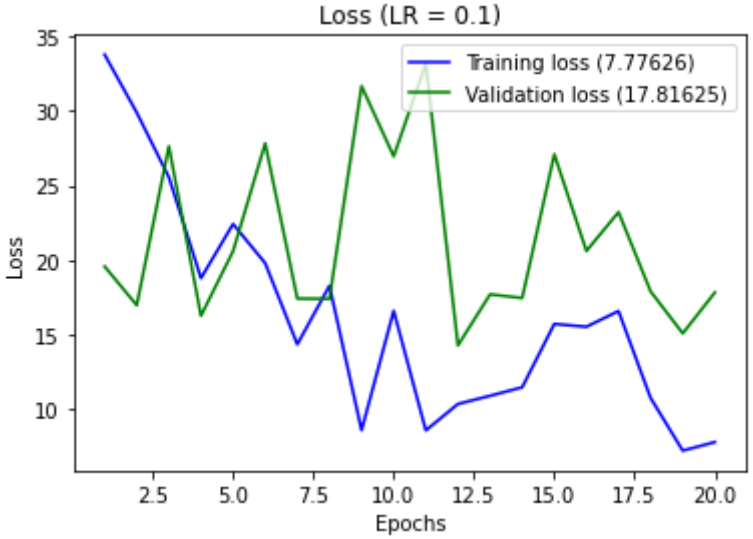
20210826 강사님 말씀 그래프의 등락이 너무 심함.- 런닝메이트 조절

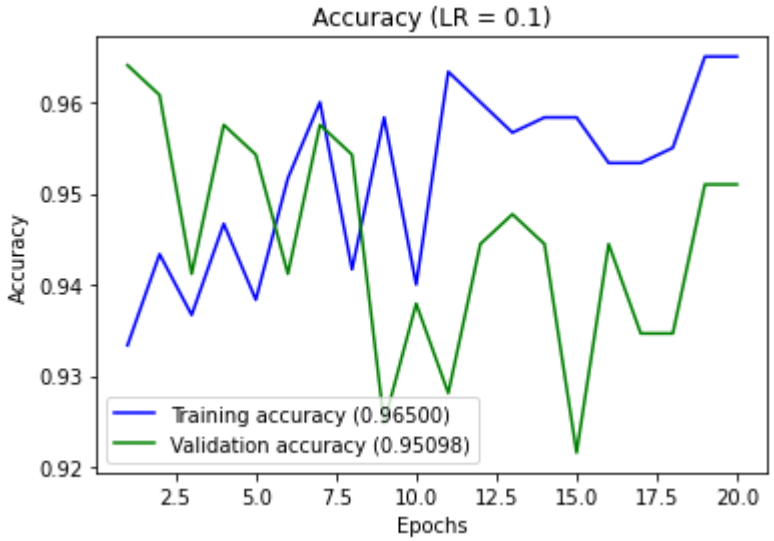
데이터 증식 + 이미지 (256*256) + VGG19 + **기본 LR = 0.001**





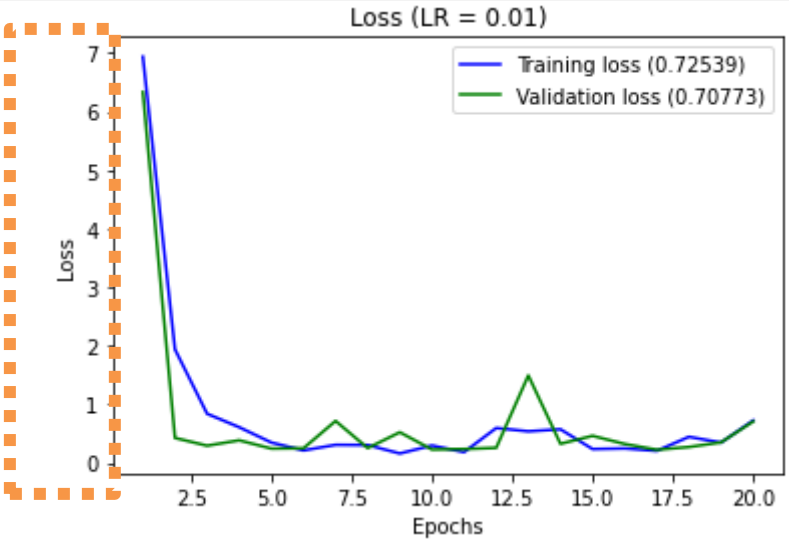
데이터 증식 + 이미지 (256*256) + VGG19 + **LR = 0.1**

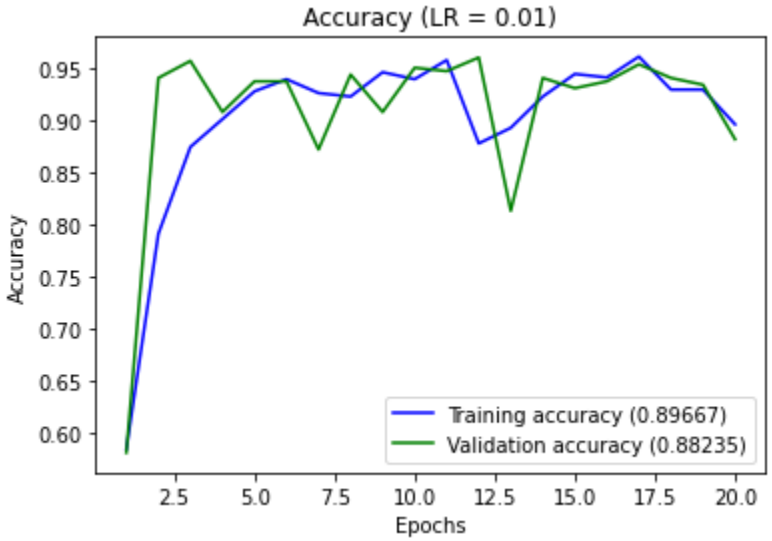




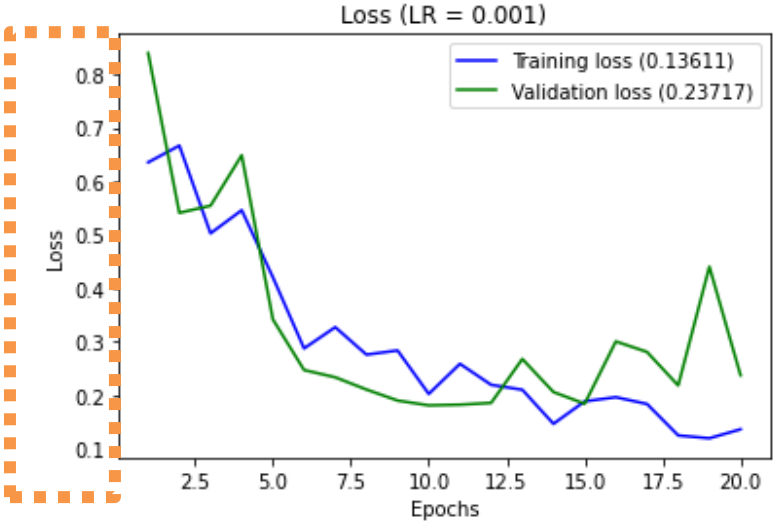
강사님말씀 : 런닝메이트 조절 결과 LR이 0.01일 때보다 0.001인 경우 Loss가 더 적게 나타나는게 보인다..

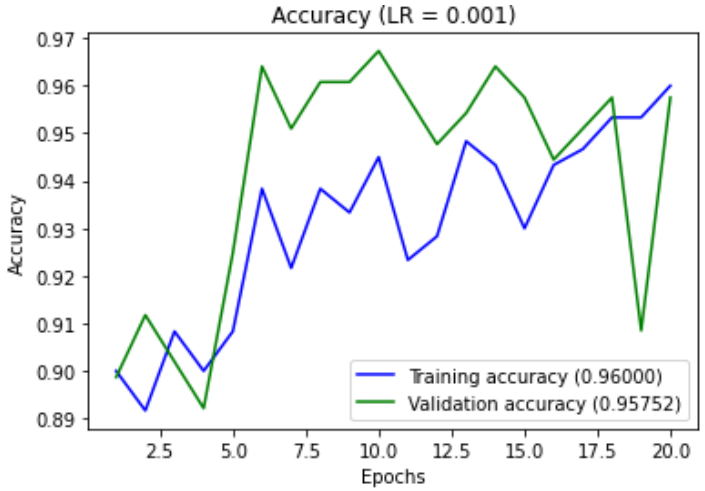
데이터 증식 + 이미지 (256*256) + VGG19 + LR = 0.01



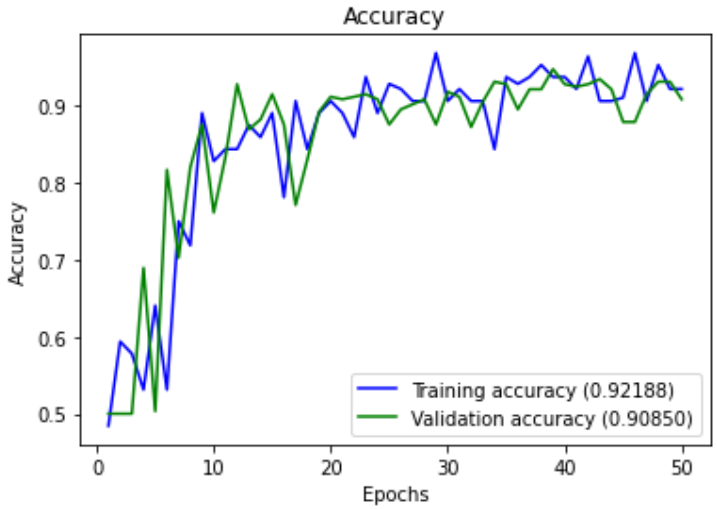


데이터 증식 + 이미지 (256*256) + VGG19 + LR = 0.001

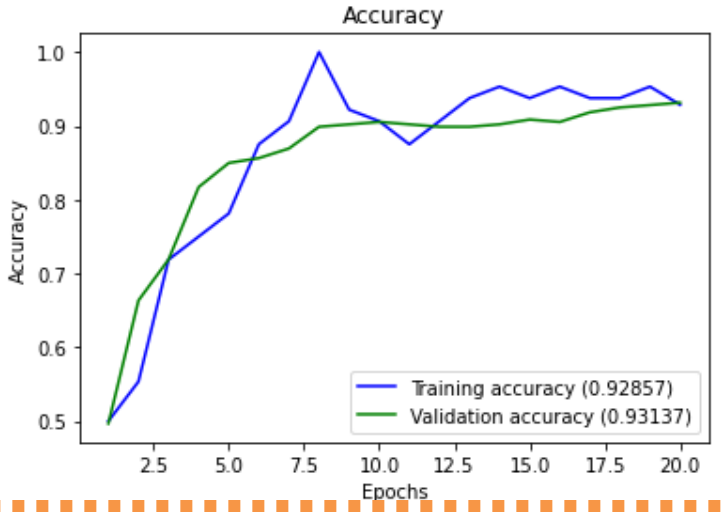




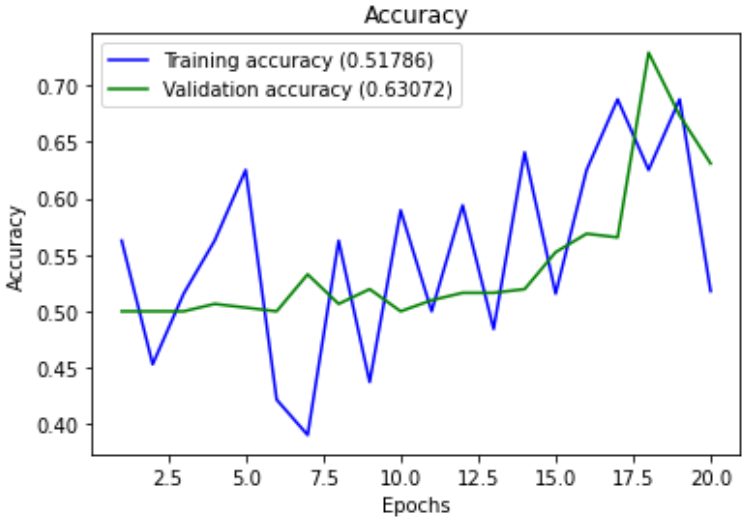
데이터증식+이미지(256*256)+**VGG19**



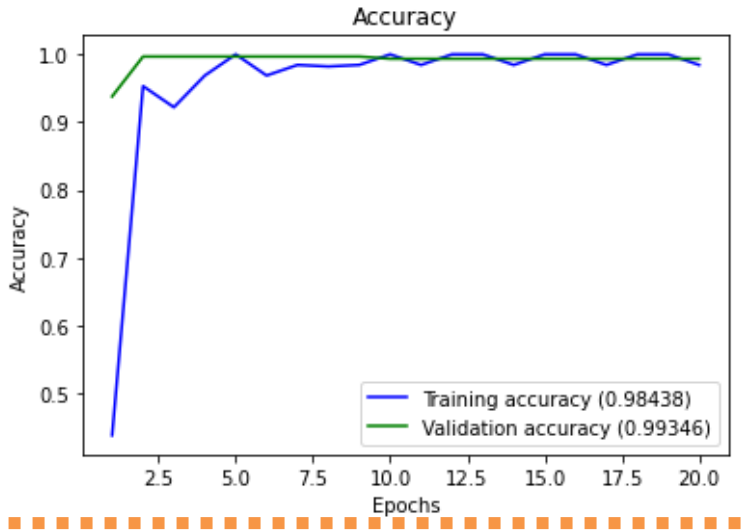
데이터증식+이미지(256*256)+**DenseNet(201)**



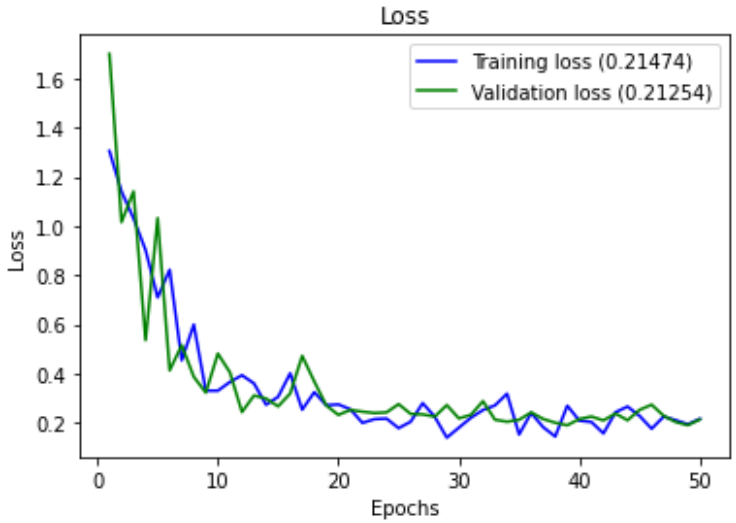
데이터증식+이미지(256*256)+**ResNet(152)**



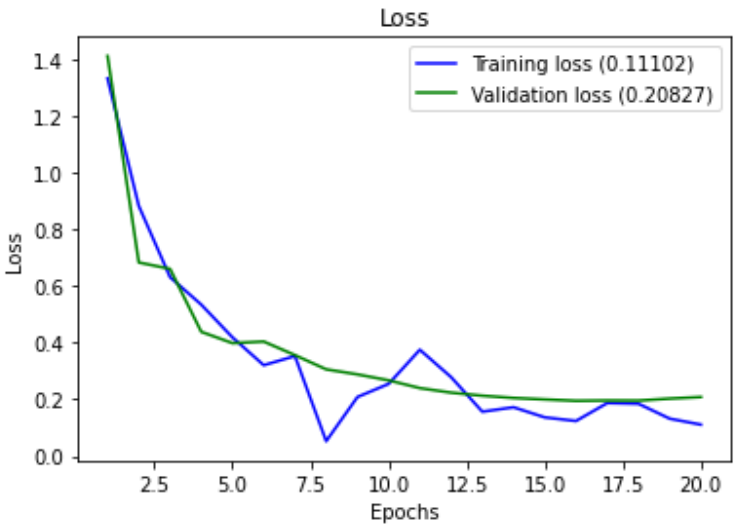
데이터증식+이미지(256*256)+**ResNetv2(152)**



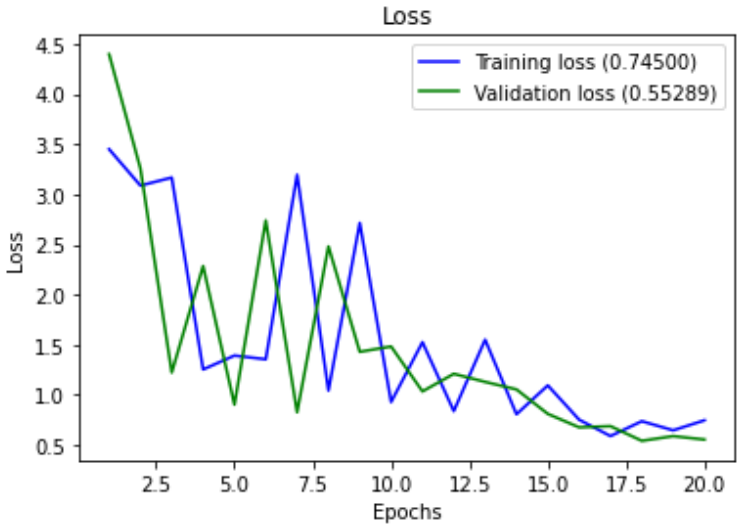
데이터증식+이미지(256*256)+**VGG19**



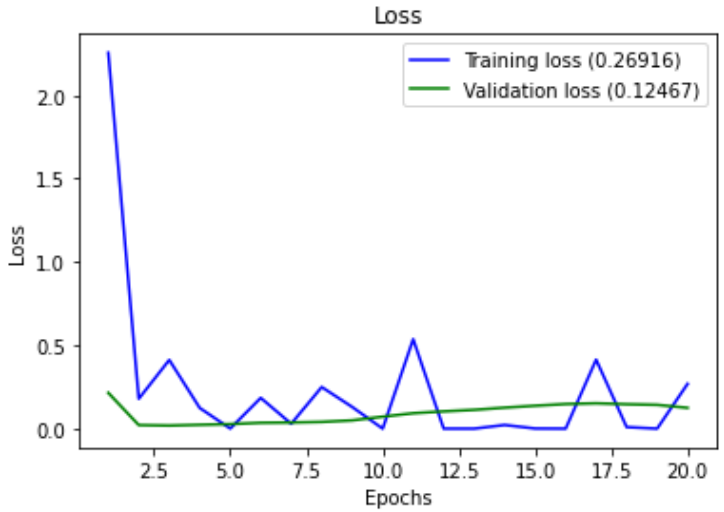
데이터증식+이미지(256*256)+**DenseNet(201)**



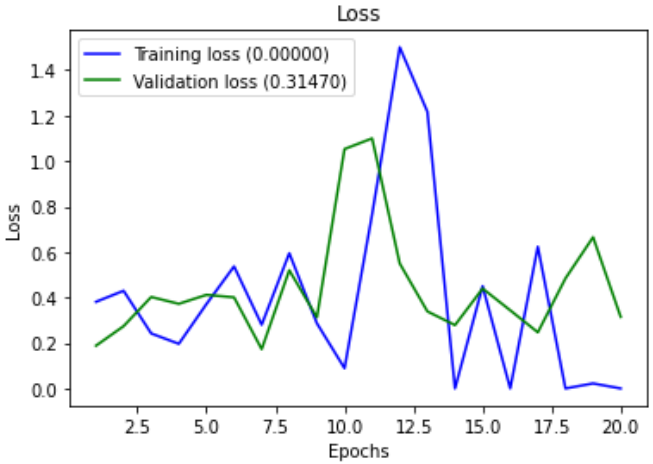
데이터증식+이미지(256*256)+**ResNet(152)**

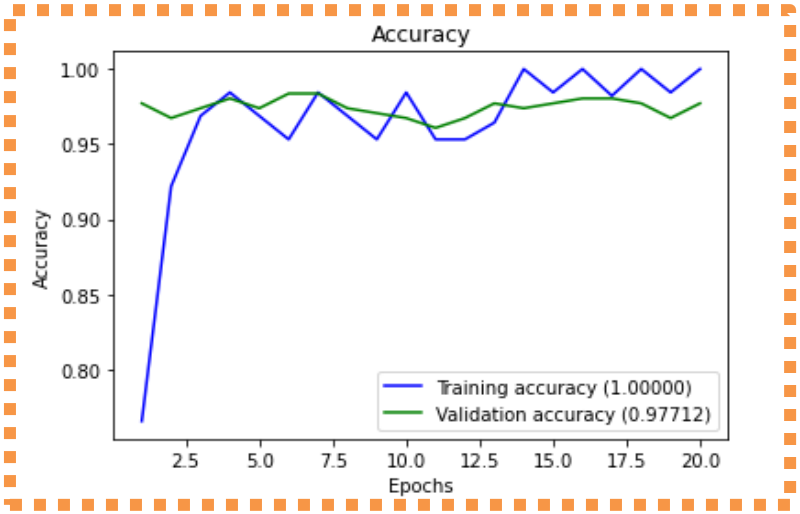


데이터증식+이미지(256*256)+**ResNetv2(152)**

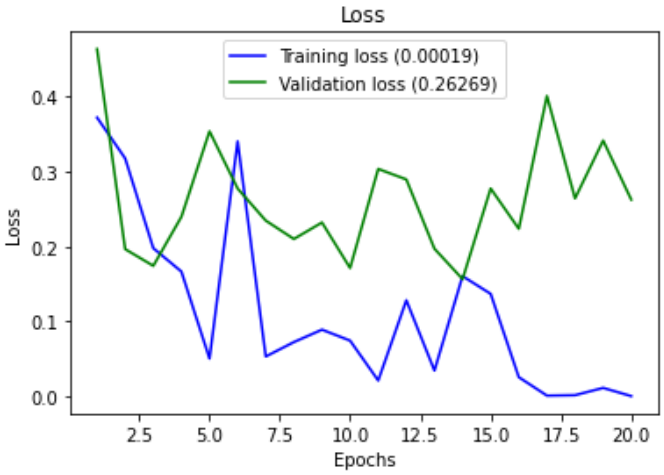


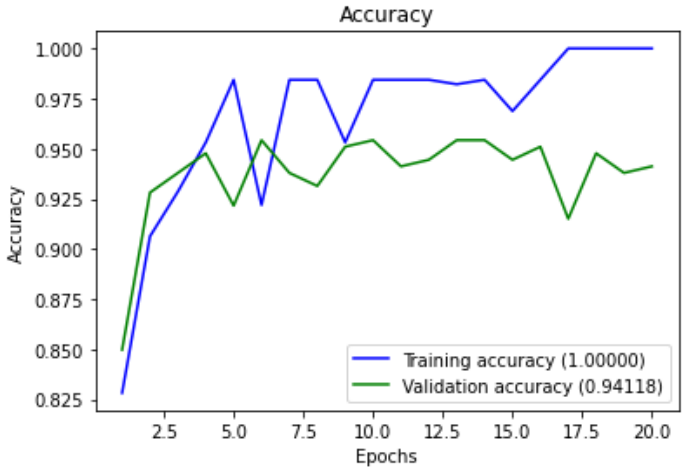
데이터 증식 + 이미지 (256*256) + **NasNet Large**





데이터 증식 + 이미지 (256*256) + **NasNet Mobile**





강사님 말씀 : 표를 볼때 검증을 봐야하며 Loss가 ResNet V2가 더 좋기 때문에 좋은 모델인 거 같음

발표 연도	모델명	특 징	훈련		검증		선정 결과
			Loss	Acc	Loss	Acc	
2014	VGG19	- 3x3의 작은 필터만 사용함. - 19레이어의 깊은 레이어를 사용한다는점	0.184	0.906	0.203	0.937	
2015	ResNet(50)	- residual block + Skip Connection 사용 - 많은 수의 레이어 사용	0.739	0.642	0.407	0.826	
	ResNet(101)		2.705	0.390	0.602	0.722	
	ResNet(152)		0.736	0.625	0.540	0.728	
2016	ResNet V2(50)	-Inception V3모델 + ResNet 장점	0.535	0.968	0.255	0.986	
	ResNet V2(101)		0.231	0.984	0.332	0.993	
	ResNet V2(152)		0.130	0.984	0.049	0.996	√
2017	DensNet(201)	- ResNet과 비슷 - ResNet은 feature map 더하기 - DensNet은 feature map간 Concatenation	0.489	0.998	1.70	0.996	
2017	ShuffleNet	- MobileNet의 개선버전 - Pointwise group Convolution/Channel Shuffle사용	미수행(시간문제)				
2018	ShuffleNet V2	- ShuffleNet 개선버전 (속도향상) - 모델경량화 지표 FLOP가 아닌 MAC을 개선한 모델					
2018	NasNet Large (이미지 사이즈 331, 331)	- 네트워크 구조를 사람이 디자인 해왔음 - Convolution Cell 단위 추정 후 전체 네트워크 구성	1.767 E-09	1.000	0.314	0.977	√
2018	NasNet Mobile (이미지 사이즈 224, 224)	- NasNet 경량화 버전	1.912 E-04	1.000	0.262	0.941	

강사님 말씀 : 오버핏이 발생하므로 튜닝 필요

1. 테스트 데이터 적용 결과 확인
2.
3.
4.
5.
6.

20210827 강사님 말씀

런닝 메이트를 줄여서 LOSS가 줄어드는걸 확인해볼 수 있음

모델선정 시 검증(Validation)에 대한 부분을 보고 결정해야함.

NasNet Large 모델에서 훈련시 ACC가 1.0에 해당하며 오버핏(과대적합)이 발생한 것 같으므로 튜닝을 해볼 수 있음.

20210827 팀원 추가 수행 내용

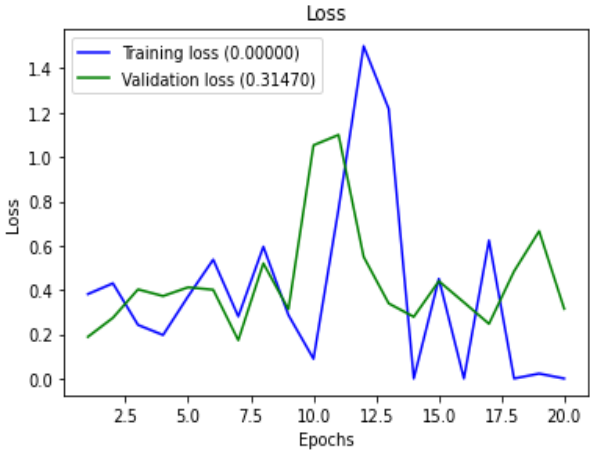
1. 테스트 데이터 적중률 확인
2. NASNET 오버핏으로 인한 튜닝
3. 개인적으로 앙상블 또는 스택킹 수행

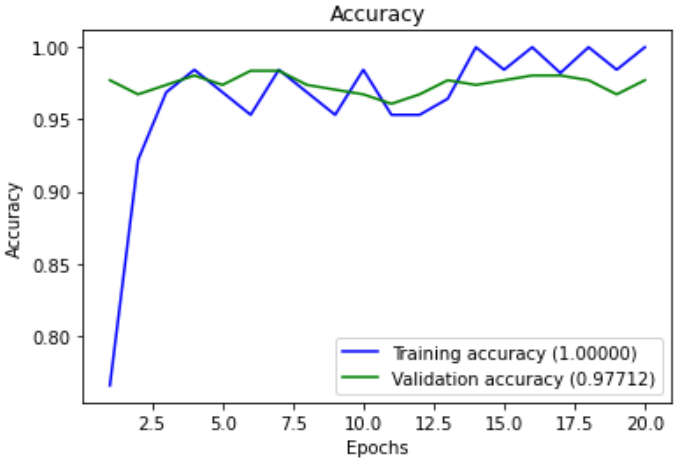
발표 연도	모델명	특 징	Train		Validation		Test	선정 결과
			Loss	Acc	Loss	Acc	Acc	
2014	VGG19	- 3x3의 작은 필터만 사용함. - 19레이어의 깊은 레이어를 사용한다는점	0.184	0.906	0.203	0.937	0.88	
2015	ResNet(50)	- residual block + Skip Connection 사용 - 많은 수의 레이어 사용	0.739	0.642	0.407	0.826		
	ResNet(101)		2.705	0.390	0.602	0.722		
	ResNet(152)		0.736	0.625	0.540	0.728		
2016	ResNet V2(50)	-Inception V3모델 + ResNet 장점	0.535	0.968	0.255	0.986		
	ResNet V2(101)		0.231	0.984	0.332	0.993		
	ResNet V2(152)		0.130	0.984	0.049	0.996	0.99	√
2017	DensNet(201)	- ResNet과 비슷 - ResNet은 feature map 더하기 - DensNet은 feature map간 Concatenation	0.489	0.998	1.70	0.996		
2017	ShuffleNet	- MobileNet의 개선버전 - Pointwise group Convolution/Channel Shuffle사용	미수행(시간문제)					
2018	ShuffleNet V2	- ShuffleNet 개선버전 (속도향상) - 모델경량화 지표 FLOP가 아닌 MAC을 개선한 모델						
2018	NasNet Large (이미지 사이즈 331, 331)	- 네트워크 구조를 사람이 디자인 해왔음 - Convolution Cell 단위 추정 후 전체 네트워크 구성	1.767 E-09	1.000	0.314	0.977	0.97	√
2018	NasNet Mobile (이미지 사이즈 224, 224)	- NasNet 경량화 버전	1.912 E-04	1.000	0.262	0.941	0.94	

▶ 모델명 : NasNet Large (이미지 사이즈 331, 331)

튜닝 파라미터	Train		Validation		Test	적용 결과
	Loss	Acc	Loss	Acc	Acc	
- 기본수행	1.767 E-09	1.000	0.314	0.977	0.97	
- LR=0.0001 적용	0.026	0.984	0.051	0.986	0.98	- 1%의 정확도 증가
- LR=0.0001 적용, 에포크 = 15	0.005	0.996	0.019	0.990	0.99	- 2%의 정확도 증가

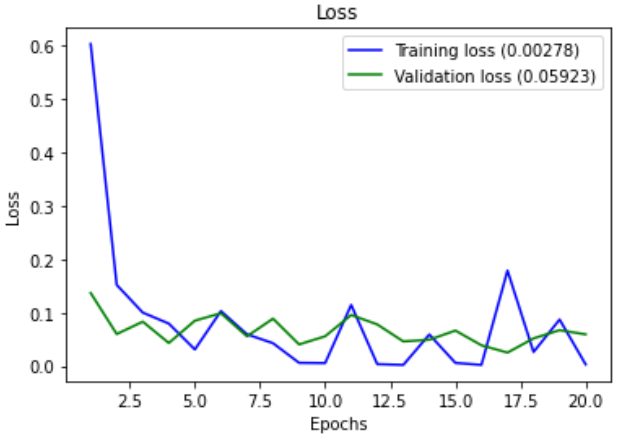
데이터 증식
+
이미지
(331*331)
+
NasNet Large
+
LR=0.001(기본)

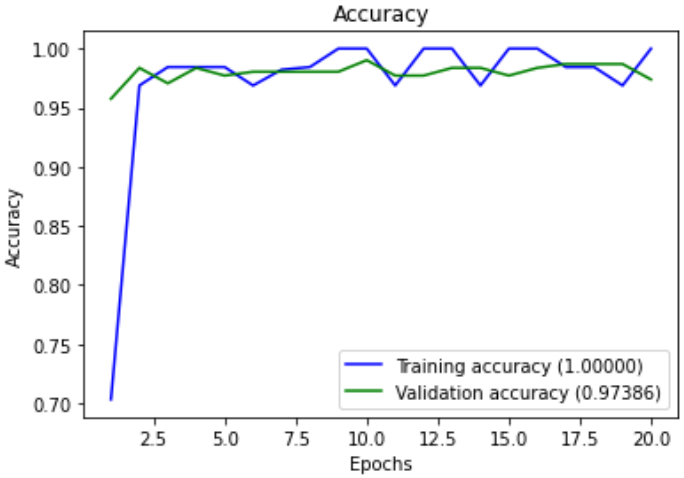




Test ACC (0.97)

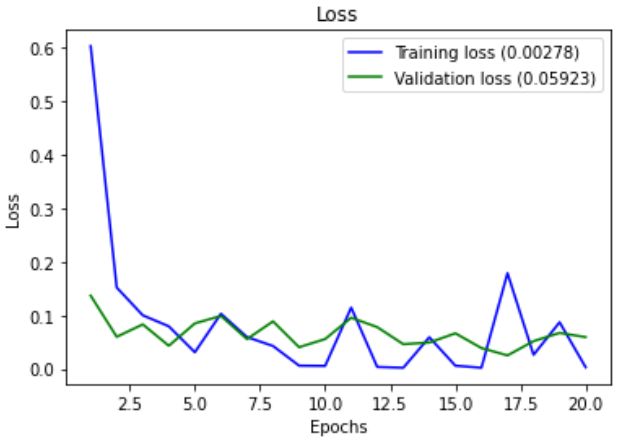
데이터 증식
+
이미지
(331*331)
+
NasNet Large
+
LR=0.0001

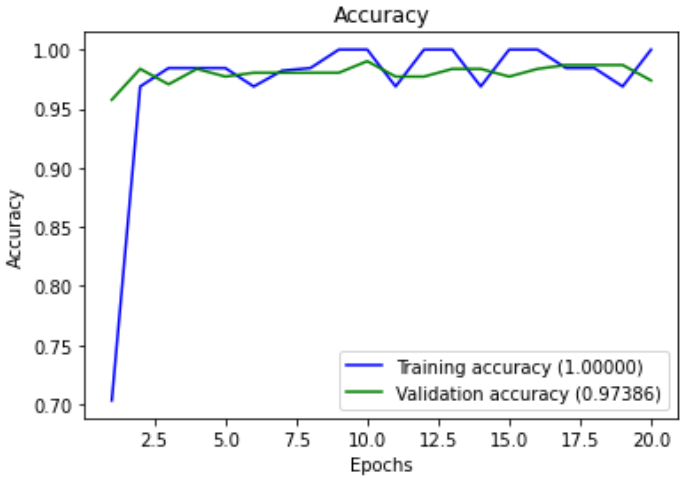




Test ACC (0.98)

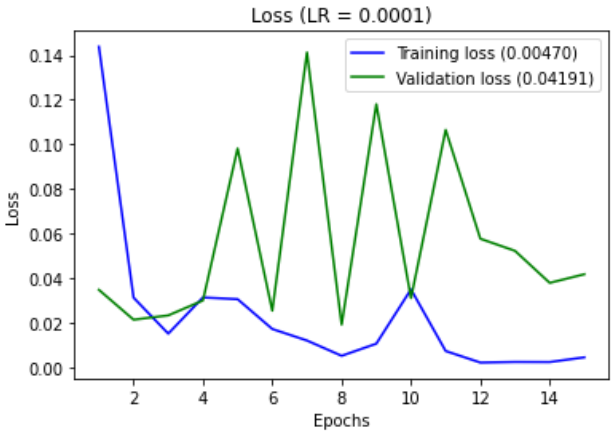
데이터 증식
+
이미지
(331*331)
+
NasNet Large
+
LR=0.0001
+
에포크 :20

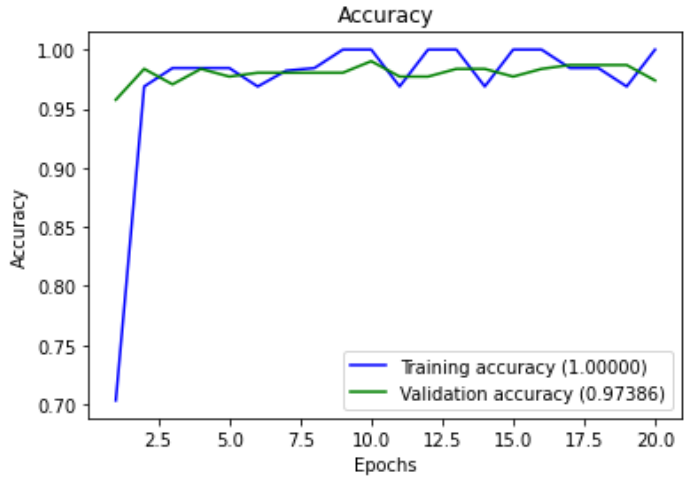




Test ACC (0.98)

데이터 증식
+
이미지
(331*331)
+
NasNet Large
+
LR=0.0001
+
에포크 : 15





Test ACC (0.99)

오버피팅을 제거하기 위해서 LR과 에포크를 줄임