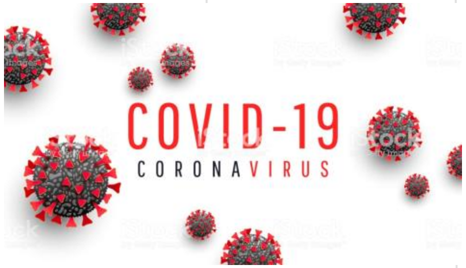
Classification 조 미니프로젝트 발표

COVID-19 CT 사진 분류하기

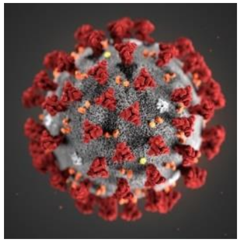
<발표자: 13기 고영희, 14기 홍승아 >

목 차

1. 주제 선정



각 대학의 수업도 비 대면 수업으로 진행되고 있고,
우리의 보아즈 또한 온라인으로 진행되고 있습니다.



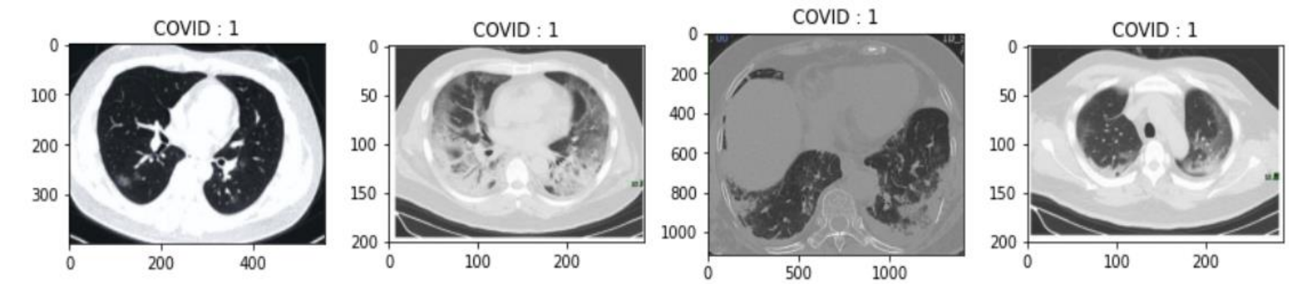
원인은 바로 전염력이 강력한 코로나19 바이러스 때문입니다.

5월 말이면 종식될 것 같았던 코로나 19는 "이태원 클럽 감염" 으로 인해
재확산되고 있습니다.

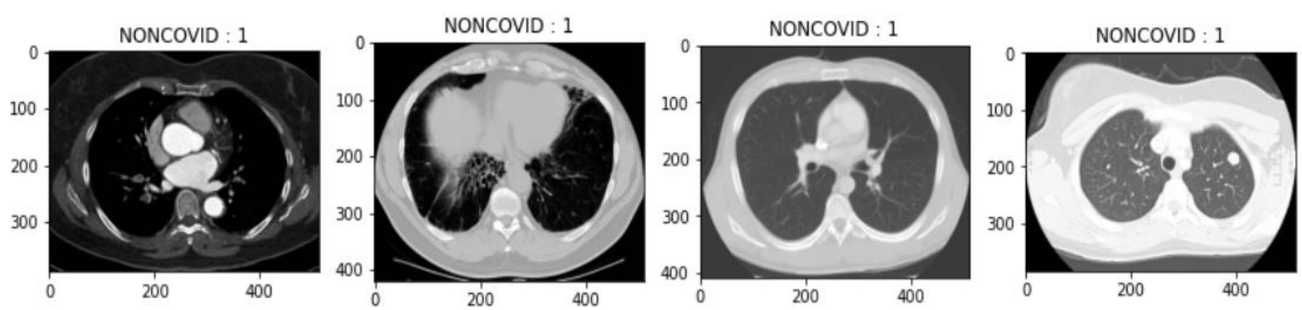
이에 저희 조는 코로나-19 확진자를 좀 더 빠르게 판별할 수 있다면
코로나 종식 시기를 좀 더 앞당길 수 있지 않을까?라는 생각으로 주제를 선정하게 되었습니다.

흉부 CT를 데이터로 사용하면서 감염된 경우와 정상여부를 분류하는 프로젝트를 진행하였습니다..

모델 적용 전 시각화



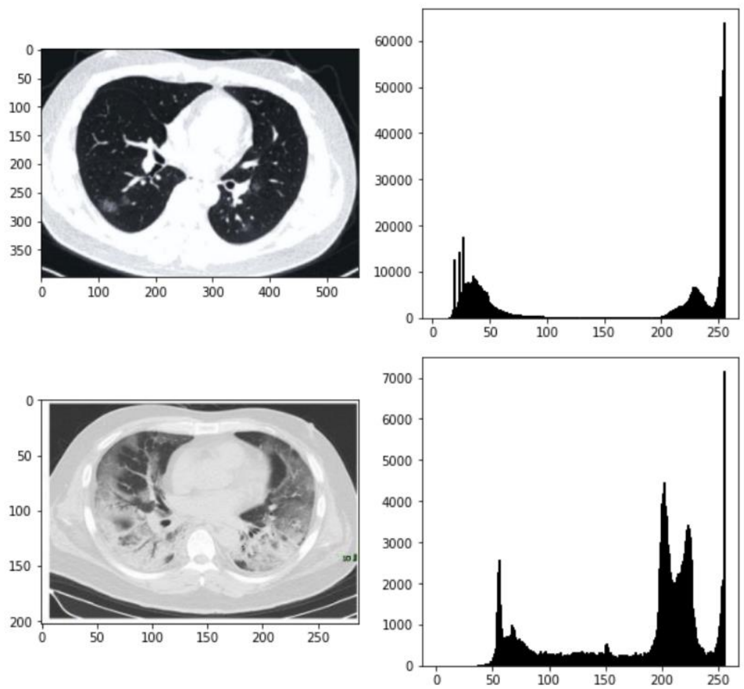
Covid 이미지 시각화



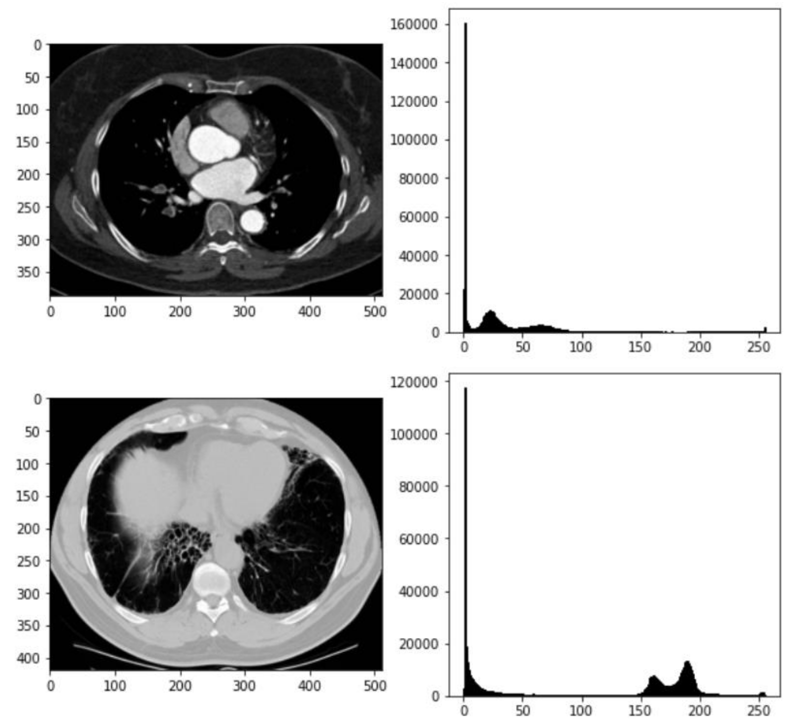
Non covid 이미지 시각화

Image histogram

디지털 이미지의 색조 분포를 그래픽으로 나타내는 히스토그램 각 색조 값의 픽셀 수 표시. 전체 색조 분포 판단 가능



Covid image histogram



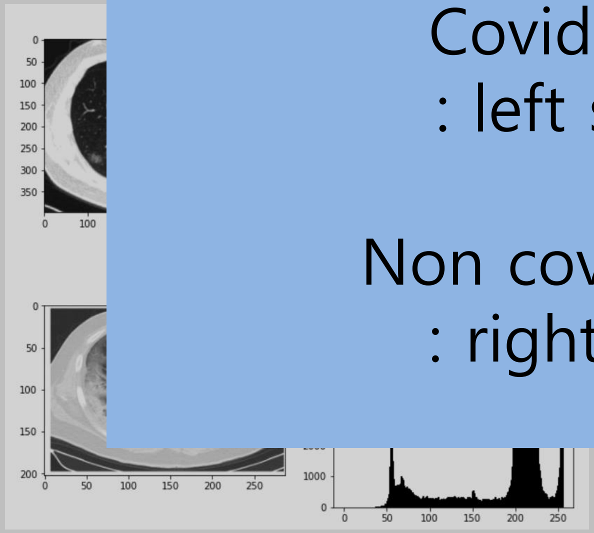
Non covid image histogram

Image histogram

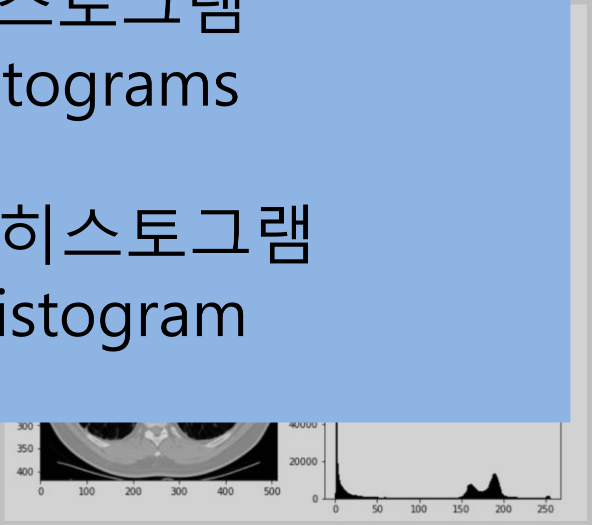
디지털 이미지의 색조 분포를 그래픽으로 나타내는 히스토그램 각 색조 값의 픽셀 수 표시. 전체 색조 분포 판단 가능

Covid 이미지 히스토그램
: left skewed histograms

Non covid 이미지 히스토그램
: right skewed histogram



Covid image histogram



Non covid image histogram

VGG Net

2. Vgg Net Architecture

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					

2. VGG 코드 : 모델 생성

```python
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras.models import Model
from keras import models
from keras import layers
from keras import optimizers
import keras.backend as K

K.clear_session() # 새로운 세션으로 시작

from keras.applications import VGG16
# 모델 불러오기
conv_layers = VGG16(weights='imagenet', include_top=False, input_shape=(minh,minv,3))
conv_layers.summary()

# Convolution Layer를 학습되지 않도록 고정
for layer in conv_layers.layers:
    layer.trainable = False


# 새로운 모델 생성하기
model = models.Sequential()

# VGG16모델의 Convolution Layer를 추가
model.add(conv_layers)

# 모델의 Fully Connected 부분을 재구성
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(2, activation='softmax'))
```

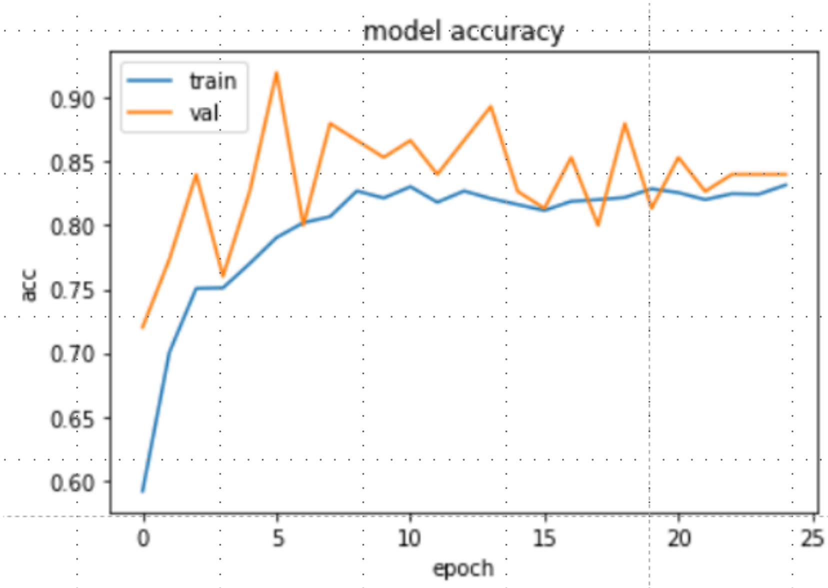
```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

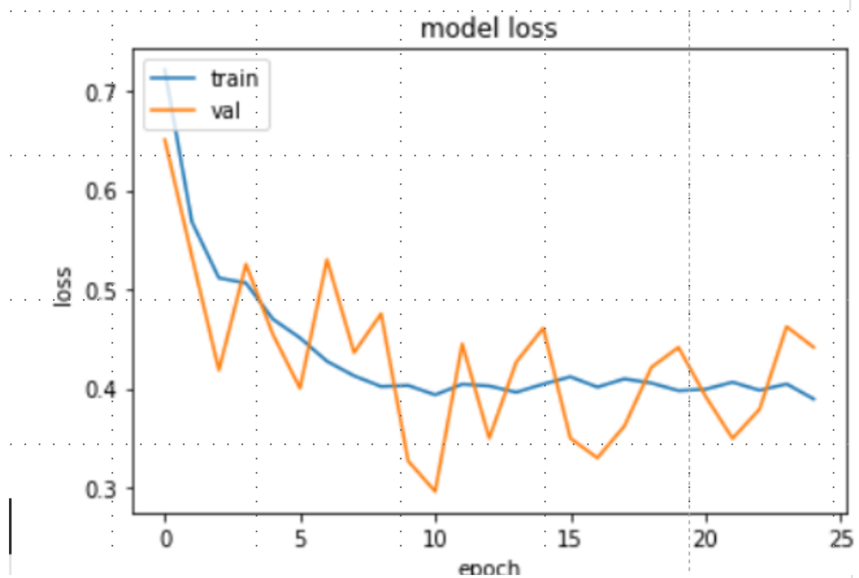
2. VGG 코드: 모델 fitting

```
history = model.fit_generator(train_generator,
                              validation_data=validation_generator,
                              epochs=25,
                              steps_per_epoch=train_x.shape[0]/8,
                              callbacks=[custom_callback])
```

```
Epoch 1/25
75/74 [==============================] - 37s 497ms/step - loss: 0.7213 - acc: 0.5914 - val_loss: 0.6509 - val_acc: 0.7200
Epoch 2/25
75/74 [==============================] - 34s 449ms/step - loss: 0.5686 - acc: 0.7003 - val_loss: 0.5334 - val_acc: 0.7733
Epoch 3/25
75/74 [==============================] - 33s 444ms/step - loss: 0.5111 - acc: 0.7504 - val_loss: 0.4185 - val_acc: 0.8400
Epoch 4/25
75/74 [==============================] - 33s 443ms/step - loss: 0.5056 - acc: 0.7509 - val_loss: 0.5253 - val_acc: 0.7600
Epoch 5/25
75/74 [==============================] - 33s 441ms/step - loss: 0.4696 - acc: 0.7700 - val_loss: 0.4544 - val_acc: 0.8267
Epoch 6/25
Learning rate reduced to 0.0001
75/74 [==============================] - 33s 441ms/step - loss: 0.4505 - acc: 0.7904 - val_loss: 0.4003 - val_acc: 0.9200
```

2. VGG


model accuracy
train
val
0.90
0.85
0.80
0.75
0.70
0.65
0.60
acc
0
5
10
15
20
25
epoch


model loss
train
val
0.7
0.6
0.5
0.4
0.3
loss
0
5
10
15
20
25
epoch

2. VGG : accuracy 확인

```
print("training_accuracy", history.history['acc'][-1])
print("validation_accuracy", history.history['val_acc'][-1])
```

```
training_accuracy 0.8316327
validation_accuracy 0.8399999737739563
```

```
#학습시킨 모델을 test data에 적용하여 일반화가 성공적으로 되었는지 확인합니다.
test_loss, test_accuracy = ₩
  model.evaluate(test_generator)
print('Test loss: %.4f accuracy: %.4f' % (test_loss, test_accuracy))
```

```
3/3 [==============================] - 1s 272ms/step
Test loss: 0.4391 accuracy: 0.8667
```

Dense Net

3. Dense Net Architecture

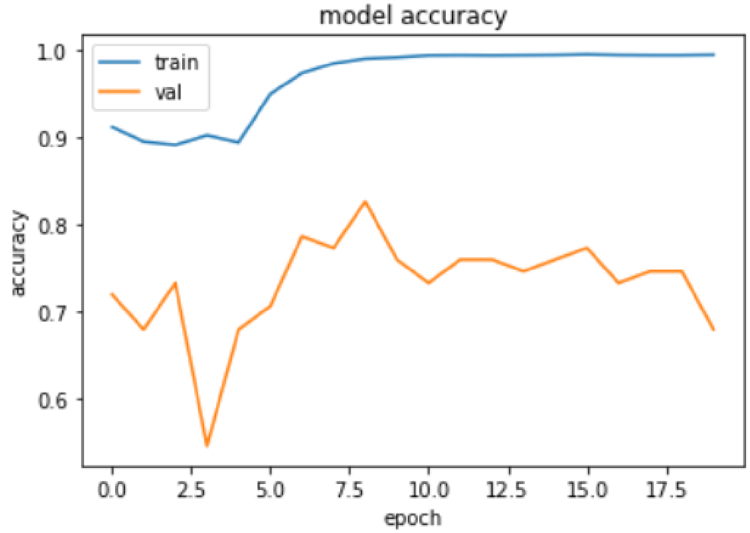
Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

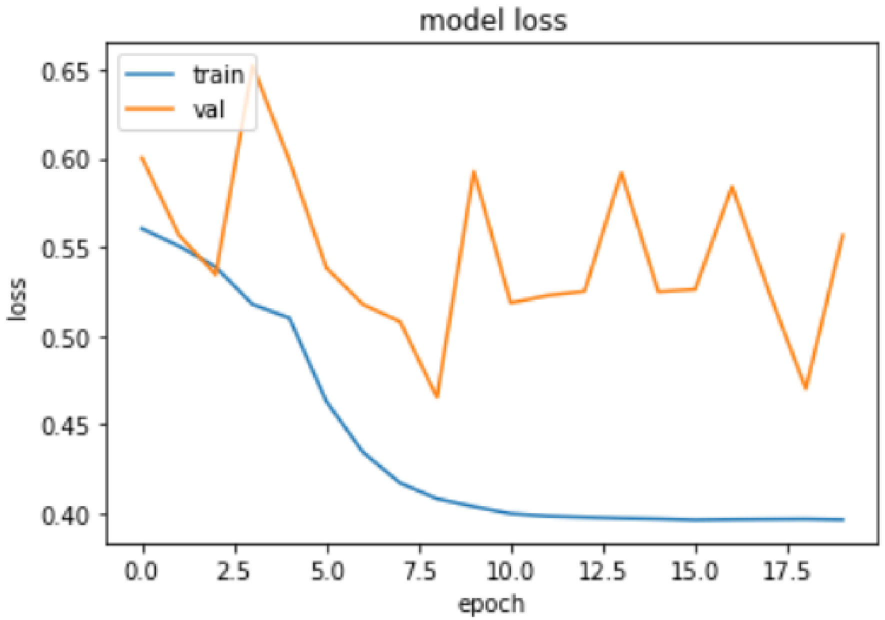
3. Dense Net : 모델 fitting

```
history = model.fit_generator(train_generator,
                              validation_data=validation_generator,
                              epochs=20,
                              steps_per_epoch=train_x.shape[0]/2,
                              callbacks=[custom_callback])
```

```
Epoch 1/20
298/298 [==============================] - 88s 296ms/step - loss: 0.5599 - accuracy: 0.9118 - val_loss: 0.6002 - val_accuracy: 0.7200
Epoch 2/20
298/298 [==============================] - 88s 297ms/step - loss: 0.5496 - accuracy: 0.8953 - val_loss: 0.5568 - val_accuracy: 0.6800
Epoch 3/20
298/298 [==============================] - 88s 297ms/step - loss: 0.5380 - accuracy: 0.8915 - val_loss: 0.5345 - val_accuracy: 0.7333
Epoch 4/20
298/298 [==============================] - 89s 298ms/step - loss: 0.5168 - accuracy: 0.9024 - val_loss: 0.6523 - val_accuracy: 0.5467
Epoch 5/20
298/298 [==============================] - 88s 297ms/step - loss: 0.5096 - accuracy: 0.8943 - val_loss: 0.5986 - val_accuracy: 0.6800
```

3. Dense Net : 정확도 확인





3. Dense Net

```
print("training_accuracy", history.history['accuracy'][-1])
print("validation_accuracy", history.history['val_accuracy'][-1])
```

```
training_accuracy 0.994649
validation_accuracy 0.6800000071525574
```

```
#학습시킨 모델을 test data에 적용하여 일반화가 성공적으로 되었는지 확인합니다.
test_loss, test_accuracy = ₩
    model.evaluate(test_generator)
print('Test loss: %.4f accuracy: %.4f' % (test_loss, test_accuracy))
```

```
5/5 [==============================] - 0s 73ms/step
Test loss: 0.3915 accuracy: 0.8133
```

Wide ResNet

4. Wide ResNet Architecture

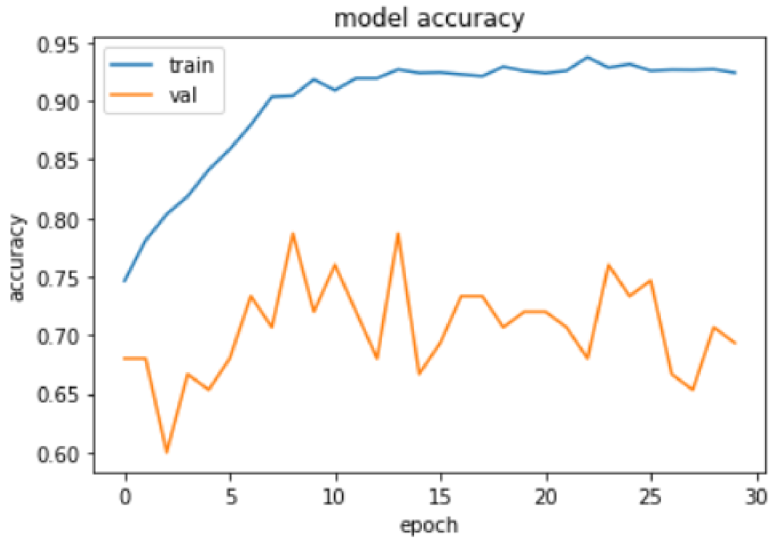
group name	output size	block type = $B(3,3)$
conv1	32×32	$[3\times3, 16]$
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

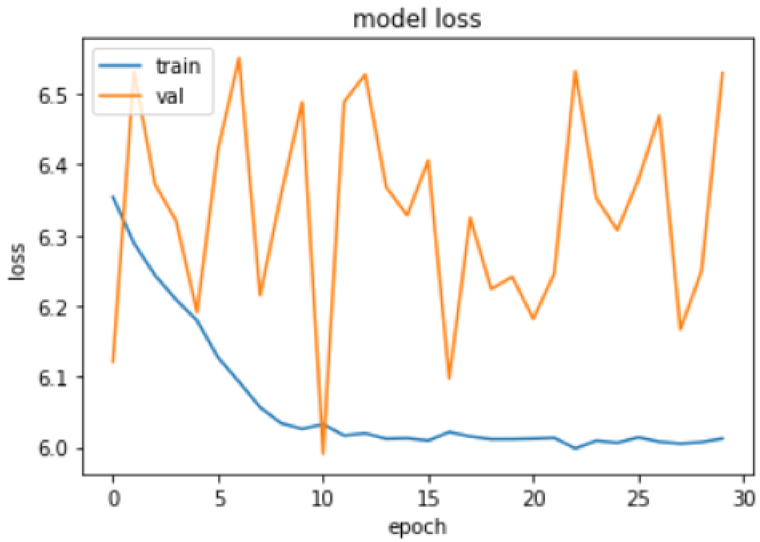
4. Wide ResNet

```
history = model.fit_generator(train_generator,
                    validation_data=validation_generator,
                    epochs=20,
                    steps_per_epoch=train_x.shape[0]/2,
                    callbacks=[custom_callback])
```

```
Epoch 1/20
298/298 [==============================] - 215s 722ms/step - loss: 6.7475 - accuracy: 0.6156 - val_loss: 6.7379 - val_accuracy: 0.5600
Epoch 2/20
298/298 [==============================] - 205s 689ms/step - loss: 6.4806 - accuracy: 0.6729 - val_loss: 6.4781 - val_accuracy: 0.6667
Epoch 3/20
298/298 [==============================] - 205s 687ms/step - loss: 6.4184 - accuracy: 0.6976 - val_loss: 6.7803 - val_accuracy: 0.6533
Epoch 4/20
298/298 [==============================] - 205s 687ms/step - loss: 6.3787 - accuracy: 0.7177 - val_loss: 6.5269 - val_accuracy: 0.6800
Epoch 5/20
298/298 [==============================] - 205s 688ms/step - loss: 6.3448 - accuracy: 0.7419 - val_loss: 6.3925 - val_accuracy: 0.7200
```

4. Wide ResNet





4. Wide ResNet

```
print("training_accuracy", history.history['accuracy'][-1])
print("validation_accuracy", history.history['val_accuracy'][-1])
```

```
training_accuracy 0.9240154
validation_accuracy 0.6933333277702332
```

```
#학습시킨 모델을 test data에 적용하여 일반화가 성공적으로 되었는지 확인합니다.
test_loss, test_accuracy = ₩
  model.evaluate(test_generator)
print('Test loss: %.4f accuracy: %.4f' % (test_loss, test_accuracy))
```

```
5/5 [==============================] - 0s 57ms/step
Test loss: 6.2340 accuracy: 0.8133
```

Efficient Net

5. Efficient Net Architecture

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

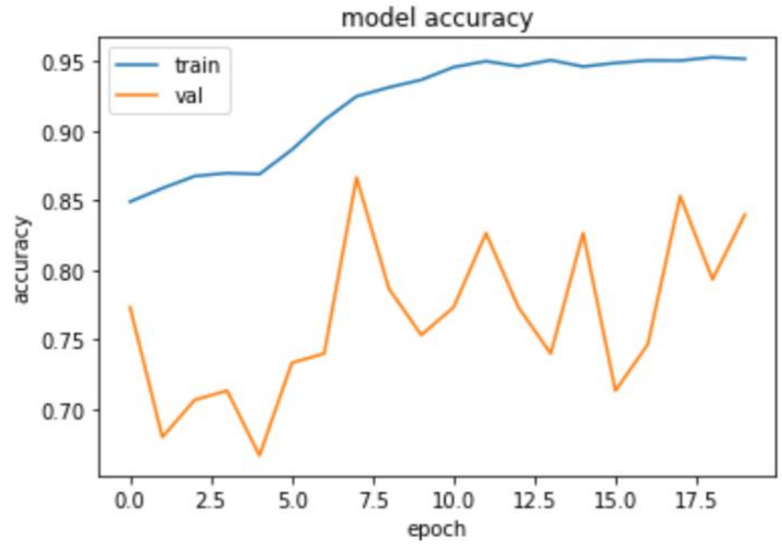
Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	28×28	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

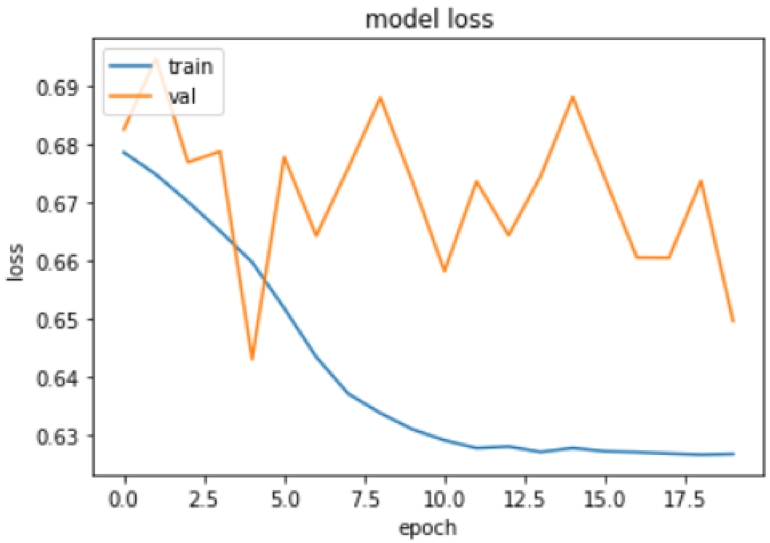
5. Efficient Net

```
history = model.fit_generator(train_generator,
                    validation_data=validation_generator,
                    epochs=20,
                    steps_per_epoch=train_x.shape[0]/2,
                    callbacks=[custom_callback])
```

```
Epoch 1/20
298/298 [==============================] - 52s 174ms/step - loss: 0.6786 - accuracy: 0.8492 - val_loss: 0.6826 - val_accuracy: 0.7733
Epoch 2/20
298/298 [==============================] - 52s 173ms/step - loss: 0.6748 - accuracy: 0.8589 - val_loss: 0.6948 - val_accuracy: 0.6800
Epoch 3/20
298/298 [==============================] - 52s 173ms/step - loss: 0.6700 - accuracy: 0.8676 - val_loss: 0.6770 - val_accuracy: 0.7067
Epoch 4/20
298/298 [==============================] - 51s 172ms/step - loss: 0.6651 - accuracy: 0.8698 - val_loss: 0.6788 - val_accuracy: 0.7133
Epoch 5/20
298/298 [==============================] - 52s 173ms/step - loss: 0.6595 - accuracy: 0.8690 - val_loss: 0.6431 - val_accuracy: 0.6667
```

5. Efficient Net





5. Efficient Net

```
print("training_accuracy", history.history['accuracy'][-1])
print("validation_accuracy", history.history['val_accuracy'][-1])
```

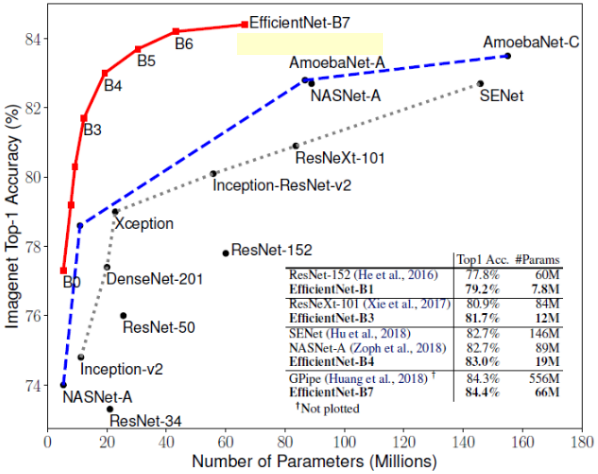
```
training_accuracy 0.95194775
validation_accuracy 0.8399999737739563
```

```
#학습시킨 모델을 test data에 적용하여 일반화가 성공적으로 되었는지 확인합니다.
test_loss, test_accuracy = ₩
  model.evaluate(test_generator)
print('Test loss: %.4f accuracy: %.4f' % (test_loss, test_accuracy))
```

```
5/5 [==============================] - 0s 37ms/step
Test loss: 0.6224 accuracy: 0.8200
```

6. 모델 성능 비교

VGG Net (0.8667) > Efficient Net (0.82) > Dense Net >= Wide ResNet (0.8133)



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k=12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k=12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k=24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k=12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k=24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k=40$)	190	25.6M	-	**3.46**	-	**17.18**	-

BOAZ

감사합니다