**Cat dog data에서 초기 모델을 단순 Convolution2D와 MaxPooling을 반복하여**

**다음과 같이 모델링 하였다.**

Model: "sequential\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_2 (Conv2D) (None, 148, 148, 32) 896

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 74, 74, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 72, 72, 64) 18496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_3 (MaxPooling2 (None, 36, 36, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 34, 34, 128) 73856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_4 (MaxPooling2 (None, 17, 17, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 36992) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 512) 18940416

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 1) 513

=================================================================

Total params: 19,034,177

Trainable params: 19,034,177

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**이때, 초기 모델을 cat dog 데이터에 학습을 시키면 val loss가 불안정하고, accuracy가 train data에 비해 현저히 떨어지는 것을 알 수 있다.**

Epoch 7/10

100/100 [==============================] - 95s 946ms/step - loss: 0.3275 - acc: 0.8605 - val\_loss: 0.6151 - val\_acc: 0.7080

Epoch 8/10

100/100 [==============================] - 95s 947ms/step - loss: 0.2961 - acc: 0.8795 - val\_loss: 0.6176 - val\_acc: 0.6980

Epoch 9/10

100/100 [==============================] - 93s 931ms/step - loss: 0.2553 - acc: 0.8990 - val\_loss: 0.6272 - val\_acc: 0.7070

Epoch 10/10

100/100 [==============================] - 92s 917ms/step - loss: 0.2199 - acc: 0.9180 - val\_loss: 0.6349 - val\_acc: 0.7080

**이를 test data set에 evaluate 시키면, accuracy가 71%가 나오고, 학습 데이터의 accuracy 91%와 비교하면 심각하게 overfitting 되어있다는 것을 알 수 있다.**

50/50 [==============================] - 13s 259ms/step - loss: 0.6205 - acc: 0.7100

Out[50]:[0.6205025911331177, 0.7099999785423279]

model\_aug=Sequential()

model\_aug.add(vgg\_base)

**VGG16 모형을 이용한 이전학습을 통해 분석성능을 높이며 과대적합을 해결하고자 한다.**

model\_aug.add(Flatten())

model\_aug.add(Dropout(0.5))

**model의 모수를 확인해보면, parameter의 수가 1400만개를 넘어서, 과대적합이 발생할 수밖에 없다. 따라서 dropout을 통해 모수를 규제화 해준다.**

model\_aug.add(Dense(256,activation='relu'))

model\_aug.add(Dense(1,activation='sigmoid',kernel\_regularizer=l1(0.1)))

**출력층의 모수는 257개인데 비해, 출력층 바로 직전 은닉층의 모수 개수가 200만개이다. 따라서 bottleneck 현상이 발생할 것이고, regularization을 통해 significant 하지 않은 모수를 줄여줄 필요가 있다.**

vgg\_base.trainable=True

trainable\_layer=True

for layer in vgg\_base.layers:

if layer.name=='block5\_conv1':

trainable\_layer=True

if trainable\_layer:

layer.trainable=True

else:

layer.trainable=False

**vgg 마지막 block5부터 모수를 trainable로 바꿔서 fine tuning을 적용해 정확도를 높이고자 한다.**

train\_datagen=ImageDataGenerator(rescale=1./255,rotation\_range=40,width\_shift\_range=0.2,

height\_shift\_range=0.2,shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,fill\_mode='nearest')

**자료 증대를 통해 noise를 키워 overfitting을 해결하고자 한다.**

….

model\_aug.compile(optimizer=optimizers.RMSprop(lr=2e-5),loss='binary\_crossentropy',metrics=['acc'])

model\_aug\_result=model\_aug.fit\_generator(train\_generator,steps\_per\_epoch=100,epochs=30,validation\_data=validation\_generator,validation\_steps=50)

Epoch 25/30

100/100 [==============================] - 29s 286ms/step - loss: 0.5080 - acc: 0.9880 - val\_loss: 0.6296 - val\_acc: 0.9610

Epoch 26/30

100/100 [==============================] - 29s 285ms/step - loss: 0.4798 - acc: 0.9915 - val\_loss: 0.6378 - val\_acc: 0.9490

Epoch 27/30

100/100 [==============================] - 29s 286ms/step - loss: 0.4679 - acc: 0.9875 - val\_loss: 0.5927 - val\_acc: 0.9710

Epoch 28/30

100/100 [==============================] - 28s 283ms/step - loss: 0.4553 - acc: 0.9870 - val\_loss: 0.5391 - val\_acc: 0.9600

Epoch 29/30

100/100 [==============================] - 28s 283ms/step - loss: 0.4238 - acc: 0.9910 - val\_loss: 0.5234 - val\_acc: 0.9670

Epoch 30/30

100/100 [==============================] - 29s 287ms/step - loss: 0.4043 - acc: 0.9905 - val\_loss: 0.5328 - val\_acc: 0.9670

**Training dataset의 acc은 99퍼센트, validation data의 acc은 96.7%정도로 나타났다. Validation loss가 지속적으로 감소하고, validation accuracy도 수렴하는 것으로 보아 과대적합 되지 않은 것으로 보인다.**

test\_datagen=ImageDataGenerator(rescale=1./255)

test\_generator=test\_datagen.flow\_from\_directory(test\_dir,target\_size=(150,150),batch\_size=20,class\_mode='binary')

loss,acc=model\_aug.evaluate\_generator(test\_generator,steps=50)

print(loss)

0.6200913190841675

print(acc)

0.9629999995231628

**Test data에 대해 적용한 결과 accuracy가 96.2%로 학습 데이터와 거의 일치하는 것을 볼 수 있다.**