빅데이터 기반 AI 응용 솔루션 개발자 전문과정

교과목명: 딥러닝알고리즘 구현

- 평가일: 2022. 11. 22
- 성명: 김영선
- 점수:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

Q1. 사람이 문장을 읽는 것처럼 이전에 나온 것을 기억하면서 단어별로 또는 한눈에 들어오는 만큼씩 처리하여 문장에 있은 의미를 자연스럽게 표현하려는 목적으로 과거 정보를 사용하고 새롭게 얻은 정보를 계속 업데이트 하는 방식이 순환 신경망(RNN) 이다. SimpleRNN을 활용하여 IMDB 영화 리뷰 데이터에 대하여 아래 사항을 수행하세요.

- 데이터 전처리 : max_features 10000, maxlen = 500, batch_size 32
- 케라스를 사용하여 입력 시퀀스에 대한 마지막 출력만 반환하는 방식으로 모델링.(embedding 층 입력 (max_features, 32))
- 학습 및 검증 옵션 : epochs 10, batch_size 128, 검증 데이터 20% ※ 학습시간 20분
- 훈련과 검증의 손실과 정확도를 그래프로 표현
- 검증 정확도를 확인하고 동 사례에 SimpleRNN 모델의 적합 여부 및 개선 방안에 대하여 기술하세요.

```
from keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
```

```
max_features = 10000
maxlen = 500
batch_size = 32

(input_train,y_train),(input_test,y_test) = imdb.load_data(num_words=max_features)
print(len(input_train))
print(len(input_test))

input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test,maxlen=maxlen)

print(input_train.shape)
print(input_test.shape)
```

```
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
    17464789/17464789 [==============================] - 1s 0us/step
    25000
    25000
```

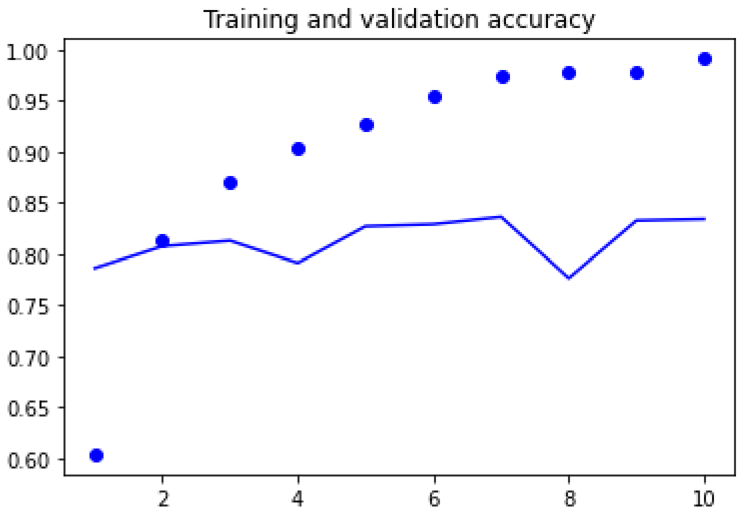
```
(25000, 500)
(25000, 500)
```

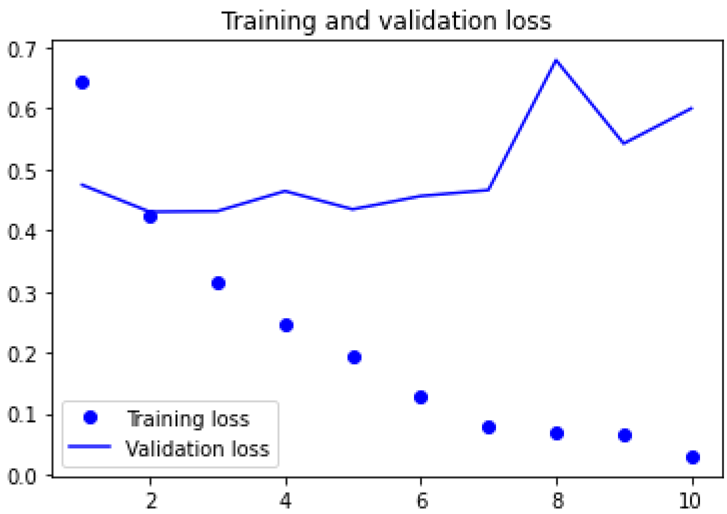
```
from keras.layers import Dense
from keras.models import Sequential
from keras.layers import Embedding,SimpleRNN

model = Sequential()
model.add(Embedding(max_features,32))
model.add(SimpleRNN(32))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
history = model.fit(input_train,y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
157/157 [==============================] - 79s 482ms/step - loss: 0.6457 - acc
Epoch 2/10
157/157 [==============================] - 74s 473ms/step - loss: 0.4230 - acc
Epoch 3/10
157/157 [==============================] - 75s 477ms/step - loss: 0.3145 - acc
Epoch 4/10
157/157 [==============================] - 73s 464ms/step - loss: 0.2473 - acc
Epoch 5/10
157/157 [==============================] - 74s 471ms/step - loss: 0.1933 - acc
Epoch 6/10
157/157 [==============================] - 74s 471ms/step - loss: 0.1289 - acc
Epoch 7/10
157/157 [==============================] - 72s 460ms/step - loss: 0.0785 - acc
Epoch 8/10
157/157 [==============================] - 74s 470ms/step - loss: 0.0686 - acc
Epoch 9/10
157/157 [==============================] - 72s 459ms/step - loss: 0.0660 - acc
Epoch 10/10
157/157 [==============================] - 74s 472ms/step - loss: 0.0299 - acc
```

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1,len(acc)+1)

plt.plot(epochs,acc,'bo',label='Training acc')
plt.plot(epochs,val_acc,'b', label='Validation acc')
plt.title('Training and validation accuracy')

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
```

```
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





Q2. Q1 문제를 LSTM 모델을 적용하여 수행하세요

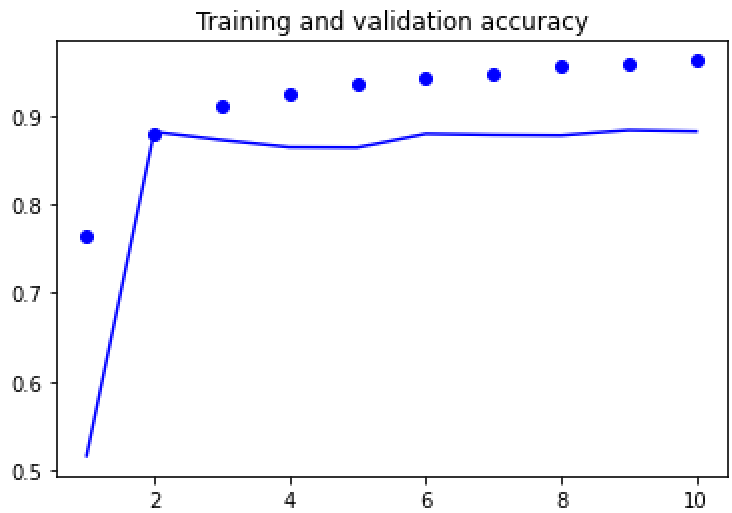
- 모델링, 학습 및 검증
- 결과 시각화

```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_features,32))
model.add(LSTM(32))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
history = model.fit(input_train,y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
    Epoch 1/10
```

```
157/157 [==============================] - 9s 24ms/step - loss: 0.5026 - acc:
Epoch 2/10
157/157 [==============================] - 3s 21ms/step - loss: 0.3073 - acc:
Epoch 3/10
157/157 [==============================] - 3s 21ms/step - loss: 0.2366 - acc:
Epoch 4/10
157/157 [==============================] - 3s 21ms/step - loss: 0.2040 - acc:
Epoch 5/10
157/157 [==============================] - 3s 21ms/step - loss: 0.1777 - acc:
Epoch 6/10
157/157 [==============================] - 3s 21ms/step - loss: 0.1618 - acc:
Epoch 7/10
157/157 [==============================] - 3s 21ms/step - loss: 0.1463 - acc:
Epoch 8/10
157/157 [==============================] - 3s 21ms/step - loss: 0.1276 - acc:
Epoch 9/10
157/157 [==============================] - 3s 22ms/step - loss: 0.1247 - acc:
Epoch 10/10
157/157 [==============================] - 4s 28ms/step - loss: 0.1082 - acc:
```

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1,len(acc)+1)

plt.plot(epochs,acc,'bo',label='Training acc')
plt.plot(epochs,val_acc,'b', label='Validation acc')
plt.title('Training and validation accuracy')

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



Q3. MNIST 숫자 이미지 데이터에 대하여 CNN 모델을 사용하여 아래사항을 수행하세요

- Conv2D와 MaxPooling2D 층을 사용하여 컨브넷을 생성(채널의 수 32개 또는 64개)
- 출력 텐서를 완전 연결 네트워크에 주입
- 10개의 클래스 분류하기 위한 분류기 추가
- 컨브넷 학습 및 평가

```
# 간단한 컨브넷 만들기
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation='relu',input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation='relu'))


# 전체 네트워크 확인
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

=================================================================
Total params: 55,744
```

```
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

```
# 컨브넷 위에 분류기 추가하기
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

```
# MNIST 이미지에 컨브넷 훈련하기
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255 # 0과 1사이의 값을 가지는 float타입으트

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(optimizer='rmsprop', # 입력된 데이터와 손실함수를 기반으로 네트워크를 업데이트하는 머
              loss='categorical_crossentropy', # 신경망의 성능을 측정하는 방법으로 네트워크가
              metrics=['accuracy']) # 훈련과 테스트 과정을 모니터링할 지표
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
11490434/11490434 [==============================] - 1s 0us/step
Epoch 1/5
938/938 [==============================] - 9s 5ms/step - loss: 0.1676 - accura
Epoch 2/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0461 - accura
Epoch 3/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0316 - accura
Epoch 4/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0252 - accura
Epoch 5/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0197 - accura
<keras.callbacks.History at 0x7f9c1bd39e90>
```

```
# 테스트 데이터에서 모델 평가
test_loss, test_acc = model.evaluate(test_images, test_labels)
test_acc
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0330 - accura
0.9902999997138977
```

Q4. cats_and_dogs_small으로 축소한 데이터 셋으로 사전 훈련된 네트워크를 사용하여 강아지 고양이 분류 과제를 아래와 같이 수행하세요.

- ImageNet 데이터셋에 훈련된 VGG16 네트워크의 합성곱 기반 층을 사용하여 유용한 특성 추출하고 이 특성으로 분류기 훈련
- ImageDataGenerator 사용 (※ 소요시간 20분)
- VGG 매개변수
  - weights는 모델을 초기화할 가중치 체크포인트를 지정 : 'imagenet'
  - include_top은 네트워크의 최상위 완전 연결 분류기를 포함할지 안할지를 지정 : False
  - input_shape은 네트워크에 주입할 이미지 텐서의 크기 :(150.150,3)
- 데이터 증식을 사용하지 않는 방법으로 수행
- 완전 연결 분류기를 정의하고 규제를 위해 드롭아웃 사용 : 0.5

```
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet', # imagenet으로 학습한 가중치
                  include_top=False,
                  input_shape=(150, 150, 3))
```

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808
```

```
 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

```
%cd /content/drive/MyDrive/강의/m9_딥러닝 기본/dataset
```

```
/content/drive/MyDrive/강의/m9_딥러닝 기본/dataset
```

```
!ls
```

```
aclImdb.zip          cats_and_dogs_small_1.h5  glove.6B.100d.txt
cats_and_dogs_small  cats_and_dogs_small_2.h5  glove.6B.zip
```

```
# 데이터 내려받기
import os, shutil
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator

base_dir = 'cats_and_dogs_small'
train_dir = os.path.join(base_dir,'train')
validation_dir = os.path.join(base_dir,'validation')
test_dir = os.path.join(base_dir,'test')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch) # conv_base에 데이터를 주입하고
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            # 제너레이터는 루프 안에서 무한하게 데이터를 만들어내므로 모든 이미지를 한 번씩 처리하고 나면
            break
    return features, labels
```

```
train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)
```

```
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
Found 1000 images belonging to 2 classes.
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 23ms/step
```

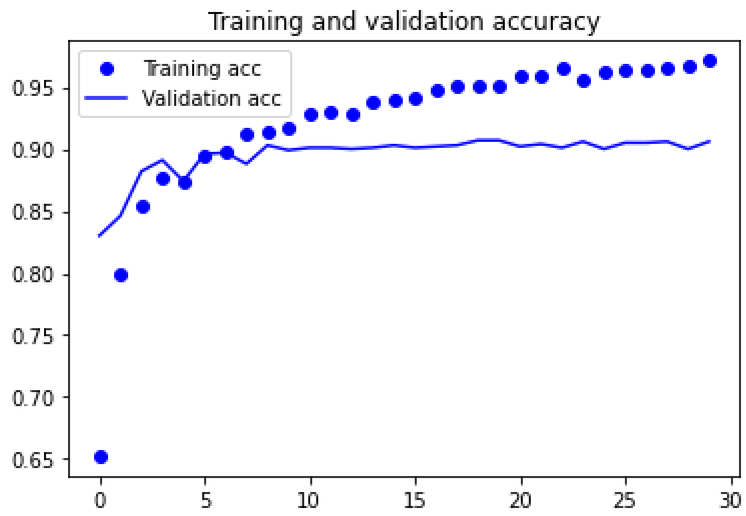
```
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

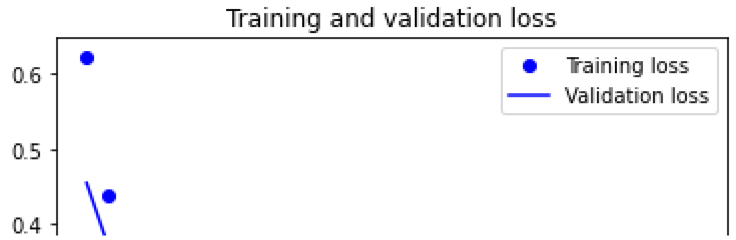
```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

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

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

```
Epoch 1/30
100/100 [==============================] - 1s 6ms/step - loss: 0.6217 - acc: 0
Epoch 2/30
100/100 [==============================] - 0s 5ms/step - loss: 0.4372 - acc: 0
Epoch 3/30
100/100 [==============================] - 0s 4ms/step - loss: 0.3524 - acc: 0
Epoch 4/30
100/100 [==============================] - 0s 5ms/step - loss: 0.3155 - acc: 0
Epoch 5/30
100/100 [==============================] - 0s 4ms/step - loss: 0.2956 - acc: 0
Epoch 6/30
100/100 [==============================] - 0s 4ms/step - loss: 0.2681 - acc: 0
Epoch 7/30
100/100 [==============================] - 0s 4ms/step - loss: 0.2513 - acc: 0
Epoch 8/30
100/100 [==============================] - 0s 5ms/step - loss: 0.2336 - acc: 0
Epoch 9/30
100/100 [==============================] - 0s 4ms/step - loss: 0.2260 - acc: 0
Epoch 10/30
100/100 [==============================] - 0s 5ms/step - loss: 0.2207 - acc: 0
Epoch 11/30
100/100 [==============================] - 0s 5ms/step - loss: 0.1989 - acc: 0
Epoch 12/30
100/100 [==============================] - 0s 4ms/step - loss: 0.1918 - acc: 0
Epoch 13/30
100/100 [==============================] - 0s 5ms/step - loss: 0.1919 - acc: 0
Epoch 14/30
100/100 [==============================] - 0s 5ms/step - loss: 0.1785 - acc: 0
Epoch 15/30
100/100 [==============================] - 0s 4ms/step - loss: 0.1689 - acc: 0
Epoch 16/30
100/100 [==============================] - 0s 4ms/step - loss: 0.1642 - acc: 0
```

```
Epoch 17/30
100/100 [==============================] - 0s 5ms/step - loss: 0.1574 - acc: 0
Epoch 18/30
100/100 [==============================] - 0s 4ms/step - loss: 0.1488 - acc: 0
Epoch 19/30
100/100 [==============================] - 0s 5ms/step - loss: 0.1429 - acc: 0
Epoch 20/30
100/100 [==============================] - 0s 4ms/step - loss: 0.1363 - acc: 0
Epoch 21/30
100/100 [==============================] - 0s 5ms/step - loss: 0.1268 - acc: 0
Epoch 22/30
100/100 [==============================] - 0s 4ms/step - loss: 0.1284 - acc: 0
Epoch 23/30
100/100 [==============================] - 0s 5ms/step - loss: 0.1214 - acc: 0
Epoch 24/30
100/100 [==============================] - 0s 4ms/step - loss: 0.1172 - acc: 0
Epoch 25/30
100/100 [==============================] - 0s 5ms/step - loss: 0.1129 - acc: 0
Epoch 26/30
100/100 [==============================] - 0s 5ms/step - loss: 0.1165 - acc: 0
Epoch 27/30
100/100 [==============================] - 0s 4ms/step - loss: 0.1080 - acc: 0
Epoch 28/30
100/100 [==============================] - 0s 4ms/step - loss: 0.1013 - acc: 0
Epoch 29/30
100/100 [==============================] - 0s 4ms/step - loss: 0.0972 - acc: 0
```

```python
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





Q5. Q4 문제를 데이터 증식을 사용한 방식으로 수행하세요.

```
from tensorflow.keras import models
from tensorflow.keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))


# 데이터 증식 제너레이터 사용 컨브넷 훈련
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode="nearest"
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150,150),
    batch_size=20,
    class_mode='binary'
```

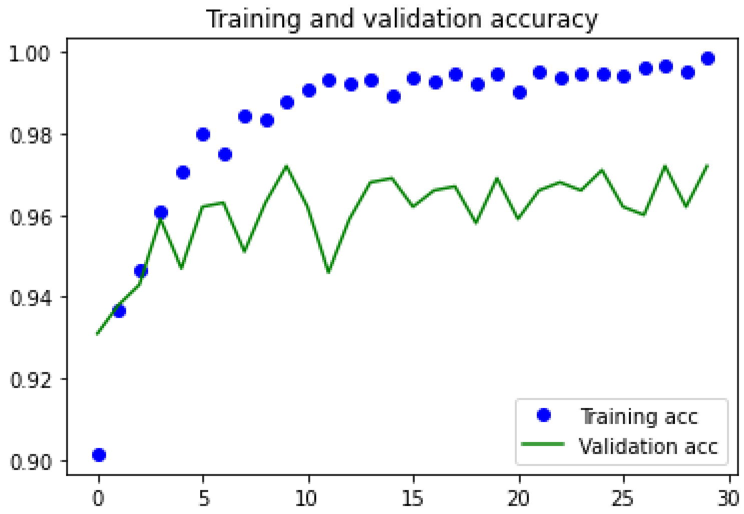
```
)

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150,150),
    batch_size=20,
    class_mode='binary'
)

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

history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50,
      verbose=2)
```

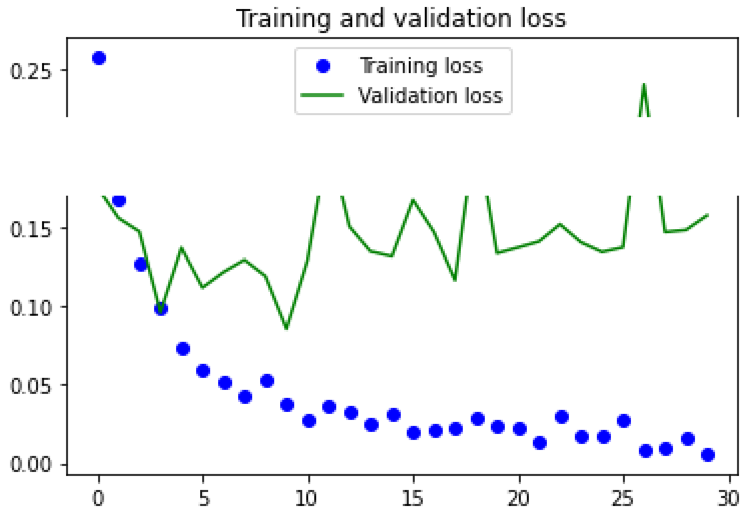
```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/rmsprop.p
  super(RMSprop, self).__init__(name, **kwargs)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:41: UserWarning:
Epoch 1/30
100/100 - 33s - loss: 0.2578 - acc: 0.9015 - val_loss: 0.1754 - val_acc: 0.931
Epoch 2/30
100/100 - 28s - loss: 0.1676 - acc: 0.9370 - val_loss: 0.1557 - val_acc: 0.938
Epoch 3/30
100/100 - 28s - loss: 0.1274 - acc: 0.9465 - val_loss: 0.1471 - val_acc: 0.943
Epoch 4/30
100/100 - 27s - loss: 0.0985 - acc: 0.9610 - val_loss: 0.0961 - val_acc: 0.959
Epoch 5/30
100/100 - 28s - loss: 0.0729 - acc: 0.9705 - val_loss: 0.1368 - val_acc: 0.947
Epoch 6/30
100/100 - 28s - loss: 0.0594 - acc: 0.9800 - val_loss: 0.1116 - val_acc: 0.962
Epoch 7/30
100/100 - 27s - loss: 0.0521 - acc: 0.9750 - val_loss: 0.1215 - val_acc: 0.963
Epoch 8/30
100/100 - 27s - loss: 0.0431 - acc: 0.9845 - val_loss: 0.1292 - val_acc: 0.951
Epoch 9/30
100/100 - 28s - loss: 0.0533 - acc: 0.9835 - val_loss: 0.1189 - val_acc: 0.963
Epoch 10/30
100/100 - 27s - loss: 0.0372 - acc: 0.9880 - val_loss: 0.0854 - val_acc: 0.972
Epoch 11/30
100/100 - 27s - loss: 0.0275 - acc: 0.9910 - val_loss: 0.1288 - val_acc: 0.962
Epoch 12/30
100/100 - 28s - loss: 0.0360 - acc: 0.9930 - val_loss: 0.2058 - val_acc: 0.946
Epoch 13/30
100/100 - 27s - loss: 0.0326 - acc: 0.9920 - val_loss: 0.1503 - val_acc: 0.959
Epoch 14/30
100/100 - 28s - loss: 0.0250 - acc: 0.9930 - val_loss: 0.1345 - val_acc: 0.968
Epoch 15/30
100/100 - 28s - loss: 0.0308 - acc: 0.9895 - val_loss: 0.1314 - val_acc: 0.969
Epoch 16/30
100/100 - 28s - loss: 0.0193 - acc: 0.9935 - val_loss: 0.1673 - val_acc: 0.962
```

```
Epoch 17/30
100/100 - 27s - loss: 0.0215 - acc: 0.9925 - val_loss: 0.1468 - val_acc: 0.966
Epoch 18/30
100/100 - 28s - loss: 0.0224 - acc: 0.9945 - val_loss: 0.1164 - val_acc: 0.967
Epoch 19/30
100/100 - 27s - loss: 0.0285 - acc: 0.9920 - val_loss: 0.2115 - val_acc: 0.958
Epoch 20/30
100/100 - 28s - loss: 0.0234 - acc: 0.9945 - val_loss: 0.1334 - val_acc: 0.969
Epoch 21/30
100/100 - 27s - loss: 0.0227 - acc: 0.9905 - val_loss: 0.1370 - val_acc: 0.959
Epoch 22/30
100/100 - 27s - loss: 0.0136 - acc: 0.9950 - val_loss: 0.1408 - val_acc: 0.966
Epoch 23/30
100/100 - 29s - loss: 0.0294 - acc: 0.9935 - val_loss: 0.1517 - val_acc: 0.968
Epoch 24/30
100/100 - 27s - loss: 0.0177 - acc: 0.9945 - val_loss: 0.1403 - val_acc: 0.966
Epoch 25/30
100/100 - 28s - loss: 0.0167 - acc: 0.9945 - val_loss: 0.1342 - val_acc: 0.971
Epoch 26/30
100/100 - 27s - loss: 0.0277 - acc: 0.9940 - val_loss: 0.1371 - val_acc: 0.962
Epoch 27/30
```

```
model.save('/content/drive/MyDrive/강의/m9_딥러닝 기본/dataset/cats_and_dogs_small_2.h5
```

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'g', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```


Training and validation accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0
5
10
15
20
25
30
Training acc
Validation acc


Training and validation loss
0.25
0.15
0.10
0.05
0.00
0
5
10
15
20
25
30
Training loss
Validation loss