CNN-week1

> 개와 고양이 이미지 분류

> 데이터셋 준비
>
> - Download

```
!wget --no-check-certificate \
https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
-O /tmp/cats_and_dogs_filtered.zip
```

> - Unzip

```
import os
import zipfile

local_zip = '/tmp/cats_and_dogs_filtered.zip'

zip_ref = zipfile.ZipFile(local_zip, 'r')

zip_ref.extractall('/tmp')
zip_ref.close()
```

> - 경로 지정

```
# 기본 경로
base_dir = '/tmp/cats_and_dogs_filtered'

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

# 훈련에 사용되는 고양이/개 이미지 경로
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')
print(train_cats_dir)
print(train_dogs_dir)

# 테스트에 사용되는 고양이/개 이미지 경로
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
print(validation_cats_dir)
print(validation_dogs_dir)
```

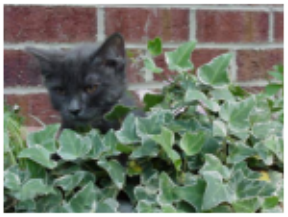
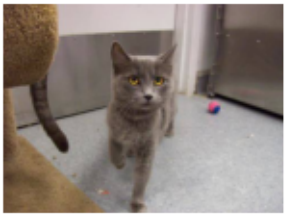
```
train_cat_fnames = os.listdir( train_cats_dir )
train_dog_fnames = os.listdir( train_dogs_dir )

print(train_cat_fnames[:5])
print(train_dog_fnames[:5])
```

os.listdir()를 사용해 파일명을 리스트로 반환

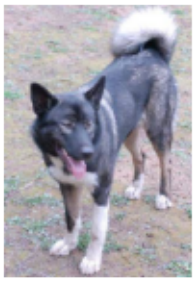
- 이미지 확인

```
%matplotlib inline

import matplotlib.image as mpimg
import matplotlib.pyplot as plt

nrows, ncols = 4, 4
pic_index = 0

fig = plt.gcf()
fig.set_size_inches(ncols*3, nrows*3)

pic_index+=8

next_cat_pix = [os.path.join(train_cats_dir, fname)
                for fname in train_cat_fnames[ pic_index-8:pic_index]]

next_dog_pix = [os.path.join(train_dogs_dir, fname)
                for fname in train_dog_fnames[ pic_index-8:pic_index]]

for i, img_path in enumerate(next_cat_pix+next_dog_pix):
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off')

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```

코드를 실행하면 다음과 같은 이미지를 볼 수 있다.

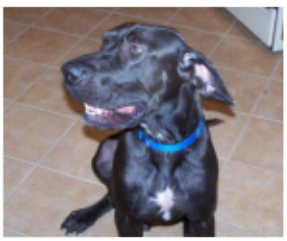













]

모델 구성

```
import tensorflow as tf


model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
```

```
tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

합성곱 신경망의 모델을 구성하고, summary()를 이용해 신경망의 구조를 확인하였다.

모델 컴파일

```
from tensorflow.keras.optimizers import RMSprop

model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics = ['accuracy'])
```

말과 사람 이미지 분류하기 예제와 마찬가지로, 손실함수로 **'binary_crossentropy'**를 사용하였다.

출력층의 활성화함수로 **'sigmoid'**를 사용하였다. 이것이 0,1로 분류되는 binary 분류 문제에 적합하기 때문이다.

옵티마이저로는 **RMSprop**를 사용하였다. RMSprop(Root Mean Square Propagation) 알고리즘은 훈련 과정 중에 학습률을 적절하게 변화시킨다.

이미지 데이터 전처리

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator


train_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

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

ImageDataGenerator 객체의 rescale 파라미터를 이용해서 모든 데이터를 255로 나누어준 다음,
flow_from_directory() 메서드를 이용해서 훈련과 테스트에 사용될 이미지 데이터를 만든다.

첫번째 인자로 이미지들이 위치한 경로를 입력하고, **batch_size**, **class_mode**를 지정한다.

target_size에 맞춰서 이미지의 크기가 조절된다.

모델 훈련

- **fit()** 메서드는 앞에서 구성한 Neural Network 모델을 훈련한다.

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

훈련과 테스트를 위한 데이터셋인 **train_generator, validation_generator**를 입력한다.

epochs는 데이터셋을 한 번 훈련하는 과정을 의미한다.

steps_per_epoch는 한 번의 에포크 (epoch)에서 훈련에 사용할 배치 (batch)의 개수를 지정한다.

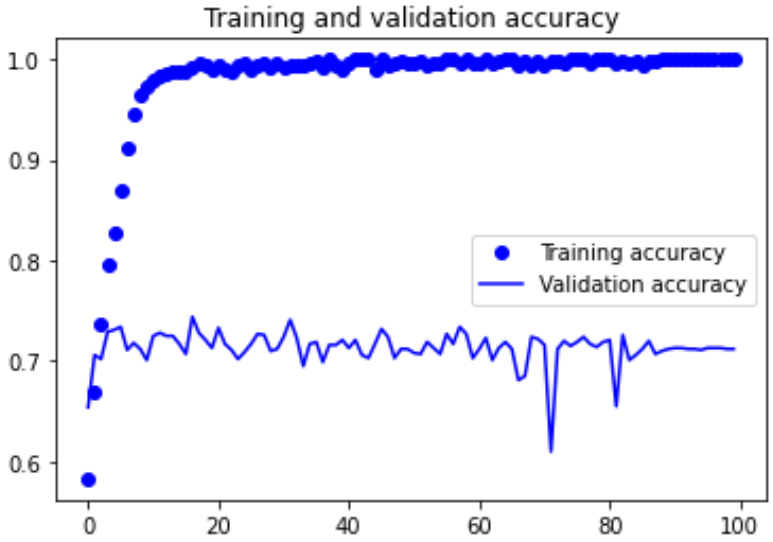
validation_steps는 한 번의 에포크가 끝날 때, 테스트에 사용되는 배치 (batch)의 개수를 지정한다.

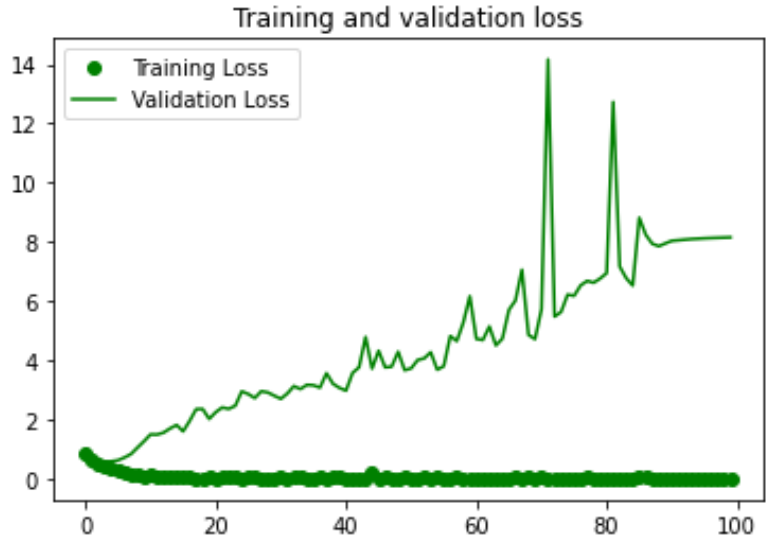
정확도와 손실 확인

```
import matplotlib.pyplot as plt

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

epochs = range(len(acc))

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

plt.figure()

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

plt.show()
```

Matplotlib 라이브러리를 이용해서 훈련 과정에서 에포크에 따른 정확도와 손실을 출력한다.

아래와 같은 이미지가 출력된다.





20회 에포크에서 훈련 정확도는 1.0에 근접한 반면, 테스트의 정확도는 100회 훈련이 끝나도 0.7 수준에 머물고 있다.

이러한 현상을 **과적합 (Overfitting)**이라고 한다.

테스트 이미지 분류

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded=files.upload()

for fn in uploaded.keys():

path='/content/' + fn
img=image.load_img(path, target_size=(150, 150))

x=image.img_to_array(img)
x=np.expand_dims(x, axis=0)
```

```
images = np.vstack([x])

classes = model.predict(images, batch_size=10)

print(classes[0])

if classes[0]>0:
 print(fn + " is a dog")
else:
 print(fn + " is a cat")
```

위의 코드는 하나 이상의 이미지를 업로드하고, 훈련된 모델을 사용해 개/고양이 분류 결과를 출력한다.

Exercise Code

```
import os
import zipfile
import random
import tensorflow as tf
import shutil
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from shutil import copyfile
from os import getcwd

# 데이터 압축해제
path_cats_and_dogs = f"{getcwd()}/../tmp2/cats-and-dogs.zip"
shutil.rmtree('/tmp')

local_zip = path_cats_and_dogs
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()

# 각 디렉토리에 몇 개의 개, 고양이 이미지가 있는지 확인
print(len(os.listdir('/tmp/PetImages/Cat/')))
print(len(os.listdir('/tmp/PetImages/Dog/')))

# Expected Output:
# 1500
# 1500

# 디렉토리 만들기
'''
디렉토리 구조
> cats-v-dogs
  > training
    > dogs
```

```
    > cats
  > testing
    > dogs
    > cats
'''
# Use os.mkdir to create your directories
# You will need a directory for cats-v-dogs, and subdirectories for training
# and testing. These in turn will need subdirectories for 'cats' and 'dogs'
try:
    os.mkdir("/tmp/cats-v-dogs")
    os.mkdir("/tmp/cats-v-dogs/training")
    os.mkdir("/tmp/cats-v-dogs/training/dogs")
    os.mkdir("/tmp/cats-v-dogs/training/cats")
    os.mkdir("/tmp/cats-v-dogs/testing")
    os.mkdir("/tmp/cats-v-dogs/testing/dogs")
    os.mkdir("/tmp/cats-v-dogs/testing/cats")
except OSError:
    pass

# 이미지 복사
# Cat 폴더에 있던 고양이 이미지를 training/cats 디렉토리와 testing/cats 디렉토리에 나누어 저장
# Dog 폴더에 있던 강아지 이미지를 training/dogs 디렉토리와 testing/dogs 디렉토리에 나누어 저장

def split_data(SOURCE, TRAINING, TESTING, SPLIT_SIZE):
# YOUR CODE STARTS HERE
# YOUR CODE ENDS HERE
    file = os.listdir(SOURCE)
    random.sample(file, len(file))

    num_of_train = int(len(file)*SPLIT_SIZE)
    num_of_test = len(file)-num_of_train

    for i in range(num_of_train):
        file_name = file.pop()
        copyfile(os.path.join(SOURCE, file_name), os.path.join(TRAINING,
file_name))
    for i in range(num_of_test):
        file_name = file.pop()
        copyfile(os.path.join(SOURCE, file_name), os.path.join(TESTING, file_name))

CAT_SOURCE_DIR = "/tmp/PetImages/Cat/"
TRAINING_CATS_DIR = "/tmp/cats-v-dogs/training/cats/"
TESTING_CATS_DIR = "/tmp/cats-v-dogs/testing/cats/"
DOG_SOURCE_DIR = "/tmp/PetImages/Dog/"
TRAINING_DOGS_DIR = "/tmp/cats-v-dogs/training/dogs/"
TESTING_DOGS_DIR = "/tmp/cats-v-dogs/testing/dogs/"

split_size = .9
split_data(CAT_SOURCE_DIR, TRAINING_CATS_DIR, TESTING_CATS_DIR, split_size)
```

```
split_data(DOG_SOURCE_DIR, TRAINING_DOGS_DIR, TESTING_DOGS_DIR, split_size)

# train/test할 개, 고양이 이미지 수 확인
print(len(os.listdir('/tmp/cats-v-dogs/training/cats/')))
print(len(os.listdir('/tmp/cats-v-dogs/training/dogs/')))
print(len(os.listdir('/tmp/cats-v-dogs/testing/cats/')))
print(len(os.listdir('/tmp/cats-v-dogs/testing/dogs/')))

# Expected output:
# 1350
# 1350
# 150
# 150

# 모델 정의
'''
여기서는 convolutional layer를 사용하였다.
여기서 주의할 점은 input_shape 지정해주는 것이다.
강아지와 고양이를 분류하는 이진 분류 문제이기 때문에 마지막 레이어의 node 수는 1개로, activation
function으로는 sigmoid를 사용해주었다.
이진 분류 문제이기 때문에 loss는 binary_crossentropy를 사용하였다.
'''
# DEFINE A KERAS MODEL TO CLASSIFY CATS V DOGS
# USE AT LEAST 3 CONVOLUTION LAYERS
model = tf.keras.models.Sequential([
# YOUR CODE HERE
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', input_shape=(300, 300,
3)),
    tf.keras.layers.MaxPool2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D(2, 2),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D((2, 2)),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer=RMSprop(lr=0.001), loss='binary_crossentropy', metrics=
['acc'])

# data generator
'''
ImageDataGenerator 만든다 (학습이 더 잘 될 수 있도록 rescale=1./255를 설정해 0~1 사이의 값으로
정규화 해주었다.)
train_datagen과 validation_datagen에서의 target_size는 무조건 동일해야 한다.
class_mode는 이진 분류 문제이기 때문에 'binary'로 설정해주었다. (실제 고양이, 강아지 이 2개의 하위폴더
만 존재)
```

```
'''

TRAINING_DIR = "/tmp/cats-v-dogs/training"
train_datagen = ImageDataGenerator(rescale=1./255)

# NOTE: YOU MUST USE A BATCH SIZE OF 10 (batch_size=10) FOR THE
# TRAIN GENERATOR.
train_generator = train_datagen.flow_from_directory(
    TRAINING_DIR,
    target_size=(300, 300),
    batch_size = 10,
    class_mode='binary'
)

VALIDATION_DIR = "/tmp/cats-v-dogs/testing"
validation_datagen = ImageDataGenerator(rescale=1./255)

# NOTE: YOU MUST USE A BACTH SIZE OF 10 (batch_size=10) FOR THE
# VALIDATION GENERATOR.
validation_generator = validation_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=(300, 300),
    batch_size=10,
    class_mode='binary'
)



# Expected Output:
# Found 2700 images belonging to 2 classes.
# Found 300 images belonging to 2 classes.

# 모델 학습
'''
모델 학습을 위해 train data, epochs 지정해준다.
추가로 validation score 확인하기 위해 validation data를 지정해주었다.
'''
history = model.fit_generator(train_generator,
                              epochs=2,
                              verbose=1,
                              validation_data=validation_generator)

# 모델 분석
'''
train data에서의 accuracy, loss 그리고 validation data에서의 accuracy, loss를 이용하여 모델
을 개선한다.
이 값을 이용해 현재 overfitting이 일어나고 있는지, underfitting이 일어나고 있는지,
어떠한 클래스를 잘 예측하지 못하는지 등을 파악할 수 있다.
'''
# PLOT LOSS AND ACCURACY
```

```
%matplotlib inline

import matplotlib.image  as mpimg
import matplotlib.pyplot as plt

#-----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
acc=history.history['acc']
val_acc=history.history['val_acc']
loss=history.history['loss']
val_loss=history.history['val_loss']

epochs=range(len(acc)) # Get number of epochs

#------------------------------------------------
# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot(epochs, acc, 'r', "Training Accuracy")
plt.plot(epochs, val_acc, 'b', "Validation Accuracy")
plt.title('Training and validation accuracy')
plt.figure()

#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss, 'r', "Training Loss")
plt.plot(epochs, val_loss, 'b', "Validation Loss")


plt.title('Training and validation loss')

# Desired output. Charts with training and validation metrics. No crash :)
```