# 강아지 vs 고양이 분류하기(1)

## 학습 내용

- 강아지와 고양이 데이터를 캐글로 부터 데이터를 준비
- 학습/검증/테스트 폴더 생성 후, 일부 데이터를 준비
- 신경망을 구성 후, 학습을 수행합니다.
- 모델을 저장하는 것에 대해 알아봅니다.

## 환경

- Google Colab

## 데이터(강아지 vs 고양이)

- url : https://www.kaggle.com/c/dogs-vs-cats/data
- test1.zip : 271.15MB
- train.zip : 543.16MB
- sampleSubmission.csv

## 01 데이터 준비하기

- 25,000개의 강아지와 고양이 이미지
- 클래스마다 12,500개를 담고 있다. 압축(543MB크기)
- 클래스마다 1,000개의 샘플로 이루어진 훈련 세트
- 클래스마다 500개의 샘플로 이루어진 검증 세트
- 클래스마다 500개의 샘플로 이루어진 테스트 세트

In [1]:
```
import os, shutil
```

## 구글 드라이브 연동하기

- 링크가 뜨면 해당 링크로 연결하여 연결 암호 정보를 빈칸에 넣어준다.

In [2]:
```
### 드라이브 마운트
from google.colab import drive
drive.mount('/content/drive')
```

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

In [3]:
```
# 데이터 셋 복사 및 확인
!cp -r '/content/drive/My Drive/dataset/cats_dogs'  '/content/'
!ls -ls '/content/cats_dogs'
```

```
total 833956
277664 -rw------- 1 root root 284321224 Dec 27 13:07 cats_and_dogs_test1.zip
556204 -rw------- 1 root root 569546721 Dec 27 13:07 cats_and_dogs_train.zip
    88 -rw------- 1 root root     88903 Dec 27 13:07 sampleSubmission.csv
```

## 압축풀기를 위한 명령어

```
!rm -rf '/content/datasets/'
!unzip '/content/cats_dogs/cats_and_dogs_test1.zip' -d
'/content/datasets/'
!unzip '/content/cats_dogs/cats_and_dogs_train.zip' -d
'/content/datasets/'
```

In [7]:
```
#!rm -rf '/content/datasets/'
#!unzip '/content/cats_dogs/cats_and_dogs_test1.zip' -d '/content/datasets/'
#!unzip '/content/cats_dogs/cats_and_dogs_train.zip' -d '/content/datasets/'
```

## 파일 확인

In [8]:
```
!ls -al '/content/datasets/train' | head -5
!ls -l '/content/datasets/train' | grep ^- | wc -l
!ls -al '/content/datasets/test1' | head -5
!ls -l '/content/datasets/test1' | grep ^- | wc -l
```

```
total 609260
drwxr-xr-x 2 root root 770048 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Dec 27 13:07 ..
-rw-r--r-- 1 root root  12414 Sep 20  2013 cat.0.jpg
-rw-r--r-- 1 root root  21944 Sep 20  2013 cat.10000.jpg
25000
total 304264
drwxr-xr-x 2 root root 274432 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Dec 27 13:07 ..
-rw-r--r-- 1 root root  54902 Sep 20  2013 10000.jpg
-rw-r--r-- 1 root root  21671 Sep 20  2013 10001.jpg
12500
```

In [9]:
```
# 원본 데이터셋을 압축 해제한 디렉터리 경로
ori_dataset_dir = './datasets/train'

# 소규모 데이터셋을 저장할 디렉터리
base_dir = './datasets/cats_and_dogs_small'
```

In [10]:
```
# 여러번 반복실행을 위해 디렉터리 삭제
if os.path.exists(base_dir):
    shutil.rmtree(base_dir)
os.mkdir(base_dir)
```

In [11]:
```
# 훈련, 검증, 테스트 분할을 위한 디렉터리
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
```

```
val_dir = os.path.join(base_dir, 'validation')
os.mkdir(val_dir)

test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)
```

In [12]:
```
# 훈련용 고양이 사진 디렉터리
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)

# 훈련용 강아지 사진 디렉터리
train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)

# 검증용 고양이 사진 디렉터리
val_cats_dir = os.path.join(val_dir, 'cats')
os.mkdir(val_cats_dir)

# 검증용 강아지 사진 디렉터리
val_dogs_dir = os.path.join(val_dir, 'dogs')
os.mkdir(val_dogs_dir)

# 테스트용 고양이 사진 디렉터리
test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)

# 테스트용 강아지 사진 디렉터리
test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)
```

## 데이터 준비

In [13]:
```
# 처음 1,000개의 고양이 이미지를 train_cats_dir에 복사합니다
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 고양이 이미지를 validation_cats_dir에 복사합니다
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(val_cats_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 고양이 이미지를 test_cats_dir에 복사합니다
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)

# 처음 1,000개의 강아지 이미지를 train_dogs_dir에 복사합니다
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 강아지 이미지를 validation_dogs_dir에 복사합니다
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
```

```
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(val_dogs_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 강아지 이미지를 test_dogs_dir에 복사합니다
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

## [훈련/검증/테스트]에 들어있는 사진의 개수를 카운트

In [14]:
```
print('훈련용 고양이 이미지 전체 개수:', len(os.listdir(train_cats_dir)))
print('훈련용 강아지 이미지 전체 개수:', len(os.listdir(train_dogs_dir)))

print('검증용 고양이 이미지 전체 개수:', len(os.listdir(val_cats_dir)))
print('검증용 강아지 이미지 전체 개수:', len(os.listdir(val_dogs_dir)))

print('테스트용 고양이 이미지 전체 개수:', len(os.listdir(test_cats_dir)))
print('테스트용 강아지 이미지 전체 개수:', len(os.listdir(test_dogs_dir)))
```

```
훈련용 고양이 이미지 전체 개수: 1000
훈련용 강아지 이미지 전체 개수: 1000
검증용 고양이 이미지 전체 개수: 500
검증용 강아지 이미지 전체 개수: 500
테스트용 고양이 이미지 전체 개수: 500
테스트용 강아지 이미지 전체 개수: 500
```

- 훈련 이미지 : 2000개
- 검증 이미지 : 1000개
- 테스트 이미지 : 1000개

## 02 신경망 구성하기

In [15]:
```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
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(128, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [16]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
```

```
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

- 150 x 150 크기에서 7 x 7 크기의 특성 맵으로 줄어든다.

## 최적화 알고리즘, 손실 함수 선택

In [17]:
```
from keras import optimizers

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

# 03 데이터 전처리

- 사진 파일을 읽기
- JPEG 콘텐츠를 RGB픽셀로 디코딩
- 부동 소수 타입의 텐서로 변환
- 픽셀 값(0~255)의 스케일을 [0,1]사이로 조정

## 준비된 유틸리티

- keras.preprocessing.image
  - ImageDataGenerator : 디스크에 있는 이미지 파일을 배치 텐서로 변경하는 파이썬 제너레이터 생성

In [18]:
```
from keras.preprocessing.image import ImageDataGenerator

# 모든 이미지를 1/255로 스케일을 조정합니다
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,  # 타깃 디렉터리
        target_size=(150, 150),  # 모든 이미지를 150 × 150 크기로 변경
        batch_size=20,
        # binary_crossentropy 손실을 사용하기 때문에 이진 레이블이 필요
        class_mode='binary')
```

```
val_generator = test_datagen.flow_from_directory(
        val_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

## 제너레이터을 활용하여 데이터와 라벨을 확인해보기

In [19]:
```
for data_batch, labels_batch in train_generator:
    print('배치 데이터 크기:', data_batch.shape)
    print('배치 레이블 크기:', labels_batch.shape)
    break
```

```
배치 데이터 크기: (20, 150, 150, 3)
배치 레이블 크기: (20,)
```

- 제너레이터는 이 배치를 무한정으로 만들어낸다.
- 타깃 폴더에 있는 이미지를 끝없이 반복한다.

## 제너레이터를 사용한 데이터의 모델 훈련

- fit_generator 메서드는 fit 메서드와 동일하고 데이터 제너레이터를 사용 가능.
  - 데이터가 끝없이 생성되기에 케라스 모델에 하나의 에포크를 정의하기 위해 제너레이터로부터 얼마나 많은 샘플을 뽑을지 알려 주어야 한다.(steps_per_epoch 이를 설정)
  - validation_data 매개변수를 사용 가능.
  - validation_steps에서 얼마나 많은 배치를 추출할지 평가할지 validation_steps 매개변수에 지정한다.

In [20]:
```
## 경로에 이미지 데이터의 개수
num_cats_tr = len(os.listdir(train_cats_dir))
num_dogs_tr = len(os.listdir(train_dogs_dir))

num_cats_val = len(os.listdir(val_cats_dir))
num_dogs_val = len(os.listdir(val_dogs_dir))

total_train = num_cats_tr + num_dogs_tr
total_val = num_cats_val + num_dogs_val

print("학습용 데이터 : ", total_train)
print("검증용 데이터 : ", total_val)

batch_size = 20
epochs = 30
```

```
학습용 데이터 :  2000
검증용 데이터 :  1000
```

In [21]:
```
%%time

history = model.fit(   # model.fit_gerator -> model.fit() 으로 최근 추가됨
      train_generator,
      steps_per_epoch= total_train // batch_size,
      epochs=30,
      validation_data=val_generator,
      validation_steps= total_val // batch_size)
```

```
Epoch 1/30
100/100 [==============================] - 16s 89ms/step - loss: 0.6966 - acc: 0.5027
```

```
- val_loss: 0.6762 - val_acc: 0.6140
Epoch 2/30
100/100 [==============================] - 9s 86ms/step - loss: 0.6669 - acc: 0.5925 -
val_loss: 0.6508 - val_acc: 0.6330
Epoch 3/30
100/100 [==============================] - 8s 85ms/step - loss: 0.6334 - acc: 0.6463 -
val_loss: 0.6290 - val_acc: 0.6430
Epoch 4/30
100/100 [==============================] - 8s 85ms/step - loss: 0.5955 - acc: 0.6710 -
val_loss: 0.6131 - val_acc: 0.6580
Epoch 5/30
100/100 [==============================] - 8s 85ms/step - loss: 0.5567 - acc: 0.7090 -
val_loss: 0.6246 - val_acc: 0.6340
Epoch 6/30
100/100 [==============================] - 8s 85ms/step - loss: 0.5349 - acc: 0.7298 -
val_loss: 0.5930 - val_acc: 0.6840
Epoch 7/30
100/100 [==============================] - 8s 85ms/step - loss: 0.4860 - acc: 0.7732 -
val_loss: 0.5850 - val_acc: 0.6860
Epoch 8/30
100/100 [==============================] - 8s 85ms/step - loss: 0.4664 - acc: 0.7946 -
val_loss: 0.5925 - val_acc: 0.6950
Epoch 9/30
100/100 [==============================] - 9s 86ms/step - loss: 0.4400 - acc: 0.7847 -
val_loss: 0.5594 - val_acc: 0.7130
Epoch 10/30
100/100 [==============================] - 8s 85ms/step - loss: 0.4059 - acc: 0.8194 -
val_loss: 0.5680 - val_acc: 0.7080
Epoch 11/30
100/100 [==============================] - 8s 85ms/step - loss: 0.3876 - acc: 0.8347 -
val_loss: 0.5445 - val_acc: 0.7310
Epoch 12/30
100/100 [==============================] - 8s 85ms/step - loss: 0.3546 - acc: 0.8407 -
val_loss: 0.5568 - val_acc: 0.7310
Epoch 13/30
100/100 [==============================] - 8s 84ms/step - loss: 0.3340 - acc: 0.8494 -
val_loss: 0.6126 - val_acc: 0.7200
Epoch 14/30
100/100 [==============================] - 9s 85ms/step - loss: 0.3136 - acc: 0.8531 -
val_loss: 0.5744 - val_acc: 0.7270
Epoch 15/30
100/100 [==============================] - 9s 85ms/step - loss: 0.3007 - acc: 0.8779 -
val_loss: 0.5759 - val_acc: 0.7340
Epoch 16/30
100/100 [==============================] - 8s 84ms/step - loss: 0.2510 - acc: 0.9086 -
val_loss: 0.7549 - val_acc: 0.6910
Epoch 17/30
100/100 [==============================] - 8s 84ms/step - loss: 0.2511 - acc: 0.8948 -
val_loss: 0.6108 - val_acc: 0.7200
Epoch 18/30
100/100 [==============================] - 8s 84ms/step - loss: 0.2171 - acc: 0.9209 -
val_loss: 0.6205 - val_acc: 0.7250
Epoch 19/30
100/100 [==============================] - 8s 85ms/step - loss: 0.1983 - acc: 0.9225 -
val_loss: 0.7427 - val_acc: 0.7010
Epoch 20/30
100/100 [==============================] - 8s 84ms/step - loss: 0.1768 - acc: 0.9366 -
val_loss: 0.6525 - val_acc: 0.7370
Epoch 21/30
100/100 [==============================] - 8s 85ms/step - loss: 0.1474 - acc: 0.9515 -
val_loss: 0.7104 - val_acc: 0.7190
Epoch 22/30
100/100 [==============================] - 8s 84ms/step - loss: 0.1437 - acc: 0.9484 -
val_loss: 0.7776 - val_acc: 0.7210
Epoch 23/30
100/100 [==============================] - 8s 83ms/step - loss: 0.1210 - acc: 0.9621 -
val_loss: 0.7065 - val_acc: 0.7240
Epoch 24/30
100/100 [==============================] - 8s 84ms/step - loss: 0.0866 - acc: 0.9798 -
```

```
val_loss: 1.0676 - val_acc: 0.6940
Epoch 25/30
100/100 [==============================] - 8s 84ms/step - loss: 0.0863 - acc: 0.9762 -
val_loss: 0.7653 - val_acc: 0.7210
Epoch 26/30
100/100 [==============================] - 9s 85ms/step - loss: 0.0653 - acc: 0.9794 -
val_loss: 1.1465 - val_acc: 0.6930
Epoch 27/30
100/100 [==============================] - 8s 85ms/step - loss: 0.0639 - acc: 0.9857 -
val_loss: 0.8870 - val_acc: 0.7260
Epoch 28/30
100/100 [==============================] - 8s 84ms/step - loss: 0.0509 - acc: 0.9890 -
val_loss: 0.8828 - val_acc: 0.7170
Epoch 29/30
100/100 [==============================] - 8s 84ms/step - loss: 0.0368 - acc: 0.9913 -
val_loss: 0.9408 - val_acc: 0.7270
Epoch 30/30
100/100 [==============================] - 8s 84ms/step - loss: 0.0308 - acc: 0.9956 -
val_loss: 0.9782 - val_acc: 0.7440
CPU times: user 4min 48s, sys: 22.7 s, total: 5min 11s
Wall time: 4min 21s
```

In [23]:
```
### CPU 30 epochs : 52분 55초
### GPU 30 epochs : 4분 21초
```

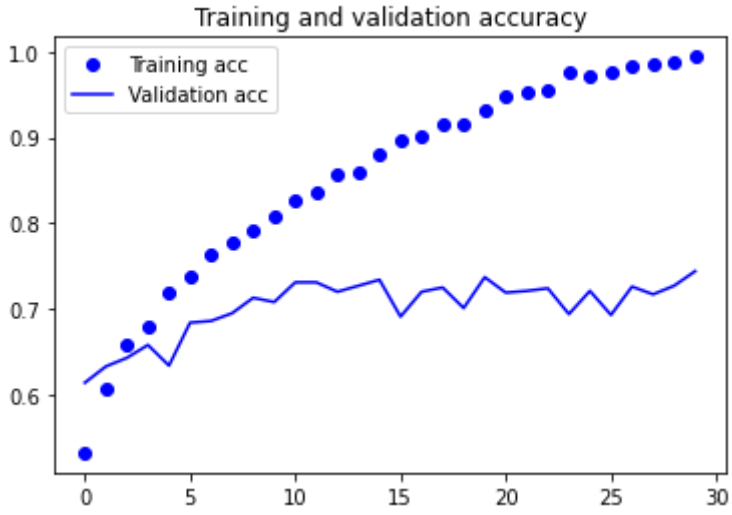
## 훈련 후, 모델 저장

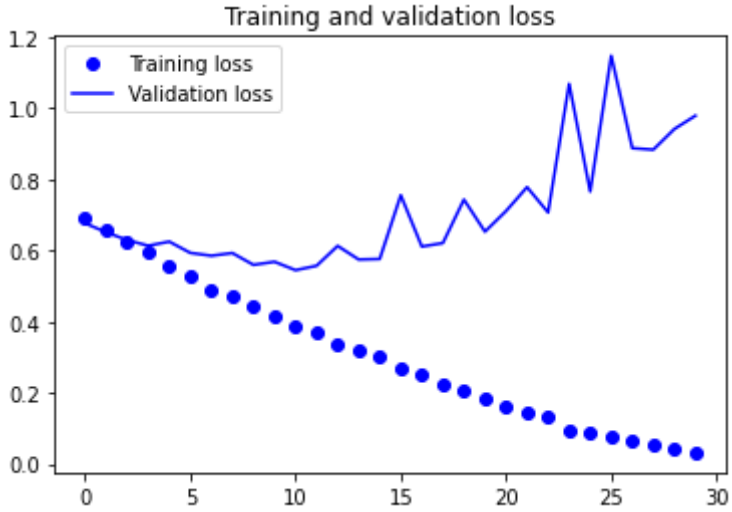
In [24]:
```
model.save('cats_and_dogs_small_1.h5')
```

## 훈련 데이터와 검증 데이터의 모델의 손실과 정확도

In [25]:
```
import matplotlib.pyplot as plt
```

In [26]:
```
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()
```




- 검증 손실은 다섯 번의 에포크만에 최소값에 다다른 이후 더 이상 진전이 없음.
- 반면 훈련 손실은 거의 0에 도달할 때까지 선형적으로 계속 감소.
- 비교적 훈련 샘플의 수(2,000)개 적기 때문에 과대 적합이 가장 중요.
  - 드롭아웃이나 가중치 감소(L2규제)와 같은 과대적합을 감소시킬 수 있는 여러가지 기법 학습.

## 실습해 보기

- 다양한 방법을 이용해서 성능을 개선시키는 것을 해 보기

In [ ]: