데이터 증식을 사용한 특성 추출

- 강아지와 고양이 데이터를 캐글로 부터 데이터를 준비
- 사전 훈련 모델에 대해 알아봅니다.
- VGG16 모델을 불러와 사용해 봅니다.
    - 데이터 증식을 사용함.
- FCL 분류기를 연결하여 모델 학습
- 일부 층을 훈련하여 성능을 확인해 봅니다.

```
import os, shutil
```

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

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

```
#!cp -r '/content/drive/My Drive/dataset/cats_dogs'  '/content/'
#!ls -ls '/content/cats_dogs'

# 압축풀기
#!rm -rf '/content/datasets/'
#!unzip '/content/cats_dogs/test1.zip' -d '/content/datasets/'
#!unzip '/content/cats_dogs/train.zip' -d '/content/datasets/'
```

```
!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 609256
drwxr-xr-x 2 root root 765952 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Nov 10 16:19 ..
-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 304280
drwxr-xr-x 2 root root 290816 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Nov 10 16:19 ..
-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
```

데이터 준비

```
# 원본 데이터셋을 압축 해제한 디렉터리 경로
ori_dataset_dir = './datasets/train'
```

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

# 반복실행을 위해 디렉터리 삭제
if os.path.exists(base_dir):
    shutil.rmtree(base_dir)
os.mkdir(base_dir)

# 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)

# 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)



# 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)

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
```

```
## 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)
```

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

모델 구축

```
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg1
58892288/58889256 [==============================] - 1s 0us/step
58900480/58889256 [==============================] - 1s 0us/step
```

```
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'))


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 256)               2097408

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
_________________________________________________________________
```

- VGG16의 합성곱 기반층은 14,714,688개의 매우 많은 파라미터 보유

- 모델을 컴파일하고 훈련하기 전에 합성곱 기반층을 동결하는 것이 아주 중요
    - 하나 이상의 층을 동결한다는 것은 훈련하는 동안 가중치가 업데이트되지 않도록 막는다는 뜻
    - 동결하지 않으면 합성곱 기반층에 의해 사전에 학습된 표현이 훈련하는 동안 수정
    - 맨 위의 Dense 층은 랜덤하게 초기화되었기 때문에 매우 큰 가중치 업데이트 값이 네트워크에 전파
    - 케라스에서는 trainable 속성을 False로 설정하여 네트워크를 동결

```
print('conv_base를 동결 전 훈련되는 가중치의 수:', len(model.trainable_weights))
conv_base.trainable = False
print('conv_base를 동결 후 훈련되는 가중치의 수:',   len(model.trainable_weights))
```

```
conv_base를 동결 전 훈련되는 가중치의 수: 30
conv_base를 동결 후 훈련되는 가중치의 수: 4
```

- 2개의 Dense층 가중치만 훈련
- 각 층마다 2개씩(가중치 행렬과 편향 벡터) 총 4개의 가중치(텐서)가 훈련
- 컴파일 단계 후에 trainable 속성을 변경하면 반드시 모델을 다시 컴파일 해야 한다.

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

batch_size = 20
epochs = 30
```

- rotation_range : 랜덤하게 사진을 회전시킬 각도 범위(0~180 사이)
- width_shift_range와 height_shift_range는 사진을 수평과 수직으로 랜덤하게 평행 이동 시킬 범위
- shear_range : 랜덤하게 전단 변환(shearing transformation)을 적용할 각도 범위
- zoom_range : 랜덤하게 사진을 확대할 범위
- horizontal_flip : 랜덤하게 이미지를 수평으로 뒤집기. 수평 대칭을 가정할 수 있을 때 사용. (예 풍경/인물 사진)
- fill_mode : 회전이나 가로/세로 이동으로 인해 새롭게 생성해야 할 픽셀을 채울 전략

```
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')

train_generator = train_datagen.flow_from_directory(
       # 타깃 디렉터리
```

```
        train_dir,
        # 모든 이미지의 크기를 150 × 150로 변경합니다
        target_size=(150, 150),
        batch_size=20,
        # binary_crossentropy 손실을 사용하므로 이진 레이블이 필요합니다
        class_mode='binary')

# 검증 데이터는 증식되어서는 안 됩니다!
test_datagen = ImageDataGenerator(rescale=1./255)

validation_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.

%%time

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

history = model.fit_generator(
            train_generator,
            steps_per_epoch=100,  # total_train // batch_size
            epochs=30,
            validation_data=validation_generator,
            validation_steps=50,  # total_val // batch_size
            verbose=2)
Epoch 3/30
100/100 - 26s - loss: 0.3520 - acc: 0.8525 - val_loss: 0.3028 - val_acc: 0.8770 - 26s/ep
Epoch 4/30
100/100 - 25s - loss: 0.3211 - acc: 0.8710 - val_loss: 0.2839 - val_acc: 0.8880 - 25s/ep
Epoch 5/30
100/100 - 26s - loss: 0.3059 - acc: 0.8660 - val_loss: 0.2729 - val_acc: 0.8990 - 26s/ep
Epoch 6/30
100/100 - 26s - loss: 0.2930 - acc: 0.8780 - val_loss: 0.2674 - val_acc: 0.8930 - 26s/ep
Epoch 7/30
100/100 - 26s - loss: 0.2797 - acc: 0.8845 - val_loss: 0.2583 - val_acc: 0.8980 - 26s/ep
Epoch 8/30
100/100 - 25s - loss: 0.2728 - acc: 0.8925 - val_loss: 0.2597 - val_acc: 0.8940 - 25s/ep
Epoch 9/30
100/100 - 26s - loss: 0.2615 - acc: 0.8965 - val_loss: 0.2524 - val_acc: 0.8980 - 26s/ep
Epoch 10/30
100/100 - 25s - loss: 0.2495 - acc: 0.8930 - val_loss: 0.2481 - val_acc: 0.9000 - 25s/ep

Epoch 11/30
100/100 - 25s - loss: 0.2499 - acc: 0.9035 - val_loss: 0.2455 - val_acc: 0.8980 - 25s/ep
Epoch 12/30
100/100 - 25s - loss: 0.2408 - acc: 0.9020 - val_loss: 0.2444 - val_acc: 0.8980 - 25s/ep
Epoch 13/30
100/100 - 25s - loss: 0.2380 - acc: 0.9070 - val_loss: 0.2444 - val_acc: 0.9040 - 25s/ep
Epoch 14/30
100/100 - 25s - loss: 0.2218 - acc: 0.9115 - val_loss: 0.2512 - val_acc: 0.8970 - 25s/ep
```

```
Epoch 15/30
100/100 - 25s - loss: 0.2217 - acc: 0.9090 - val_loss: 0.2392 - val_acc: 0.9040 - 25s/ep
Epoch 16/30
100/100 - 25s - loss: 0.2218 - acc: 0.9125 - val_loss: 0.2459 - val_acc: 0.9010 - 25s/ep
Epoch 17/30
100/100 - 26s - loss: 0.2241 - acc: 0.9080 - val_loss: 0.2409 - val_acc: 0.9000 - 26s/ep
Epoch 18/30
100/100 - 25s - loss: 0.2164 - acc: 0.9105 - val_loss: 0.2366 - val_acc: 0.9020 - 25s/ep
Epoch 19/30
100/100 - 25s - loss: 0.2038 - acc: 0.9200 - val_loss: 0.2511 - val_acc: 0.8950 - 25s/ep
Epoch 20/30
100/100 - 25s - loss: 0.2090 - acc: 0.9195 - val_loss: 0.2372 - val_acc: 0.9050 - 25s/ep
Epoch 21/30
100/100 - 25s - loss: 0.2012 - acc: 0.9225 - val_loss: 0.2363 - val_acc: 0.9020 - 25s/ep
Epoch 22/30
100/100 - 25s - loss: 0.2096 - acc: 0.9140 - val_loss: 0.2360 - val_acc: 0.8990 - 25s/ep
Epoch 23/30
100/100 - 25s - loss: 0.2083 - acc: 0.9245 - val_loss: 0.2351 - val_acc: 0.9030 - 25s/ep
Epoch 24/30
100/100 - 25s - loss: 0.2057 - acc: 0.9175 - val_loss: 0.2412 - val_acc: 0.9000 - 25s/ep
Epoch 25/30
100/100 - 25s - loss: 0.1910 - acc: 0.9290 - val_loss: 0.2473 - val_acc: 0.9010 - 25s/ep
Epoch 26/30
100/100 - 25s - loss: 0.1928 - acc: 0.9220 - val_loss: 0.2353 - val_acc: 0.9080 - 25s/ep
Epoch 27/30
100/100 - 25s - loss: 0.1919 - acc: 0.9230 - val_loss: 0.2391 - val_acc: 0.9080 - 25s/ep
Epoch 28/30
100/100 - 25s - loss: 0.1856 - acc: 0.9275 - val_loss: 0.2402 - val_acc: 0.9020 - 25s/ep
Epoch 29/30
100/100 - 25s - loss: 0.1815 - acc: 0.9300 - val_loss: 0.2395 - val_acc: 0.9030 - 25s/ep
Epoch 30/30
100/100 - 25s - loss: 0.1859 - acc: 0.9280 - val_loss: 0.2393 - val_acc: 0.9030 - 25s/ep
```

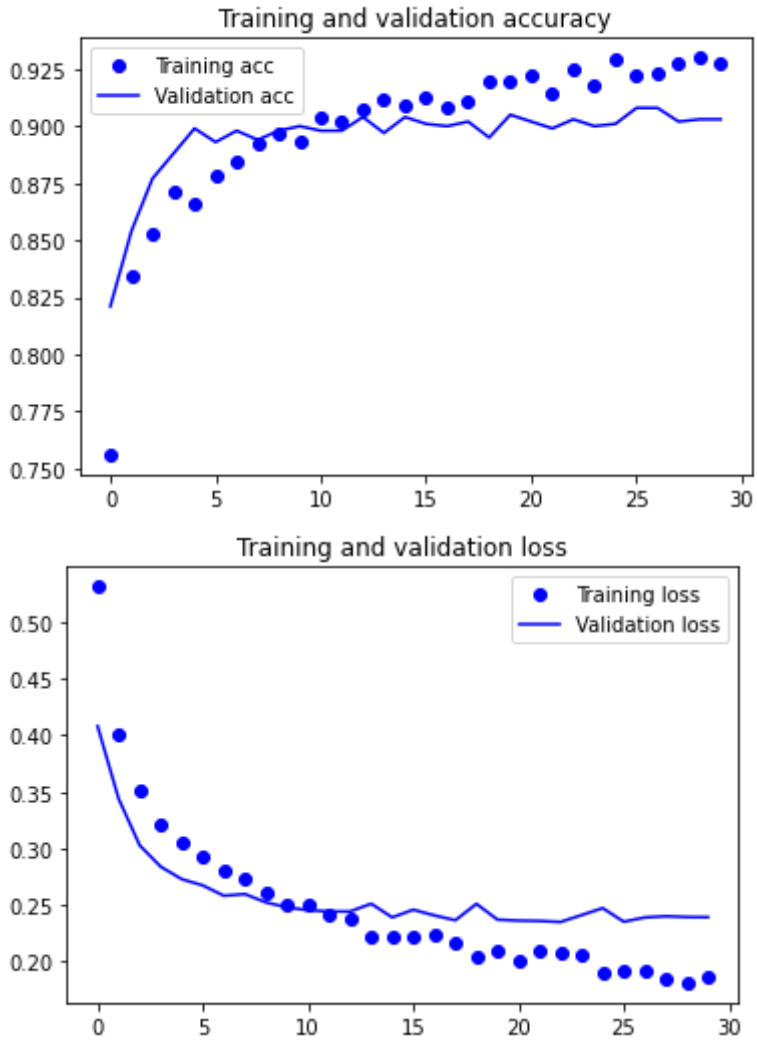
모델 저장

```
model.save('cats_dogs_small_3_dataAug_VGG_30_epoch.h5')
```

결과 시각화

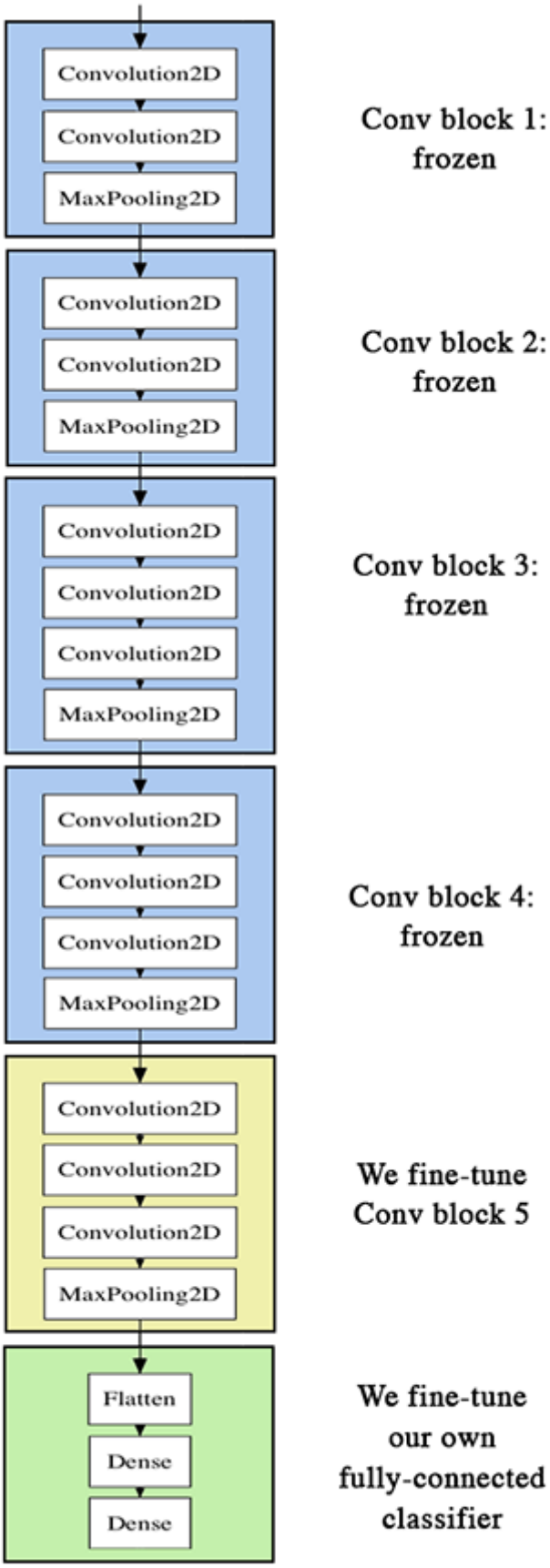
```
import matplotlib.pyplot as plt


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()
```



미세 조정 - 일부 훈련

- 모델을 재사용하는 데 널리 사용되는 또 하나의 기법은 특성 추출을 보완하는 미세 조정
- 미세 조정은 특성 추출에 사용했던 동결 **모델의 상위 층 몇 개를 동결에서 해제**한다.
    - 모델에 새로 추가한 층(여기서는 완전 연결 분류기)과 함께 훈련하는 것.
    - 주어진 문제에 조금 더 밀접하게 재사용 **모델의 표현을 일부 조정**하기 때문에 **미세 조정**이라고 부른다.



- 같은 이유로 맨 위에 있는 분류기가 훈련된 후에 합성곱 기반의 상위 층을 미세 조정 가능.
- 분류기가 미리 훈련되지 않으면 훈련되는 동안 너무 큰 오차 신호가 네트워크에 전파됩니다.

    - 미세 조정될 층들이 사전에 학습한 표현들을 망가뜨리게 될 것입니다.
    - 네트워크를 미세 조정하는 단계:
        - 01 사전에 훈련된 기반 네트워크 위에 새로운 네트워크를 추가합니다.
        - 02 훈련된 기반 네트워크를 동결합니다.
        - 03 새로 추가한 네트워크를 훈련합니다.
        - 04 기반 네트워크에서 일부 층의 동결을 해제합니다.
        - 05 동결을 해제한 층과 새로 추가한 층을 함께 훈련합니다.
- 처음 세 단계는 특성 추출을 할 때 이미 완료했습니다.
- 네 번째 단계를 진행해 보죠. conv_base의 동결을 해제하고 개별 층을 동결하겠습니다.

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (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: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

- 마지막 세 개의 합성곱 층을 미세 조정하겠습니다.
    - 즉, block4_pool까지 모든 층은 동결
    - block5_conv1, block5_conv2, block5_conv3 층은 학습 대상

```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False


%%time

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['acc'])


history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,  # total_train // batch_size
      epochs=50,
      validation_data=validation_generator,
      validation_steps=50,  # total_val // batch_size
      verbose=1)
```

```
      100/100 [==============================] - 26s 264ms/step - loss: 0.0185 - acc: 0.9940 -
      Epoch 24/50
      100/100 [==============================] - 27s 265ms/step - loss: 0.0216 - acc: 0.9920 -
      Epoch 25/50
      100/100 [==============================] - 26s 264ms/step - loss: 0.0104 - acc: 0.9975 -
      Epoch 26/50
      100/100 [==============================] - 27s 266ms/step - loss: 0.0145 - acc: 0.9955 -
      Epoch 27/50
      100/100 [==============================] - 27s 267ms/step - loss: 0.0147 - acc: 0.9960 -
      Epoch 28/50
      100/100 [==============================] - 27s 270ms/step - loss: 0.0131 - acc: 0.9975 -
      Epoch 29/50
      100/100 [==============================] - 27s 269ms/step - loss: 0.0113 - acc: 0.9955 -
      Epoch 30/50
      100/100 [==============================] - 27s 268ms/step - loss: 0.0119 - acc: 0.9950 -
      Epoch 31/50
      100/100 [==============================] - 27s 268ms/step - loss: 0.0117 - acc: 0.9955 -
```

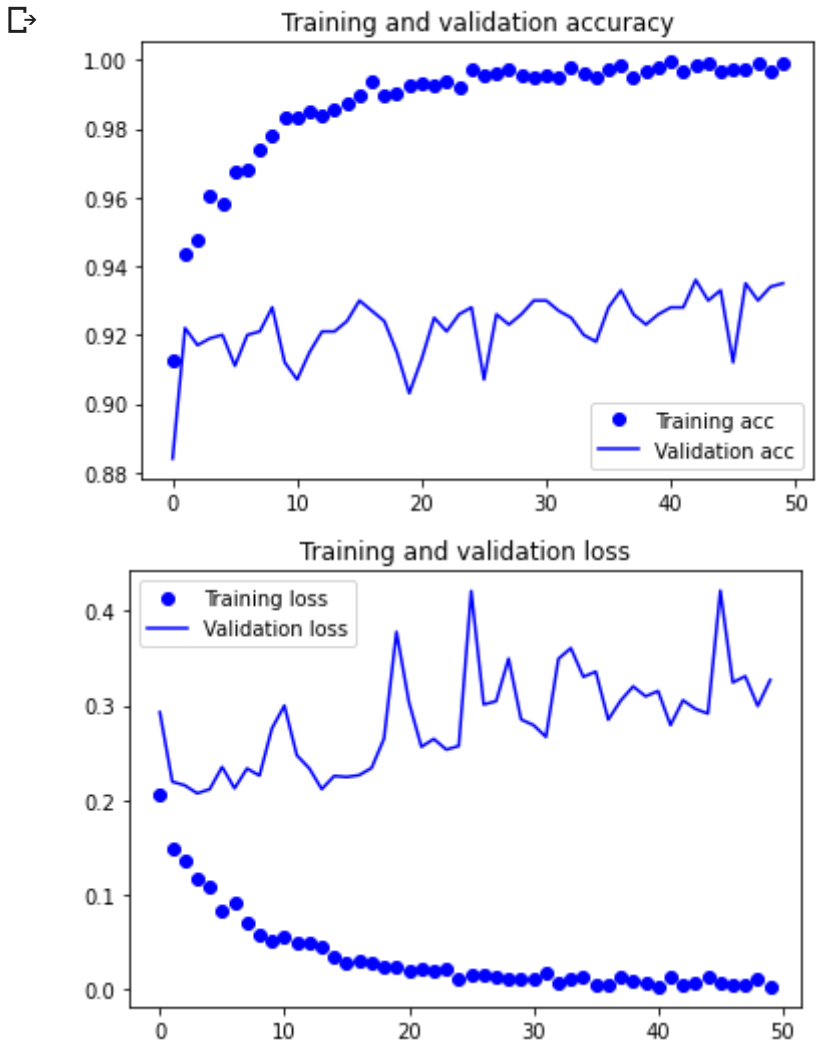
```
Epoch 32/50
100/100 [==============================] - 27s 267ms/step - loss: 0.0168 - acc: 0.9950 -
Epoch 33/50
100/100 [==============================] - 27s 267ms/step - loss: 0.0070 - acc: 0.9980 -
Epoch 34/50
100/100 [==============================] - 27s 266ms/step - loss: 0.0106 - acc: 0.9960 -
Epoch 35/50
100/100 [==============================] - 27s 269ms/step - loss: 0.0135 - acc: 0.9950 -
Epoch 36/50
100/100 [==============================] - 27s 268ms/step - loss: 0.0054 - acc: 0.9975 -
Epoch 37/50
100/100 [==============================] - 27s 267ms/step - loss: 0.0051 - acc: 0.9985 -
Epoch 38/50
100/100 [==============================] - 27s 267ms/step - loss: 0.0121 - acc: 0.9950 -
Epoch 39/50
100/100 [==============================] - 26s 263ms/step - loss: 0.0087 - acc: 0.9965 -
Epoch 40/50
100/100 [==============================] - 27s 266ms/step - loss: 0.0067 - acc: 0.9980 -
Epoch 41/50
100/100 [==============================] - 27s 265ms/step - loss: 0.0031 - acc: 0.9995 -
Epoch 42/50
100/100 [==============================] - 27s 265ms/step - loss: 0.0128 - acc: 0.9965 -
Epoch 43/50
100/100 [==============================] - 26s 264ms/step - loss: 0.0055 - acc: 0.9985 -
Epoch 44/50
100/100 [==============================] - 27s 265ms/step - loss: 0.0061 - acc: 0.9990 -
Epoch 45/50
100/100 [==============================] - 27s 266ms/step - loss: 0.0129 - acc: 0.9965 -
Epoch 46/50
100/100 [==============================] - 27s 266ms/step - loss: 0.0061 - acc: 0.9975 -
Epoch 47/50
100/100 [==============================] - 27s 265ms/step - loss: 0.0045 - acc: 0.9975 -
Epoch 48/50
100/100 [==============================] - 26s 264ms/step - loss: 0.0038 - acc: 0.9990 -
Epoch 49/50
100/100 [==============================] - 26s 264ms/step - loss: 0.0109 - acc: 0.9965 -
Epoch 50/50
100/100 [==============================] - 26s 263ms/step - loss: 0.0026 - acc: 0.9990 -
CPU times: user 25min 5s, sys: 1min 6s, total: 26min 11s
Wall time: 22min 52s
```

```
## 모델 저장
model.save('cats_and_dogs_small_4_misetunning_50.h5')
```

시각화

```
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()
```



모델 평가

```
test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
print('test acc:', test_acc)
```

```
Found 1000 images belonging to 2 classes.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning: `Model.evaluate_
  import sys
test acc: 0.9359999895095825
```