5장 컴퓨터비전을 위한 딥러닝

1조 다음은 2조

목차

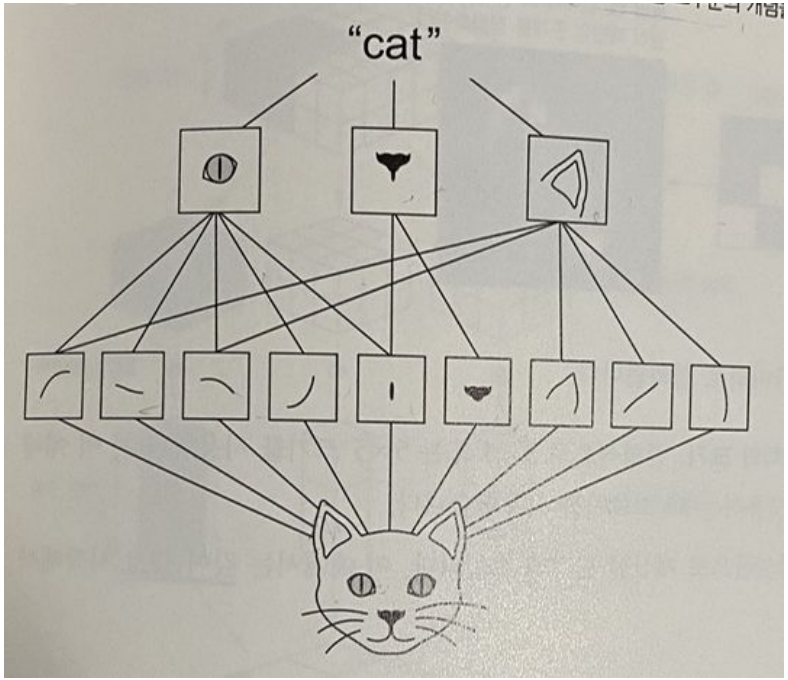
5.1.1 합성곱 연산(conv2D) 이란?

1) 비교

- Dense층은 전역패턴 학습 != 합성곱(지역패턴 학습)

2) 특징

- 평행이동 불변성
  : 학습된 패턴은 다른 지역에서도 인식 가능
- 패턴의 공간적 계층구조 학습
  : 앞의 합성곱 층의 패턴으로 더 큰 패턴을 학습



특성맵과 응답맵

1) 특성맵

: 깊이 축에 있는 각 차원은 하나의 특성 ==> (필터)

2) 응답 맵

: 입력의 위치에서 각 패턴(특성맵)에 대한 2D 맵

합성곱 과정

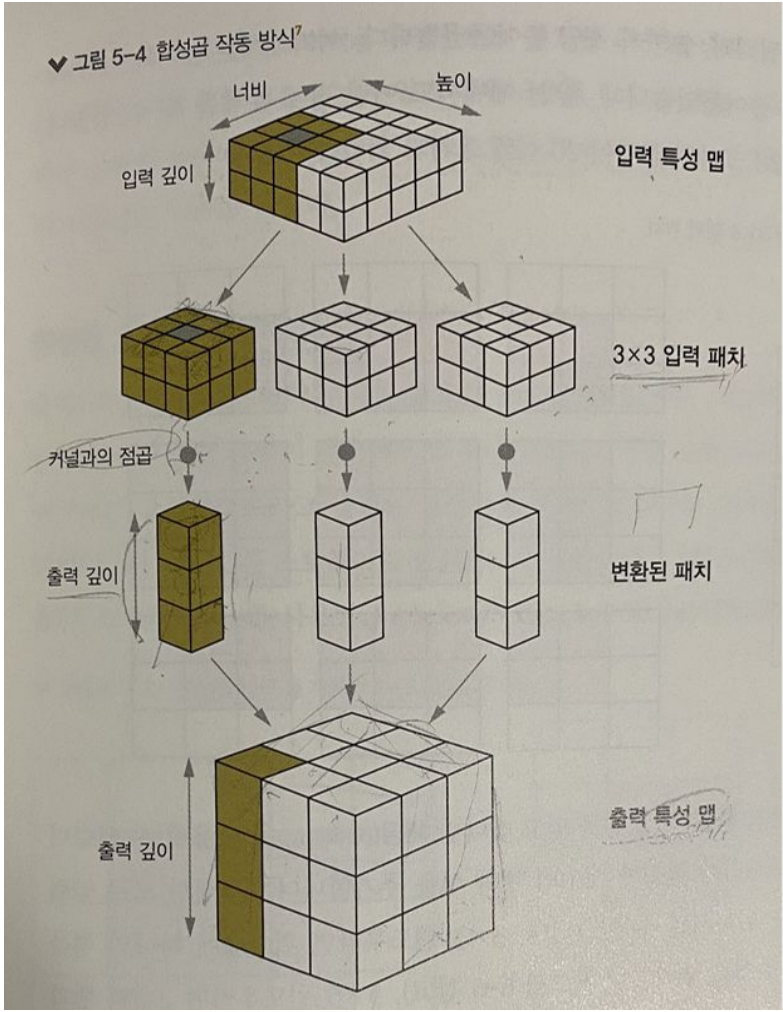
: 특성맵 입력 ->

(3* 3또는5 * 5) 정도의 윈도우만큼의 크기만큼 검사 ->

출력깊이 크기의 1차원 벡터로 변환 ->

1차원 벡터들 모두 합하면 (높이,너비,출력 깊이) 크기의

3차원 맵으로 재구성



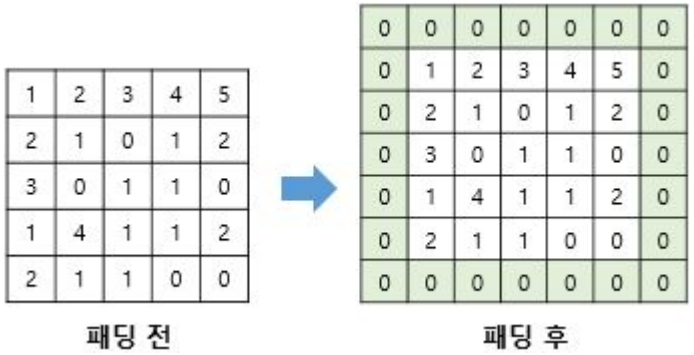
경계문제와 패딩

1) 경계문제

: 윈도우의 크기에 따라 입력 특성맵과 응답맵의 타일의 개수가 차이가 남 => 패딩으로 해결

2) 패딩

: 타일의 개수가 동일하게 맞추도록 입력특성맵에 행,열 추가



예) padding 매개변수로 사용 (valid : 패딩사용 안함, same : 입력특성맵과 출력맵의 크기를 맞춘다)

5.1.2 최대풀링 연산(Maxpooling2D)

1) 최대풀링 연산(Maxpooling2D) 이란?

: 강제적으로 특성맵을 다운 샘플링

2) 다운 샘플링의 이유?

- 처리할 특성맵의 가중치 개수 줄이기 -> 과대적합 방지
- 원본 입력 면에서 윈도우가 적용되는 범위가 점점 커짐 -> 공간적 계층 학습 가능

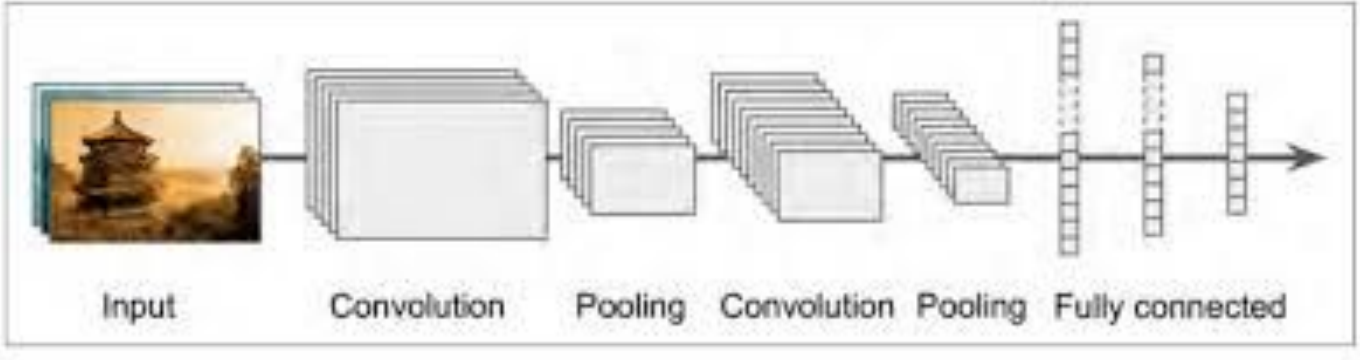
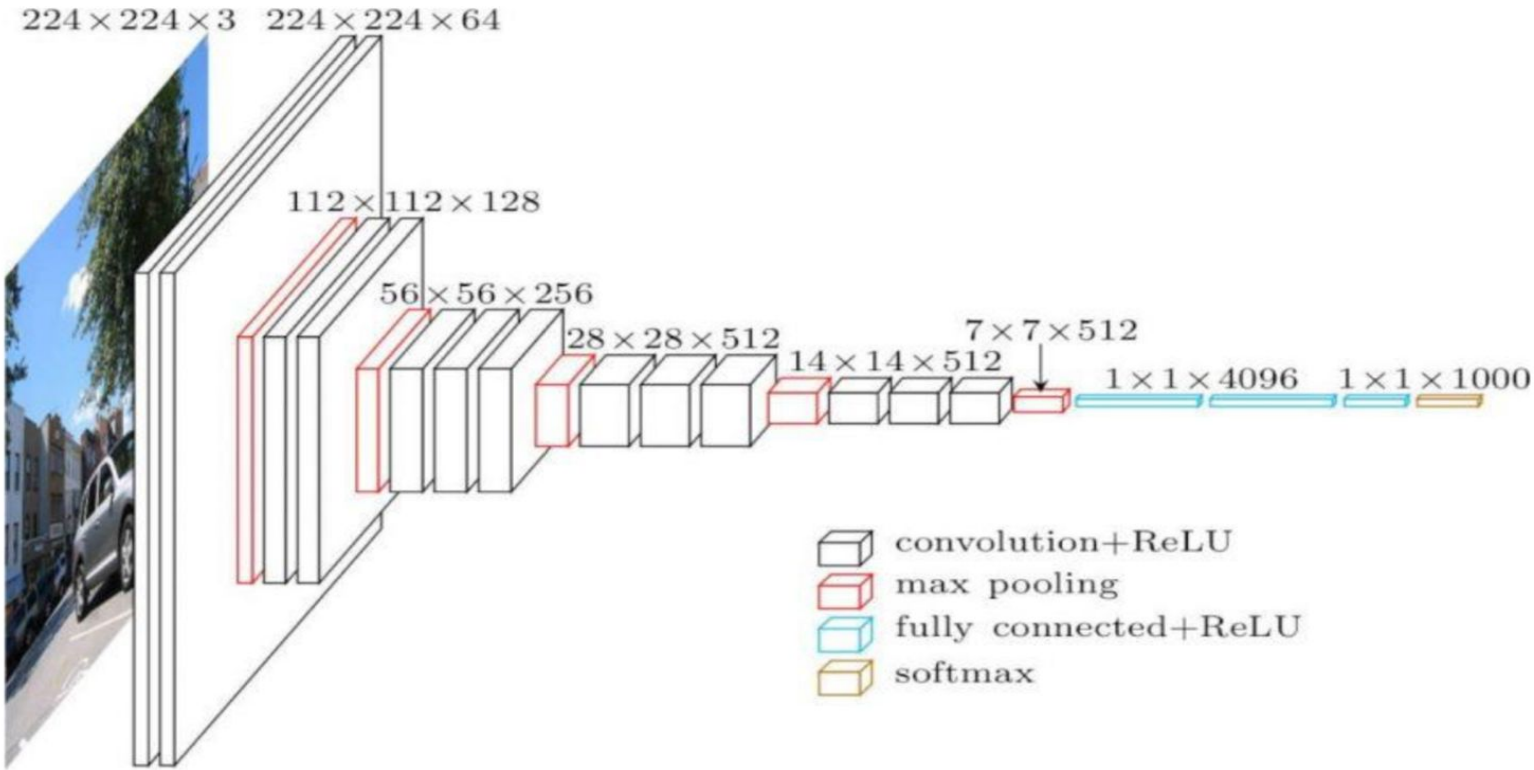


Figure 13-9. Typical CNN architecture

합성곱 신경망



초창기 신경망인 VGG-16보다 더 발전된, 아직 현역인 신경망 구조를 알아보자.

합성곱, 최대풀링 예제

```
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'))
```

- Conv2D, MaxPooling2D 층 사용
- mnist 이미지 포맷이(28,28,1) 크기로 input_shape() 사용

텐서 변환 예제

Conv2D와 MaxPooling2D 층의 3차원 텐서 -> 1차원 Dense층에 넣어야 한다

Flatten() 층으로 변환

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

주목할 점

- (3, 3, 64)의 3차원 텐서가 (576) 1차원 텐서로 변환됨
- 마지막 10개의 클래스로 분류

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
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 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

5.2.1 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련하기

- '적은' 샘플이란 수백 개에서 수만 개 사이를 의미합니다. 실용적인 예제로 4,000개의 강아지와 고양이 사진(2,000개는 강아지, 2,000개는 고양이)으로 구성된 데이터셋에서 강아지와 고양이 이미지를 분류해 보겠습니다. 훈련을 위해 2,000개의 사진을 사용하고 검증과 테스트에 각각 1,000개의 사진을 사용
- 세 가지 전략(처음부터 작은 모델을 훈련하기, 사전 훈련된 모델을 사용해 특성 추출하기, 사전 훈련된 모델을 세밀하게 튜닝하기)

5.2.2 작은 데이터셋 문제에서 딥러닝의 타당성¶

- 딥러닝의 근본적인 특징은 훈련 데이터에서 특성 공학의 수작업 없이 흥미로운 특성을 찾을 수 있는 것입니다. 이는 훈련 샘플이 많아야만 가능합니다. 입력 샘플이 이미지와 같이 매우 고차원인 문제에서는 특히 그렇습니다.
- 대규모 데이터셋에서 훈련시킨 이미지 분류 모델이나 스피치-투-텍스트 모델을 조금만 변경해서 완전히 다른 문제에 재사용
- 컴퓨터 비전에서는 (보통 ImageNet 데이터셋에서 훈련된) 사전 훈련된 모델들이 다운로드받을 수 있도록 많이 공개되어 있어서 매우 적은 데이터에서 강력한 비전 모델을 만드는데 사용할 수 있음

5.2.3 데이터 내려받기

```
# 원본 데이터셋을 압축 해제한 디렉터리 경로
original_dataset_dir = '/content/drive/My Drive/original_dataset/trainphoto'

# 소규모 데이터셋을 저장할 디렉터리
base_dir = '/content/drive/My Drive/dataset/'
if os.path.exists(base_dir):  # 반복적인 실행을 위해 디렉토리를 삭제합니다.
    shutil.rmtree(base_dir)   # 이 코드는 책에 포함되어 있지 않습니다.
os.mkdir(base_dir)

# 훈련, 검증, 테스트 분할을 위한 디렉터리
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_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
validation_cats_dir = os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)

# 검증용 강아지 사진 디렉터리
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.mkdir(validation_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(original_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(original_dataset_dir, fname)
    dst = os.path.join(validation_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(original_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(original_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(original_dataset_dir, fname)
    dst = os.path.join(validation_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

5.2.3 데이터 내려받기

```
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(validation_cats_dir)))
print('검증용 강아지 이미지 전체 개수:', len(os.listdir(validation_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
```

5.2.3 네트워크 구성하기

- 네트워크를 좀 더 크게 구성
- conv2D+MaxPooling2D 단계를 하나 더 추가
- 이렇게 하면 네트워크의 용량을 늘리고 Flatten 층의 크기가 너무 커지지 않도록 특성 맵의 크기를 줄일 수 있음
- Note : 특성 맵의 깊이는 네트워크에서 점진적으로 증가하지만 (32~128), 특성 맵의 크기는 감소합니다.(150x150에서 7x7까지)

5.2.3 네트워크 구성하기

```
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
```

```
from keras import optimizers

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

5.2.4 데이터 전처리

1. 사진 파일을 읽습니다.
2. JPEG 콘텐츠를 RGB 픽셀 값으로 디코딩합니다.
3. 그다음 부동 소수 타입의 텐서로 변환합니다.
4. 픽셀 값(0에서 255 사이)의 스케일을 [0,1] 사이로 조정합

```
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(
        # 타깃 디렉터리
        train_dir,
        # 모든 이미지를 150 × 150 크기로 바꿉니다
        target_size=(150, 150),
        batch_size=20,
        # binary_crossentropy 손실을 사용하기 때문에 이진 레이블이 필요합니다
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_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.
```

5.2.4 데이터 전처리

```
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(
        # 타깃 디렉터리
        train_dir,
        # 모든 이미지를 150 × 150 크기로 바꿉니다
        target_size=(150, 150),
        batch_size=20,
        # binary_crossentropy 손실을 사용하기 때문에 이진 레이블이 필
        class_mode='binary')

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

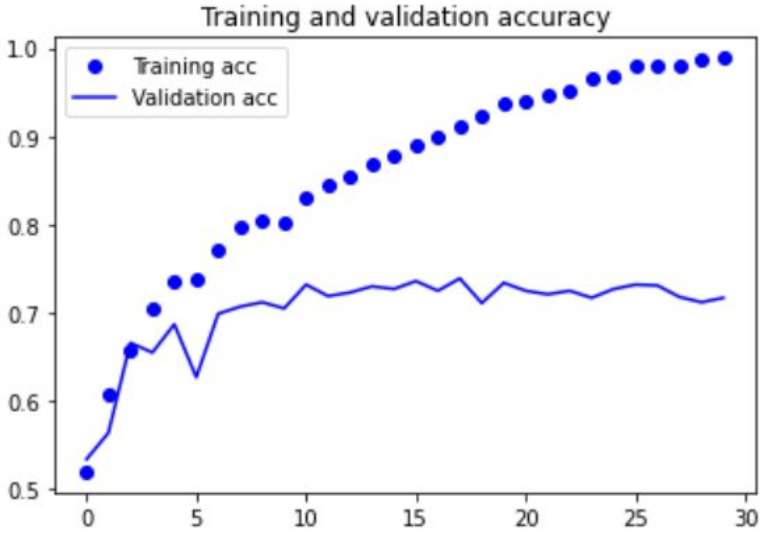
```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

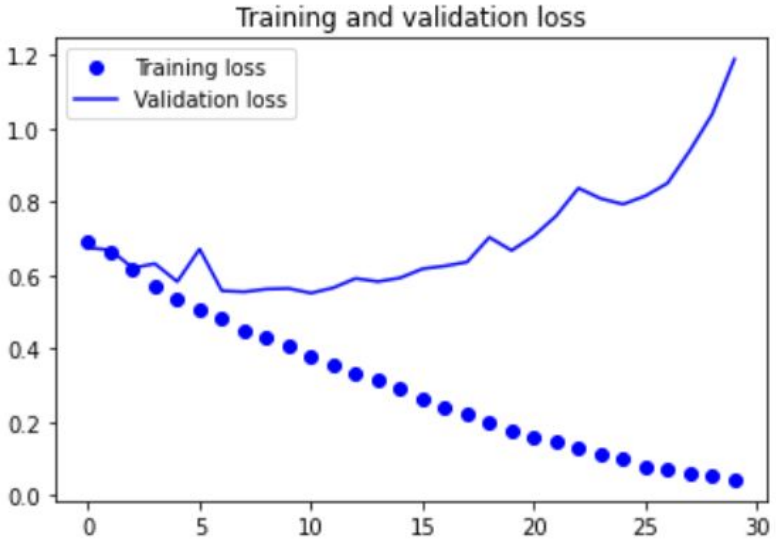
```
for data_batch, labels_batch in train_generator:
    print('배치 데이터 크기:', data_batch.shape)
    print('배치 레이블 크기:', labels_batch.shape)
    break
```

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

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

5.2.4 데이터 전처리


Training and validation accuracy
Training acc
Validation acc
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30


Training and validation loss
Training loss
Validation loss
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

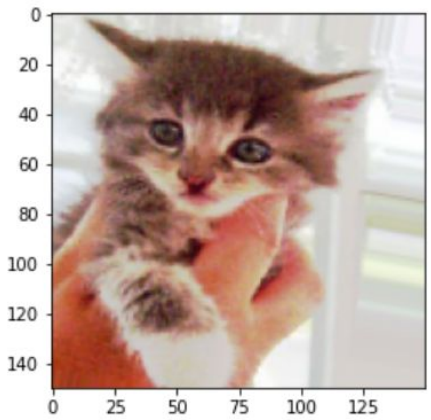
5.2.4 데이터 전처리

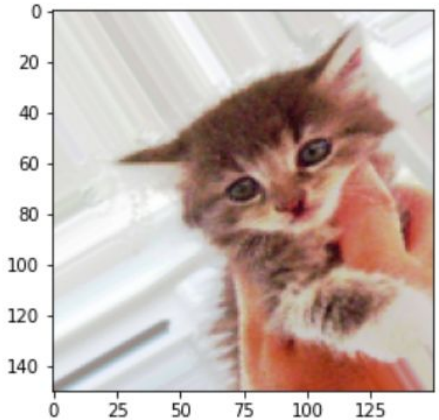
```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

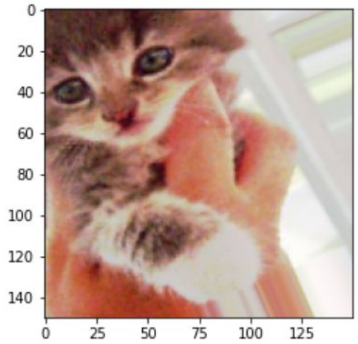
otation_range는 랜덤하게 사진을 회전시킬 각도 범위입니다(0-180 사이).
idth_shift_range와 height_shift_range는 사진을 수평과 수직으로 랜덤하게 평행 이동시킬 범위입니다(전체 넓이와 이에 대한 비율).
hear_range는 랜덤하게 전단 변환을 적용할 각도 범위입니다.
oom_range는 랜덤하게 사진을 확대할 범위입니다.
orizontal_flip은 랜덤하게 이미지를 수평으로 뒤집습니다. 수평 대칭을 가정할 수 있을 때 사용합니다(예를 들어, 풍경/ 물 사진).
ill_mode는 회전이나 가로/세로 이동으로 인해 새롭게 생성해야 할 픽셀을 채울 전략입니다.

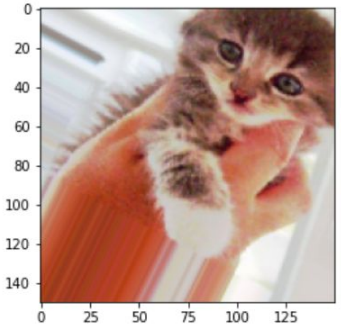
```
# 이미지 전처리 유틸리티 모듈
from keras.preprocessing import image

fnames = sorted([os.path.join(train_cats_dir, fname) for fname in os.listdir(train_cats_dir)])

# 증식할 이미지 선택합니다
img_path = fnames[3]

# 이미지를 읽고 크기를 변경합니다
img = image.load_img(img_path, target_size=(150, 150))

# (150, 150, 3) 크기의 넘파이 배열로 변환합니다
x = image.img_to_array(img)

# (1, 150, 150, 3) 크기로 변환합니다
x = x.reshape((1,) + x.shape)

# flow() 메서드는 랜덤하게 변환된 이미지의 배치를 생성합니다.
# 무한 반복되기 때문에 어느 지점에서 중지해야 합니다!
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break

plt.show()
```

5.2.4 데이터 전처리


0
20
40
60
80
100
120
140
0
25
50
75
100
125


0
20
40
60
80
100
120
140
0
25
50
75
100
125


0
20
40
60
80
100
120
140
0
25
50
75
100
125


0
20
40
60
80
100
120
140
0
25
50
75
100
125

5-2 모델 설정(드롭아웃 추가)

```
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.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

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

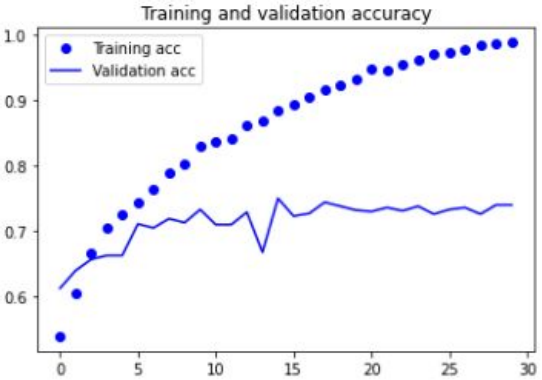
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 6272)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6272)              0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               3211776
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

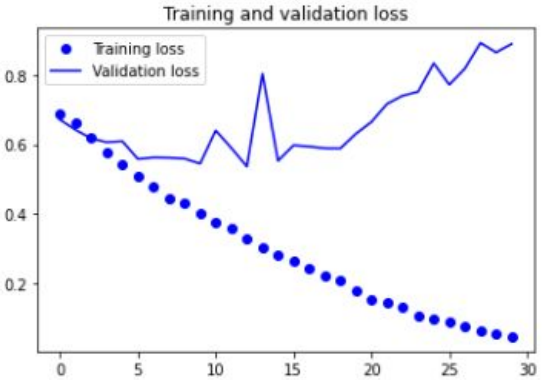
5-2 데이터 전처리

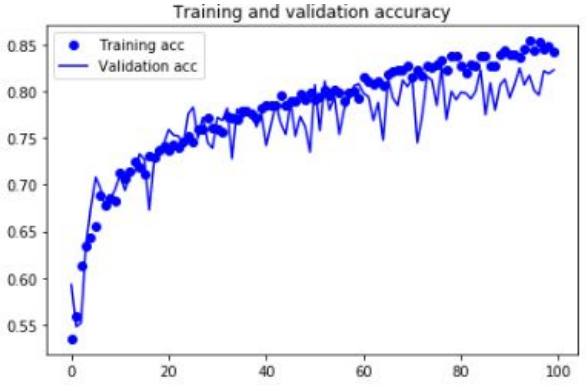
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)


test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
```

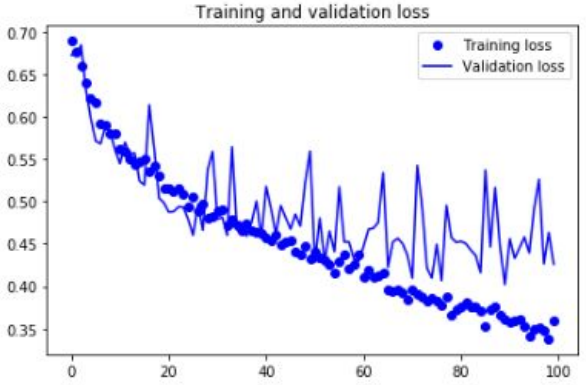
```
validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

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

5-2 결과 비교


Training and validation accuracy
Training acc
Validation acc

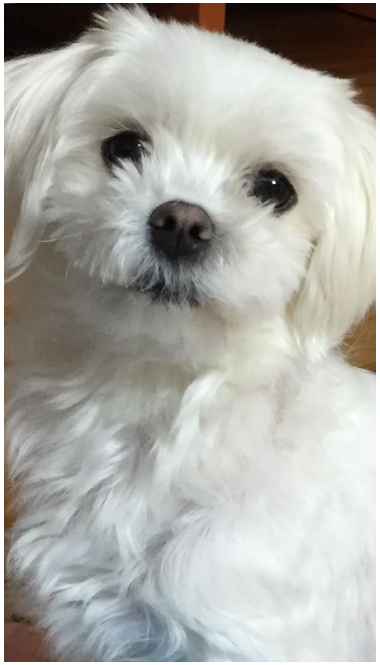

Training and validation loss
Training loss
Validation loss


Training and validation accuracy
Training acc
Validation acc

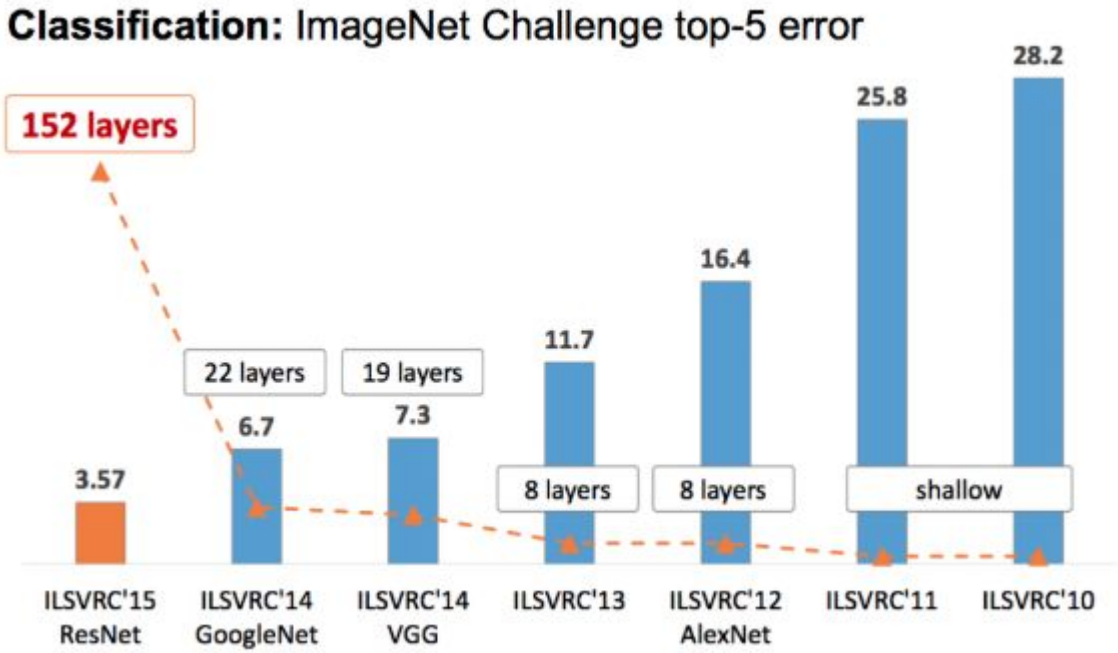

Training and validation loss
Training loss
Validation loss

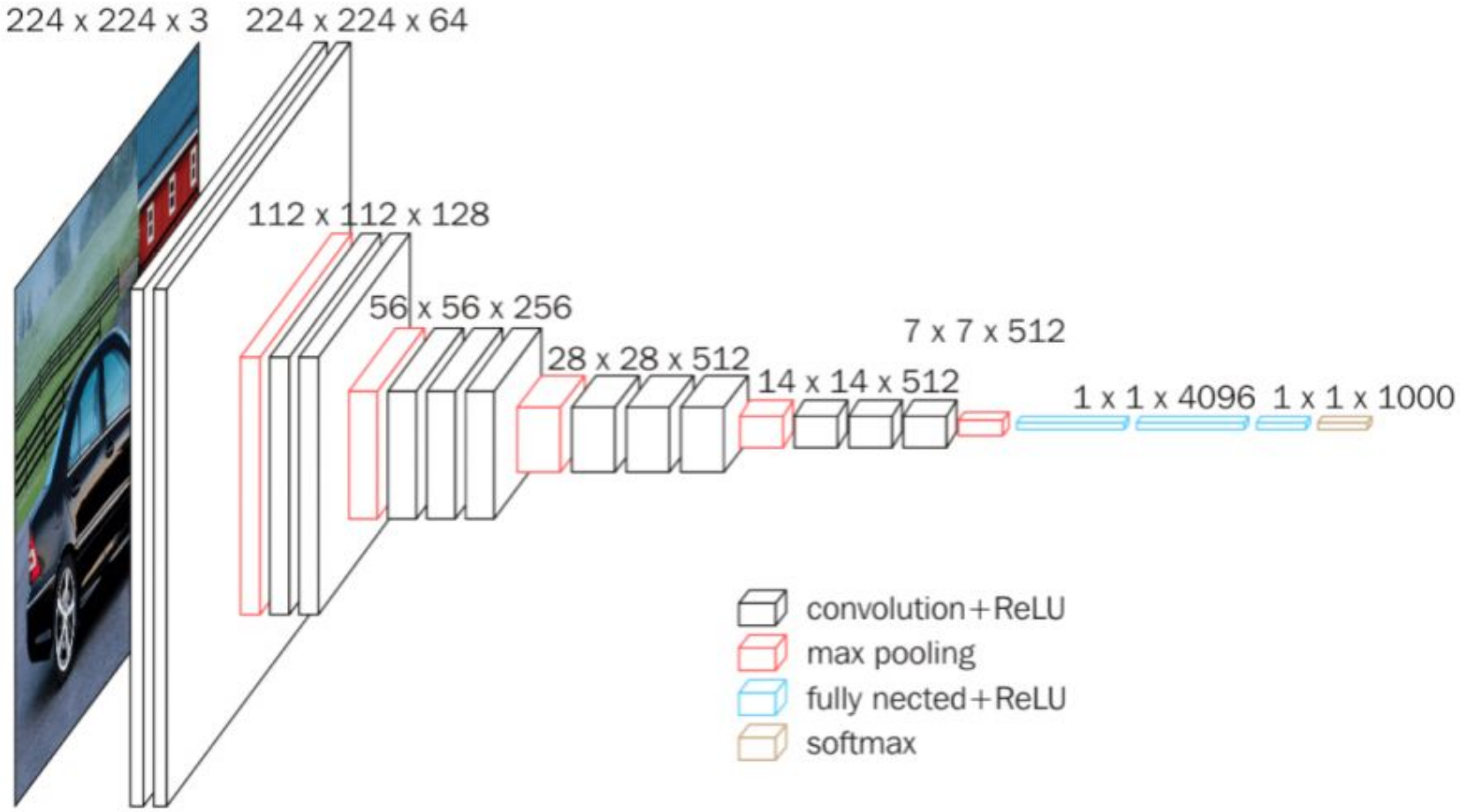
5.3 사전 훈련된 컨브넷 사용하기

or



5.3.1 특성추출 - VGG16




224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

5.3.1 특성추출 - 합성곱 기반 층

```
1 from keras.applications import VGG16
2
3 conv_base = VGG16(weights='imagenet', #가중치
4                   include_top=False, #완전 연결 분류기 포함 안 함
5                   input_shape=(150,150,3)) #텐서 크기
```

```
_________________________________________________________________
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
_________________________________________________________________
```

5.3.1 특성 추출 - 완전 연결 분류기(데이터 증식 X)

```
#완전 연결 분류기


from keras import models
from keras import layers
from 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))
```

5.3.1 특성 추출 - 데이터 증식 O

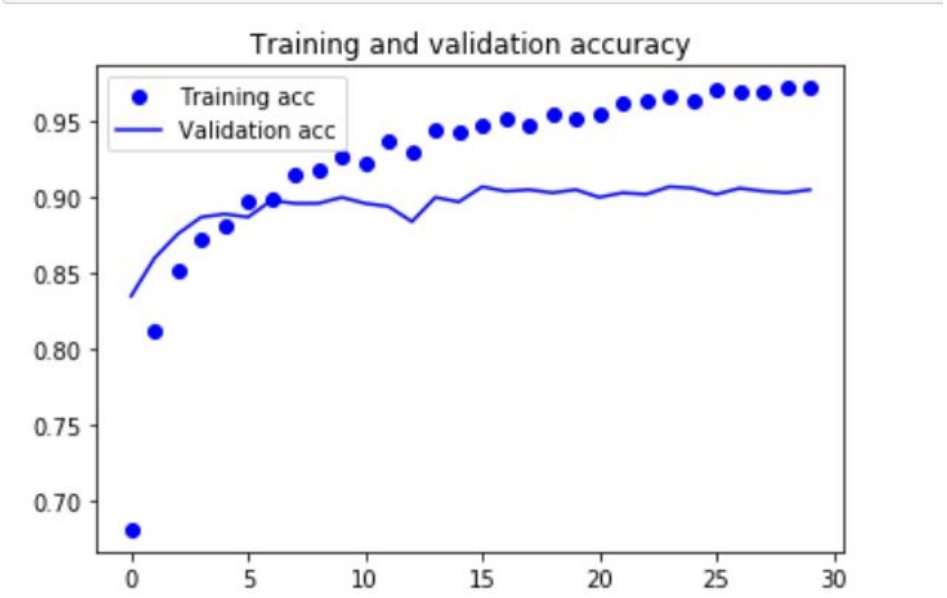
```
#연결분류기 추가
from keras import models
from keras import layers
model=models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

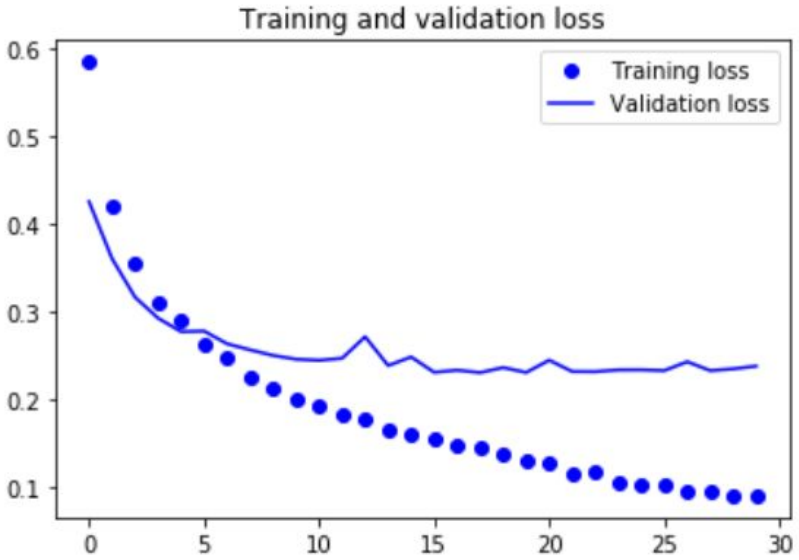
5.3.1 특성 추출 - 데이터 증식 O

```python
#엔드 투 엔드 훈련
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
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_datagedn = ImageDataGenerator(rescale=1./255)

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

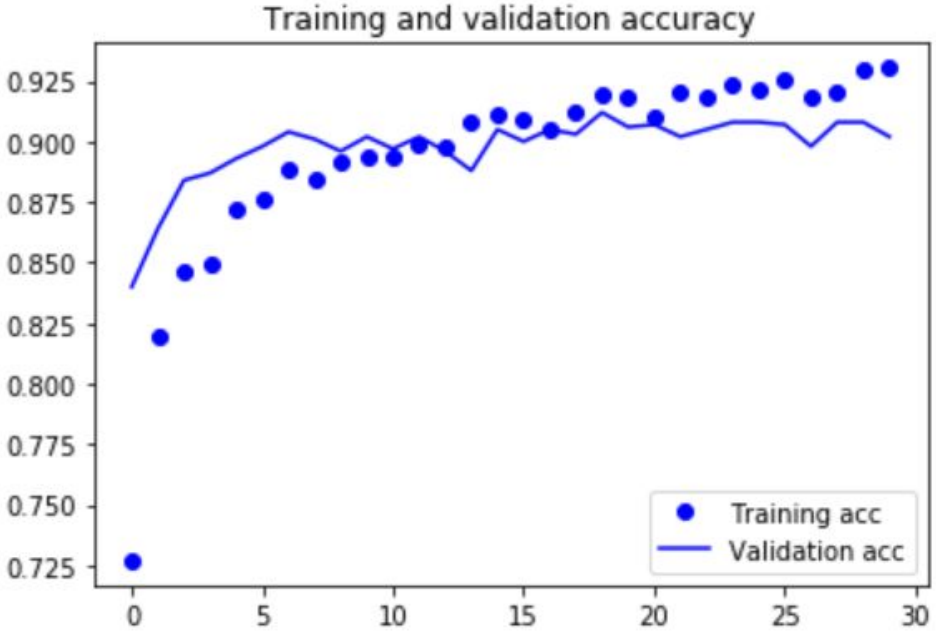
```python
validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150,150),
    batch_size=20,
    class_mode='binary'
)

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metric=['acc'])
history = model.fit(train_generator,
                    train_generator,
                    steps_per_epoch=30,
                    epochs=30,
                    validation_data=validation_generator,
                                    validation_steps=50,
                                    verbose=2)
```

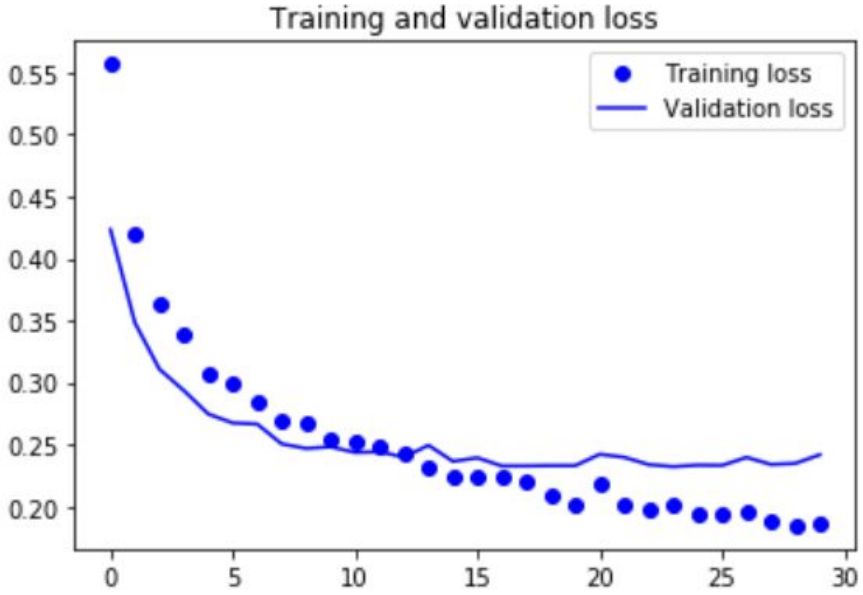
5.3.1 특성 추출 - 완전 연결 분류기(데이터 증식 X)





5.3.1 특성 추출 - 데이터 증식 O





미세조정

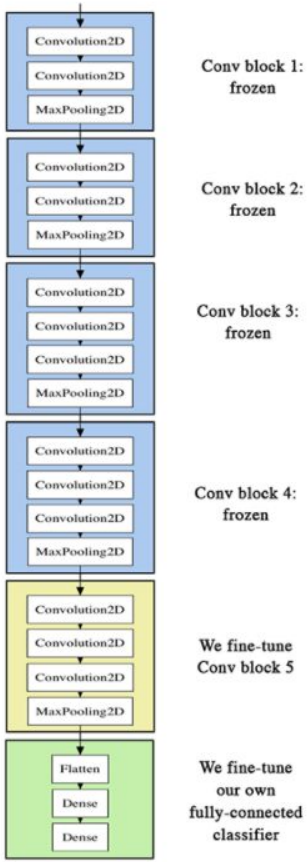
특성 추출과 더불어, 사전 훈련된 컨브넷 모델을 재사용하는데 사용

미세조정(Fine-tuning)

: 특성 추출에 사용했던 동결 모델의 상위 층 몇 개를 동결에서 해제 하고, 모델에 새로 추가한 층과 함께 훈련하는 것

주어진 문제에, 조금 더 밀접하게 재사용 모델의 표현을 일부 조정하기 때문에 미세조정이라고 한다.

미세조정



< 절차 >

1. 사전에 훈련된 기반 네트워크 위에 새로운 네트워크 추가
2. 기반 네트워크를 동결
3. 새로 추가한 네트워크를 훈련

=== 여기까지는 앞에서 설명한, 특성 추출과정 ===

4. 기반 네트워크에서 일부 층의 동결을 해제
5. 동결을 해제한 층과 새로 추가한 층을 함께 훈련

미세조정

•왜, 더 많은 층을 미세조정하지 않는가 ?

- 하위층 일수록, 일반적이고, 재사용 가능한 특성을 인코딩
  상위층 일수록, 좀 더 특성화(구체적인 특징)을 인코딩

  새로운 문제에 재활용해서 수정이 필요한 것은, 구체적인 특징 -> 상위층
  하위층으로 갈수록 미세조정에 대한 효과는 감소
- 훈련해야 할 파라미터가 많을수록 -> 과대적합 위험성 증가

•결론은, 이런 상황에선, 합성곱 기반층에서, 최상위 2~3개 층만 미세조정이 효과적

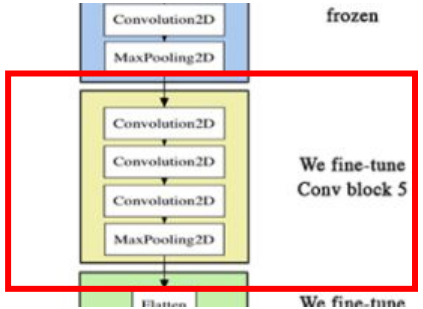
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(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

Total params: 14,714,688

미세조정

```
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
```

-> 예제에선, block-5를 동결해제




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

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

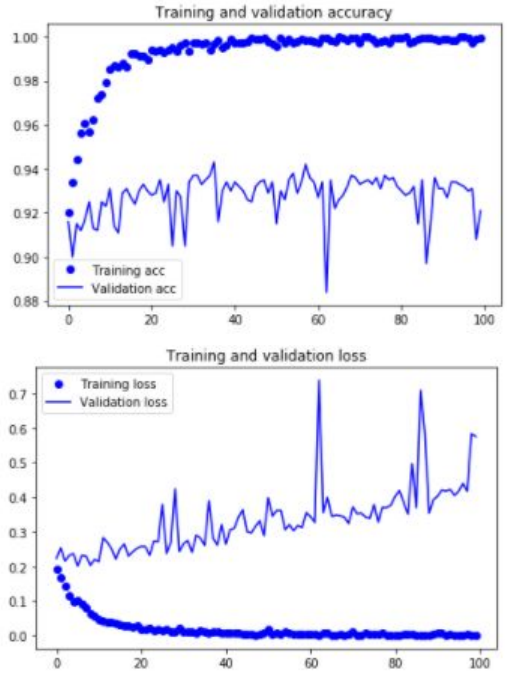
-> 실직적, 미세조정 시작

= 이때, 옵티마이저의 학습률을 낮춰서 사용 !!
미세조정하는 층의 이미 학습된 표현을, 조금씩 수정하기 위해서

학습률이 커서, 변경량이 크게 되면, 오히려 나쁜 영향을 끼칠 수 있다.

미세조정

그래프가, 이쁜 곡선 형태가 아니다.



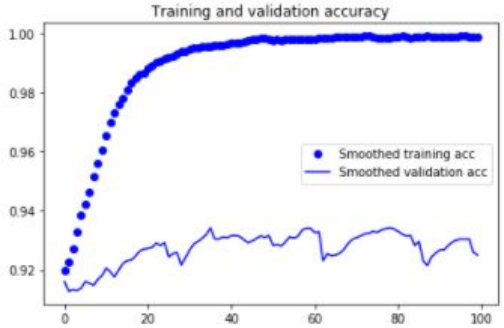
지수 이동 평균을 적용해, 정확도와 손실값 그래프를 부드럽게 표현할 수 있다.

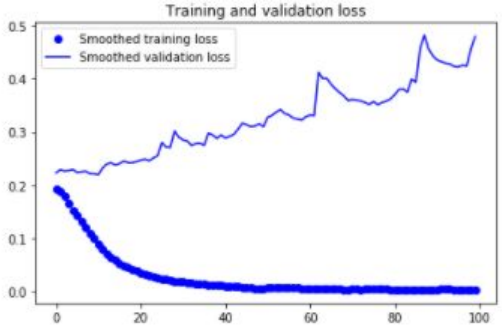
```
def smooth_curve(points, factor=0.8):
  smoothed_points = []
  for point in points:
    if smoothed_points:
      previous = smoothed_points[-1]
      smoothed_points.append(previous * factor + point * (1 - factor))
    else:
      smoothed_points.append(point)
  return smoothed_points
```

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

plt.figure()

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

plt.show()
```

미세조정

지수 이동 평균을 적용한, 미세조정 모델 성능





테스트 데이터를 통한, 최종 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.
test acc: 0.9169999933242798
```