9. 합성곱 신경망

인공 신경망은 파라미터만 잘 조정해주면 매우 다양한 함수를 나타낼 수 있다. 그래서 우리가 원하는 결과를 얻기 위해 모형의 형태에 제약을 주기도 한다. 이러한 방법 중에 대표적인 것으로 합성곱 신경망과 순환신경망이 있다.

이미지를 기계학습으로 처리할 경우, 이미지의 점 하나가 변수가 된다. 가로 28 X 세로 28인 이미지가 있다면, 이 이미지는 784개의 점으로 이뤄져 있다. 즉 784개의 변수로된 데이터를 분석하는 것과 같게 된다.

이미지는 점들이 모여 작은 형태를 이루고, 작은 형태들이 모여 더 큰 형태를 이루는 특징이 있다. 따라서 이미지들을 단순히 여러 개의 점들로 보는 것보다, 이런 점들이 모여 만드는 형태들을 모형에 포함시킨다면 이미지 기계학습의 성능을 향상시킬 수 있다. 실제로 생물의 시각도 같은 방식으로 작동한다. 생물에서 작은 영역에 반응하는 세포들을 receptive field라고 부른다. 여기에서 착안한 것이 합성곱 신경망(convolutional neural network, CNN)이다.

9.1. 합성곱

합성곱(convolution)은 원래 이미지 처리에 많이 사용되는 방법이다.예를 들어 다음과 같은 이미지에

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

아래와 같은 필터를

$$\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

합성곱 해보자. 위의 필터는 수직선에 반응하는 비슷한 기능을 한다.

계산은 다음과 같이 이뤄진다. 먼저 왼쪽 위의 3x3 부분만을 본다.

$$\begin{bmatrix} 1 & 0 & \cdot \\ 1 & 0 & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

그 다음 같은 위치에 있는 값들끼리 곱해준다.

$$\begin{bmatrix} 1 \times 1 & -1 \times 0 \\ 1 \times 1 & -1 \times 0 \end{bmatrix}$$

그러면 계산 결과가 다음과 같다.

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

마지막으로 이들의 합을 구하면 2가 된다.

다음으로는 한 칸 오른쪽으로 옮겨서 다음 3x3 부분에

$$\begin{bmatrix} \cdot & 0 & 0 \\ \cdot & 0 & 0 \\ \cdot & \cdot & \cdot \end{bmatrix}$$

똑같은 계산을 적용하면 0이 된다. 이렇게 계산을 반복하고 그 결과를 아래와 같은 feature map으로 만든다.

$$\begin{bmatrix} 2 & 0 \\ 1 & 0 \end{bmatrix}$$

위의 행렬을 보면 그림의 왼쪽 위가 수직선 패턴이 가장 강하다는 것을 알 수 있다. 이번에는 수평선 패턴에 반응하는 아래와 같은 필터를

$$\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$$

똑같은 이미지에 합성곱을 하면 다음과 같은 feature map을 만들 수 있다.

$$\begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix}$$

위의 행렬을 보면 그림의 오른쪽 아래가 수평선 패턴이 가장 강하다는 것을 알 수 있다.

이런 식으로 다양한 필터를 만들어 합성곱을 해주면 이미지에서 특정 패턴을 추출할 수 있다. 합성곱은 패턴 추출에만 사용되는 것이 아니고 흐리게 하기(blur)나 날카롭게 하기(sharpen) 등 이미지 편집에 사용되는 여러 필터 적용에도 쓸 수 있다.

합성곱 신경망에서 필터 역할을 하는 레이어를 합성곱 레이어라고 한다. 기존의 이미지 처리가 미리 만들어진 필터를 적용하는 방식이라면, CNN은 이러한 필터 자체를 학습한다는 것이 특징이다. 필터라는 것은 결국 그림의 일부에 일정한 계수를 곱해주고 그 결과를 더하는 것이로 일반적인 신경망의 구조와 다를 것이 없다. 다만, 같은 계수를 그림의 부분들에 반복적으로 적용한다는 점만 차이가 있을 뿐이다.

9.2. Pooling

합성곱을 하면 feature map에서 근처의 값들끼리 비슷해지는 경향을 띈다. 그래서 근방의 영역에 있는 값들을 묶어 하나의 값으로 줄이는 pooling을 한다. pooling의 방법에는 다음과 같은 것들이 있다.

- max: 영역의 최대값
- average: 영역의 평균
- L2: 영역의 제곱합의 제곱근

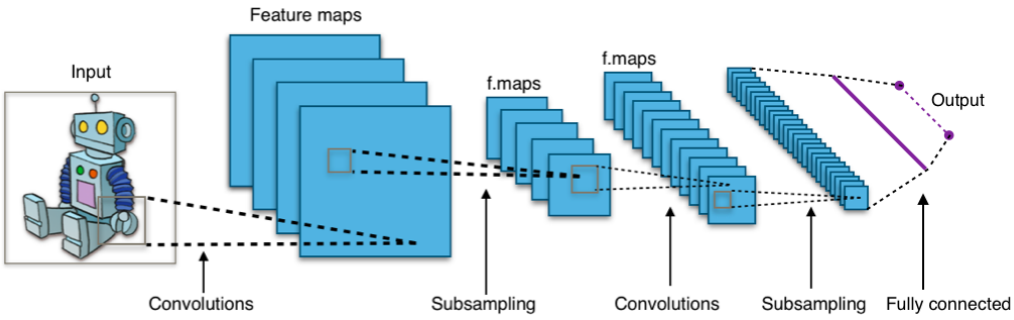
앞에서 예로 들었던 feature map에 2x2 max pooling을 적용하면 똑같이 2가 나오게 된다.

따라서 우리는 정확한 위치는 알 수 없지만 3x3 그림에 수직선과 수평선이 하나 있다는 것을 알 수 있게 된다. 이러한 pooling은 정보를 줄이게 되지만, 과적합도 줄어들 뿐만 아니라 그림이 일부 변형되더라도 학습결과를 유지할 수 있게 해준다. 왜냐하면 그림이 수직이나 수평 방향으로 1픽셀 이동하더라도 pooling을 거친 결과는 동일하기 때문이다.

합성곱 레이어와 달리 풀링 레이어는 별도의 학습이 이뤄지지 않는다.

9.3. 합성곱 신경망

이제 이미지를 입력 받아 그 위로 합성곱 레이어와 풀링 레이어를 반복해서 쌓으면 CNN이 된다. 두 가지 레이어를 반복해서 쌓는 이유는 작은 형태를 바탕으로 다시 큰 형태를 처리하게 하기 위해서다.



9.4. CNN 실습

```
from keras.preprocessing.image import ImageDataGenerator

img_gen = ImageDataGenerator(
        rescale=1./255,          # 픽셀 값을 0~1 범위로 변환
        rotation_range=40,       # 40도까지 회전
        width_shift_range=0.2,   # 20%까지 좌우 이동
        height_shift_range=0.2, # 20%까지 상하 이동
        shear_range=0.2,         # 20%까지 기울임
        zoom_range=0.2,          # 20%까지 확대
        horizontal_flip=True,    # 좌우 뒤집기
    )

train = img_gen.flow_from_directory(
    'dog-vs-cat/train',      # 이미지 디렉토리
    target_size=(100, 100), # 변환할 크기는 가로 100, 세로 100
    color_mode='rgb',        # 컬러는 rgb, 흑백은 grayscale. 생략하면 컬러로 처리한다
    class_mode='binary')     # 고양이 vs. 개로 binary 분류
```

```
valid = ImageDataGenerator(rescale=1.0/255).flow_from_directory(
    'dog-vs-cat/validation',
    target_size=(100, 100),
    class_mode='binary',
    shuffle=False)
```

9.4.1. 모형

```
from keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D
from keras.models import Sequential
```

첫번째 레이어는 5x5 형태의 합성곱 레이어를, 두번째 레이어는 3x3 형태의 합성곱 레이어를 사용하는 CNN을 만든다. 32는 필터의 종류를 32개로 한다는 뜻이다. 따라서 가로 100 x 세로 100의 이미지가 첫 레이어를 거치면 가로 96 x 세로 96의 feature map 32개가 된다.

```
model = Sequential()
model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(100, 100, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

```
model.summary()
```

```
from keras.optimizers import Adam, RMSprop
```

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

학습 기록을 `log_cnn` 디렉토리로 저장하고 텐서보드로 모니터한다.

```
from keras.callbacks import ModelCheckpoint, TensorBoard
```

```
model.fit_generator(
    train, validation_data=valid, epochs=5,
    callbacks=[
        TensorBoard(log_dir='log_cnn'),
        ModelCheckpoint('cnn-{epoch:02d}.hdf5', save_best_only=True),
    ])
```

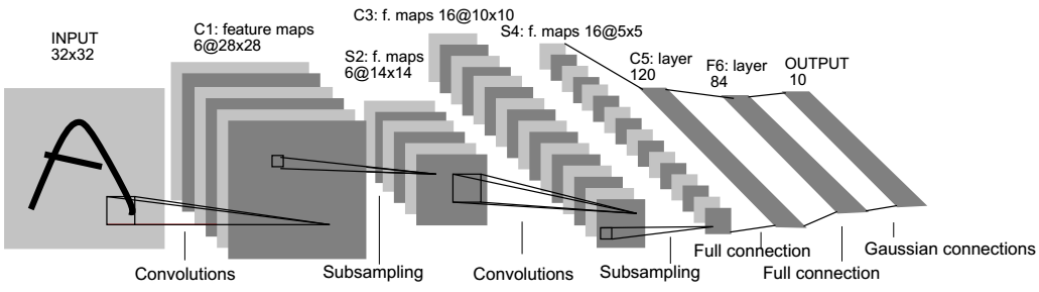
```
<keras.callbacks.History at 0x7f5ff70c33c8>
```

작업 디렉토리 명령창에 `tensorboard --host=127.0.0.1 --logdir=log_cnn` 이라고 입력하여 텐서보드를 실행하고 웹 브라우저 주소창에 http://127.0.0.1:6006 또는 http://localhost:6006 으로 텐서보드에 접속한다.

9.5. CNN의 주요 모형

9.5.1. LeNet-5

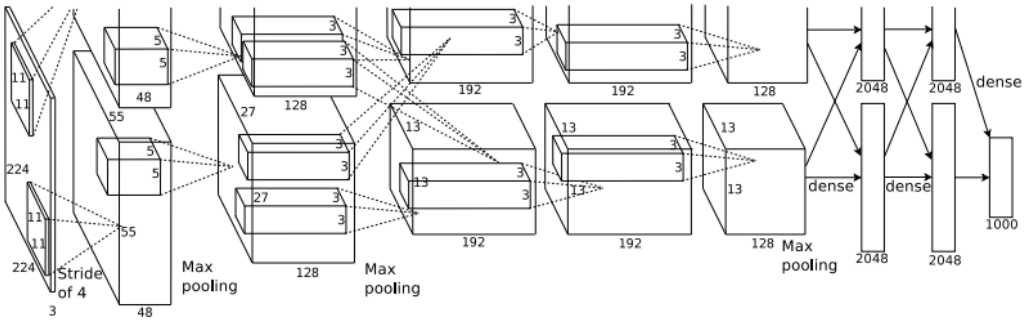
CNN은 1989년 얀 르쿤에 의해 처음 제안되었다. 르쿤이 고안한 CNN 구조를 LeNet-5라고 한다. LeNet-5은 합성곱 레이어와 다운샘플링 레이어를 번갈아가며 사용하고 그 뒤로 `keras` 의 `Dense` 에 해당하는 Full Connection(FC) 레이어들이 있는 구조이다.



9.5.2. AlexNet

ImageNet은 물체 인식(object recognition) 연구를 위한 데이터셋으로 약 2만종의 물체를 담은 1천 4백만개의 사진으로 이뤄져 있다. ILSVRC는 이 ImageNet 데이터셋을 바탕으로 하는 기계학습 대회다. 이 대회는 Top-5 오류율(error rate)라는 성능 지표를 쓴다. 이는 모형이 사진 속의 물체를 최대 5번까지 추측해도 맞추지 못한 경우의 비율이다.

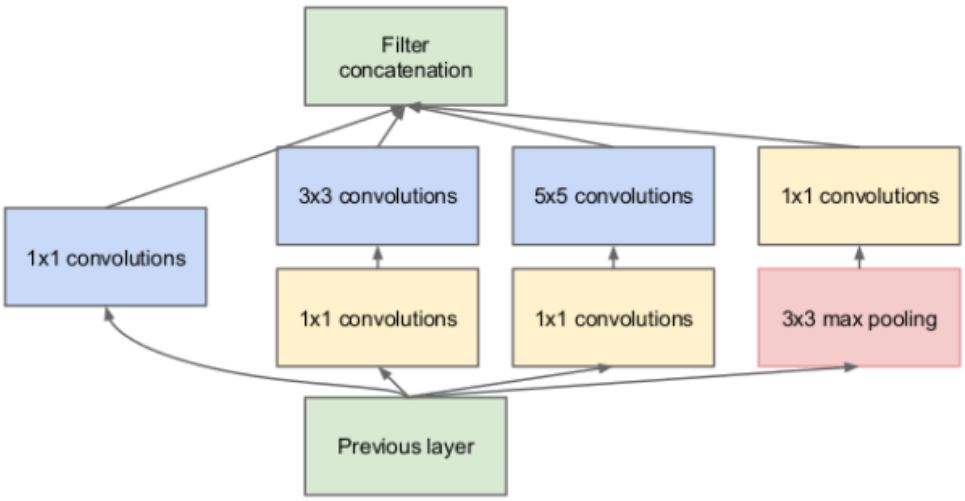
AlexNet은 2012년 ILSVRC에서 우승을 차지한 모형으로 15.4%를 틀려서 26.2%를 틀린 2등을 큰 차이로 따돌렸다. 이 사건을 계기로 인공신경망이 다시 한 번 큰 관심을 받게 된다. AlexNet에 사용된 ReLU와 Dropout 등은 현재도 널리 쓰인다.



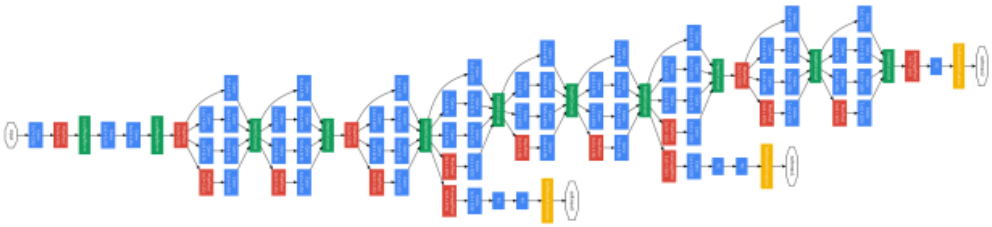
9.5.3. GoogLeNet

GoogLeNet은 ILSVRC 2014에서 1등을 한 모형이다. AlexNet은 8층으로 된 모형이지만 이 모형은 22층으로 된 훨씬 깊은 모형이다. Top-5 오류율도 6.7%까지 낮췄다.

GoogLeNet은 인셉션(inception)이라는 모듈을 반복해서 사용하는 구조로 되어있다. 인셉션 모듈은 여러 개의 합성곱 레이어와 풀링 레이어를 조합한 것이다. 특히 1x1 합성곱 레이어를 많이 사용하는데 이것은 feature map의 수를 줄여주는 역할을 한다.



이런 인셉션 모듈을 여러 번 반복해서 쌓은 것이 GoogLeNet이다.



노란색으로 된 부분은 이미지의 사물을 분류하는 소프트맥스 레이어이다. 신경망이 깊어지면서 학습이 잘 이뤄지지 않기 때문에 보조 분류기(auxilary classifier)를 중간 중간에 넣어서 학습을 돕는다. 보조 분류기는 학습을 시킬 때만 사용하고, 실제 예측에서는 맨 마지막에 있는 분류기만 사용한다.

또 후반부의 FC를 생략하여 파라미터의 수를 크게 줄였다.

9.5.4. VGG

VGG는 ILSVRC 2014에서 GoogLeNet에 밀려 2등을 했지만, 구조가 단순하여 더 많이 쓰이고 있다. VGG는 합성곱 레이어와 풀링 레이어를 번갈아가며 쌓는 전통적 구조를 하고 있지만 몇 가지 특이한 점이 있다.

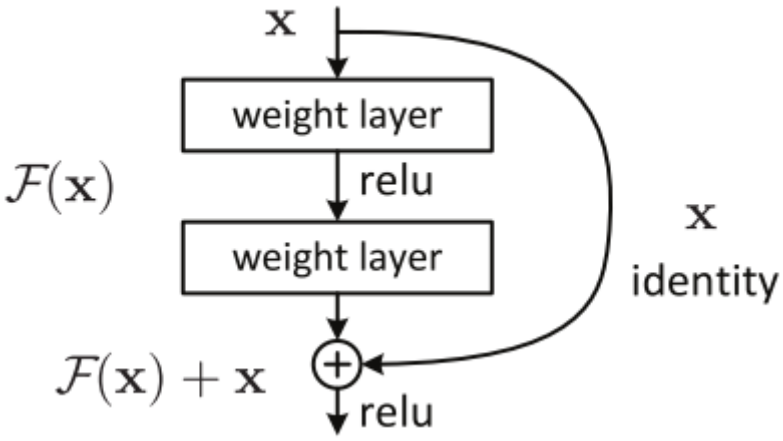
첫째 3x3 합성곱 레이어만 사용한다. 둘째, 합성곱(C) 레이어와 풀링(P) 레이어를 CPCP 식으로 쌓는 것이 아니라 CCP나 CCCP식으로 합성곱 레이어를 여러 번 겹쳐 쌓는다. 이렇게면 3x3 합성곱 레이어를 2번 겹쳐 쌓으면 5x5 합성곱 레이어와 비슷한 결과가 되지만, 파라미터의 수는 3x3 + 3x3 = 18로 5x5 = 25보다 적어지는 장점이 있다.

VGG는 구조는 단순하지만 전통적인 모형들과 마찬가지로 FC 레이어가 붙어있기 때문에 전체 파라미터의 수는 매우 많다.

VGG의 의의 중에 하나는 단순히 레이어를 많이 쌓는 것만으로 얼마나 성능이 좋아지는지 알아본 데 있다. 논문에 따르면 VGG의 경우 16층까지는 성능이 좋아졌지만 16층과 19층 사이에는 큰 차이가 없는 것으로 나타났다.

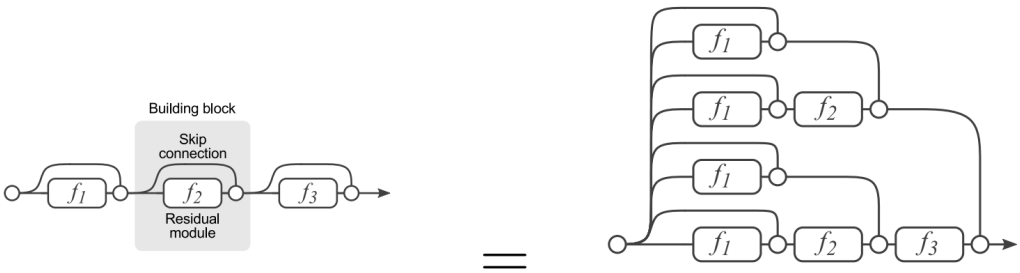
9.5.5. ResNet

ResNet은 ILSVRC 2015에서 Top-5 오류율 3.6%로 1등을 차지한 모형이다. ResNet의 깊이는 무려 152층에 달한다. ResNet의 기본 구조는 VGG와 비슷하지만, 입력값이 중간을 건너 뛰고 전달되는 지름길(shortcut)이 추가되었다.



신경망이 깊어지면 학습이 잘되지 않는 이유가 사라지는 경사 때문인데, ResNet은 신경망이 깊어지더라도 지름길을 따라 경사가 더 잘 전달될 수 있다.

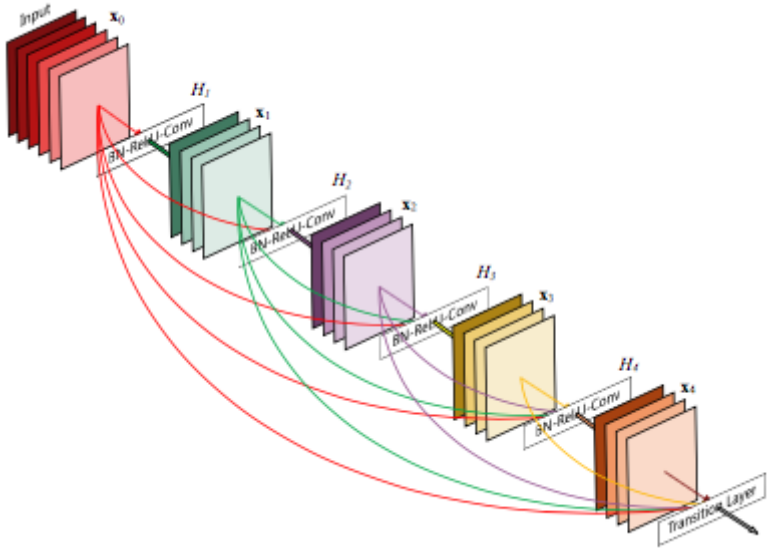
또한 f1, f2, f3 3개의 블록에 지름길을 덧붙여두면 실제적으로는 오른쪽 그림처럼 3개의 블록을 조합한 여러 개의 신경망의 앙상블을 만든 것과 비슷한 효과를 얻게 된다.



(a) Conventional 3-block residual network (b) Unraveled view of (a)

9.5.6. DenseNet

2017년 제안된 DenseNet은 ResNet의 연장선상에 있는 모형이다. 아래 그림에서 볼 수 있듯이 단지 하나의 블록을 건너뛰는 지름길만이 아니라 2개, 3개, 4개 등 여러 블록을 건너뛰는 지름길을 빽빽하게(dense) 넣은 것이 특징이다.



9.6. ResNet 실습

`keras` 에 미리 학습되어 있는 ResNet 모형을 사용할 것이다. 이 모형은 224x224 이미지에 학습되어 있으므로 사진을 불러올 때 해당 크기로 불러온다.

```
from keras.preprocessing.image import ImageDataGenerator

img_gen = ImageDataGenerator(
        rotation_range=40,      # 40도까지 회전
        width_shift_range=0.2,  # 20%까지 좌우 이동
        height_shift_range=0.2, # 20%까지 상하 이동
        shear_range=0.2,        # 20%까지 기울임
        zoom_range=0.2,         # 20%까지 확대
        horizontal_flip=True,   # 좌우 뒤집기
    )

train = img_gen.flow_from_directory(
    'dog-vs-cat/train',     # 이미지 디렉토리
    target_size=(224, 224), # 변환할 크기는 가로 224, 세로 224
    color_mode='rgb',       # 컬러는 rgb, 흑백은 grayscale. 생략하면 컬러로 처리한다
    class_mode='binary')    # 고양이 vs. 개로 binary 분류
```

```
valid = ImageDataGenerator().flow_from_directory(
    'dog-vs-cat/validation',
    target_size=(224, 224),
    class_mode='binary',
    shuffle=False)
```

9.6.1. ResNet50

미리 학습되어 있는 `ResNet` 모형을 불러온다.

```
from keras.applications.resnet50 import ResNet50, preprocess_input, decode_predictions
```

모형을 초기화하면 모형의 파라미터를 다운로드 받는다.

```
resnet = ResNet50()
```

요약을 확인해보면 모형의 내용을 볼 수 있다.

```
resnet.summary()
```

9.6.2. 분류

검증용 데이터셋에서 한 배치를 가져온다.

```
x, y = valid.next()
```

```
x.shape
```

```
(32, 224, 224, 3)
```

이미 학습된 모형이므로 학습시킬 필요 없이 바로 사용할 수 있다. ResNet을 학습시킬 때 사용했던 전처리를 그대로 사용한다.

```
inp = preprocess_input(x.copy())
pred = resnet.predict(inp)
```

```
pred.shape
```

```
(32, 1000)
```

배치의 첫번째 사진을 출력한다.

```
from matplotlib import pyplot as plt
```

```
plt.imshow((x[0] / 256))
plt.axis('off')
```

```
(-0.5, 223.5, 223.5, -0.5)
```

```
<matplotlib.figure.Figure at 0x7f5c62442470>
```

첫번째 사진의 예측 결과를 해석해서 본다.

```
decode_predictions(pred)[0]
```

```
[('n02328150', 'Angora', 0.19449782),
 ('n02124075', 'Egyptian_cat', 0.18835779),
 ('n02123045', 'tabby', 0.08570625),
 ('n04005630', 'prison', 0.0750663),
 ('n04589890', 'window_screen', 0.039134055)]
```

9.6.3. 전이 학습

다른 데이터에 학습된 모형을 새로운 데이터에 학습시키는 것을 전이 학습(transfer learning)이라 한다. ImageNet에 학습된 ResNet50을 우리의 개-고양이 데이터에 전이학습시켜보자.

다시 `ResNet50` 모형을 초기화한다. `include_top=False` 로 최상단의 `softmax` 레이어를 제거한다.

```
resnet = ResNet50(include_top=False)
```

resnet이 더이상 학습되지 않도록 모든 레이어의 파라미터를 고정한다.

```
for layer in resnet.layers:
    layer.trainable = False
```

`Dense` 레이어를 만들어서 `resnet` 의 출력을 입력으로 받게 만든다.

```
from keras.layers import Dense, GlobalAveragePooling2D
```

```
flat = GlobalAveragePooling2D()(resnet.output)
```

```
fc = Dense(1, activation='sigmoid')(flat)
```

이제 ResNet의 입력에서 `fc` 의 출력으로 이어지는 모형을 새로 만든다.

```
from keras.models import Model

model = Model(inputs=resnet.input, outputs=fc)
```

모형 요약을 해보면 ResNet 뒤에 새로운 FC가 붙었다는 것을 알 수 있다.

```
model.summary()
```

모형을 컴파일 한다.

```
from keras.optimizers import Adam

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

모형을 학습시킨다.

```
from keras.callbacks import ModelCheckpoint, TensorBoard
```

```
def preproc_generator(gen):
    for batch_x, batch_y in gen:
        yield preprocess_input(batch_x), batch_y
```

```
model.fit_generator(
    preproc_generator(train),
    steps_per_epoch=train.n // 32,
    validation_data=preproc_generator(valid),
    validation_steps=3,
    epochs=3,
    callbacks=[
        ModelCheckpoint('resnet-{epoch:02d}.hdf5', save_best_only=True),
        TensorBoard(log_dir='log_resnet')
    ]
)
```

```
<keras.callbacks.History at 0x7f5c0bf73b00>
```