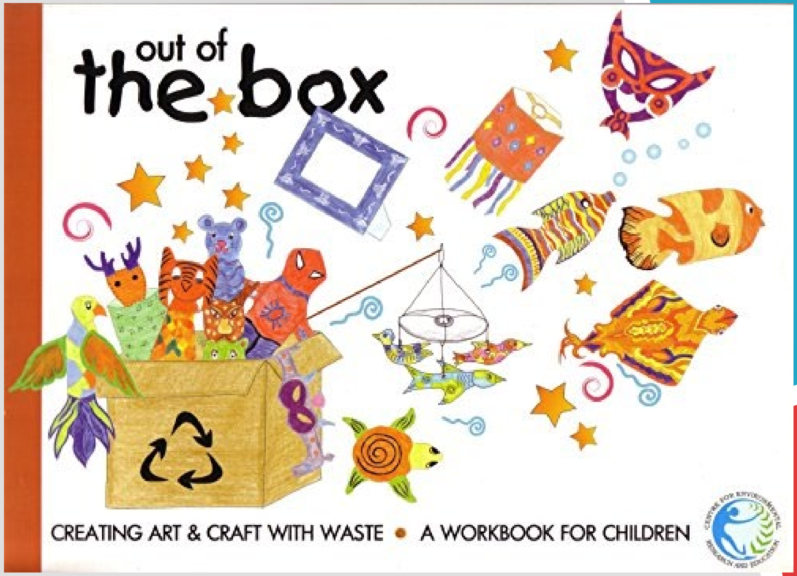
5장 *Deep learning for computer vision*

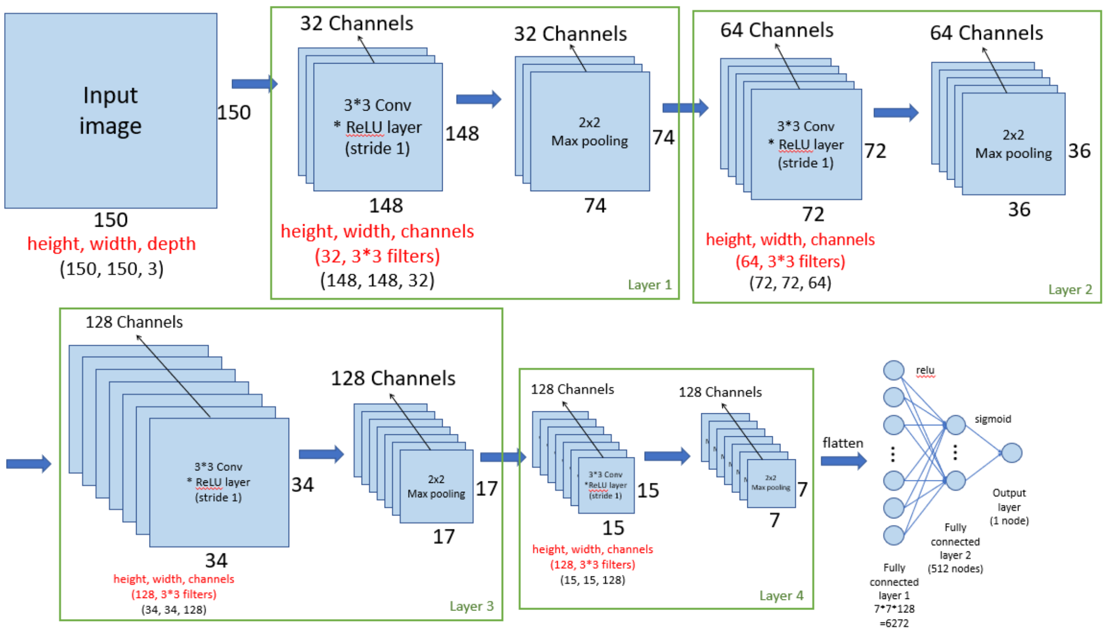
“Out of the Box”



5.2 Training a convnet from scratch on a small dataset

실습

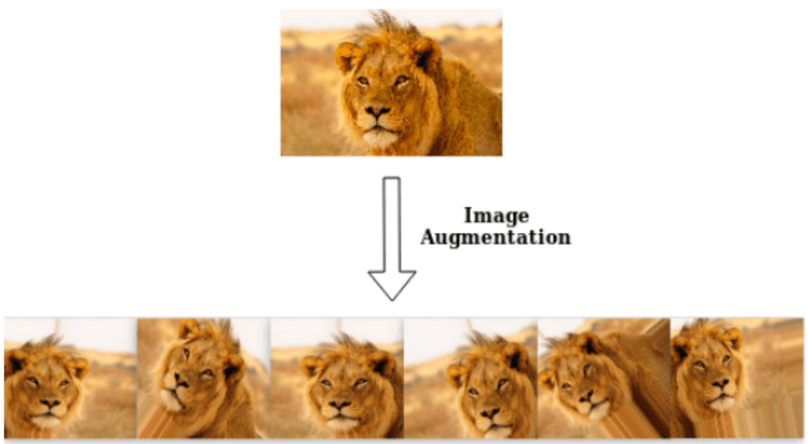
- p.182-p.192의 Dogs vs. Cats 프로그램을 실행하고 다음 그림을 사용하여 설명하시오.
- layer 수를 변경하고 매개변수를 조정하여 얻은 결과를 비교하여 설명하시오.

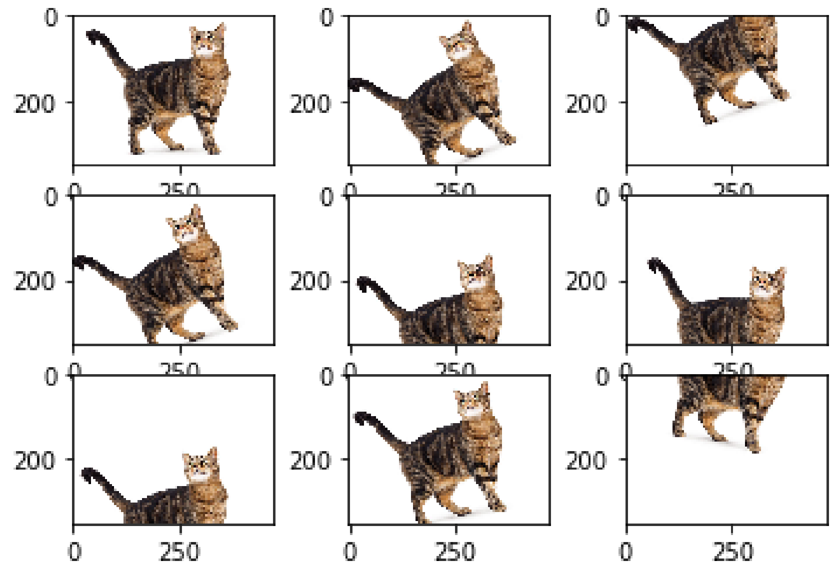


5.2 Training a convnet from scratch on a small dataset

5.2.5 Using data augmentation

- Data augmentation - generating more training data via a number of random transformations
- expose the model to more aspects of the data and generalize better
- `ImageDataGenerator` instance - number of random transformations

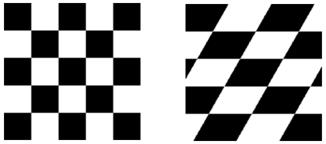




5.2 Training a convnet from scratch on a small dataset

5.2.5 Using data augmentation

Listing 5.11 Setting up a data augmentation configuration via ImageDataGenerator



shear

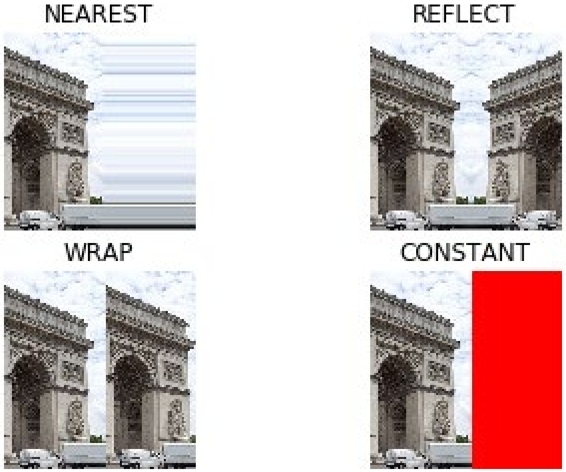
```
datagen=ImageDataGenerator(rotation_range=40 #degrees
        width_shift_range=0.2,height_shift_range=0.2,
        shear_range=0.2, zoom_range=0.2,
        horizontal_flip=True, fill_mode='nearest')
```

- These are just a few of the options available (for more, see the Keras documentation):
  - **rotation_range** - degrees (0–180), a range within which to randomly rotate pictures.
  - **width_shift** and **height_shift** - ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
  - **shear_range** - randomly applying shearing transformations.
  - **zoom_range** - randomly zooming inside pictures.
  - **horizontal_flip** - randomly flipping half the images horizontally—relevant when there are no assumptions of horizontal asymmetry (for example, real-world pictures).

5.2 Training a convnet from scratch on a small dataset

5.2.5 Using data augmentation

- `fill_mode` is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift. {"constant", "nearest", "reflect"}.



5.2 Training a convnet from scratch on a small dataset

5.2.5 Using data augmentation

Listing 5.12 Displaying some randomly augmented training images

cat.100.jpg

```
from keras.preprocessing import image # 이미지 전처리 유틸리티 모듈
fnames = sorted([os.path.join(train_cats_dir, fname) for
         fname in os.listdir(train_cats_dir)])
img_path = fnames[3] # 증식할 이미지 선택
# 이미지를 읽고 크기 변경
img = image.load_img(img_path, target_size=(150, 150))
# (150, 150, 3) 크기의 넘파이 배열로 변환, [:,:,0:3] 반환
x = image.img_to_array(img)
x = x.reshape((1,)+x.shape) # (1,150,150,3) 크기로 변환
# flow() 메서드는 랜덤하게 변환된 이미지의 배치를 생성
# 무한 반복되기 때문에 어느 지점에서 중지해야 합니다!
i = 0
# flow-이미지를 배치 단위로 가져옴
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()
```

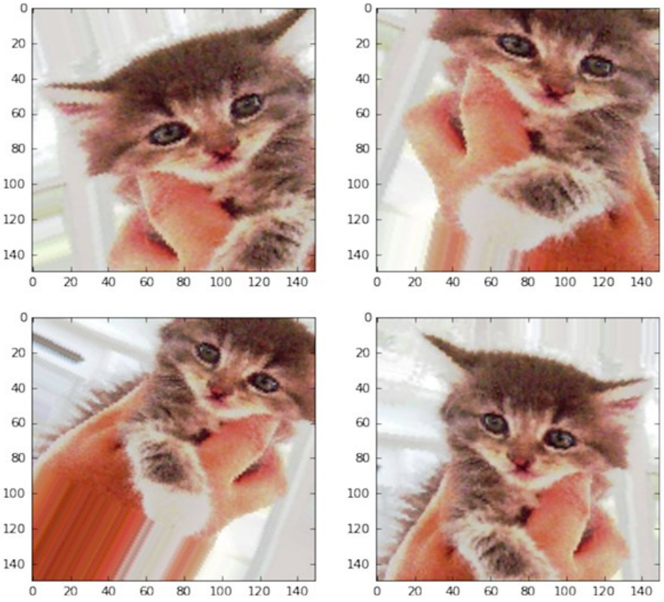


Figure 5.11 Generation of cat pictures via random data augmentation

5.2 Training a convnet from scratch on a small dataset

5.2.5 Using data augmentation

- data-augmentation - never produce the same input twice.
- overfitting - remix existing inputs are still heavily intercorrelated
- add a `Dropout` layer to your model, right before the densely connected classifier

Listing 5.13 Defining a new convnet that includes dropout

```
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)) # Flatten 다음, FCN 전
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'])
```

5.2 Training a convnet from scratch on a small dataset

Listing 5.14 Training the convnet using data-augmentation

```
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) # 검증 데이터는 증식하지 않음
train_generator = train_datagen.flow_from_directory(
        train_dir, # 타깃 디렉터리
        target_size=(150, 150), # 150 × 150 크기로 바꿉니다
        batch_size=32, # data augmented
        class_mode='binary') # 이진 레이블
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=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

5.2 Training a convnet from scratch on a small dataset

Listing 5.15 Saving the model generators

```
model.save('cats_and_dogs_small_2.h5')
```

- data augmentation and dropout - no longer overfitting: the training curves are closely tracking the validation curves.
- accuracy of 82%, a 15% **relative** improvement over the non-regularized model.
- use a pretrained model

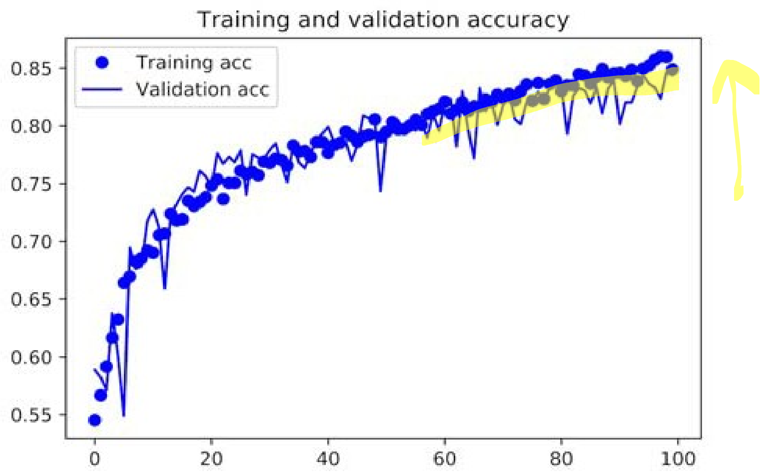


Figure 5.12 Training and validation accuracy with data augmentation

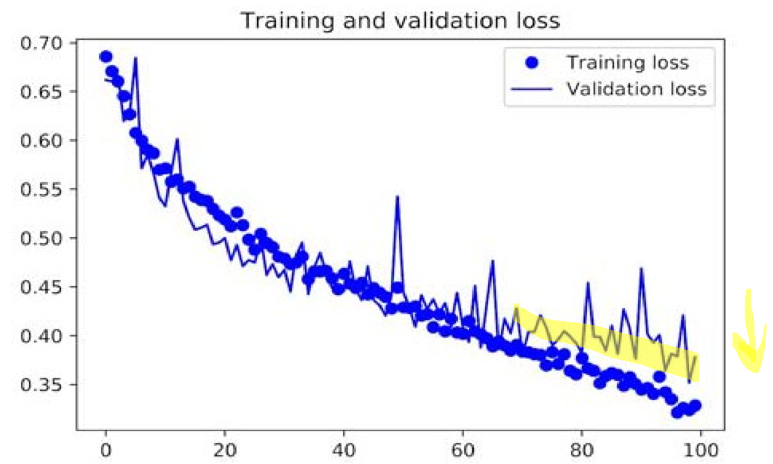


Figure 5.13 Training and validation loss with data augmentation

5.3 Using a pretrained convnet

- pretrained network - previously trained on a large dataset, typically on a large-scale image-classification task
- large and general enough dataset - generic model, useful for many different computer-vision problems
- train a network on ImageNet (1000 of classes, 1.4 million of images)
- a key advantage of deep learning - portability of learned features across different problems, very effective for small-data problems.

5.3 Using a pretrained convnet

- ImageNet contains many animal classes, including different species of cats and dogs – perform well on the dogs-versus-cats classification problem.
- VGG16 convnet architecture for ImageNet - by Karen Simonyan and Andrew Zisserman in 2014, easy to understand without introducing any new concepts
- Previous works - VGG, ResNet, Inception, Inception-ResNet, Xception
- There are two ways to use a pretrained network: feature extraction and fine-tuning.

5.3 Using a pretrained convnet

5.3.1 Feature extraction

- Feature extraction - learned by a previous network to extract interesting features from new samples. These features are then run through a new classifier.

- convnets used for image classification comprise two parts:

  - *convolutional base* - series of pooling and convolution layers

  - *densely connected classifier*

5.3 Using a pretrained convnet

5.3.1 Feature extraction

- Running the new data through it, and training a new classifier on top of the output (figure 5.14).

densely connected classifier

convolutional base

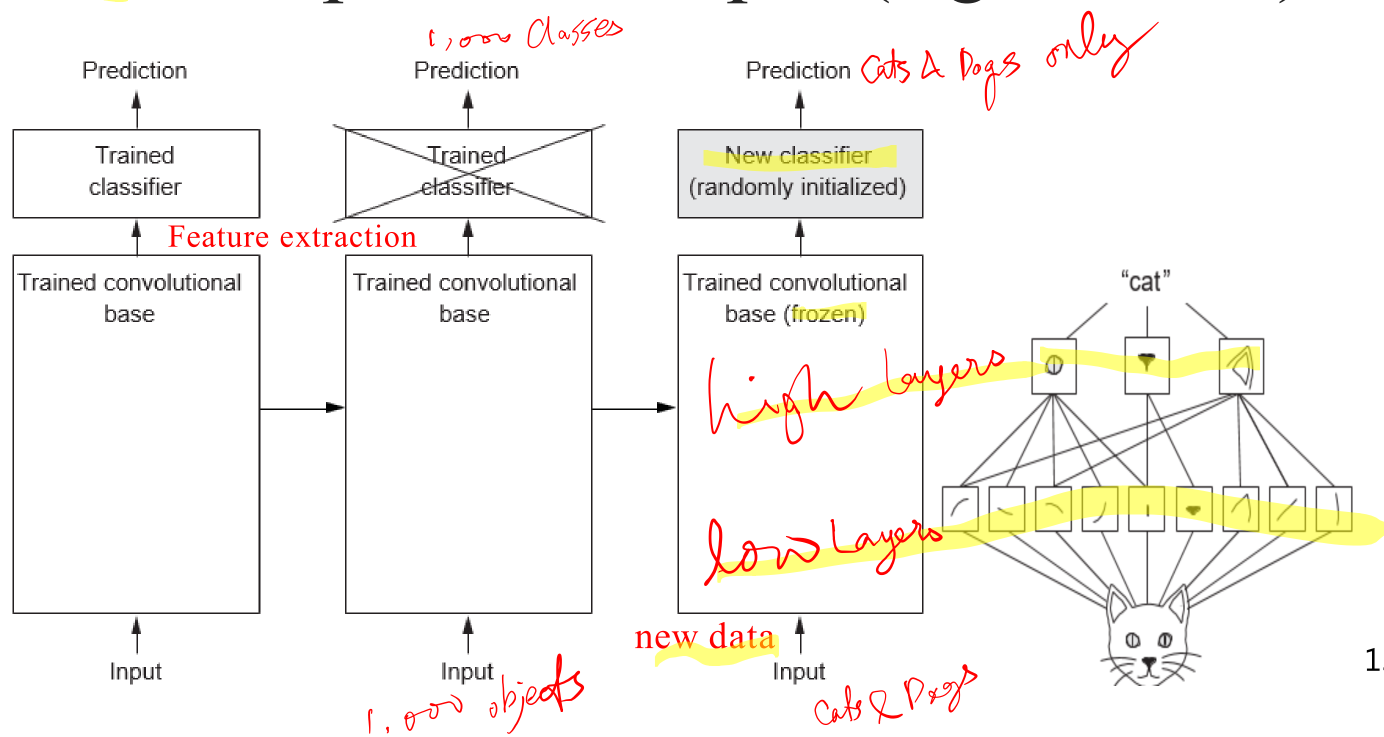


Figure 5.14 Swapping classifiers while keeping the same convolutional base

5.3 Using a pretrained convnet

5.3.1 Feature extraction

- Reuse the densely connected classifier? should be avoided.
- The feature maps of a convnet are presence maps of generic concepts over a picture.
- level of generality - depth of the layer in the model.
  - low layers - highly generic feature maps (such as visual edges, colors, and textures)
  - high layers - more-abstract concepts (such as “cat ear” or “dog eye”)

5.3 Using a pretrained convnet

5.3.1 Feature extraction

- VGG16 network trained on ImageNet - train a dogs-versus-cats
- Import it from the `keras.applications` module.
- Here's the list of image-classification models (all pretrained on the ImageNet dataset) that are available as part of `keras.applications`:
  - Xception
  - Inception V3
  - ResNet50
  - VGG16
  - VGG19
  - MobileNet

5.3 Using a pretrained convnet

5.3.1 Feature extraction

Listing 5.16 Instantiating the VGG16 convolutional base

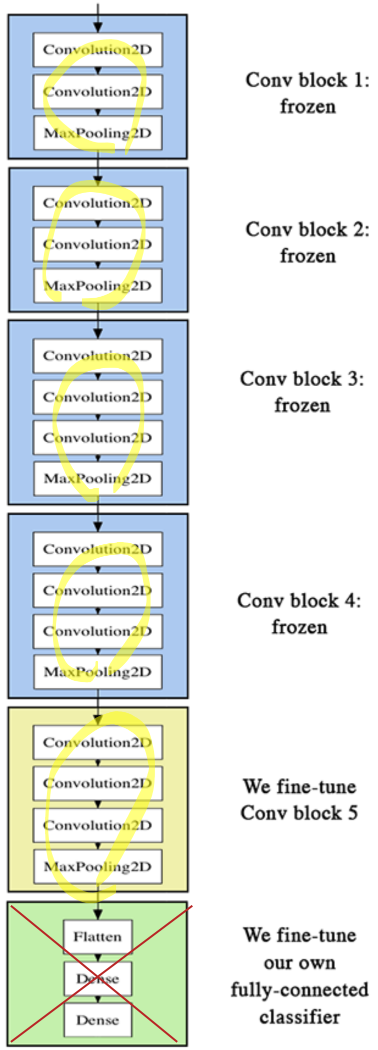
```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
      include_top=False,  # densely connected classifier
      input_shape=(150,150,3))  # optional
```

- Three arguments to the constructor:
  - `weights` - weights initialization
  - `include_top` – densely connected classifier on top of the network.
  - By default, 1,000 classes from ImageNet. For the two classes of `cat` and `dog`, don't include it.
  - `input_shape` – shape of the image tensors

5.3 Using a pretrained convnet

>>> conv_base.summary()

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Convolution2D)	(None, 150, 150, 64)	1792
block1_conv2 (Convolution2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Convolution2D)	(None, 75, 75, 128)	73856
block2_conv2 (Convolution2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Convolution2D)	(None, 37, 37, 256)	295168
block3_conv2 (Convolution2D)	(None, 37, 37, 256)	590080
block3_conv3 (Convolution2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Convolution2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Convolution2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Convolution2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0



5.3 Using a pretrained convnet

5.3.1 Feature extraction

- The final feature map has shape `(4, 4, 512)`. That's the feature on top of densely connected classifier.
- two ways to proceed:
  - Running the convolutional base over your dataset → recording its output to a Numpy array (features) → input to densely connected classifier - running the convolutional base once for every input image without data augmentation.
  - Running the whole thing end to end on the input data with data augmentation