5.3 Using a pretrained convnet

- fine-tune - the layers block5_conv1, block5_conv2, and block5_conv3 should be trainable.
- Why not fine-tune more layers? Why not fine-tune the entire convolutional base? consider the following:
  - Earlier layers - more-generic, reusable features
  - Higher layers - more-specialized features.
  - The more parameters you're training, the more you're at risk of overfitting.
  - The convolutional base has 15 million parameters, so it would be risky to attempt to train it on your small dataset.

Listing 5.22 Freezing all layers up to a specific one

```
conv_base.trainable = True # T,T,…,T
set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable: # F,F,…,F,T,T,T
        layer.trainable = True
    else:  # freeze before block5_conv1
        layer.trainable = False
```

```
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
_________________________________________________________________
```

freeze

unfreeze

5.3 Using a pretrained convnet

Listing 5.23 Fine-tuning the model

```
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)
```

5.3 Using a pretrained convnet

Listing 5.24 Smoothing the plots

```
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()
```

- The validation accuracy curve from about 96% to above 97%
- Note that the loss curve doesn't show any real improvement (in fact, it's deteriorating)

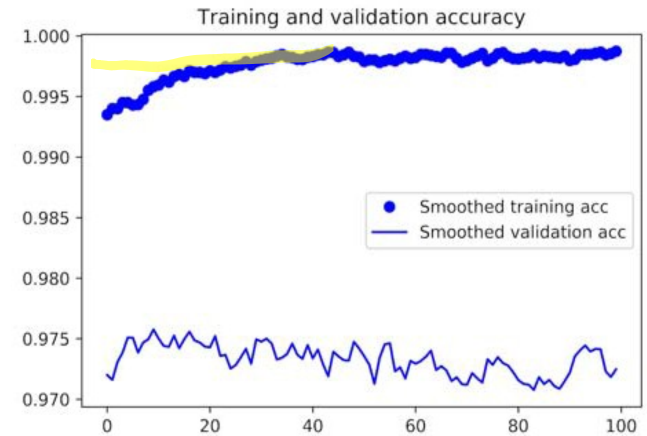


Figure 5.22 Smoothed curves for training and validation accuracy for fine-tuning

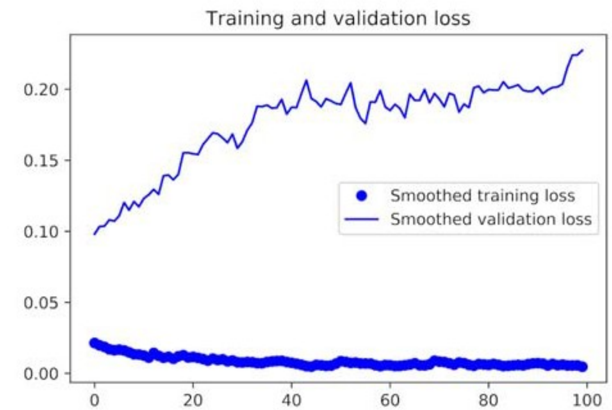


Figure 5.23 Smoothed curves for training and validation loss for fine-tuning

5.3 Using a pretrained convnet

- You can now finally evaluate this model on the test data:

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

- test accuracy of 92% - In the original Kaggle competition around this dataset, this would have been one of the top results using only a small fraction of the training data available (about 10%). There is a huge difference between being able to train on 20,000 samples compared to 2,000 samples!

5.3 Using a pretrained convnet

- 실습

다음 convolution block들을 unfreeze하고 fine tuning 하는 프로그램과 그 결과를 ppt 58 page와 비교하시오.

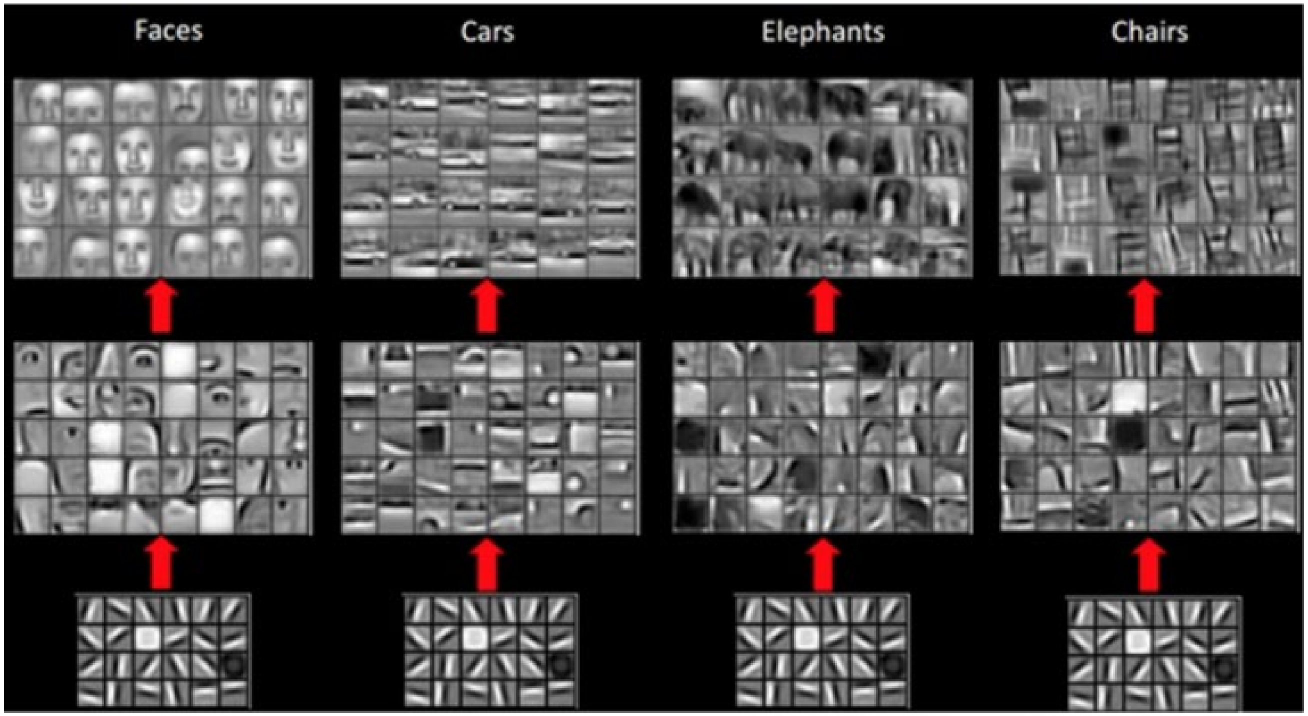
1) block5_conv3

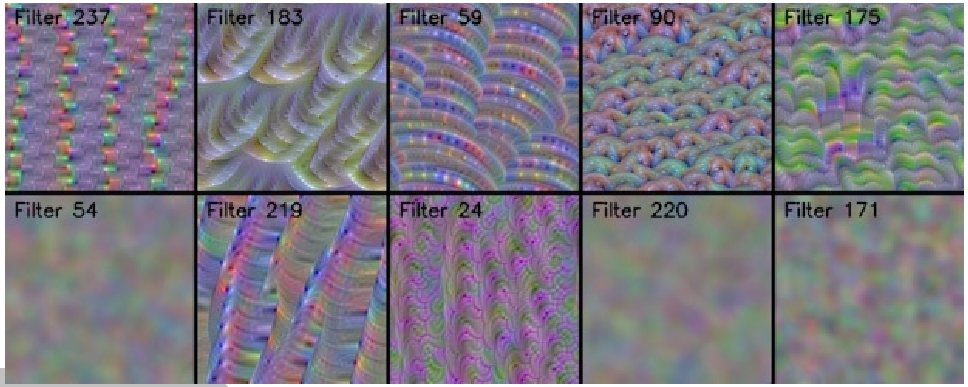
2) block5_conv3, block5_conv2

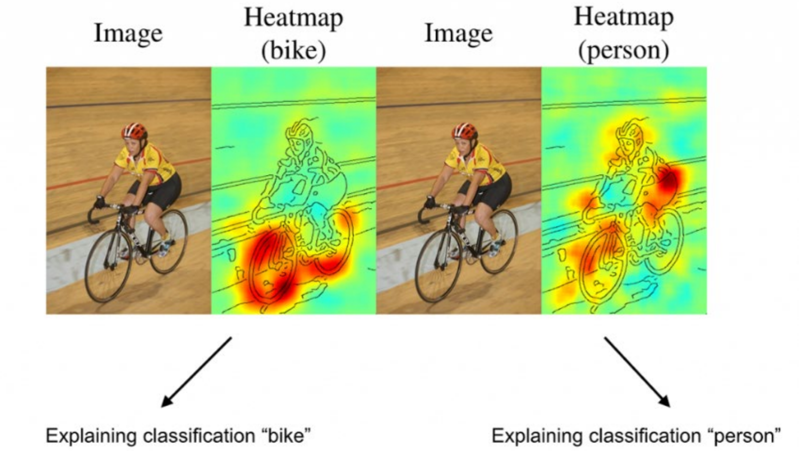
5.4 Visualizing what convnets learn

- "black boxes": learning representations that are difficult to extract and present in a human-readable form
- convnets are highly amenable to visualization - *representations of visual concepts*
- Since 2013, a wide array of techniques have been developed for visualizing and interpreting these representations:
  - Visualizing intermediate convnet outputs (intermediate activations) — Understanding convnet layers transform their input, and individual convnet filters
  - Visualizing convnets filters — understanding precisely what visual pattern or concept each filter in a convnet is receptive to.
  - Visualizing heatmaps of class activation in an image—identified as belonging to a given class, localize objects in images.

5.4 Visualizing what convnets learn







5.4 Visualizing what convnets learn

5.4.1 Visualizing intermediate activations

- Visualizing intermediate activations -**convolution** and **pooling** layers
- How an input is decomposed into the different filters learned by the network.
- visualize feature maps - plotting the contents of every channel as a 2D image
- Let's start by loading the model that you saved in section 5.2:

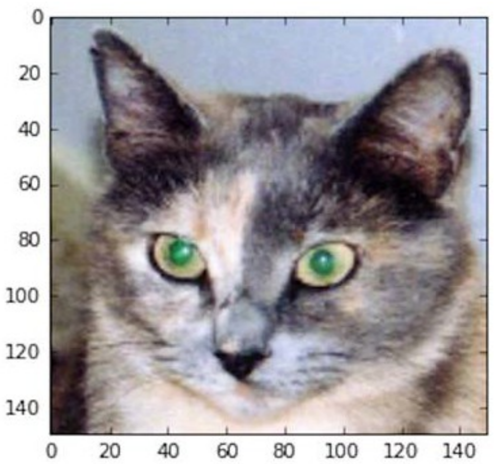
```
>>> from keras.models import load_model
>>> model= load_model('cats_and_dogs_small_2.h5')
>>> model.summary() # As a reminder

Layer (type)                     Output Shape              Param #
================================================================
conv2d_5 (Conv2D)                (None, 148, 148, 32)      896
________________________________________________________________
max_pooling2d_5 (MaxPooling2     (None, 74, 74, 32)        0
________________________________________________________________
conv2d_6 (Conv2D)                (None, 72, 72, 64)        18496
________________________________________________________________
max_pooling2d_6 (MaxPooling2     (None, 36, 36, 64)        0
________________________________________________________________
conv2d_7 (Conv2D)                (None, 34, 34, 128)       73856
________________________________________________________________
max_pooling2d_7 (MaxPooling2     (None, 17, 17, 128)       0
________________________________________________________________
conv2d_8 (Conv2D)                (None, 15, 15, 128)       147584
________________________________________________________________
max_pooling2d_8 (MaxPooling2     (None, 7, 7, 128)         0
________________________________________________________________
flatten_2 (Flatten)              (None, 6272)              0
________________________________________________________________
dropout_1 (Dropout)              (None, 6272)              0
________________________________________________________________
dense_3 (Dense)                  (None, 512)               3211776
________________________________________________________________
dense_4 (Dense)                  (None, 1)                 513
================================================================
```

Listing 5.25 Preprocessing a single image

```
img_path =
 './datasets/cats_and_dogs_small/test/cats/cat.1700.jpg'

# 이미지를 4D 텐서로 변경합니다
from keras.preprocessing import image
import numpy as np
img = image.load_img(img_path, target_size=(150, 150))
img_tensor = image.img_to_array(img) # (150, 150, 3) # 컬러 이미지
img_tensor = np.expand_dims(img_tensor, axis=0) # 맨 앞차원에 추가
# img_tensor = img_tensor.reshape((1,) + img_tensor.shape)
# 모델이 훈련될 때 입력에 적용한 전처리 방식
img_tensor /= 255.

print(img_tensor.shape) # (1, 150, 150, 3)
```

Listing 5.25 Preprocessing a single image

```
import matplotlib.pyplot as plt

plt.imshow(img_tensor[0])
plt.show()
```



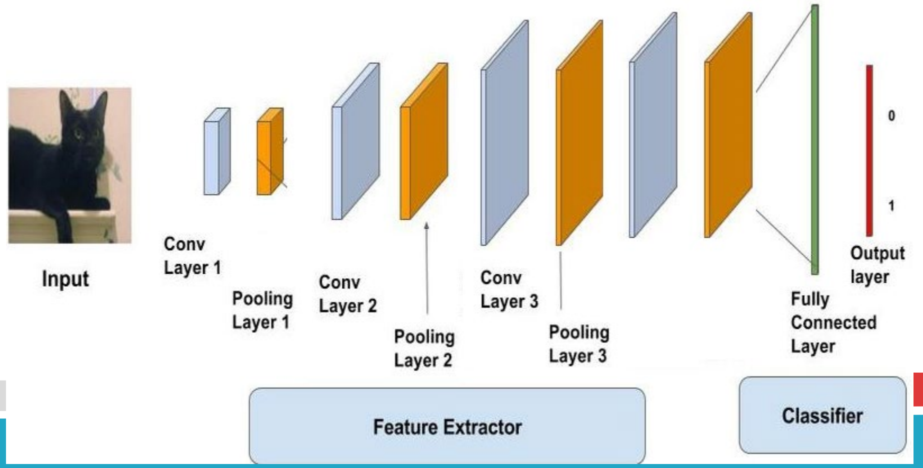
5.4 Visualizing what convnets learn

- outputs the activations of all convolution and pooling layers.
- Keras class Model – an **input** tensor (or list of input tensors) and an **output** tensor (or list of output tensors)

Listing 5.27 Instantiating a model from an input tensor and a list of output tensors

```
from keras import models

# Extracts the outputs of the top eight layers
layer_outputs = [layer.output for layer in
                          model.layers[:8]]
                        # model= load_model('cats_and_dogs_small_2.h5')
# Creates a model that will return these outputs, given the model input
activation_model = models.Model(inputs=model.input, #cat image
              outputs=layer_outputs) # 8 output layers
```

- multi-output model - one input and eight outputs, one output per layer activation



5.4 Visualizing what convnets learn

Listing 5.28 Running the model in predict mode

```
# Returns a list of 8 Numpy arrays: one array per layer activation
activations = activation_model.predict(img_tensor) #cat image
```

- activation of the first convolution layer for the cat image input:

```
>>> first_layer_activation = activations[0]
>>> print(first_layer_activation.shape)
(1, 148, 148, 32)
```

- It's a 148 × 148 feature map with 32 channels. Let's try plotting the **fourth channel** of the activation of the **first layer** of the original model (see figure 5.25).

Listing 5.29 Visualizing the fourth channel

/32

```
import matplotlib.pyplot as plt
plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')
```

- This channel appears to encode a **diagonal edge** detector.

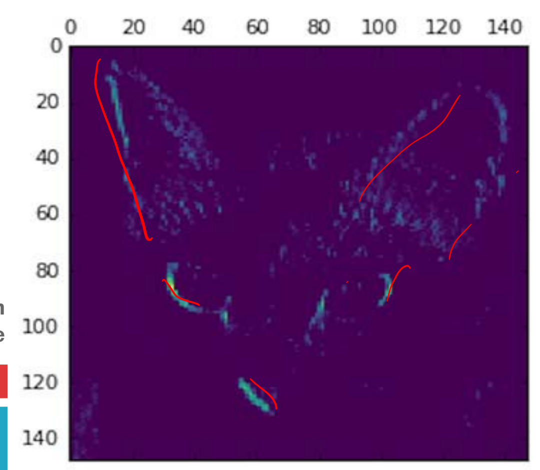


Figure 5.25 Fourth channel of the activation of the first layer on the test cat picture

5.4 Visualizing what convnets learn

- Let's try **the seventh** channel — the specific filters learned by convolution layers aren't deterministic.

Listing 5.30 Visualizing the seventh channel

```
plt.matshow(first_layer_activation[0, :, :, 7],
                                  cmap='viridis')
```

- This one looks like a "bright green dot" detector, useful to encode cat eyes.
- extract and plot every channel in each of the eight activation maps

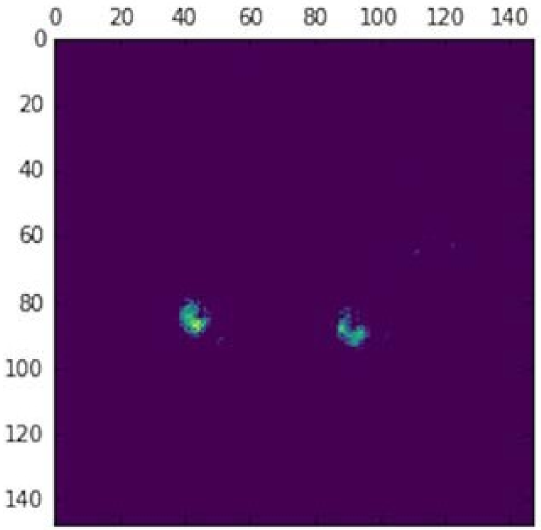


Figure 5.26 Seventh channel of the activation of the first layer on the test cat picture