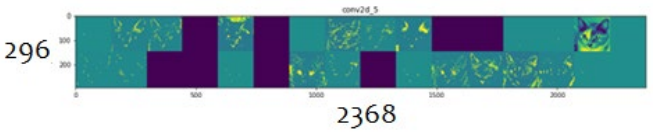
5.4 Visualizing what convnets learn

Listing 5.31 Visualizing every channel in every intermediate activation

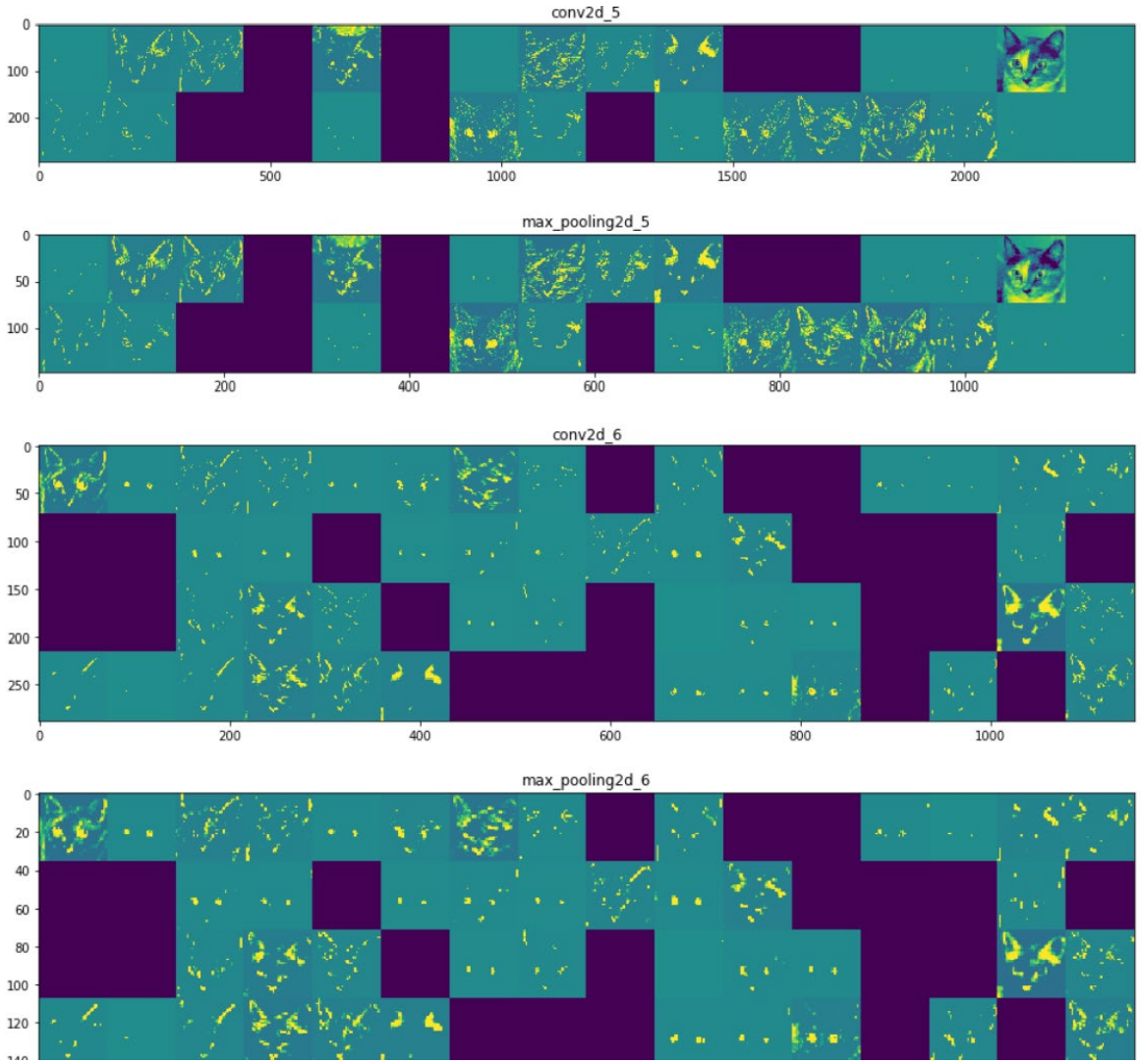
```
layer_names = [] # Names of the layers
for layer in model.layers[:8]:
    layer_names.append(layer.name)
images_per_row = 16
# Displays the feature maps
for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1] # 맨뒤-32,32,64,64,128,128,128,128
    size = layer_activation.shape[1] # (1, size, size, n_features) # 148,74,72,…
    n_cols = n_features // images_per_row  # 32 // 16 몫 2
    display_grid = np.zeros((size * n_cols, images_per_row * size)) #(296,2368)
    for col in range(n_cols): # col : 0~1
        for row in range(images_per_row): # 16
            channel_image = layer_activation[0, :, :,
                                             col * images_per_row + row] # row : 0~15, 16~31
            # Post-processes the feature to make it visually palatable
            channel_image -= channel_image.mean()
            channel_image /= channel_image.std() # z-dist
            channel_image *= 64 # -64 ~ 64 정도의 값
            channel_image += 128 # 양수로 만듬
            channel_image = np.clip(channel_image, 0, 255).astype('uint8') # 0~255, ex:-3→0, 283→256
                          #np.clip(a, a_min, a_max, out=None)
            display_grid[col * size : (col + 1) * size,
                         row * size : (row + 1) * size] = channel_image
    scale = 1. / size # Displays the grid
    plt.figure(figsize=(scale * display_grid.shape[1], # 1./148 * 2368 = 16, x축
                        scale * display_grid.shape[0])) # 1./148 * 296 = 2, y축
    plt.title(layer_name)
    plt.grid(False)  # 격자선 제거
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
plt.show()
```

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

```
>>> numbers = [1, 2, 3]
>>> letters = ["A", "B", "C"]
>>> for pair in zip(numbers, letters):
        print(pair)

(1, 'A') (2, 'B') (3, 'C')
```



5.4 Visualizing what convnets learn



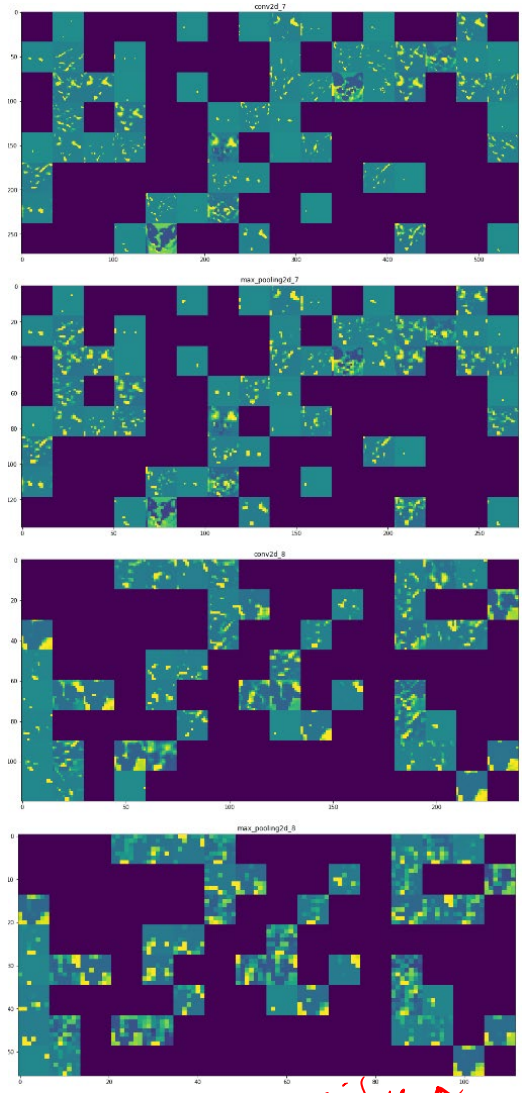


Figure 5.27 Every channel of every layer activation on the test cat picture

Sparsity ↑

5.4 Visualizing what convnets learn

- There are a few things to note here:
  - The first layer acts as a collection of various edge detectors in the initial picture.
  - As you go higher, the activations become increasingly abstract and less visually interpretable. – the more information about the target "cat ear" and "cat eye."
  - The sparsity of the activations increases with the depth of the layer: no pattern encoded by the filter in the input image.
- A deep neural network effectively acts as an *information distillation pipeline*
- A human can remember which abstract objects were present in it (bicycle, tree) but can't remember the specific appearance of these objects. (see, for example, figure 5.28).
- Your brain has learned to completely abstract its visual input—to transform it into high-level visual concepts while filtering out irrelevant visual

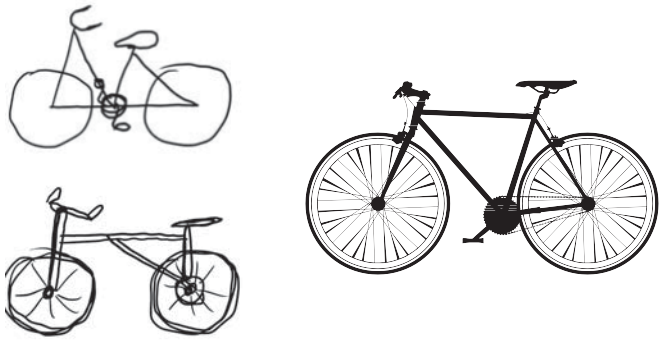
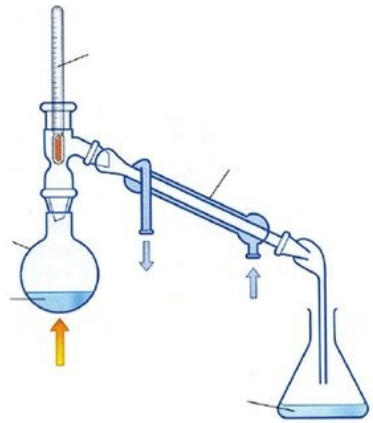


Figure 5.28 Left: attempts to draw a bicycle from memory. Right: what a schematic bicycle should look like.



5.4 Visualizing what convnets learn

5.4.2 Visualizing convnet filters

- Display the visual pattern of each filter – *gradient ascent in input space* : applying *gradient descent* to the value of the input image of a convnet so as to *maximize* the response of a specific filter, starting from a blank input image. The resulting input image will be one that the chosen filter is maximally responsive to.
- The process is simple: build a loss function that maximizes the value of a given filter in a given convolution layer, and then use stochastic gradient descent to adjust the values of the input image so as to maximize this activation value. For instance, here's a loss for the activation of filter 0 in the layer `block3_conv1` of the VGG16 network, pretrained on ImageNet.

Listing 5.32 Defining the loss tensor for filter visualization

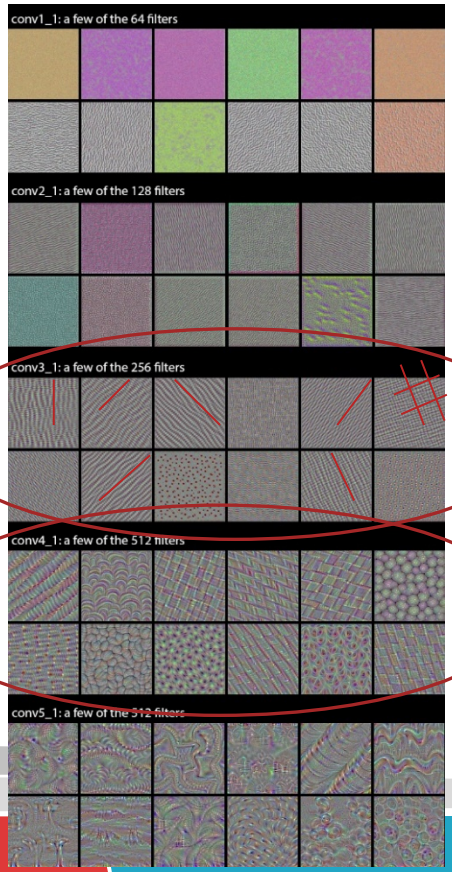
```
from keras.applications import VGG16
from keras import backend as K

model = VGG16(weights='imagenet',include_top=False)

layer_name = 'block3_conv1'
filter_index = 0

layer_output = model.get_layer(layer_name).output # (?,37,37,256)
loss = K.mean(layer_output[:, :, :, filter_index]) #  filter 0
```

5.4 Visualizing what convnets learn

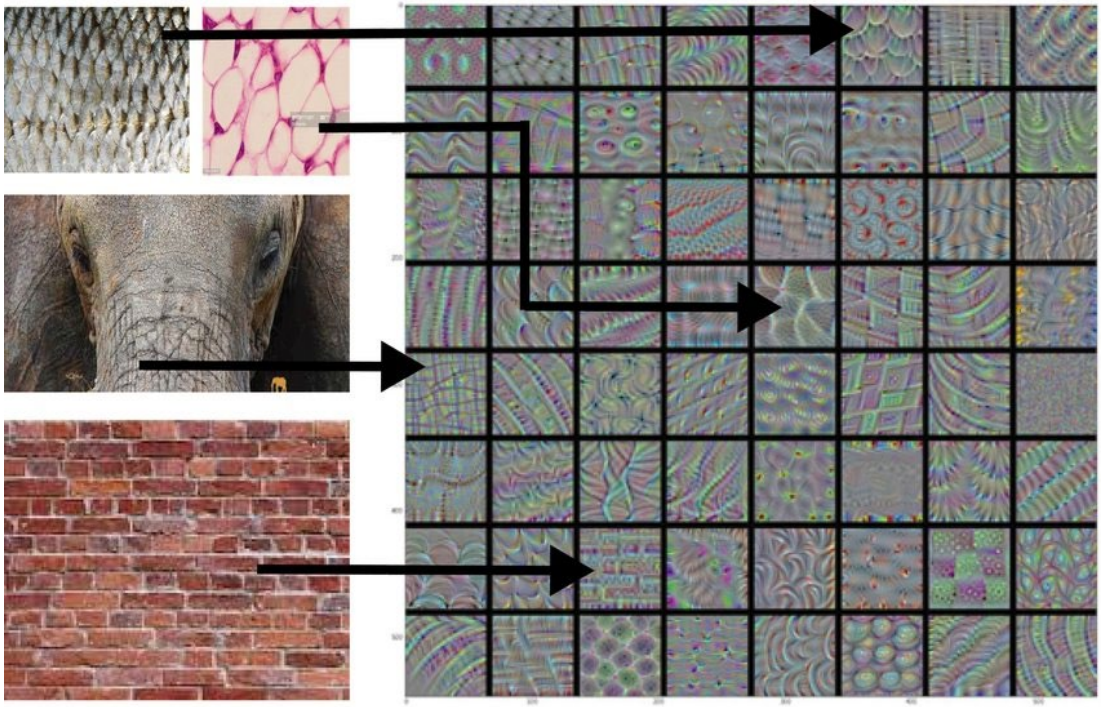
- gradient of this loss with respect to the model's input – **gradients** function packaged with the **backend** module of Keras.

Listing 5.33 Obtaining the gradient of the loss with regard to the input

```
grads = K.gradients(loss, model.input)[0] # filter 0, (?, ?, ?, 3)
# The call to gradients returns a list of tensors (of size 1 in this case).
# compute the gradient of the input picture wrt this loss
```



64 filters of the VGG16 Block4 conv1 layer



5.4 Visualizing what convnets learn

- Normalize the gradient tensor by dividing it by its L2 norm (the square root of the average of the square of the values in the tensor).
- This ensures that the magnitude of the updates done to the input image is always within the same range.

Listing 5.34 Gradient-normalization trick

```
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
# + 1e-5: avoid accidentally dividing by 0
```

- loss tensor and the gradient tensor computation for given an input image - define a Keras backend function: **iterate**, a function that takes a Numpy tensor (as a list of tensors of size 1) and returns a list of two Numpy tensors: the loss value and the gradient value.

Listing 5.35 Fetching Numpy output values given Numpy input values

```
iterate = K.function([model.input], [loss, grads])
import numpy as np
loss_value, grads_value = iterate([np.zeros((1, 150, 150, 3))])
```

5.4 Visualizing what convnets learn

- At this point, you can define a Python loop to do stochastic gradient descent.

Listing 5.36 Loss maximization via stochastic gradient descent

```
input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128. # [0.0, 1.0)*20+128
  # Starts from a gray image with some noise
step = 1. # Magnitude of each gradient update
for i in range(40):   # Runs gradient ascent for 40 steps
    # Computes the loss value and gradient value
    loss_value, grads_value = iterate([input_img_data])
    # Adjusts the input image in the direction that maximizes the loss
    input_img_data += grads_value * step
```

- The resulting image tensor is a floating-point tensor of shape `(1, 150, 150, 3)` → values of integers within **[0, 255]**.

Listing 5.37 Utility function to convert a tensor into a valid image

```
def deprocess_image(x):  # p.75
    # Normalizes the tensor: centers on 0, ensures that std is 0.1
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1
    x += 0.5
    x = np.clip(x, 0, 1) # Clips to [0, 1]
    # Converts to an RGB array
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

5.4 Visualizing what convnets learn

- Let's put them together into a Python function that takes as input a layer name and a filter index, and returns a valid image tensor representing the pattern that maximizes the activation of the specified filter.

Listing 5.38 Function to generate filter visualizations

```
def generate_pattern(layer_name, filter_index, size=150): # block3_conv1, filter 0
    # Builds a loss function that maximizes the activation of the nth filter of the layer
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])

    # Computes the gradient of the input picture with regard to this loss
    grads = K.gradients(loss, model.input)[0]

    # Normalization trick: normalizes the gradient
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5) # L2

    # Returns the loss and grads given the input picture
    iterate = K.function([model.input], [loss, grads])

    # Starts from a gray image with some noise
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.

    step = 1. # Magnitude of each gradient update
    for i in range(40):   # Runs gradient ascent for 40 steps
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step
    img = input_img_data[0]  # gray
    return deprocess_image(img) # numpy→image format
```

a polka-dot pattern

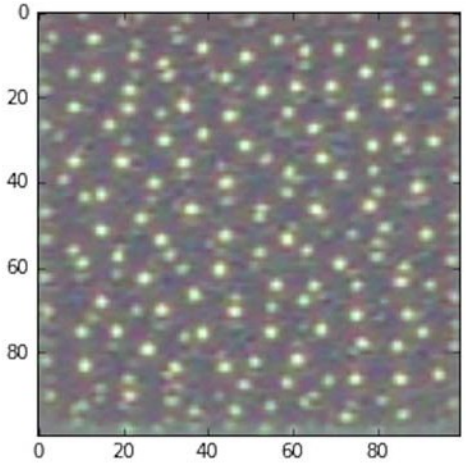


Figure 5.29 Pattern that the zeroth channel in layer block3_conv1 responds to maximally

- Let's try it (see figure 5.29):

```
>>> plt.imshow(generate_pattern('block3_conv1', 0))
```

5.4 Visualizing what convnets learn

- It seems that filter 0 in layer `block3_conv1` is responsive to a polka-dot pattern.
- look at the first 64 filters in each layer of convolution blocks - `block1_conv1`, `block2_conv1`, `block3_conv1`, `block4_ conv1`, `block5_conv1`
- Arrange the outputs on an 8 × 8 grid of 64 × 64 filter patterns, with some black margins between each filter pattern

Listing 5.39 Generating a grid of all filter response patterns in a layer

```
for layer_name in ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1']:
    size = 64
    margin = 5
    # 결과를 담을 빈 (검은) 이미지
    results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3), dtype='uint8')

    for i in range(8):  # results 그리드의 행을 반복합니다
        for j in range(8):  # results 그리드의 열을 반복합니다
            # layer_name에 있는 i + (j * 8)번째 필터에 대한 패턴 생성합니다
            filter_img = generate_pattern(layer_name, i + (j * 8), size=size)

            # results 그리드의 (i, j) 번째 위치에 저장합니다
            horizontal_start = i * size + i * margin
            horizontal_end = horizontal_start + size
            vertical_start = j * size + j * margin
            vertical_end = vertical_start + size
            results[horizontal_start: horizontal_end, vertical_start: vertical_end, :] = filter_img

    # results 그리드를 그립니다
    plt.figure(figsize=(20, 20))
    plt.imshow(results)
    plt.show()
```

5.4 Visualizing what convnets learn

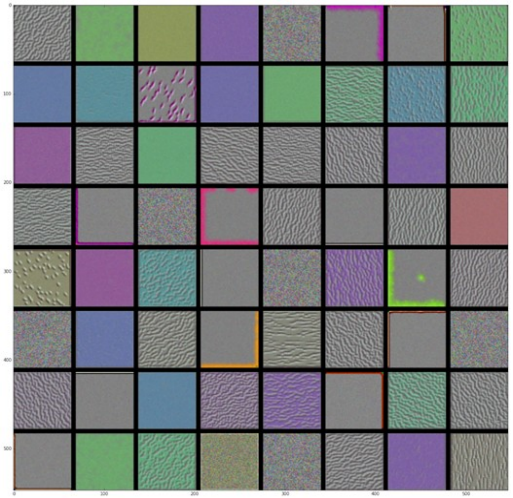
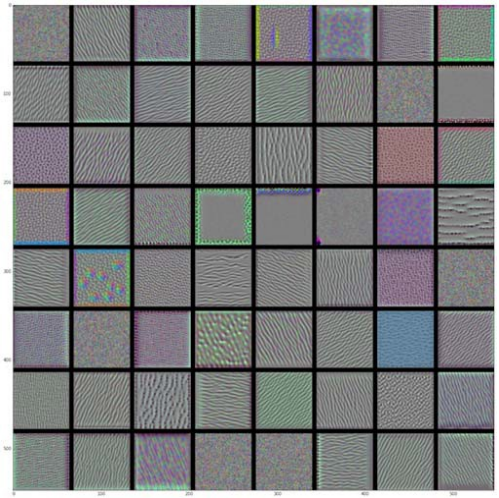
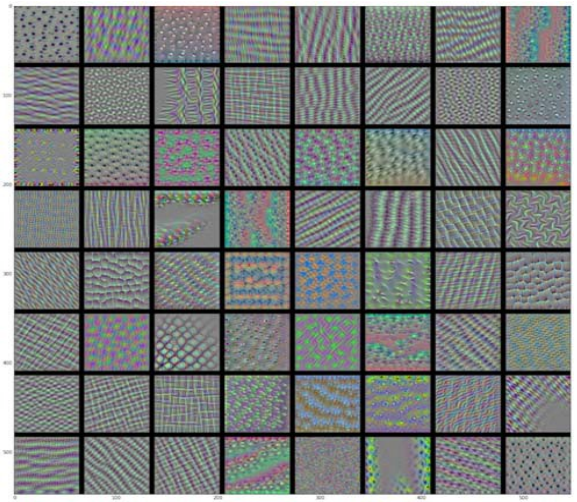


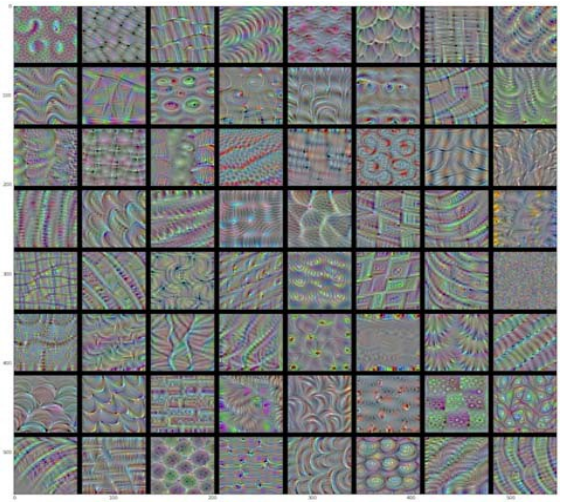
Figure 5.30 Filter patterns for layer `block1_conv1`



Filter patterns for layer `block2_conv1`



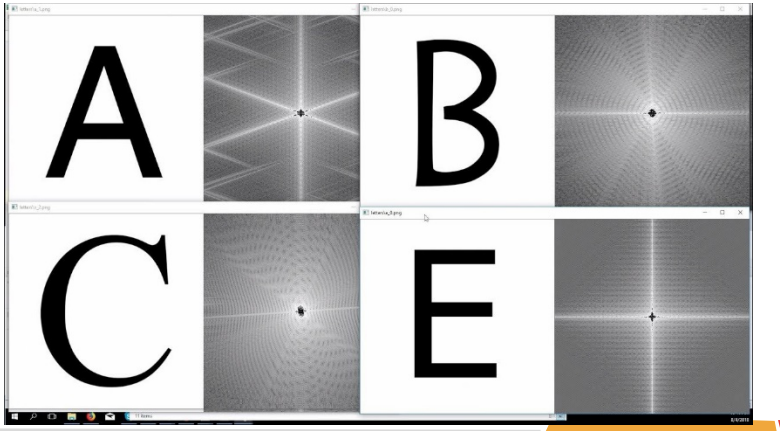
Filter patterns for layer `block3_conv1`



Filter patterns for layer block4_conv1

5.4 Visualizing what convnets learn

- These filter visualizations - how convnet layers see the world
- This is similar to how the Fourier transform decomposes signals onto a bank of cosine functions.
- The filters in these convnet filter banks get increasingly complex and refined as you go higher in the model:
  - The filters from the first layer in the model (`block1_conv1`) encode simple directional edges and colors (or colored edges, in some cases).
  - The filters from `block2_conv1` encode simple textures made from combinations of edges and colors.
  - The filters in higher layers begin to resemble textures found in natural images: feathers, eyes, leaves, and so on.



Fourier transform

5.4 Visualizing what convnets learn

5.4.3 Visualizing heatmaps of class activation

- heatmaps - debugging the decision process of a convnet of the case of a classification mistake, finding specific objects locations in an image.
- *class activation map* (CAM) - producing heatmaps of class activation over input images
- CAM is a 2D grid of scores associated with a specific output class - how important each location is with respect to the class under consideration.
- "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization."
- output feature map of a convolution layer → weighing every channel in that feature map by the gradient of the class with respect to the channel.
- weighting a spatial map - "how intensely the input image activates the class."

5.4 Visualizing what convnets learn

5.4.3 Visualizing heatmaps of class activation

- demonstrate this technique using the pretrained VGG16 network

Listing 5.39 Generating a grid of all filter response patterns in a layer

```
from keras.applications.vgg16
import VGG16 model = VGG16(weights='imagenet')
# include the densely connected classifier on top
```

- Consider the image of two African elephants (under a Creative Commons license)
- Convert this image into something the VGG16 model can read
- The model was trained on images of size 224 × 244, preprocessed according to a few rules that are packaged in the utility function `keras.applications.vgg16.preprocess_input`.
- Load the image, resize to 224 × 224, convert it to a Numpy `float32` tensor, and apply these preprocessing rules.

5.4 Visualizing what convnets learn

5.4.3 Visualizing heatmaps of class activation

Listing 5.41 Preprocessing an input image for VGG16

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np
img_path = './datasets/creative_commons_elephant.jpg'
# 224 × 224 크기의 Python Imaging Library (PIL) image 객체로 반환
img = image.load_img(img_path, target_size=(224, 224))

#float32 Numpy array of shape (224, 224, 3)
x = image.img_to_array(img)

# Adds a dimension to transform the array into a batch of size (1, 224, 224, 3)
x = np.expand_dims(x, axis=0)

# Preprocesses the batch (this does channel-wise color normalization)
x = preprocess_input(x)
```

- You can now run the pretrained network on the image and decode its prediction vector back to a human-readable format:

```
>>> preds = model.predict(x)
>>> print('Predicted:', decode_predictions(preds, top=3)[0])
Predicted:', [(u'n02504458', u'African_elephant', 0.92546833), (u'n01871265',
u'tusker', 0.070257246),(u'n02504013', u'Indian_elephant', 0.0042589349)]
```

5.4 Visualizing what convnets learn

5.4.3 Visualizing heatmaps of class activation

- The top three classes predicted for this image are as follows:
  - African elephant (with 92.5% probability)
  - Tusker (with 7% probability)
  - Indian elephant (with 0.4% probability)
- The entry in the prediction vector that was maximally activated is the one corresponding to the "African elephant" class, at index 386:

```
>>> np.argmax(preds[0]) 386
```

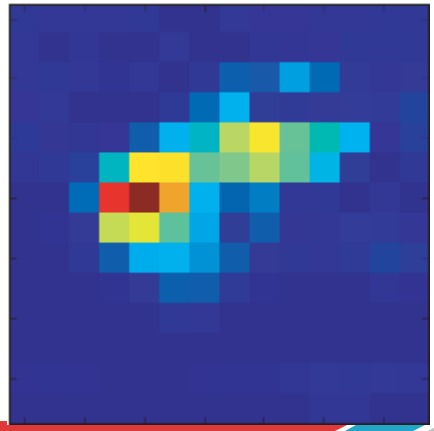
- To visualize which parts of the image are the most African elephant–like, let's set up the Grad-CAM process.

5.4 Visualizing what convnets learn

5.4.3 Visualizing heatmaps of class activation

Listing 5.42 Setting up the Grad-CAM algorithm

```
# 예측 벡터의 '아프리카 코끼리' 항목
african_elephant_output = model.output[:, 386]
# VGG16의 마지막 합성곱 층인 block5_conv3 층의 특성 맵
last_conv_layer = model.get_layer('block5_conv3')
# block5_conv3의 특성 맵 출력에 대한 '아프리카 코끼리' 클래스의 그래디언트
grads = K.gradients(african_elephant_output, last_conv_layer.output)[0]
# 특성 맵 채널별 그래디언트 평균 값이 담긴 (512,) 크기의 벡터
pooled_grads = K.mean(grads, axis=(0, 1, 2)) # loss
# 샘플 이미지가 주어졌을 때 방금 전 정의한 pooled_grads와 block5_conv3의 특성 맵 출력을 구합니다
iterate = K.function([model.input], [pooled_grads, last_conv_layer.output[0]])
# 두 마리 코끼리가 있는 샘플 이미지를 주입하고 두 개의 넘파이 배열을 얻습니다
pooled_grads_value, conv_layer_output_value = iterate([x])
# "아프리카 코끼리" 클래스에 대한 "채널의 중요도"를 특성 맵 배열의 채널에 곱합니다
for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]
# 만들어진 특성 맵에서 채널 축을 따라 평균한 값이 클래스 활성화의 히트맵입니다
heatmap = np.mean(conv_layer_output_value, axis=-1)
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
plt.show()
```



5.4 Visualizing what convnets learn

5.4.3 Visualizing heatmaps of class activation

Listing 5.44 5.44 Superimposing the heatmap with the original picture

```
import cv2
# cv2 모듈을 사용해 원본 이미지를 로드합니다
img = cv2.imread(img_path)
# heatmap을 원본 이미지 크기에 맞게 변경합니다
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
# heatmap을 RGB 포맷으로 변환합니다
heatmap = np.uint8(255 * heatmap)
# 히트맵으로 변환합니다
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
# 0.4는 히트맵의 강도입니다
superimposed_img = heatmap * 0.4 + img
# 디스크에 이미지를 저장합니다
cv2.imwrite('./datasets/elephant_cam.jpg', superimposed_img)
```

5.4 Visualizing what convnets learn

5.4.3 Visualizing heatmaps of class activation

- This visualization technique answers two important questions:
  - Why did the network think this image contained an African elephant?
  - Where is the African elephant located in the picture?
- In particular, it's interesting to note that the ears of the elephant calf are strongly activated: this is probably how the network can tell the difference between African and Indian elephants.

5.4 Visualizing what convnets learn

실습

- cats_dogs predict
- 각 10장의 이미지 결과
- 실행 분석

```
preds = model.predict(x)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

Predicted: [('n02098286', 'West_Highland_white_terrier', 0.10355645), ('n02095889', 'Sealyham_terrier', 0.07420319), ('n02085936', 'Maltese_dog', 0.06565421)]