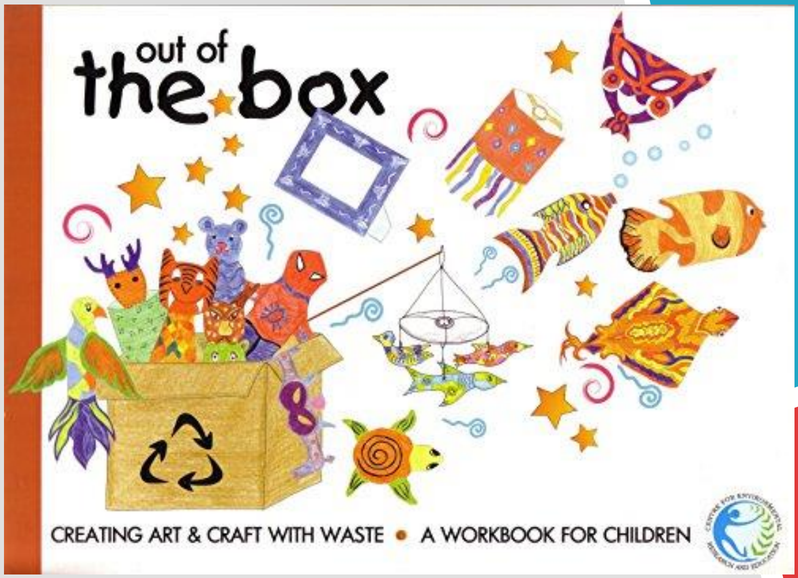
5장 *Deep learning for computer vision*

"Out of the Box"



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

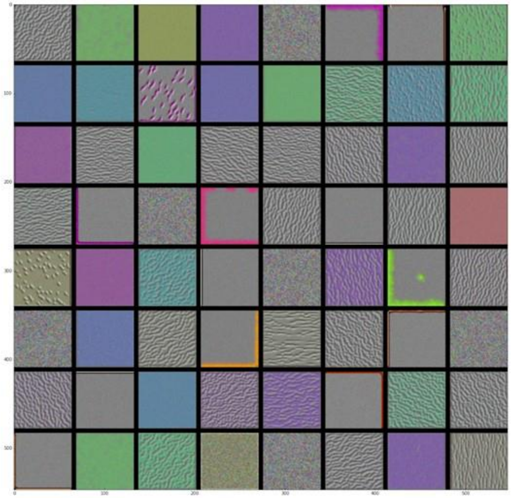
```
for layer_name in ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1']:
    size = 64
    margin = 5
    # 결과를 담을 빈 (검은) 이미지
    results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3), dtype='uint8')

    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()
```

5.4Visualizing what convnets learn



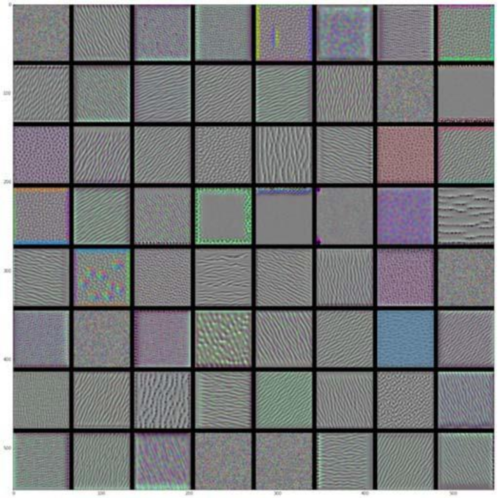
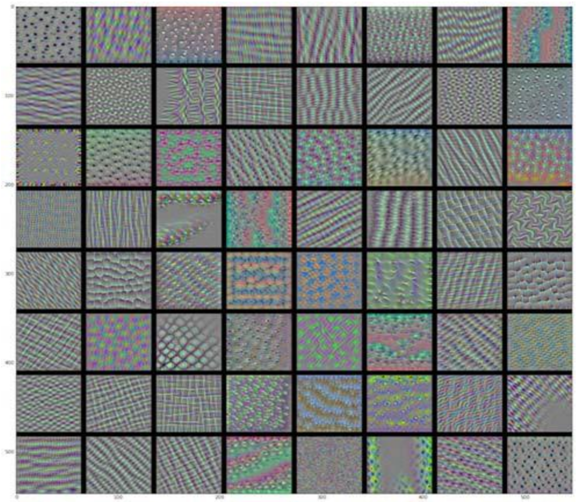
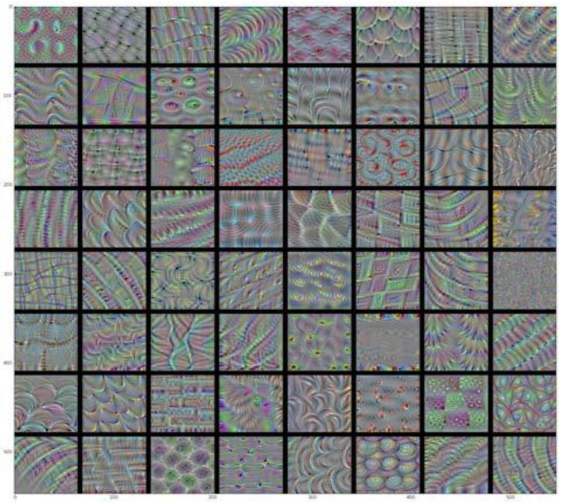


Figure 5.30 Filter patterns for layer `block1_conv1`　**Filter patterns for layer `block2_conv1`**

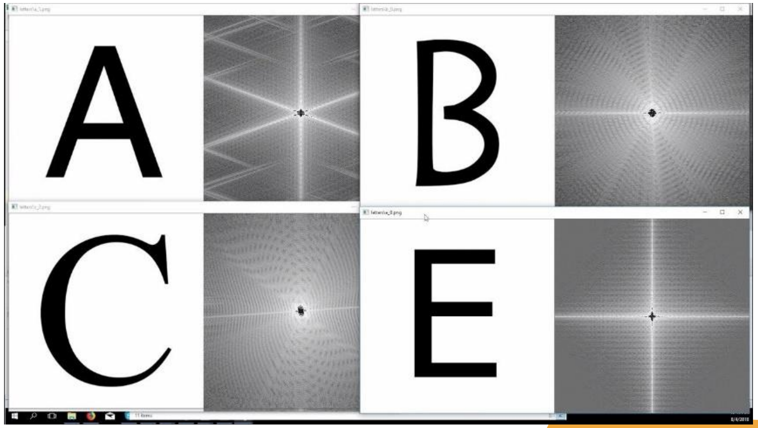




Filter patterns for layer `block3_conv1`　**Filter patterns for layer block4_conv1**

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)]
```

5.4.3 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:

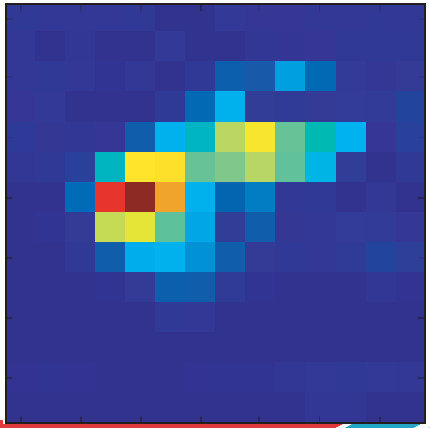
```
>>> np.argmax(preds[0]) 386
```

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

5.4.3 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()
```



5.4.3 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 acti- vated: this is probably how the network can tell the difference between African and Indian elephants.