5.4 컨브넷 학습 시각화

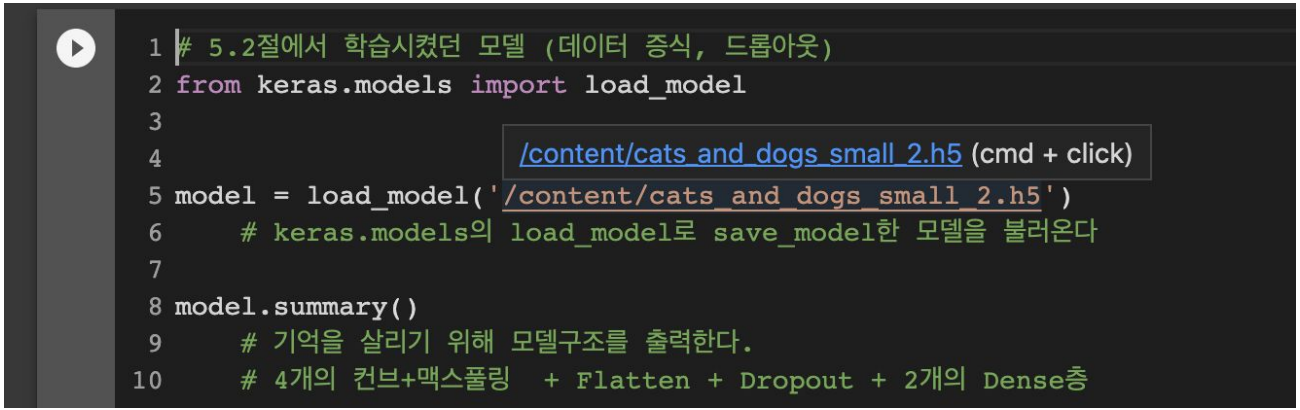
[가장 사용이 편하고 유용한 세 가지 기법]

1. **컨브넷 중간층의 출력을 시각화**

입력이 들어올 때,
네트워크에 있는 여러 합성곱과 풀링층이 출력하는 출력을 그리는 것

2. 컨브넷 필터 시각화
3. 클래스 활성화에 대한 히트맵을 이미지에 시각화

5.4.1 중간층의 활성화 시각화하기



```
# 5.2절에서 학습시켰던 모델 (데이터 증식, 드롭아웃)
from keras.models import load_model

                        /content/cats_and_dogs_small_2.h5 (cmd + click)
model = load_model('/content/cats_and_dogs_small_2.h5')
    # keras.models의 load_model로 save_model한 모델을 불러온다

model.summary()
    # 기억을 살리기 위해 모델구조를 출력한다.
    # 4개의 컨브+맥스풀링  + Flatten + Dropout + 2개의 Dense층
```

중간층 활성화 시각화 :

입력이 주어지면 여러 합성곱과 풀링 층이 출력하는 특성 맵 시각화

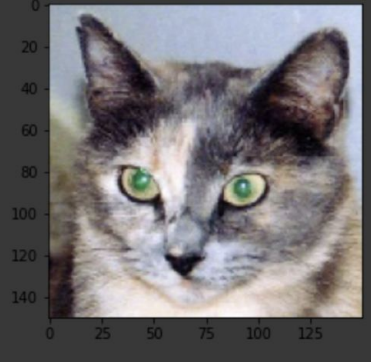
네트워크에 의해 학습된 필터들이 어떻게 입력을 분해하는지 보여준다

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1568)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1568)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               803328
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 832,481
Trainable params: 832,481
Non-trainable params: 0
_________________________________________________________________
```

```
# 5-25 개별 이미지 전처리하기
img_path = '/content/drive/My Drive/Colab Notebooks/DLWP/cats_and_dogs_small/test/cats/cat.1700.jpg'

from keras.preprocessing import image
import numpy as np

img = image.load_img(img_path, target_size=(150,150)) # PIL image instances를 반환, (path, gray_scale, color_mode, target_size, interpolation)
img_tensor = image.img_to_array(img)                  # PIL image instances를 Numpy array로 반환
print('expand 전:',img_tensor.shape)

img_tensor = np.expand_dims(img_tensor, axis=0) # 이미지를 4D 텐서로 변경, 맨 앞 차원에 1이 생김, 이미지 1개를 한차원 늘려서 보는 거라 맨앞(sample)이 1
img_tensor /= 255.                              # 훈련할 때 하던 전처리 방식 그대로 사용

print('expand 후:',img_tensor.shape)
```

```
expand 전: (150, 150, 3)
expand 후: (1, 150, 150, 3)
```

```
# 5-26 테스트 사진 출력하기
import matplotlib.pyplot as plt

plt.imshow(img_tensor[0]) # img_tensor가 4D이므로 인덱싱으로 차원을 하나 벗겨서 3D로 입력값에 전달
plt.show()
```



```
# 5-27 입력 텐서와 출력 텐서의리스트로 모델 객체 만들기

from keras import models   # 모든 합성곱과 풀링 층의 활성화를 출력하는 케라스 모델
                           # Sequential과는 달리 여러 출력을 가진 모델 생성 가능
                           # 자세한 내용은 7.1절 참고
layer_outputs = [layer.output for layer in model.layers[:8]] # Conv2D+MaxPooling2D 총 4셋트가 있는 8개의 층에서, 각 층의 output들을 리스트에 저장
activation_model = models.Model(inputs = model.input, outputs = layer_outputs)  # 입력에 대해 8개의 층을 반환하는 모델 생성, 원래 입출 1개씩이었는데 여긴 출력 8개
```

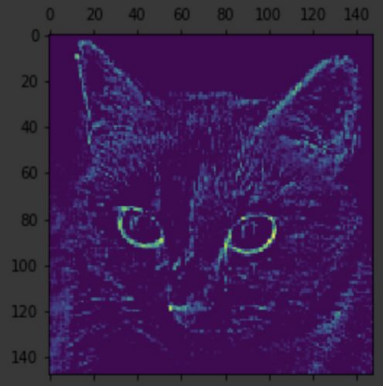
```
# 5-28 예측 모드로 모델 실행하기
activations = activation_model.predict(img_tensor) # 각 층이 활성화 될 때 마다 출력이 넘파이 배열로 반환
```

```
first_layer_activation=activations[0]
print(first_layer_activation.shape) # 원래 (1,150,150,3)이었는데 filter 크기가 3x3이라 가로,세로가 2씩 줄어듬.
```

```
(1, 148, 148, 32)
```

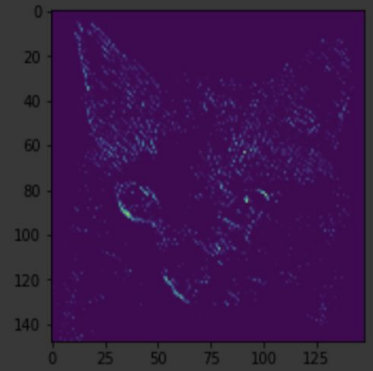
```
# 5-29 20번째 채널 시각화하기              총 32개의 채널 중에서 20번째 채널
# 대각선 에지를 감지하도록 인코딩된 채널
import matplotlib.pyplot as plt
plt.matshow(first_layer_activation[0,:,:,19],cmap = 'viridis')
```

```
<matplotlib.image.AxesImage at 0x7f8b998bfd30>
```



```
# 5-30 16번째 채널 시각화하기
# 밝은 녹색 점을 감지하는 채널
plt.imshow(first_layer_activation[0,:,:,15],cmap='viridis')
```

```
<matplotlib.image.AxesImage at 0x7f8b3c3df908>
```

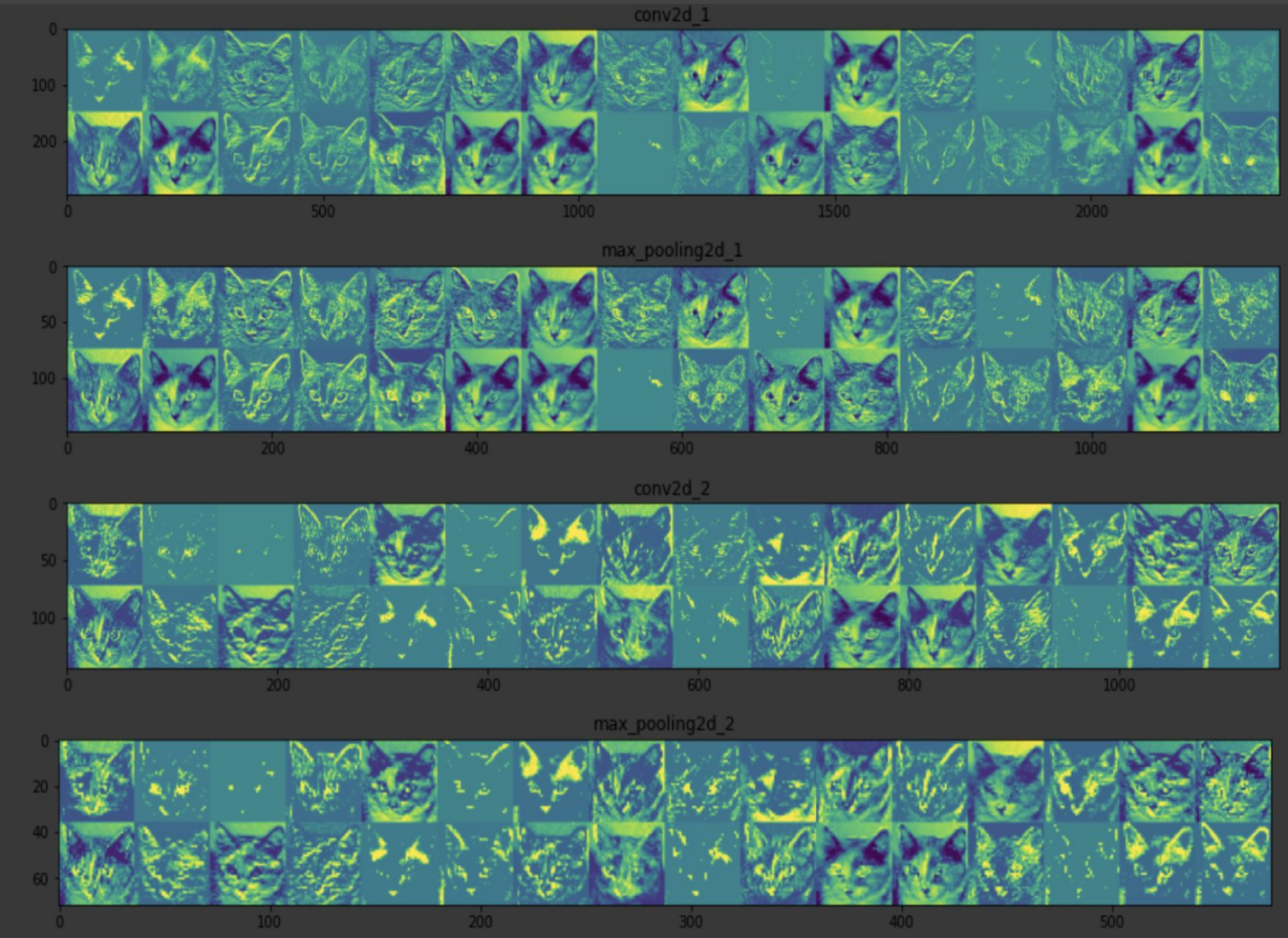


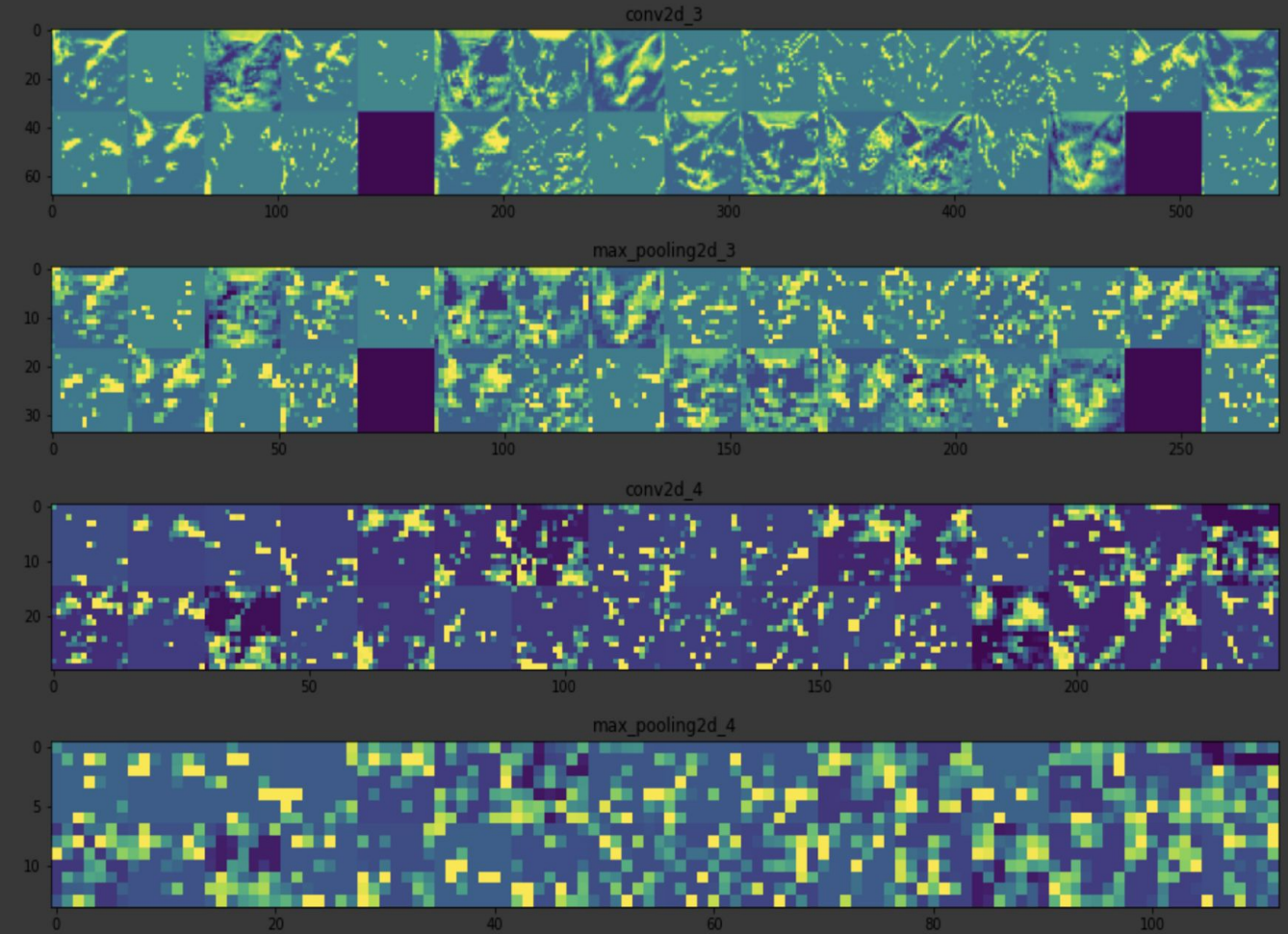

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1568)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1568)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               803328
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 832,481
Trainable params: 832,481
Non-trainable params: 0
_________________________________________________________________
```

▼ 중간층의 모든 활성화에 있는 채널 시각화하기

```
# 5-31 중간층의 모든 활성화에 있는 채널 시각화하기
# 층의 이름을 그래프 제목으로 사용합니다
layer_names =[]
for layer in model.layers[:8]:
  layer_names.append(layer.name)

images_per_row = 16
for layer_name, layer_activation in zip(layer_names, activations):    # 특성 맵을 그립니다
  n_features = layer_activation.shape[-1]     # 특성 맵에 있는 특성의 수

  size = layer_activation.shape[1]            #특성맵의 크기는(1, size, size, n_features)
  n_cols = n_features // images_per_row       #활성화 채널을 위한 그리드 크기 구하기
  display_grid = np.zeros((size * n_cols, images_per_row * size))

  for col in range(n_cols):                   # 각 활성화를 하나의 큰 그리드에 채운다
    for row in range(images_per_row):
      channel_image = layer_activation[0,:,:,col * images_per_row + row]
      channel_image -= channel_image.mean()   # 그래프로 나타내기 좋게 특성 처리
      channel_image /= channel_image.std()
      channel_image *= 64
      channel_image += 128
      channel_image = np.clip(channel_image,0,255).astype('uint8')
      display_grid[col * size : (col + 1) * size,     # 그리드를 출력
                   row * size : (row + 1) * size] = channel_image

  scale = 1./size
  plt.figure(figsize=(scale * display_grid.shape[1],
                      scale * display_grid.shape[0]))
  plt.title(layer_name)
  plt.grid(False)
  plt.imshow(display_grid,aspect='auto', cmap='viridis')
plt.show()
```


conv2d_1
0
100
200
0
500
1000
1500
2000
max_pooling2d_1
0
50
100
0
200
400
600
800
1000
conv2d_2
0
50
100
0
200
400
600
800
1000
max_pooling2d_2
0
20
40
60
0
100
200
300
400
500


conv2d_3
max_pooling2d_3
conv2d_4
max_pooling2d_4

5.4.2 컨브넷 필터 시각화하기

컨브넷이 학습한 필터를 조사하는 방법 중 하나는 각 필터가 반응하는 시각적 패턴을 그려보는 것입니다.

빈 입력 이미지에서 시작해서 특정 필터의 응답을 최대화하기 위해 컨브넷 입력 이미지에 경사 상승법을 적용합니다.

결과적으로 입력 이미지는 선택된 필터가 최대로 응답하는 이미지가 될 것입니다.

5.3.2 필터 시각화를 위한 손실텐서 정의하기

```
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
loss = K.mean(layer_output[:, :, :, filter_index])
```

손실 텐서를 정해야합니다. 책의 예시는 ImageNet에 사전 훈련된 VGG16 네트워크에서 block3_conv1 층 필터 0번의 활성화를 손실로 정의합니다.

5.3.3~ 5.3.5

```
grads = K.gradients(loss, model.input)[0]

grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

iterate = K.function([model.input],[loss,grads])

import numpy as np
loss_value, grads_value = iterate([np.zeros((1,150,150,3))])
```

gradients 함수가 반환하는 텐서 리스트에서 추출한 후 L2 노름으로 나누어 정규화 합니다.

주어진 입력 이미지에 대해 손실 텐서와 그래디언트 텐서를 계산해야 합니다.

5.36 경사 상승법 구현

```
# 코드 5-36 확률적 경사 상승법을 사용한 손실 최대화하기

# 잡음이 섞인 회색 이미지로 시작합니다
input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.

# 업데이트할 그레디언트의 크기
step = 1.
for i in range(40):   # 경사 상승법을 40회 실행합니다
    # 손실과 그레디언트를 계산합니다
    loss_value, grads_value = iterate([input_img_data])
    # 손실을 최대화하는 방향으로 입력 이미지를 수정합니다
    input_img_data += grads_value * step
```

경사 상승법을 사용하기 때문에 옵티마이저를 사용할 수 없어서 직접 학습 단계를 구현해야 합니다.

처음엔 잡음이 섞인 회색이미지에서 점차 특정 필터가 나타내는 이미지 값으로 바뀌게 됩니다.

5.37 텐서를 이미지로

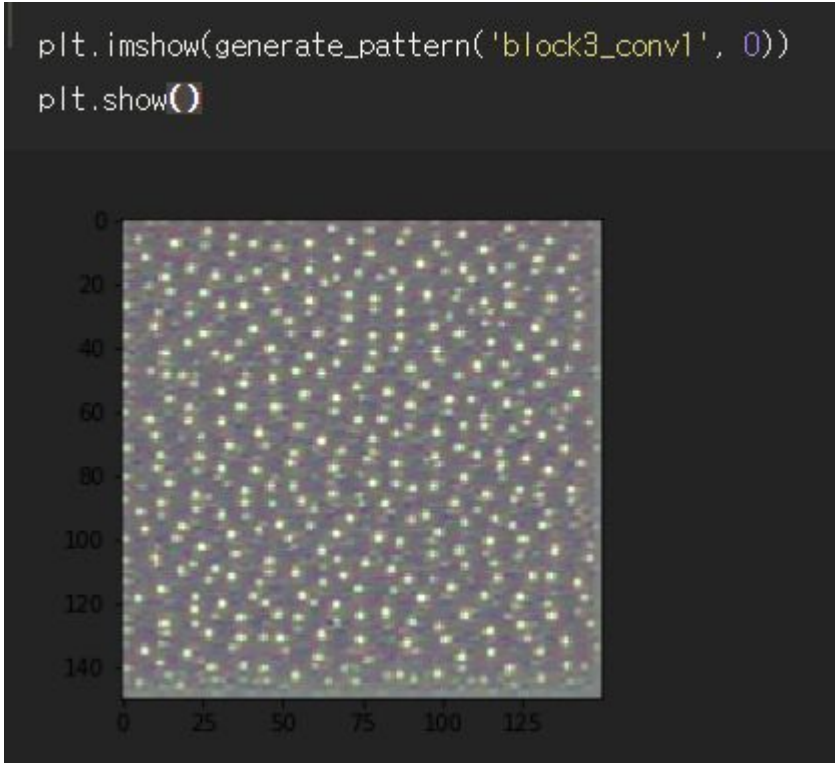
```python
# 코드 5-37 텐서를 이미지 형태로 변환하기 위한 유틸리티 함수

def deprocess_image(x):
    # 텐서의 평균이 0, 표준 편차가 0.1이 되도록 정규화합니다
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1

    # [0, 1]로 클리핑합니다
    x += 0.5
    x = np.clip(x, 0, 1)

    # RGB 배열로 변환합니다
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

경사 상승법으로 구현한
텐서를 출력 가능한 이미지로
변경하기 위한 후처리
코드입니다.

5.38 시각화

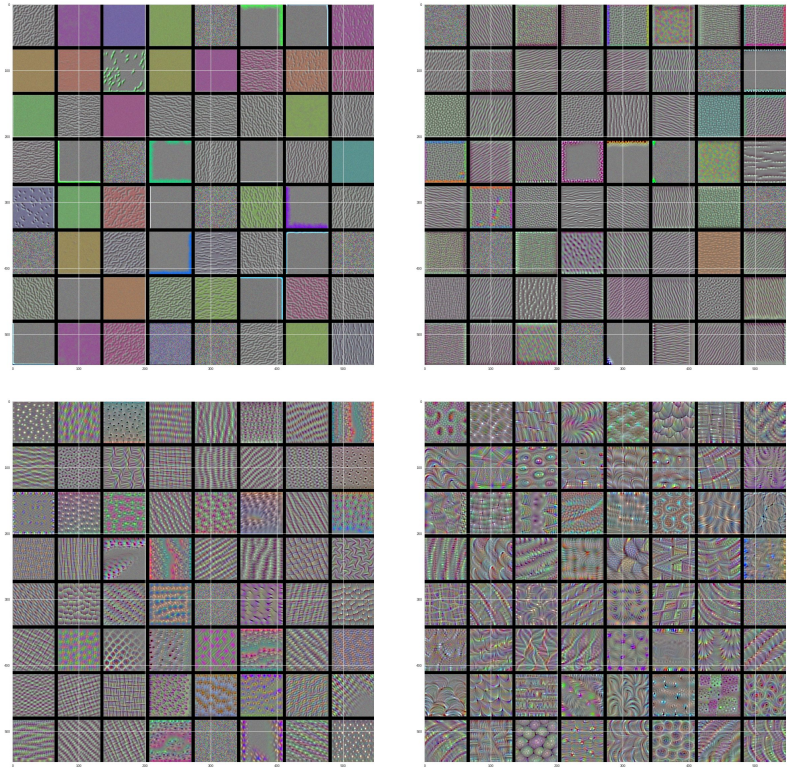
```
def generate_pattern(layer_name, filter_index, size=150):
    # 주어진 층과 필터의 활성화를 최대화하기 위한 손실 함수를 정의합니다
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])

    # 손실에 대한 입력 이미지의 그래디언트를 계산합니다
    grads = K.gradients(loss, model.input)[0]

    # 그래디언트 정규화
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

    # 입력 이미지에 대한 손실과 그래디언트를 반환합니다
    iterate = K.function([model.input], [loss, grads])

    # 잡음이 섞인 회색 이미지로 시작합니다
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.

    # 경사 상승법을 40 단계 실행합니다
    step = 1.
    for i in range(40):
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step

    img = input_img_data[0]
```

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



5.39 모든 필터 시각화

```
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()
```



5.42 정리

이런 필터 시각화를 통해 컨브넷 층이 바라보는 방식을 이해할 수 있습니다.

이 컨브넷 필터들은 모델의 상위 층으로 갈수록 점점 더 복잡해지고 개선됩니다.

- 모델에 있는 첫 번째 층의 필터는 간단한 대각선 방향의 에지와 색깔을 인코딩합니다.
- 에지나 색깔의 조합으로 만들어진 간단한 질감을 인코딩합니다.
- 더 상위 층 필터는 자연적인 이미지에서 찾을 수 있는 질감을 닮아 가기 시작합니다.

5.4.3 클래스 활성화의 히트맵 시각화하기

이 방법은 이미지의 어느 부분이 컨브넷의 최종 분류 결정에 기여하는지 이해하는 데 유용합니다. 분류에 실수가 있는 경우 컨브넷의 결정 과정을 디버깅 하는 데 도움이 됩니다. 또 특정 물체가 있는 위치를 파악하는 데 사용할 수도 있습니다.

5.40 사전 훈련된 가중치로 VGG16 네트워크 로드하기

```
from keras.applications.vgg16 import VGG16

model = VGG16(weights = 'imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467904/553467096 [==============================] - 388s 1us/step
```

사전 훈련된 VGG16 네트워크를 가지고 코끼리 이미지로 시각화해 보도록 하겠습니다.

5.41 VGG16을 위해 입력 이미지 전처리하기

```python
# 코드 5-41 VGG16을 위해 입력 이미지 전처리하기

from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

# 이미지 경로
img_path = './datasets/creative_commns_elephant.jpg'

# 224 × 224 크기의 파이썬 이미징 라이브러리(PIL) 객체로 반환됩니다
img = image.load_img(img_path, target_size=(224, 224))

# (224, 224, 3) 크기의 넘파이 float32 배열
x = image.img_to_array(img)

# 차원을 추가하여 (1, 224, 224, 3) 크기의 배치로 배열을 변환합니다
x = np.expand_dims(x, axis=0)

# 데이터를 전처리합니다(채널별 컬러 정규화를 수행합니다)
x = preprocess_input(x)
```

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

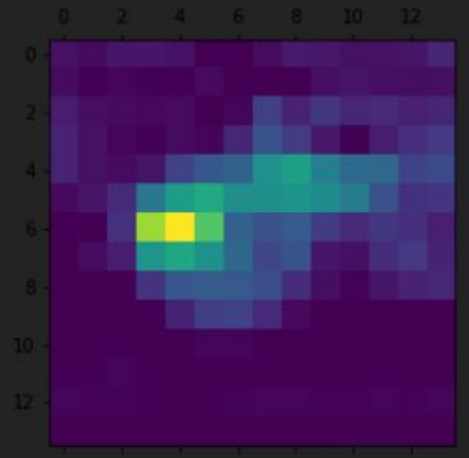
```
WARNING:tensorflow:From C:\Users\user\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training_v1.py:2070: Model.state_updates (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
40960/35363 [==================================] - 0s 2us/step
Predicted: [('n02504458', 'African_elephant', 0.909421), ('n01871265', 'tusker', 0.086182885), ('n02504013', 'Indian_elephant', 0.0043545826)]
```

아프리카 코끼리 91프로
터스커 8프로
인도 코끼리0.4프로

5.42 Grad-CAM 알고리즘 설정하기

```
# 코드 5-42 Grad-CAM 알고리즘 설정하기

idx_ele = np.argmax(preds[0]) # 386

# 예측 벡터의 '아프리카 코끼리' 항목
african_elephant_output = model.output[:, idx_ele]

# 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))

# 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)
```

```
# 코드 5-43 히트맵 후처리하기
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
plt.show()
```



5.44 원본 이미지에 히트맵 덧붙이기

```
# 코드 5-44 원본 이미지에 히트맵 덧붙이기

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_cam1.jpg', superimposed_img)
```

5.5 요약

컨브넷은 시각적인 분류 문제를 다루는 데 최상의 도구입니다.

컨브넷은 우리가 보는 세상을 표현하기 위한 패턴의 계층 구조와 개념을 학습합니다.

학습된 표현은 쉽게 분석할 수 있습니다.

시각화를 할 수 있습니다.