9 - c 중간 활성화 시각화

컨브넷이 학습한 것 해석하기

1. 컨브넷 중간층 출력 시각화 (중간층 활성화)
2. 컨브넷 필터 시각화
3. 클래스 활성화 히트맵 시각화

중간층 활성화 시각화

rescaling_1 (Rescaling)	(None, 180, 180, 3)	0
conv2d_5 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_4 (MaxPooling2	(None, 89, 89, 32)	0
conv2d_6 (Conv2D)	(None, 87, 87, 64)	18496
max_pooling2d_5 (MaxPooling2	(None, 43, 43, 64)	0
conv2d_7 (Conv2D)	(None, 41, 41, 128)	73856
max_pooling2d_6 (MaxPooling2	(None, 20, 20, 128)	0
conv2d_8 (Conv2D)	(None, 18, 18, 256)	295168
max_pooling2d_7 (MaxPooling2	(None, 9, 9, 256)	0
conv2d_9 (Conv2D)	(None, 7, 7, 256)	590080

총 9개의 conv와 max_pool 가지고 있음

첫번째는 1,178,178,32

두번째는 1, 89, 89, 32 ...

```
from tensorflow import keras
import numpy as np

# 훈련에서 사용하지 않은 테스트 이미지를 내려받는다.
img_path = keras.utils.get_file(
    fname="cat.jpg",
    origin="https://img-datasets.s3.amazonaws.com/cat.jpg")

def get_img_array(img_path, target_size):
    img = keras.utils.load_img(
        img_path, target_size=target_size)  # 1.

    array = keras.utils.img_to_array(img)   # 2.

    array = np.expand_dims(array, axis=0)   # 3.
    return array

img_tensor = get_img_array(img_path, target_size=(180, 180))
```

1. 이미지 파일을 로드하고 크기를 맞춘다. 여기서는 (180,180)
2. 이미지를 (180, 180, 3) 크기의 float32 넘파이 배열로 변환한다.
3. 배열을 단일 이미지의 '배치'로 변환하기 위해서 차원을 추가한다. 배열은 (1, 180, 180, 3) 크기가 된다.

```
from tensorflow.keras import layers

layer_outputs = []
layer_names = []

# 모든 Conv2D와 MaxPooling2D층의 출력을 하나의 리스트에 추가한다.
for layer in model.layers:
    if isinstance(layer, (layers.Conv2D, layers.MaxPooling2D)): # 1.
        layer_outputs.append(layer.output)
        layer_names.append(layer.name)  # 나중을 위해 층 이름을 저장한다.

# 모델 입력이 주어졌을 때 층의 출력을 반환하는 모델을 만든다.
activation_model = keras.Model(inputs=model.input, outputs=layer_outputs)
```

1. layer가 Conv2D나 MaxPooling2D층 둘 중 하나에 속하면 추가한다.

```
# 층 활성봐마다 배열 하나씩 총 9개의 넘파이 배열로 구성된 리스트를 반환한다.
# 다중 출력 모델 (9개의 layers)
activations = activation_model.predict(img_tensor)
```

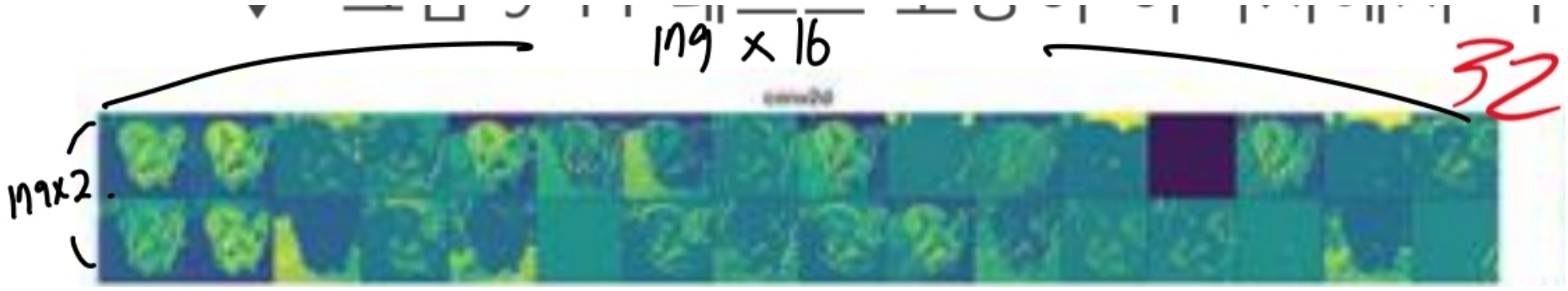
```
first_layer_activation = activations[0] # 첫번째 합성곱 층의 활성화 값
print(first_layer_activation.shape)

import matplotlib.pyplot as plt
# 첫번쨔 활성화 중 6번째 채널
plt.matshow(first_layer_activation[0, :, :, 5], cmap="viridis")
plt.show()
```

모든 층의 활성화에 있는 전체 채널 시각화하기

하나의 큰 이미지 그리드에시각화한다.

```
images_per_row = 16 # 가로 줄은 16개

# 활성화(그리고 해당 층 이름)에 대해 루프를 순회 - 총 9번 순회
for layer_name, layer_activation in zip(layer_names, activations):

    # 층 활성화 크기는 (1, size, size, n_features)
    n_features = layer_activation.shape[-1] # 1층은 32,
    size = layer_activation.shape[1] # 178?

    n_cols = n_features // images_per_row # 1층은 2

    # 활성화에 있는 모든 채널을 출력하기 위한 빈 그래프를 준비한다.
    display_grid = np.zeros(((size + 1) * n_cols - 1,
                             images_per_row * (size + 1) - 1))

    for col in range(n_cols):
        for row in range(images_per_row):
            channel_index = col * images_per_row + row
            channel_image = layer_activation[0, :, :, channel_index].copy() # 하나의 채널(또는 특성) 이미지이다.

            # 채널 값을 [0,255] 범위로 정규화한다. 모두 0인 채널은 그대로 둔다.
            if channel_image.sum() != 0:
                channel_image -= channel_image.mean()
                channel_image /= channel_image.std()
                channel_image *= 64
                channel_image += 128
```

```
            channel_image = np.clip(channel_image, 0, 255).astype("uint8")

            display_grid[
                # 빈 그리드에 채널 행렬을 저장한다.
                col * (size + 1): (col + 1) * size + col,
                row * (size + 1) : (row + 1) * size + row
                ] = channel_image

    # 그리드를 출력한다.
    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.axis("off")
    plt.imshow(display_grid, aspect="auto", cmap="viridis")
```



이미지는 size x size 인데 이를 32개가 들어가는 그리드를 만드는것.

단계의 활성화에는 초기 이미지에 있는 모든 정보가 유지된다.

층이 깊어질수록 활성화는 점점 더 추상적으로 되고 시각적으로 이해하기 어려워진다.