Convolution 중간과정 시각화

최석재 *lingua@naver.com*

중간 과정 시각화

- 딥러닝 모델은 내부에서 진행되는 일을 파악하기 어려워 블랙박스 모델로 알려져 있지만
- 합성곱 신경망(Convolution Neural Network)은 시각 데이터를 학습하므로 중간 과정을 시각화 해 볼 수 있다
- 중간 과정 시각화를 통해 CNN 모델이 이미지를 어떻게 보고 있는지를 어느 정도 이해할 수 있다

중간 과정 시각화

중간 활성화 시각화

- CNN 아키텍처는 여러 층의 Convolution layer로 구성되어 있다
- 각 Convolution layer가 입력을 어떻게 변형하는지 알아보기 위해 그 출력을 시각화한다
- 층의 출력은 활성화 함수를 거쳐 이루어지므로 “중간 활성화 시각화” 라고도 한다

구글 드라이브 접속

- from google.colab import drive
- drive.mount('/content/gdrive')

- 경로 변경
- %cd /content/gdrive/MyDrive/pytest_img/opencv/
- !ls # retriever.png 파일 확인

대상 파일 출력

- from IPython.display import display
- from IPython.display import Image as _Imgdis
- import numpy as np

- display(_Imgdis(filename="retriever.png", width=600, height=400))

넘파이 배열로 로딩

- 신경망 네트워크에서 처리하기 위하여 이미지 파일을 넘파이 배열로 로드한다

파일 로딩

- from keras.preprocessing.image import array_to_img, img_to_array, load_img

불러올 모델이 (32, 32, 3)의 입력을 받은 파일로 만들어졌기 때문에 대상 파일도 이에 맞춘다

- img = load_img("retriever.png", target_size=(32, 32)) # (32, 32) 사이즈로 로딩
- print("Original:", type(img)) # PIL 타입 확인

```
Original: <class 'PIL.Image.Image'>
```

load_img() 함수는 PIL(Python Image Library) 타입으로 파일을 로드한다

넘파이 배열로 로딩

파일을 넘파이 배열로 변환

- img_array = img_to_array(img)
- print(type(img_array))
- print("type:", img_array.dtype)
- print("shape:", img_array.shape) # 높이, 너비, 채널

```
<class 'numpy.ndarray'>
type: float32
shape: (32, 32, 3)
```

배열 확인

3차원 넘파이 배열로 된 파일을 확인한다

- print(img_array)

```
array([[[207., 207., 207.],
        [207., 207., 207.],
        [203., 203., 203.],
        ...,
        [191., 190., 185.],
        [192., 191., 186.],
        [191., 190., 185.]],

       [[205., 205., 205.],
        [204., 204., 204.],
        [202., 202., 202.],
        ...,
        [189., 188., 183.],
        [189., 188., 183.],
        [188., 187., 182.]],

       [[202., 202., 202.],
        [198., 198., 198.],
        [203., 203., 203.],
        ...,
        [185., 184., 179.],
        [185., 184., 179.],
        [185., 184., 179.]],

       ...,
```

4D로 확장

- 사용할 모델은 4차원 데이터를 학습해 만들었으므로
- 대상 파일도 4차원 배열로 확장해야 한다

- img_array_4d = np.expand_dims(img_array, axis=0)
- print(type(img_array_4d))
- print("type:", img_array_4d.dtype)
- print("shape:", img_array_4d.shape) # 배치, 높이, 너비, 채널

```
<class 'numpy.ndarray'>
type: float32
shape: (1, 32, 32, 3)
```

배열 확인

4차원 넘파이 배열로 된 파일을 확인한다

- print(img_array_4d)

```
[[[[207. 207. 207.]
   [207. 207. 207.]
   [203. 203. 203.]
   ...
   [191. 190. 185.]
   [192. 191. 186.]
   [191. 190. 185.]]

  [[205. 205. 205.]
   [204. 204. 204.]
   [202. 202. 202.]
   ...
   [189. 188. 183.]
   [189. 188. 183.]
   [188. 187. 182.]]

  [[202. 202. 202.]
   [198. 198. 198.]
   [203. 203. 203.]
   ...
   [185. 184. 179.]
   [185. 184. 179.]
   [185. 184. 179.]]

  ...
```

모델 로딩

- 앞에서 만든 모델을 로딩하여 사용한다
- 사용할 모델은 3채널, 4차원의 CIFAR-10 이미지로 학습되었다

- import os
- from keras.models import load_model

- os.chdir("/content/gdrive/MyDrive/pytest_img/models")
- loaded_model = load_model("cifar10.h5")
- loaded_model.summary()

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896

max_pooling2d (MaxPooling2   (None, 15, 15, 32)        0
D)

conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496

conv2d_2 (Conv2D)            (None, 11, 11, 128)       73856

max_pooling2d_1 (MaxPoolin   (None, 5, 5, 128)         0
g2D)

conv2d_3 (Conv2D)            (None, 3, 3, 256)         295168

flatten (Flatten)            (None, 2304)              0

dense (Dense)                (None, 128)               295040

dropout (Dropout)            (None, 128)               0

dense_1 (Dense)              (None, 10)                1290

=================================================================
Total params: 684746 (2.61 MB)
Trainable params: 684746 (2.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

input 층 확인

- 현재 모델은 Sequential 모델로 만들어졌지만,
- 중간층을 분석하려면 이를 Functional API 방식으로 변환해야 한다
- Functional API는 모델의 각 층을 개별적으로 다룰 수 있게 한다

- Functional API는 모델의 시작점을 의미하는 inputs 매개변수와
- 모델의 마지막 지점을 의미하는 outputs 매개변수를 갖는다

- print(loaded_model.input)

```
KerasTensor(type_spec=TensorSpec(shape=(None, 32, 32, 3), dtype=tf.float32, name='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d_input'")
```

앞에서 모델을 만들 때 input_shape를 아래와 같이 하였다
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu'))

Functional API 모델로 변환

Sequntial 모델을 Functional API 모델로 변환한다

- from keras import models
- inputs = loaded_model.input # 로딩한 시퀀셜 모델의 입력을 사용

원하는 레이어의 출력을 추출하기 위한 레이어 리스트 설정

- layer_outputs = [loaded_model.layers[0].output, # conv2d
  loaded_model.layers[1].output, # max_pooling2d
  loaded_model.layers[2].output, # conv2d_1
  loaded_model.layers[3].output, # conv2d_2
  loaded_model.layers[4].output, # max_pooling2d_1
  loaded_model.layers[5].output] # conv2d_3

선택한 레이어의 출력을 사용하여 활성화 시각화 모델 생성

- activation_model = models.Model(inputs=inputs, outputs=layer_outputs)

Functional API는 input에서 output까지의 과정을 연결하고, 이를 Model 클래스에 전달한다
현재 모델에서는 inputs 매개변수를 통해 입력이 들어가고,
outputs 매개변수를 통해 layer_outputs의 각 층을 통과하여 최종 출력을 반환한다

이미지 처리

- 입력된 이미지가 모델을 통과할 때 각 레이어의 출력을 알아보기 위하여 predict 메소드 수행
  - predict 메소드를 사용하면 입력 이미지가 각 레이어를 통과할 때의 출력을 계산한다
  - 모델의 각 레이어가 주어진 입력에 대해 정의된 활성화 함수를 적용하여 출력을 생성하는 과정 확인 가능
- 각 레이어의 출력 결과, 즉 활성화 결과를 '중간 활성화 시각화'라고 한다
- 이를 통해 각 레이어가 입력 이미지의 어떤 특징을 추출하고 강조하는지 알 수 있다

- activations = activation_model.predict(img_array_4d)
- print(activations[0].shape)

```
1/1 [==============================] - 0s 63ms/step
(1, 30, 30, 32)
```

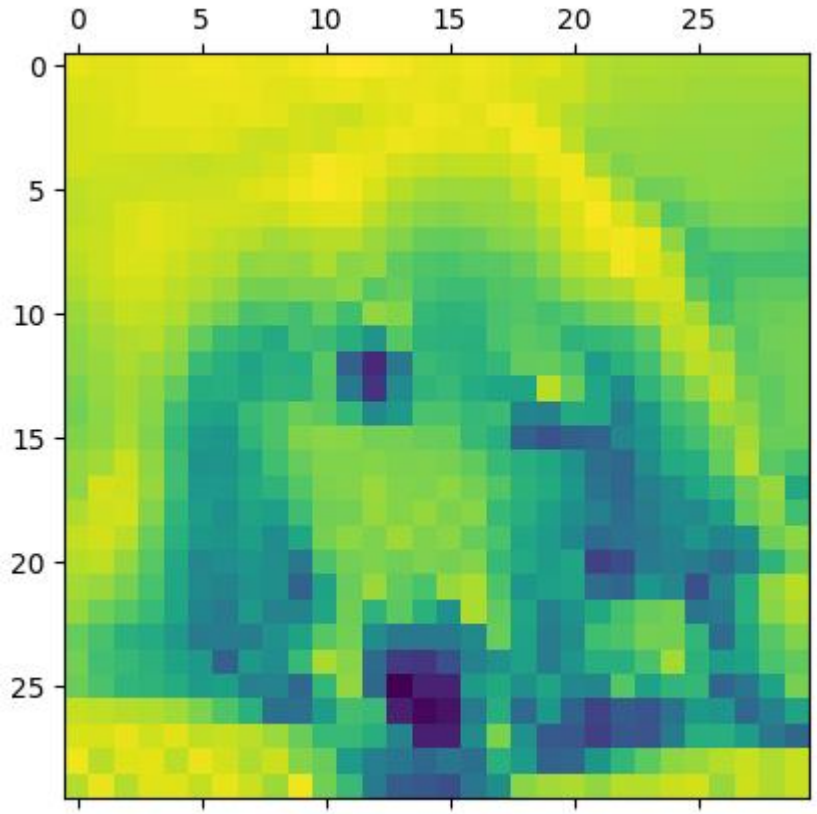
시각화

- plt.matshow()를 이용하여 행렬 데이터를 시각화
- 이 함수는 행렬의 값에 따라 색상의 강도를 다르게 하여 데이터의 구조를 이해하게 한다

- import matplotlib.pyplot as plt

첫 번째 레이어의 2번째 필터 결과

- plt.matshow(activations[0][0, :, :, 1])
- plt.show()



전체 필터 결과 출력

col

row	0	1	2	3	4	5	6	7

// 연산자는 몫을 반환
필터의 총 개수가 32개이고, 열의 개수가 8개일 때, 각 행(row)은 8개의 필터를 포함한다
예로, i가 0에서 7일 때는 첫 번째 행(0번째 행)에, i가 8에서 15일 때는 두 번째 행(1번째 행)에 위치한다
따라서 row = i // cols는 현재 필터가 몇 번째 행에 위치해야 하는지를 결정한다

% 연산자는 나머지 연산을 수행
각 행(row)에는 열의 개수만큼의 필터가 순서대로 배치된다
예로, i가 0일 때 col은 0, i가 1일 때 col은 1, ... i가 7일 때 col은 7이 된다
i가 8이 되면 다시 첫 번째 열(0번째 열)로 돌아간다
따라서 col = i % cols는 현재 필터가 해당 행에서 몇 번째 열에 위치해야 하는지를 결정한다

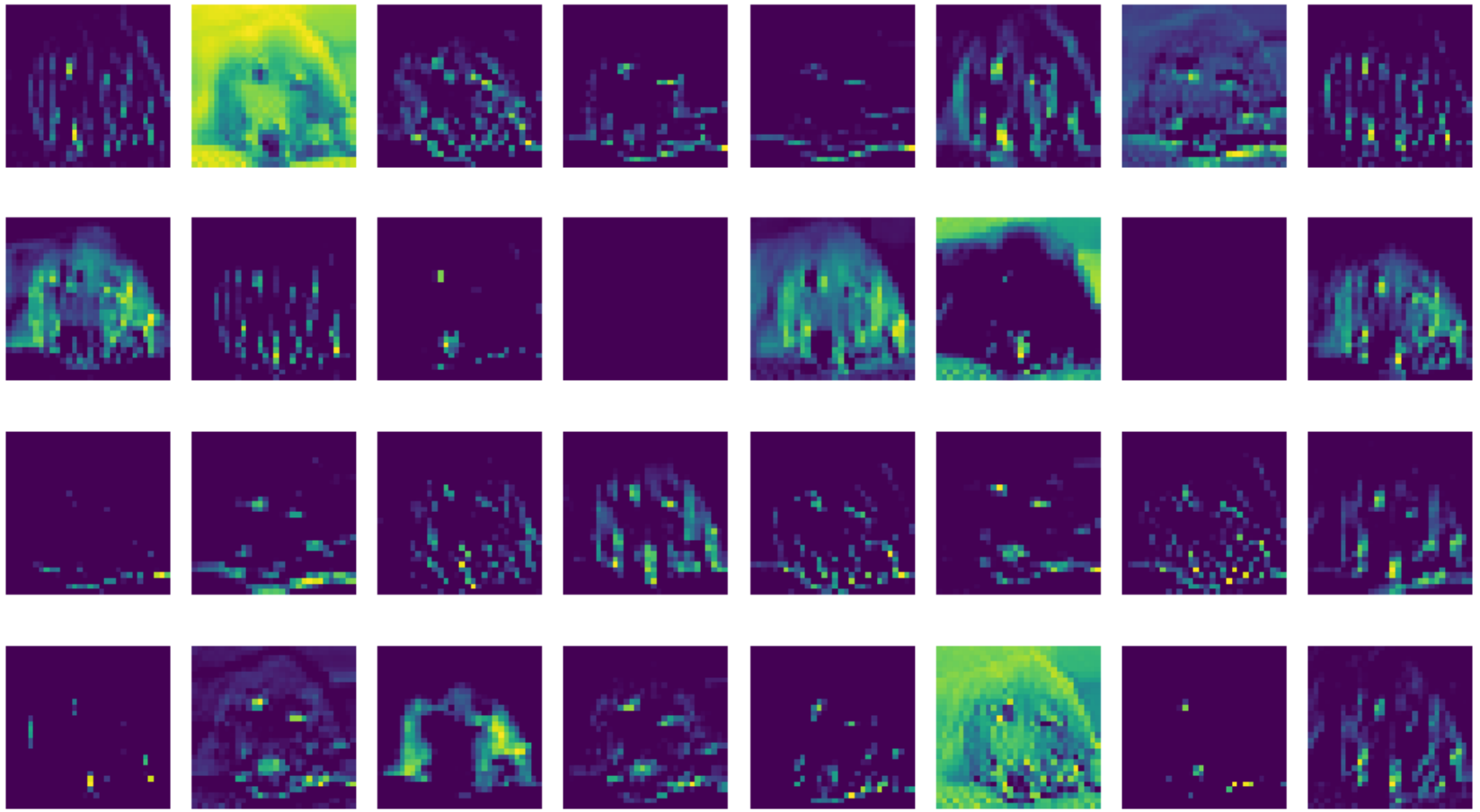
- import matplotlib.pyplot as plt
- def plot_all_filters(activations):

```
    filters_num = 32                          # 필터의 총 개수
    rows = 4                                  # 행의 개수
    cols = 8                                  # 열의 개수

    fig, axes = plt.subplots(rows, cols, figsize=(10, 6))  # 크기 설정

    for i in range(filters_num):
        row = i // cols                       # 행의 인덱스. 현재 필터가 몇 번째 행에 위치해야 하는지를 결정
        col = i % cols                        # 열의 인덱스. 현재 필터가 해당 행에서 몇 번째 위치에 있어야 하는지를 결정
        ax = axes[row, col]
        ax.imshow(activations[0][0, :, :, i]) # 0번(첫번째) 레이어의 각 필터 활성화 결과를 출력
        ax.axis('off')

    plt.tight_layout()
    plt.show()
```

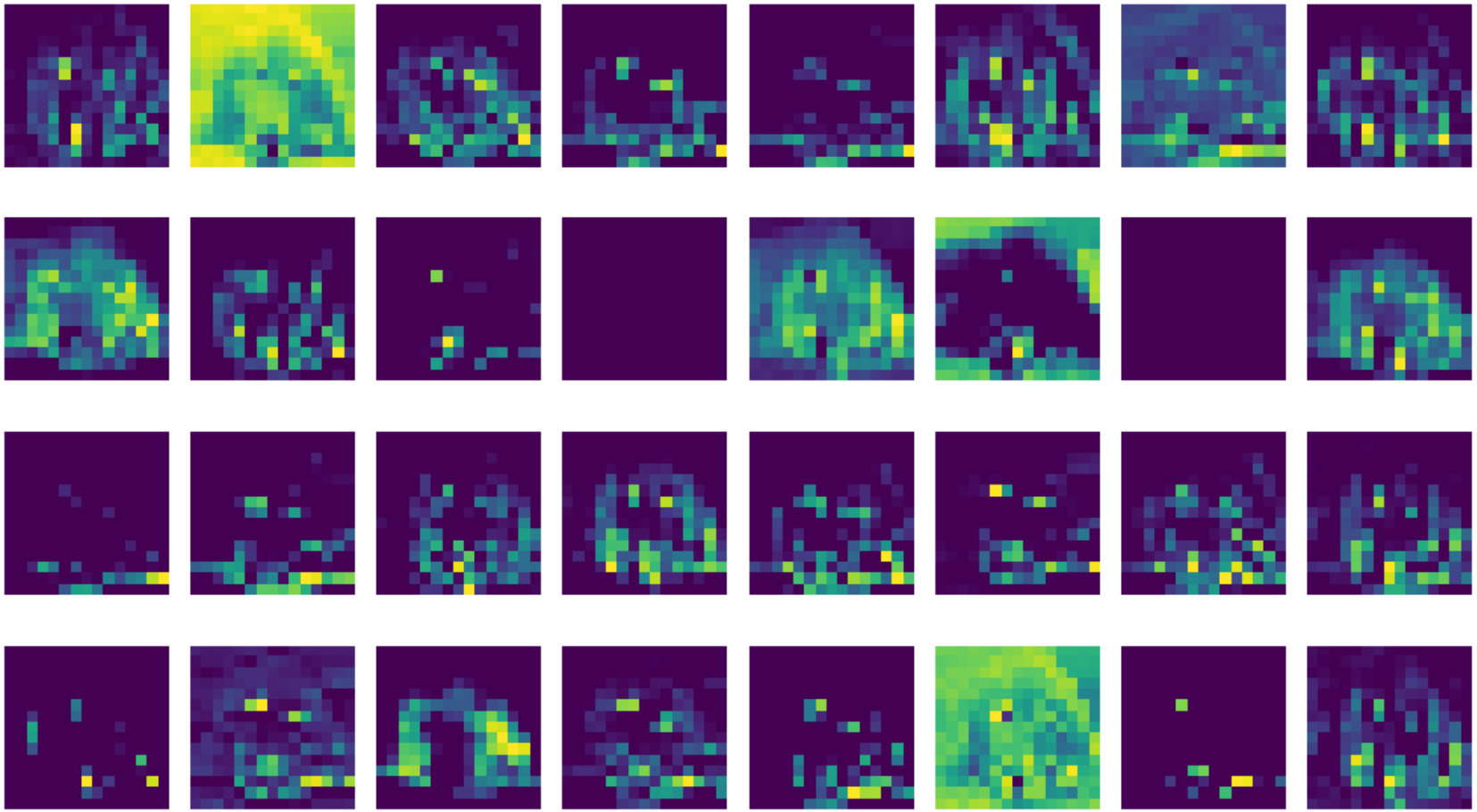
- 32개 전체 필터의 결과를 출력하면 다음과 같다

첫번째 레이어 결과

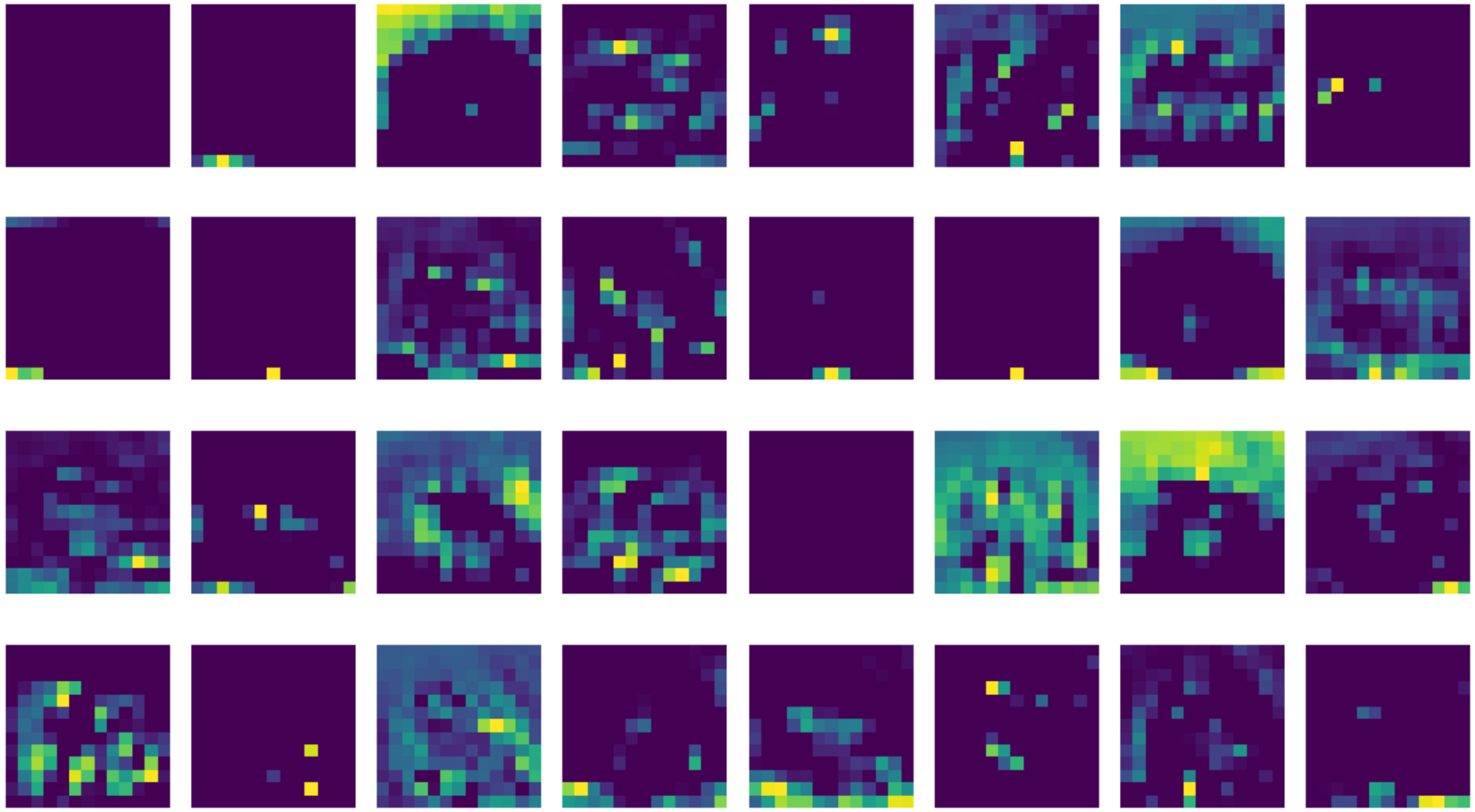
얕은 층에서는 엣지와 같은 저수준 정보가 잘 파악된다
어떤 필터는 외곽선을 더 잘 감지한다



두번째 레이어 결과

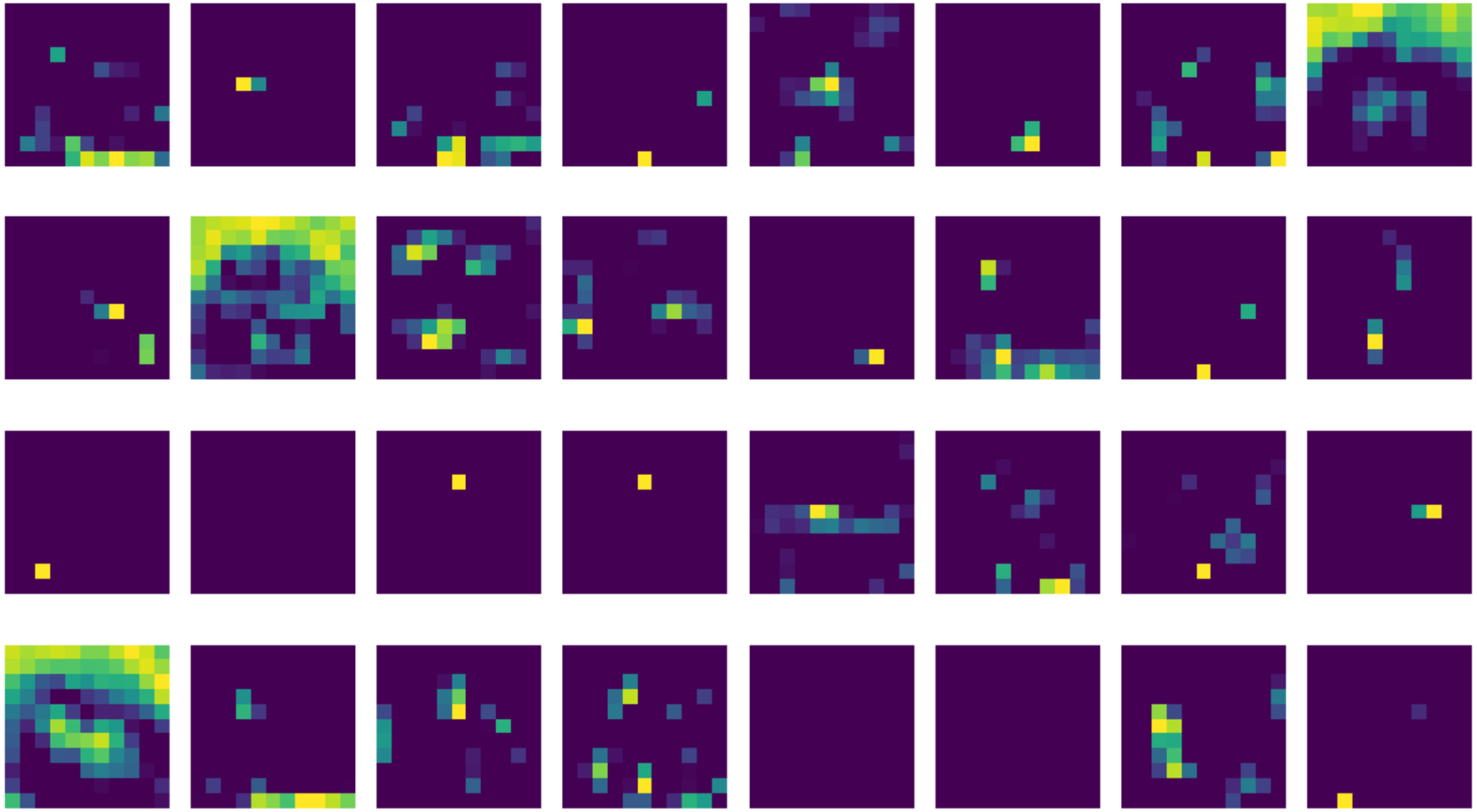


세번째 레이어 결과

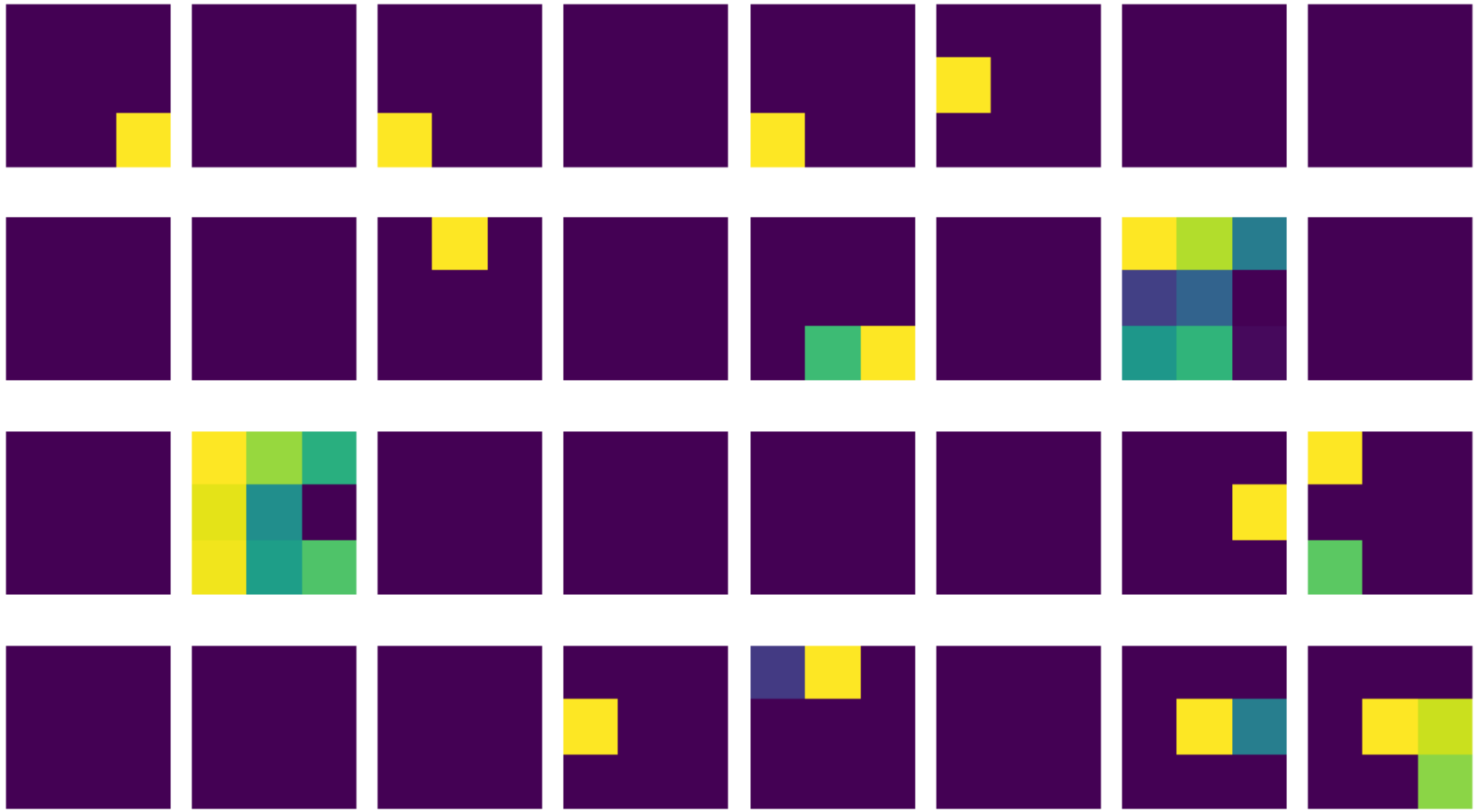


네번째 레이어 결과

층이 깊어질수록 결과는 추상적으로 된다
눈, 귀, 코와 같은 특정 특징들이 어떻게 배열되어 있는지
또는 털 무늬나 체형과 같은 특징들을 파악한다
시각적으로는 이해하기 어려워진다



다섯번째 레이어 결과



여섯번째 레이어 결과

더욱 고수준으로 인코딩되었다
이러한 필터들은 이미지가 고양인지, 개인지를 파악한다
외곽선 뿐만 아니라, 개별 요소의 배열, 주변 환경 등도 학습된다
이러한 특징들은 기계 학습 모델에 의해서만 해석될 수 있는 형태로, 사람의 시각으로 이해하기는 어렵다
활성화되지 못하는 필터도 늘어나는데, 해당 필터에는 파악할 수 있는 패턴이 나타나지 않았다는 것을 의미한다

깊은 층의 활성화 결과

- 층이 깊어질수록 각 레이어는 입력 데이터에 대한 점점 더 추상적인 정보를 학습한다
- 낮은 층의 레이어는 외곽선, 색상, 질감과 같은 입력 이미지의 저수준 특징을 감지한다.
- 이러한 정보는 이미지의 세부적인 부분을 포착한다

- 이후의 레이어는 이러한 특징을 결합하여 고수준의 더 복잡하고 추상적인 특징을 추출한다
- 눈, 귀, 코의 존재와 위치, 배열 형태, 방향, 주변 환경 등의 정보가 종합적으로 인식된다
- 예) 고양이의 세부 종, 감정, 미술 작품의 예술적 의미, 자연 재해 발생 가능 지역

- 타깃에 관한 정보는 깊은 층에서 더 많이 나타난다
- 층이 깊어질수록 입력 이미지와 관련된 타깃에 대한 정보를 더 많이, 잘 학습하게 되며,
- 최종 분류 작업에 필요한 특징을 더 잘 파악하고 활용할 수 있게 된다