케라스 창시자에게 배우는 딥러닝

컴퓨터 비전을 위한 딥러닝

- 합성곱 신경망 소개
- 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련하기
- 사전 훈련된 컨브넷 사용하기
- 컨브넷 학습 시각화
- 요약

한윤진 qkr030@naver.com

합성곱 신경망(Convolutional neural network)

- 컨브넷(Convnet)이라고도 불림
- 거의 대부분의 컴퓨터 비전(**기계의 시각**에 해당하는 부분을 연구하는 컴퓨터 과학의 최신 연구 분야)에 사용됨

완전 연결 계층(Fully connected layer)과의 차이점

완전 연결 계층

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

- 신경망 층인 Dense 층의 연속
- 마지막 Dense 층은 확률 점수가 들어있는 배열을 반환하는 Softmax 층
- 각 점수는 10개의 클래스 중 어떤 클래스에 속할지에 대한 확률
- 이미지 전체의 모든 픽셀에 대한 패턴을 학습(전역패턴 학습)

합성곱 신경망

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

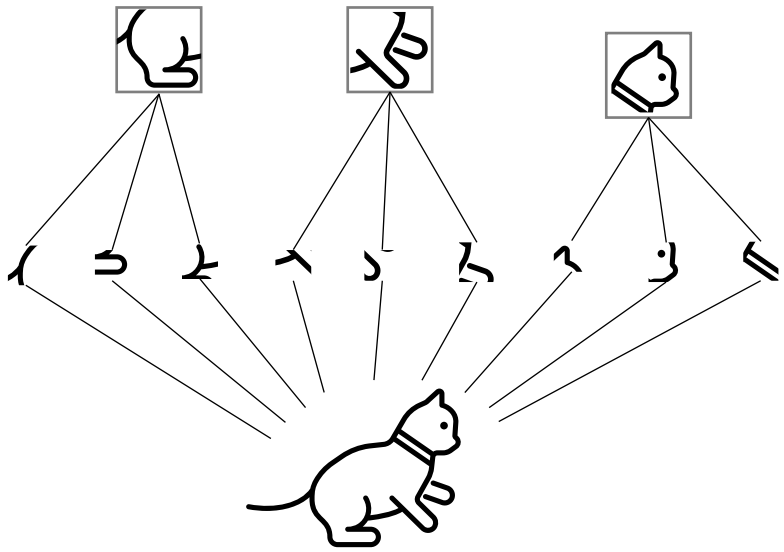
- Dense 층 전에 Conv2D 층과 MaxPooling2D 층 존재
- 작은 2D 윈도우(3X3)로 부분 픽셀에 대한 패턴을 학습(지역패턴 학습)
- Conv2D와 MaxPooling2D 층의 출력은 (height, width, channels) 크기의 3D 텐서
- Dense 층 전에 Flatten 함수를 사용하여 3D 텐서를 1D 텐서로 펼침

합성곱 신경망의 특징

두 가지 핵심 특징

① 학습된 패턴이 새로운 위치에서 인식되어도 이를 새로운 패턴으로 인식하지 않고 효율적으로 처리함

② 학습된 패턴의 크기가 바뀌어도 이를 새로운 패턴으로 인식하지 않고 효율적으로 처리함



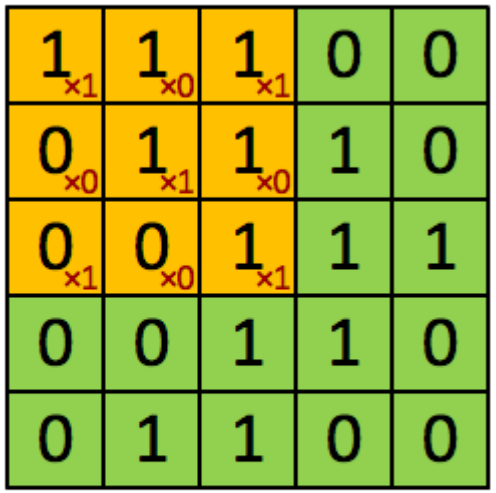
- 합성곱 연산은 특성 맵(Feature Map)이라 부르는 3D 텐서에 적용됨
- 이 텐서는 높이와 넓이, 깊이(채널: 컬러) 축으로 구성됨
- 합성곱 연산은 특성 맵(입력)에서 작은 패치들을 추출하고,
  모든 패치에 같은 변환을 적용하여 출력 특성 맵을 만듦
- 출력 특성 맵도 높이와 넓이를 가진 3D 텐서
- 출력 특성 맵은 입력 특성 맵과 다르게 깊이(채널)가 컬러를 뜻하지 않음

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

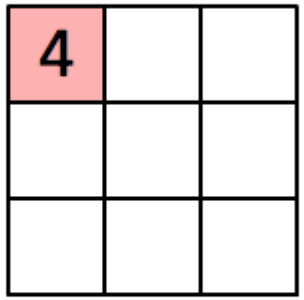
- 레이어를 쌓을 때 설정했던 필터(Filter)의 개수만큼 깊이가 출력됨
- 필터란 입력 데이터의 어떠한 특성을 인코딩하는 것(특성이 64개)

합성곱 연산

합성곱 연산

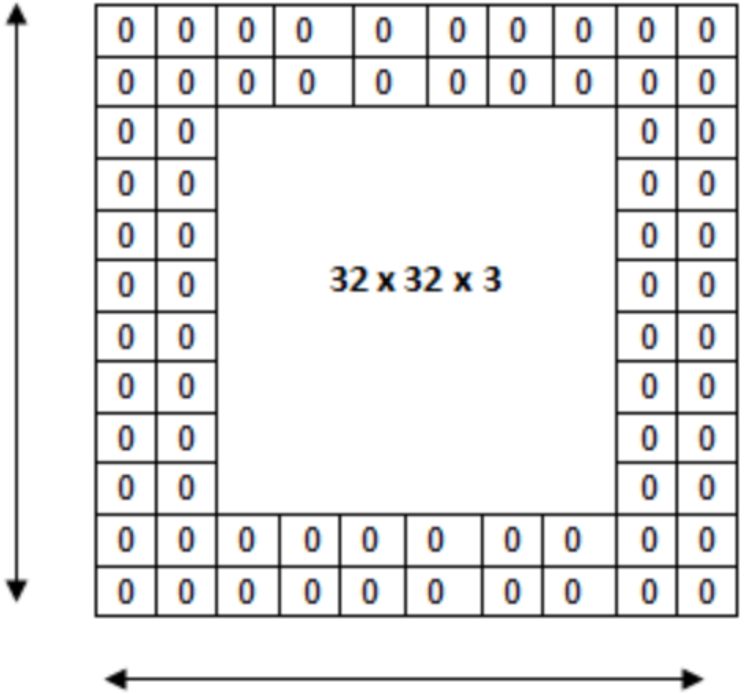


Image



Convolved Feature

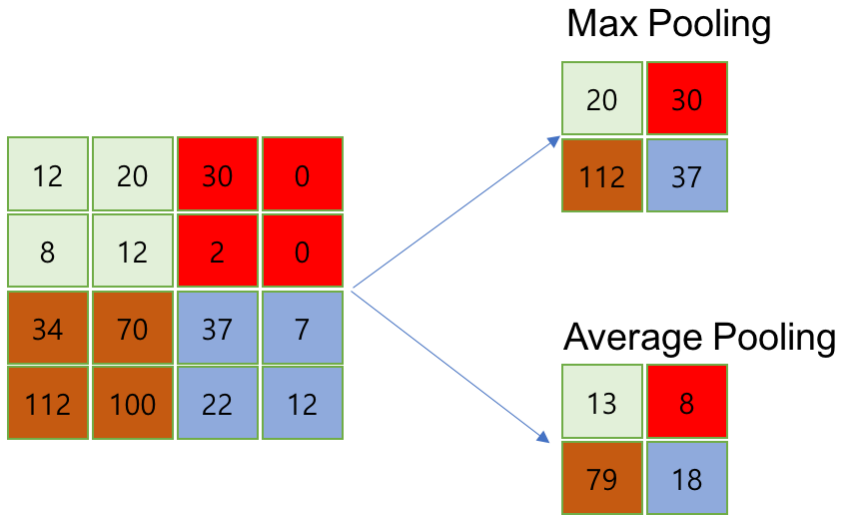
패딩(Padding)



- **입력과 동일한 높이와 넓이를 가진 출력 특성 맵을 얻고 싶을 때**,
- 입력 데이터의 외각에 지정된 픽셀만큼 특정 값으로 채워 넣는 것

최대 풀링 연산

- 특성 맵을 다운샘플링하는 것
  `model.add(layers.MaxPooling2D((2, 2)))`
- 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용
- 레이어 생성시 설정했던 윈도우 크기 **영역 안에 값의 최댓값**을 모으는 것



특성 맵을 다운샘플링 하지 않는다면?

① 특성의 공간적 계층 구조를 학습하는데 도움이 되지 않음
② 최종 특성 맵이 가지는 가중치의 수가 너무 많음
③ 심한 과대적합이 발생함

소규모 데이터셋으로 CNN 학습

- 학습 데이터 2,000개, 검증 데이터 1,000개, 테스트 데이터 1,000개를 사용
- 과대적합 학습이 된 모델을 데이터 증식(Data augmentation)을 사용하여 해결

강아지 vs 고양이

1. Kaggle 데이터(25,000개) 사용
2. 그 중 4,000개의 소규모 데이터만 사용
3. CNN 네트워크 구성

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

4. 데이터 전처리

- 픽셀 값[0-255]을 [0-1] 사이로 조정
- 신경망은 작은 입력 값을 선호함

```
from keras.preprocessing.image import ImageDataGenerator

# 모든 이미지를 1/255로 스케일을 조정
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir, # 타깃 디렉터리
        target_size=(150, 150), # 모든 이미지를 150 × 150 크기
        batch_size=20,
        class_mode='binary') # binary_crossentropy 손실을 사용하기 때문에 이진 레이블이 필요
```

5. 배치 설정

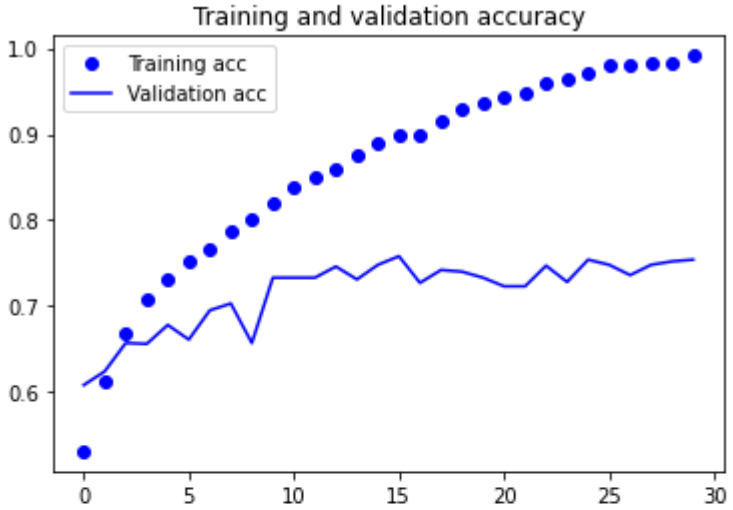
```
for data_batch, labels_batch in train_generator:
    print('배치 데이터 크기:', data_batch.shape)
    print('배치 레이블 크기:', labels_batch.shape)
    break
```

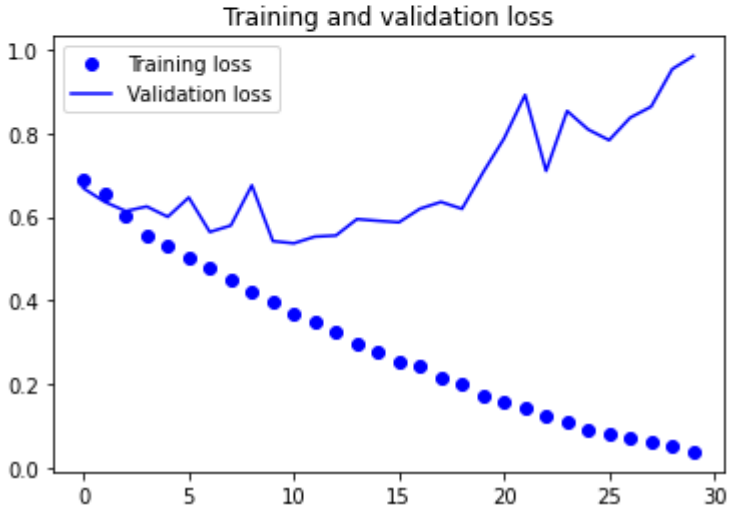
```
배치 데이터 크기: (20, 150, 150, 3)
배치 레이블 크기: (20,)
```

6. 모델 학습

소규모 데이터셋으로 CNN 학습

loss: 0.0334 acc: 0.9919 / val_loss: 0.9854 val_acc: 0.7540





데이터 증식하기

- 학습할 데이터가 너무 적어 새로운 데이터에 일반화할 수 있는 모델을 훈련시킬 수 없을 때 사용
- **기존의 학습 데이터로부터 더 많은 학습 데이터를 생성하는 방법**
- 여러 가지 랜덤한 변환을 적용하여 샘플을 늘림
- 학습할 때 모델이 같은 데이터를 두 번 만나지 않도록 하는 것이 목표

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

- Rotation_range : 랜덤하게 사진을 회전시킬 각도 범위(0-180)
- Width_shift_range/Height_shift_range : 사진을 수평과 수직으로 랜덤하게 평행 이동 시킬 범위
- Shear_range : 랜덤하게 전단 변환할 각도 범위(사진을 회전할 때 y축 방향으로 각도 증가)
- Zoom_range : 랜덤하게 사진을 확대할 범위
- Horizontal_flip : 랜덤하게 이미지를 수평으로 뒤집음
- Fill_mode : 회전이나 가로/세로 이동으로 인해 새롭게 생성할 픽셀을 채우는 전략

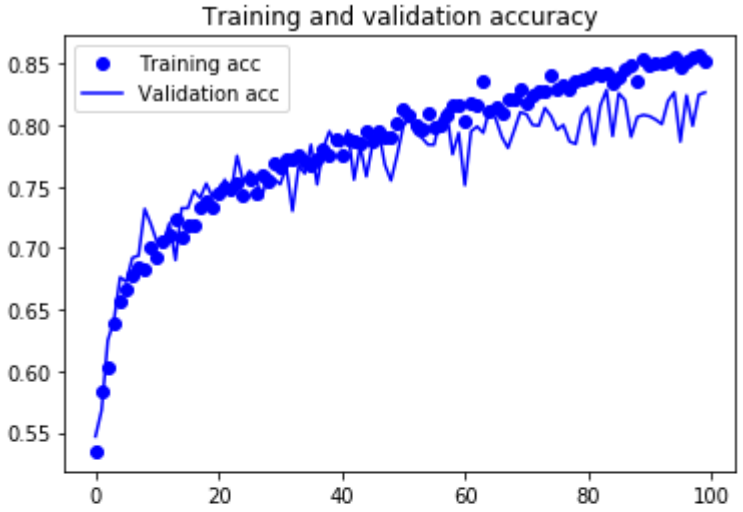
Dropout 추가

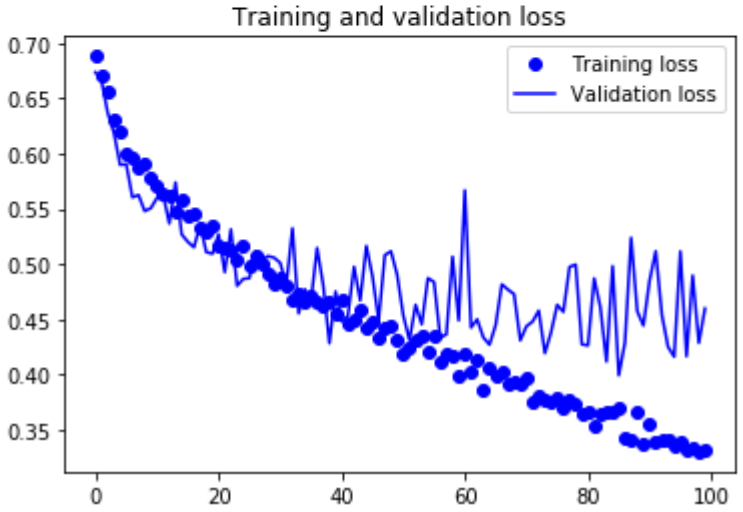
- 데이터가 적기 때문에 데이터를 증식 시켰다 하더라도 데이터들 사이에 상호 연관성이 큼
- 과대적합을 제거하기에 완전히 충분하지 않음
- 드롭아웃 층을 추가하여 과대적합을 더욱 억제하도록 함

```
model.add(layers.Dropout(0.5))
```

데이터 증식하기

loss: 0.3332 acc: 0.8525 / val_loss: 0.4596 val_acc: 0.8267





사전 훈련된 CNN 사용하기

특성 추출

- 사전에 학습된 네트워크의 표현을 사용하여 데이터에서 특성을 뽑아내는 것
- 뽑아진 특성을 사용하여 새로운 분류 모델(완전 연결 계층)을 훈련함

ImageNet 데이터셋에서 훈련된 대규모 CNN 사용

- 1.4백만 개의 이미지와 1,000개의 클래스
- 다양한 종의 강아지와 고양이를 포함하여 많은 동물들을 포함

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

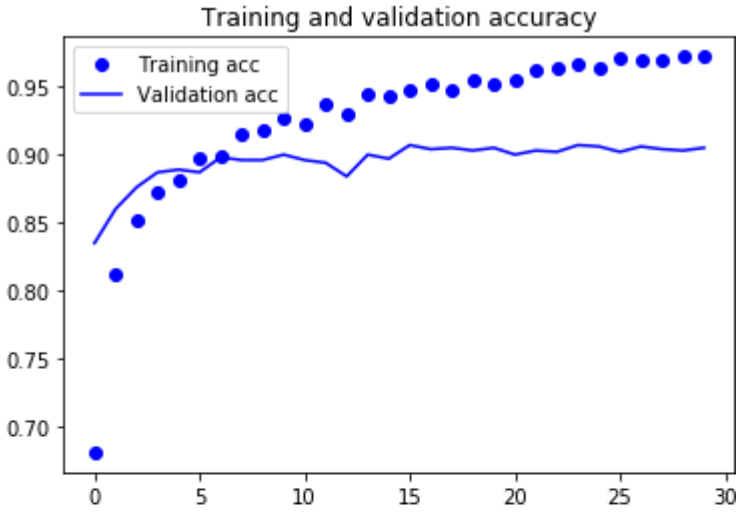
- Weights : 모델을 초기화할 가중치 체크포인트를 지정
- Include_top : 네트워크의 최상위 완전 연결 분류기를 포함할지 안 할지를 지정
- Input_shape : 네트워크에 주입할 이미지 텐서의 크기

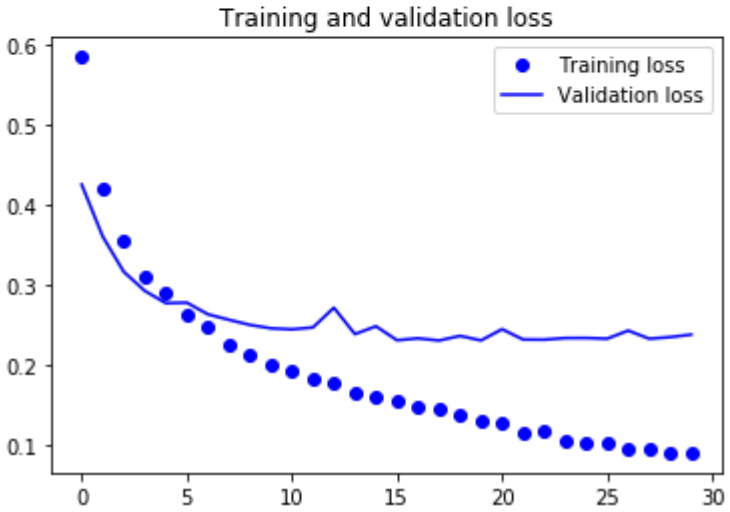
- 4*4*512 크기의 특성을 추출할 수 있음
- 이후 추가된 완전 연결 계층으로 특성과 라벨을 입력시켜 훈련함
- 2개의 Dense 층만 처리하면 되기 때문에 학습이 매우 빠름

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

사전 훈련된 CNN 사용하기

loss: 0.0905 acc: 0.9720 / val_loss: 0.2376 val_acc: 0.9050





CNN 학습 시각화

- 딥러닝 모델을 블랙박스 같다고 함
- 학습된 표현에서 사람이 이해하기 쉽게 뽑아 내기 어렵기 때문
- 그러나 CNN의 표현은 시각적인 개념을 학습한 것이기 때문에 시각화하기 좋음

시각화 세 가지 기법

① CNN 중간 층의 출력 시각화 : 연속된 컨볼루션 층이 입력을 어떻게 변형시키는지 이해하고, 필터의 의미를 파악하는데 도움이 됨

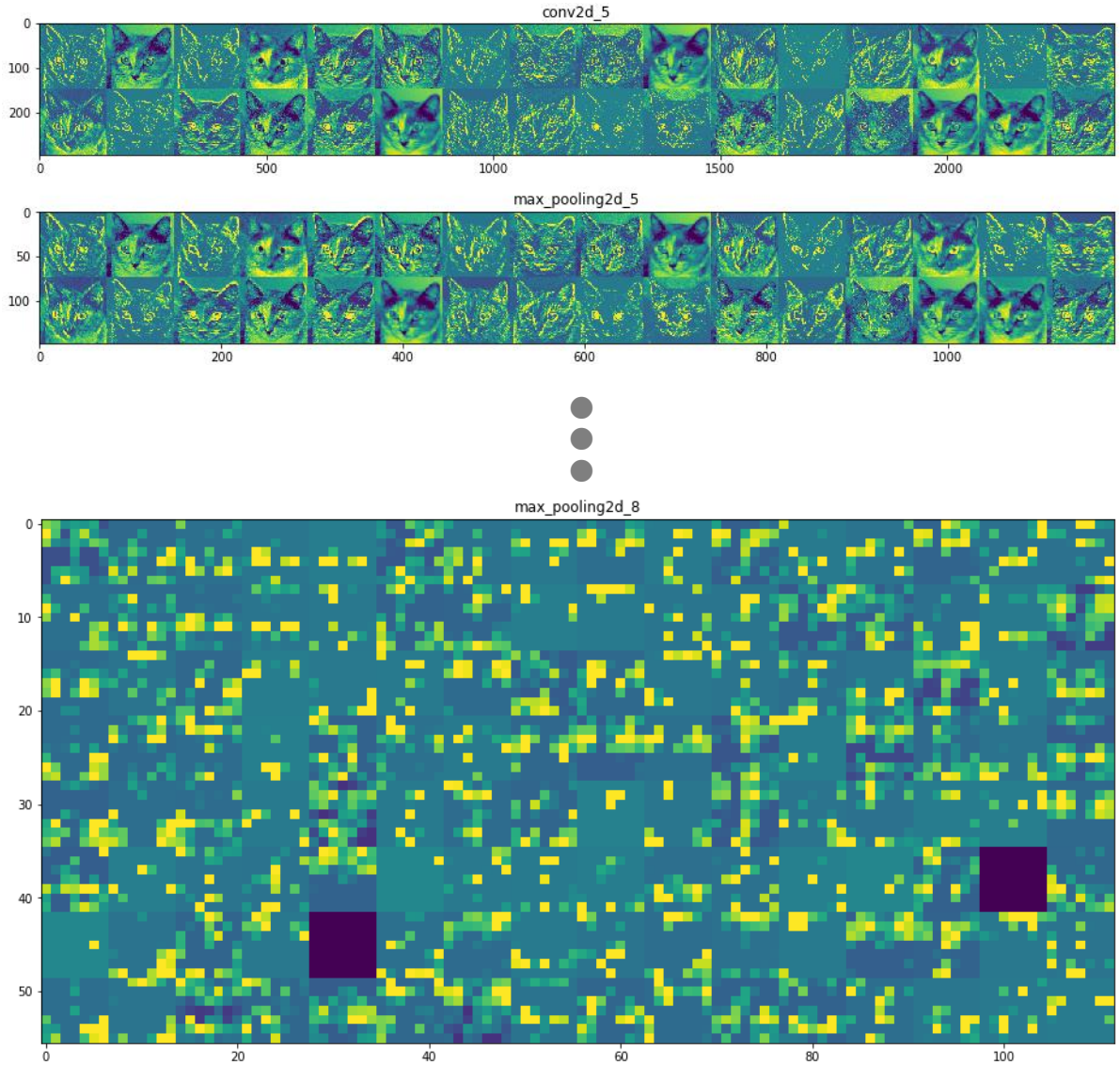
② CNN 필터 시각화 : 필터가 찾으려는 시각적인 패턴과 개념이 무엇인지 이해하는데 도움이 됨

③ 클래스 활성화에 대한 히트맵을 이미지에 시각화 : 이미지의 어느 부분이 주어진 클래스에 속하는데 기여했는지 이해하는데 도움이 됨

※ 히트맵(Heatmap) : 데이터를 컬러 색상의 강도로 표현하는 그래프

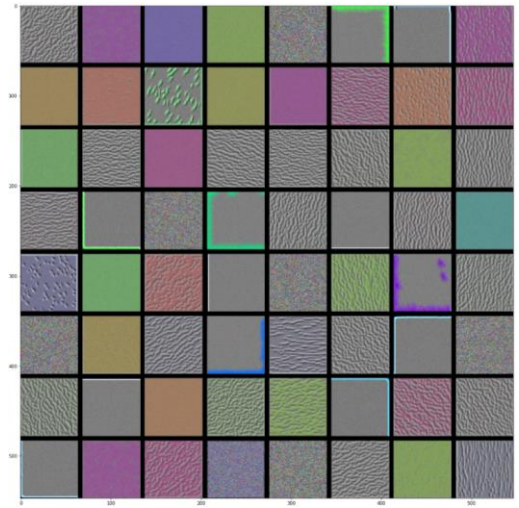
CNN 중간 층의 출력 시각화

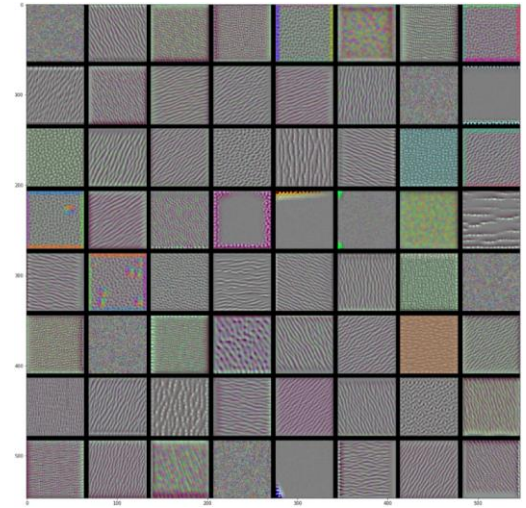
```python
# 층의 이름을 그래프 제목으로 사용합니다
layer_names = []
for layer in model.layers[:8]:
    layer_names.append(layer.name)

images_per_row = 16

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

    # 특성 맵의 크기는 (1, size, size, n_features)입니다
    size = layer_activation.shape[1]

    # 활성화 채널을 위한 그리드 크기를 구합니다
    n_cols = n_features // images_per_row
    display_grid = np.zeros((size * n_cols, images_per_row * size))

    # 각 활성화를 하나의 큰 그리드에 채웁니다
    for col in range(n_cols):
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                             :, :,
                                             col * images_per_row + row]
            # 그래프로 나타내기 좋게 특성을 처리합니다
            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[col * size : (col + 1) * size,
                         row * size : (row + 1) * size] = channel_image

    # 그리드를 출력합니다
    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()
```

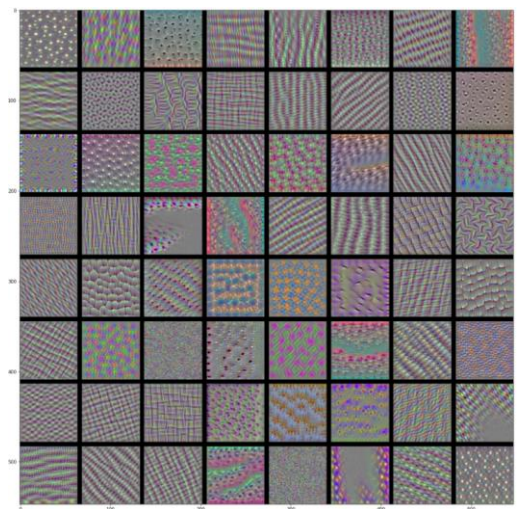


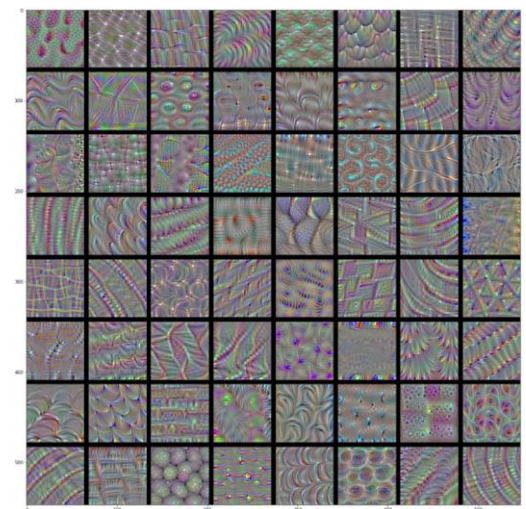
CNN 필터 시각화

```python
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()
```





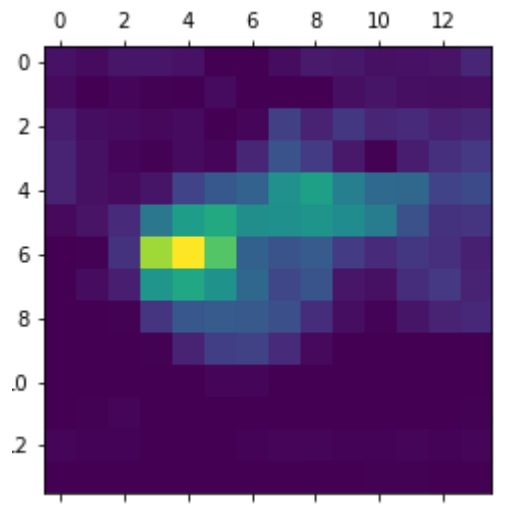




클래스 활성화에 대한 히트맵을 이미지에 시각화

```
# 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))

# 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()
```



클래스 활성화에 대한 히트맵을 이미지에 시각화