Convolutions

IMAGE PROCESSING WITH KERAS IN PYTHON

Ariel Rokem

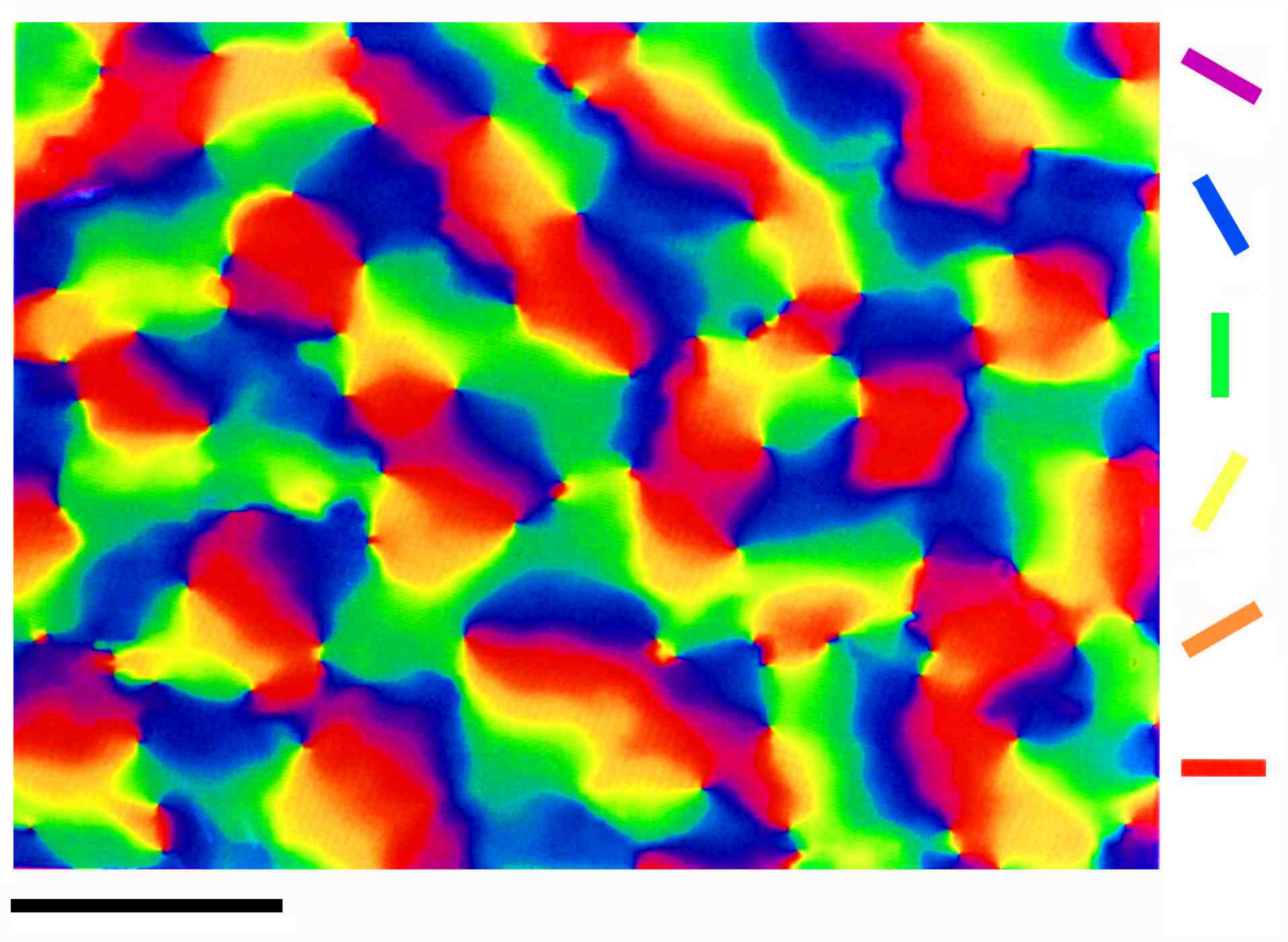
Senior Data Scientist, University of Washington

Using correlations in images

- Natural images contain spatial correlations
- For example, pixels along a contour or edge
- How can we use these correlations?

이미지에 있는 픽셀은 서로 인접한 픽셀들은 유사한 패턴을 갖는 경향이 있습니다.

Biological inspiration



이미지의 모든 위치에서 특정 방향과 같은 동일한 기능을 찾는 것은 컨볼루션이라고 불리는 수학적 연산과 같습니다.(?)

What is a convolution?

```
array = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
kernel = np.array([-1, 1])
conv = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0])
conv[0] = (kernel * array[0:2]).sum()
conv[1] = (kernel * array[1:3]).sum()
conv[2] = (kernel * array[2:4]).sum()
...
for ii in range(8):
    conv[ii] = (kernel * array[ii:ii+2]).sum()
conv
```

```
array([0, 0, 0, 0, 1, 0, 0, 0, 0])
```

1차원에서의 컨볼루션을 5개의 0과 5개의 1을 포함하는 배열을 만들고 다음 커널을 array에 따라서 이동하여 배열의 값과 커널이 값을 곱하고 그 값을 요약한 결과가 해당 위치에 대한 결과입니다.

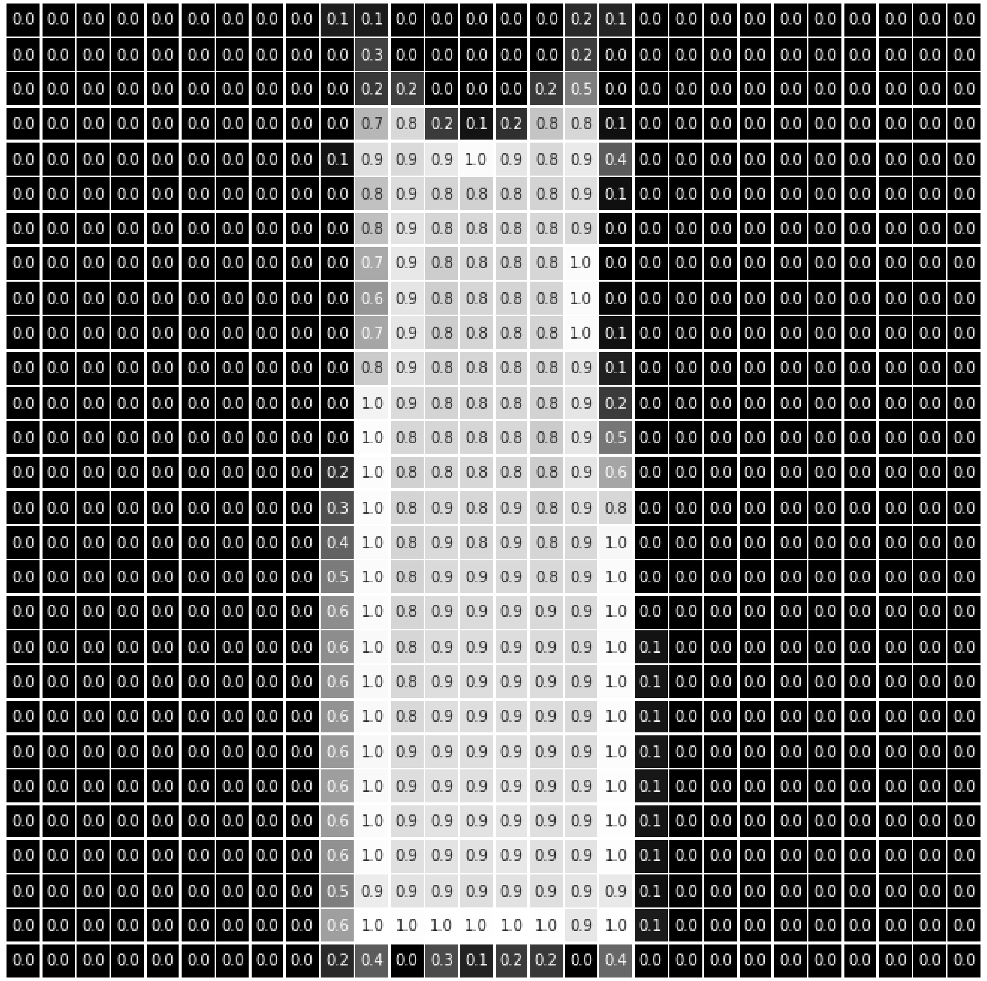
Convolution in one dimension

```
array = np.array([0, 0, 1, 1, 0, 0, 1, 1, 0, 0])
kernel = np.array([-1, 1])
conv = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0])
for ii in range(8):
    conv[ii] = (kernel * array[ii:ii+2]).sum()
conv
```

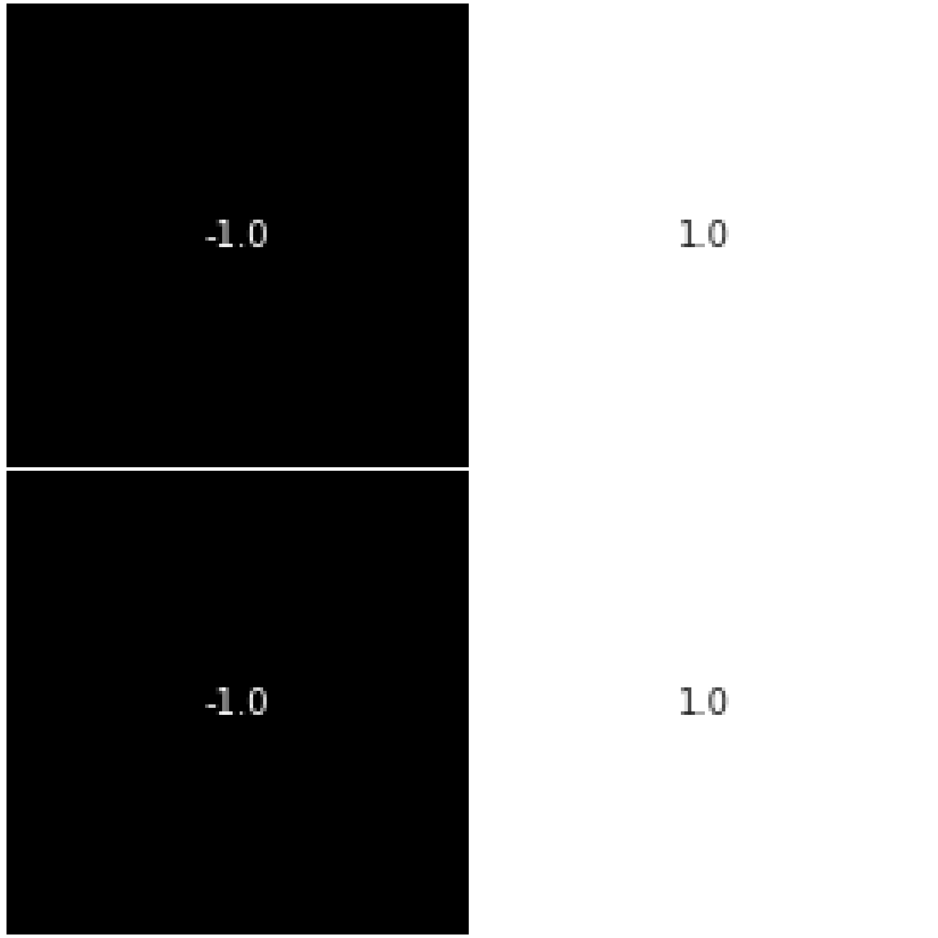
```
array([ 0,  1,  0, -1,  0,  1,  0, -1,  0])
```

이 예제에서는 배열이 0과 1 사이를 두번 이동합니다.
이때 1에서 0으로 가는 edge는 커널의 반대이고 이러한 위치에서 컨볼루션은 음수가 됩니다.

Image convolution

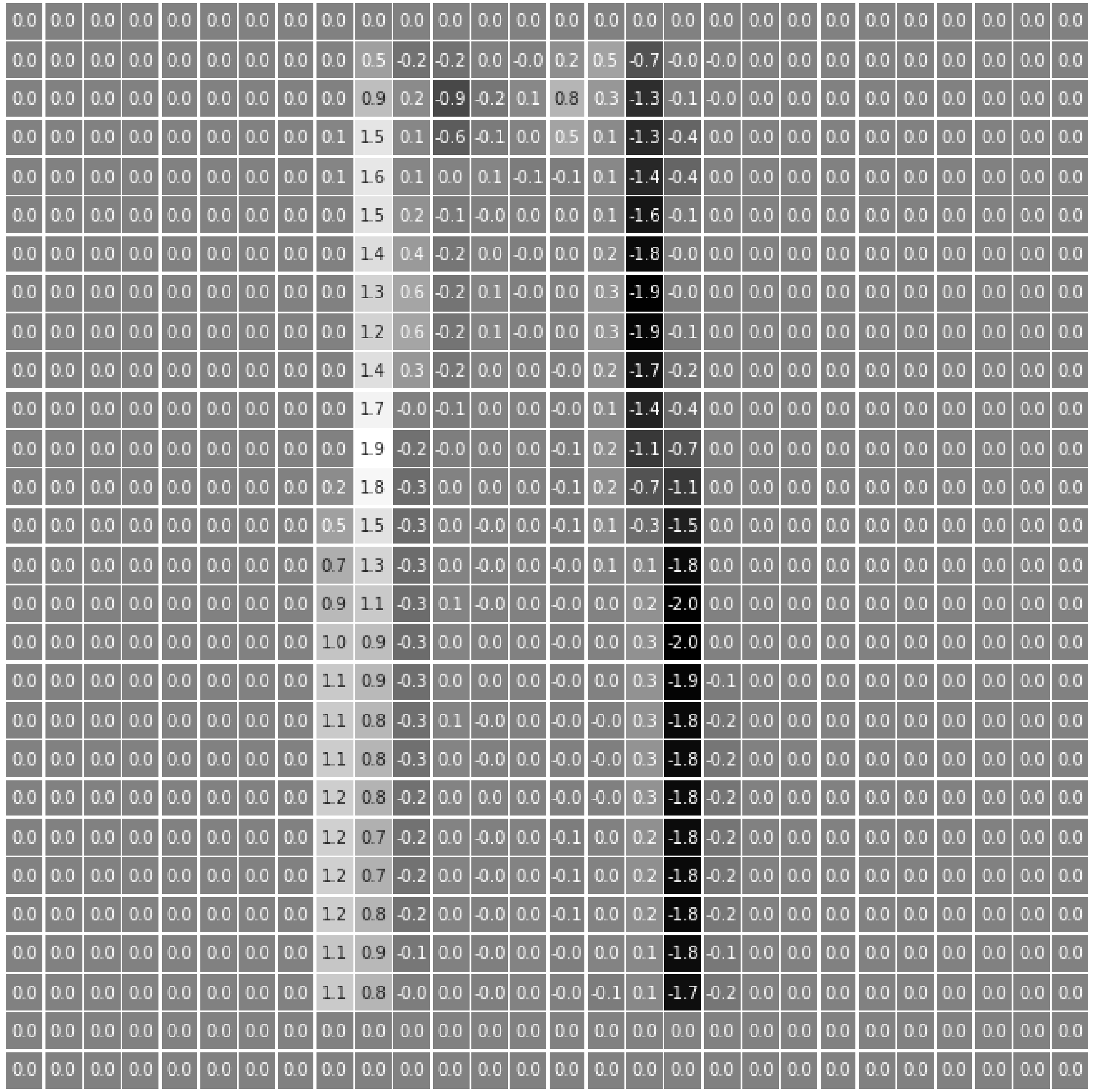


0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.0	0.0	0.0	0.2	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.8	0.2	0.1	0.2	0.8	0.8	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.9	0.9	0.9	1.0	0.9	0.8	0.9	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.9	0.8	0.8	0.8	0.8	0.9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.9	0.8	0.8	0.8	0.8	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.9	0.8	0.8	0.8	0.8	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.9	0.8	0.8	0.8	0.8	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.9	0.8	0.8	0.8	0.8	1.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.9	0.8	0.8	0.8	0.8	0.9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.9	0.8	0.8	0.8	0.8	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.8	0.8	0.8	0.8	0.8	0.9	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	1.0	0.8	0.8	0.8	0.8	0.8	0.9	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	0.8	0.9	0.8	0.9	0.8	0.9	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	1.0	0.8	0.9	0.8	0.9	0.8	0.9	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.8	0.9	0.9	0.9	0.8	0.9	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.8	0.9	0.9	0.9	0.9	0.9	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.8	0.9	0.9	0.9	0.9	0.9	1.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.8	0.9	0.9	0.9	0.9	0.9	1.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.8	0.9	0.9	0.9	0.9	0.9	1.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.9	0.9	0.9	0.9	0.9	0.9	1.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.9	0.9	0.9	0.9	0.9	0.9	1.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.9	0.9	0.9	0.9	0.9	0.9	1.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.9	0.9	0.9	0.9	0.9	0.9	1.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.4	0.0	0.3	0.1	0.2	0.2	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0



-1.0	1.0
-1.0	1.0

Image convolution



0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	-0.2	-0.2	0.0	-0.0	0.2	0.5	-0.7	-0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.2	-0.9	-0.2	0.1	0.8	0.3	-1.3	-0.1	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	1.5	0.1	-0.6	-0.1	0.0	0.5	0.1	-1.3	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	1.6	0.1	0.0	0.1	-0.1	-0.1	0.1	-1.4	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.5	0.2	-0.1	-0.0	0.0	0.0	0.1	-1.6	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.4	0.4	-0.2	0.0	-0.0	0.0	0.2	-1.8	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.3	0.6	-0.2	0.1	-0.0	0.0	0.3	-1.9	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.2	0.6	-0.2	0.1	-0.0	0.0	0.3	-1.9	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.4	0.3	-0.2	0.0	0.0	-0.0	0.2	-1.7	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.7	-0.0	-0.1	0.0	0.0	-0.0	0.1	-1.4	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.9	-0.2	-0.0	0.0	0.0	-0.1	0.2	-1.1	-0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	1.8	-0.3	0.0	0.0	0.0	-0.1	0.2	-0.7	-1.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.5	-0.3	0.0	-0.0	0.0	-0.1	0.1	-0.3	-1.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	1.3	-0.3	0.0	-0.0	0.0	-0.0	0.1	0.1	-1.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	1.1	-0.3	0.1	-0.0	0.0	-0.0	0.0	0.2	-2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.9	-0.3	0.0	0.0	0.0	-0.0	0.0	0.3	-2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.1	0.9	-0.3	0.0	0.0	0.0	-0.0	0.0	0.3	-1.9	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.1	0.8	-0.3	0.1	-0.0	0.0	-0.0	-0.0	0.3	-1.8	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.1	0.8	-0.3	0.0	-0.0	0.0	-0.0	-0.0	0.3	-1.8	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.2	0.8	-0.2	0.0	0.0	0.0	-0.0	-0.0	0.3	-1.8	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.2	0.7	-0.2	0.0	-0.0	0.0	-0.1	0.0	0.2	-1.8	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.2	0.7	-0.2	0.0	-0.0	0.0	-0.1	0.0	0.2	-1.8	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.2	0.8	-0.2	0.0	-0.0	0.0	-0.1	0.0	0.2	-1.8	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.1	0.9	-0.1	0.0	-0.0	0.0	-0.0	0.0	0.1	-1.8	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.1	0.8	-0.0	0.0	-0.0	0.0	-0.0	-0.1	0.1	-1.7	-0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

커널로 합성을 할 때 왼쪽 엣지가 강조된다는 것을 의미하고,
드레스의 오른쪽 콘볼루션은 음수이다.

Two-dimensional convolution

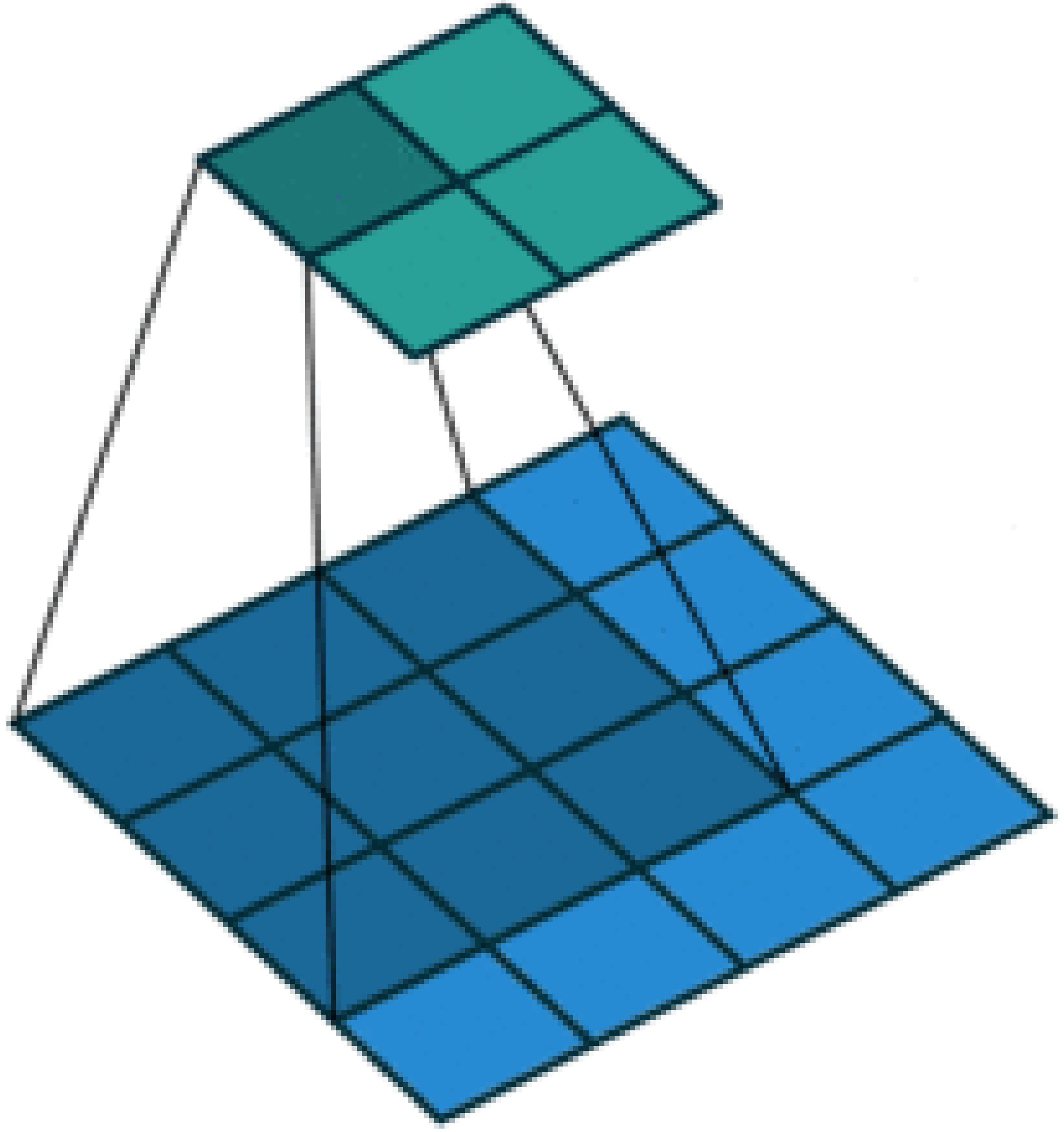
```
kernel = np.array([[-1, 1],
                   [-1, 1]])


conv = np.zeros((27, 27)
for ii in range(27):
    for jj in range(27):
        window = image[ii:ii+2, jj:jj+2]
        conv[ii, jj] = np.sum(window * kernel)
```

이제 코드에서 컨볼루션의 구현에 대해 알아보겠습니다. 먼저 커널을 만듭니다. 그런 다음 컨볼루션 결과를 저장할 배열을 만듭니다. 이미지의 모든 위치에서 반복합니다. 각 위치에서 커널 크기의 창을 선택하고 해당 창에 커널을 곱한 다음 요약합니다. 그런 다음 이 합계가 해당 위치에서 컨볼루션된 이미지의 값으로 입력됩니다. 루프의 끝에서 배열은 컨볼루션 결과를 포함합니다.

Convolution

컨볼루션 작동을 보여주는 그림이고 커널은 3X3상자입니다.



다음은 컨볼루션 작동을 보여주는 그림입니다. 커널은 하단의 파란색 입력 이미지 위로 미끄러지는 회색 3x3 상자입니다. 각 위치에서 창은 커널의 값과 곱하고 더해져 위쪽의 결과 녹색 배열에 있는 픽셀 중 하나의 값을 생성합니다. 신경망에서, 우리는 이 결과 배열을 "피처 맵"이라고 부른다. 왜냐하면 이 배열은 이 커널에 의해 표현되는 피처와 일치하는 이미지의 위치 맵을 포함하고 있기 때문이다.

Let's practice!

IMAGE PROCESSING WITH KERAS IN PYTHON

Implementing convolutions in Keras

IMAGE PROCESSING WITH KERAS IN PYTHON

Ariel Rokem

Senior Data Scientist, University of Washington

Keras Convolution layer

```python
from keras.layers import Conv2D
Conv2D(10, kernel_size=3, activation='relu')
```

10개의 컨볼루션 유닛이 있고 kernel size=3, 활성화 함수는 relu로 설정했습니다.
이때 kernel size가 3이므로 각 장치의 커널이 9 픽셀이고 layer에 10개의 유닛이 있으므로
kernel 에 대해 90개의 매개변수가 있습니다.

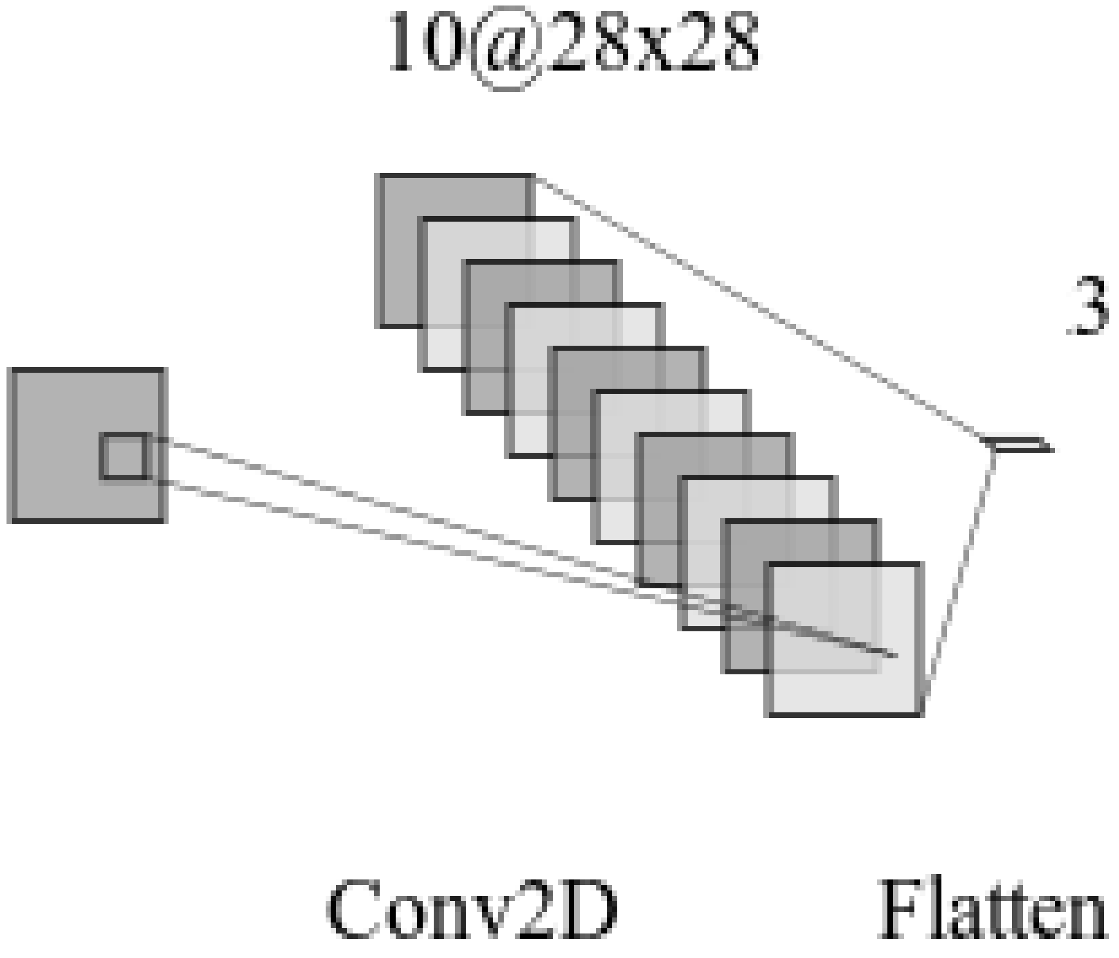
Integrating convolution layers into a network

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
model = Sequential()
model.add(Conv2D(10, kernel_size=3, activation='relu',
          input_shape=(img_rows, img_cols, 1)))
model.add(Flatten())
model.add(Dense(3, activation='softmax'))
```

레이어를 다음 레이어에 연결하기 위해 Flatten 레이어를 추가합니다.
(이것은 우리가 이전에 "피쳐맵"이라고 불렀던 컨볼루션의 출력을 가져와 1차원 배열로 병합합니다.)

Our CNN

입력은 28 x 28 픽셀의 이미지입니다. Conv2D 작업은 각각 28x28 픽셀의 10개 기능 맵으로 이동하고 Flatten 작업은 출력의 3개 단위로 이동합니다.



Fitting a CNN

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
train_data.shape
```

28X28 픽셀 이미지이고 50개의 흑백 샘플입니다.

```
(50, 28, 28, 1)
```

```
model.fit(train_data, train_labels, validation_split=0.2,
          epochs=3)


model.evaluate(test_data, test_labels, epochs=3)
```

Let's practice!

IMAGE PROCESSING WITH KERAS IN PYTHON

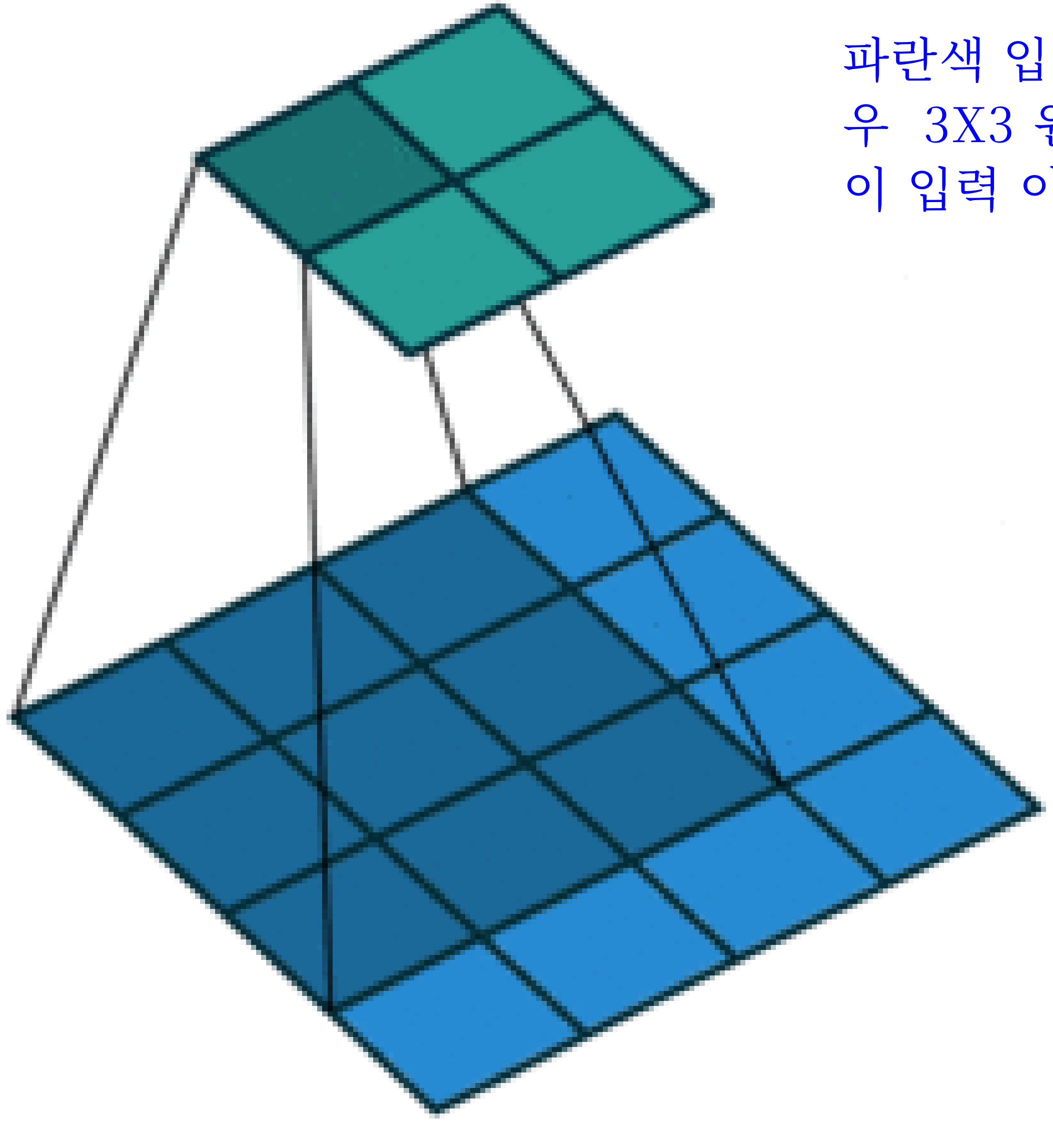
Tweaking your convolutions

IMAGE PROCESSING WITH KERAS IN PYTHON

Ariel Rokem

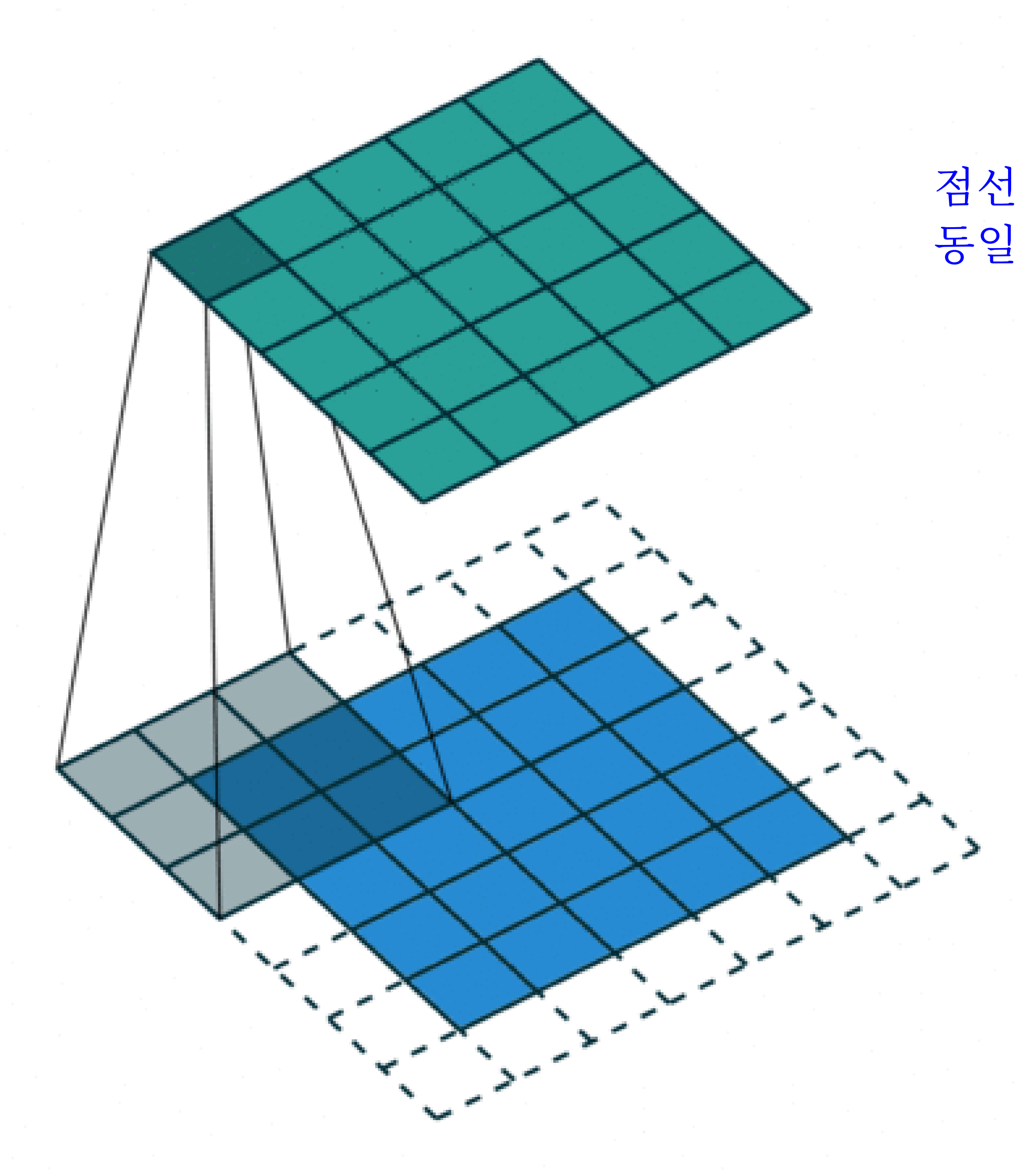
Senior Data Scientist, University of Washington

Convolution



파란색 입력 이미지가 녹색 출력 이미지보다 큰데 이때 kernel size가 3X3인 경우 3X3 윈도우를 출력 이미지의 한 픽셀로 변환하는데 이 문제를 처리하는 것이 입력 이미지를 0으로 채우는 것입니다.

Convolution with zero padding



점선을 포함한 입력 이미지가 0으로 채워지면 출력 feature map은 입력과 동일한 크기를 갖습니다.

Zero padding in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
                 padding='valid')
```

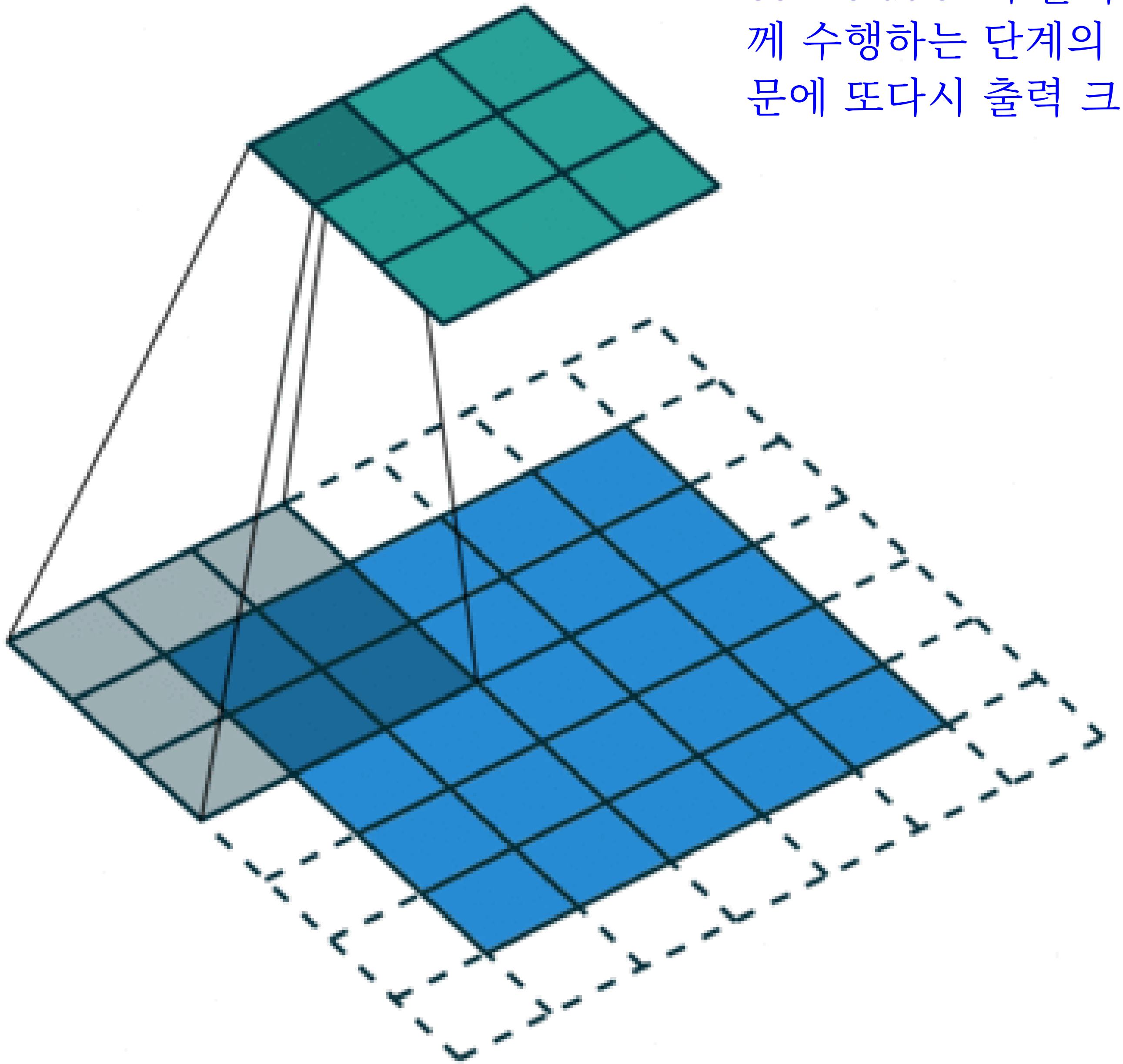
padding의 defual값은 valid이며 padding에 'valid'를 주면 제로패딩이 추가되지 않습니다.

Zero padding in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
                 padding='same')
```

반면에 "same" 값을 제공하면 이 레이어에 대한 입력에 0 패딩이 적용되어
컨볼루션의 출력이 컨볼루션의 입력과 동일한 크기를 갖게 됩니다.

Strides



convolution의 출력 크기에 영향을 미치는 또 다른 요소는 입력 픽셀 사이에서 커널과 함께 수행하는 단계의 크기인데 예를 들어 strides를 2로 지정했다면 2픽셀씩 이동하기 때문에 또다시 출력 크기가 입력크기보다 작아집니다.

Strides in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
          strides=1)
```

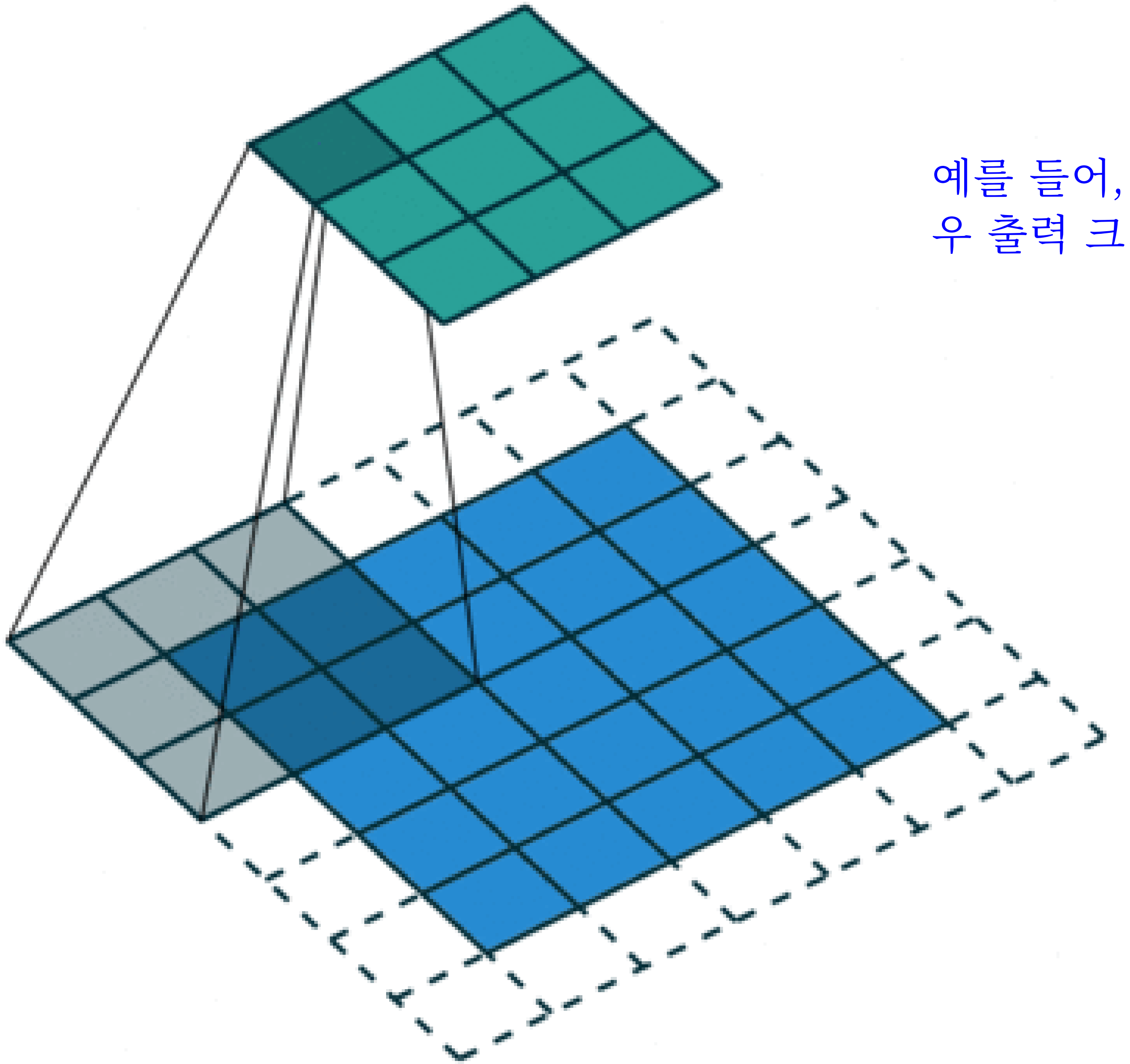
stride의 기본값은 1로 설정합니다.

Strides in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
          strides=2)
```

strides를 2로 설정하고 출력크기가 입력크기보다 작아집니다.

Example



예를 들어, 입력 이미지가 5 x 5이고 패딩이 0이고 보폭이 2로 설정된 경우 출력 크기는 3 x 3입니다.

Calculating the size of the output

$$O = ((I - K + 2P)/S) + 1$$

where

일반적으로 다음과 같은 간단한 공식을 사용하여 출력 크기를 계산할 수 있습니다. (I - K + 2P) / (S + 1) 여기서 I은 입력 크기, K는 커널 크기, P는 크기 제로 패딩, S는 보폭입니다.

- I = size of the input
- K = size of the kernel
- P = size of the zero padding
- S = strides

Calculating the size of the output

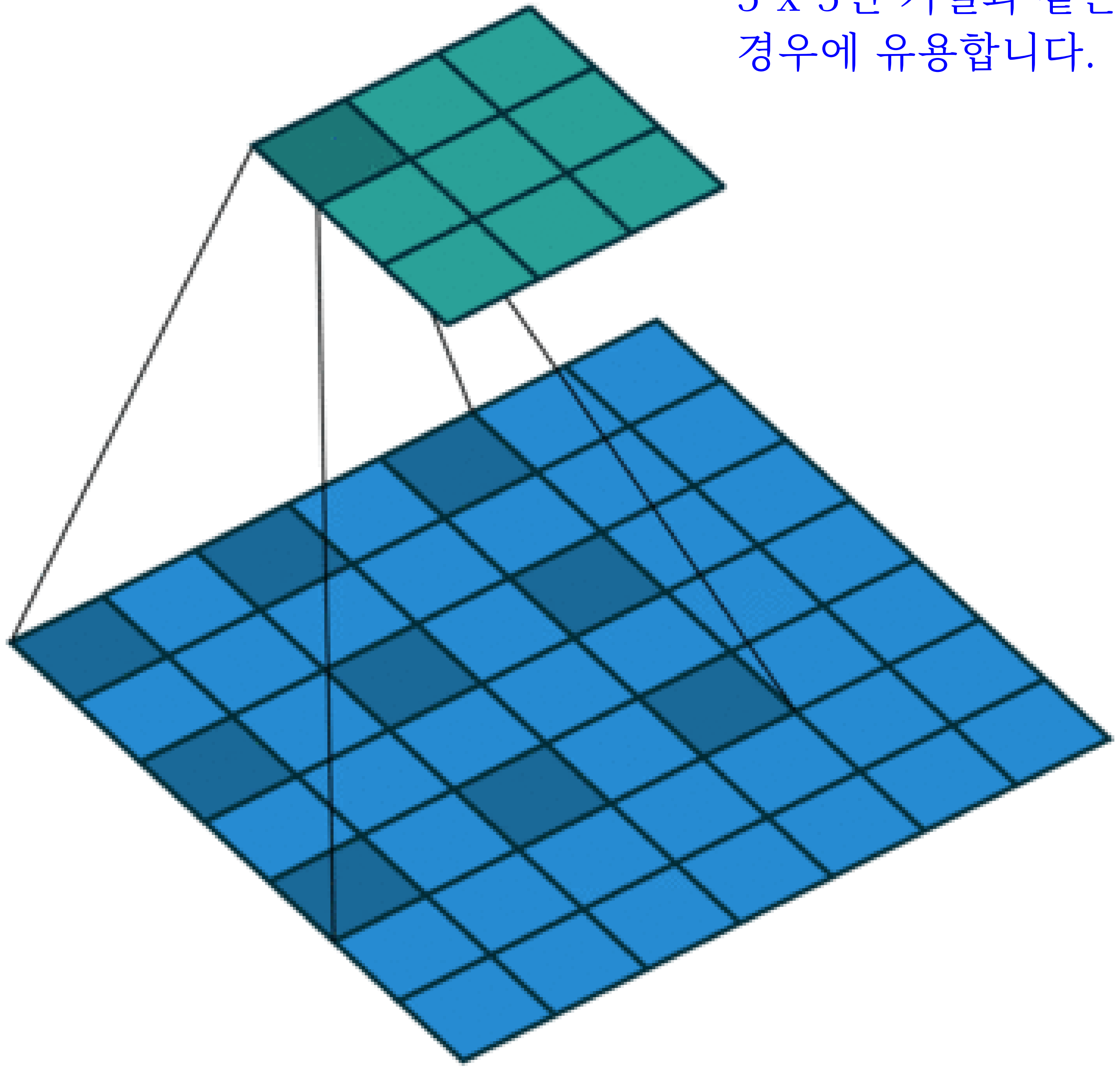
$$28 = ((28 - 3 + 2)/1) + 1$$

$$10 = ((28 - 3 + 2)/3) + 1$$

예를 들어, 입력이 28픽셀이고 커널이 3x3이고 패딩이 1이고 보폭이 1이면 이 숫자는 (28 - 3 + 2) / 1 + 1 = 28이 됩니다.
stride는 3이고 출력 크기는 10 x 10입니다.

Dilated convolutions

마지막으로 커널의 영향을 받는 픽셀 사이의 간격을 조정할 수도 있습니다. 이것을 확장된 컨볼루션이라고 합니다. 이 경우 컨볼루션 커널에는 9개의 매개변수만 있지만 크기가 5 x 5인 커널과 같은 시야각을 갖습니다. 이는 여러 척도에 걸쳐 정보를 집계해야 하는 경우에 유용합니다.



Dilation in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
          dilation_rate=2)
```

이 역시 후속 픽셀 사이의 거리를 설정하는 키워드 인수 "dilation_rate"를 통해 제어됩니다.

Let's practice!

IMAGE PROCESSING WITH KERAS IN PYTHON