




Stride & Zero Padding

최석재 lingua@naver.com

스트라이드 *stride*

- 필터 커널이 이미지에 적용된 후, 이동하는 크기를 스트라이드라고 한다
- 이미지의 크기를 유지하려고 하는 경우는 $\text{stride}=1$ 을,
- 이미지의 크기를 절반으로 줄이려고 하는 경우는 $\text{stride}=2$ 를 많이 사용한다
- 자원 소모를 줄이기 위해 초기에 $\text{stride}=2$ 를 사용하다가,
- 세부적인 특징을 추출하려고 할 때 $\text{stride}=1$ 을 사용하기도 한다

- 패딩을 같이 써야 $\text{stride}=1$ 일 때 이미지의 사이즈가 보존된다
- 풀링 윈도우에서도 스트라이드 개념을 사용한다



| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 |

스트라이드 1

Stride = 1

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 |

Stride = 1

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |

Filter

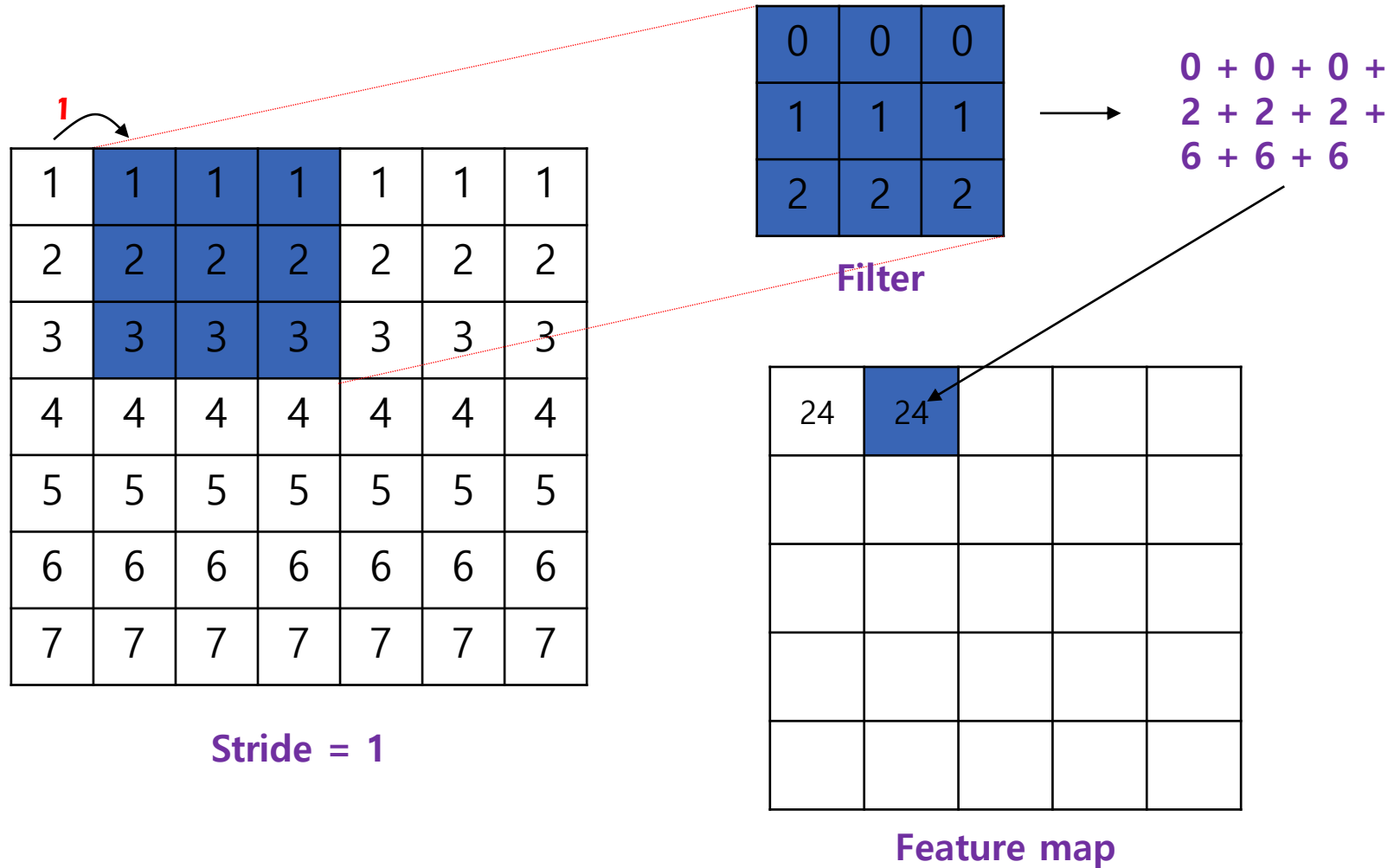
필터 = 커널 = 마스크 = 윈도우
가중치는 랜덤

$$\begin{aligned} &0 + 0 + 0 + \\ &2 + 2 + 2 + \\ &6 + 6 + 6 \end{aligned}$$

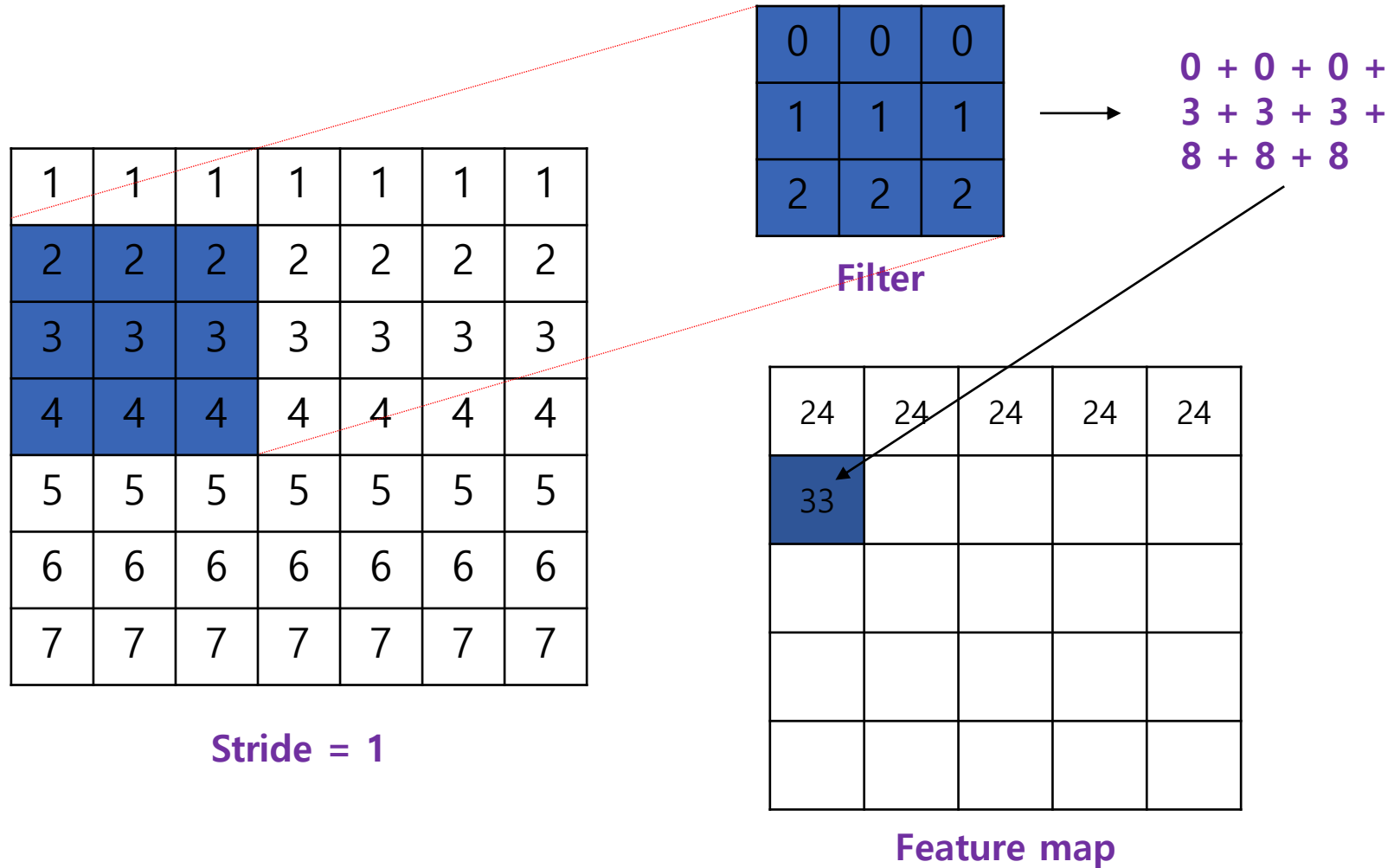
| | | | | |
|----|--|--|--|--|
| 24 | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Feature map
(Activation map)

스트라이드 1

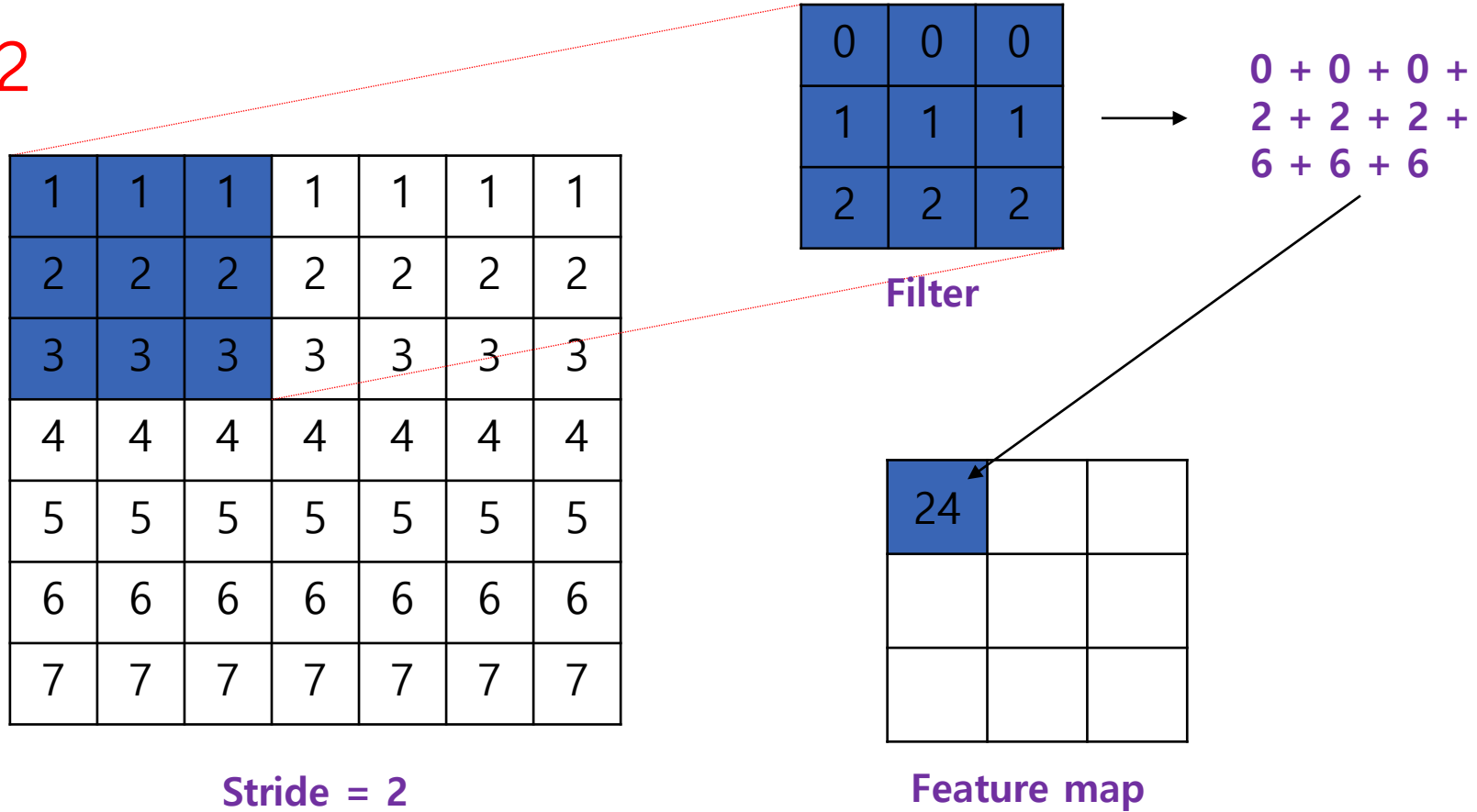


스트라이드 1

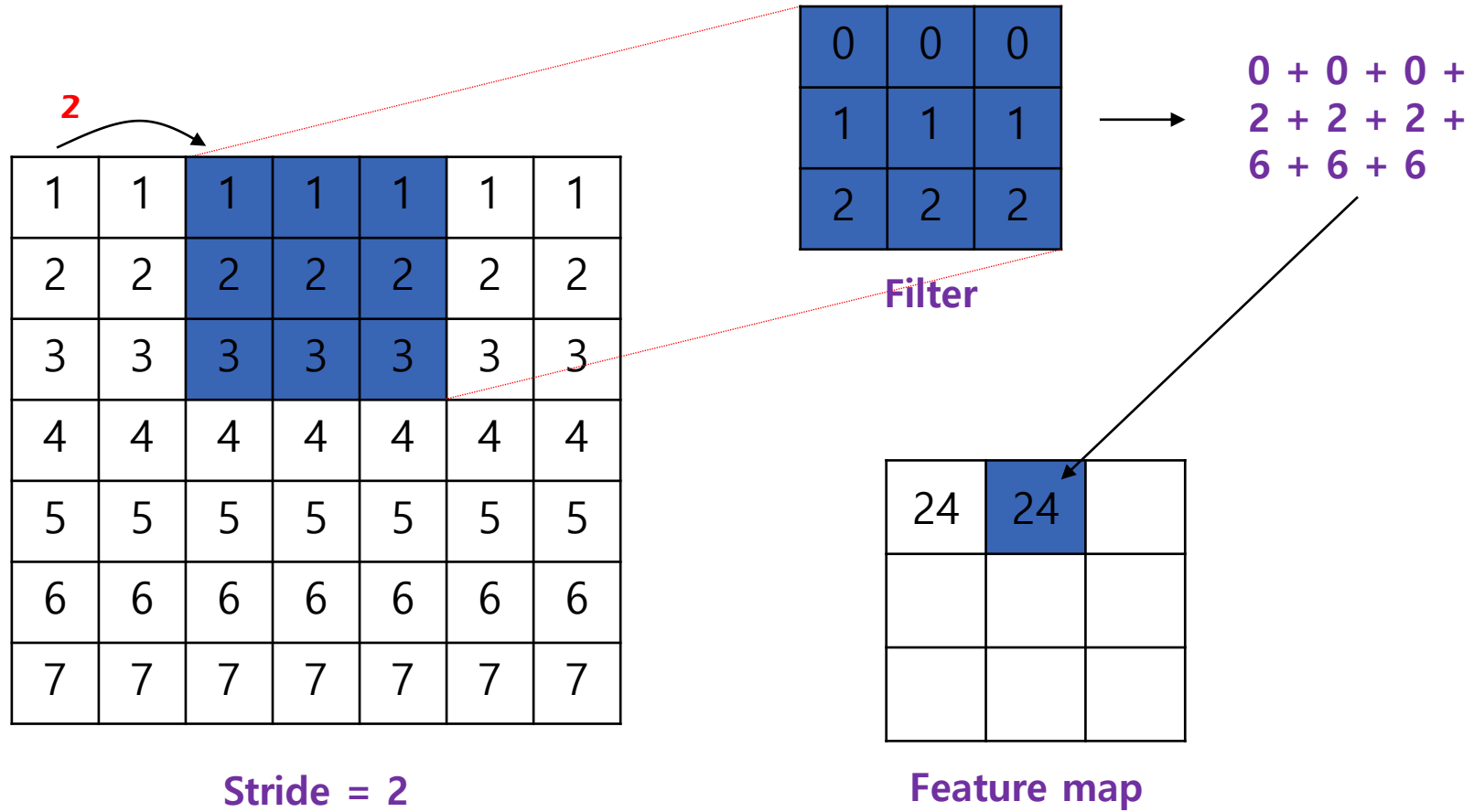


스트라이드 2

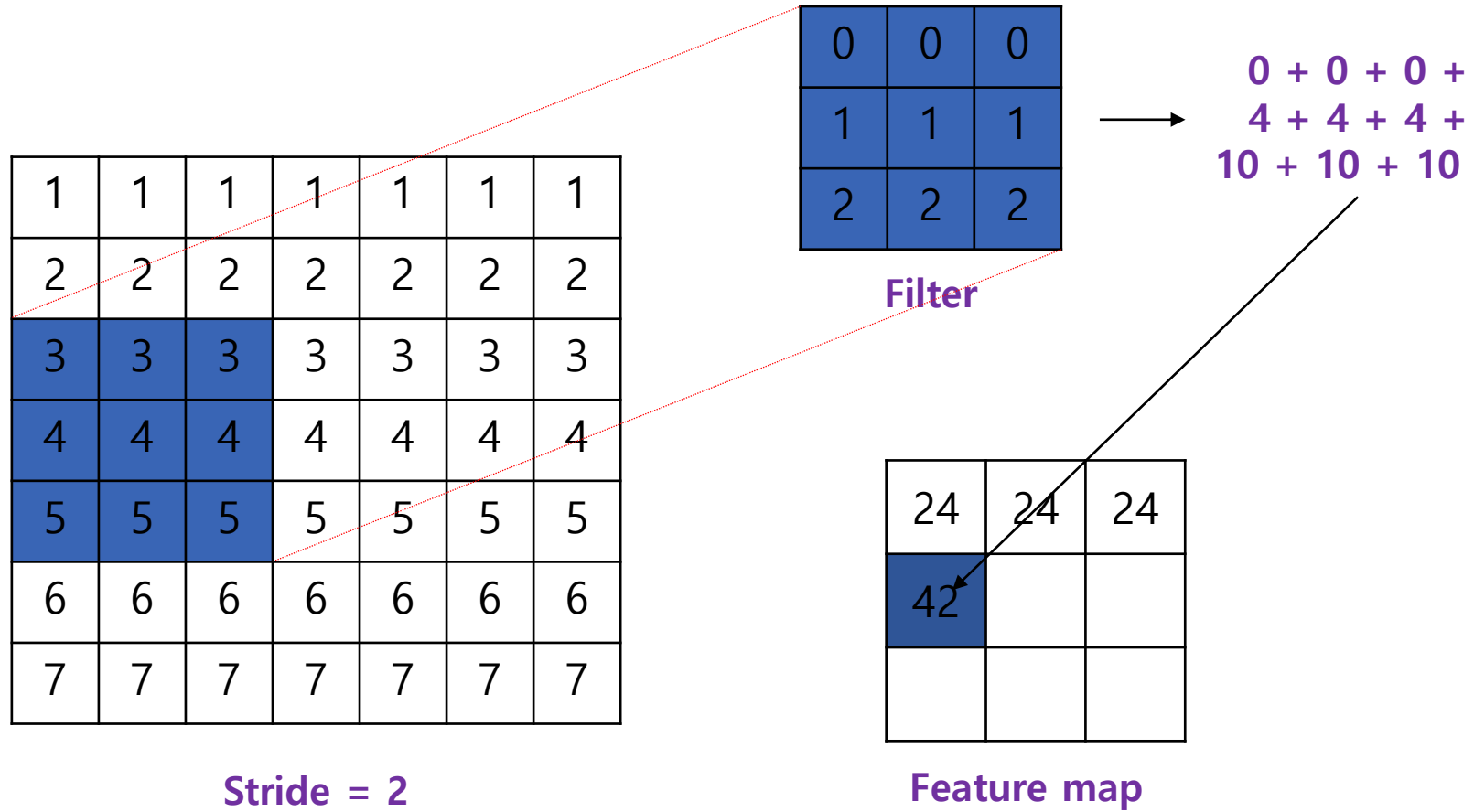
Stride = 2



스트라이드 2



스트라이드 2



Stride가 고려된 합성곱 함수 (전체)

```
• import numpy as np
• def Conv2D(img, kernel=None, stride=1):
    h, w = img.shape
    kh, kw = kernel.shape

    output_height = ((h - kh) // stride) + 1
    output_width = ((w - kw) // stride) + 1
    img_out = np.zeros((output_height, output_width), dtype=np.uint8)

    for y in range(output_height):
        for x in range(output_width):
            roi = img[y*stride:y*stride+kh, x*stride:x*stride+kw]
            filtered = roi * kernel
            conv_value = np.abs(np.sum(filtered))
            img_out[y, x] = np.uint8(conv_value)

    return img_out
```

이미지의 높이와 너비

커널(필터)의 높이와 너비

stride 폭만큼으로 나누어 결과 이미지 높이 계산

stride 폭만큼으로 나누어 결과 이미지 너비 계산

결과 이미지 초기화

관심 영역 추출

필터링

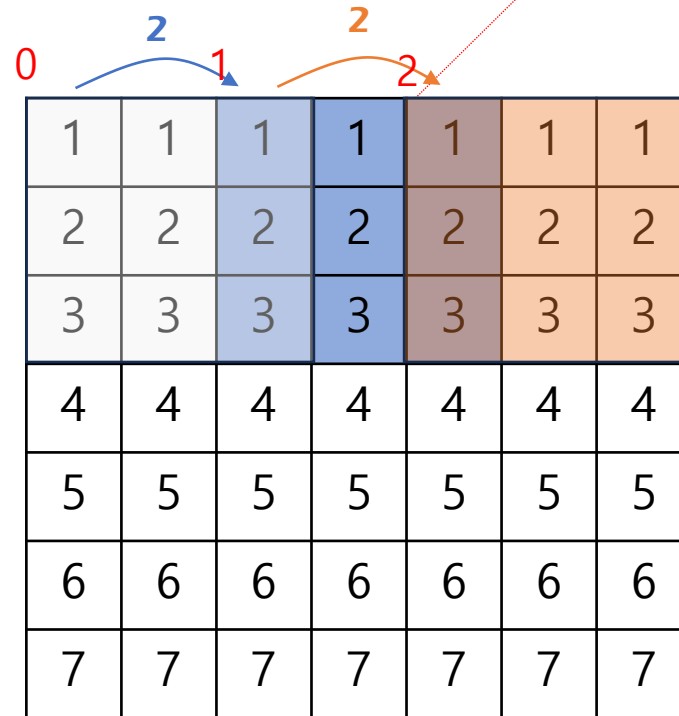
절대값 적용 후 합산

결과 저장

절대값은 사용하지 않아도 되나,
효과적인 엣지 검출을 위해서 사용되기도 한다

결과 이미지 너비 계산 부분

- $\text{output_width} = ((w - kw) \text{ // } \text{stride}) + 1$
- stride 크기만큼 건너뛰며 결과 이미지를 만든다



Stride = 2

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |

$$\begin{aligned} &0 + 0 + 0 + \\ &2 + 2 + 2 + \\ &6 + 6 + 6 \end{aligned}$$

| | | |
|----|----|----|
| 24 | 24 | 24 |
| | | |
| | | |

stride 2, 이미지 너비 7, 필터 너비 3 일 때
결과 이미지의 너비는 $(7 - 3) \text{ // } 2 + 1 = 3$

필터 시작 위치

- $\text{roi} = \text{img}[\text{y} * \text{stride} : \text{y} * \text{stride} + \text{kh}, \text{x} * \text{stride} : \text{x} * \text{stride} + \text{kw}]$
- y와 x의 값은 처음에는 (0, 0)이므로 처음 시작 위치는 (0, 0)이 되고,
- 1번 순회를 거친 다음에는 stride의 크기만큼 이동하여 합성곱 연산을 수행한다

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 |

Stride = 2

구글 드라이브 접속

- 구글 드라이브 접속
- `from google.colab import drive`
- `drive.mount('/content/gdrive')`
- 경로 변경
- `%cd /content/gdrive/MyDrive/pytest_img/opencv/`

시각화

- `import cv2`
- `from google.colab.patches import cv2_imshow`
- `img = cv2.imread("mountain.jpg", cv2.IMREAD_GRAYSCALE)` # 단채널 이미지로 읽기 위해
- `kernel = np.array([[0,0,0], [-1,2,-1], [0,0,0]])` # 커널
- `output = Conv2D(img, kernel=kernel, stride=2)` # Conv2D 함수 실행
- `cv2_imshow(img)`
- `cv2_imshow(output)`
- `cv2.waitKey(0)` # 사용자의 반응을 기다리는 함수
- `cv2.destroyAllWindows()` # 모든 윈도우를 닫는다

시각화

- `print(img.shape)` # (700, 1024)
- `print(output.shape)` # (349, 511)



절반 크기로 출력되었다

제로 패딩 *Zero Padding*

- 제로 패딩은 합성곱 연산 시 이미지의 크기가 줄어들지 않도록
- 원본 이미지 테두리에 화소를 추가하여 연산하는 것을 말한다
- 필터의 크기가 1x1보다 크면
- 결과 feature map은 원본 이미지에 비하여 사이즈가 줄게 된다
- 합성곱 연산을 반복적으로 하게 되면 출력 사이즈가 결국 1까지 된다
- 이와 같이 되는 것을 막기 위하여 이미지의 주변에 패딩을 넣어 출력 사이즈를 보존한다
- 케라스에서는 padding='same'은 동일한 사이즈의 출력을 내놓게 하는 패딩을,
- padding='valid'로 패딩을 넣지 않게 할 수 있다

Feature Map

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 |

7 x 7 x 1 image

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |

3 x 3 x 1 filter

stride=1

| | | | | |
|----|----|----|----|----|
| 24 | 24 | 24 | 24 | 24 |
| 33 | 33 | 33 | 33 | 33 |
| 42 | 42 | 42 | 42 | 42 |
| 51 | 51 | 51 | 51 | 51 |
| 60 | 60 | 60 | 60 | 60 |

5 x 5 x 1 feature map

or

stride=2

| | | |
|----|----|----|
| 24 | 24 | 24 |
| 42 | 42 | 42 |
| 60 | 60 | 60 |

3 x 3 x 1 feature map

Feature Map

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |
| 1 | 2 | 3 | 4 | 5 | | |
| | | | | | | |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 |

7 x 7 x 1 image

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |

3 x 3 x 1 filter

padding='valid'

| | | | | |
|----|----|----|----|----|
| 24 | 24 | 24 | 24 | 24 |
| 33 | 33 | 33 | 33 | 33 |
| 42 | 42 | 42 | 42 | 42 |
| 51 | 51 | 51 | 51 | 51 |
| 60 | 60 | 60 | 60 | 60 |

5 x 5 x 1 feature map

7 x 7 윈도우에는 stride=1에서 3 x 3 필터가
높이와 너비 방향에 5번씩 올 수 있으므로
출력층은 5 x 5 윈도우를 갖게 된다

Zero Padding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | | |
| | | | | | | | | |
| 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 0 |
| 0 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 0 |
| 0 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 0 |
| 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 0 |
| 0 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |

3 x 3 x 1 filter

padding='same'

| | | | | | | |
|----|----|----|----|----|----|----|
| 10 | 15 | 15 | 15 | 15 | 15 | 10 |
| 16 | 24 | 24 | 24 | 24 | 24 | 16 |
| 22 | 33 | 33 | 33 | 33 | 33 | 22 |
| 28 | 42 | 42 | 42 | 42 | 42 | 28 |
| 34 | 51 | 51 | 51 | 51 | 51 | 34 |
| 40 | 60 | 60 | 60 | 60 | 60 | 40 |
| 14 | 21 | 21 | 21 | 21 | 21 | 14 |

7 x 7 x 1 feature map

CNN은 출력 결과인 feature map이 작아지므로,
반복적으로 사용하는 데 제약이 따른다

이에 padding을 사용하면 동일한 사이즈를 조절할 수 있다
예를 들어, stride=1에 0값을 갖는 padding을 행과 열에 추가하면
입력과 동일한 사이즈로 출력된다

Conv2D 층에서는 padding='same' 으로 구현되며,
같은 수가 나올 수 있도록 자동으로 적절한 개수의 행과
열이 추가된다

단, stride=1에서만 유지되고, stride=2 이상에서는
padding='same'을 하더라도 사이즈가 대략 절반으로 줄어든다

Zero Padding이 고려된 합성곱 함수 (1/2)

- `import numpy as np`
- `def conv2d_with_padding(img, kernel, stride=1, padding=1):`
 - `h, w = img.shape` # 이미지의 높이와 너비
 - `kh, kw = kernel.shape` # 커널(필터)의 높이와 너비
 - `padded_h = h + 2 * padding` # 패딩이 적용된 이미지의 높이 계산
 - `padded_w = w + 2 * padding` # 패딩이 적용된 이미지의 너비 계산
 - `padded_img = np.zeros((padded_h, padded_w))` # 패딩된 크기로 결과 이미지를 초기화
 - `padded_img[padding:h + padding, padding:w + padding] = img` # 패딩 위치를 제외하고 이미지를 삽입
 - `output_height = ((padded_h - kh) // stride) + 1` # stride가 적용된 결과 이미지 높이 계산
 - `output_width = ((padded_w - kw) // stride) + 1` # stride가 적용된 결과 이미지 너비 계산
 - `img_out = np.zeros((output_height, output_width), dtype=np.uint8)` # stride가 적용된 결과 이미지 초기화

Zero Padding이 고려된 합성곱 함수 (2/2)

합성곱 연산 수행

for y in range(output_height):

for x in range(output_width):

roi = padded_img[y*stride:y*stride+kh, x*stride:x*stride+kw] # 관심 영역 추출

filtered = roi * kernel # 필터링

conv_value = np.abs(np.sum(filtered)) # 절대값 적용 후 합산

img_out[y, x] = np.uint8(conv_value) # 결과 저장

return img_out, padded_img # 패딩 결과 확인 위해 padded_img 출력

```
import numpy as np

def conv2d_with_padding(img, kernel, stride=1, padding=1):
    h, w = img.shape
    kh, kw = kernel.shape

    # 패딩이 적용된 이미지의 크기 계산
    padded_h = h + 2 * padding
    padded_w = w + 2 * padding

    # 패딩이 적용된 이미지 생성
    padded_img = np.zeros((padded_h, padded_w))
    padded_img[padding:h + padding, padding:w + padding] = img

    # 출력 이미지의 크기 계산
    output_height = ((padded_h - kh) // stride) + 1
    output_width = ((padded_w - kw) // stride) + 1
    img_out = np.zeros((output_height, output_width), dtype=np.uint8)

    # 합성곱 연산 수행
    for y in range(output_height):
        for x in range(output_width):
            roi = padded_img[y*stride:y*stride+kh, x*stride:x*stride+kw]
            filtered = roi * kernel
            conv_value = np.abs(np.sum(filtered))
            img_out[y, x] = np.uint8(conv_value)

    return img_out, padded_img
```

시각화

- `import cv2`
- `from google.colab.patches import cv2_imshow`
- `img = cv2.imread("mountain.jpg", cv2.IMREAD_GRAYSCALE)`
- `kernel = np.array([[0,0,0], [-1,2,-1], [0,0,0]])`
- `output, padded_img = conv2d_with_padding(img, kernel=kernel, stride=2, padding=1)`
- `cv2_imshow(img)`
- `cv2_imshow(output)`
- `cv2.waitKey(0)`
- `cv2.destroyAllWindows()`

시각화

- `print(img.shape)` # (700, 1024)
- `print(output.shape)` # (350, 512)



패딩으로 (349, 511) → (350, 512) 사이즈가 되었다

패딩 확인

- 합성곱 연산 전의 패딩된 이미지를 확인해본다
- `print("원본 이미지:\n", img)`
- `print("패딩 이미지:\n", padded_img)`

원본 이미지:

```
[[196 194 193 ... 167 166 166]
 [201 199 197 ... 156 153 151]
 [207 204 201 ... 150 145 142]
 ...
 [127 174 149 ... 45 49 49]
 [ 88 135 156 ... 44 37 36]
 [150 119 146 ... 50 33 19]]
```

패딩 이미지:

```
[[ 0.  0.  0. ...  0.  0.  0.]
 [ 0. 196. 194. ... 166. 166.  0.]
 [ 0. 201. 199. ... 153. 151.  0.]
 ...
 [ 0.  88. 135. ... 37. 36.  0.]
 [ 0. 150. 119. ... 33. 19.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]]
```

- 이미지의 주변에 `padding=1`이 추가되었다
- `stride=1`의 결과도 확인해본다