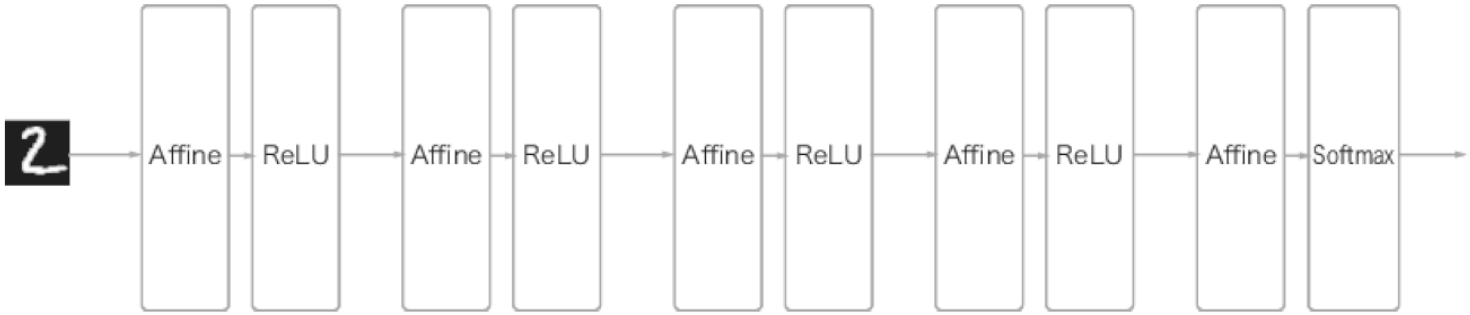
CNN using Python

Hee-il Hahn

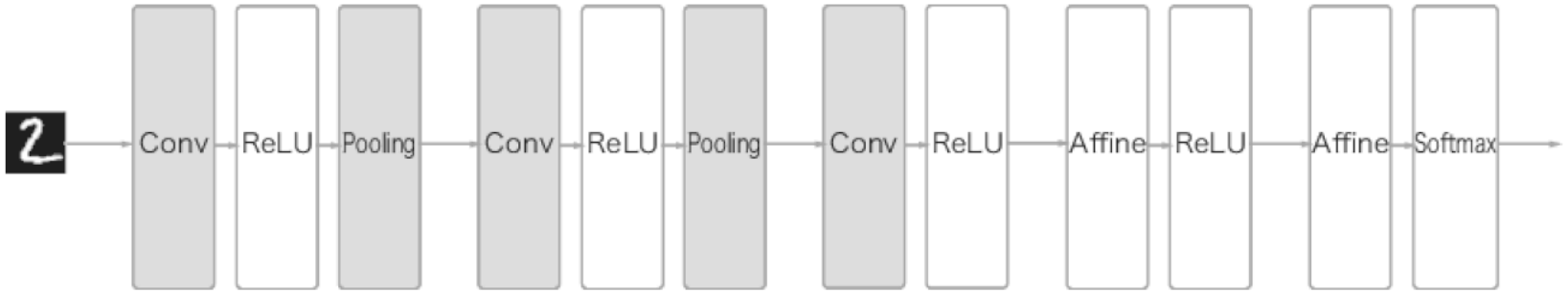
Professor
Department of Information and Communications Engineering
Hankuk University of Foreign Studies
hihahn@hufs.ac.kr

전체구조

- Fully-Connected Network



- CNN



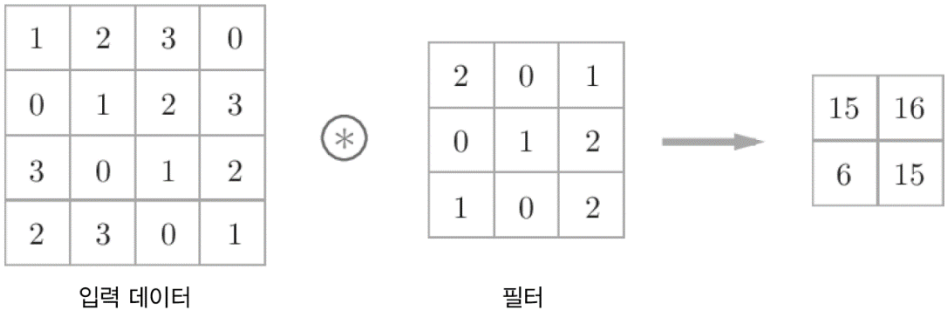
Convolution Layer

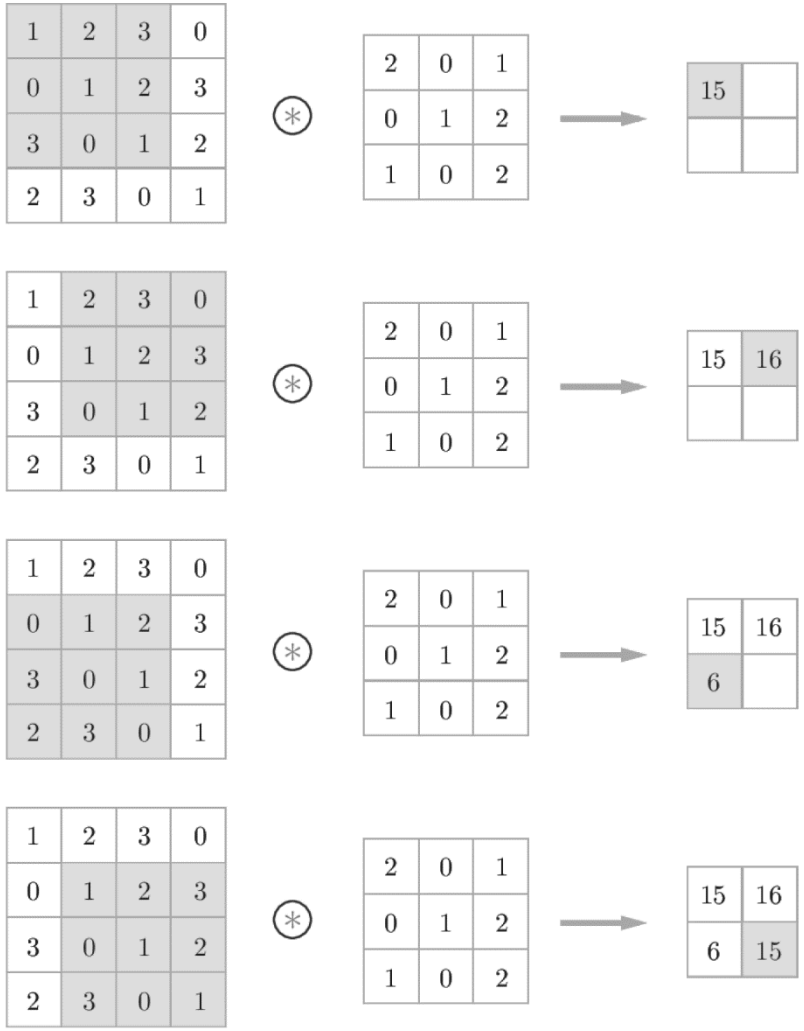
- Fully-Connected Network의 문제점
  - 예를 들어, 입력 데이터가 이미지인 경우, 삼차원 데이터를 평평한 일차원 데이터로 변환해 주어야 한다.
  - 이미지는 3차원 형상이고, 공간적 정보가 담겨 있다.
  - 즉, 공간적으로 가까운 픽셀은 매우 유사하고 RGB 각 채널은 서로 밀접하게 관련되어 있다.
  - 그러나, FCN은 형상을 무시하고 모든 입력 데이터를 동등한 뉴런으로 취급하여 형상에 담긴 점보를 살리지 못한다.

- CNN의 특징
  - 이미지는 삼차원 데이터로 입력 받고, 다음 계층에도 삼차원 데이터로 전달한다.
  - 이미지처럼 형상을 가진 데이터를 제대로 이해할 가능성이 높다.

Convolution Layer – cont.

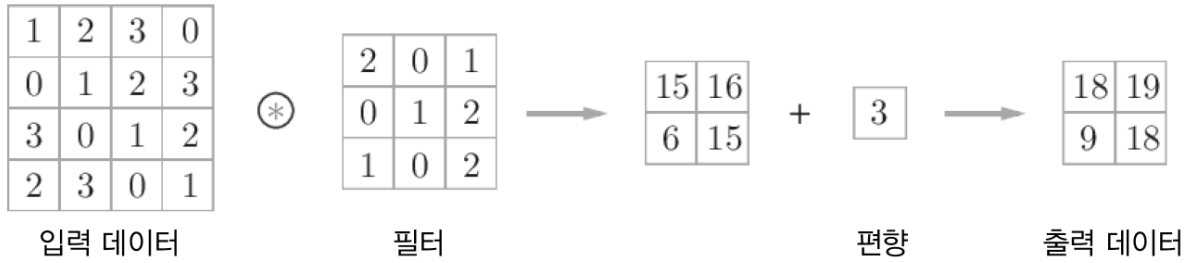
- Convolution 연산





Convolution Layer – cont.

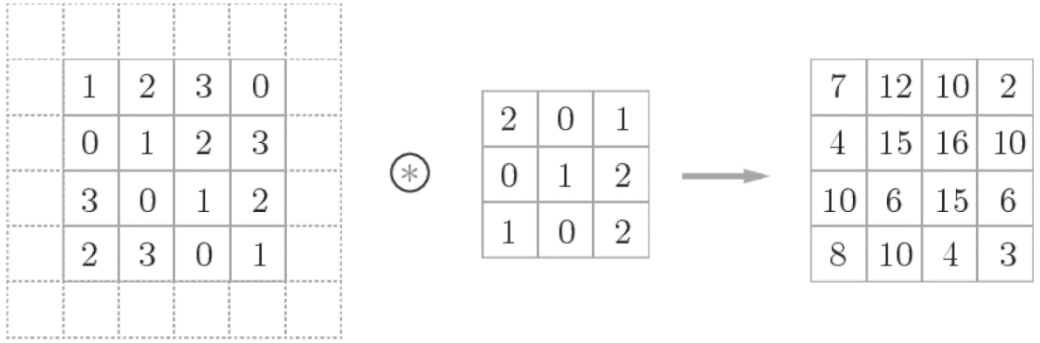
- Convolution 연산 – 계속
  - Bias를 고려하면



  - 필터를 적용한 모든 원소에 하나의 값을 더한다.

Convolution Layer – cont.

- Padding
  - Convolution을 계산하기 전에 입력 데이터 주변을 0으로 채운다.



(4, 4)
입력 데이터(패딩 : 1)

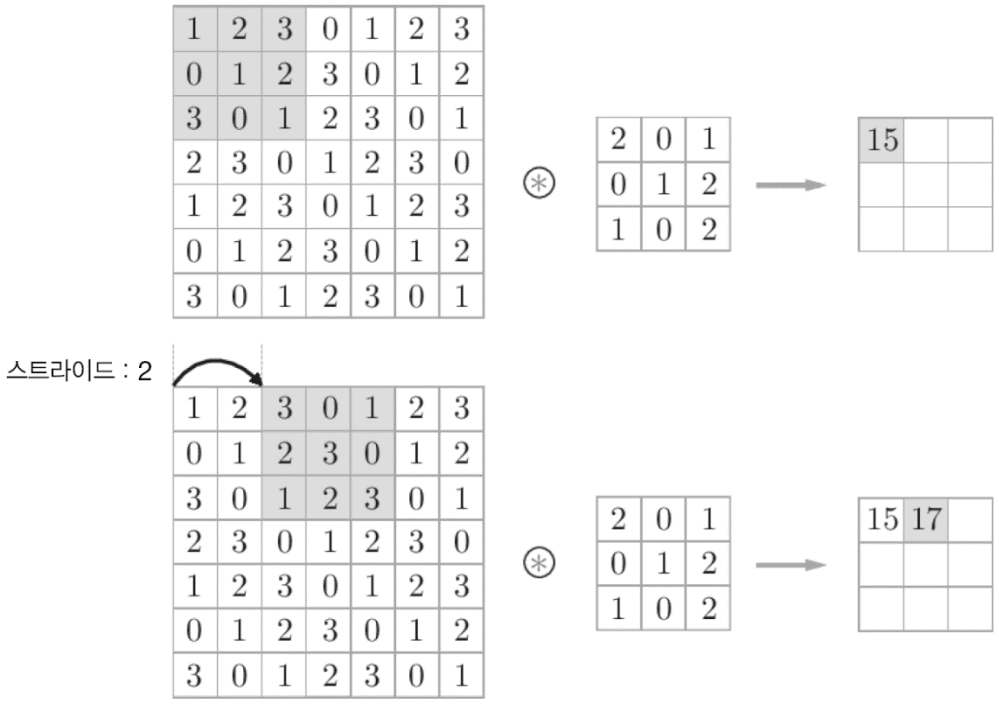
(3, 3)
필터

(4, 4)
출력 데이터

  - 출력 크기를 조정할 목적으로 사용한다.

Convolution Layer – cont.

- Stride
  - 필터를 적용하는 위치의 간격을 말한다.
  - Stride가 2인 convolution 연산은 다음과 같다.

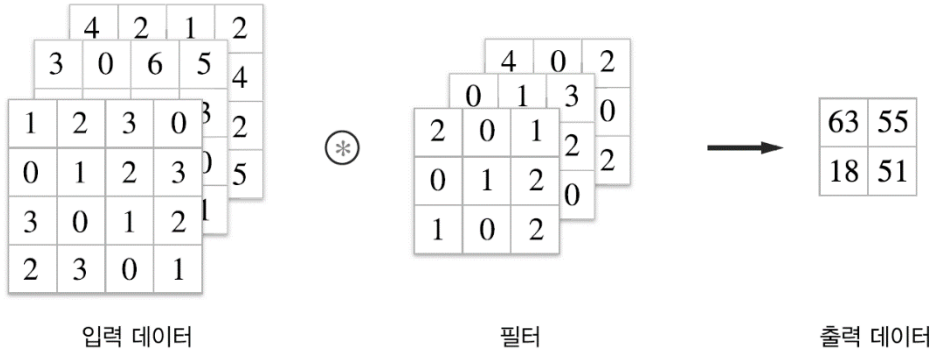


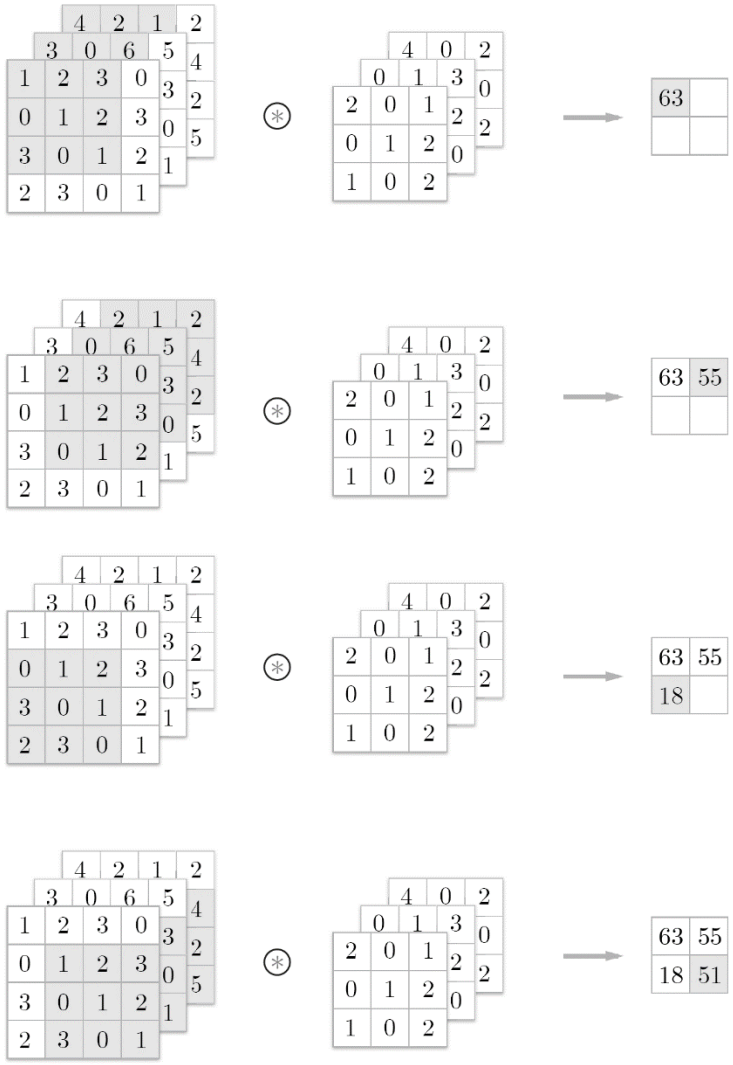
$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

Convolution Layer – cont.

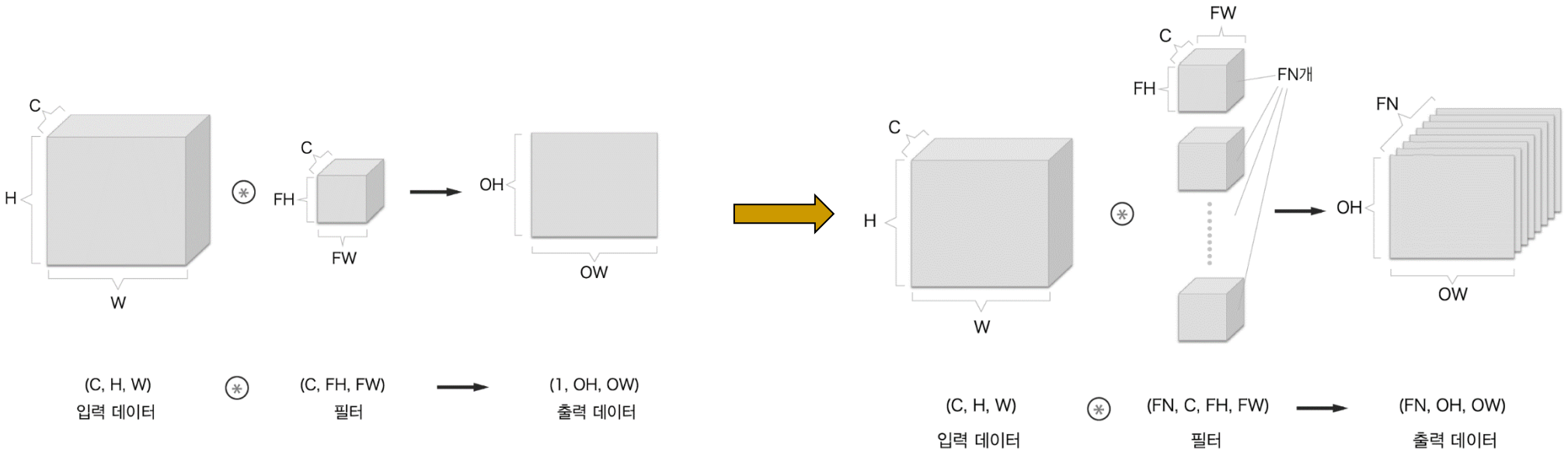
- 삼차원 데이터의 Convolution 연산
  - 채널마다 convolution연산을 수행하고
    그 결과를 더해서 하나의 출력을 얻는다.





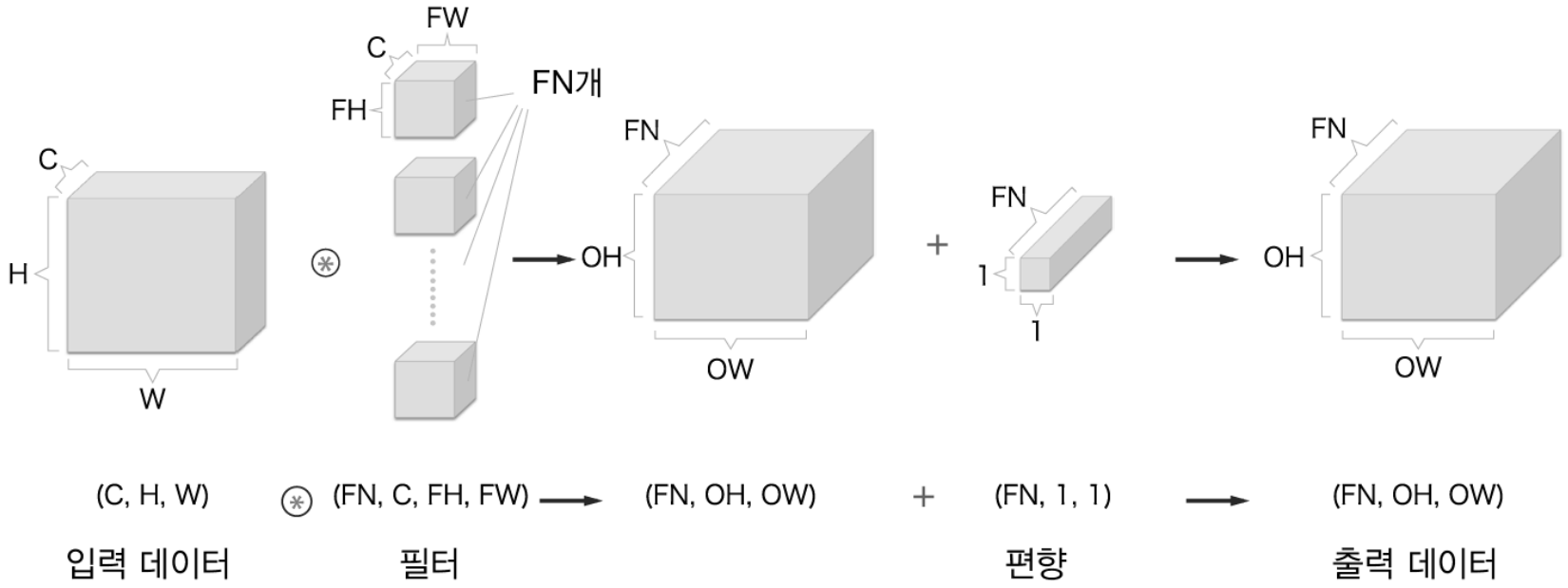
Convolution Layer – cont.

- Block으로 생각하면
  - Convolution연산을 직육면체 블록으로 생각한다.
  - 다수의 채널을 출력하고자 하면 우측 그림과 같다.



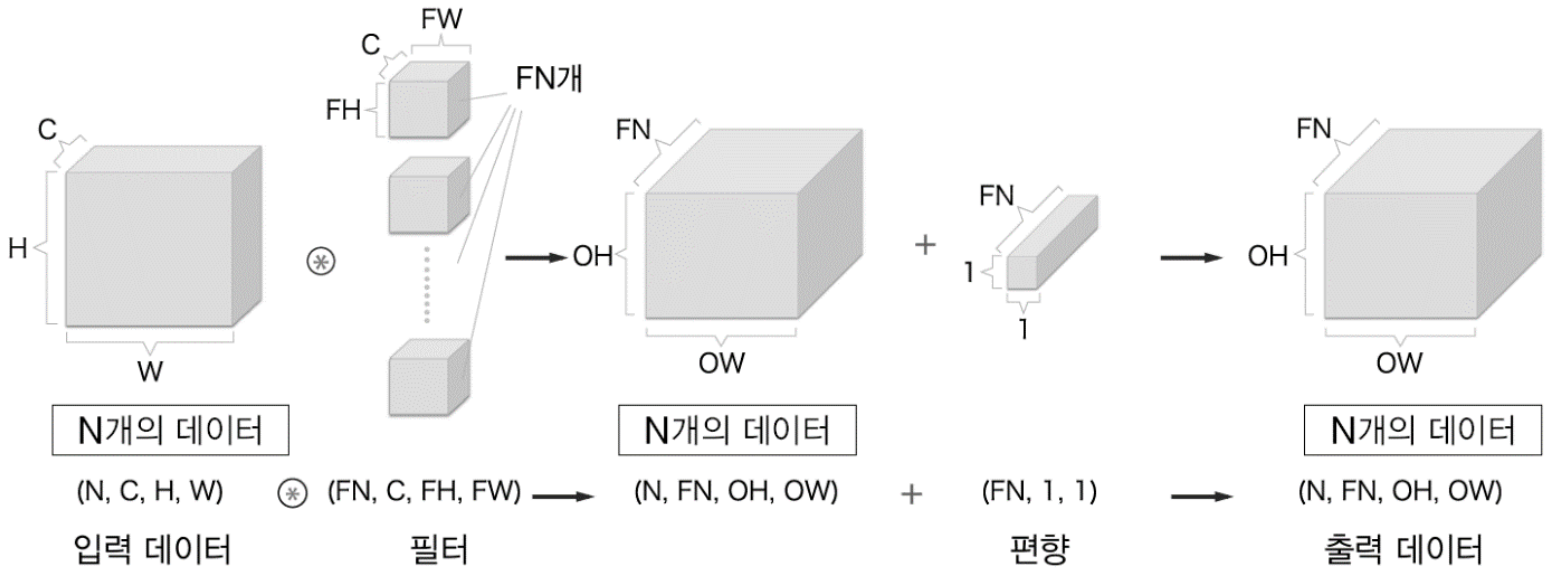
Convolution Layer – cont.

- Block으로 생각하면 - 계속
  - Bias를 추가하면 다음과 같다.



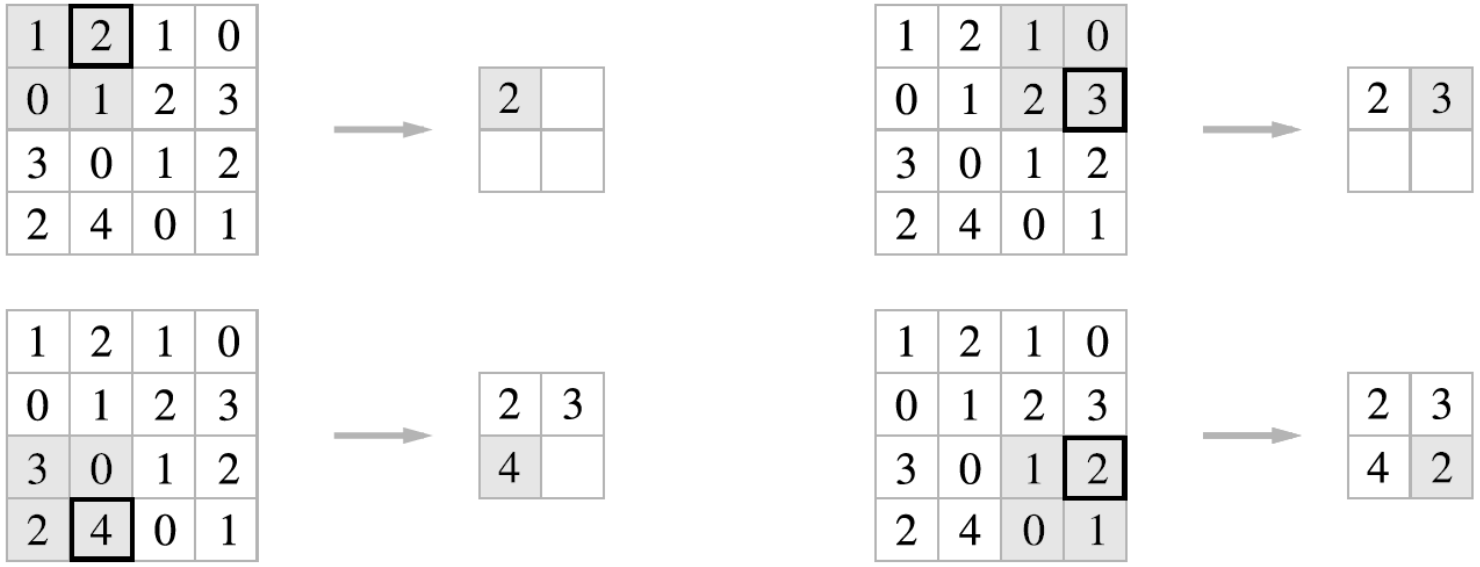
Convolution Layer – cont.

- Batch Processing
  - Convolution 연산도 배치처리를 지원한다.



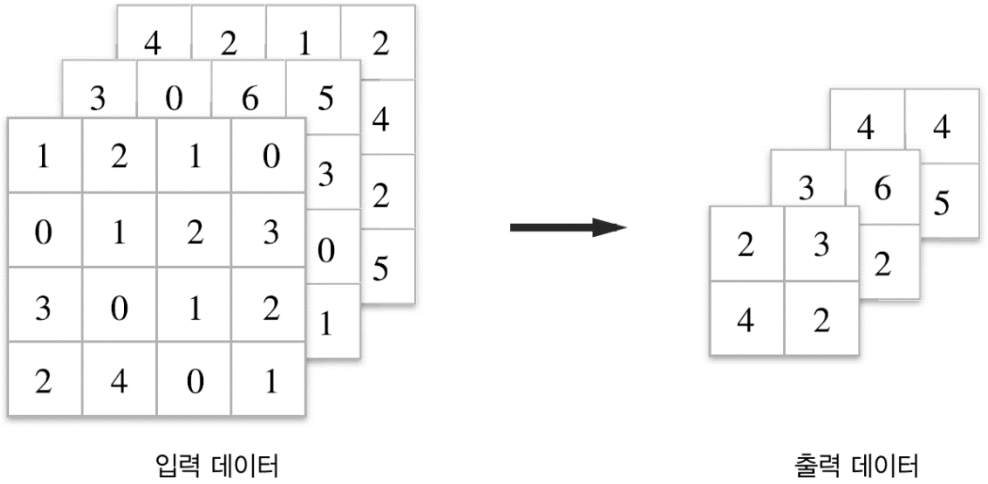
Pooling Layer

- Pooling
  - Pooling은 가로, 세로 방향의 공간을 줄이는 연산이다.
  - 2X2 max pooling 연산과정을 stride 2로 처리하면 다음과 같다.
  - Stride 크기는 풀링의 window 크기에 따라 결정한다.
  - Max pooling 외에도 average pooling, weighted average pooling 등이 있다.

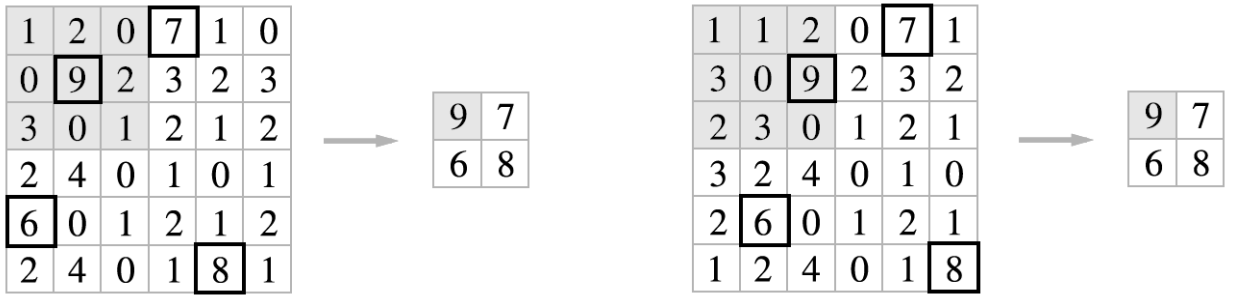


Pooling Layer

- Pooling layer의 특징
  - 학습해야 할 매개변수가 없다.
  - 채널 수가 변하지 않는다.



  - 입력의 변화에 영향을 적게 받는다(robust).

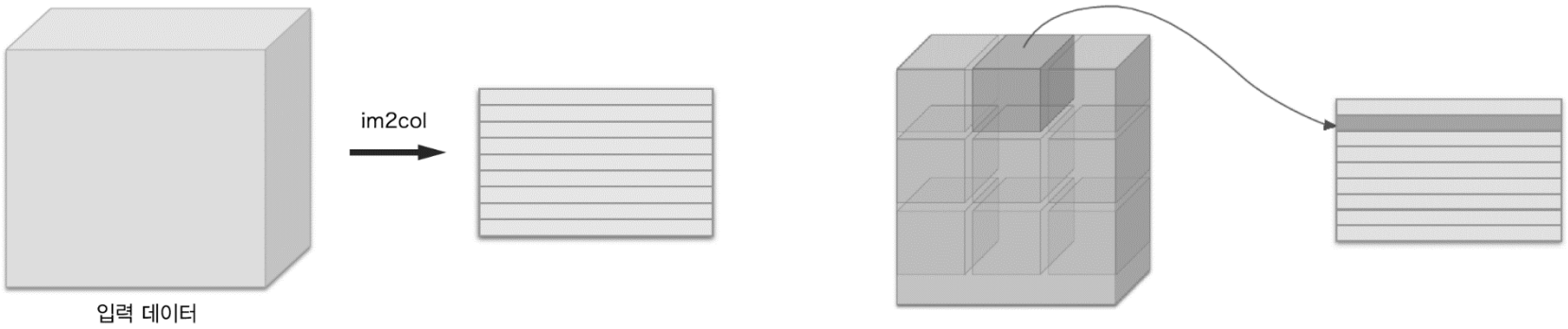


Convolution Layer / Pooling Layer 구현

- 4차원 배열
  - CNN에서 계층 사이를 흐르는 데이터는 4차원이다.
  - 예를 들어 데이터 shape가 (10, 1, 28, 28)이라면 채널 1의 28X28 이미지가 10장임을 나타낸다.
  - 이를 python으로 구현하면 >>> x = np.random.rand(10, 1, 28, 28)

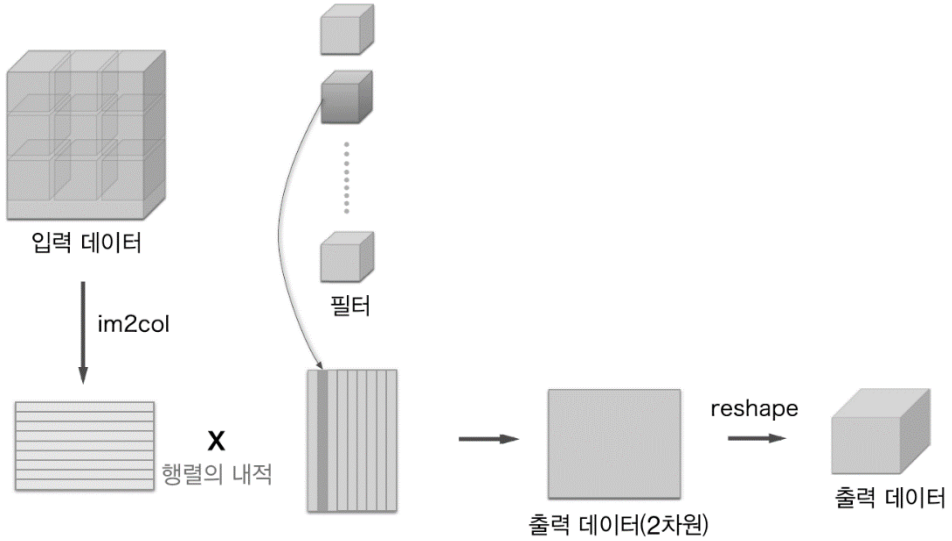
Convolution Layer / Pooling Layer 구현

- im2col로 데이터 전개
  - for 문 대신에 im2col 이라는 함수를 이용해 계산량을 줄인다.
  - im2col은 입력 데이터를 필터링하기 좋게 데이터를 전개하는 함수이다.
  - 다음과 같이 삼차원 입력 데이터에 im2col을 적용하면 이차원 행렬을 얻는다.



Convolution Layer / Pooling Layer 구현

- im2col로 데이터 전개 - 계속
  - im2col로 입력 데이터를 전개한 다음 필터를 1열로 전개하고, 두 행렬의 내적을 계산한다.



  - Im2col 방식으로 출력한 결과는 이차원 행렬이므로 4차원으로 reshape한다.

Convolution Layer / Pooling Layer 구현

- Convolution layer 구현
  - Im2col 사용 예

```
x1=np.random.rand(1, 3, 7, 7) # 데이터 수, 채널 수, 높이, 너비
col1=im2col(x1, 5, 5, stride=1, pad=0)
print(col1.shape)
x2=np.random.rand(10, 3, 7, 7) # 데이터 수, 채널 수, 높이, 너비
col2=im2col(x2, 5, 5, stride=1, pad=0)
print(col2.shape)
```

executed in 12ms, finished 17:50:35 2020-06-05

```
(9, 75)
(90, 75)
```

```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    """다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화).

    Parameters
    ----------
    input_data : 4차원 배열 형태의 입력 데이터(이미지 수, 채널 수, 높이, 너비
    filter_h : 필터의 높이
    filter_w : 필터의 너비
    stride : 스트라이드
    pad : 패딩

    Returns
    -------
    col : 2차원 배열
    """
    N, C, H, W = input_data.shape
    out_h = (H + 2*pad - filter_h)//stride + 1
    out_w = (W + 2*pad - filter_w)//stride + 1

    img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'consta
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride*out_h
        for x in range(filter_w):
            x_max = x + stride*out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N*out_h*out_w, -1)
    return col
```

Convolution Layer / Pooling Layer 구현

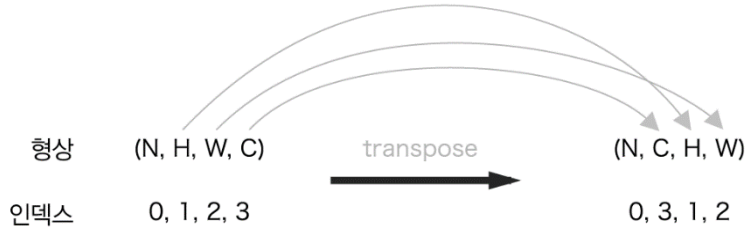
- Convolution layer 구현 - 계속
  - Im2col 을 이용하여 convolution layer를 Convolution이라는 class로 구현한다.

```python
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

        # 중간 데이터 (backward 시 사용)
        self.x = None
        self.col = None
        self.col_W = None

        # 가중치와 편향 매개변수의 기울기
        self.dW = None
        self.db = None
```

```python
    def forward(self, x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
        out_w = 1 + int((W + 2*self.pad - FW) / self.stride)

        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T

        out = np.dot(col, col_W) + self.b
        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        self.x = x
        self.col = col
        self.col_W = col_W
```

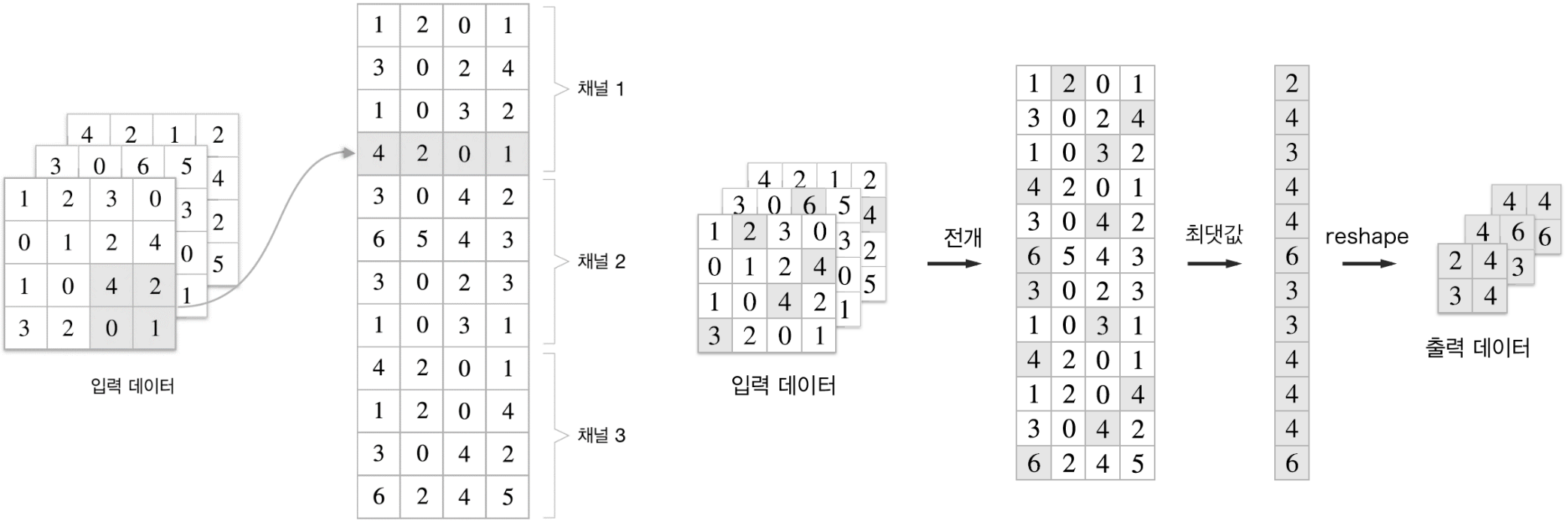
```python
    def backward(self, dout):
        FN, C, FH, FW = self.W.shape
        dout = dout.transpose(0,2,3,1).reshape(-1, FN)

        self.db = np.sum(dout, axis=0)
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)

        dcol = np.dot(dout, self.col_W.T)
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)

        return dx
```

  - Np의 transpose함수로 축 순서 변경



Convolution Layer / Pooling Layer 구현

- Pooling layer 구현
  - Pooling layer도 Im2col 을 이용하여 데이터를 전개한다.



  - 전개한 행렬에서 행 별 최대값을 구하고 reshape한다.

Convolution Layer / Pooling Layer 구현

- Pooling layer 구현 - 계속
  - Class Pooling.

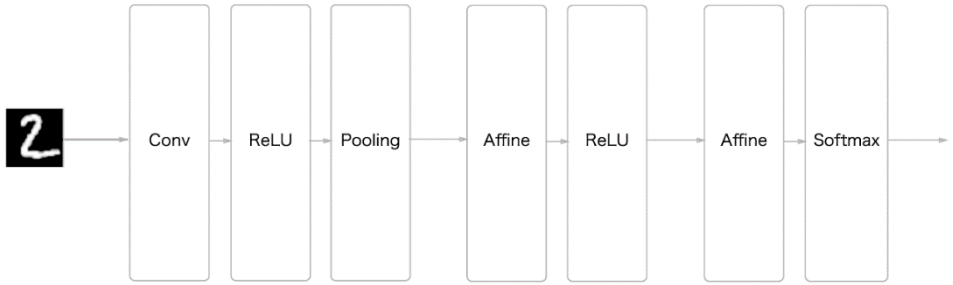
```python
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

        self.x = None
        self.arg_max = None
```

```python
    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)

        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)

        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        self.x = x
        self.arg_max = arg_max

        return out
```

```python
    def backward(self, dout):
        dout = dout.transpose(0, 2, 3, 1)

        pool_size = self.pool_h * self.pool_w
        dmax = np.zeros((dout.size, pool_size))
        dmax[np.arange(self.arg_max.size), self.arg_max.flatten()] = dout.flat
        dmax = dmax.reshape(dout.shape + (pool_size,))

        dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
        dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride,

        return dx
```

CNN 구현

- 구현할 CNN 구조 및 코드




```python
class SimpleConvNet:
    def __init__(self, input_dim=(1, 28, 28),
                 conv_param={'filter_num':30, 'filter_size':5, 'pad':0, 'stric
                 hidden_size=100, output_size=10, weight_init_std=0.01):
        filter_num = conv_param['filter_num']
        filter_size = conv_param['filter_size']
        filter_pad = conv_param['pad']
        filter_stride = conv_param['stride']
        input_size = input_dim[1]
        conv_output_size = (input_size - filter_size + 2*filter_pad) / filter_
        pool_output_size = int(filter_num * (conv_output_size/2) * (conv_outpu
```

```python
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * W
                            np.random.randn(filter_num, input_dim[0], filter_s
        self.params['b1'] = np.zeros(filter_num)
        self.params['W2'] = weight_init_std * W
                            np.random.randn(pool_output_size, hidden_size)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = weight_init_std * W
                            np.random.randn(hidden_size, output_size)
        self.params['b3'] = np.zeros(output_size)
```

```python
        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Conv1'] = Convolution(self.params['W1'], self.params['b
                                           conv_param['stride'], conv_param[
        self.layers['Relu1'] = Relu()
        self.layers['Pool1'] = Pooling(pool_h=2, pool_w=2, stride=2)
        self.layers['Affine1'] = Affine(self.params['W2'], self.params['b2']
        self.layers['Relu2'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W3'], self.params['b3']

        self.last_layer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x
```

CNN 구현 – cont.

- 구현할 CNN 구조 및 코드 - 계속

```python
    def loss(self, x, t):
        y = self.predict(x)
        return self.last_layer.forward(y, t)

    def accuracy(self, x, t, batch_size=100):
        if t.ndim != 1 : t = np.argmax(t, axis=1)
        acc = 0.0
        for i in range(int(x.shape[0] / batch_size)):
            tx = x[i*batch_size:(i+1)*batch_size]
            tt = t[i*batch_size:(i+1)*batch_size]
            y = self.predict(tx)
            y = np.argmax(y, axis=1)
            acc += np.sum(y == tt)

        return acc / x.shape[0]
```

```python
    def gradient(self, x, t):
        # forward
        self.loss(x, t)
        # backward
        dout = 1
        dout = self.last_layer.backward(dout)
        layers = list(self.layers.values())
        layers.reverse()
        for layer in layers:
            dout = layer.backward(dout)
        # 결과 저장
        grads = {}
        grads['W1'], grads['b1'] = self.layers['Conv1'].dW, self.layers['Con
        grads['W2'], grads['b2'] = self.layers['Affine1'].dW, self.layers['A
        grads['W3'], grads['b3'] = self.layers['Affine2'].dW, self.layers['A

        return grads
```

CNN 구현 – cont.

- 구현할 CNN 구조 및 코드 – 계속
  - Main function

```
# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=False)

max_epochs = 20

network = SimpleConvNet(input_dim=(1,28,28),
                        conv_param = {'filter_num': 30, 'filter_size': 5, 'pad
                        hidden_size=100, output_size=10, weight_init_std=0.01)

trainer = Trainer(network, x_train, t_train, x_test, t_test,
                  epochs=max_epochs, mini_batch_size=100,
                  optimizer='Adam', optimizer_param={'lr': 0.001},
                  evaluate_sample_num_per_epoch=1000)
trainer.train()
```

수고하셨습니다.