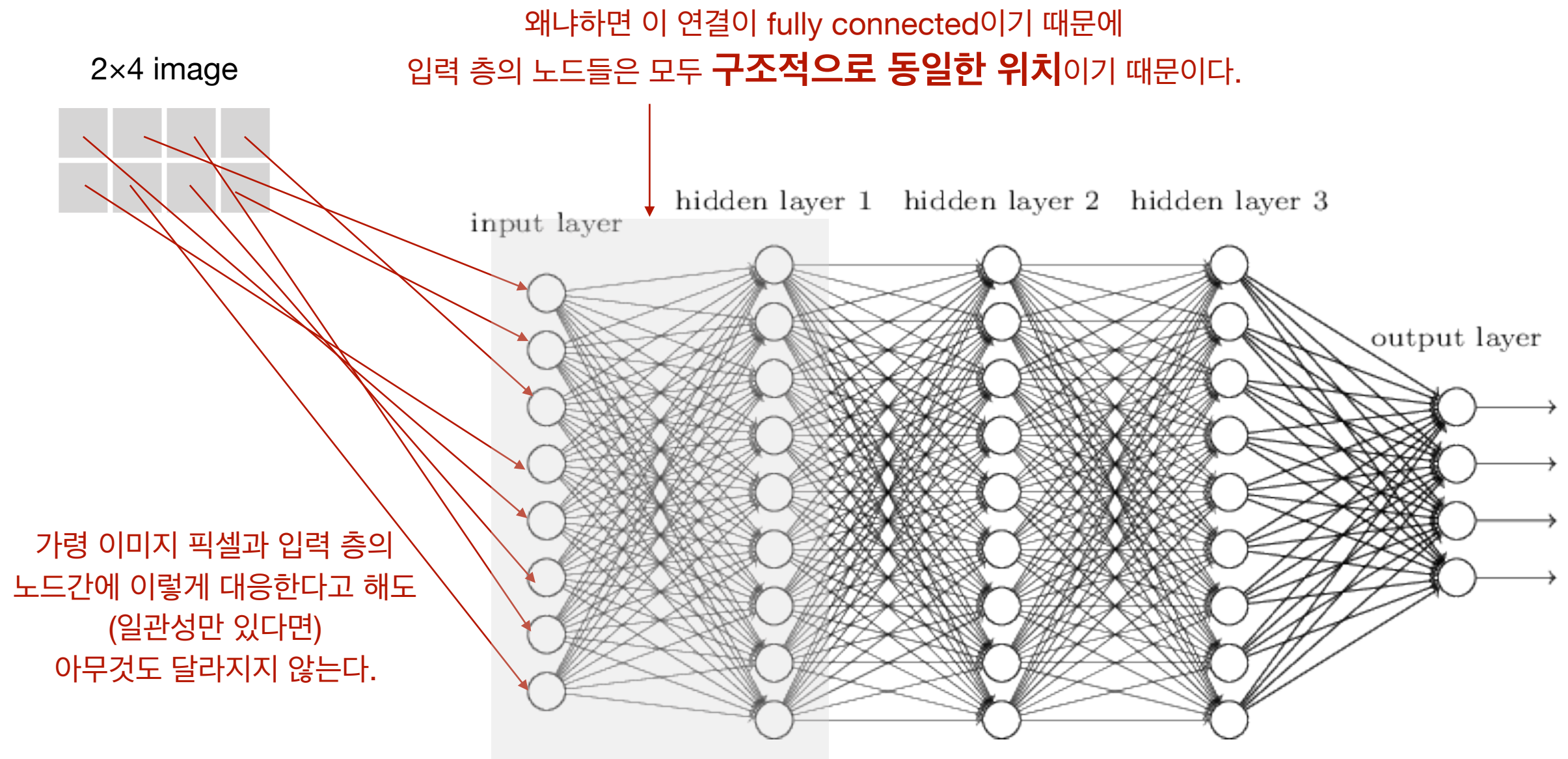


[2017 Winter]

Tensorflow Tutorial

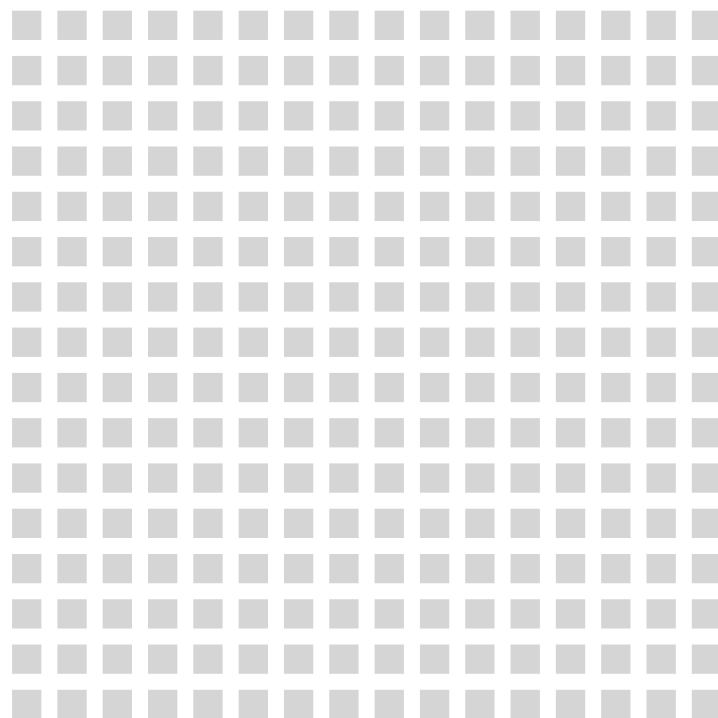
03. Convolutional Neural Network (CNN)

- 이미지는 다른 유형의 데이터와 달리 데이터 내에서 각 구성요소의 상대적 위치가 중요한 의미를 가진다.
- Fully connected layer는 이미지가 가진 지역적 특성(local feature)를 효과적으로 추출하지 못한다.

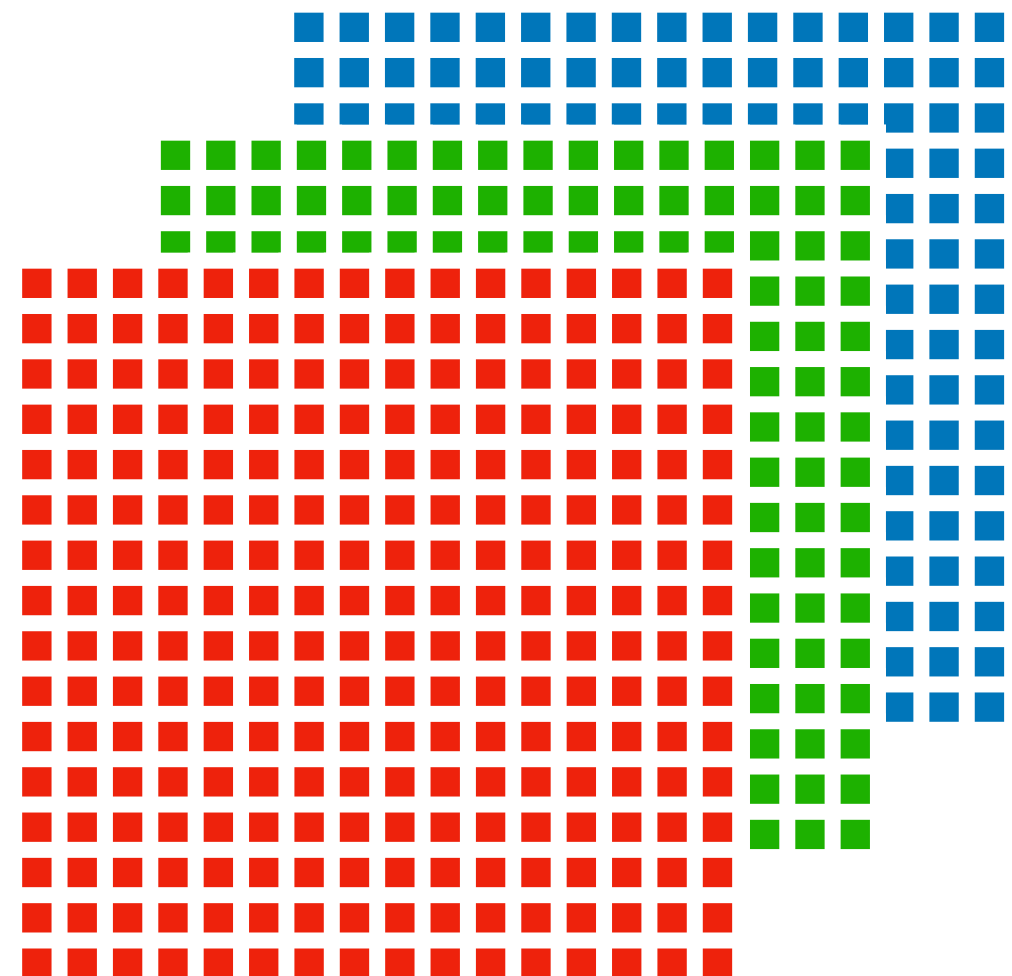


- Fully connected 네트워크는 이미지가 가진 **국부적(local) 특성**을 효과적으로 추출하지 못한다.

- 입력 이미지를 1차원 벡터로 변환하지 않고 2차원 (혹은 RGB이미지의 경우 3차원) 구조를 그대로 유지한다.

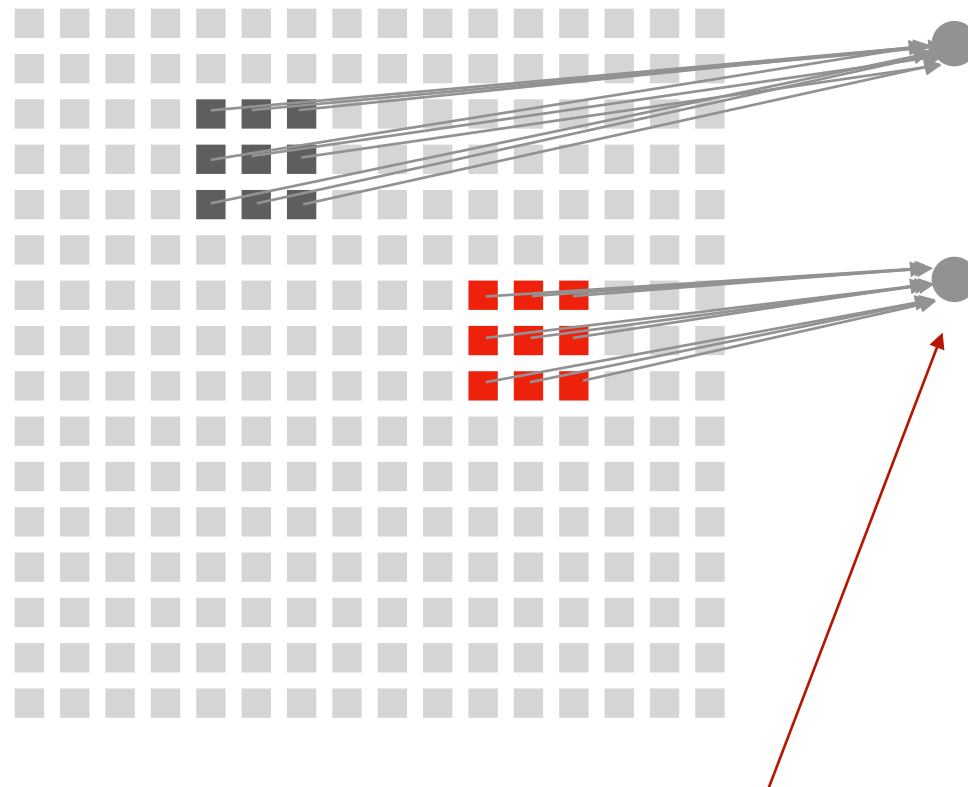


1 채널 이미지의 경우

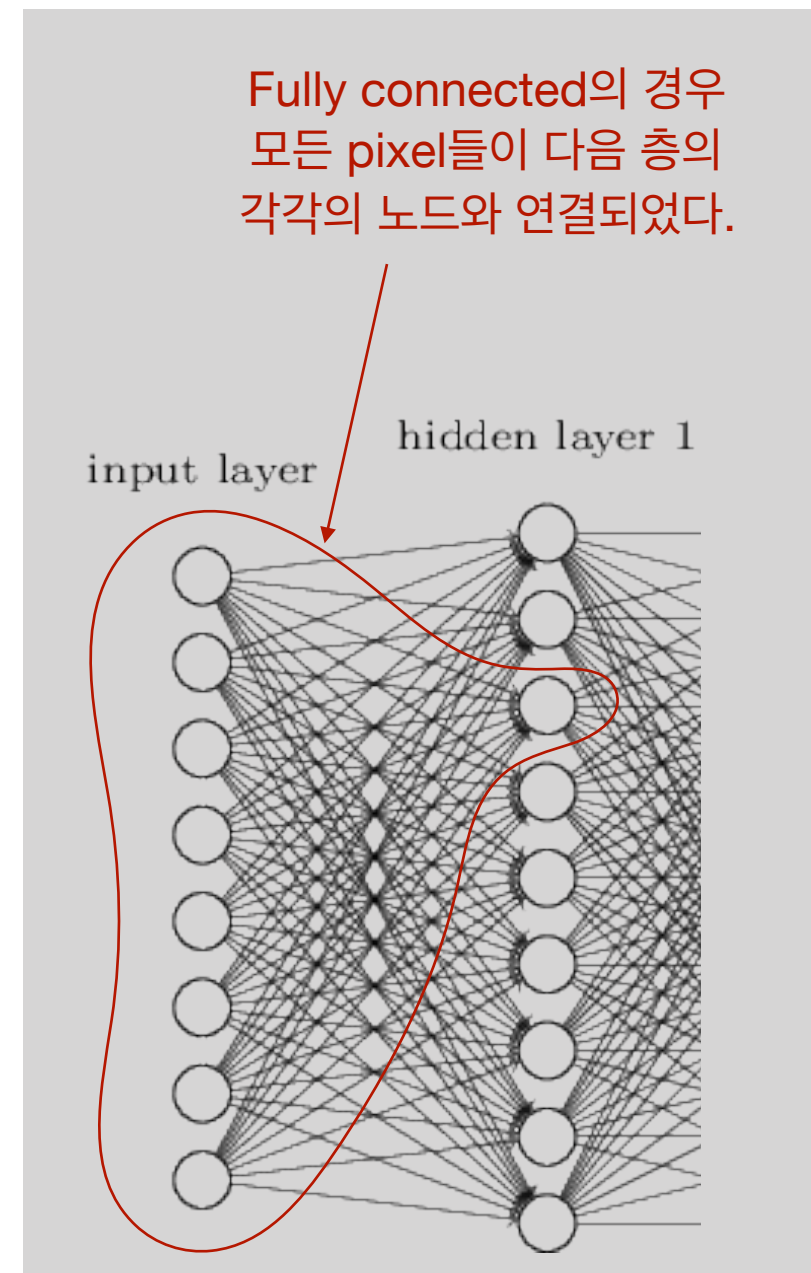


3 채널 이미지의 경우

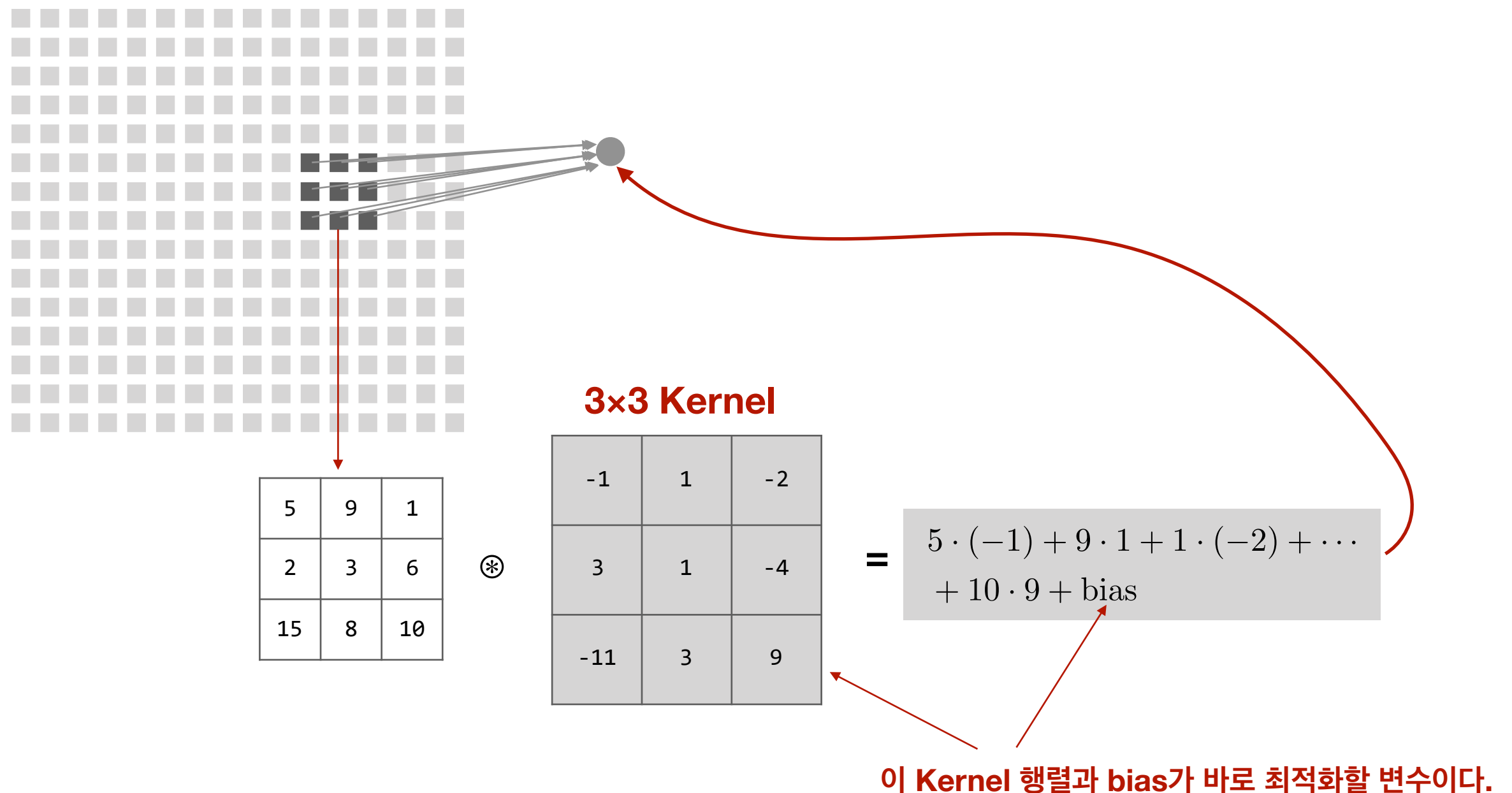
- 모든 픽셀들을 다음 layer의 각 노드에 연결하는 대신 “작은 지역적인 영역 (local receptive field)”만을 다음 층의 한 노드에 연결한다.



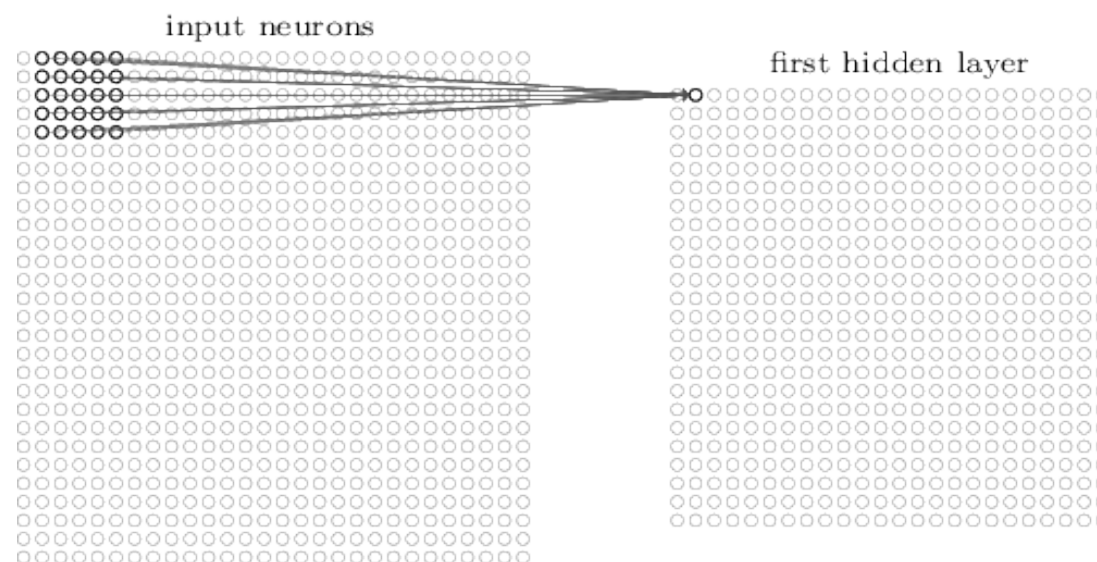
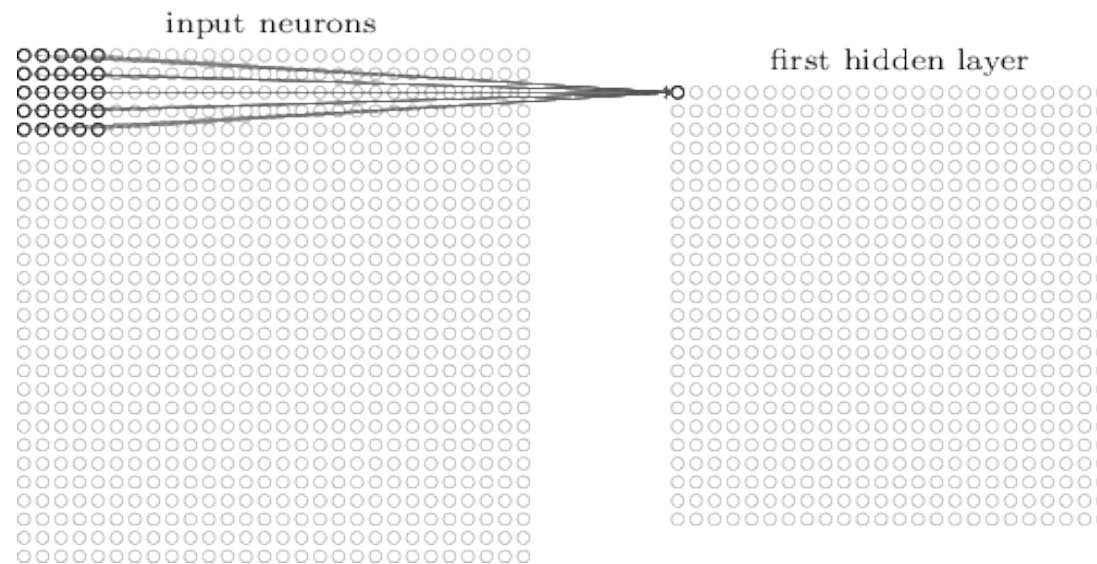
이 노드에는 빨간 9개의 픽셀만이 연결된다. 즉 이 노드는 9개의 픽셀로 구성된 국부적인 한 영역의 특징을 추출하는 역할을 한다.
이 예에서는 3x3 크기의 영역을 사용하였다.
보통 3x3 혹은 5x5 크기가 주로 사용된다.



- “작은 지역적인 영역(local receptive field)”의 특징을 추출하기 위해서 하나의 3×3 가중치 행렬을 사용한다. 이 행렬을 보통 **커널(kernel)**이라고 부른다.
- 이 행렬은 모든 receptive field에 대해서 **공유(shared)**된다.

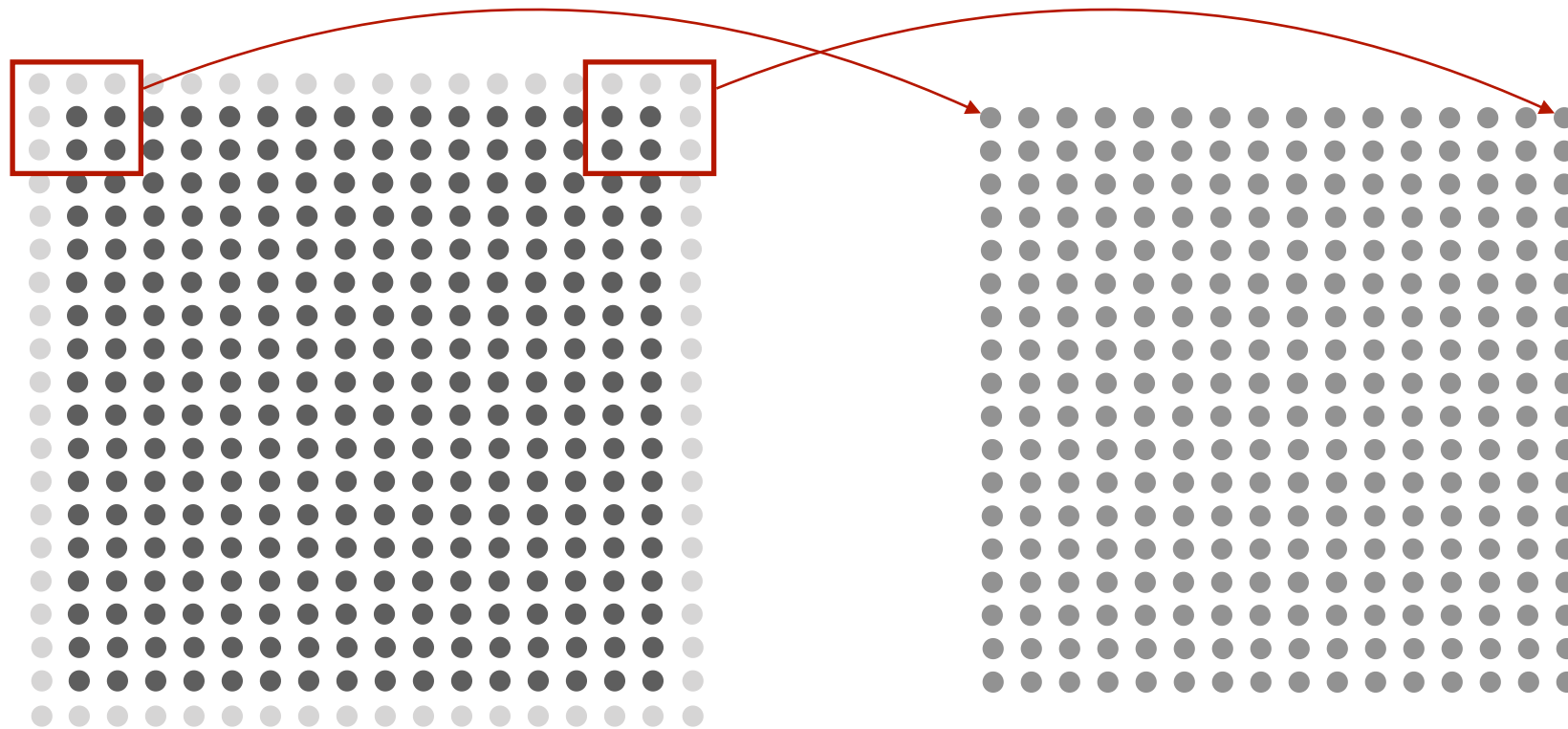


- **Receptive field를 슬라이딩 시키면서 공유된 kernel 행렬을 사용해 계산한다.** 아래의 그림은 커널의 크기가 5x5인 경우이다.



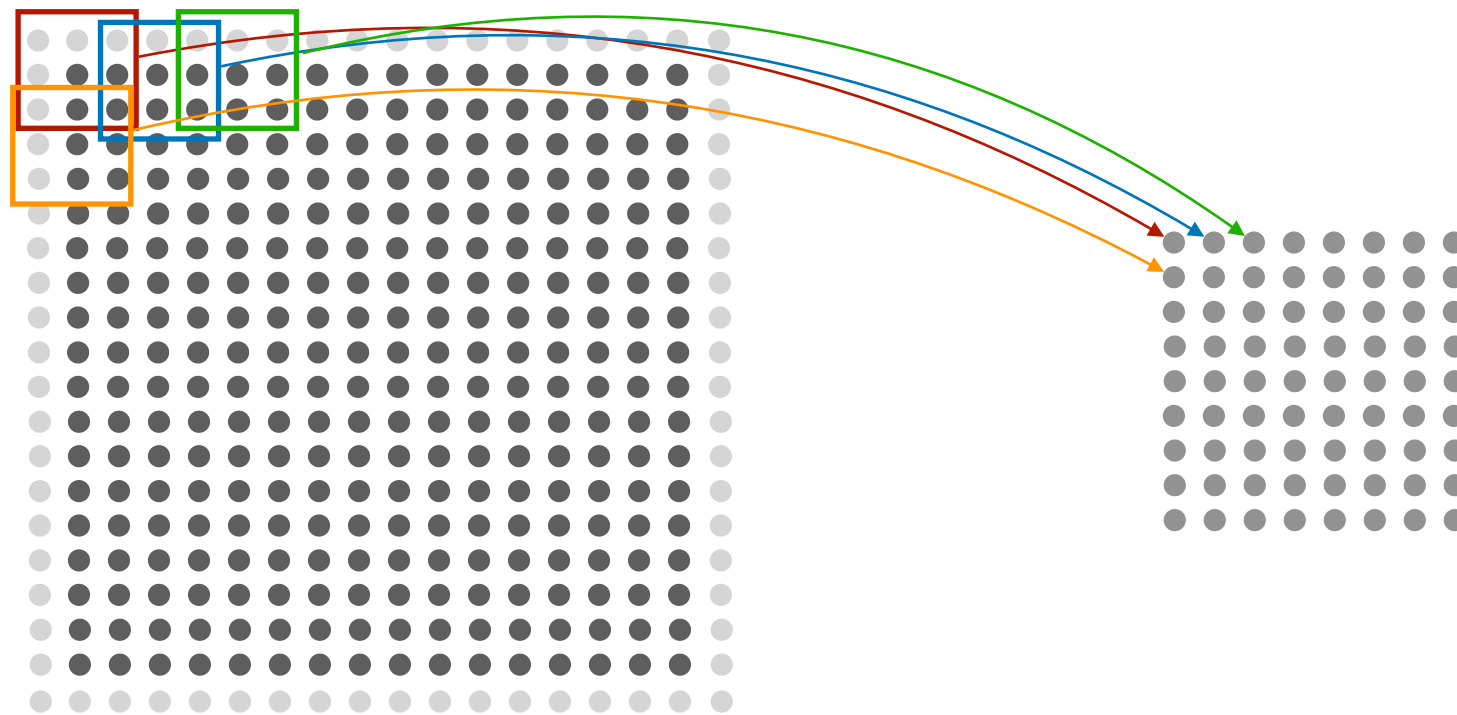
원래 이미지가 $N \times M$ 크기이면
 $(N-4) \times (M-4)$ 크기로 줄어든다.
만약 커널이 3×3 이면 $(N-2) \times (M-2)$
로 줄어든다.
이것은 약간 성가시다.

- 이미지의 상하좌우에 가상의 픽셀을 추가(padding)하여 크기가 줄어드는 것을 방지한다. 3×3 kernel을 사용한다면 1줄씩, 5×5 커널을 사용한다면 2줄씩 추가한다. 추가된 행과 열의 픽셀값은 보통 0으로 가정하거나 혹은 인접한 실제 픽셀의 값을 사용하기도 한다.

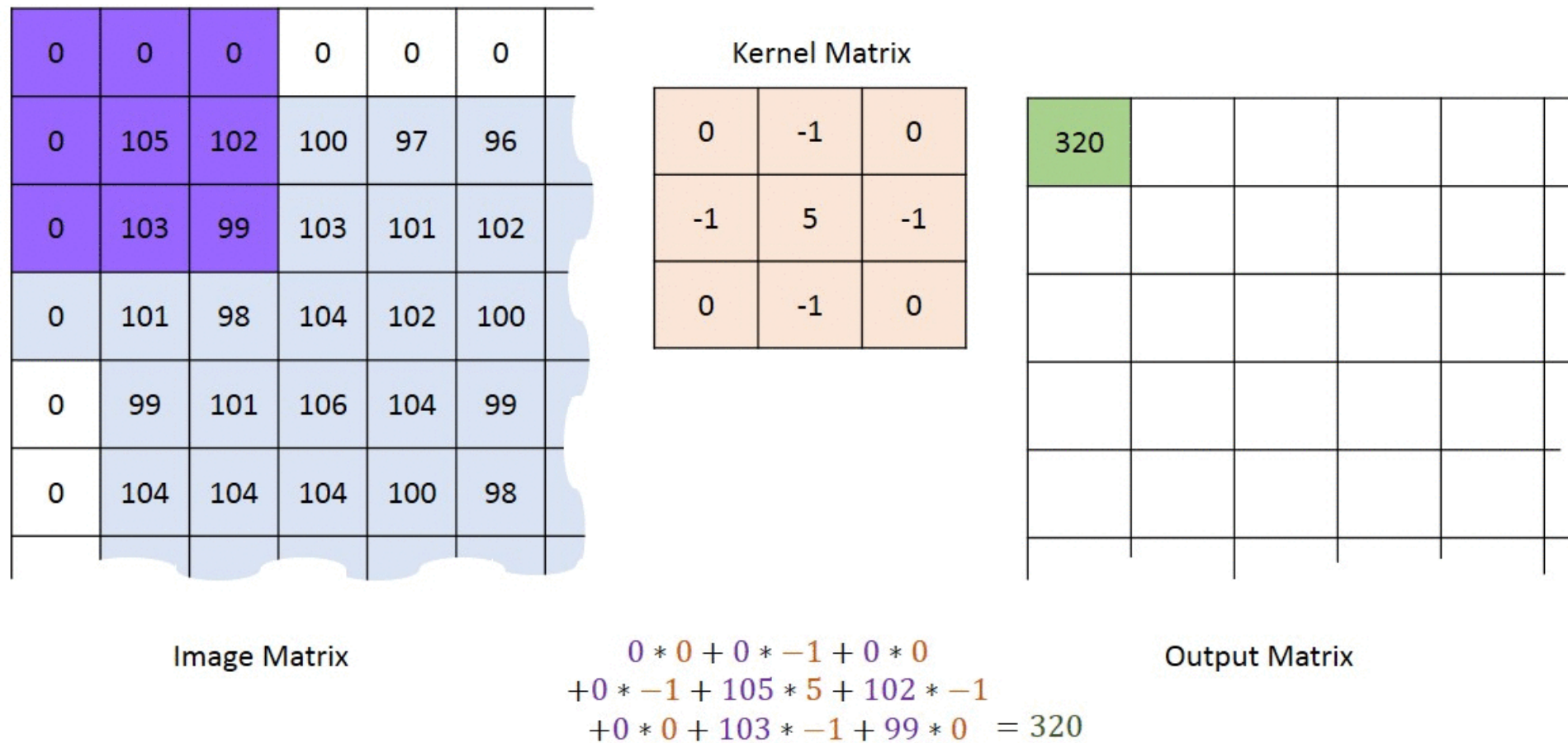


이미지의 크기가 그대로 유지된다.

- 원도우를 슬라이딩할 때 반드시 한 픽셀씩 슬라이딩 해야하는 것은 아니다. 가령 2칸씩 슬라이딩 할 수도 있다. 이 경우 이미지의 크기는 1/2로 줄어든다. 이렇게 슬라이딩하는 단위를 stride라고 부른다. 가로와 세로의 stride 값을 다르게 할 수도 있다.

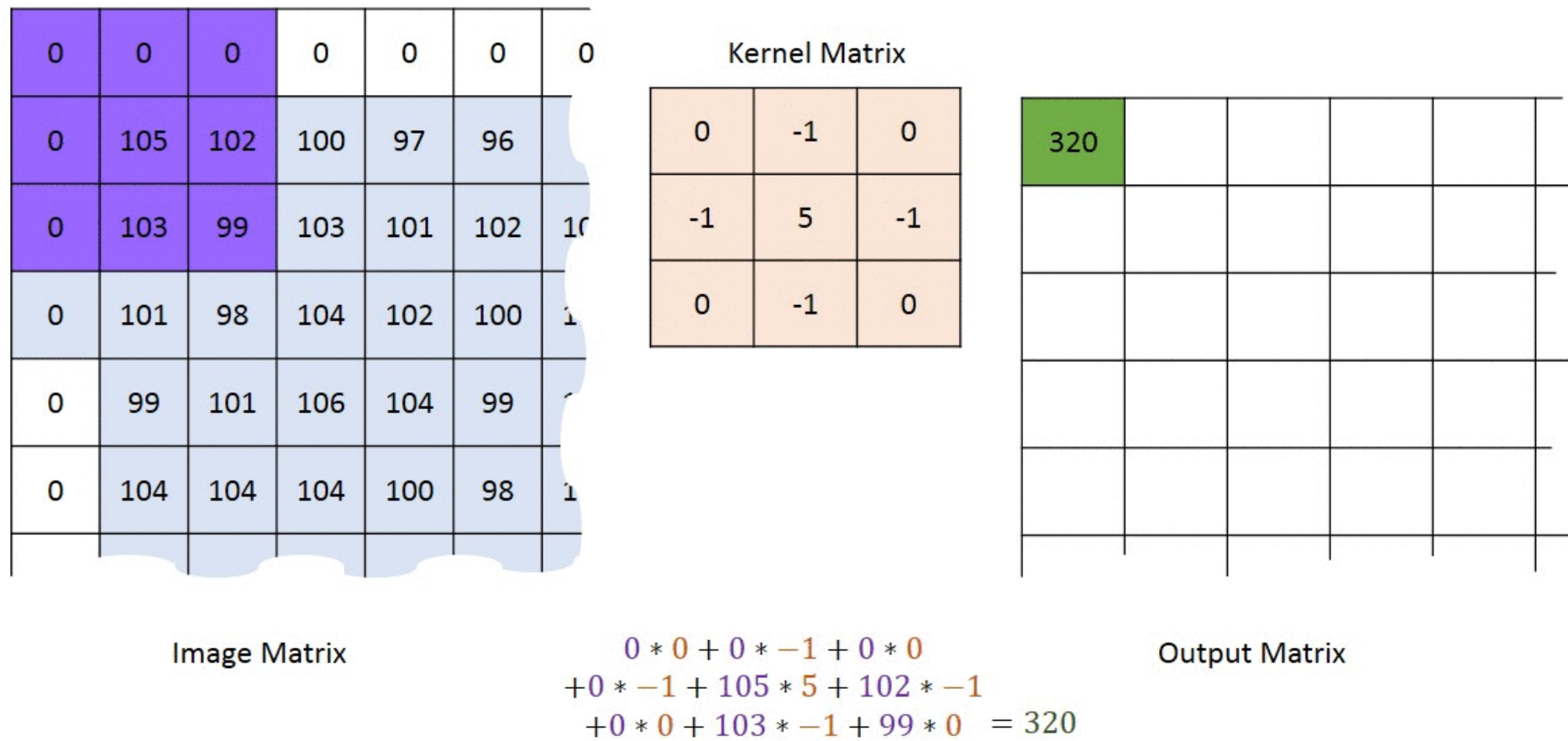


가로/세로 stride 값이 2라면 이미지의 크기가 1/2로 줄어든다.



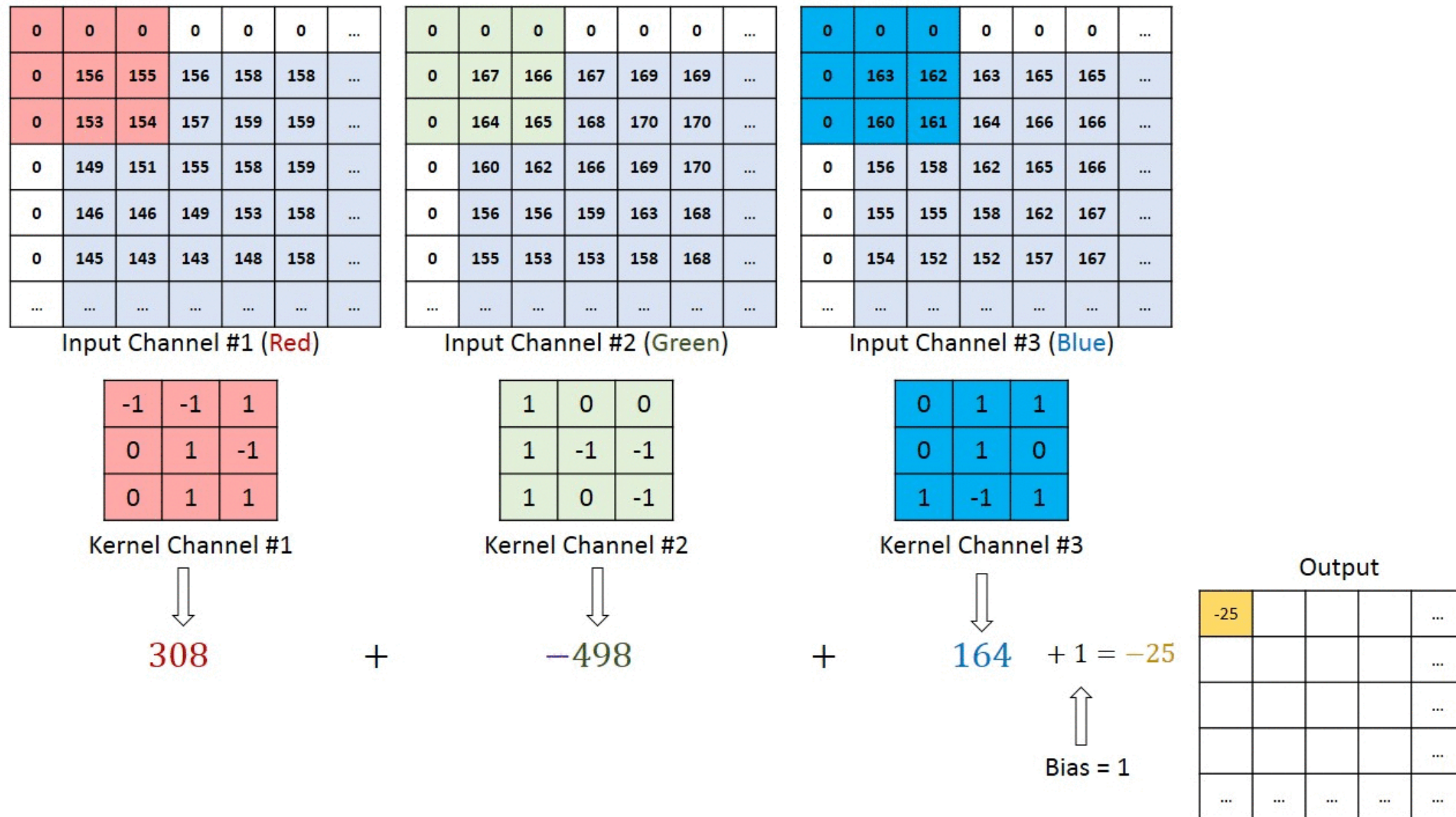
Convolution with horizontal and
vertical strides = 1

Download and watch [this](#) GIF animation



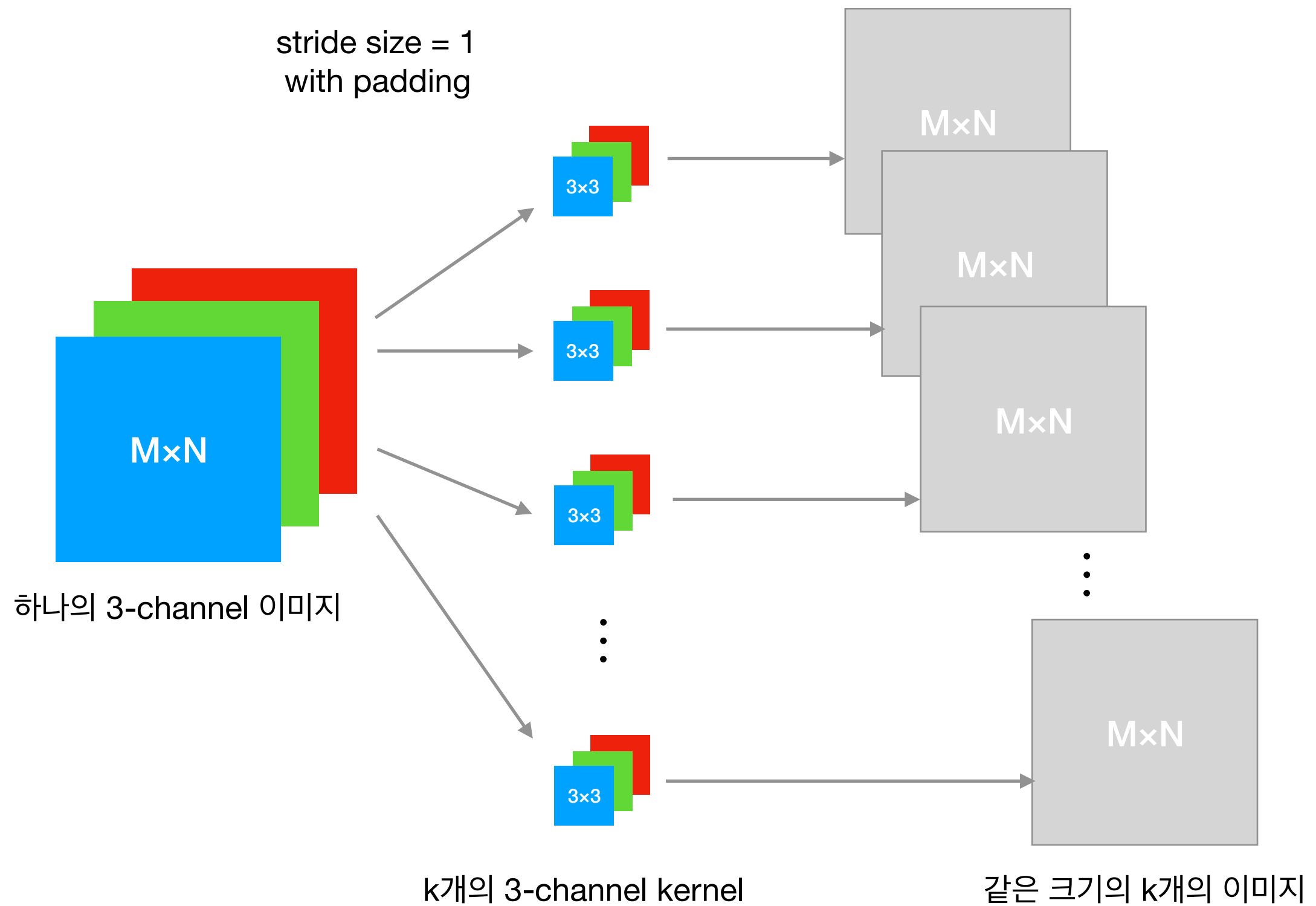
Convolution with horizontal and vertical strides = 2

Download and watch [this](#) GIF animation

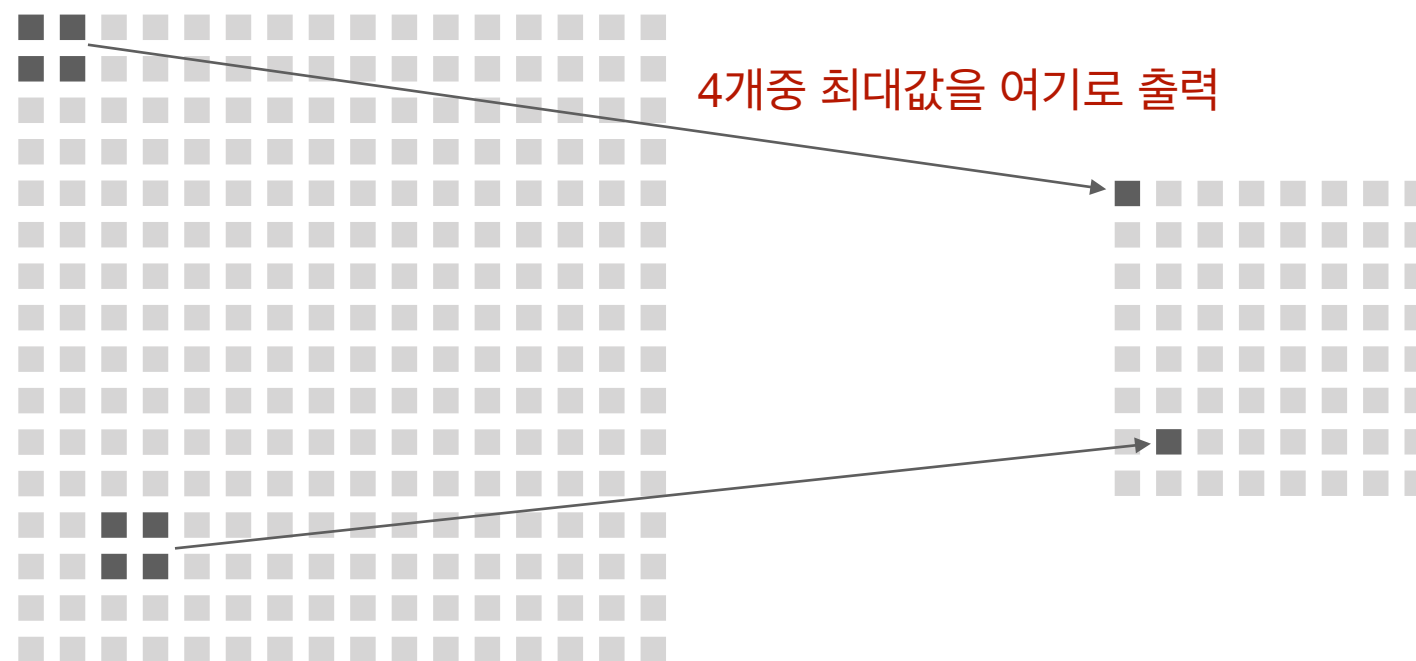


Download and watch [this](#) GIF animation

Single Convolution Layer

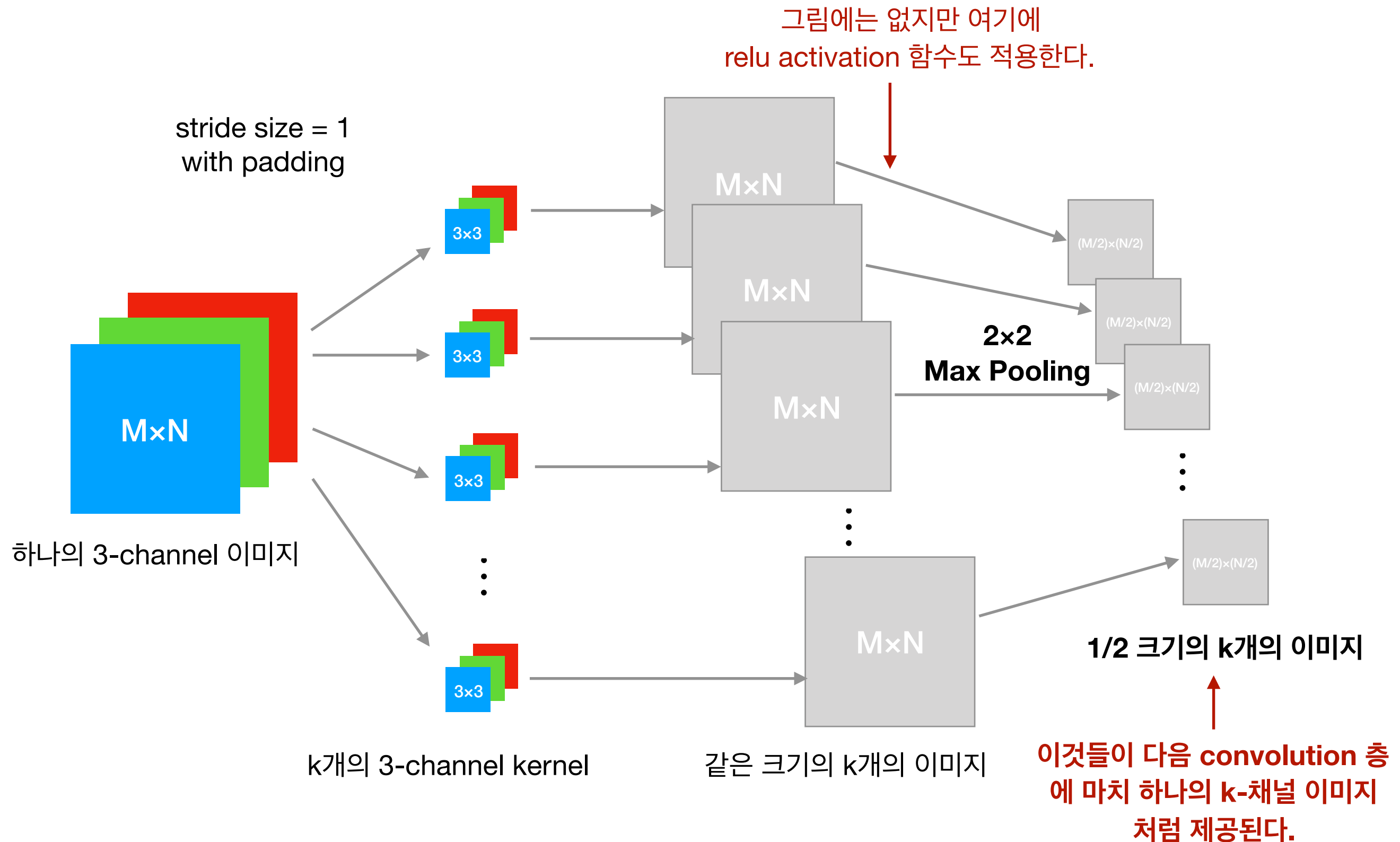


- 일반적으로 convolution layer에 뒤이어 pooling layer를 적용한다.
- Pooling layer의 역할은 convolution layer의 출력을 압축(혹은 단순화)하는 것이다.
- Pooling에는 max-pooling, L2-pooling 등이 있으며, **max-pooling**이 가장 흔하게 사용된다.
- 2×2 max pooling에서는 2×2 영역의 값 중에서 최대값을 선택한다.

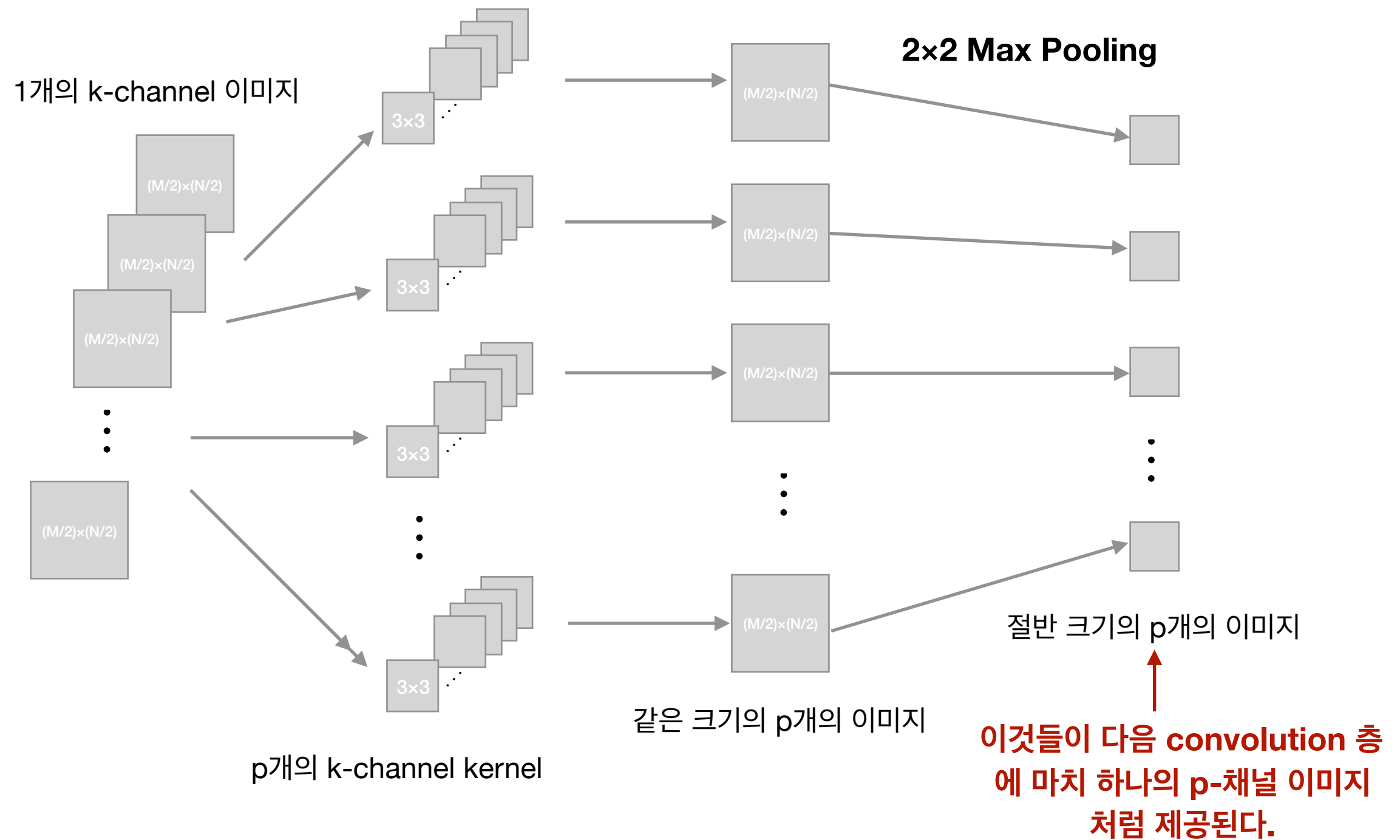


2×2 max pooling을 적용하면 이미지의 크기가 가로/세로 각각 반으로 줄어든다.

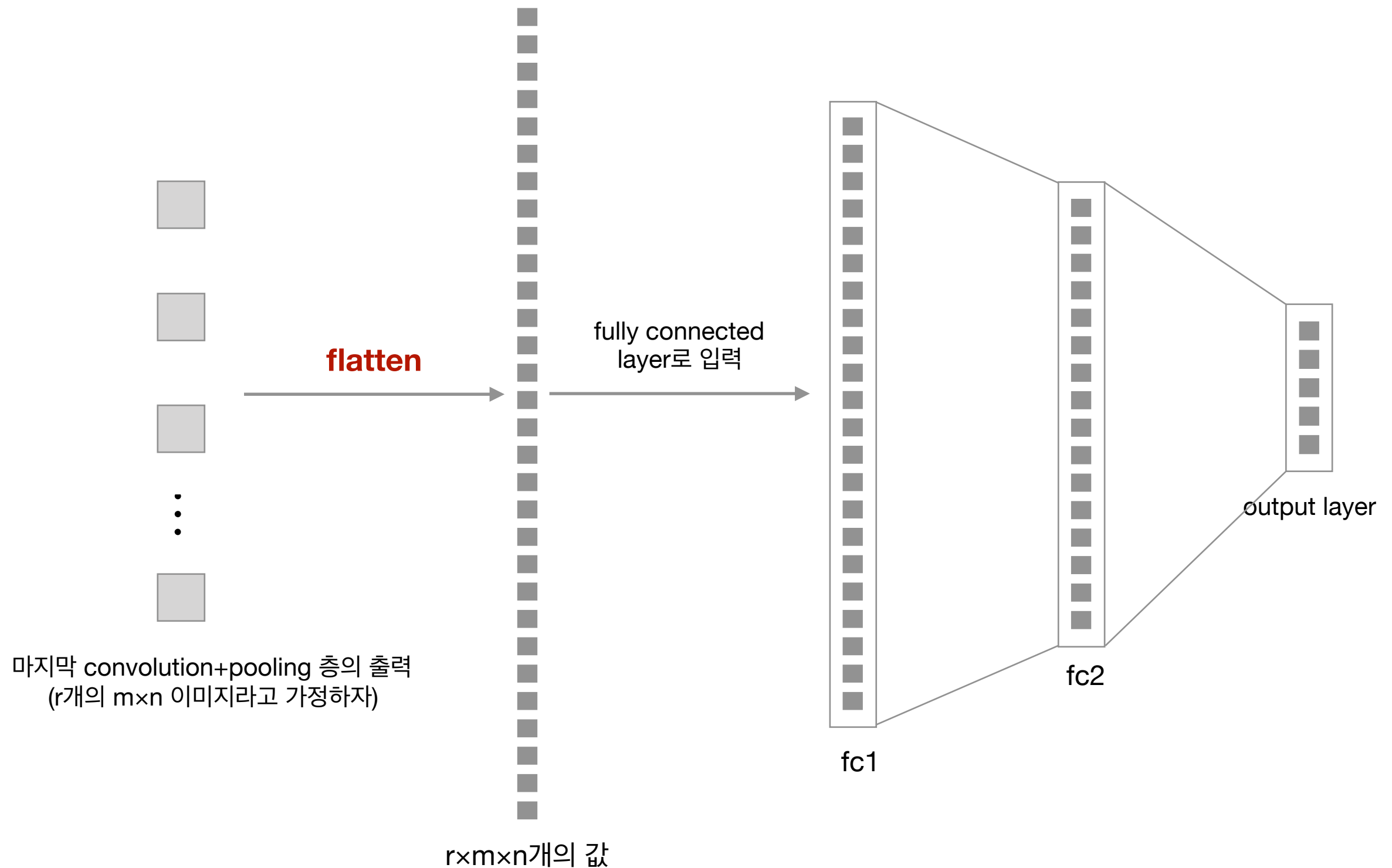
Convolution + Max Pooling Layer

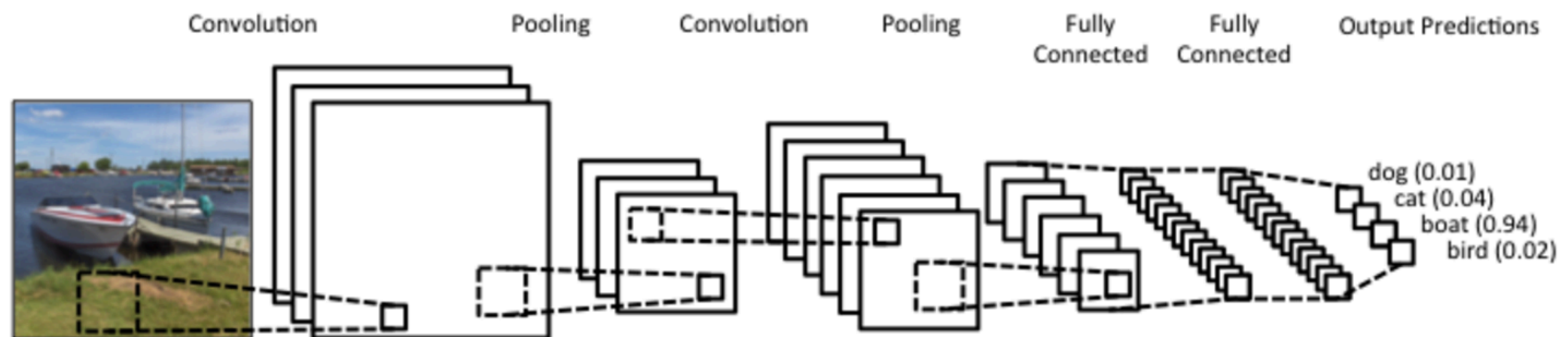


Next (Convolution + Pooling) Layer

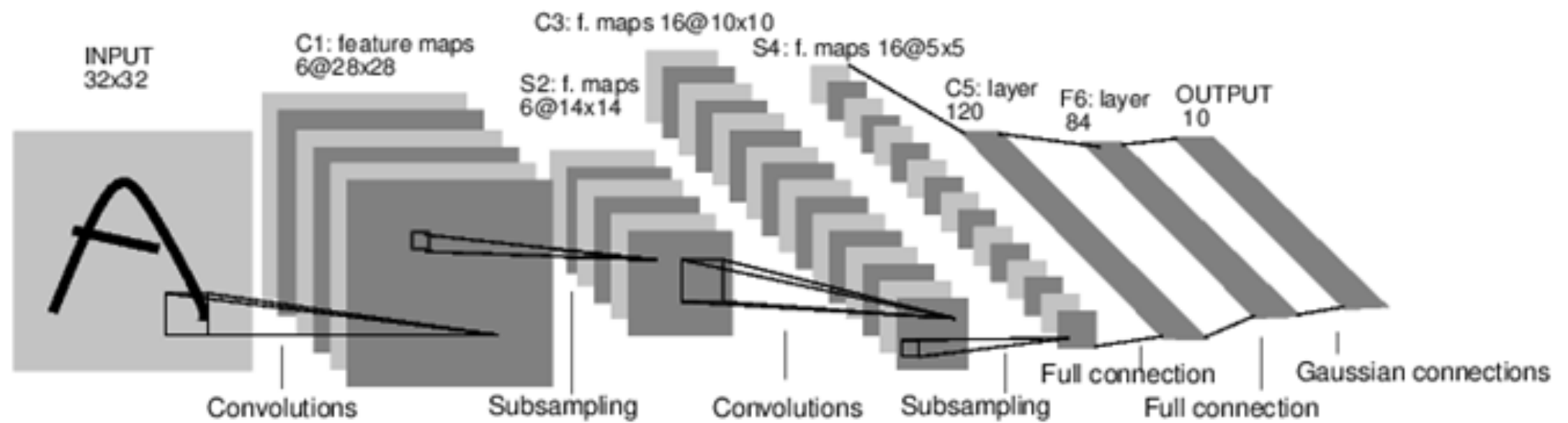


- 일반적으로 convolution + pooling 층들 뒤에 몇 개의 fully connected 층을 배치한다.



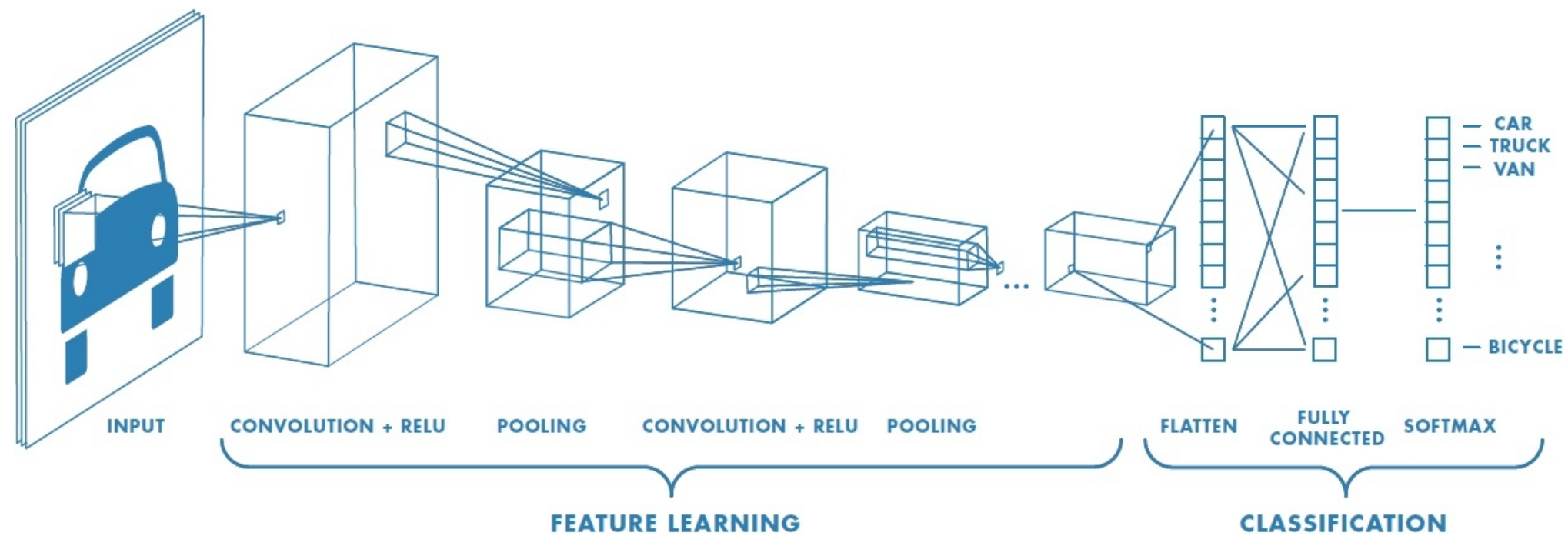


<http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>



A Full Convolutional Neural Network (LeNet)

<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>



<https://www.mathworks.com/discovery/convolutional-neural-network.html>

CNN 구현

- 2 convolution layers with 32 and 64 kernels each
- one fully-connected layer with dropout

```
#  
# 이 부분 이전의 코드는 지난 주와 완전히 동일하다.  
#  
  
import tensorflow as tf  
  
images_batch = tf.placeholder(dtype=tf.float32,  
                              shape=[None, IMG_HEIGHT*IMG_WIDTH*NUM_CHANNEL])
```

-1은 None과 같은 의미

```
x_image = tf.reshape(images_batch, [-1, IMG_HEIGHT, IMG_WIDTH, NUM_CHANNEL])
```

load image 함수에서 이미지를 flatten해두는 바람에 여기에서 다시 3차원 배열로 복구해야 한다.

```
labels_batch = tf.placeholder(dtype=tf.int32, shape=[None, ])
```

First Convolution + Pooling Layer

5×5 크기의 3채널 kernel 32개를 사용한다.

```
W_conv1 = tf.get_variable(name='W_conv1', shape=[5, 5, 3, 32], dtype=tf.float32)
b_conv1 = tf.get_variable(name='b_conv1', shape=[32], dtype=tf.float32)
h_conv1 = tf.nn.relu(
    tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1], padding='SAME')
    + b_conv1)
```


가로, 세로 stride는 1이다.

입력인 x_image의 dimension은 [batch_size, image_height, image_width, num_channel]로 4차원이다.
strides의 각 원소는 각 차원에 대한 stride 값이므로 두번째와 3번째 원소가 이미지의 가로/세로에 해당한다.

```
h_pool1 = tf.nn.max_pool(h_conv1,
    ksize=[1, 2, 2, 1],
    strides=[1, 2, 2, 1], padding='SAME')
```


Second Convolution + Pooling Layer

5x5 크기의 32채널 kernel 64개를 사용한다.



```
W_conv2 = tf.get_variable(name='W_conv2', shape=[5, 5, 32, 64], dtype=tf.float32)
b_conv2 = tf.get_variable(name='b_conv2', shape=[64], dtype=tf.float32)
h_conv1 = tf.nn.relu( tf.nn.conv2d(h_pool1,
                                   W_conv2, strides=[1, 1, 1, 1], padding='SAME') + b_conv2)
h_pool2 = tf.nn.max_pool( h_conv2, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
```

max pooling을 2번 지나면서 이미지 크기가 1/4로 줄었고,
마지막 convolution layer에서 64개의 kernel을 사용했으므로
총 값의 개수는 이렇게 된다.

FC 층에는 1024개의 노드를 사용했다.

$w1 = \text{tf.get_variable}("w1", [\text{IMG_HEIGHT} // 4 * \text{IMG_WIDTH} // 4 * 64, 1024])$

$b1 = \text{tf.get_variable}("b1", [1024])$

$h_pool2_flat = \text{tf.reshape}(h_pool2, [-1, \text{IMG_HEIGHT} // 4 * \text{IMG_WIDTH} // 4 * 64])$

FC 층에 입력하기 위해서 flatten해 준다.

$fc1 = \text{tf.nn.relu}(\text{tf.matmul}(h_pool2_flat, w1) + b1)$

- ▶ Dropout은 overfitting을 억제하기 위한 대표적인 기법들 중 하나이다. 이것에 대해서는 다음 주에 다룰 예정이다.
- ▶ 일단은 그냥 넘어가자.

```
keep_prob = tf.placeholder(tf.float32)
```

```
h_fc1_drop = tf.nn.dropout(fc1, keep_prob)
```

```
w2 = tf.get_variable("w2", [1024, NUM_CLASS])
b2 = tf.get_variable("b2", [NUM_CLASS])

y_pred = tf.matmul(h_fc1_drop, w2) + b2

class_prediction = tf.argmax(y_pred, 1, output_type=tf.int32)
correct_prediction = tf.equal(class_prediction, labels_batch)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
loss = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y_pred,  
                                                       labels=labels_batch)  
loss_mean = tf.reduce_mean(loss)  
train_op = tf.train.AdamOptimizer().minimize(loss)
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

iter_ = train_data_iterator()
for step in range(500):

    images_batch_val, labels_batch_val = next(iter_)

    accuracy_, _, loss_val = sess.run([accuracy, train_op, loss_mean],
                                       feed_dict={
                                           images_batch:images_batch_val,
                                           labels_batch:labels_batch_val,
                                           keep_prob: 0.5
                                       })
    print('Iteration {}: {}, {}'.format(step, accuracy_, loss_val))

print('Training Finished....')
```

```
print('Test begins...')

TEST_BSIZE = 50
for i in range(int(len(test_features)/TEST_BSIZE)):

    images_batch_val = test_features[i*TEST_BSIZE:(i+1)*TEST_BSIZE]/255.
    labels_batch_val = test_labels[i*TEST_BSIZE:(i+1)*TEST_BSIZE]

    loss_val, accuracy_ = sess.run([loss_mean, accuracy], feed_dict={
        images_batch:images_batch_val,
        labels_batch:labels_batch_val,
        keep_prob: 1.0
    })
    print('ACC = {}, LOSS = {}'.format(accuracy_, loss_val))
```

- 어떤 테스트 데이터가 잘못 분류되었는지, 그리고 무엇으로 잘못 분류되었는지 알고싶다.
또한 모든 클래스 쌍 (A, B)에 대해서 A에 속한 이미지가 B로 잘못 분류된 퍼센티지를 구하여 하나의 행렬로 출력 하려면 어떻게 해야 할까?