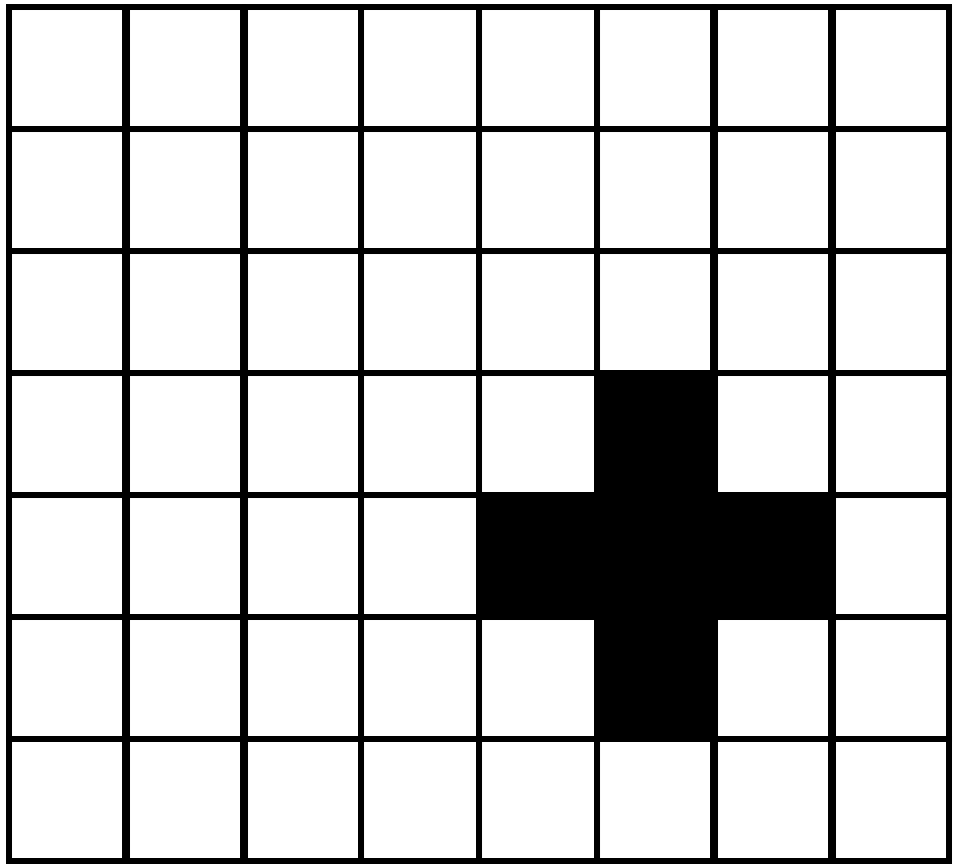
CNN
(Convolution Neural Network)

복습 : 필터(Filter)

- 사진에서 + 를 찾을려면 어떤 필터를 쓰면 될까?

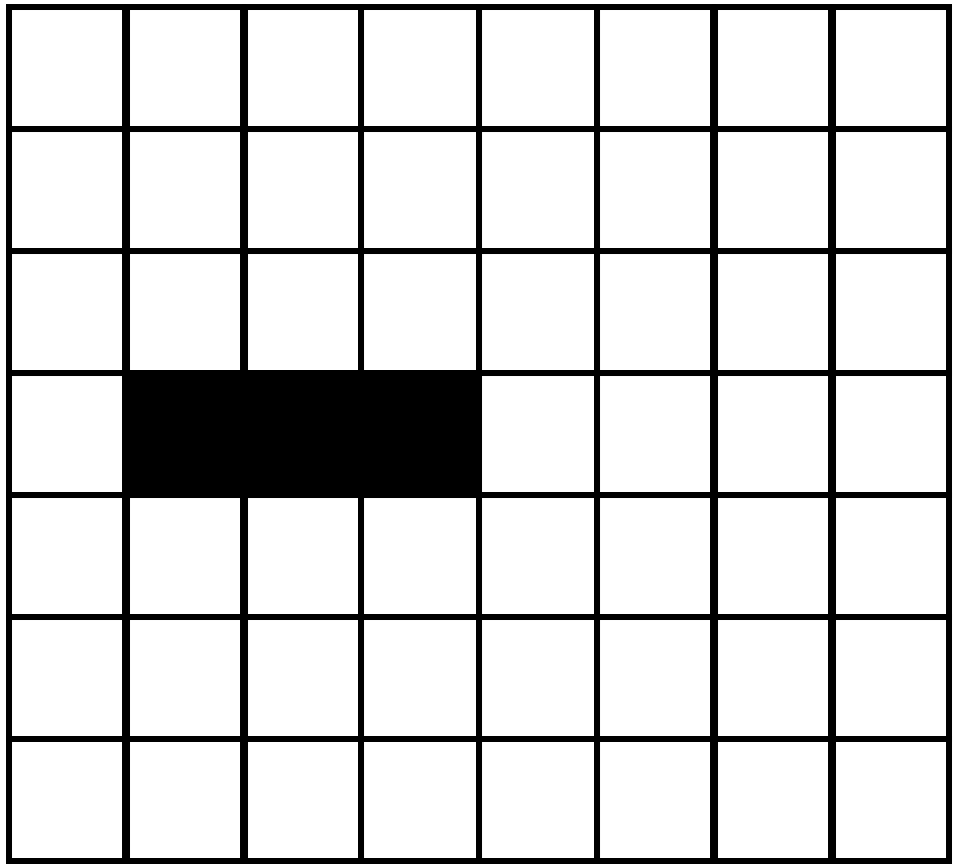


-1	1	1
1	1	1
-1	1	-1

즉 원하는부분은 1,안쓰는 부분은 0이아닌 -로 가중치.

복습 : 필터(Filter)

- 사진에서 --- 를 찾을려면 어떤 필터를 쓰면 될까?

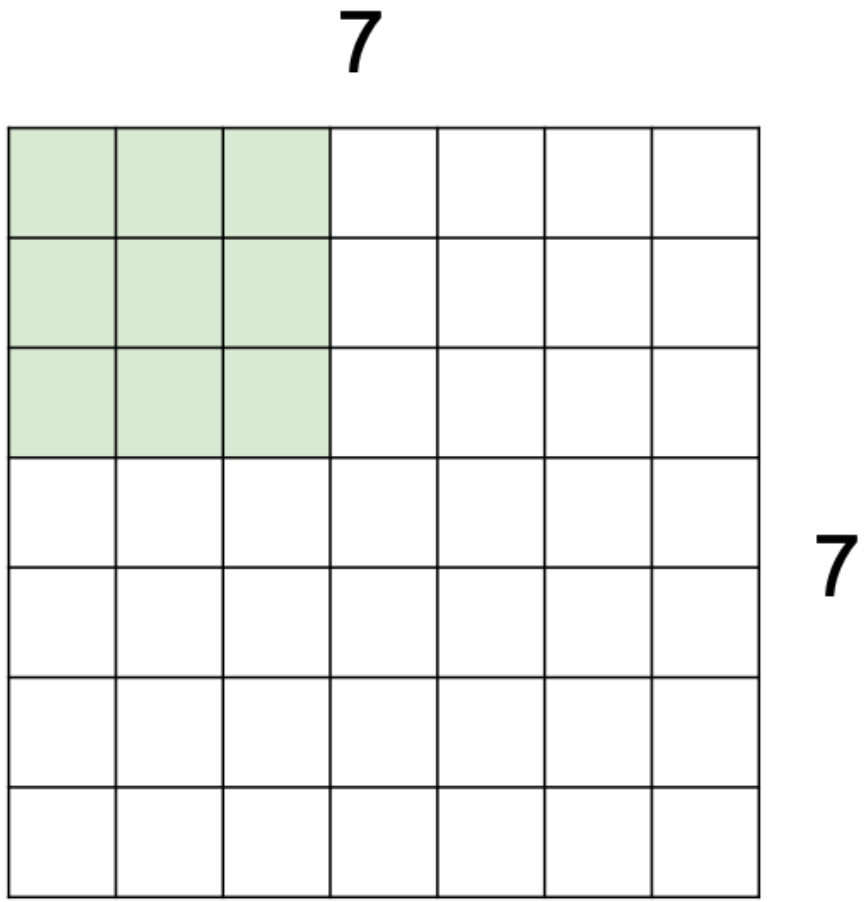


-2	-2	-2
1	1	1
-2	-2	-2

Convolution 연산

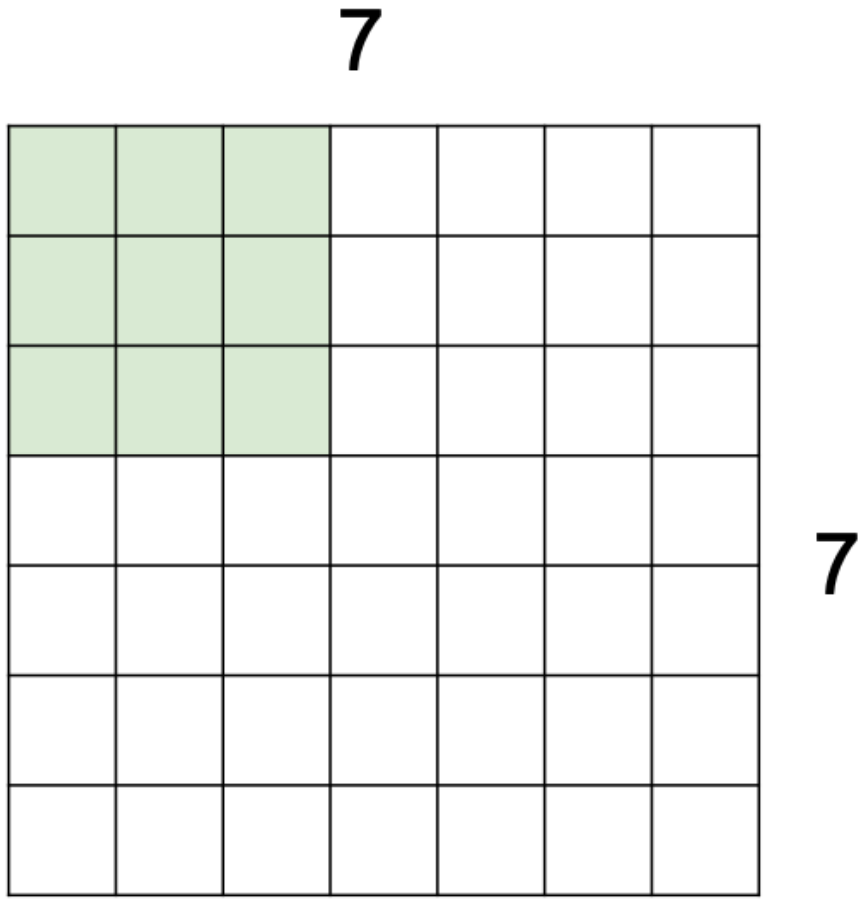
입력 크기와 출력 크기 관점에서

세부 개념도



7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

세부 개념도

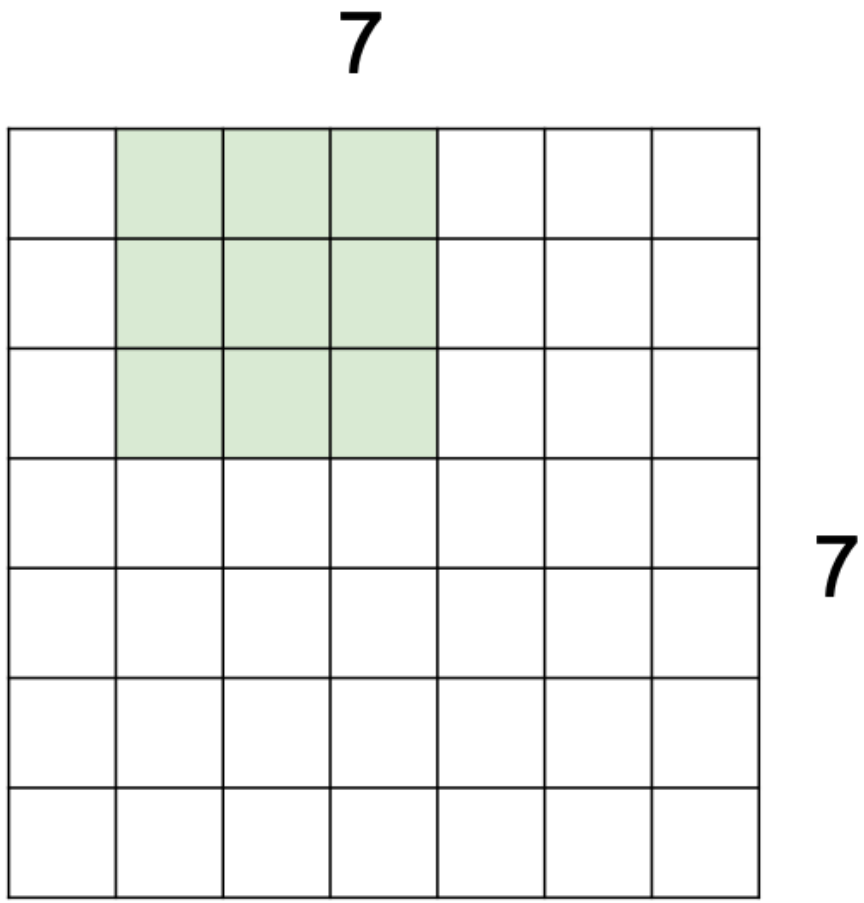


7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

$$w_{11} * p_{x_{i-1},x_{j-1}} + w_{12} * p_{x_i,x_{j-1}} + w_{13} * p_{x_{i+1},x_{j-1}} +$$
$$w_{21} * p_{x_{i-1},x_j} + w_{22} * p_{x_i,x_j} + w_{23} * p_{x_{i+1},x_j} +$$
$$w_{31} * p_{x_{i-1},x_{j+1}} + w_{32} * p_{x_i,x_{j+1}} + w_{33} * p_{x_i,+1\,x_{j+1}}$$

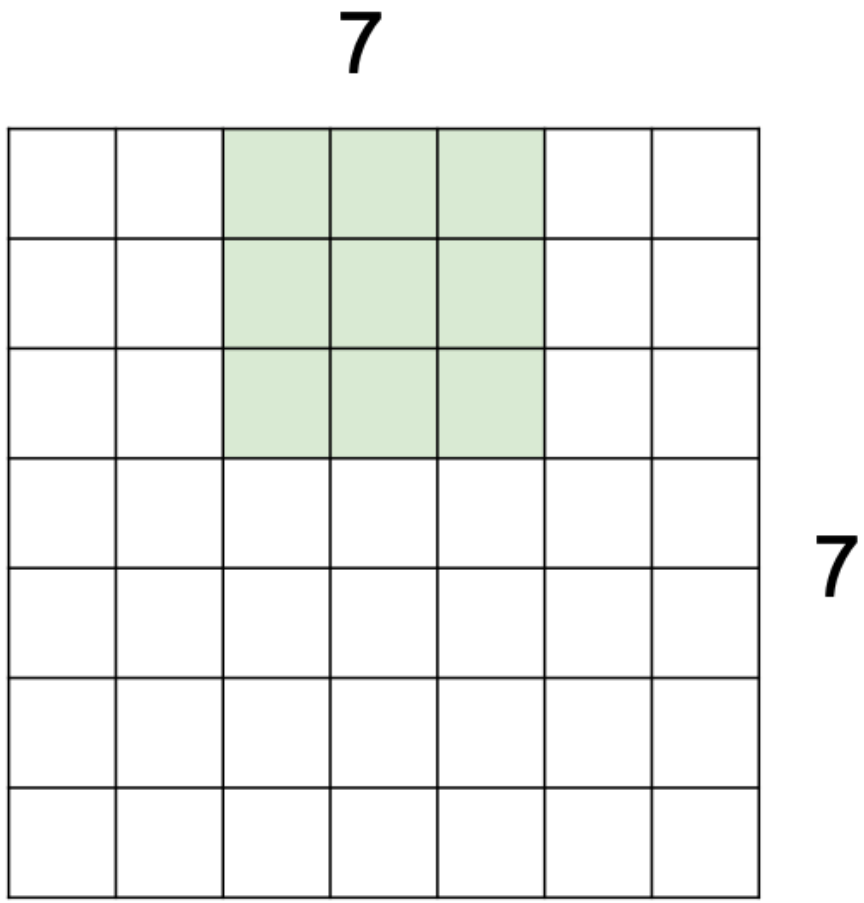
= 새로운 특징점 하나 생성

세부 개념도



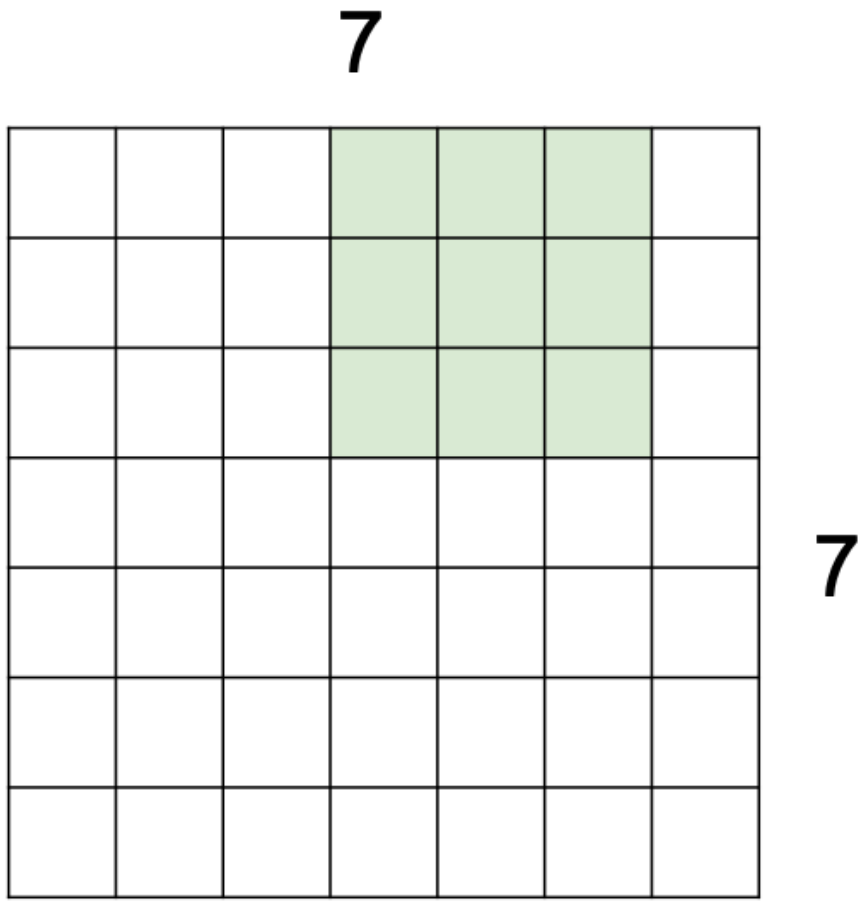
7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

세부 개념도



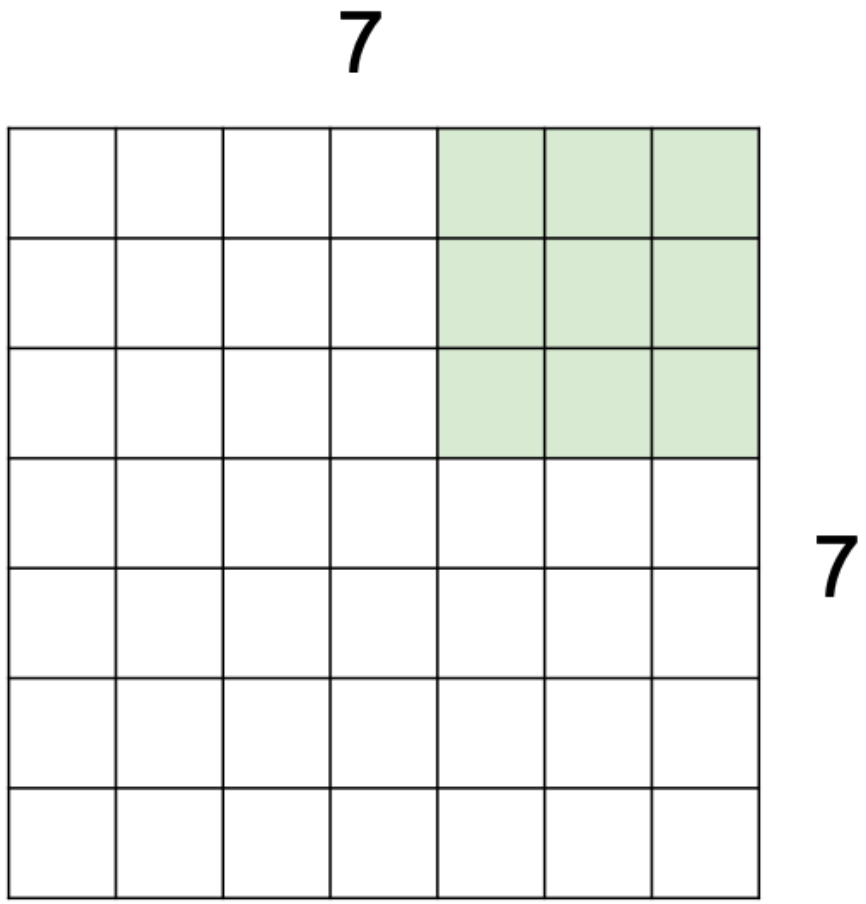
7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

세부 개념도



7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

세부 개념도

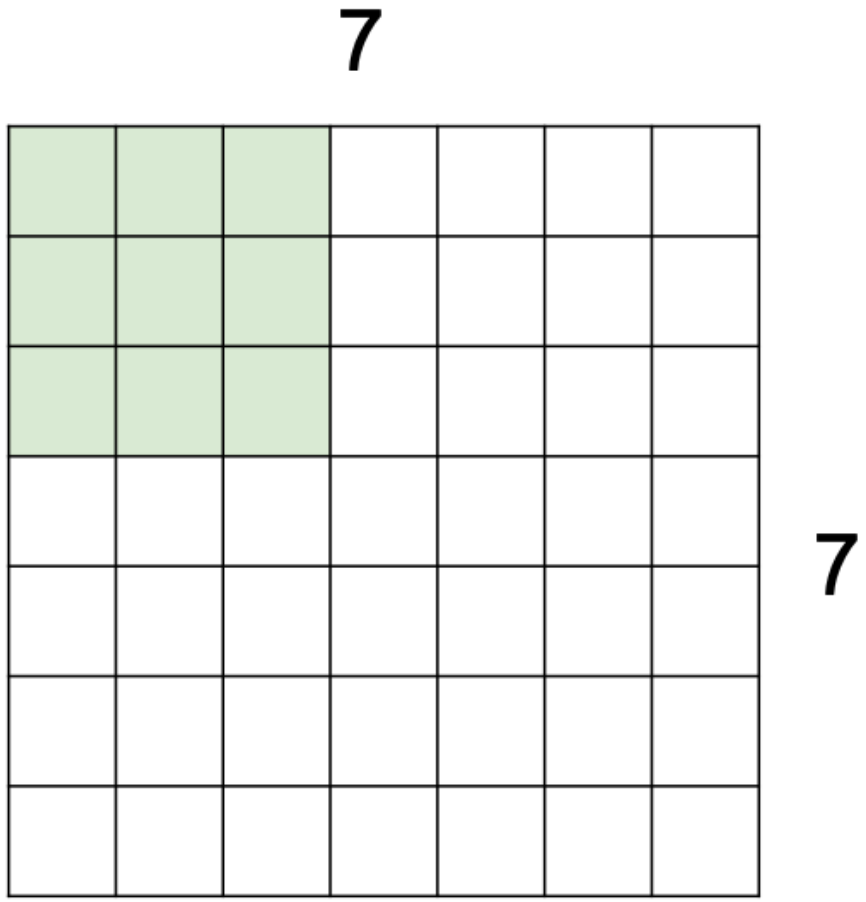


7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

=> 5 x 5의 출력결과 끝까지 가는게 5번

즉 개수는 필터- (필터size-1)이다.

세부 개념도

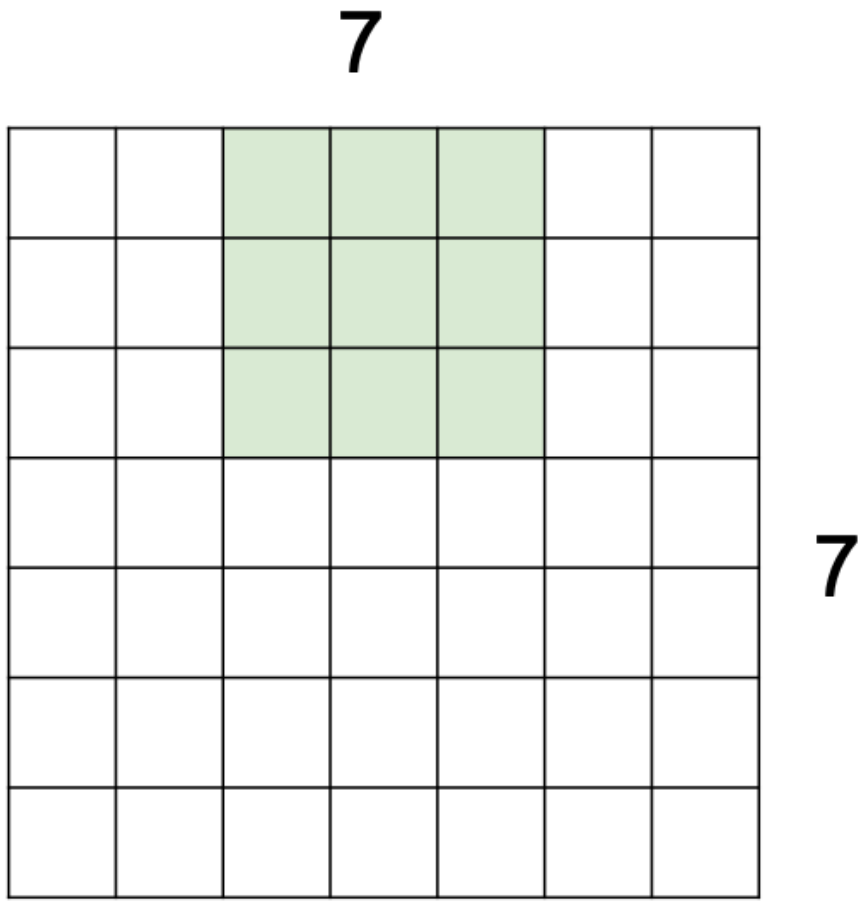


7 x 7로 이루어진 이미지와 3 x 3 필터가 주어진 경우

두 칸씩 이동하면 stride 2

한칸씩 보단 두칸식 움직여 필터를 조금 더 효율적으로도 사용가능.

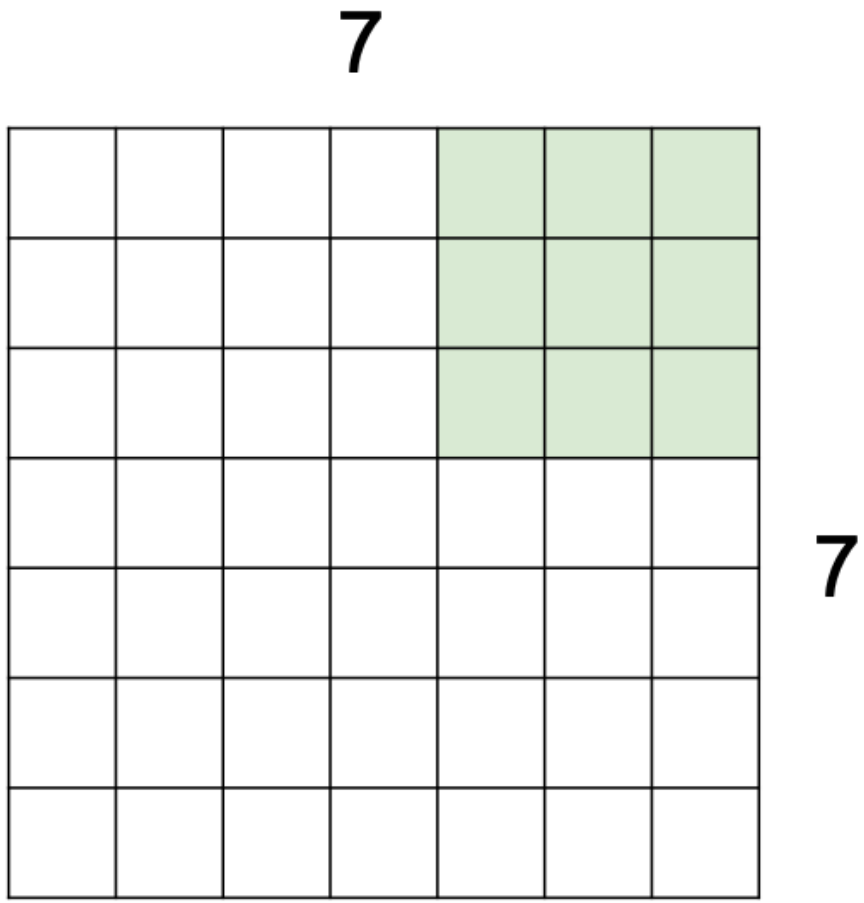
세부 개념도



7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

두 칸씩 이동하면 stride 2

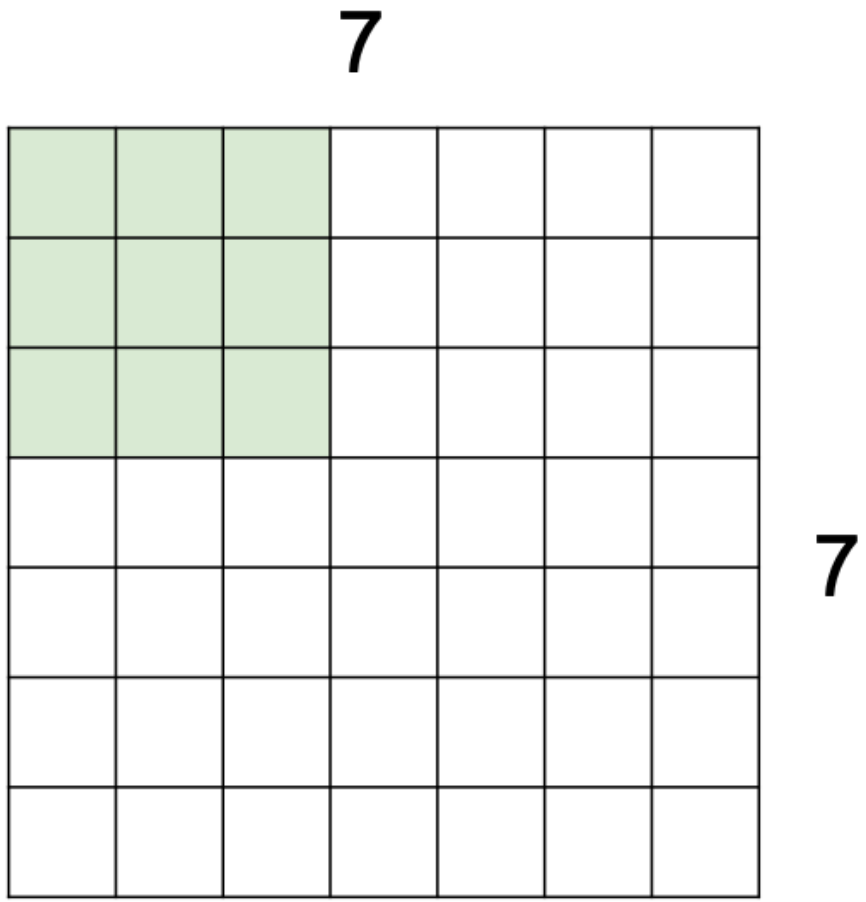
세부 개념도



7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

두 칸씩 이동하면 stride 2

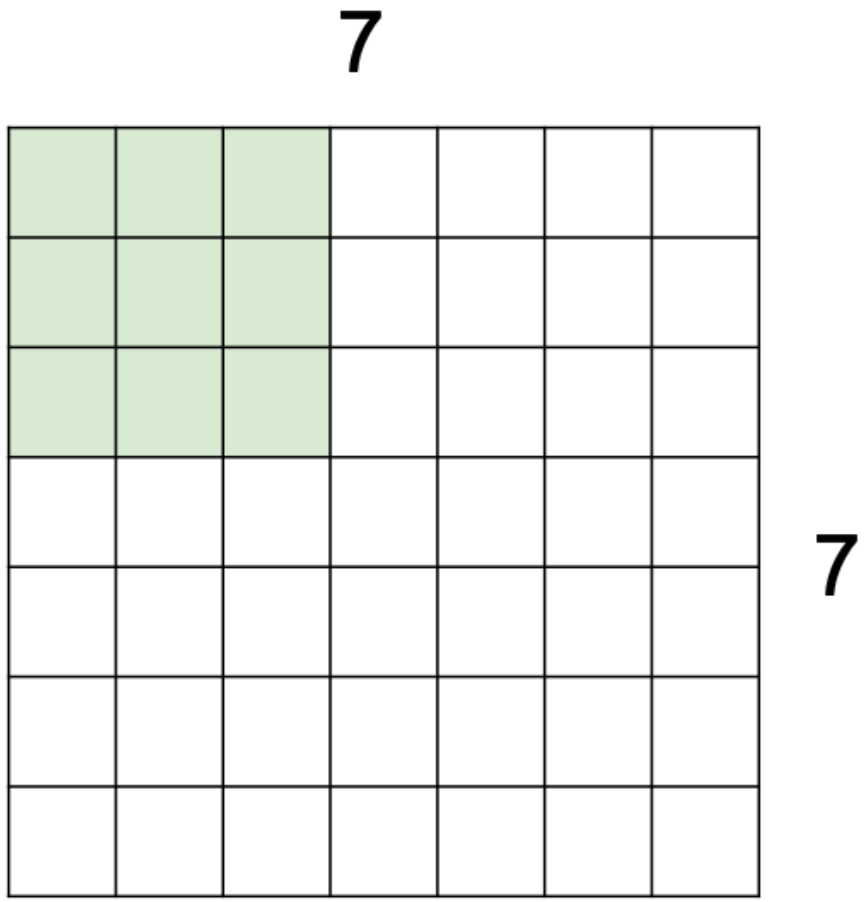
세부 개념도



7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

세 칸씩 이동하면 stride 3

세부 개념도



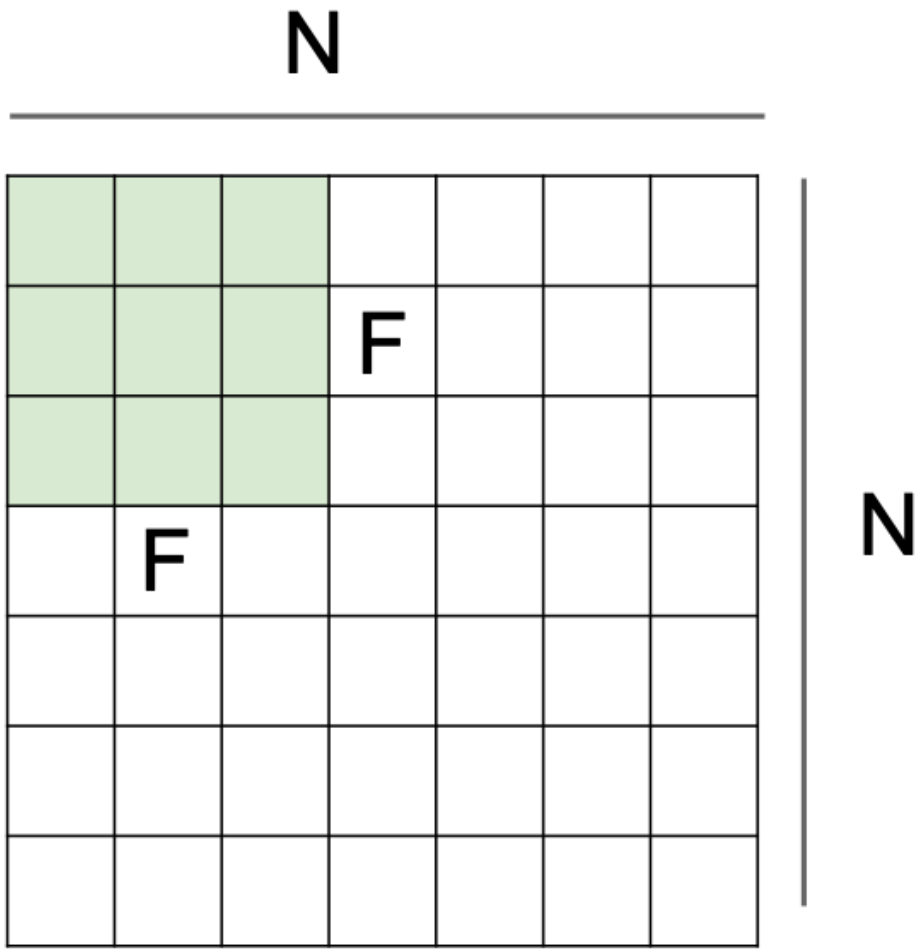
7 x 7로 이루어진 이미지와
3 x 3 필터가 주어진 경우

세 칸씩 이동하면 stride 3

7x7 이미지
3x3 필터에서는
3칸 이동은 불가

이걸 zero padding으로 알아서 맞춰준다.

세부 개념도

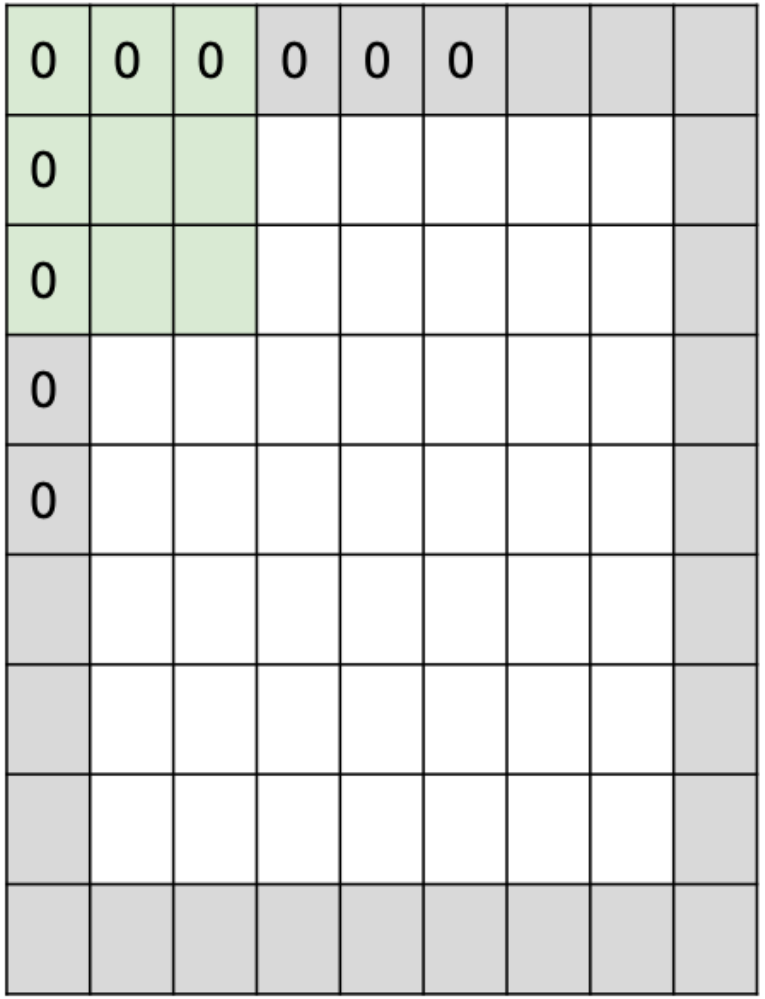


일반식으로 나타내면

Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

동일 크기 만들기 : Zero Padding을 경계선에 적용



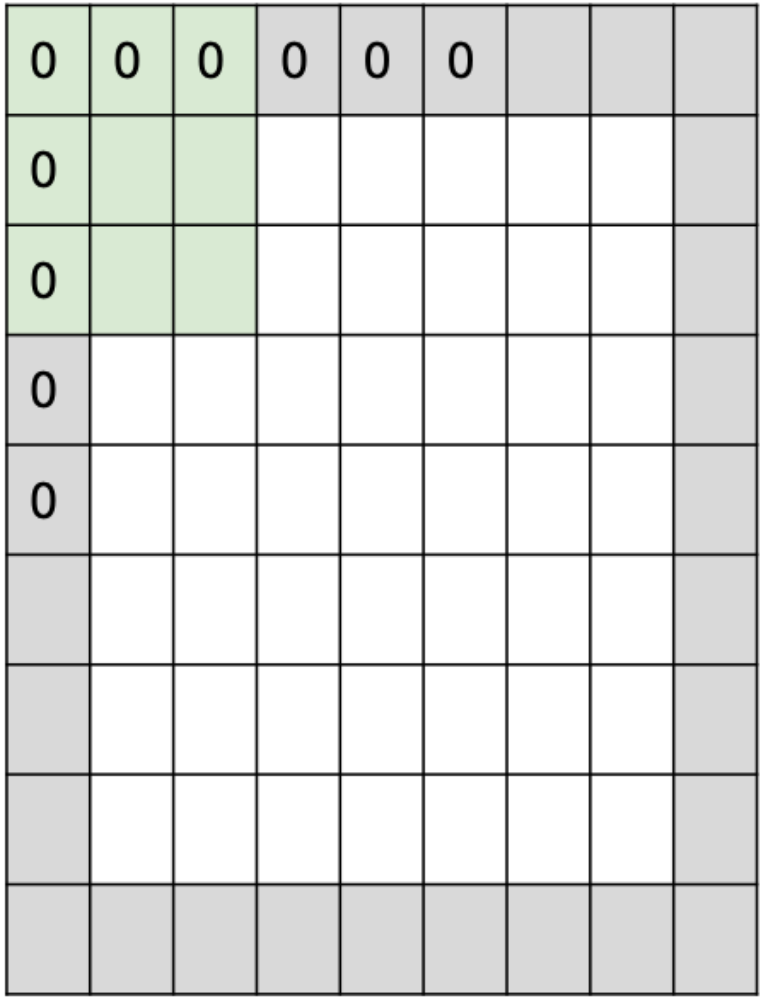
7 x 7로 이루어진 이미지에
Zero padding 크기 1 적용

3 x 3 필터가 주어진 경우

한 칸씩 이동하면 (stride 1)

(recall:)
(N - F) / stride + 1

동일 크기 만들기 : Zero Padding을 경계선에 적용



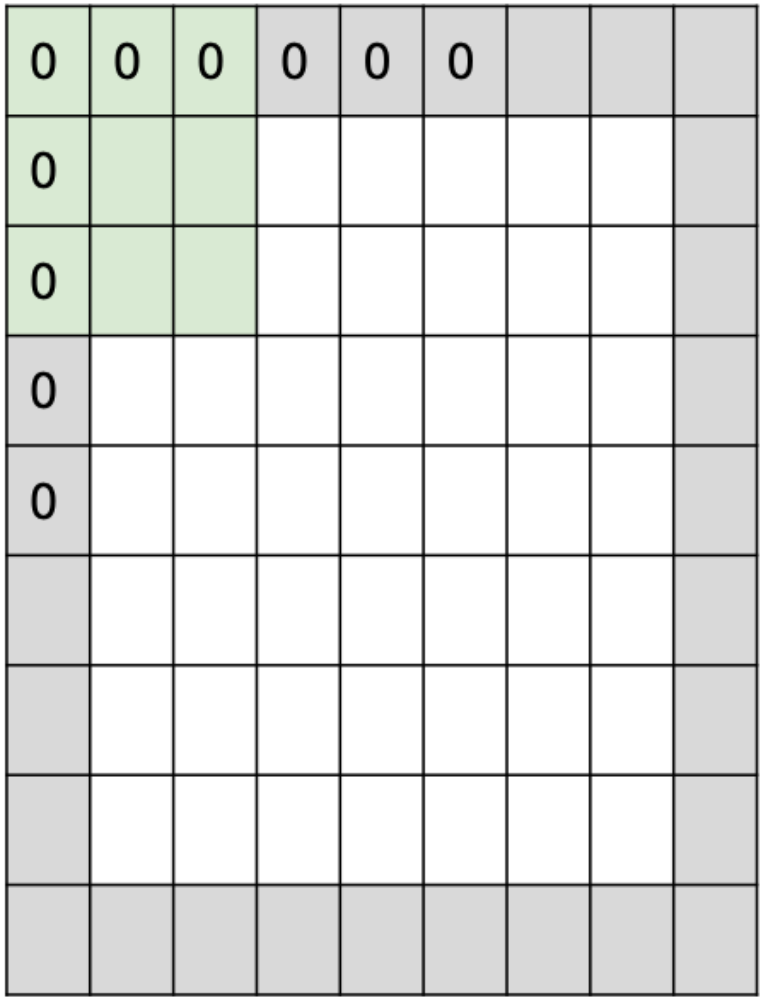
7 x 7로 이루어진 이미지에
Zero padding 크기 1 적용

3 x 3 필터가 주어진 경우

한 칸씩 이동하면 (stride 1)

최종적으로 7 x 7

동일 크기 만들기 : Zero Padding을 경계선에 적용



필터가 F x F 인 경우, padding 크기 (F – 1)/2 설정 시,
입력과 동일한 크기의 특징 생성

F = 3 이면 padding 크기 1
F = 5 이면 padding 크기 2
F = 7 이면 padding 크기 3

Convolutional Neural Networks

Convolution Neural Network

- 이미지에서 특징을 찾기 위한 기존의 방법 중 하나 (검증된 방법)

- 학습을 통해 데이터에서 목적에 맞는 필터(Filter)를 찾아냄
  - 기존에는 필터를 연구자들이 수학적으로 찾아냄

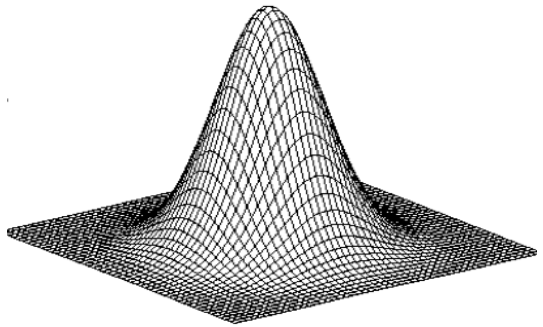
- 특징1 : 물체가 이미지에 어디에 있던지 탐지 가능
- 특징2 : Dense대비 파라미터의 수가 아주 적음
- 특징3 : 해당 데이터셋에만 특화된 특징을 추출 (장점? 단점?)

ex) dense는 input = 입력크기*필터개수
CNN input = 필터크기 * 필터개수

desne는 input size에 비례하지만
이것은 inputsize랑 연관없이 독립적

Convolution Neural Network (CNN)

연구자들이 고안한 필터



Gaussian

$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

하고자하는 데이터셋에 맞는 필터를 써야지 확률 업

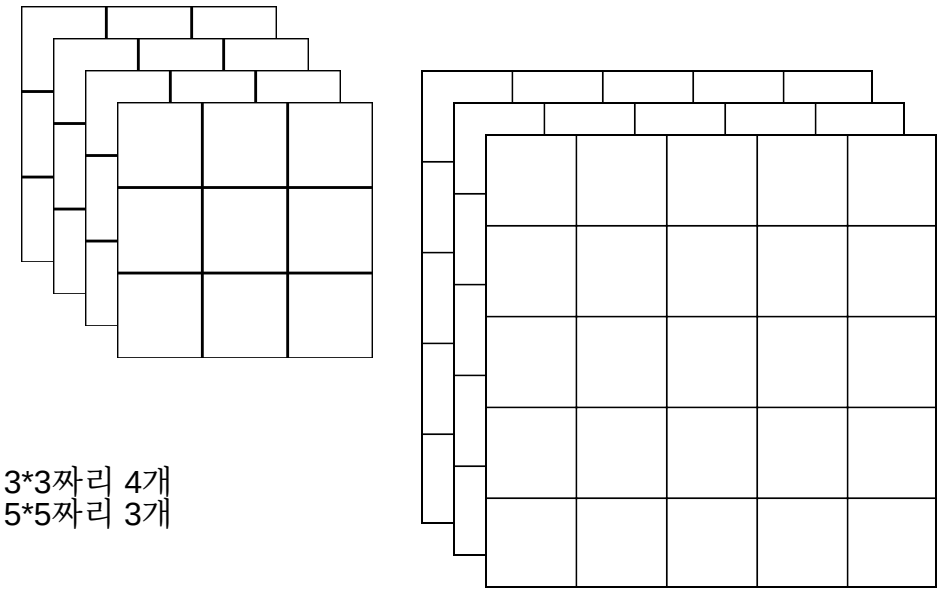
필터를 데이터에서 학습 ?

필터 = 특징

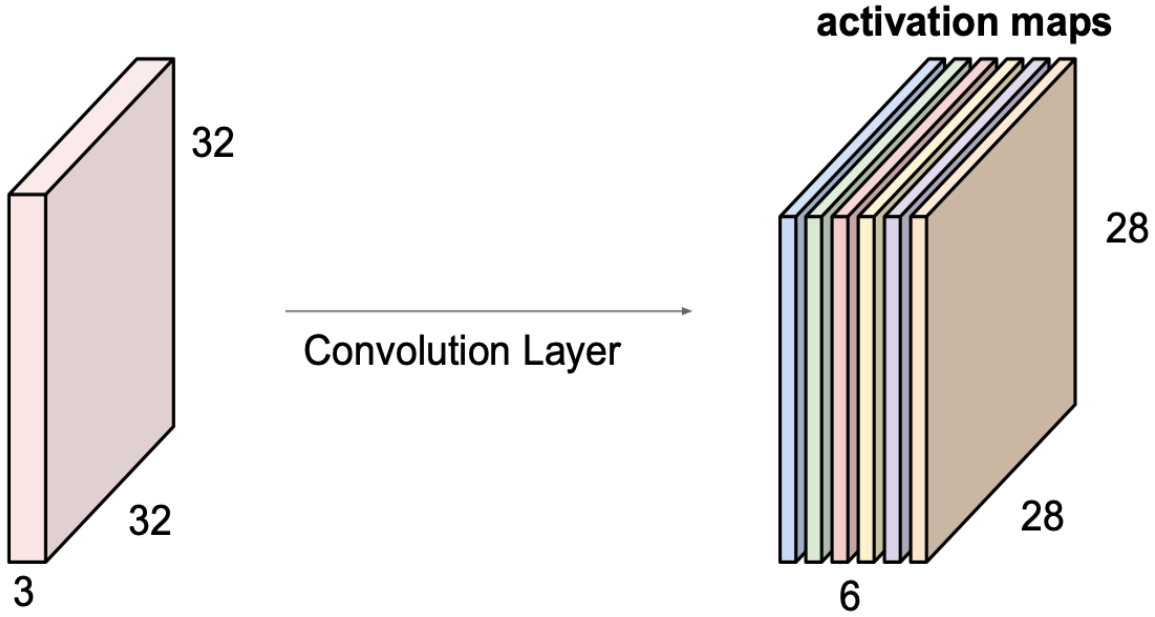
Convolution Neural Network (CNN)

데이터셋에 특화된 학습 필터
다양한 필터로 구성

이미지를 다양한 특징 벡터로 변경



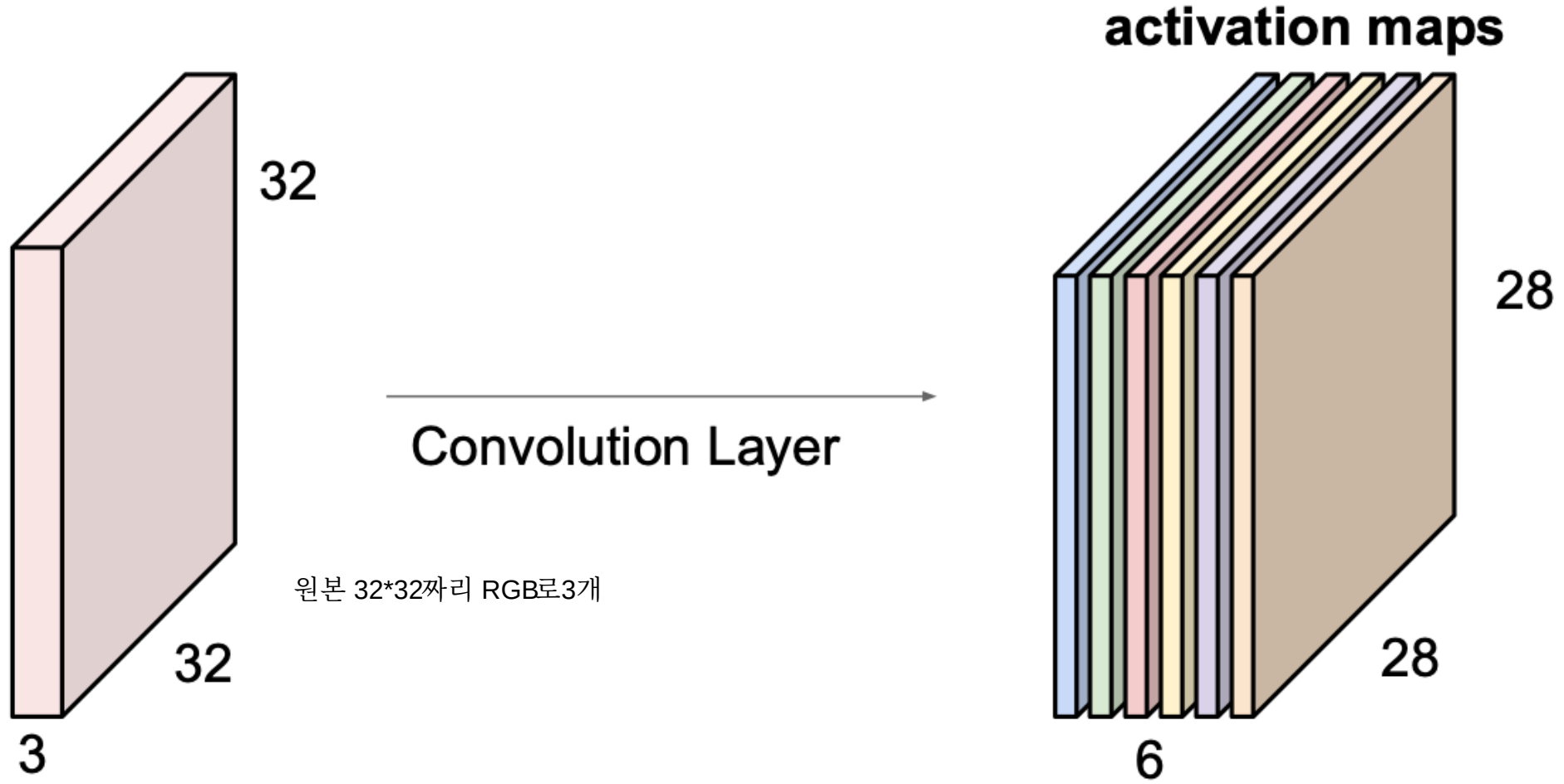
3*3짜리 4개
5*5짜리 3개



32-28 =4 = 필터size-1이므로 필터size는 5
따라서, 이그림을 통해 5*5필터를 적용했다는 것을 확인가능.

만약 이 사이즈 변경이 싫다면 padding기법을 통해 유지가능!

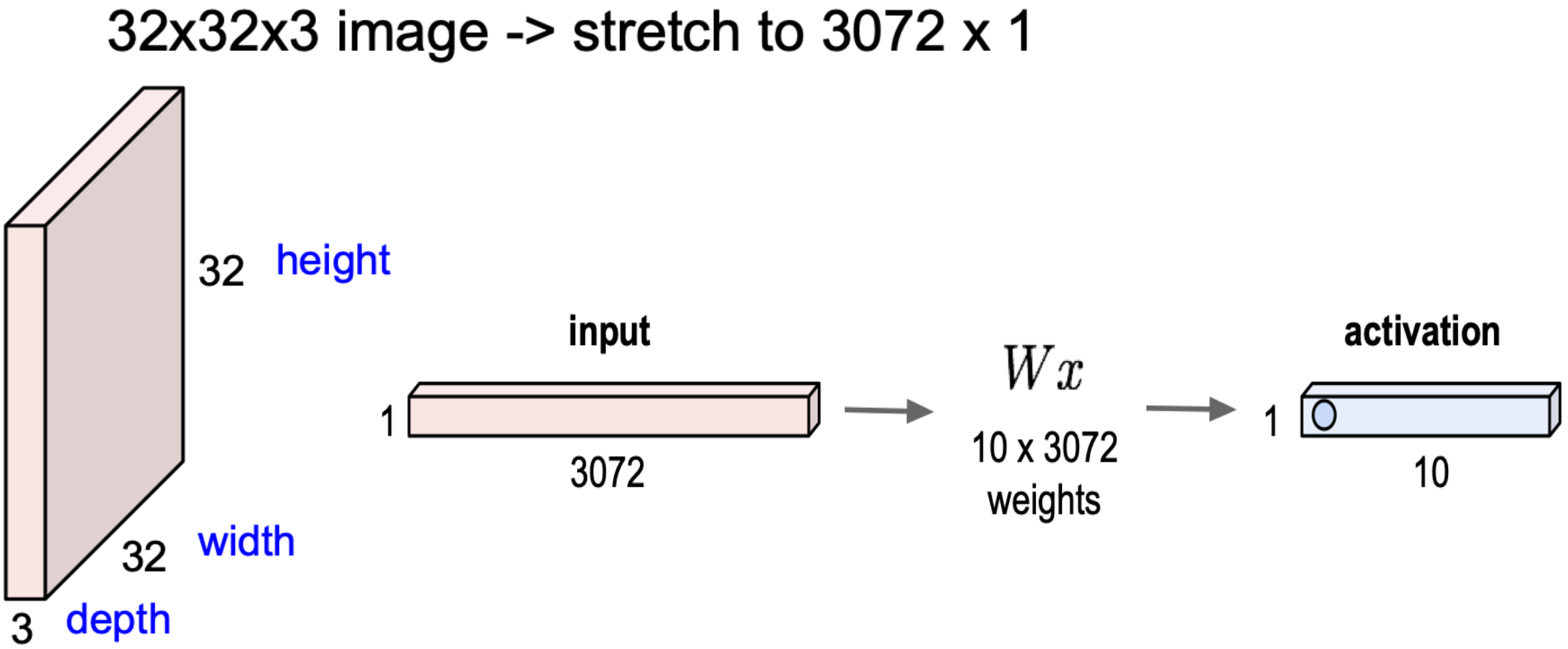
예를 들어, 6개의 5 x 5 (가로 5 * 세로 5)의 필터가 있다고 하면,
각 필터에 해당하는 6개의 서로다른 특징 추출



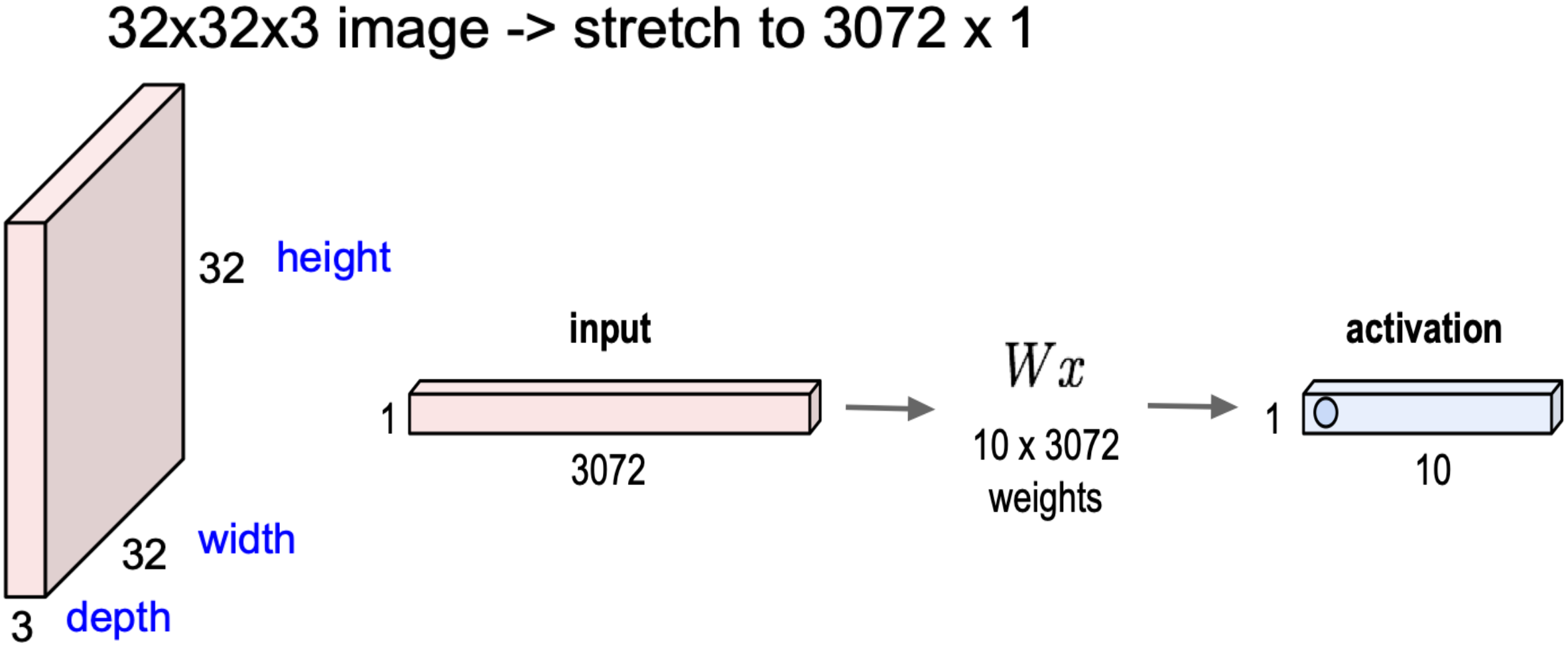
원본 32*32짜리 RGB로3개

이 6개의 추출 특징들을 하나의 새로운 이미지 생성 (크기 28 * 28 * 6)

Dense Layer


32x32x3 image -> stretch to 3072 x 1
32
height
32
width
3
depth
input
1
3072
Wx
10 x 3072
weights
activation
1
10

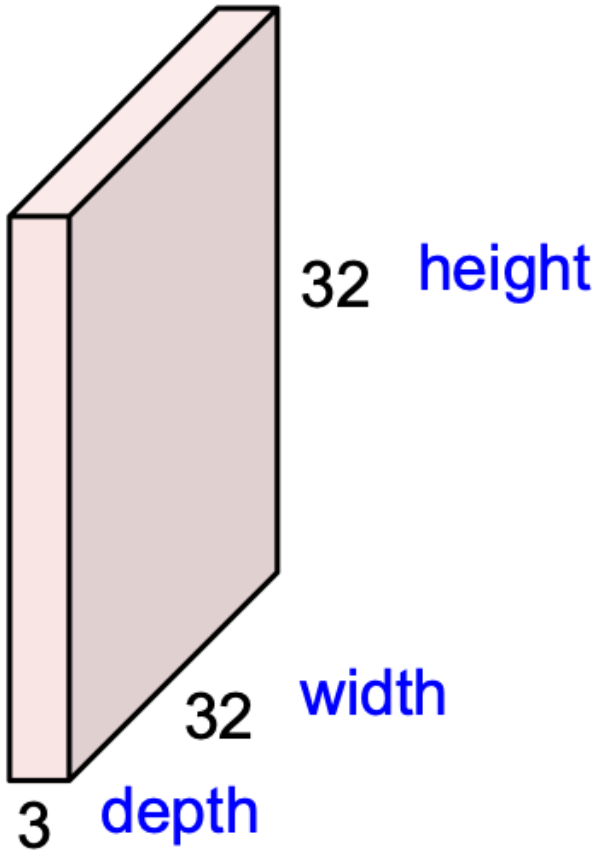
Dense Layer



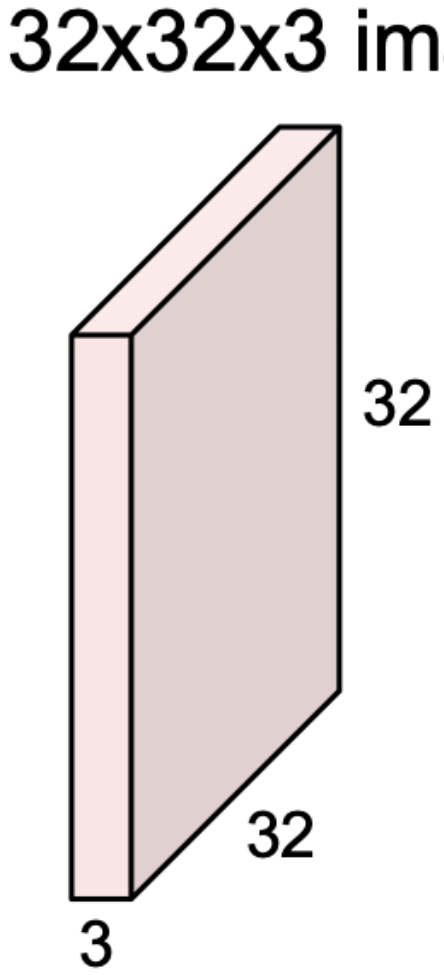
하나의 점 생성시 3072개의 입력 모두 이용

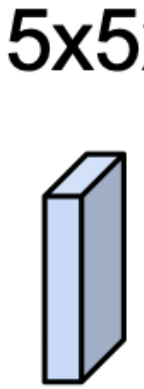
Convolution Layer

32x32x3 image -> 공간 구조 정보가 유지됨



Convolution Layer





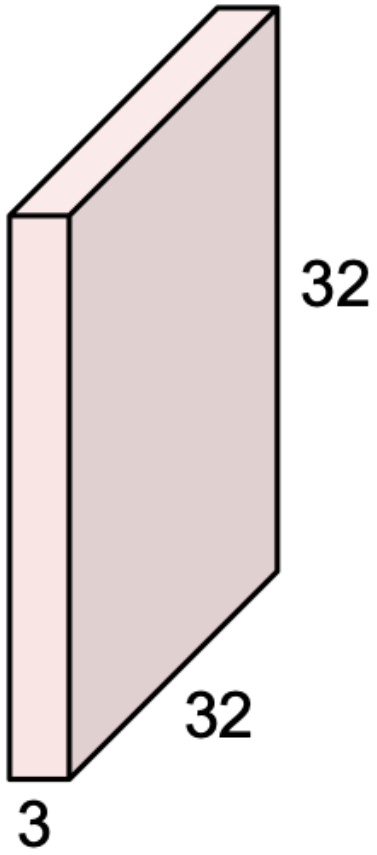
7 x 7로 이루어진 이미지와
Convolve란 필터가 이미지 위를
공간적으로 슬라이딩 하면서
내적(dot product)를 수행

Convolution Layer

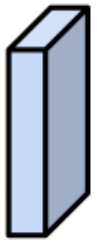
depth of the input volume

같은 필터를 썼다는 가정하엔 특징의 수는 같다.

32x32x3 image

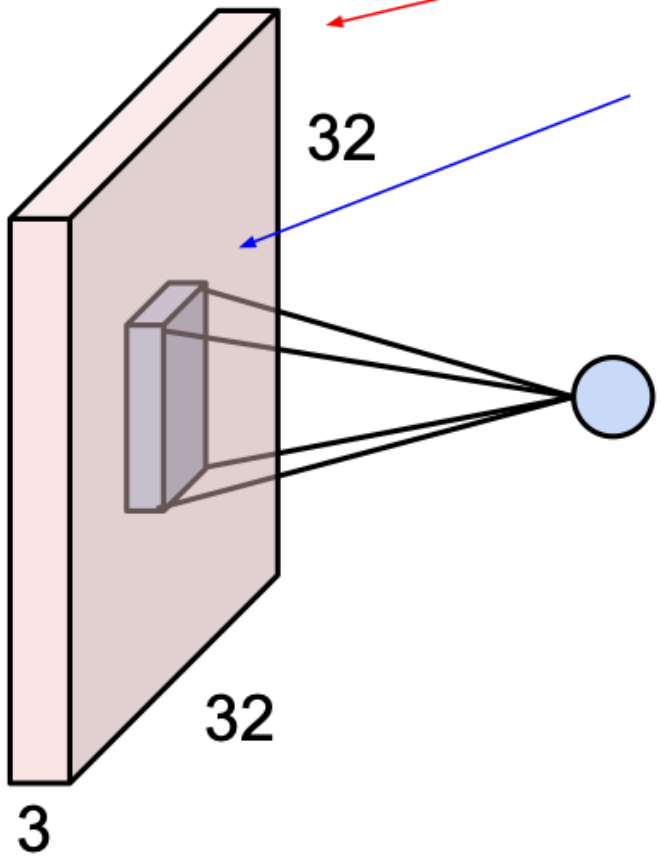


5x5x3 filter



7 x 7로 이루어진 이미지와
Convolve란 필터가 이미지 위를
공간적으로 슬라이딩 하면서
내적(dot product)를 수행

Convolution Layer



32x32x3 image

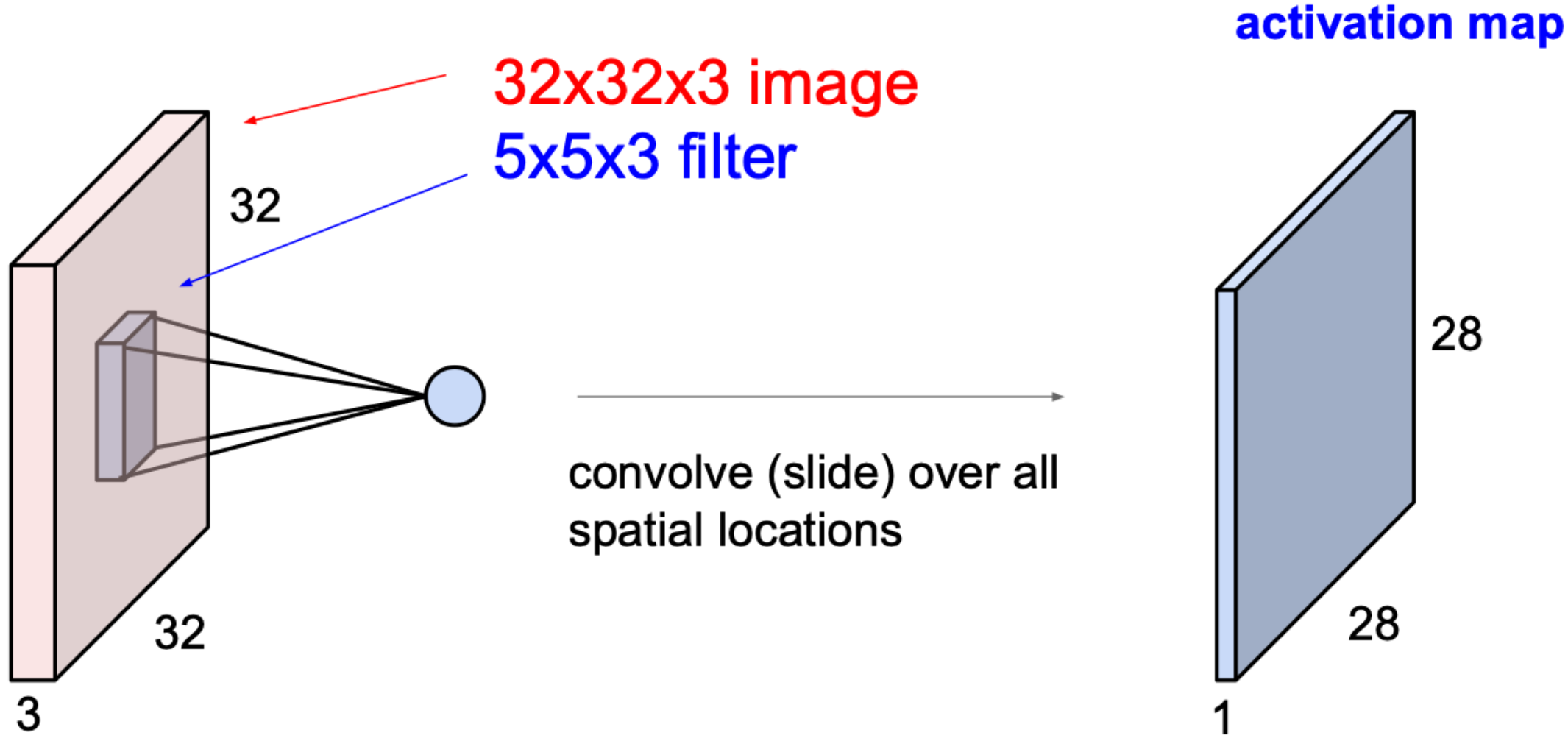
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

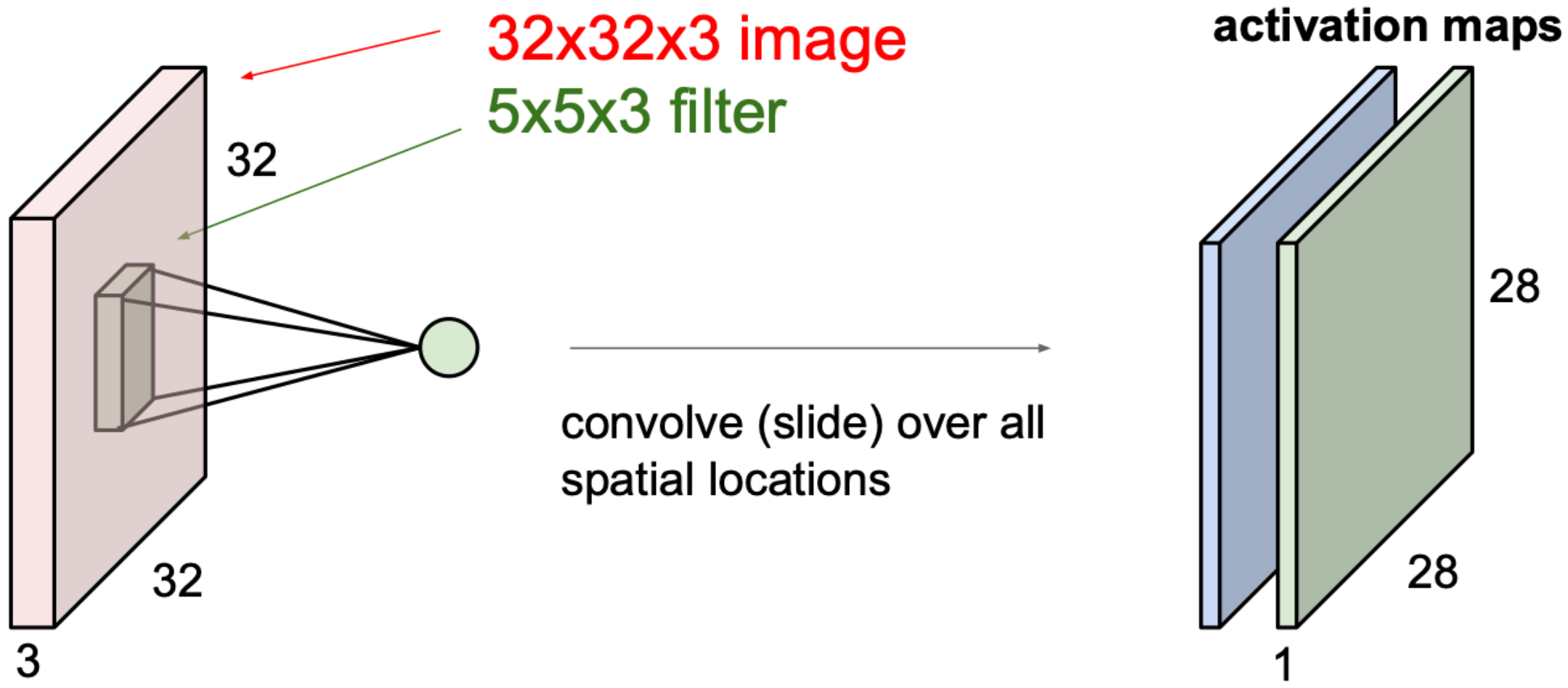
$$w^T x + b$$

5x5x3의 작은 청크이미지마다 하나의 점 생성

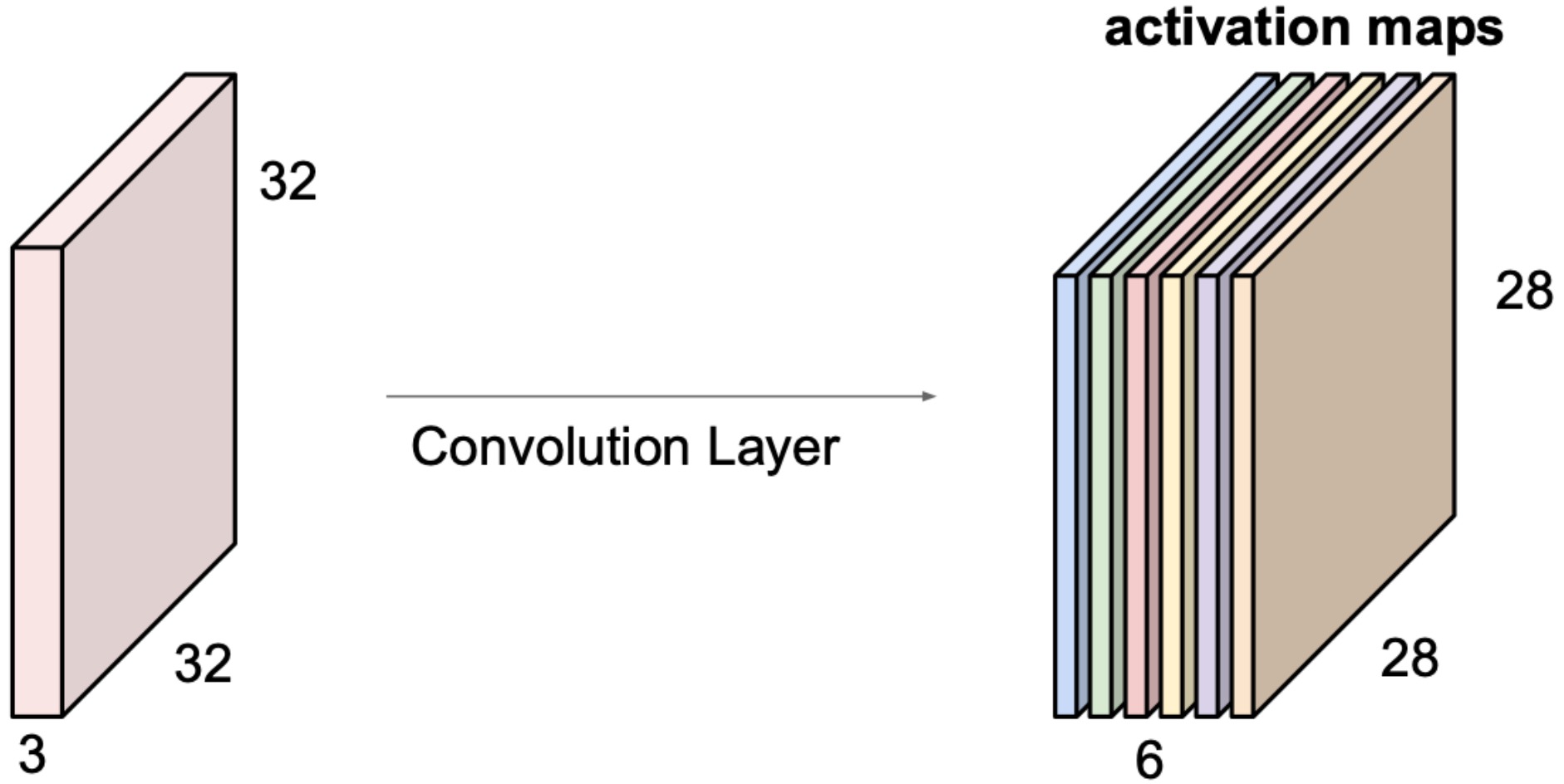
Convolution Layer



Convolution Layer

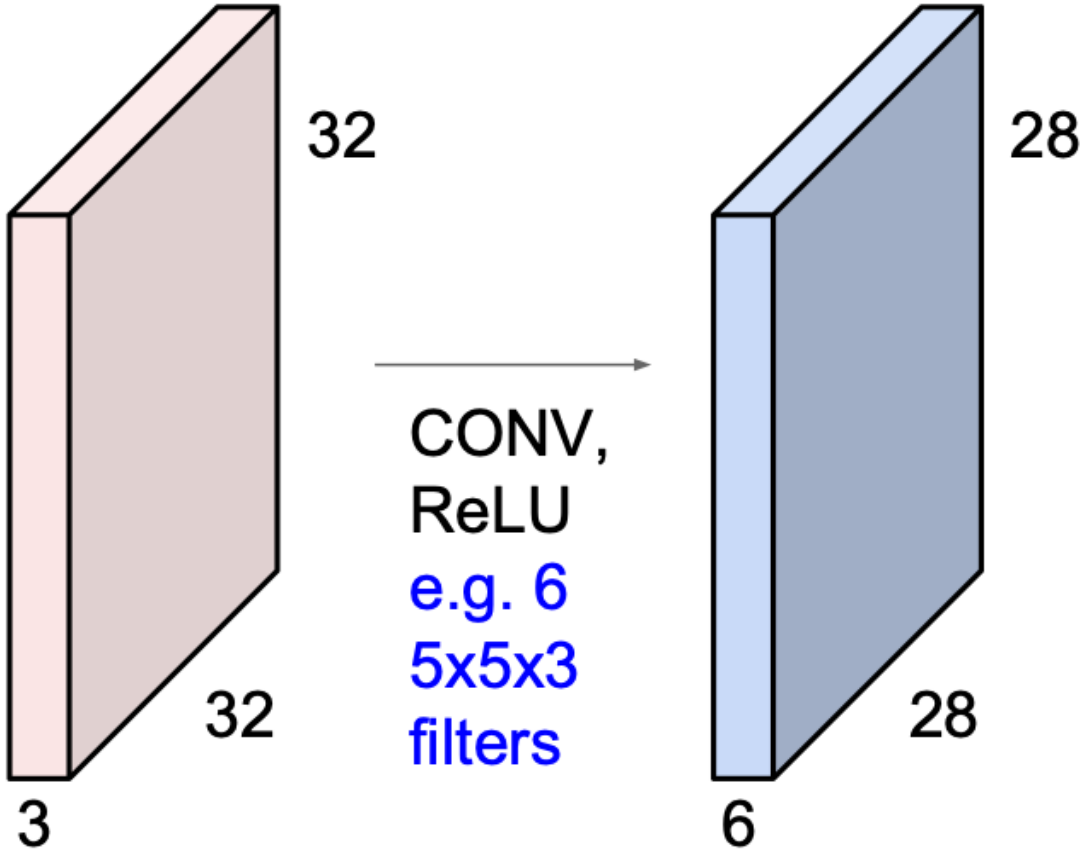


예를 들어, 6개의 5 x 5 (가로 5 * 세로 5)의 필터가 있다고 하면,
각 필터에 해당하는 6개의 서로다른 특징 추출



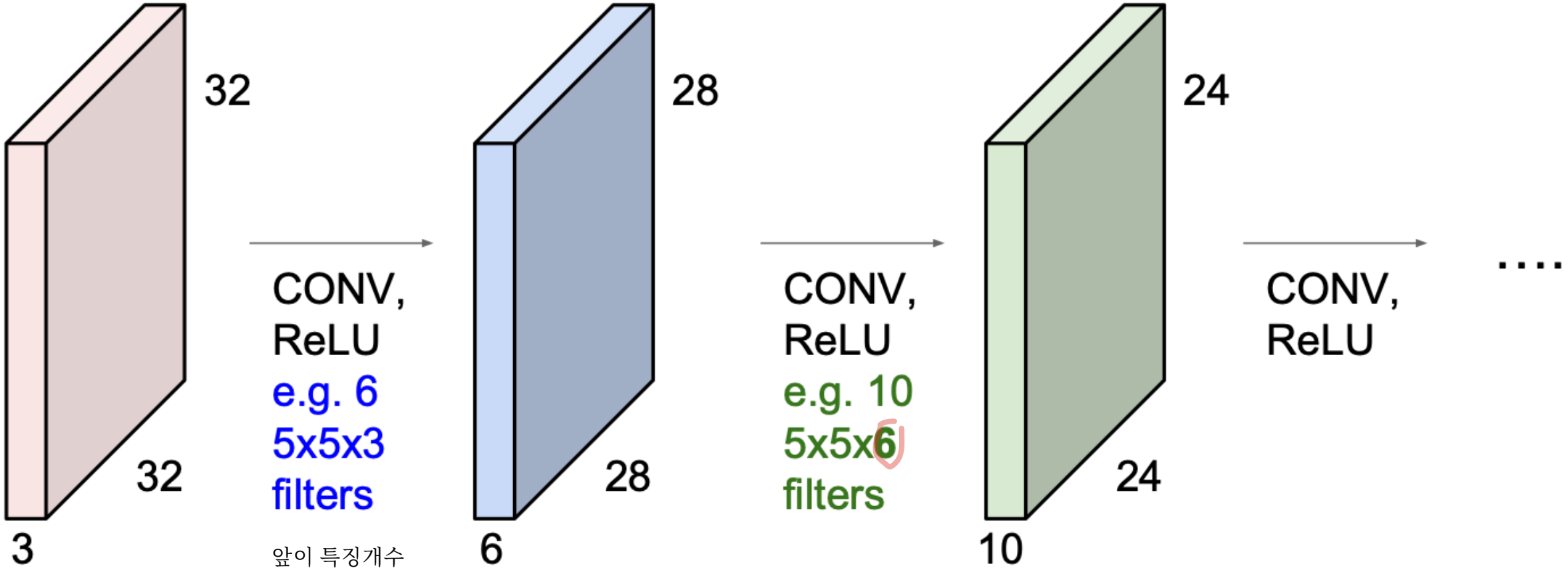
이 6개의 추출 특징들을 하나의 새로운 이미지 생성 (크기 28 * 28 * 6)

ConvNet : Convolution Layer의 시퀀스 (+ Activation functions)



ConvNet : Convolution Layer의 시퀀스 (+ Activation functions)

그밀으로 판단을 할 줄 알아야함.
필터개수 필터특징 등등



앞이 특징개수

곱하기 되어있는건 학습에 썻던 개수이므로 이전 특징 개수

one filter =>
one activation map

Activations:

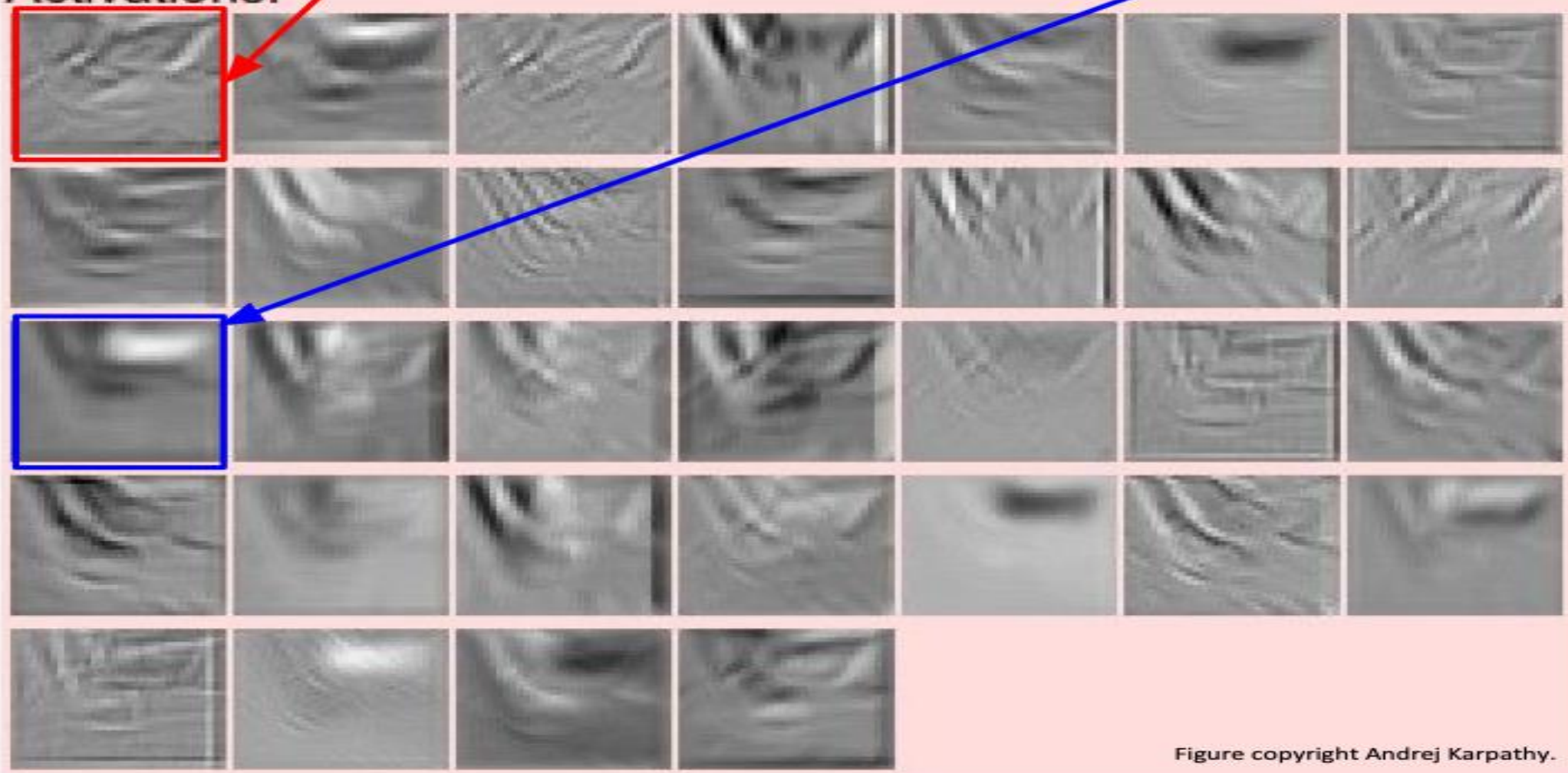


Figure copyright Andrej Karpathy.

CNN 프로그래밍

API 설명 : Conv2D

ex.) 기본포맷 : Conv2D (64, (3*3))이고 이 뒤는 옵션느낌

- Conv2D(filters, kernel_size, strides, padding, …)
- - filters : 필터 개수. 즉, 추출하고자 하는 서로 다른 특징의 개수 특징의 개수
- - kernel_size : 필터 크기 [예: (3,3)]
- - strides : Convolution 동작시 건너뛰는 크기 [예 : (2,2)]
- - padding : 데이터 주면에 0을 padding 할지 여부 [예: ‘same’, ‘valid’]

- 입력 (None, W, H, C) -> 출력 (None, W’, H’, filters)

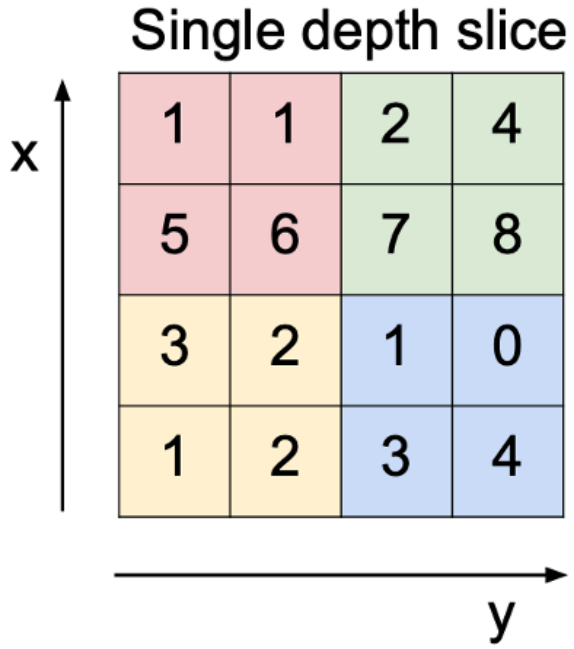
가로,세로, 컬러특징

API 설명 : MaxPooling2D

cf. averagePooling도 있음. 이건 평균값으로 가져감. 하지만 딥러닝에선 보통 이 특징을 가져와야 하기 때문에 max를 많이 사용.

- MaxPooling2D(pool_size, strides)
  - pool_size : pooling할 대상의 크기 (예: (2,2))
  - strides : 동작시 건너뛰는 크기 [예 : (2,2)]

이미지 특성상 주변 영역은 거의 비슷하게 나올 것 (확 오차범위 넘어서 나올 확률이 적음)



max pool with 2x2 filters
and stride 2

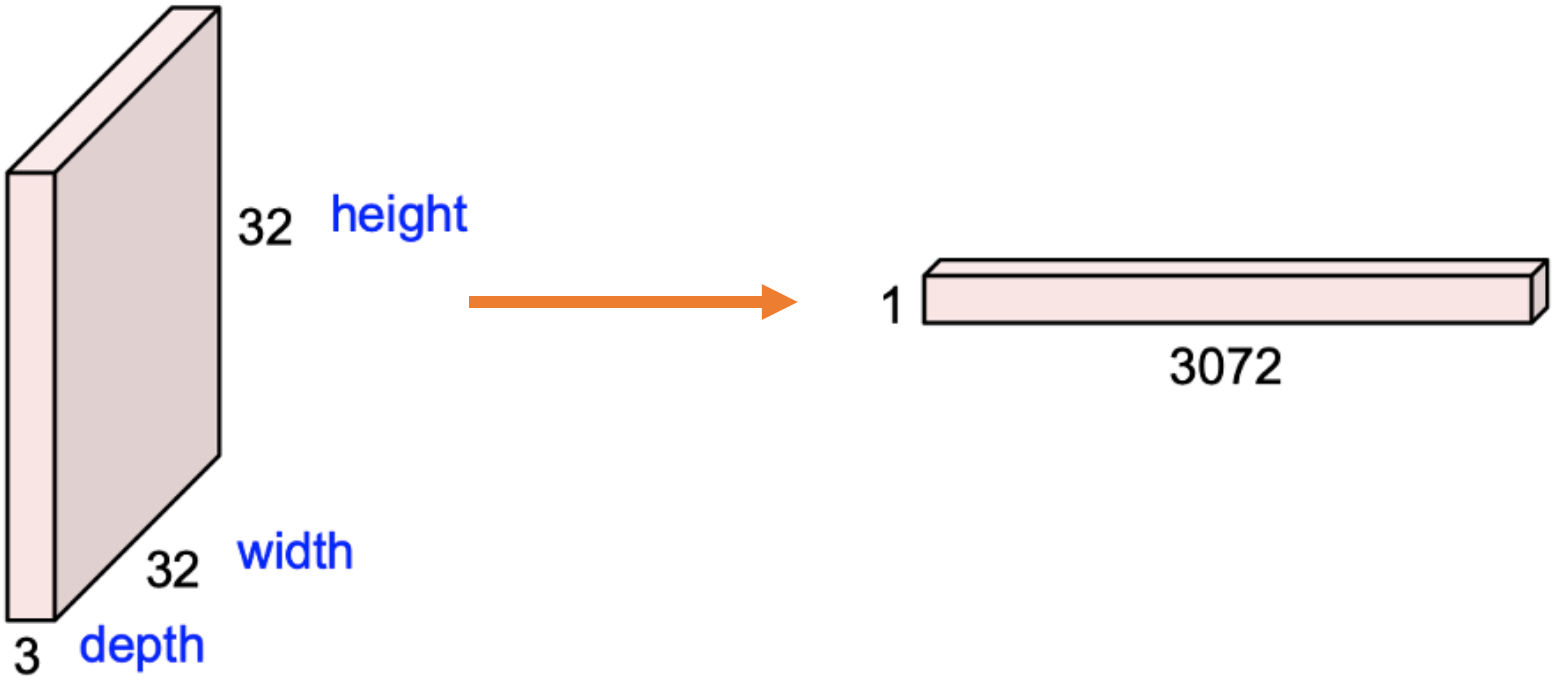
영역을 나눠 그중 최댓값만
남기는 구조

6	8
3	4

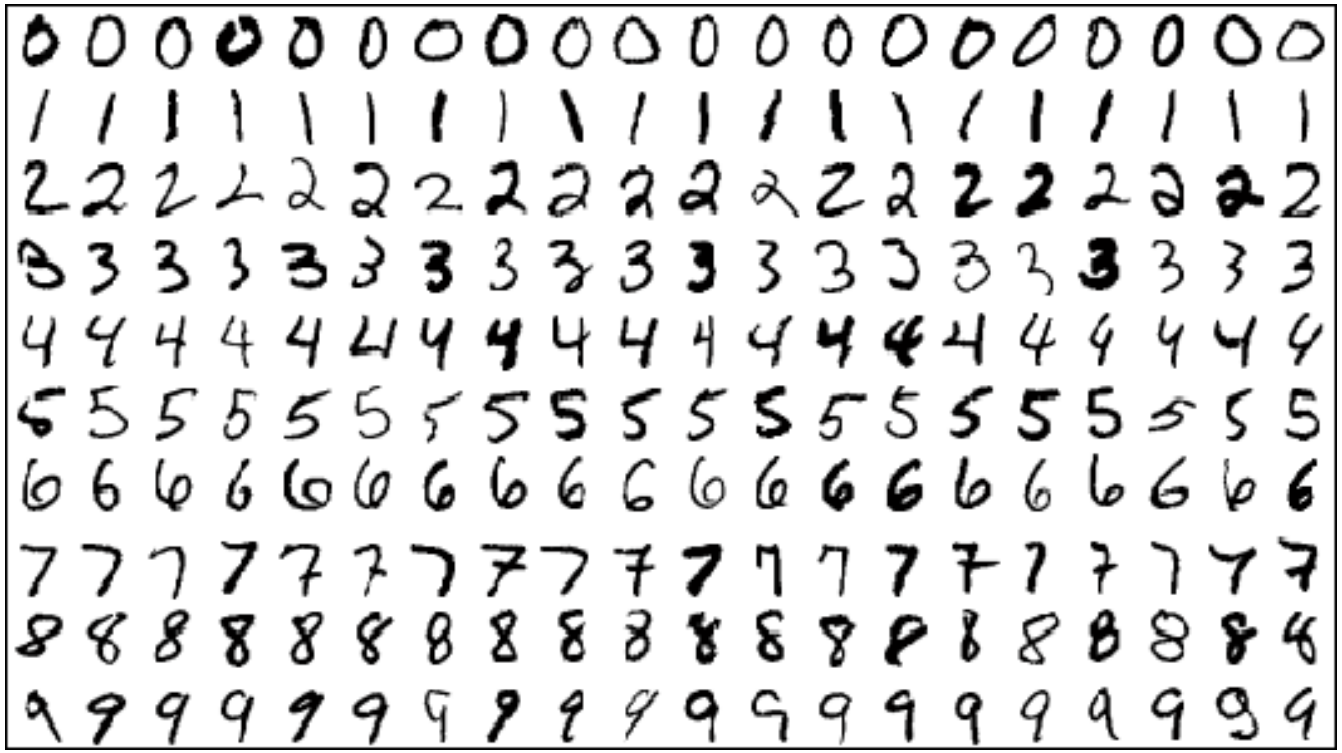
API 설명 : Flatten()

tenser라는 타입으로 데이터를 모델에선 변화해서 사용하기 때문에 reshape을 사용할순 없고 Flatten()을 사용함.

- 추가 파라미터 없이 단독으로 사용
- 다차원의 데이터를 1차원 데이터로 변환



예제 : MNIST with CNN



참고 : https://github.com/rickiepark/deep-learning-with-python-notebooks/blob/master/5.1-introduction-to-convnets.ipynb

예제 : MNIST

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

참고 : https://github.com/rickiepark/deep-learning-with-python-notebooks/blob/master/5.1-introduction-to-convnets.ipynb

예제 : MNIST

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

사이즈를 줄여줌.

확률개념으로 가장 높은걸 취하기 때문.

참고 : https://github.com/rickiepark/deep-learning-with-python-notebooks/blob/master/5.1-introduction-to-convnets.ipynb

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 131,242
Trainable params: 131,242
Non-trainable params: 0
_________________________________________________________________
```

입력 input (28,28,1)여기서 1이 입력채널의 개수로 이미지 사이즈와는 무관

가로세로만 줄어듦 (특징 두께는 그대로)

why? -> 3*3필터로 9개+y절편으로 총 10
파라메타가 존재 이게 32개가 있으므로
32 * 10= 320 (총필터개수)
처음껀 채널개수가 1이므로 320*1

가 32개이므로
개수)
289*32 = 9248

만약 여기서 conv2d_2로
하나 더 추가할때 (32,3,3,

개수는
. 같음.

64 * (1024+1)

10 * (64+1)

즉 dense는 내모델 * (이전모델 +y절편1)

예제 : MNIST

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5
938/938 [==============================] - 5s 6ms/step - loss: 0.1428 - accuracy: 0.9557
Epoch 2/5
938/938 [==============================] - 5s 6ms/step - loss: 0.0409 - accuracy: 0.9875
Epoch 3/5
938/938 [==============================] - 5s 6ms/step - loss: 0.0285 - accuracy: 0.9914
Epoch 4/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0219 - accuracy: 0.9934
Epoch 5/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0178 - accuracy: 0.9950
<tensorflow.python.keras.callbacks.History at 0x7f55da0746a0>
```

참고 : https://github.com/rickiepark/deep-learning-with-python-notebooks/blob/master/5.1-introduction-to-convnets.ipynb

Dense VS CNN

- Dense의 경우 파라미터 개수가 CNN대비 엄청 큼

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

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

CNN의 경우

```
Total params: 131,242
Trainable params: 131,242
Non-trainable params: 0
```