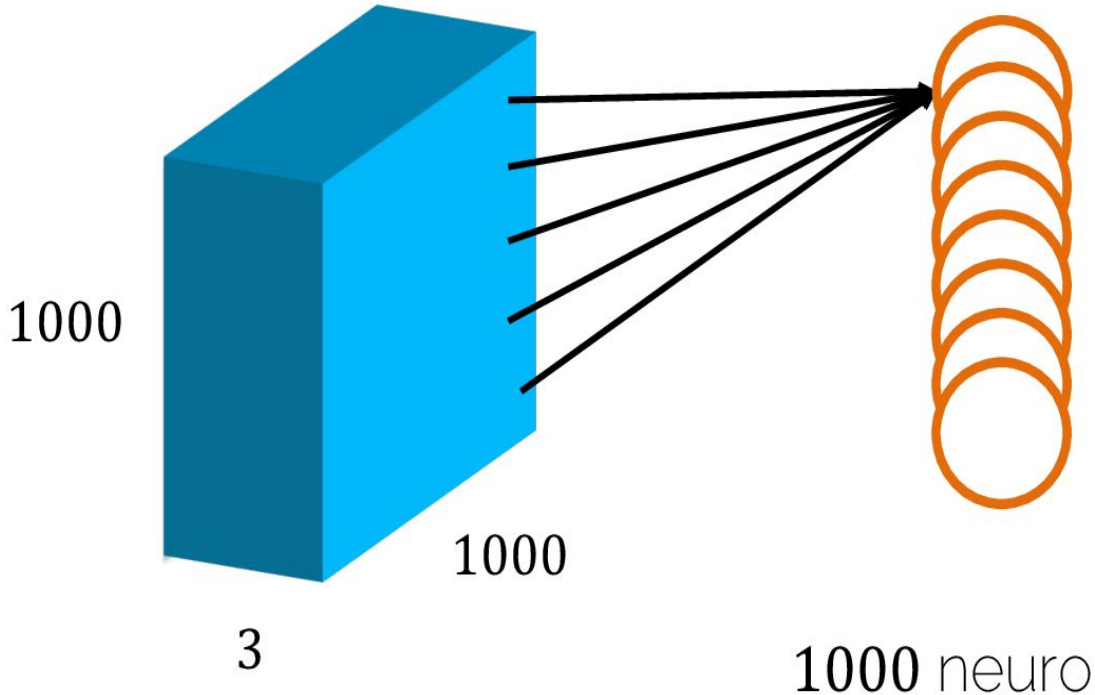
Deep learning for Computer Vision

Problems using FC Layers on Images

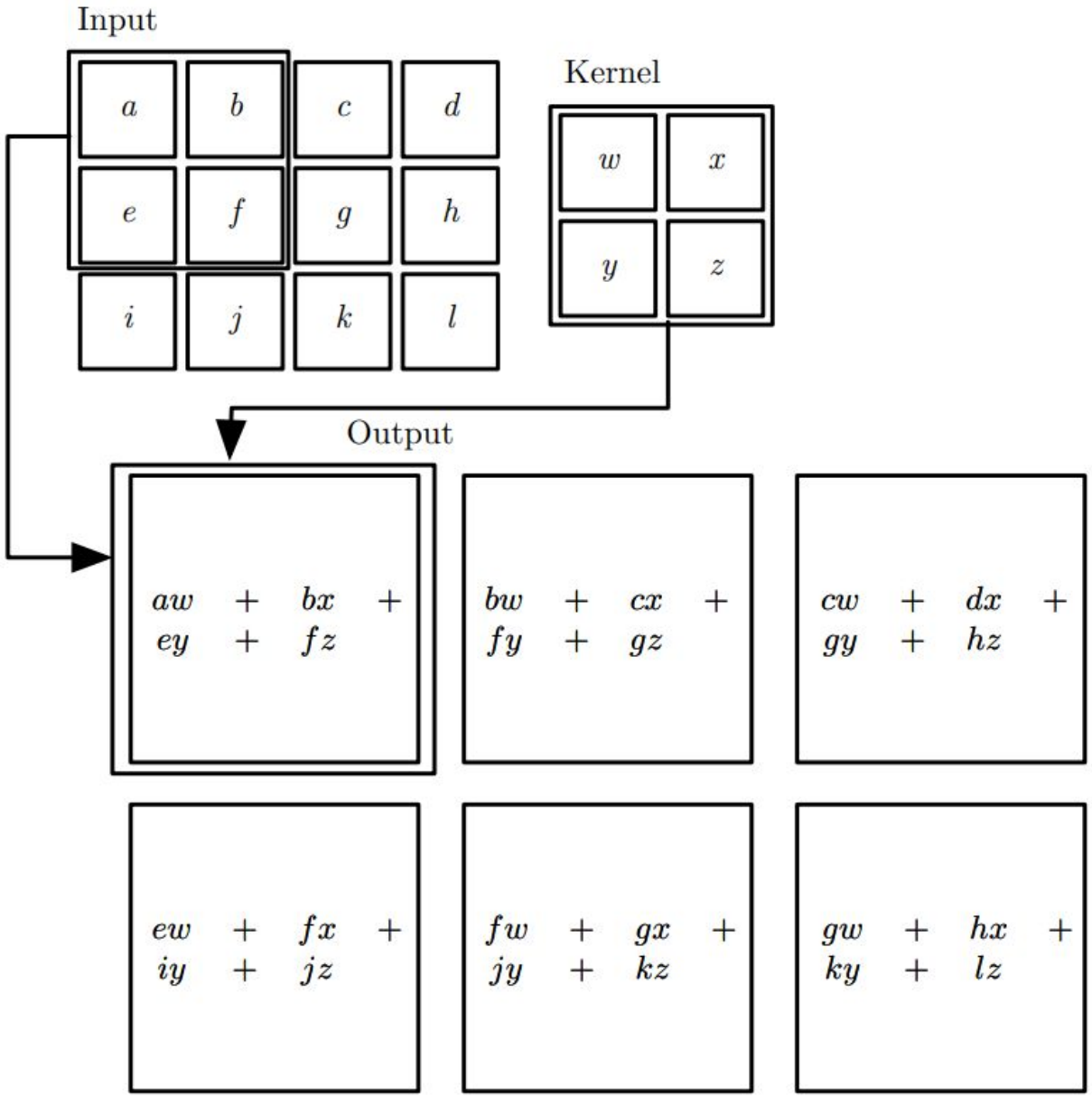
How to process a tiny image with FC layers



3 *billion* weights

IMPRACTICAL

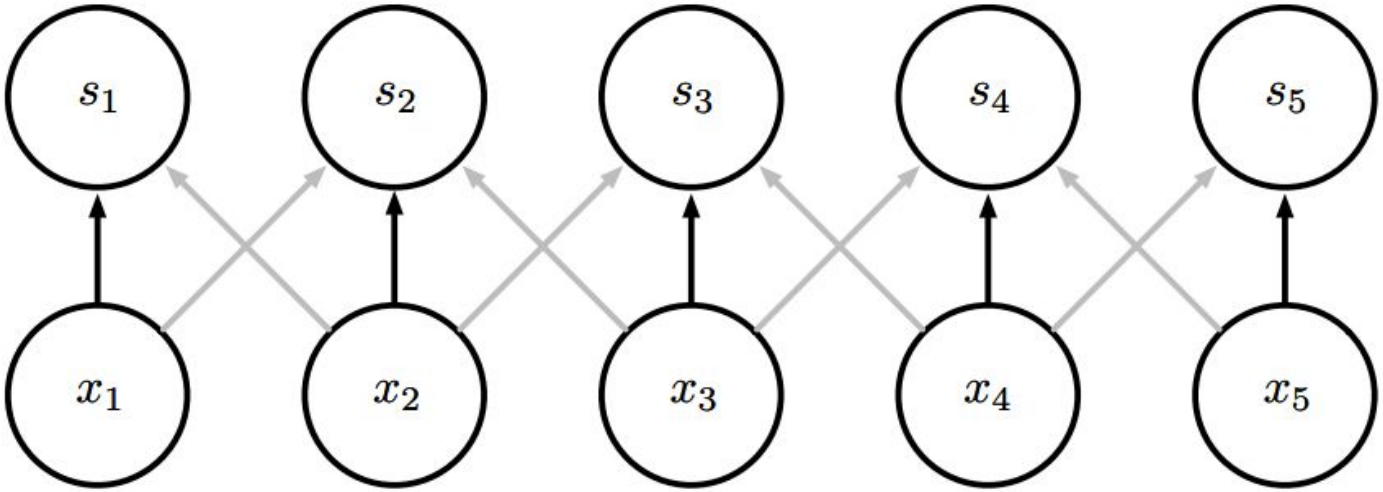
Solution is Convolution



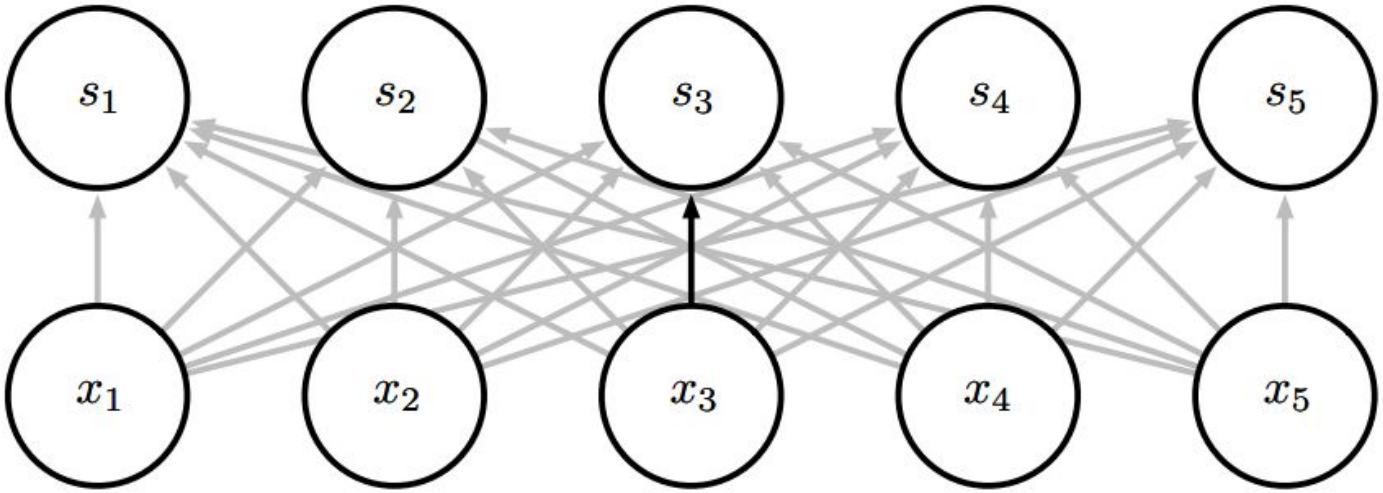
Parameter Sharing

Parameter Sharing → how padhleya?

Convolution shares the same parameters across all spatial locations



Traditional matrix multiplication does not share any parameters

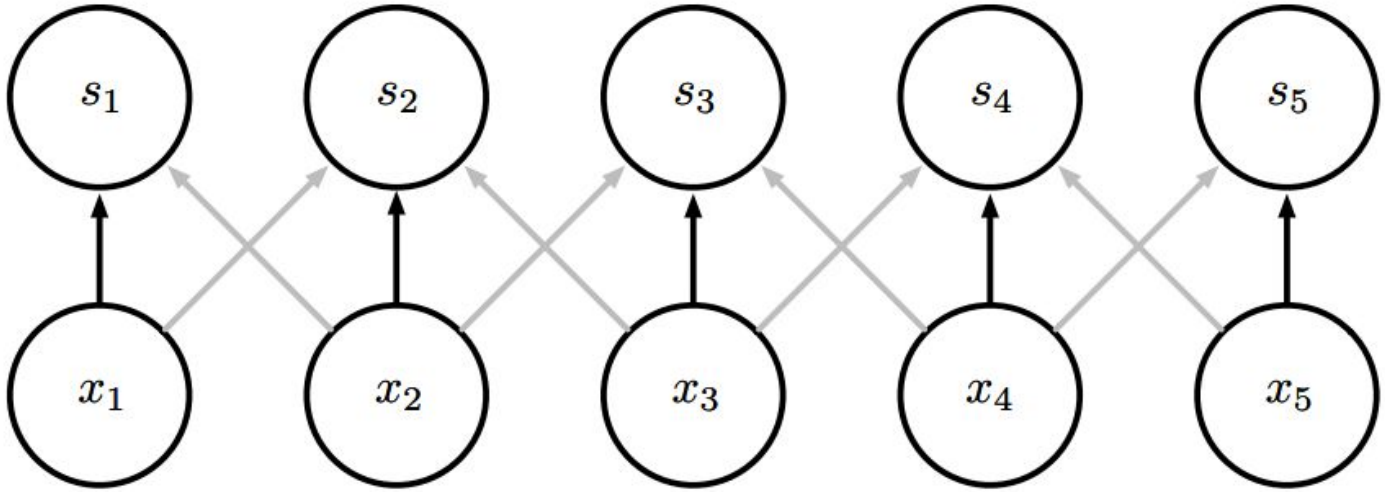


Parameter Sharing

→ Enables to learn translation invariance features.

↓ Translation invariance.

Convolution shares the same parameters across all spatial locations

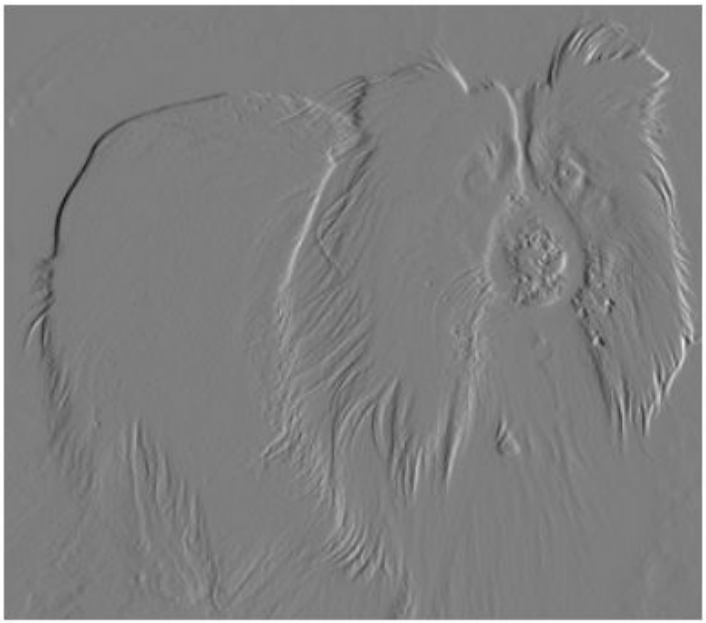


- the same set of weights is used to compute the output for all neurons in the same feature map.
- Parameter sharing reduces the number of parameters needed to train.
- Parameter Sharing enables the model to learn translation-invariant features, for example, kernel for edge.

Edge Detection by Convolution

Input

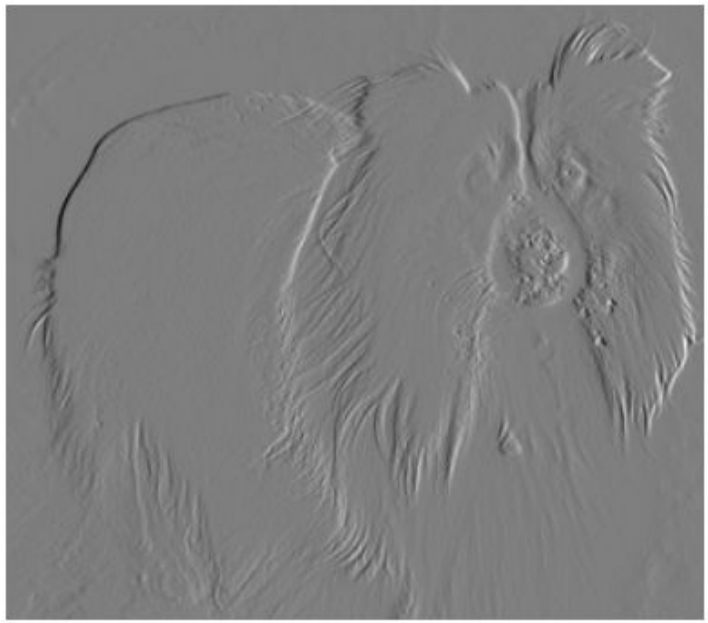
?



Output

Edge Detection by Convolution

Input



Output

1	-1

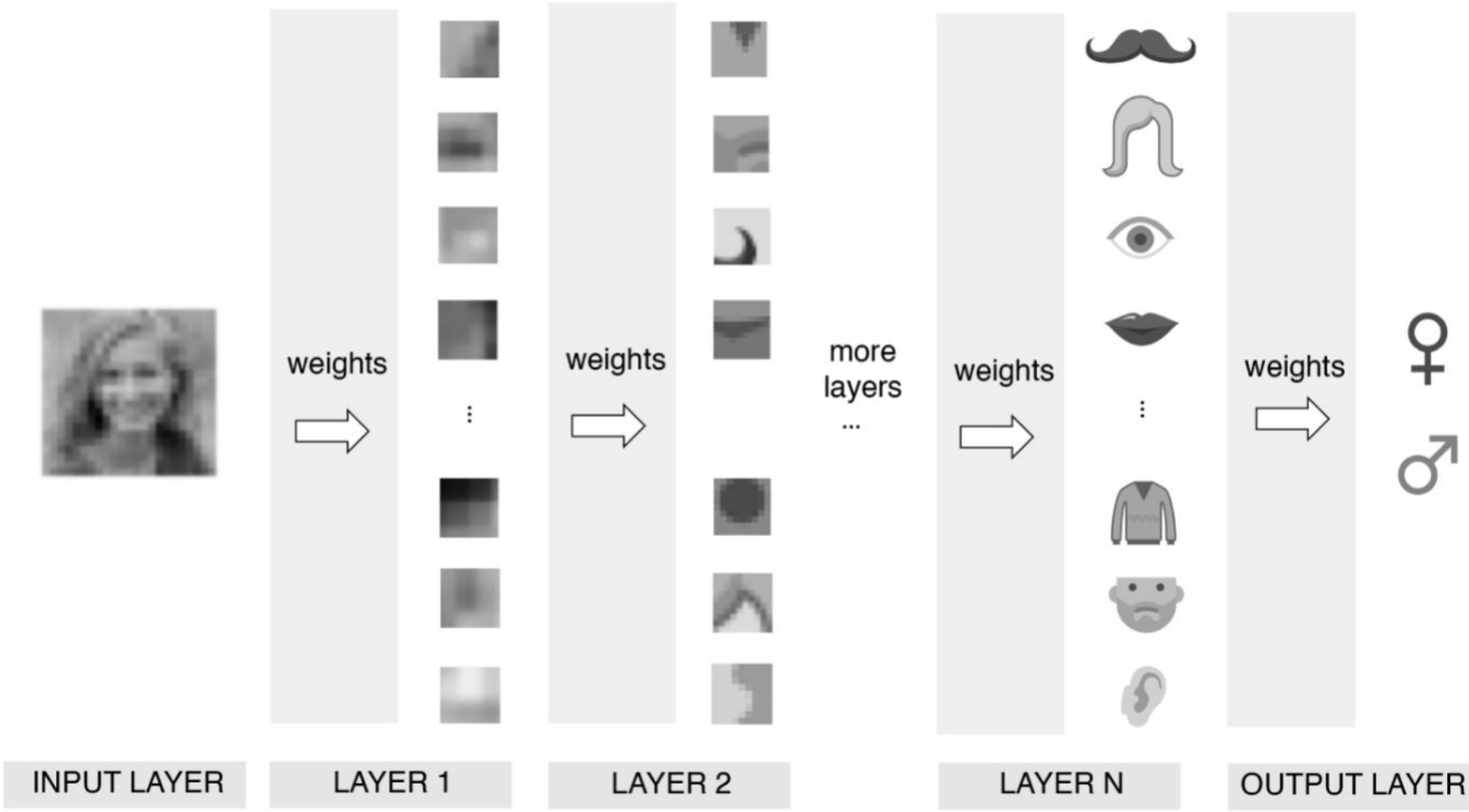
Kernel

Efficiency of Convolution

Input: 320 x 280
Kernel: 2 x 1
Output: 319 x 280

	Convolution	Dense matrix
Stored floats	2	319*280*320*280
Float muls or adds	319*280*3 = 267,960	Huge

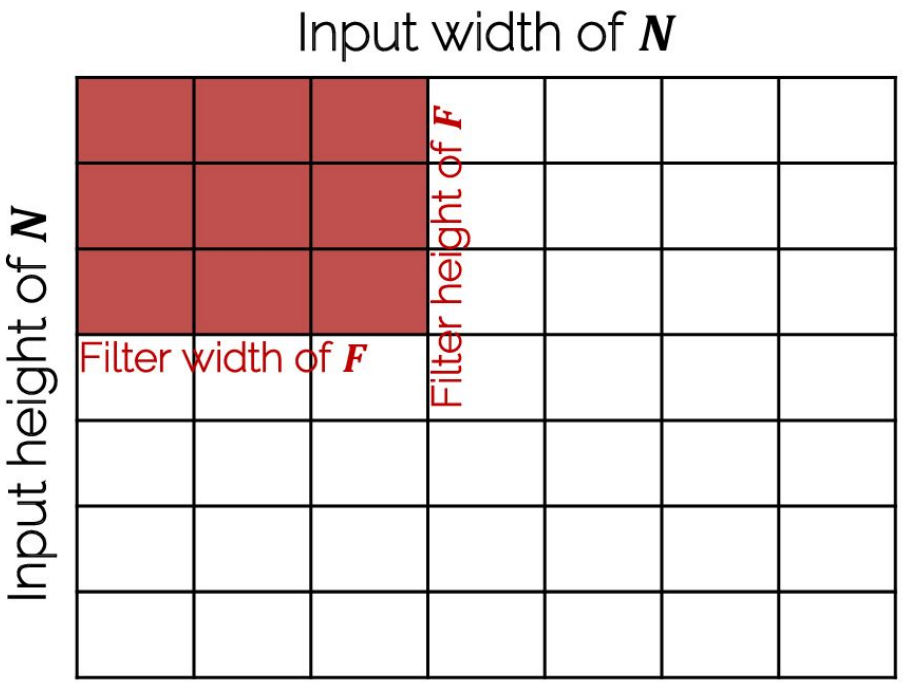
Feature Extraction using CNN



Aim is to learn useful kernel using DL

Padding

Recall Stride



Input	$N \times N$
Filter	$F \times F$
Stride	S
Output	$\left(\frac{N-F}{S}+1\right) \times \left(\frac{N-F}{S}+1\right)$

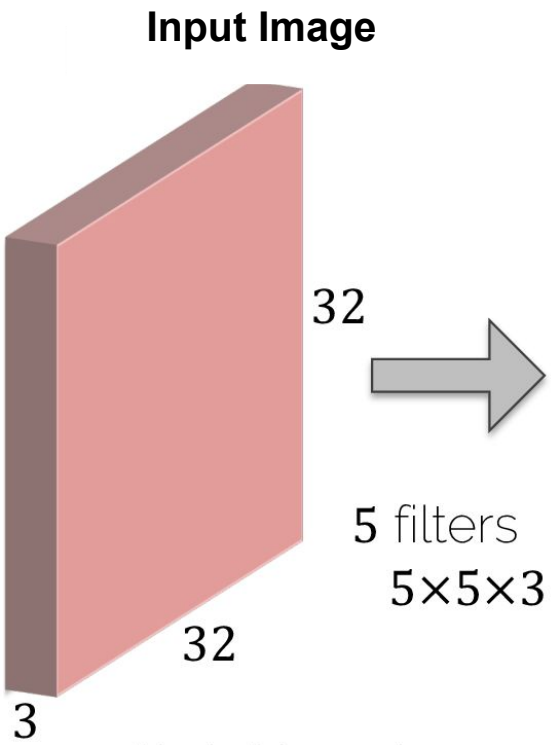
$N = 7, F = 3, S = 1$: $\frac{7-3}{1}+1 = 5$

$N = 7, F = 3, S = 2$: $\frac{7-3}{2}+1 = 3$

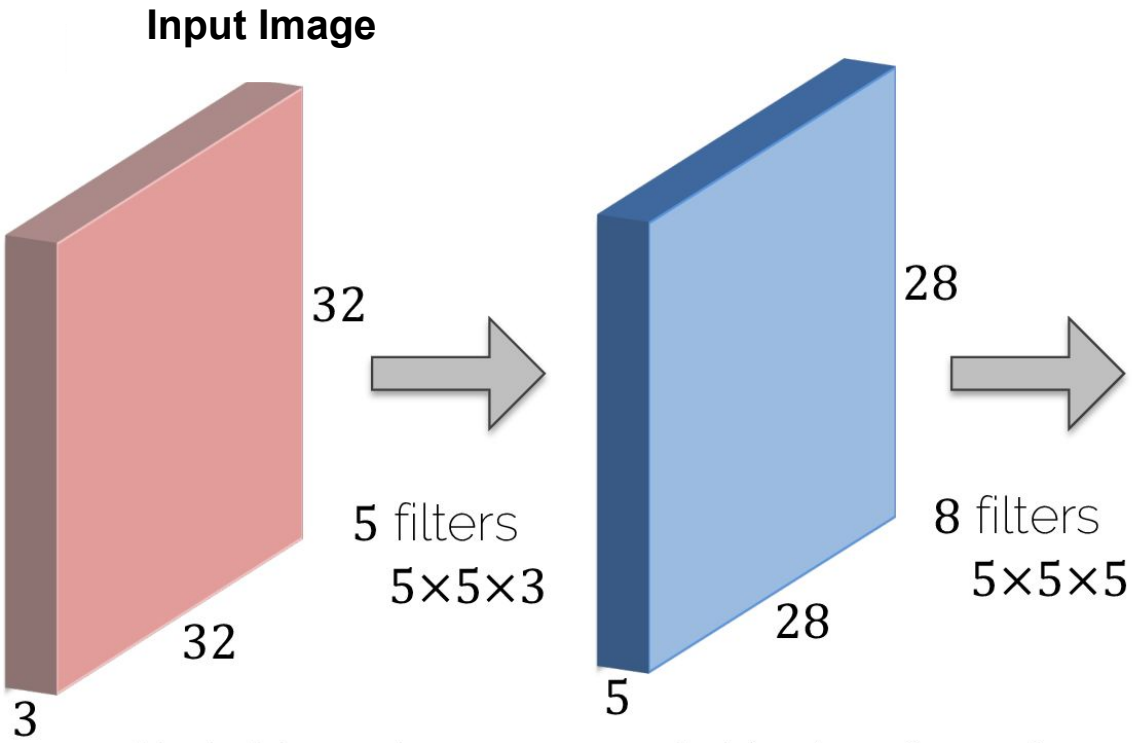
$N = 7, F = 3, S = 3$: $\frac{7-3}{3}+1 = 2.\overline{3}$

Fractions are illegal

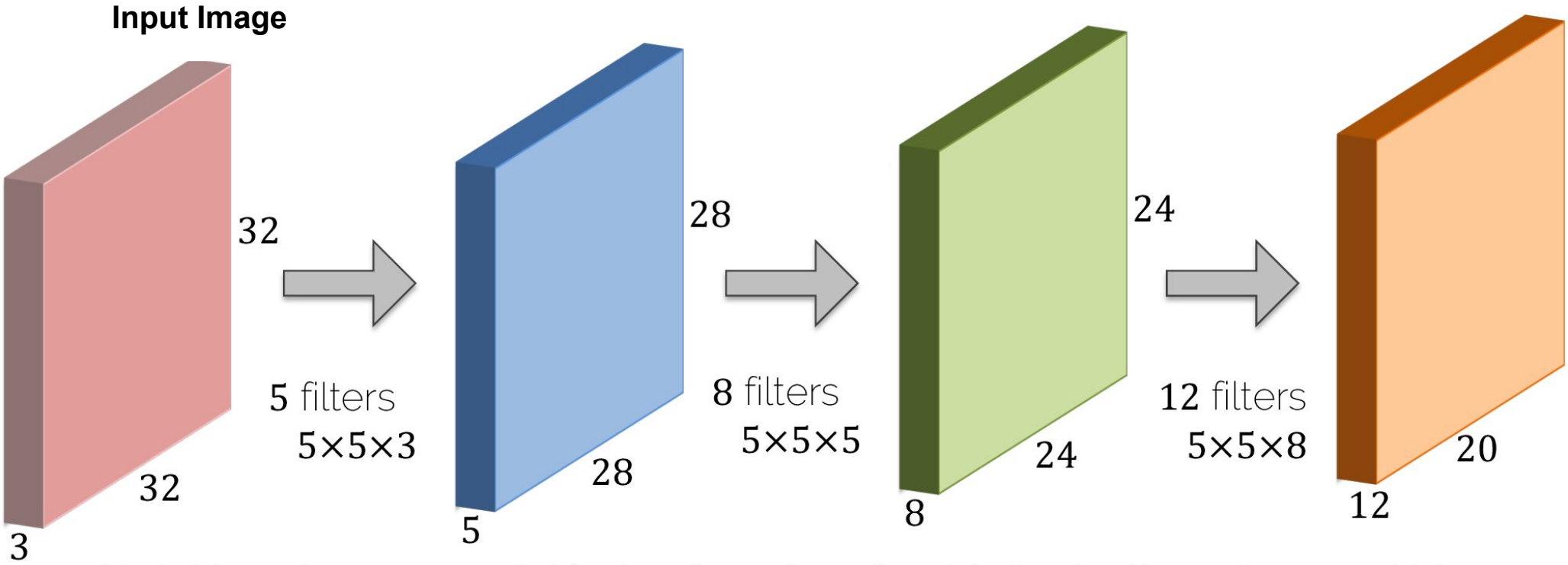
Convolution Layers:Dimensions



Convolution Layers:Dimensions

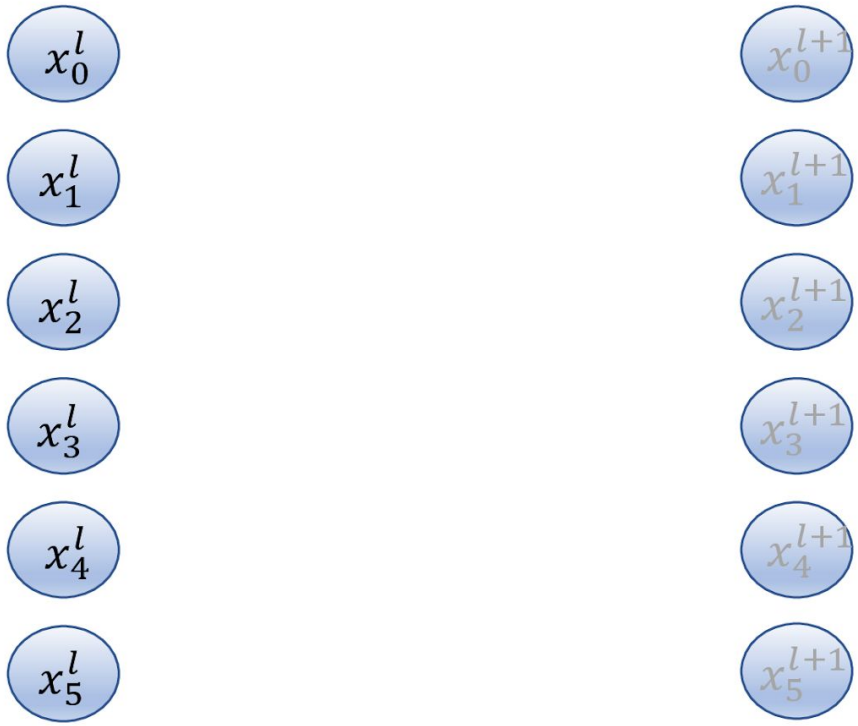


Convolution Layers:Dimensions

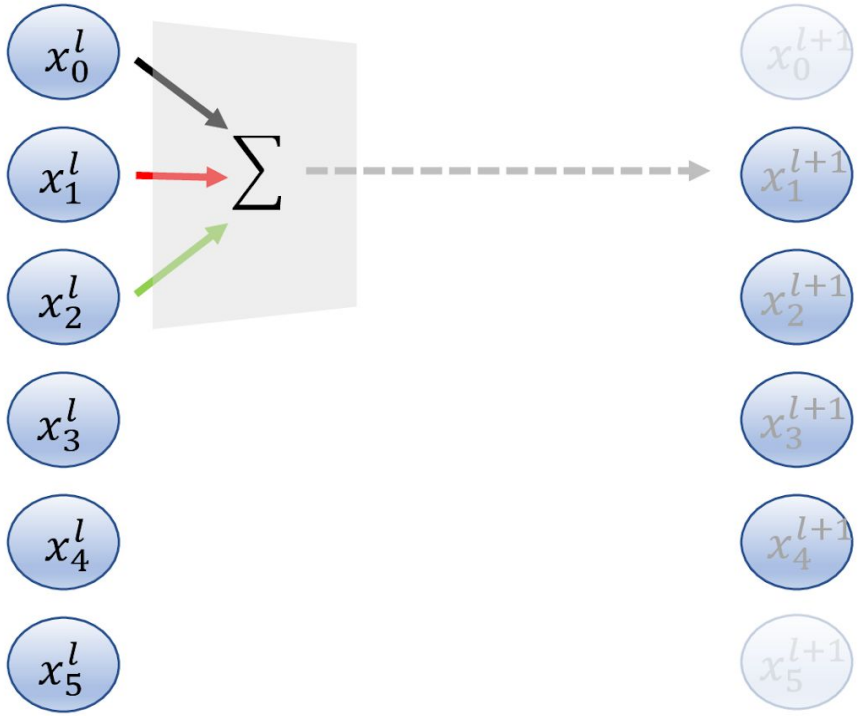


Always shrinking down may not a good approach (information loss)

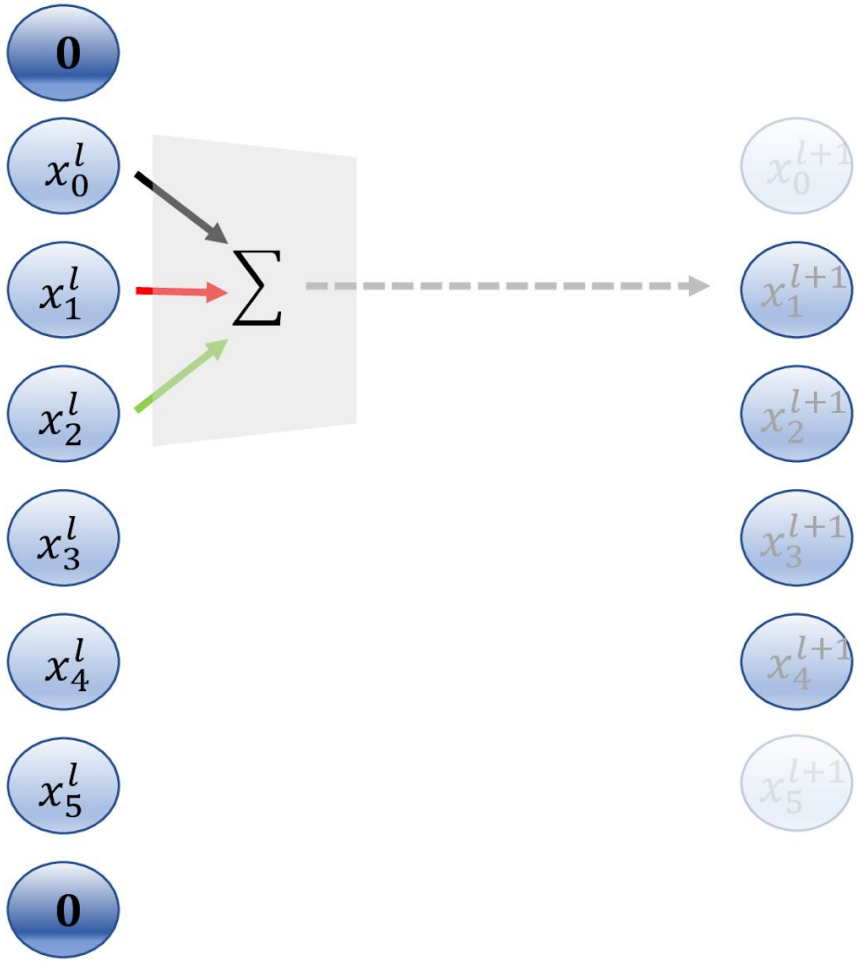
Padding (Key idea)



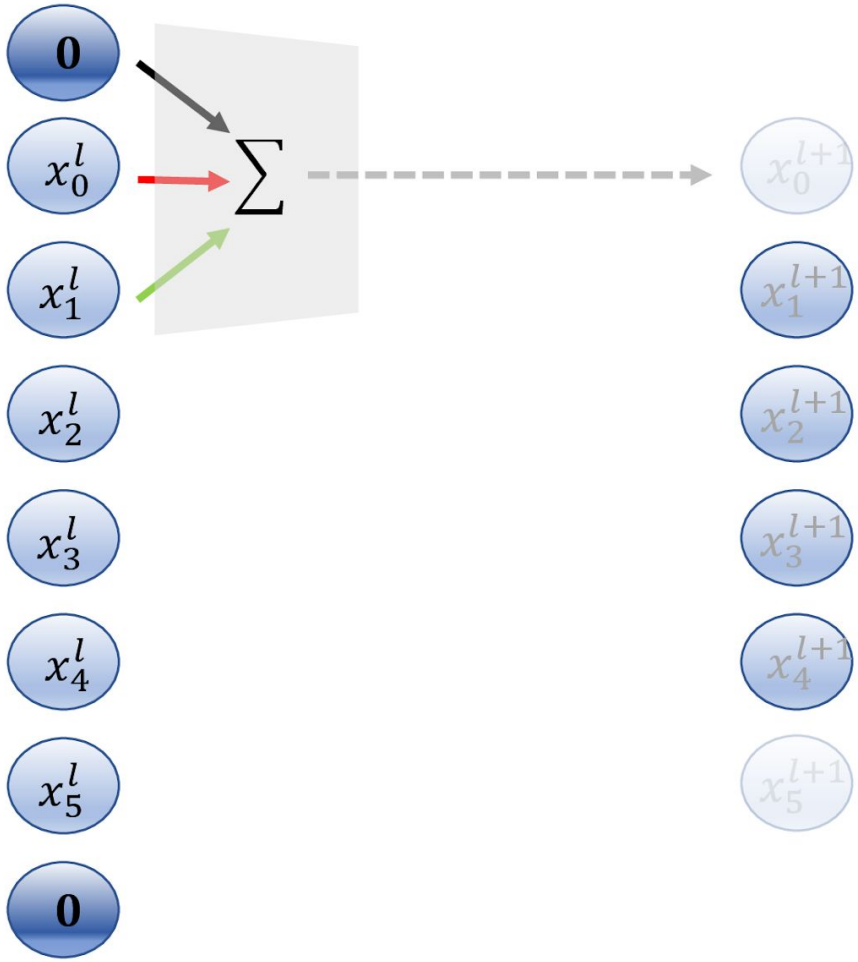
Padding (Key idea)



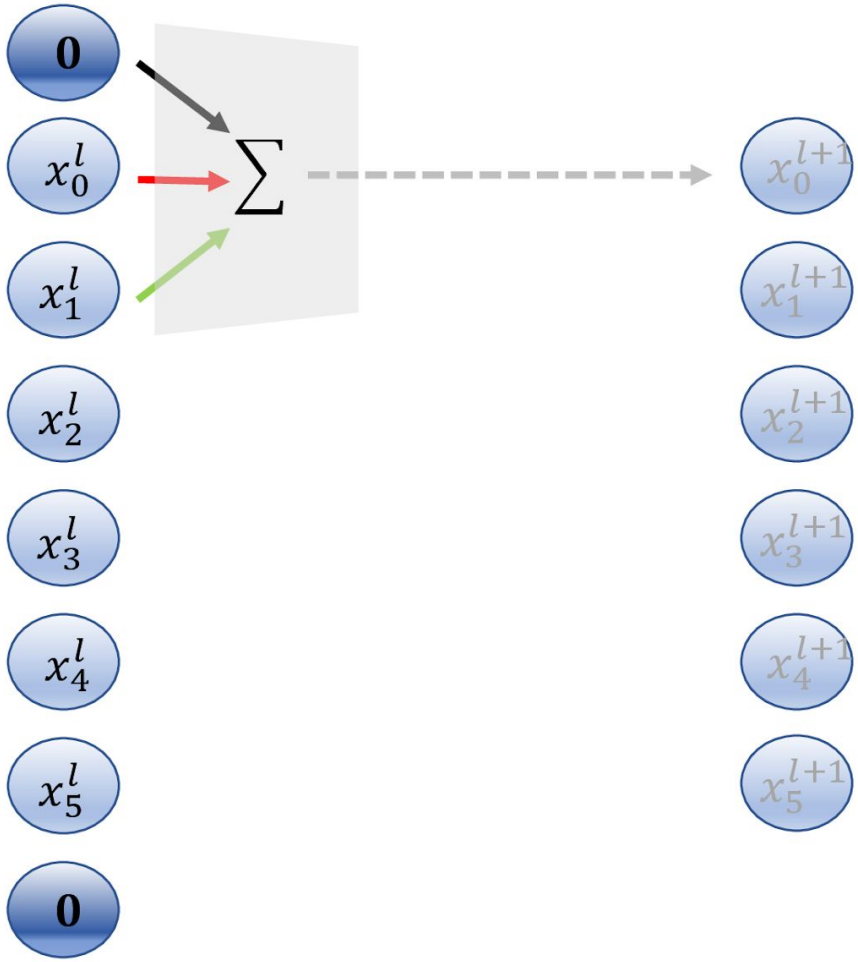
Padding (Key idea)



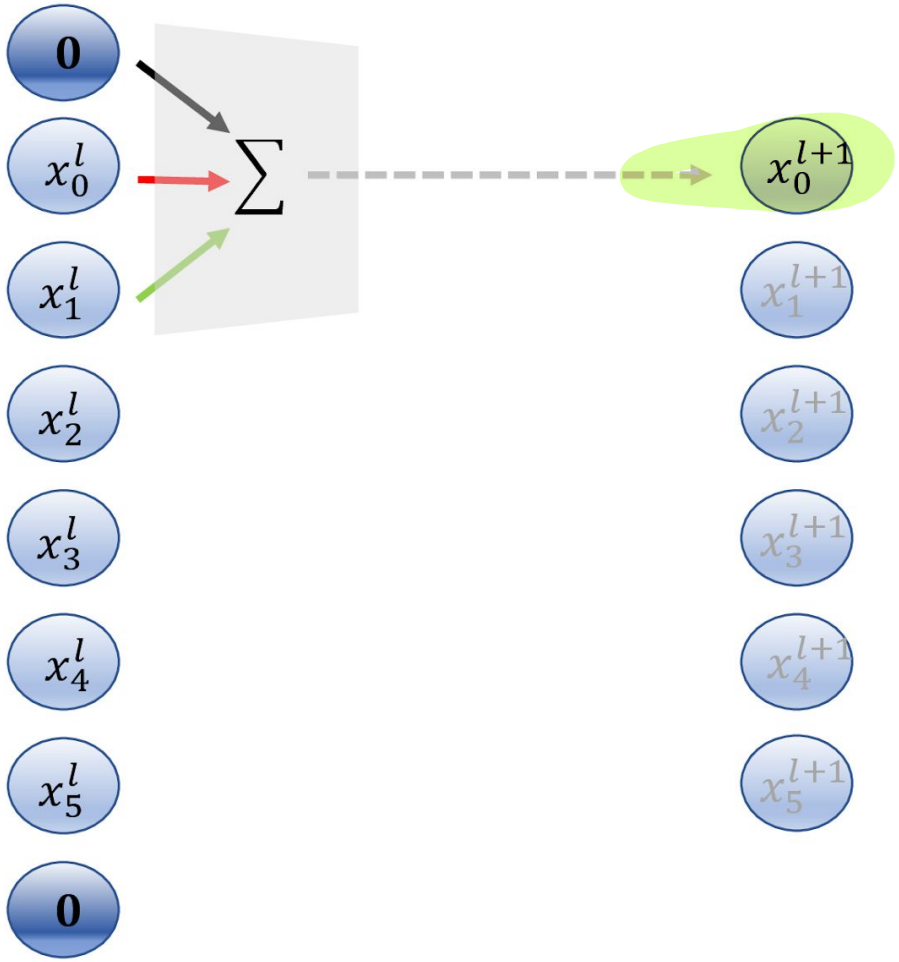
Padding (Key idea)



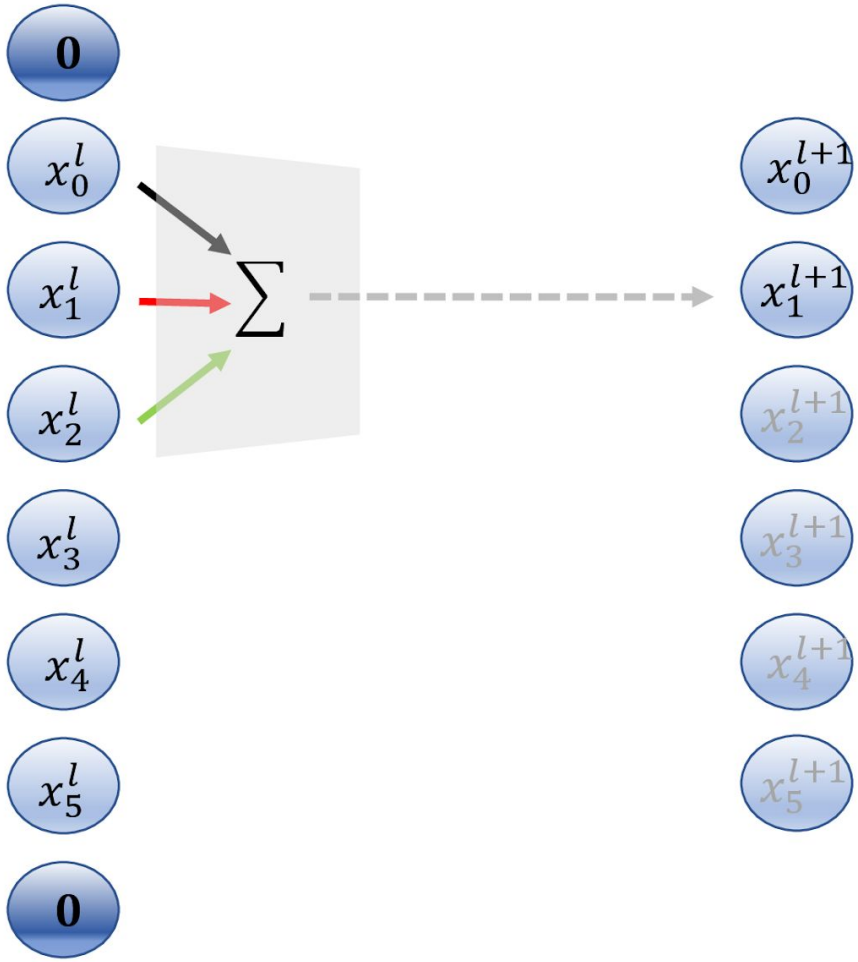
Padding (Key idea)



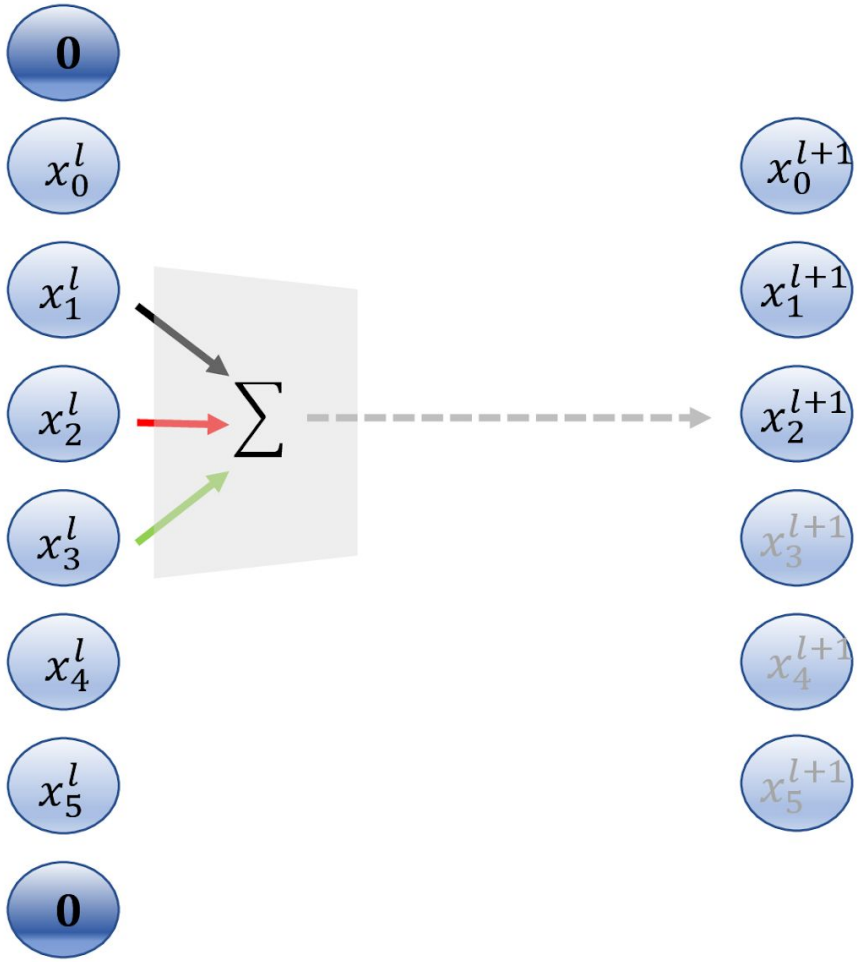
Padding (Key idea)



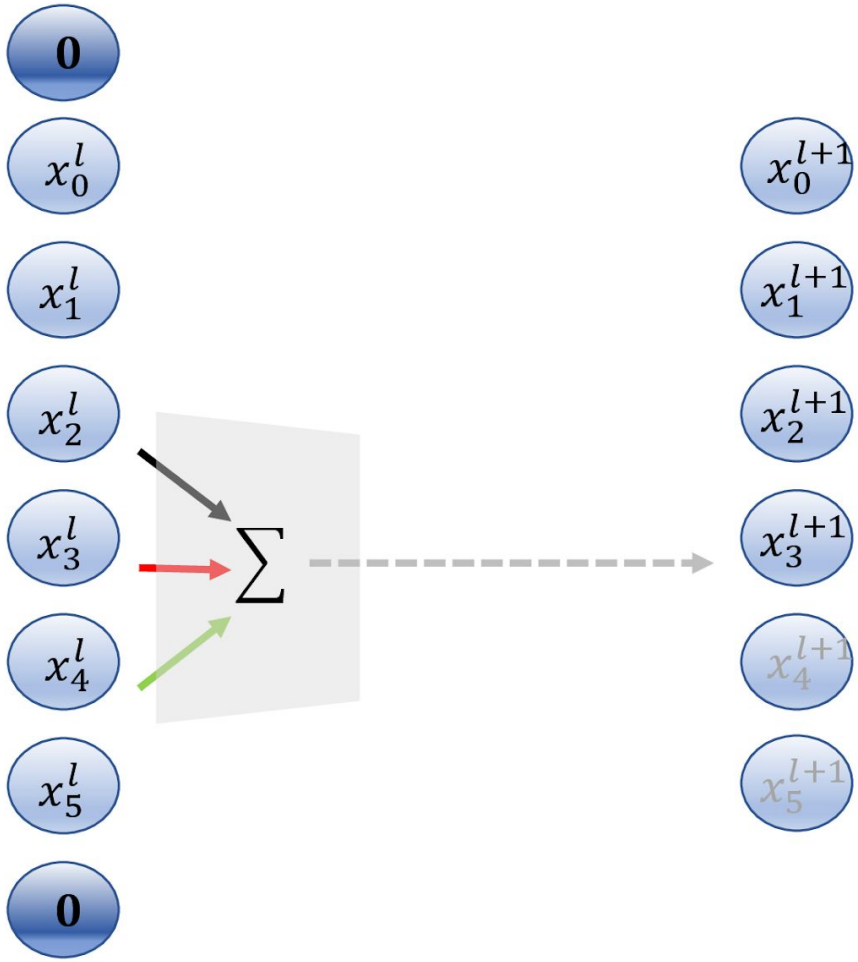
Padding (Key idea)



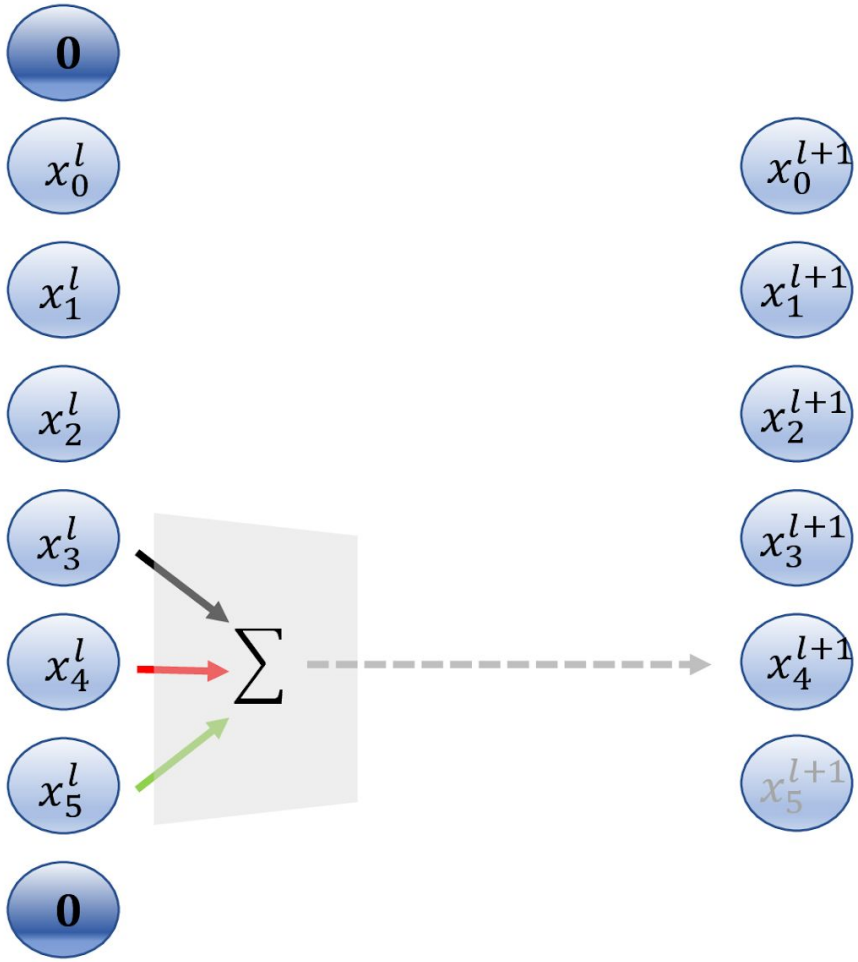
Padding (Key idea)



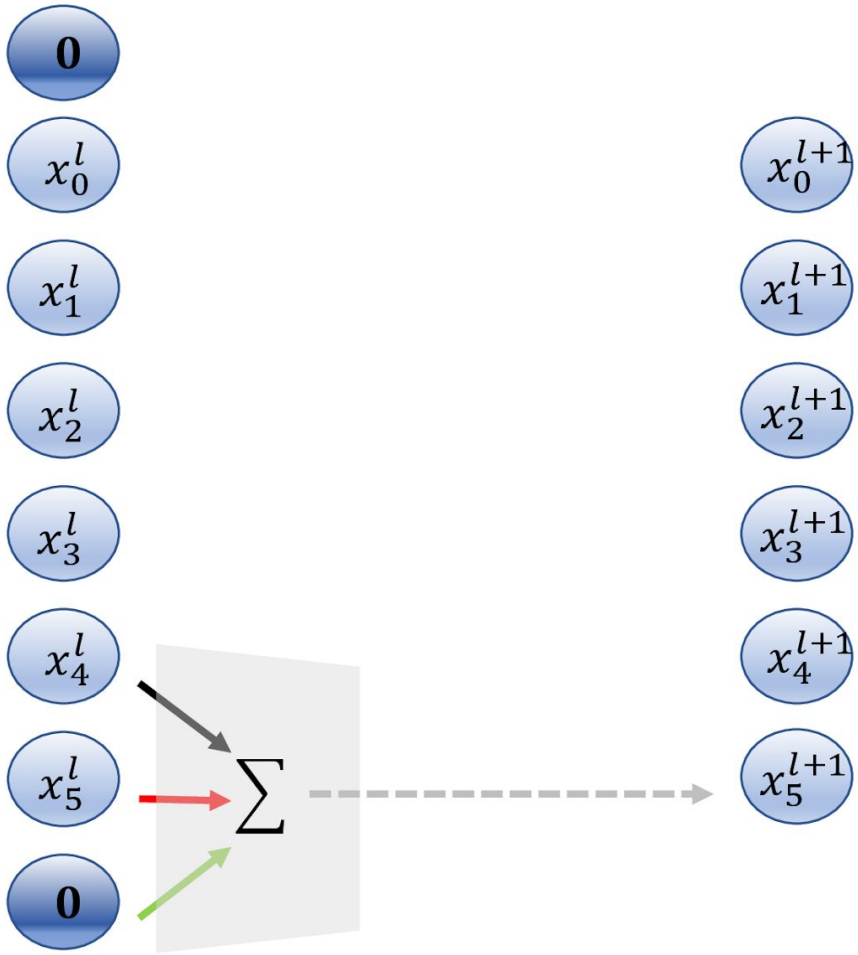
Padding (Key idea)



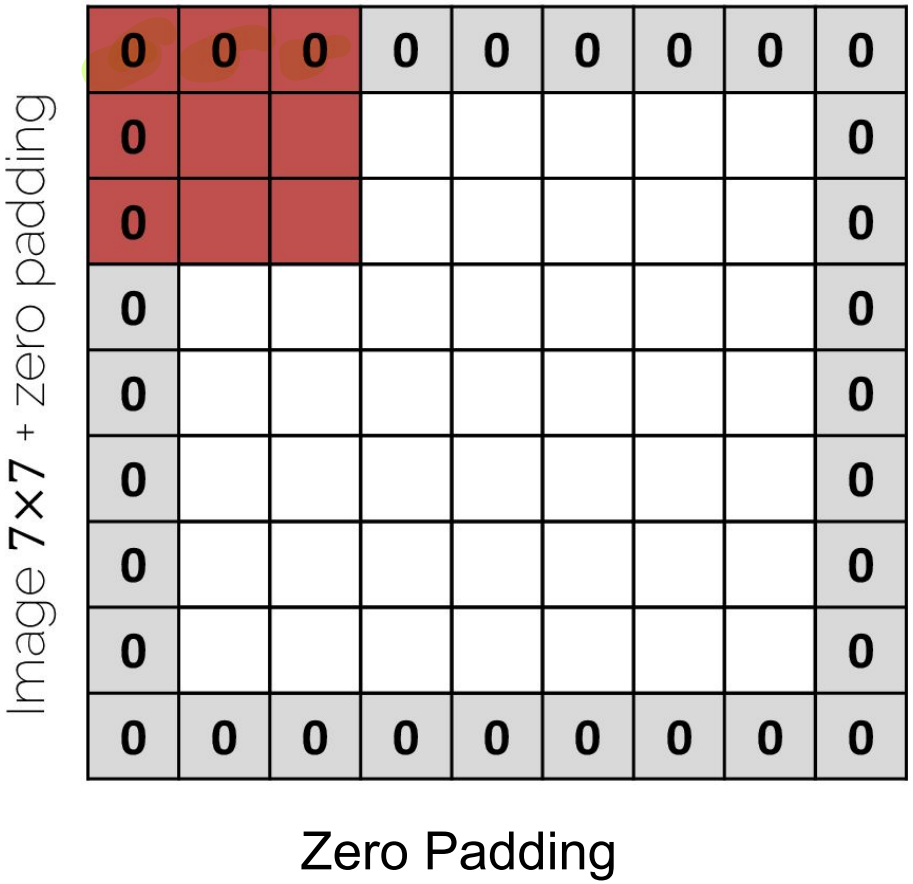
Padding (Key idea)



Padding (Key idea)

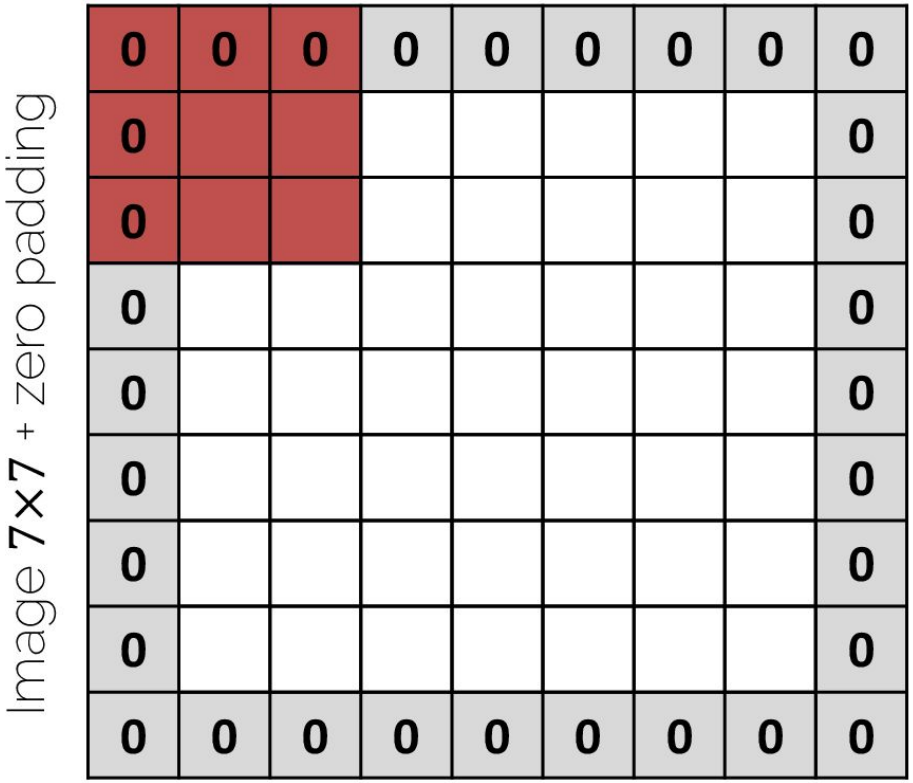


Convolution Layers: Padding



- Padding refers to the process of adding additional rows and columns of zeros around the edges of the input image.

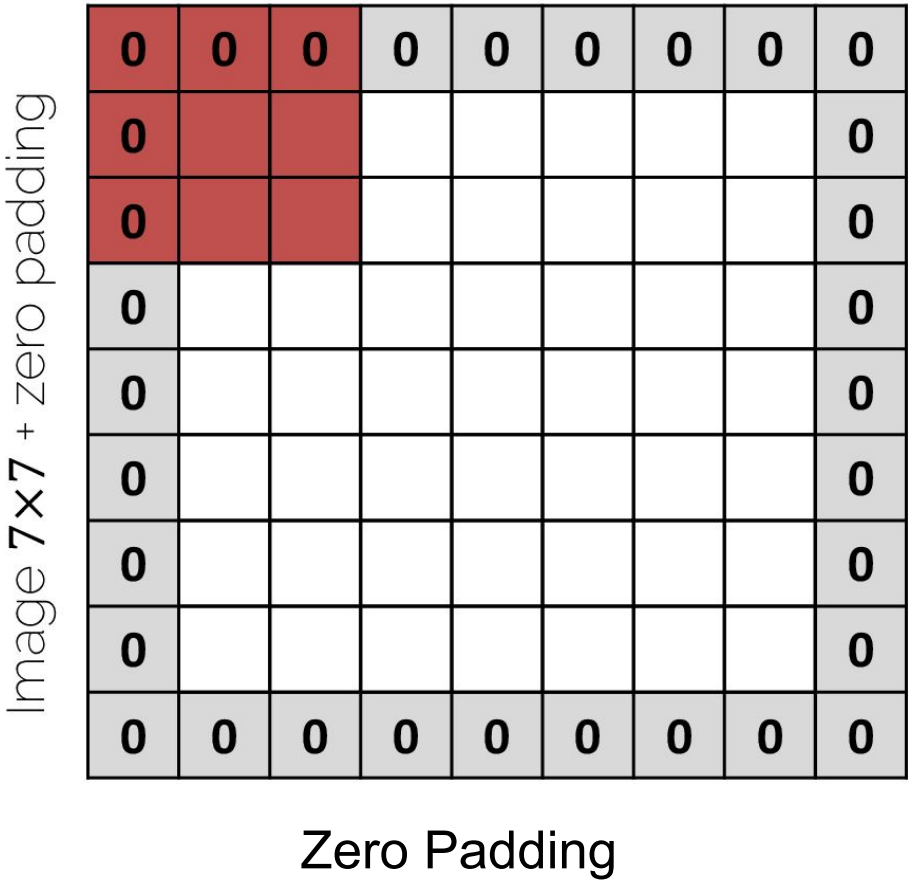
Convolution Layers: Padding



Why padding?

- Sizes get smaller too quickly
- Corner pixel is only used once

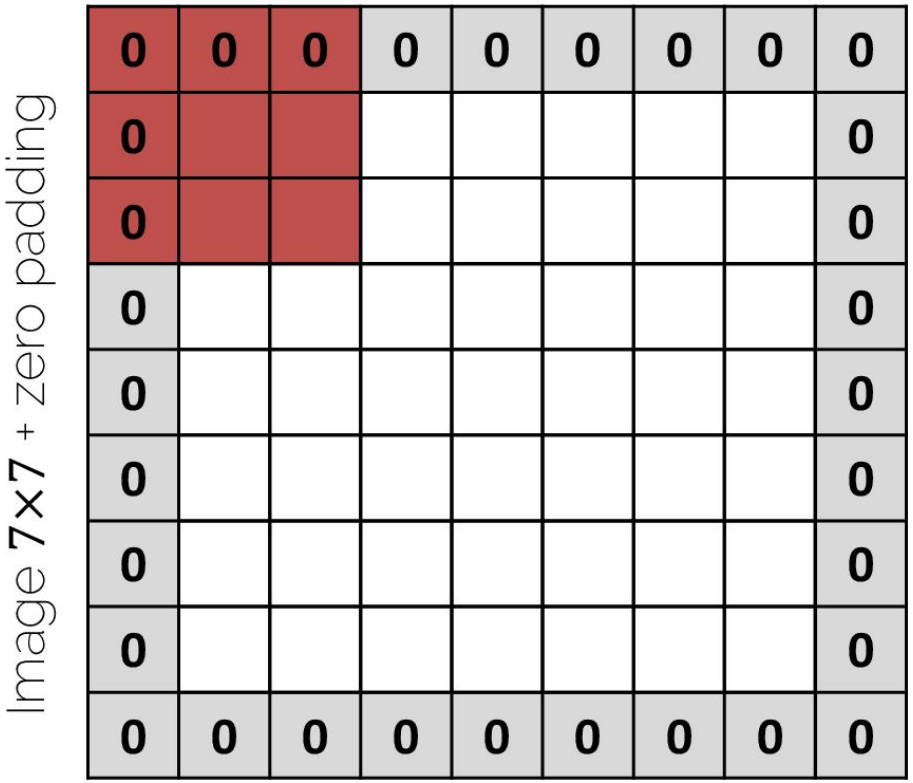
Convolution Layers: Padding



Zero Padding

- Preserving the spatial dimensions of the output feature map
- Reducing information loss at the edges of the image
- Reducing the effect of edge pixels on the output

Convolution Layers: Padding



Input $(N \times N)$: 7×7
Filter $(F \times F)$: 3×3
Padding (P) : 1
Stride (S) : 1
Output 7×7

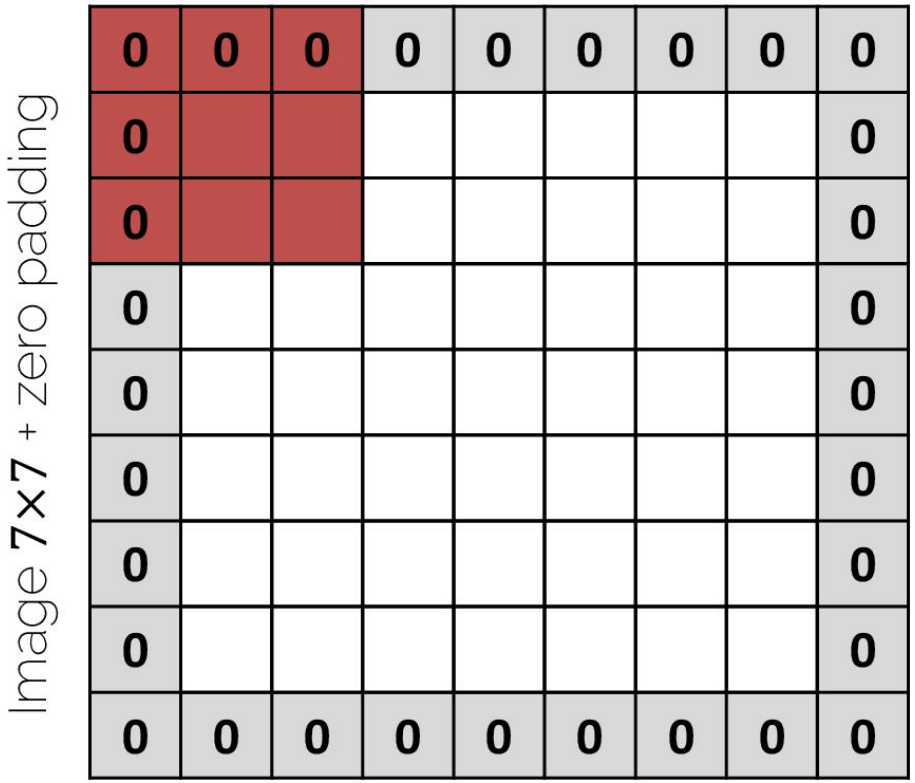
Most common is zero padding

$$\frac{N+2p-f}{S}+1$$

$$7-3+2+1$$

$$= 7$$

Feature Map Dimension



Input $(N \times N)$: 7×7
Filter $(F \times F)$: 3×3
Padding (P): 1
Stride (S): 1
Output 7×7

Most common is zero padding

Output Size:

$$\left(\left\lfloor \frac{N + 2 \cdot P - F}{S} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{N + 2 \cdot P - F}{S} \right\rfloor + 1\right)$$

$\lfloor \ \rfloor$ denotes the floor operator

valid padding is no padding at all.
Same " is that feature maps size is

Same as the input image size

Padding Types

- **Valid Padding**:
  - **No padding** at all.
  - The output feature map is smaller than the input feature map.

```
# Define the input layer
input_layer = Input(shape=(28, 28, 1))

# Create a convolutional layer with valid padding
conv_layer = Conv2D(32, (3, 3), padding='valid')(input_layer)
```

Padding Types

$N = N - F + 2P + 1$

$P = \frac{F-1}{2}$

```
# Create a convolutional layer with "same" padding
conv_layer = Conv2D(32, (3, 3), padding='same')(input_layer)
```

- **Same Padding**:
  - Adding enough padding to the input image so that the output feature map has the same size as the input image.

    Set padding to P = $\frac{F-1}{2}$ with stride S =1
  - Verify formula

$$\left(\left\lfloor \frac{N+2\cdot P-F}{S} \right\rfloor +1\right) \times \left(\left\lfloor \frac{N+2\cdot P-F}{S} \right\rfloor +1\right)$$

Padding

- **Reflective Padding:**
  - The padded pixels are not filled with zeros, but with the reflected values of the input image.
  - This type of padding is useful when the input image contains edges or other sharp features that would be distorted by zero padding.

```
Original tensor
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
Reflected padded tensor
[[5 4 5 6 5]
 [2 1 2 3 2]
 [5 4 5 6 5]
 [8 7 8 9 8]
 [5 4 5 6 5]]
```

Padding

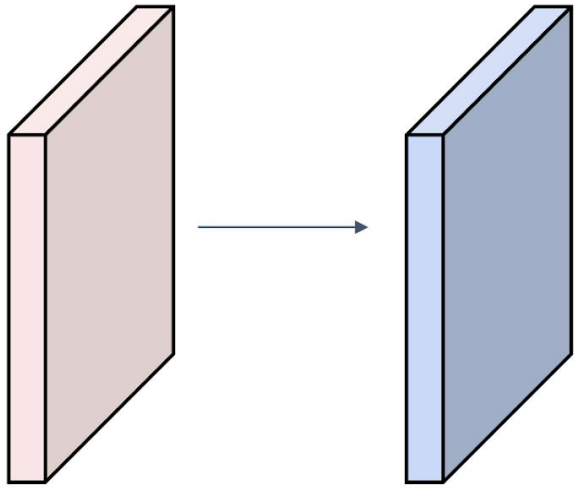
- **Symmetric Padding:**

```
Original tensor
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
Symmetric padded tensor:
[[1 1 2 3 3]
 [1 1 2 3 3]
 [4 4 5 6 6]
 [7 7 8 9 9]
 [7 7 8 9 9]]
```

Feature Map Dimension (padding and stride)

Convolution Example



Input volume: 3 x 32 x 32

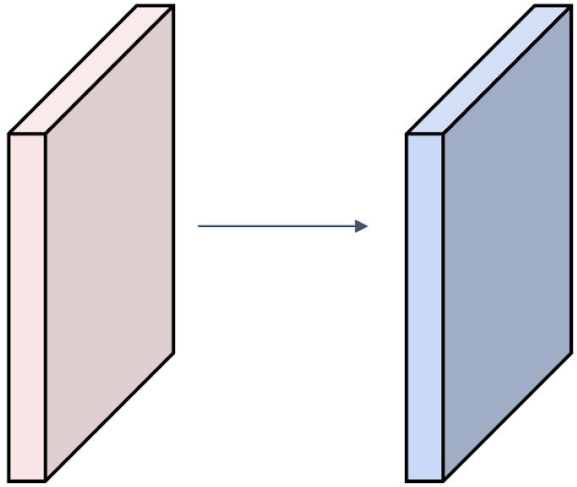
10 5x5 filters with stride 1, pad 2

```
Conv2D(filters=10, kernel_size=(5, 5), strides=1, padding='same', input_shape=(3, 32, 32))
```

$32 - 5 + 1 = 28$

$28 \times 28 \times 10$

Convolution Example



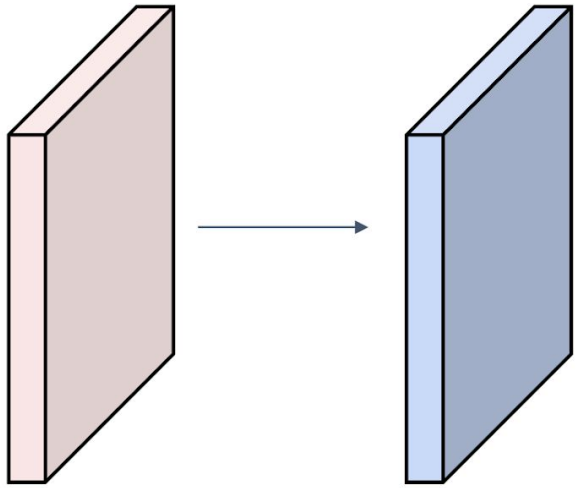
Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: ?

$32 - 5 + 4 + 1$

$= 32 \times 32 \times 10$

Convolution Example



Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

Output volume size:

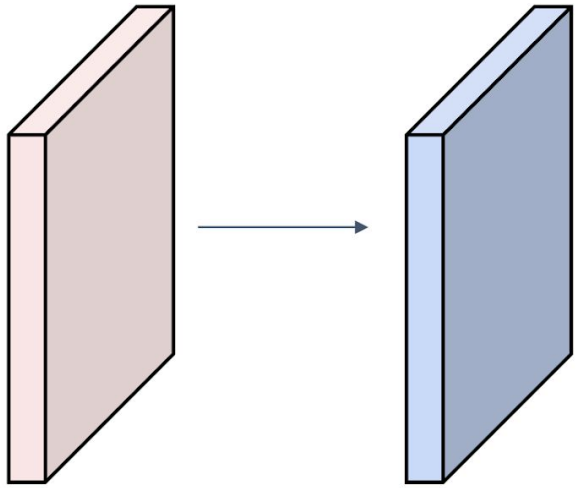
(32+2*2-5)/1+1 = 32 spatially, so

10 x 32 x 32

32 - 5 + 4 + 1

= 32

Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

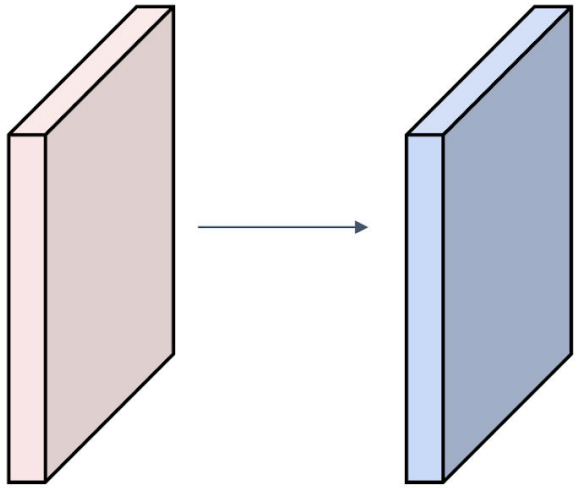
Output volume size: 10 x 32 x 32
Number of learnable parameters: ?

$10(5 \times 5 \times 3 + 1)$

$10\ (5 \times 5 \times 3 + 1)$

$= 760$

Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: **760**
Parameters per filter: 3*5*5 + 1 (for bias) = 76
10 filters, so total is 10 * 76 = **760**

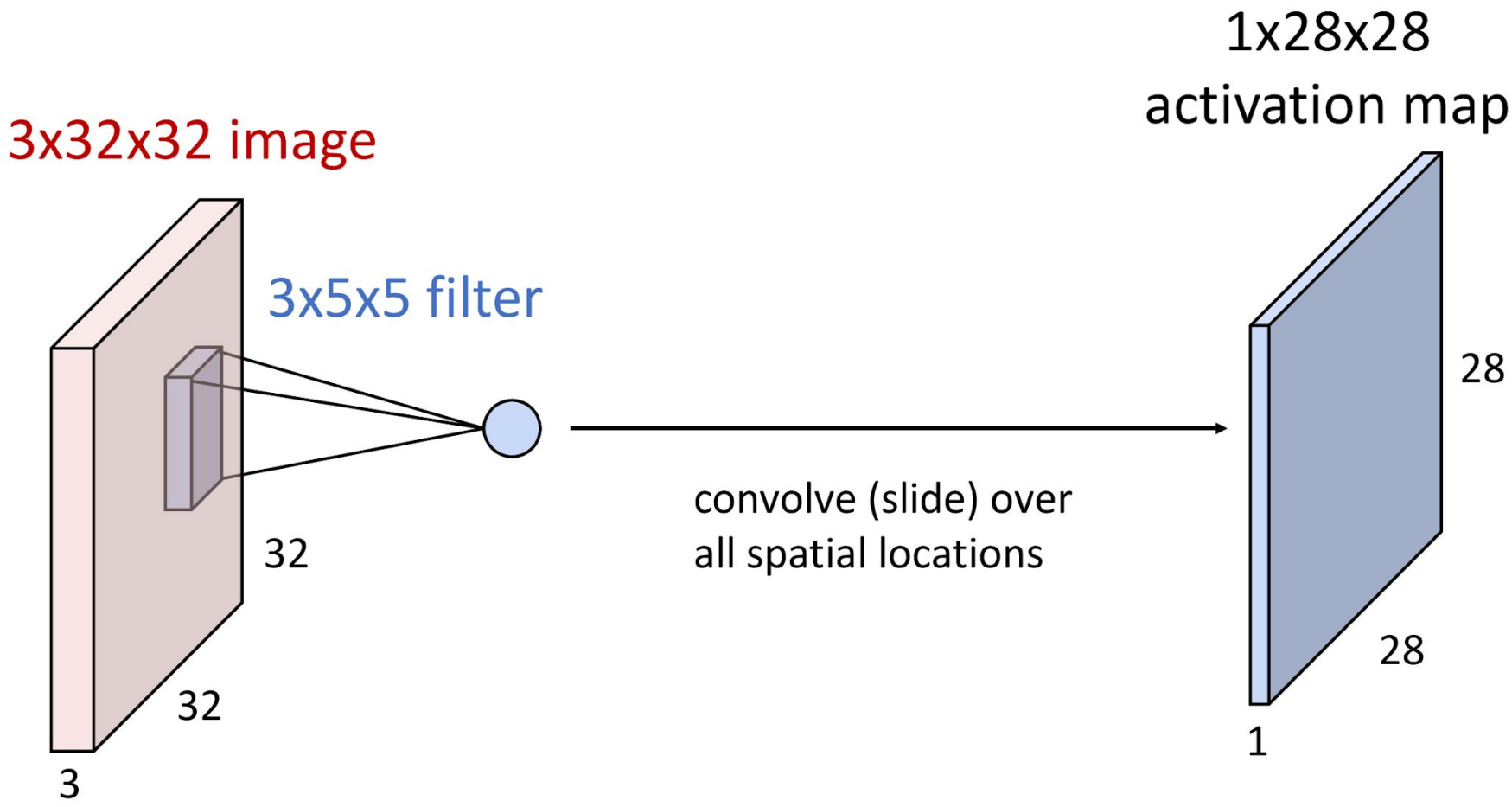
Stacking Convolution Layers

Convolution Layer

- A basic layer is defined by
  - — Filter width and height *(depth is implicitly given)*
  - — Number of different filter (#weight sets)

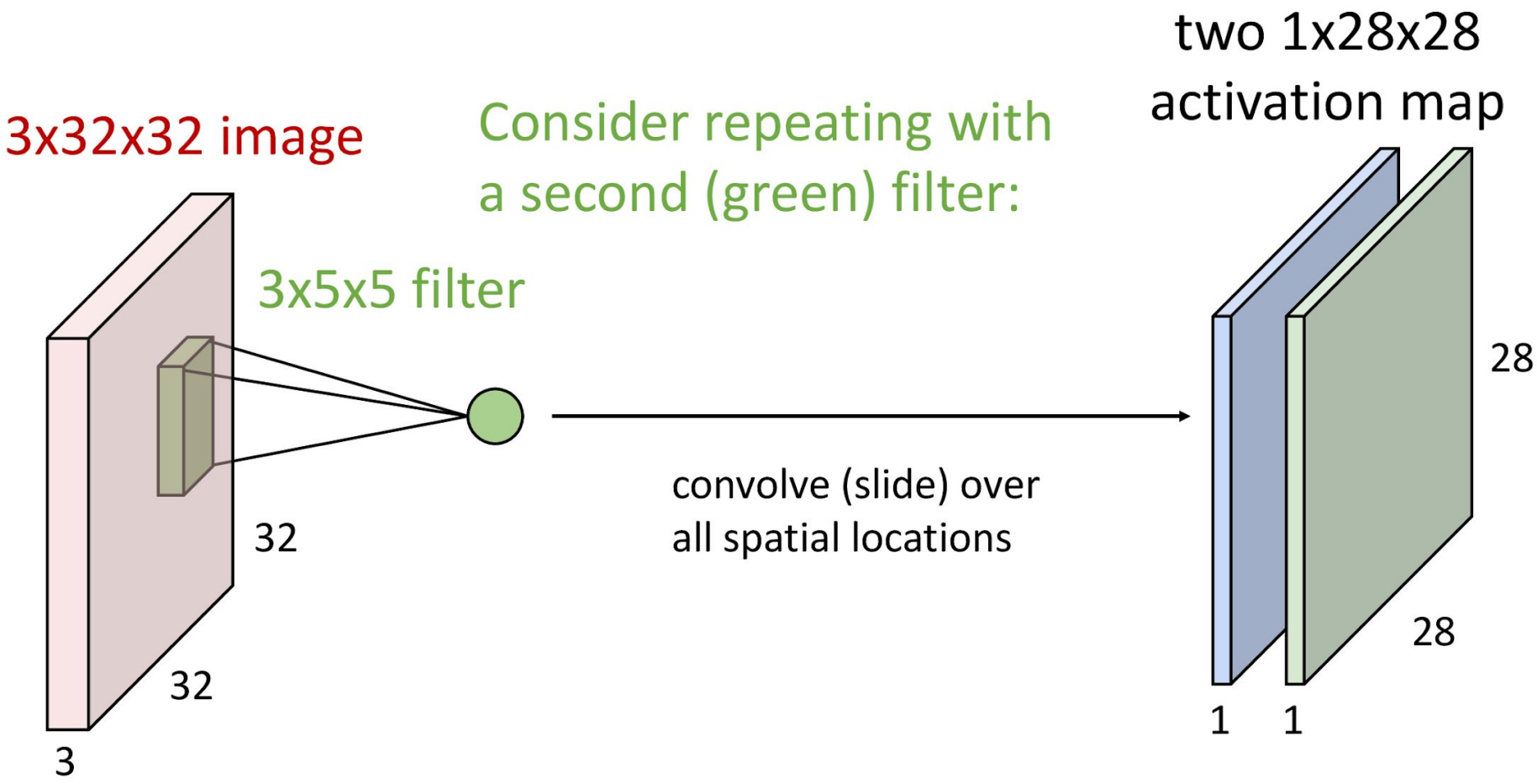
- Each filter captures a different image characteristic

Convolution Layer

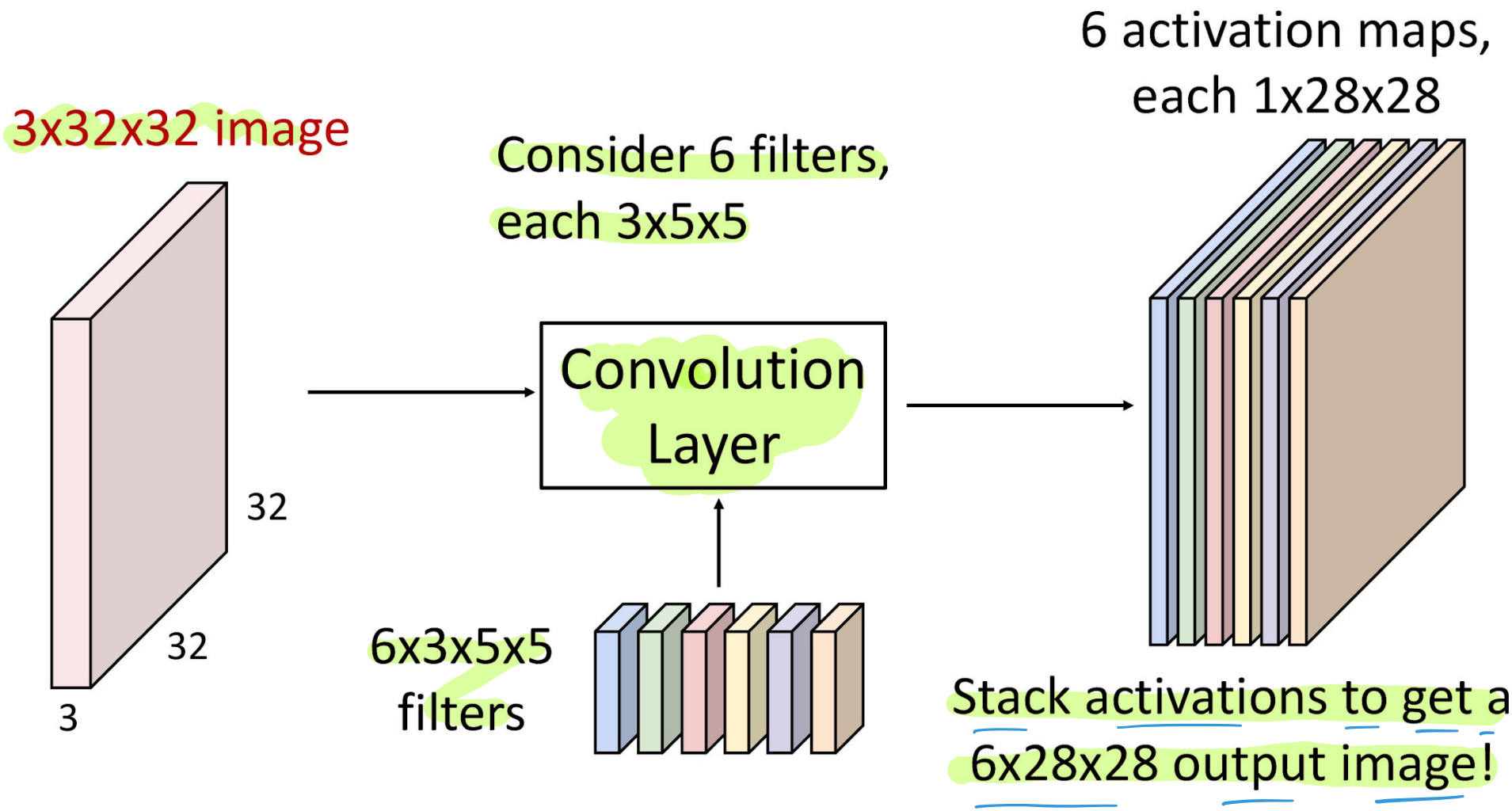



```
Conv2D(filters=1, kernel_size=(5, 5), strides=1, padding='valid', input_shape=(3, 32, 32))
```

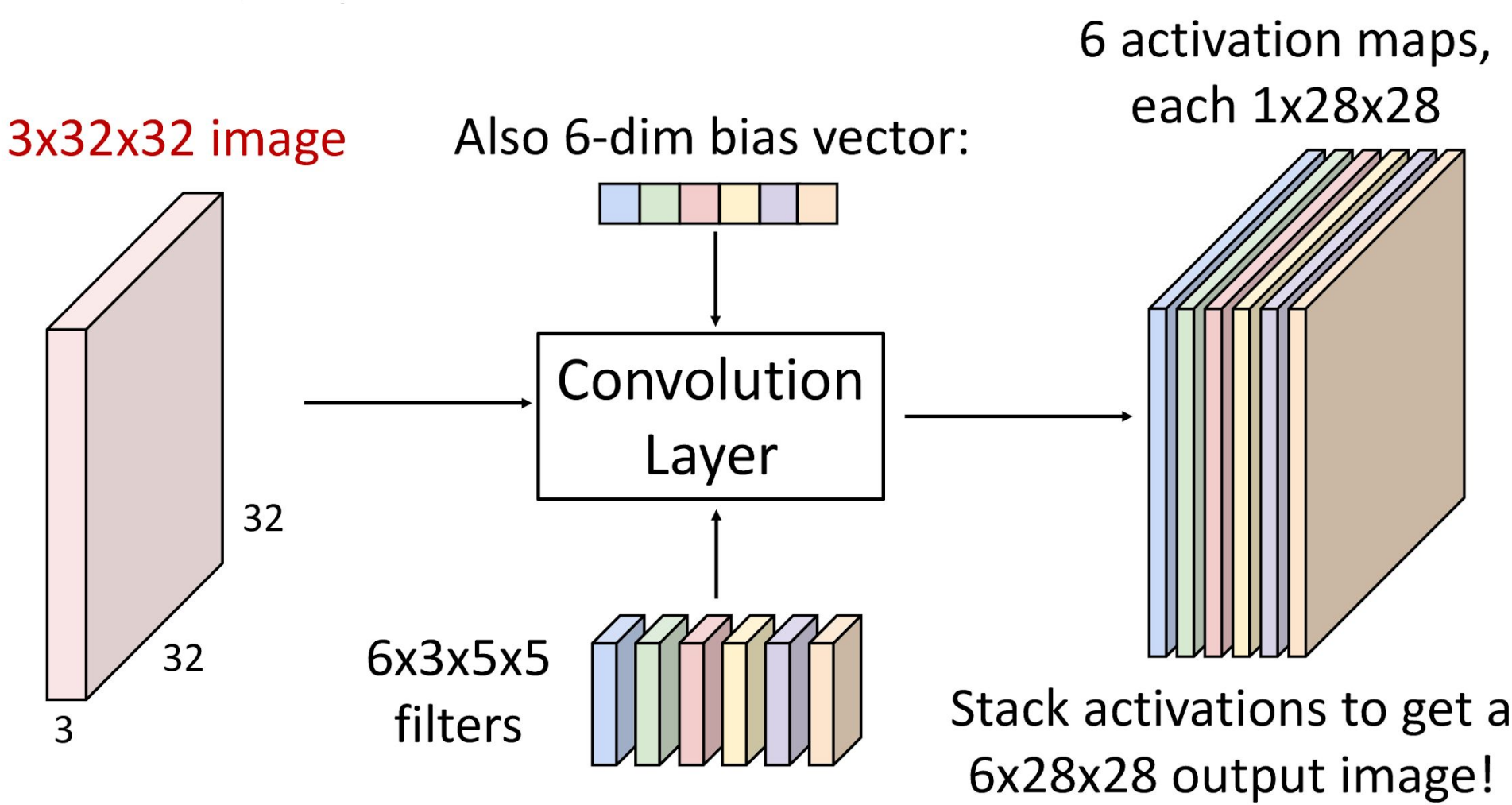
Convolution Layer



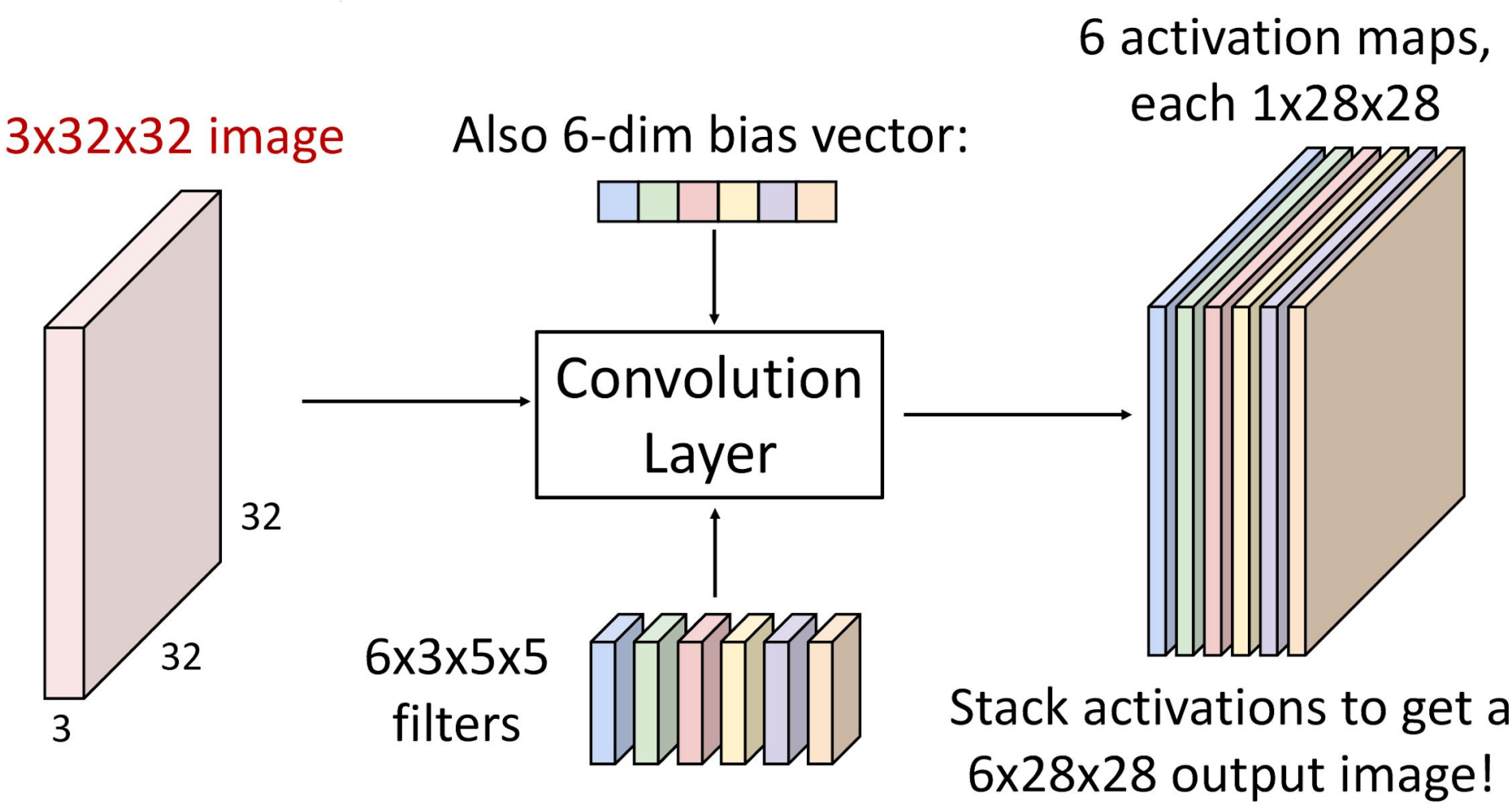
Convolution Layer



Convolution Layer

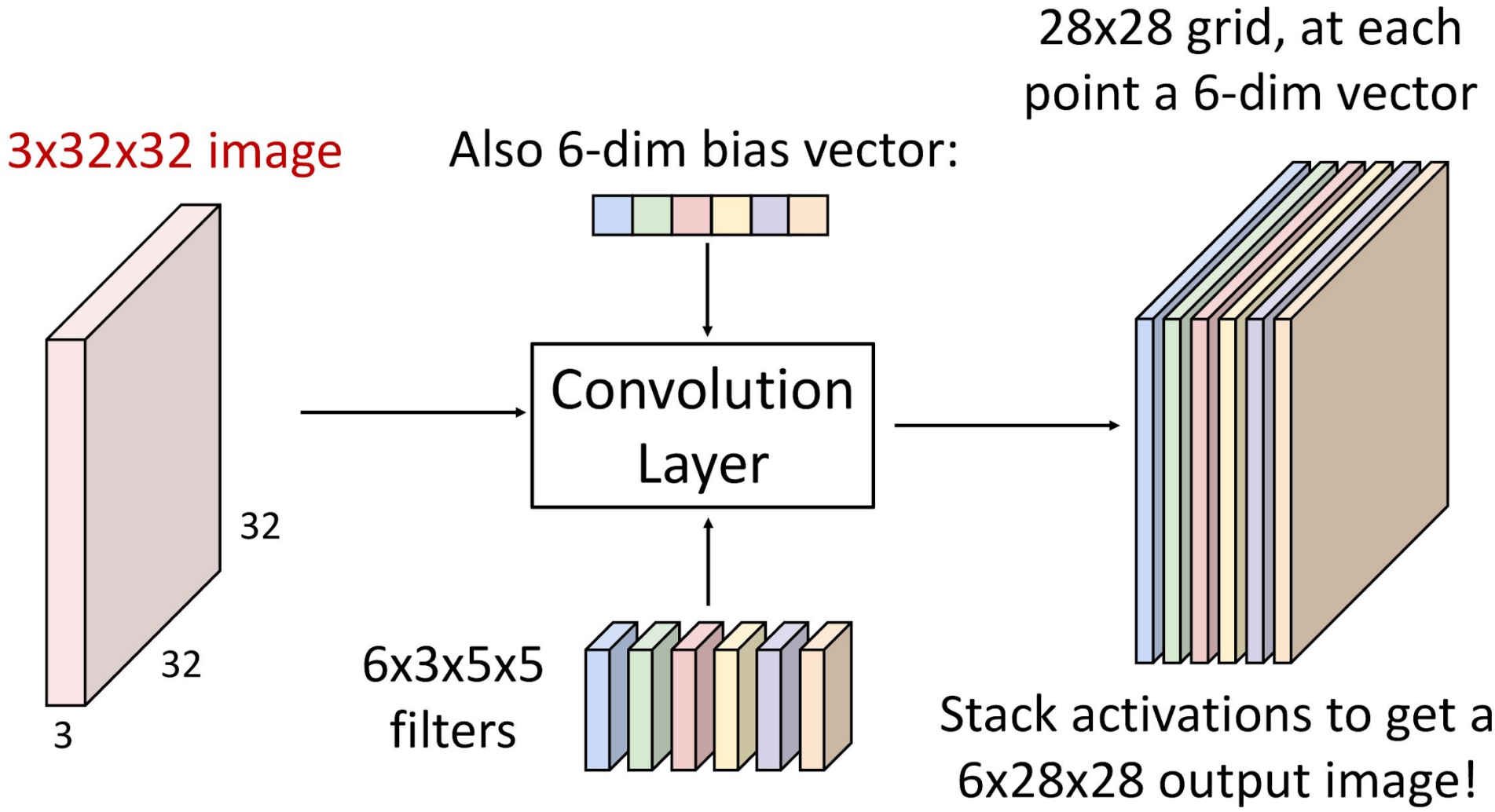


Convolution Layer

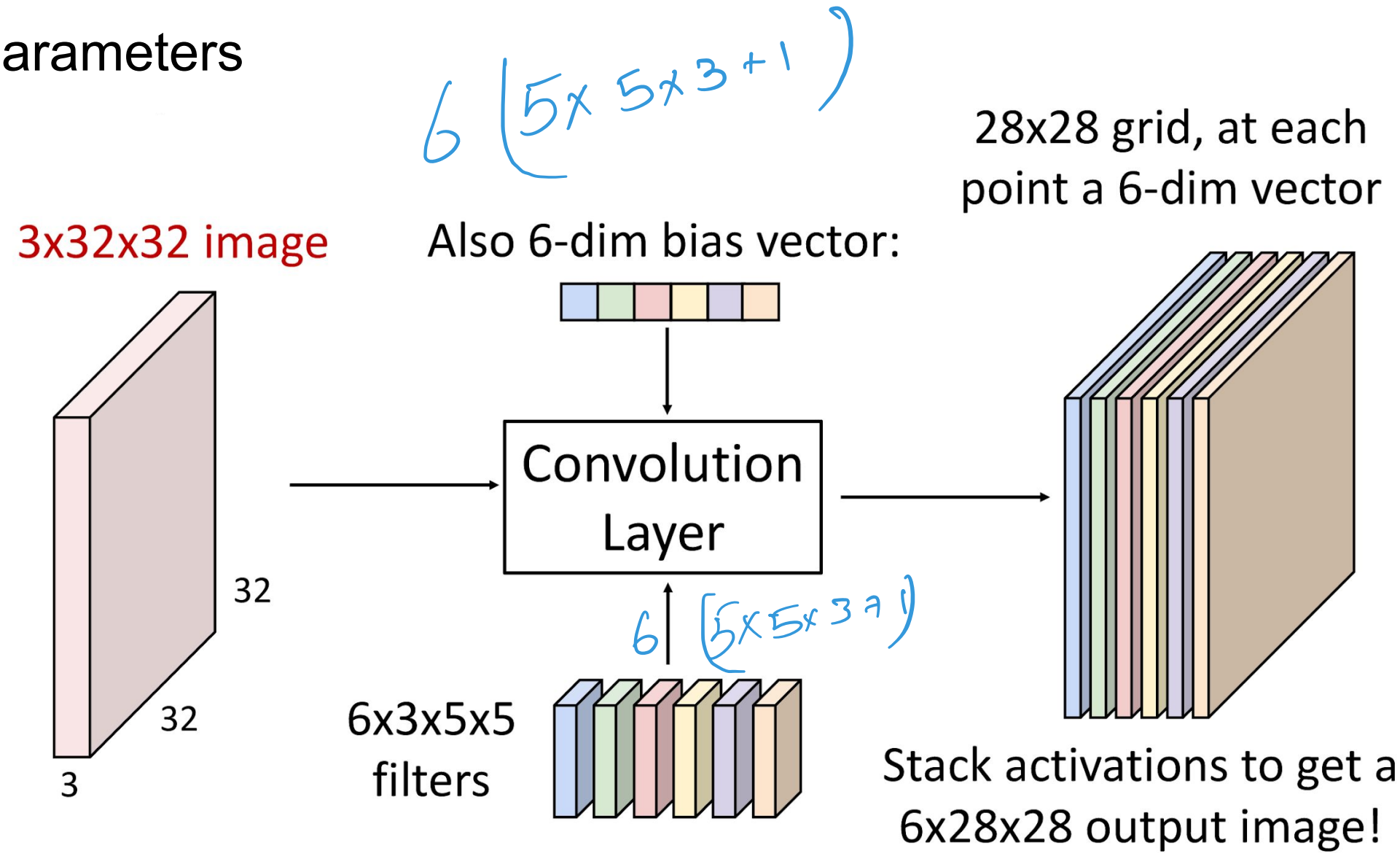



```
Conv2D(filters=6, kernel_size=(5, 5), strides=1, padding='valid', input_shape=(3, 32, 32))
```

Convolution Layer

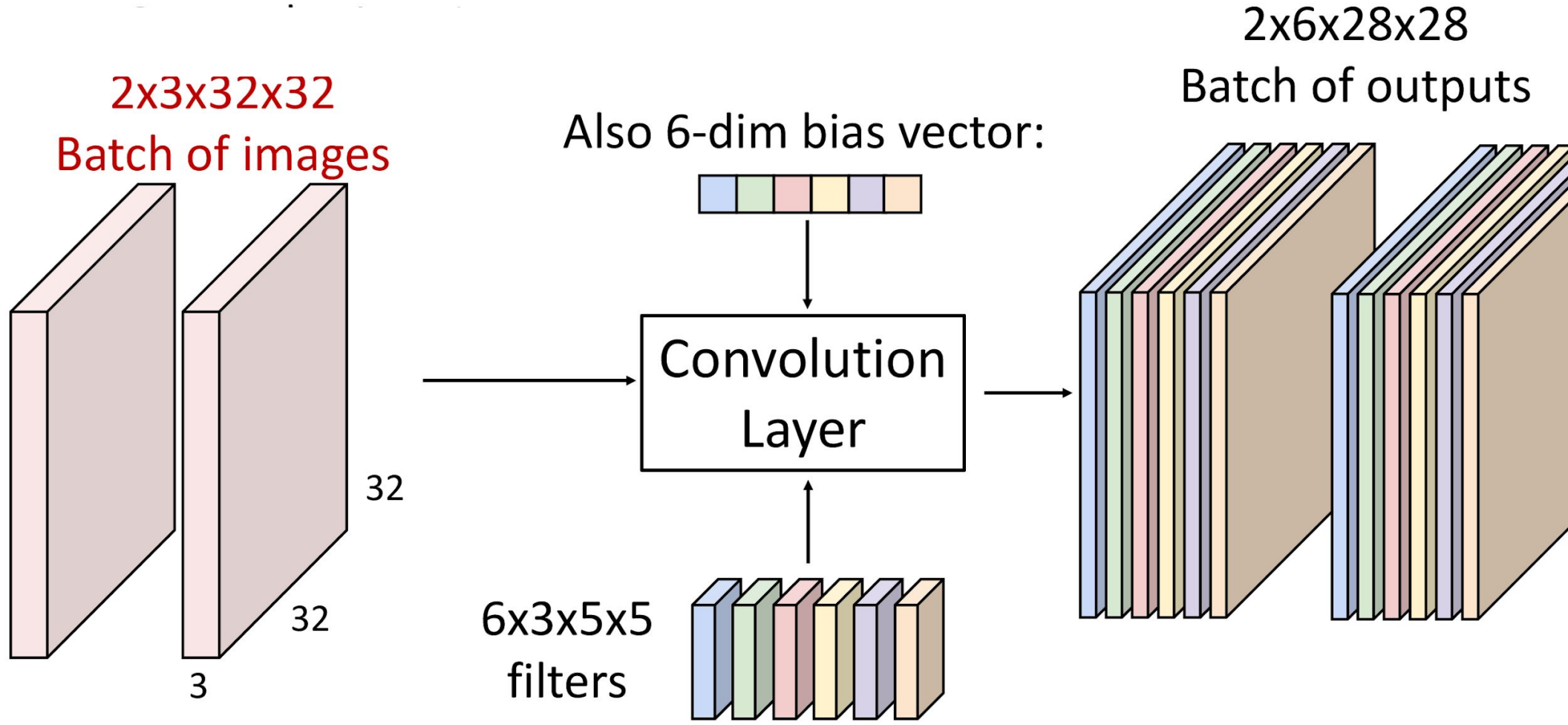


#parameters

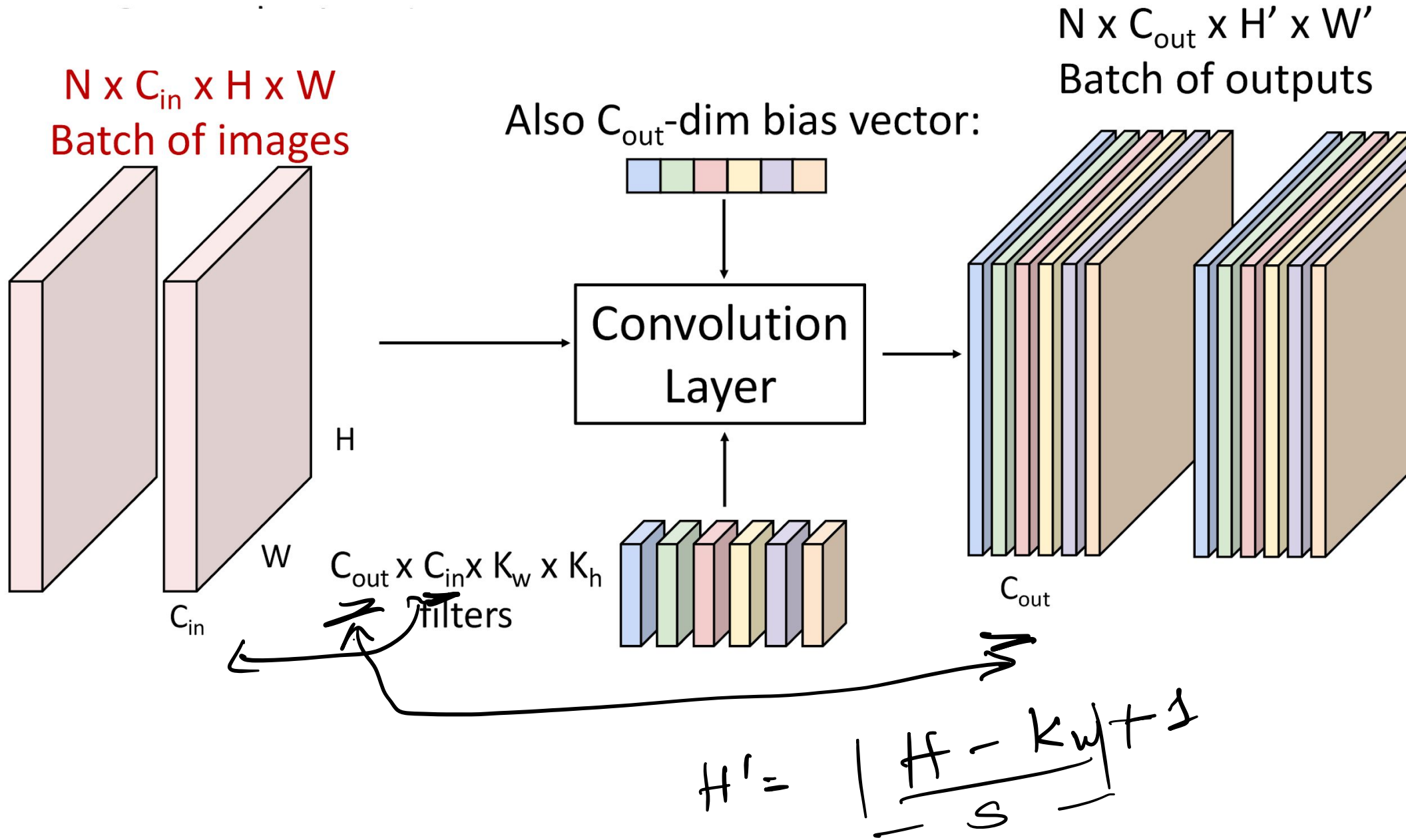


- Each filter has a size of 5x5x3 (since there are 3 input channels).
- There are 6 filters in the layer. Plus there is one bias term for each filter.
- Therefore, the total number of parameters is **6*(5x5x3+1) = 456**.

Convolution Layer

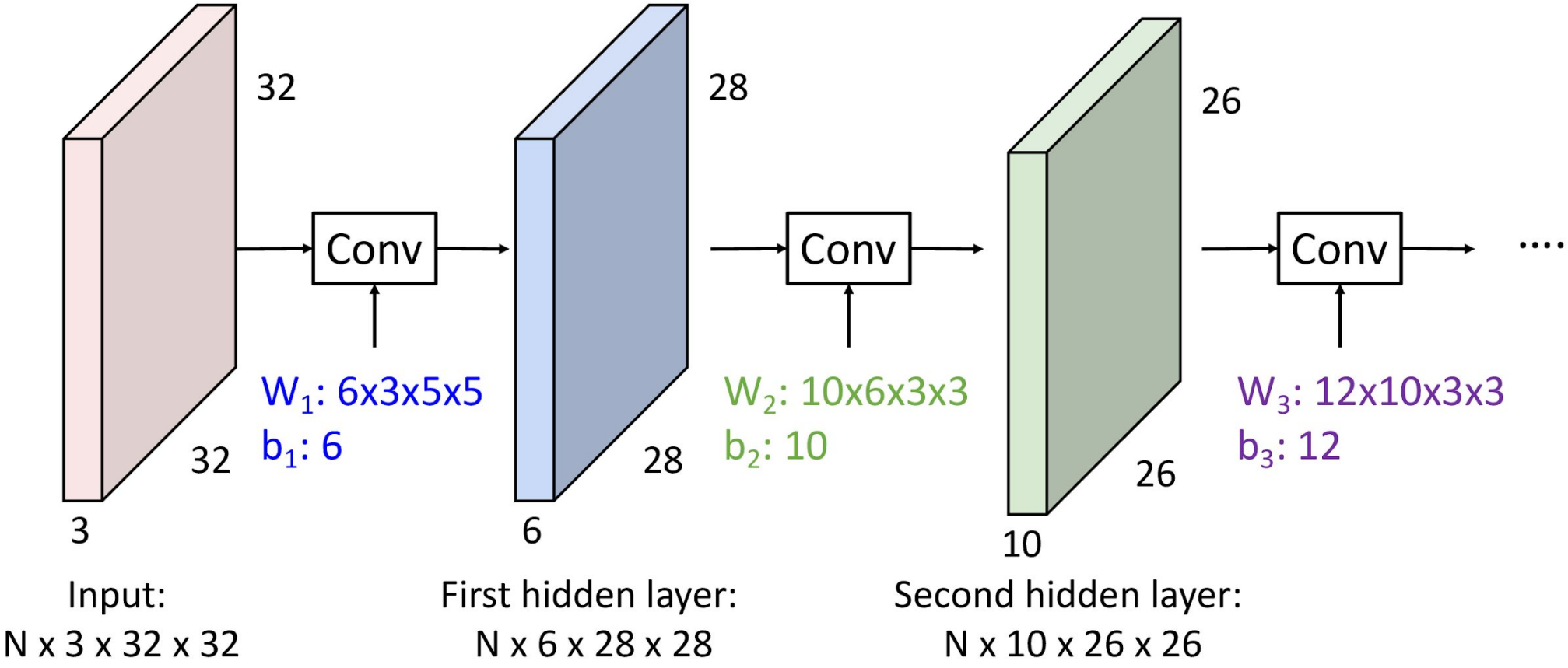


Convolution Layer



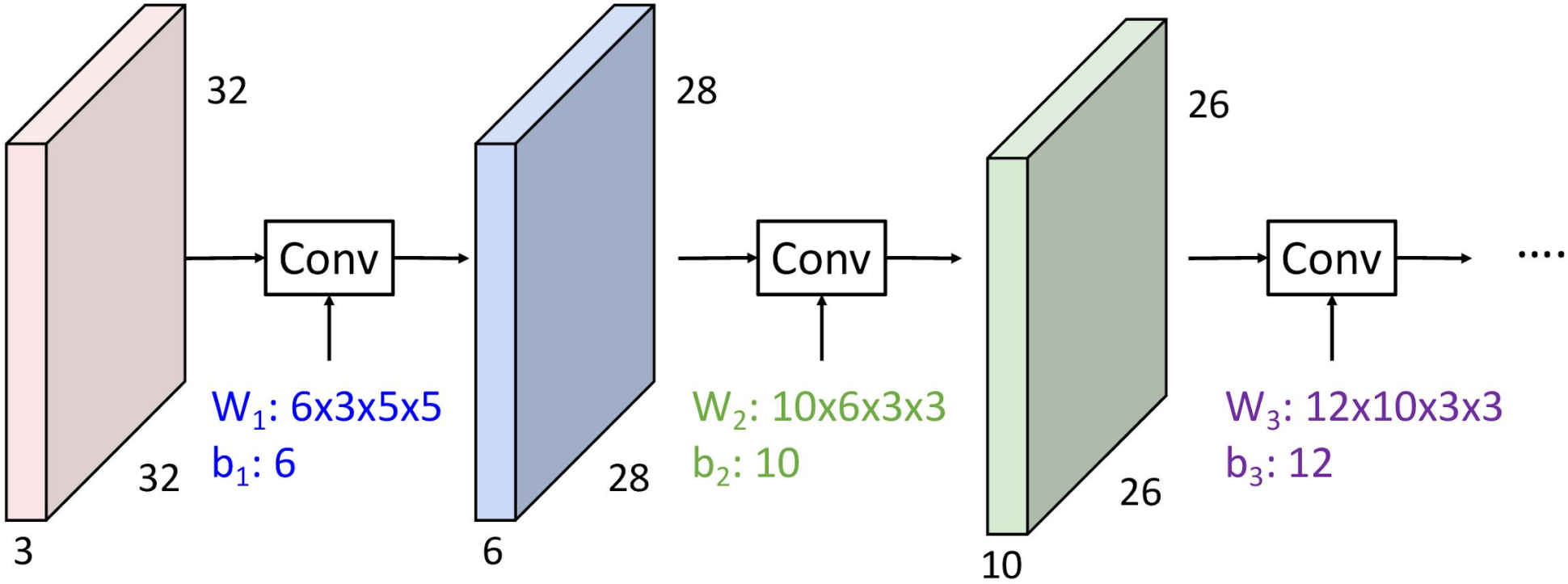
$$H' = \left| \frac{H - K_w}{S} + 1 \right|$$

Stacking Convolution layers



Stacking Convolution layers

Q: What happens if we stack two convolution layers?



Input:
N x 3 x 32 x 32

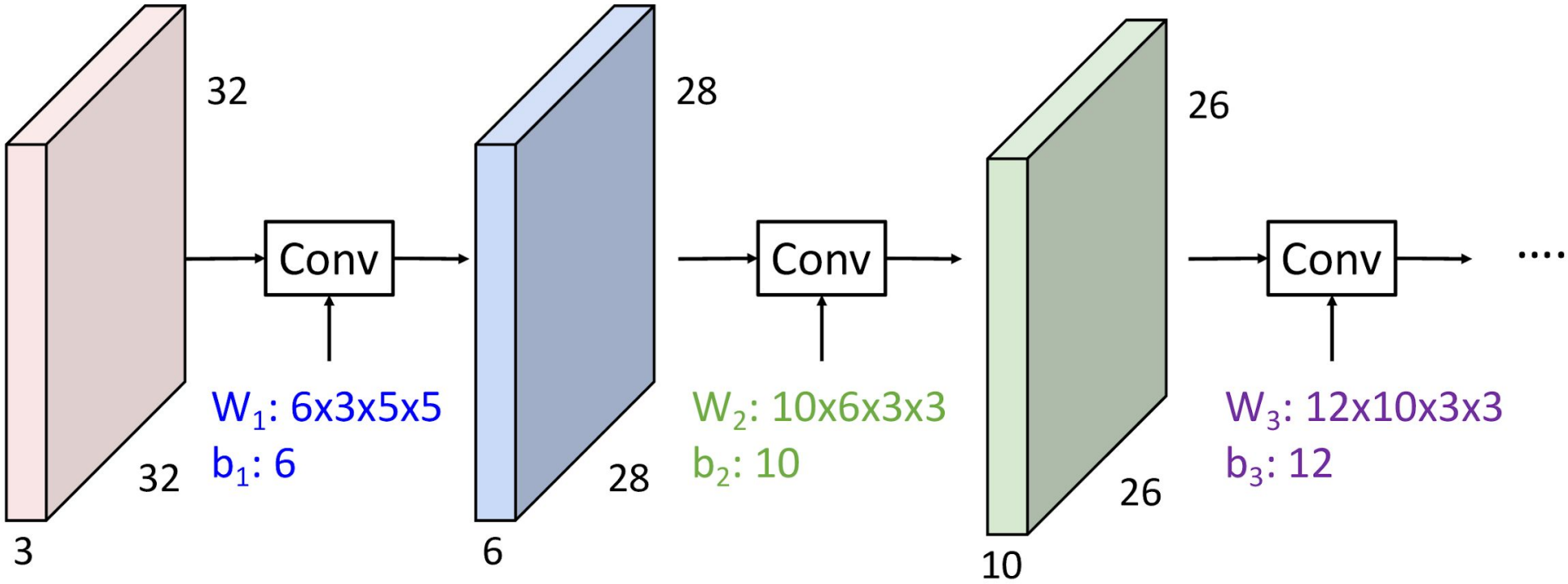
First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Stacking Convolution layers

Q: What happens if we stack two convolution layers?

A: We get another convolution!



Input:
N x 3 x 32 x 32

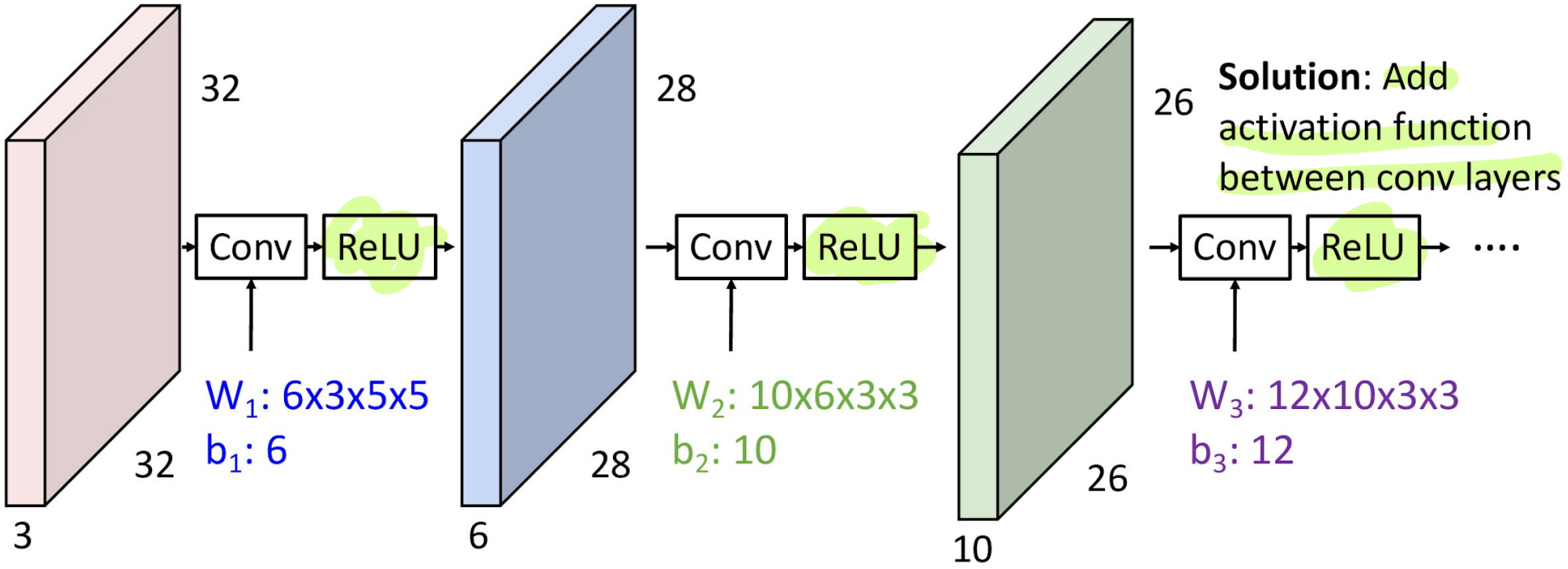
First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Stacking Convolution layers

Q: What happens if we stack two convolution layers?

A: We get another convolution!



Solution: Add activation function between conv layers

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Pooling
(sub-sampling)

Pooling

- **Conv Layer** = Feature Extraction
  - — Computes a feature in a given region
- **Pooling Layer** = Feature Selection
  - — Picks the strongest activation in a region

Pooling Layer: Max Pooling

3	1	3	5
6	0	7	9
3	2	1	4
0	2	4	3

6	9
3	4

```
inputs = Input(shape=(4, 4, 1))

# Max pooling layer
max_pool = MaxPooling2D(pool_size=(2, 2), strides=2)(inputs)
```