Deep learning for Computer Vision

Pooling
(sub-sampling)

Pooling Layer: Max Pooling

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

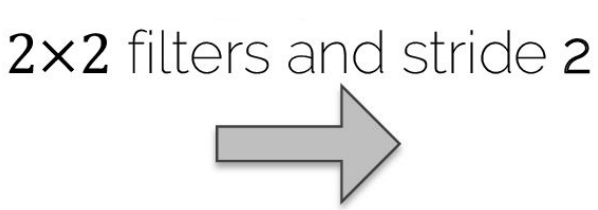
6	9
3	4

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

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

Pooling Layer: Average Pooling

- Typically used deeper in the network

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



2.5	6
1.75	3

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

# Average pooling layer
avg_pool = AveragePooling2D(pool_size=(2, 2), strides=2)(inputs)
```

Conv. Activation Pooling

Pooling Layer

Two hyperparams ↓ Filter and stride.

- Input is a volume of size $W_{in} \times H_{in} \times D_{in}$
- Two hyper-parameters

  Spatial filter extent F

  Stride S

Output volume is of size $W_{\text{out}} \times H_{\text{out}} \times D_{\text{out}}$

- $W_{\text{out}} = \frac{W_{in} - F + 2P}{S} + 1$
- $H_{\text{out}} =$
- $D_{\text{out}} =$

Does not contain parameters; e.g. it's fixed function

Pooling Layer

- Input is a volume of size $W_{in} \times H_{in} \times D_{in}$
- Two hyper-parameters

  Spatial filter extent F

  Stride S

Output volume is of size $W_{\text{out}} \times H_{\text{out}} \times D_{\text{out}}$

Oh, It's about the pooling layer.

$$- W_{\text{out}} = \frac{W_{in} - F}{S} + 1$$

$$- H_{\text{out}} = \frac{H_{in} - F}{S} + 1$$

$$- D_{\text{out}} = D_{in}$$

Does not contain parameters; e.g. it's fixed function

Pooling Layer

- Input is a volume of size $W_{in} \times H_{in} \times D_{in}$
- Two hyper-parameters

  Spatial filter extent F

  Stride S

Common Setting
F=2, ***S=2***
F=3, ***S=2***

Output volume is of size $W_{\text{out}} \times H_{\text{out}} \times D_{\text{out}}$

$$- W_{\text{out}} = \frac{W_{in} - F}{S} + 1$$

$$- H_{\text{out}} = \frac{H_{in} - F}{S} + 1$$

$$- D_{\text{out}} = D_{in}$$

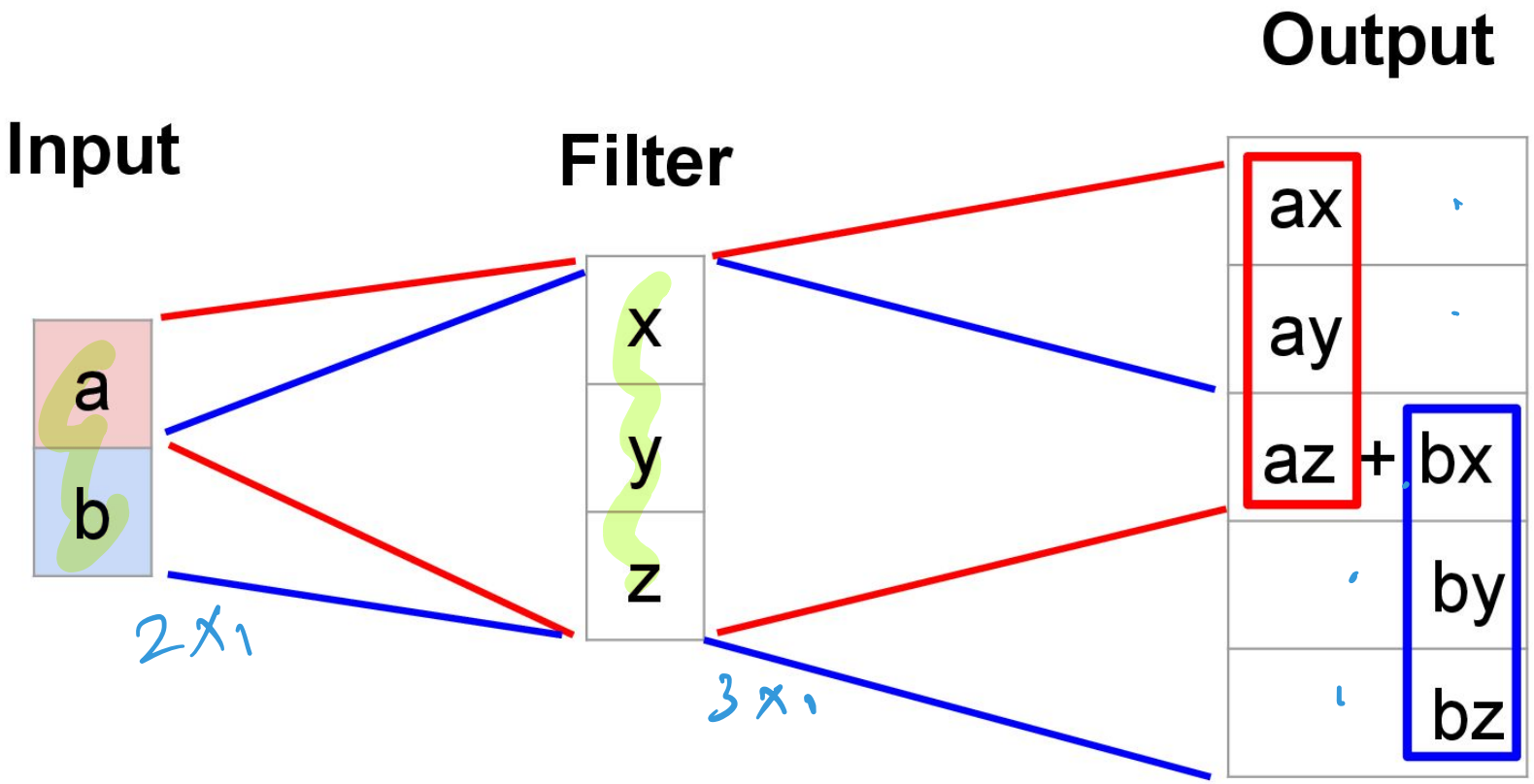
Does not contain parameters; e.g. it's fixed function

Pooling Layer

- Pooling Layer **reduces the spatial dimensions** of the input, which reduces the computational cost of the network.
- Pooling layers aimed to important features, which help the network to learn more **robust representations** of the input data.
- Pooling layers can improve the **translation invariance** of the network by selecting the maximum or average value in a given region.
- Pooling layers provides **distortion invariance** by selecting the maximum or average value in a region, which reduces effects of small variations.

Transpose Convolution: 1D Example

4 - 3 + 1 = 2



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

$\frac{n-f}{s}+1 = 2$

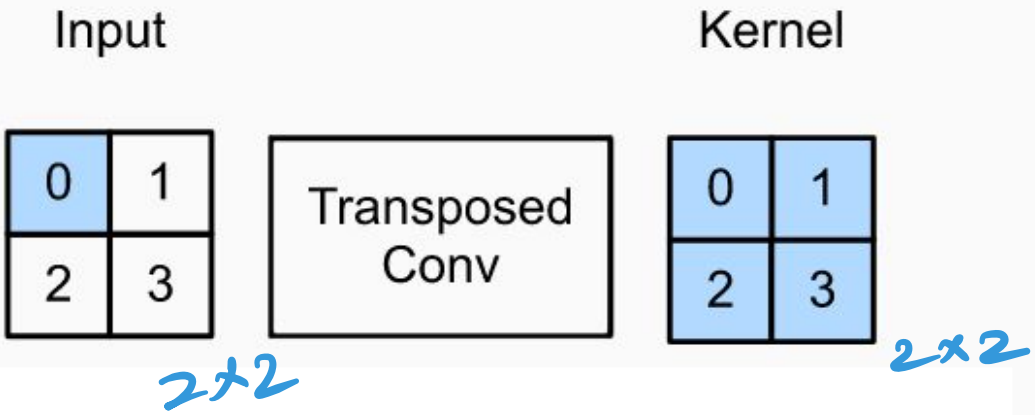
$n - 3 + 1 = 2$ ⇒ $n = 5$, too s = 2.

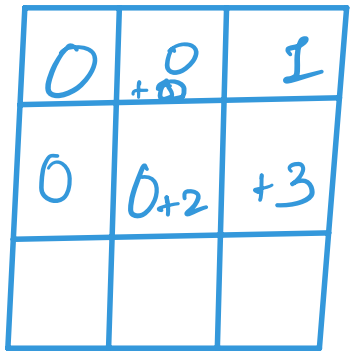
Transposed Convolution

- Convolution outputs tensor with a smaller spatial dimension, whereas transposed convolution outputs tensor with a larger spatial dimension.
- Like convolution, transposed convolution uses learnable weights.

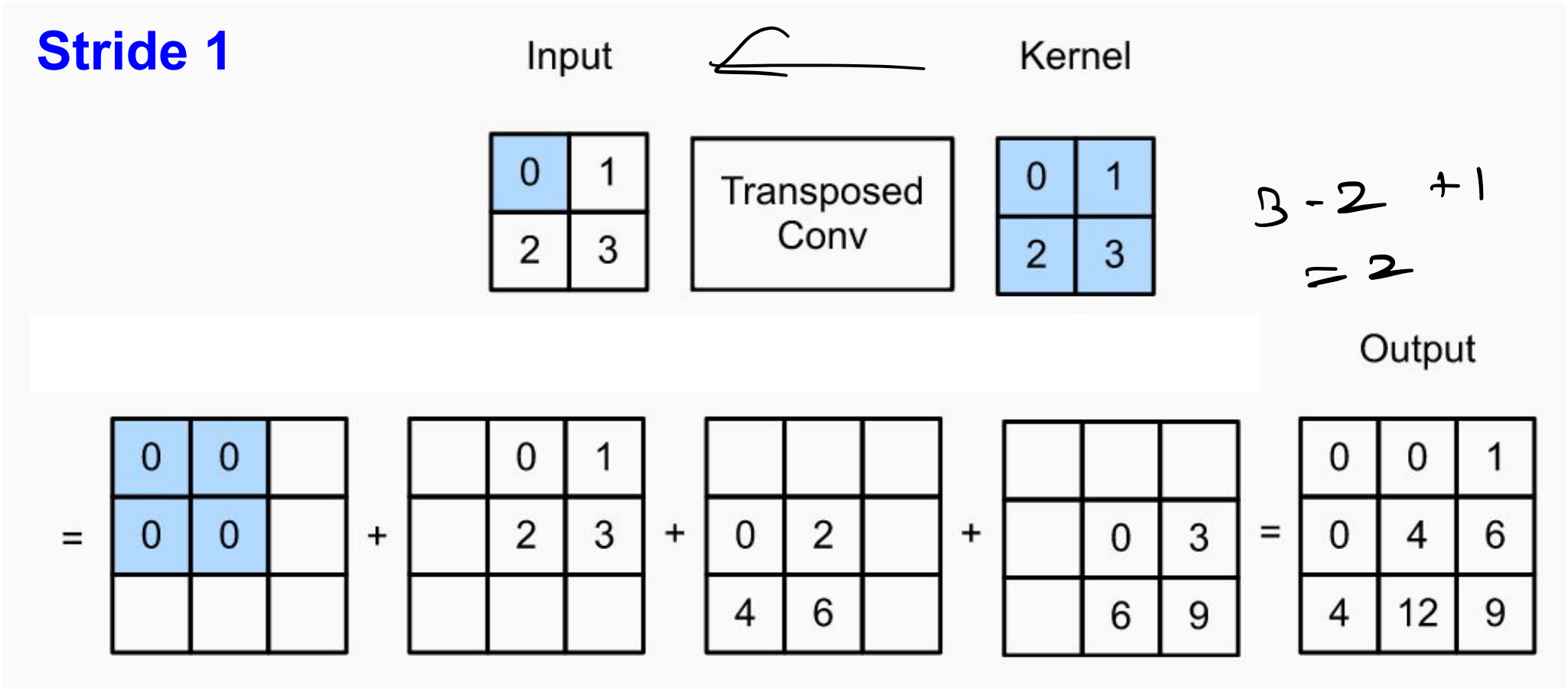
Transpose Convolution: 2D Example

Stride 1



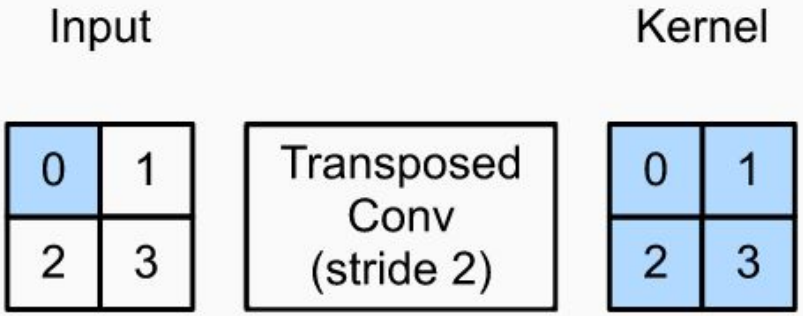


Transpose Convolution: 2D Example



Transposed convolution with a 2x2 kernel. The shaded portions are a portion of an intermediate tensor as well as the input and kernel tensor elements used for the computation.

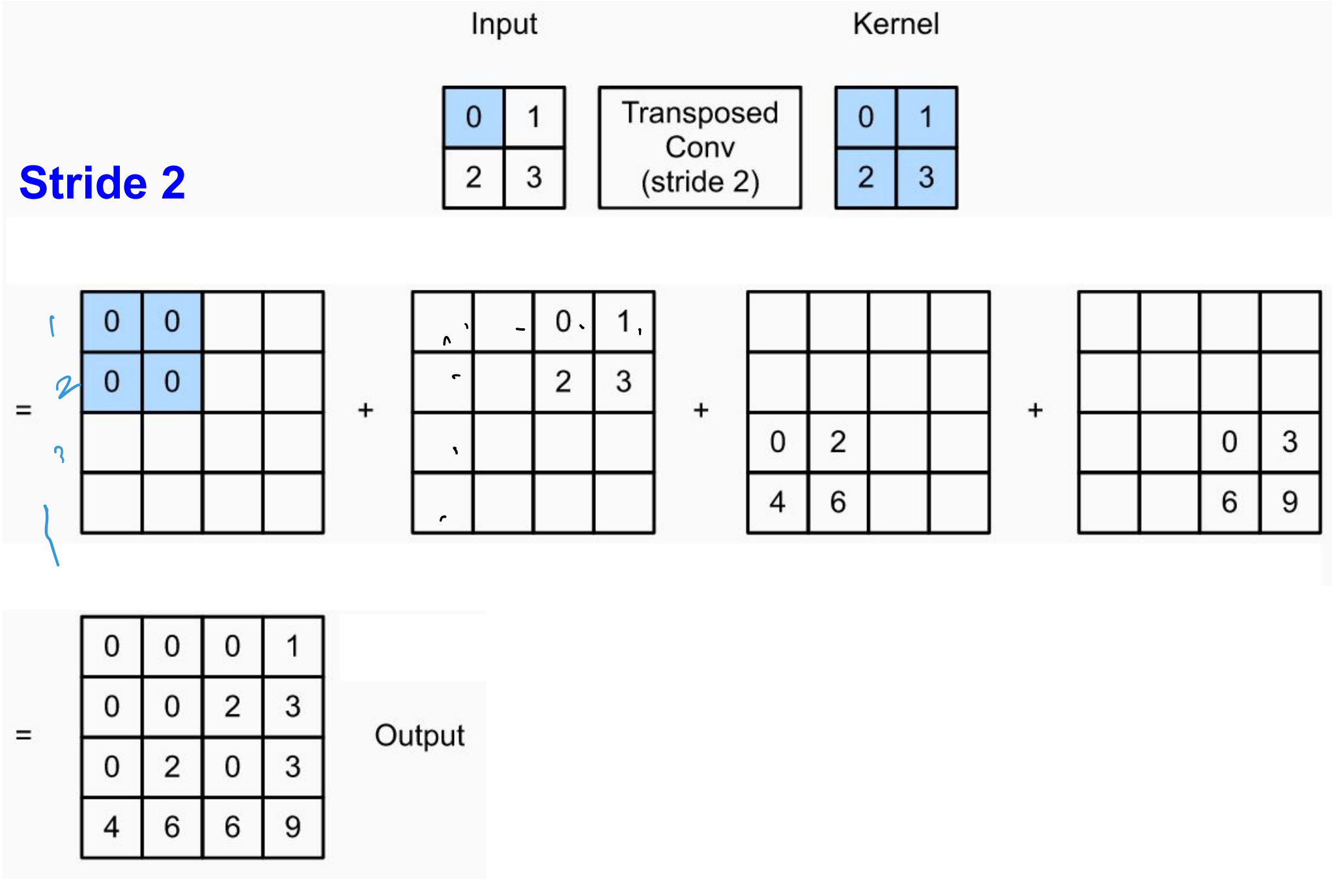
Stride 2



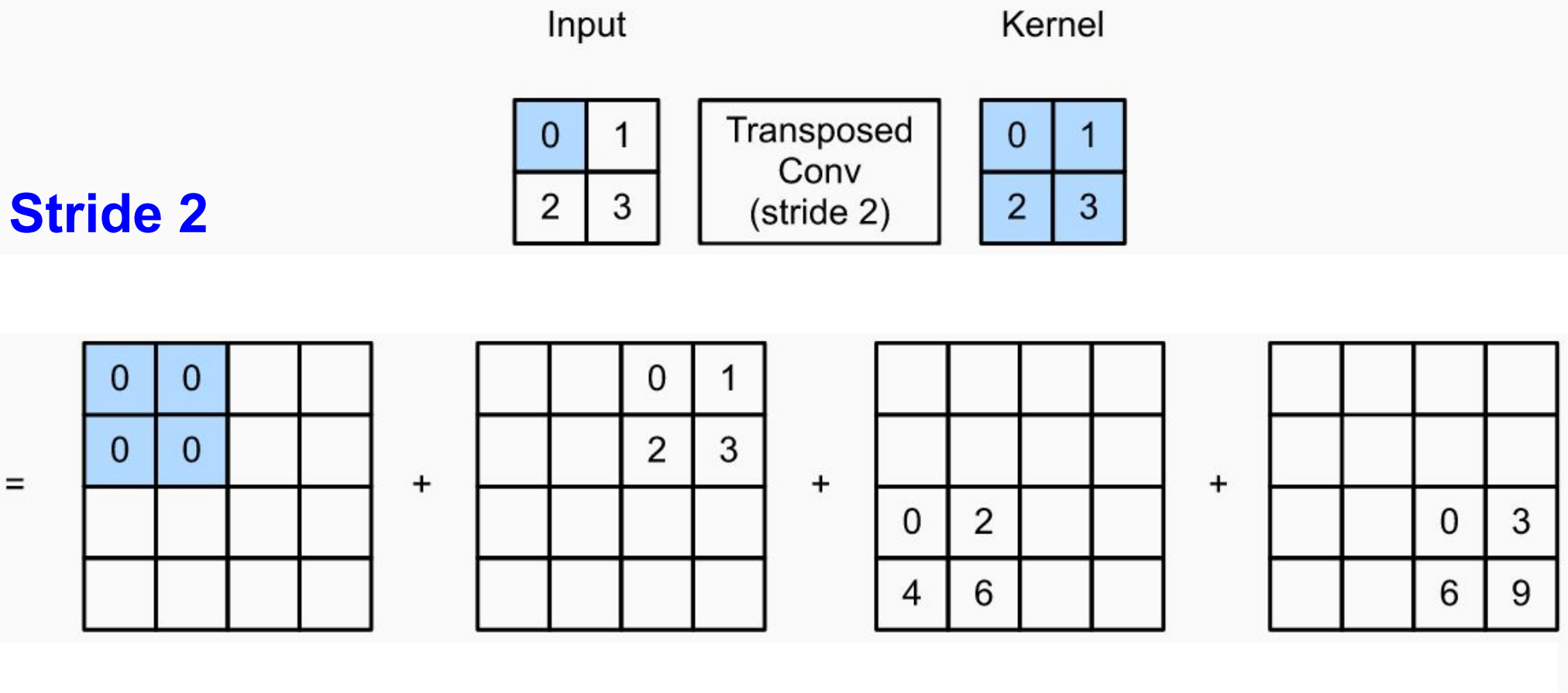
$$\frac{5-3}{2}+1=3$$

$$\frac{4-2}{2}+1=2$$

Stride 2


Input
Kernel
0 1
2 3
Transposed Conv (stride 2)
0 1
2 3
=
0 0
0 0
+
0 1
2 3
+
0 2
4 6
+
0 3
6 9
=
0 0 0 1
0 0 2 3
0 2 0 3
4 6 6 9
Output

Stride 2




```
# Add a transposed convolutional layer with 1 filters, kernel size 2x2, stride 2, no padding
Conv2DTranspose(2, (2, 2), strides=(2, 2), padding='valid', input_shape=(height, width, channels))
```

Transposed Convolution

Applications of Transposed Convolution:

- Generating high-resolution images from low-resolution input
- Encoder-Decoder networks
  - Transposed Convolution used for image denoising, inpainting, and super-resolution
  - Transposed Convolution used for semantic segmentation

Transposed Convolution

Disadvantages of Transposed Convolution:

- Can generate low-quality or unrealistic images if the training data is insufficient or the model architecture is not well-suited for the task
- Can lead to checkerboard artifacts in the output images, where the same pixel values are repeated in a grid-like pattern

Transposed Convolution

Alternatives to Transposed Convolution:

- Dense upsampling layers, computationally expensive and require more memory than convolutional layers.
- Upsampling layers, such as bilinear or nearest-neighbor interpolation (without learning any new parameters).

Receptive Field

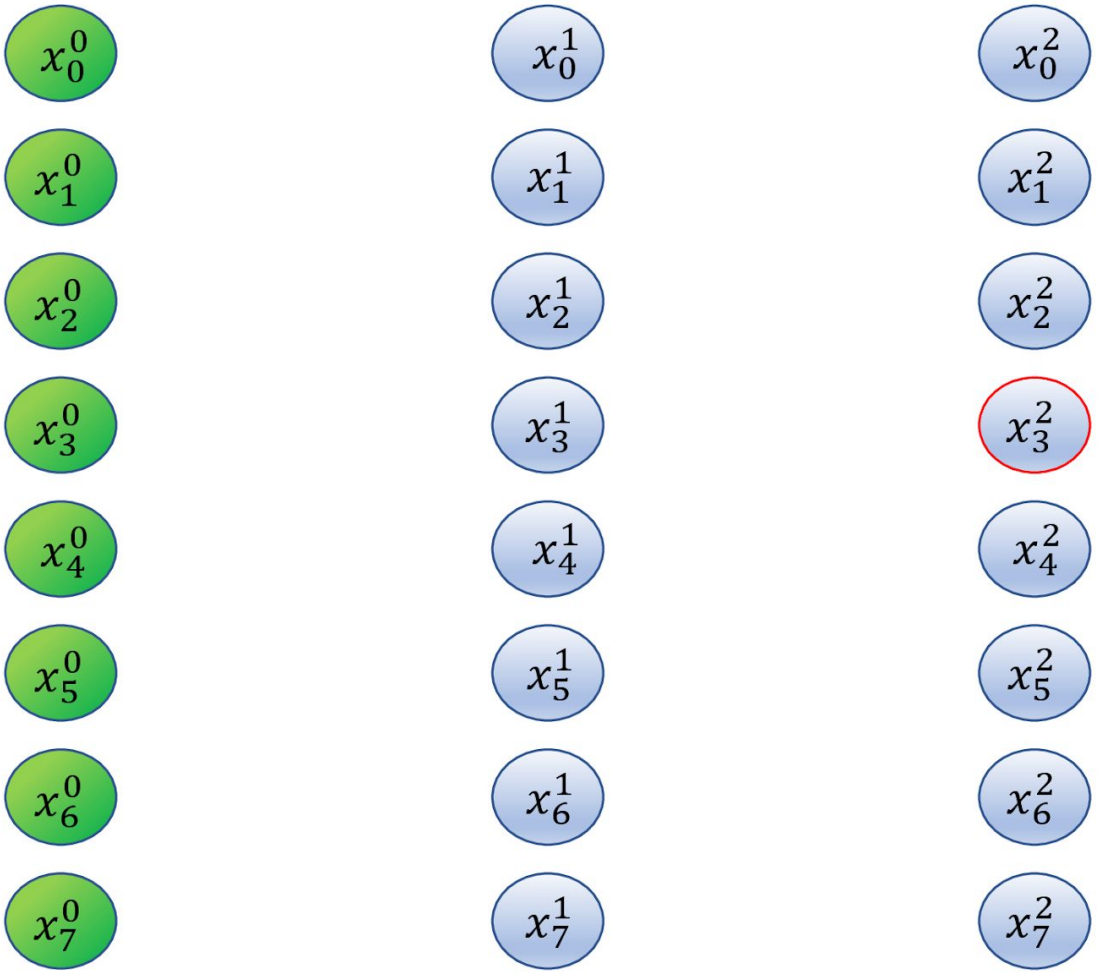
Receptive Field

- Receptive field refers to the area of the input that is used by a particular neuron or feature map in the network.

Area of input used by particular neuron or feature maps.

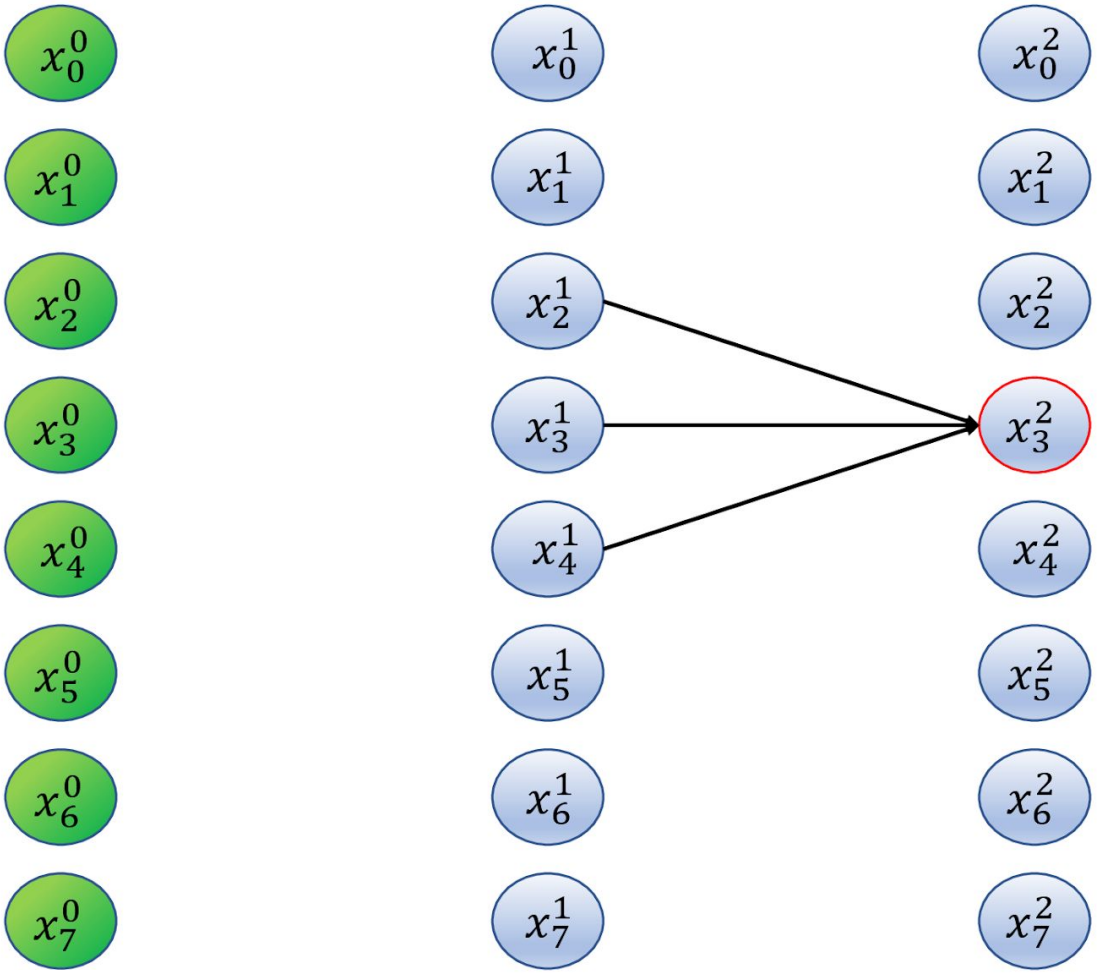
Receptive Field (example)

$[Filter\ size: 3]$



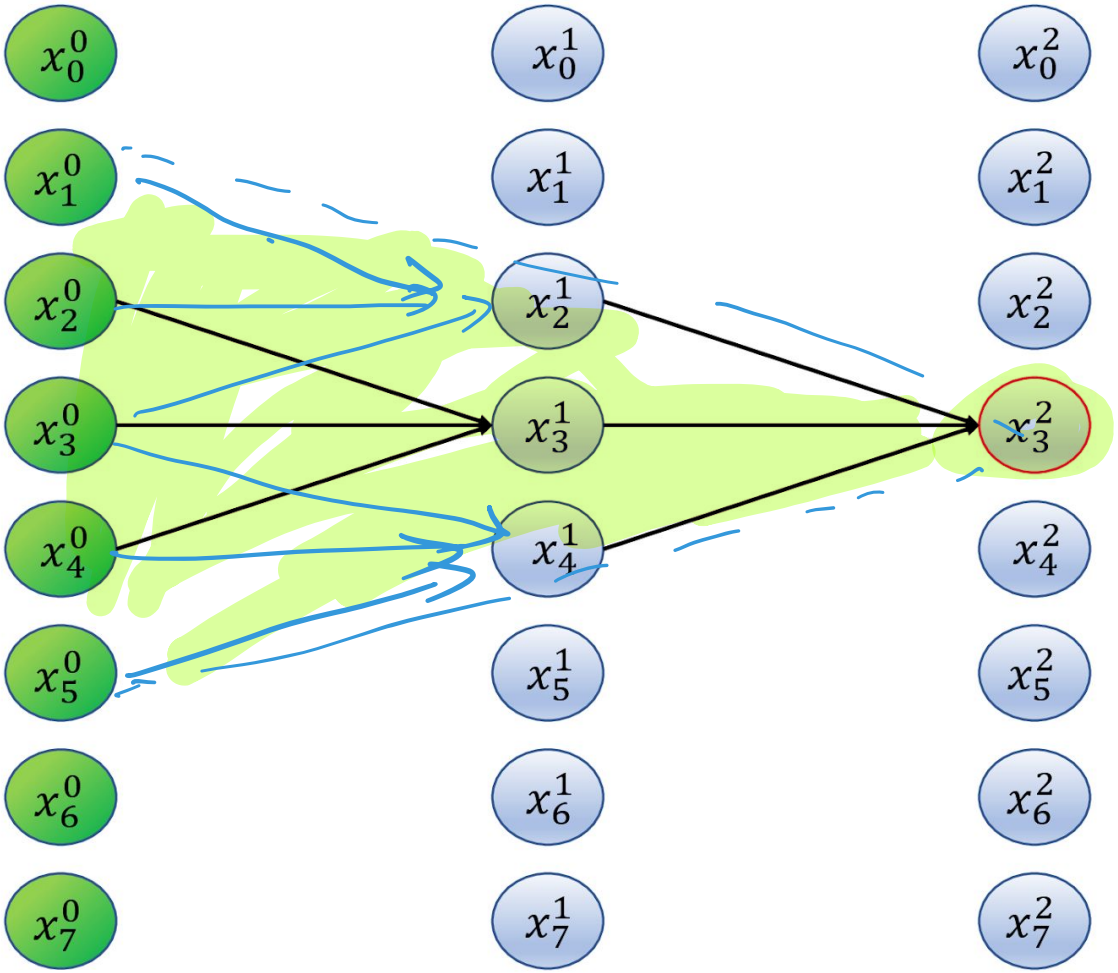
Receptive Field (example)

$[Filter\ size: 3]$

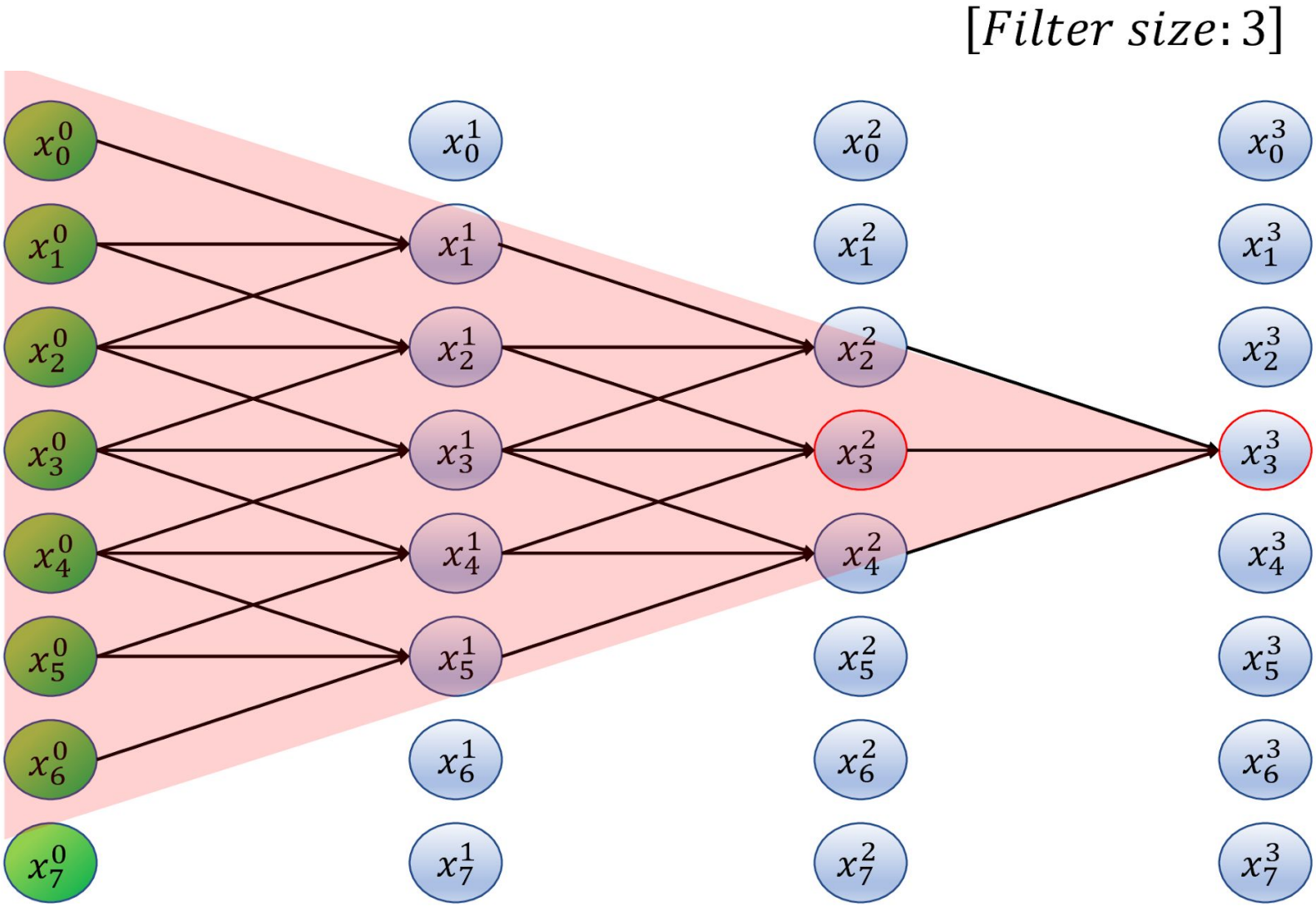


Receptive Field (example)

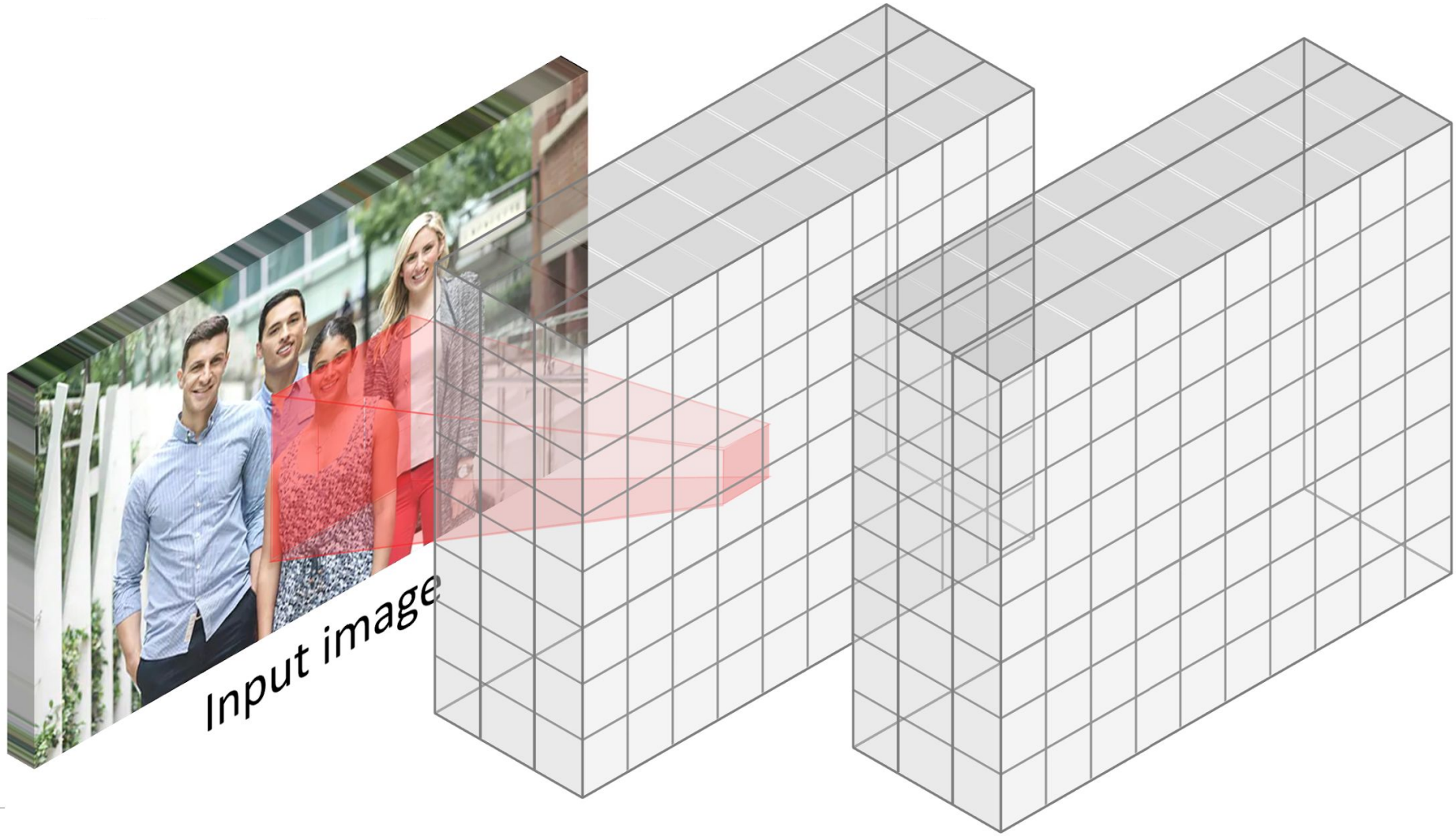
$[Filter\ size: 3]$



Receptive Field (example)

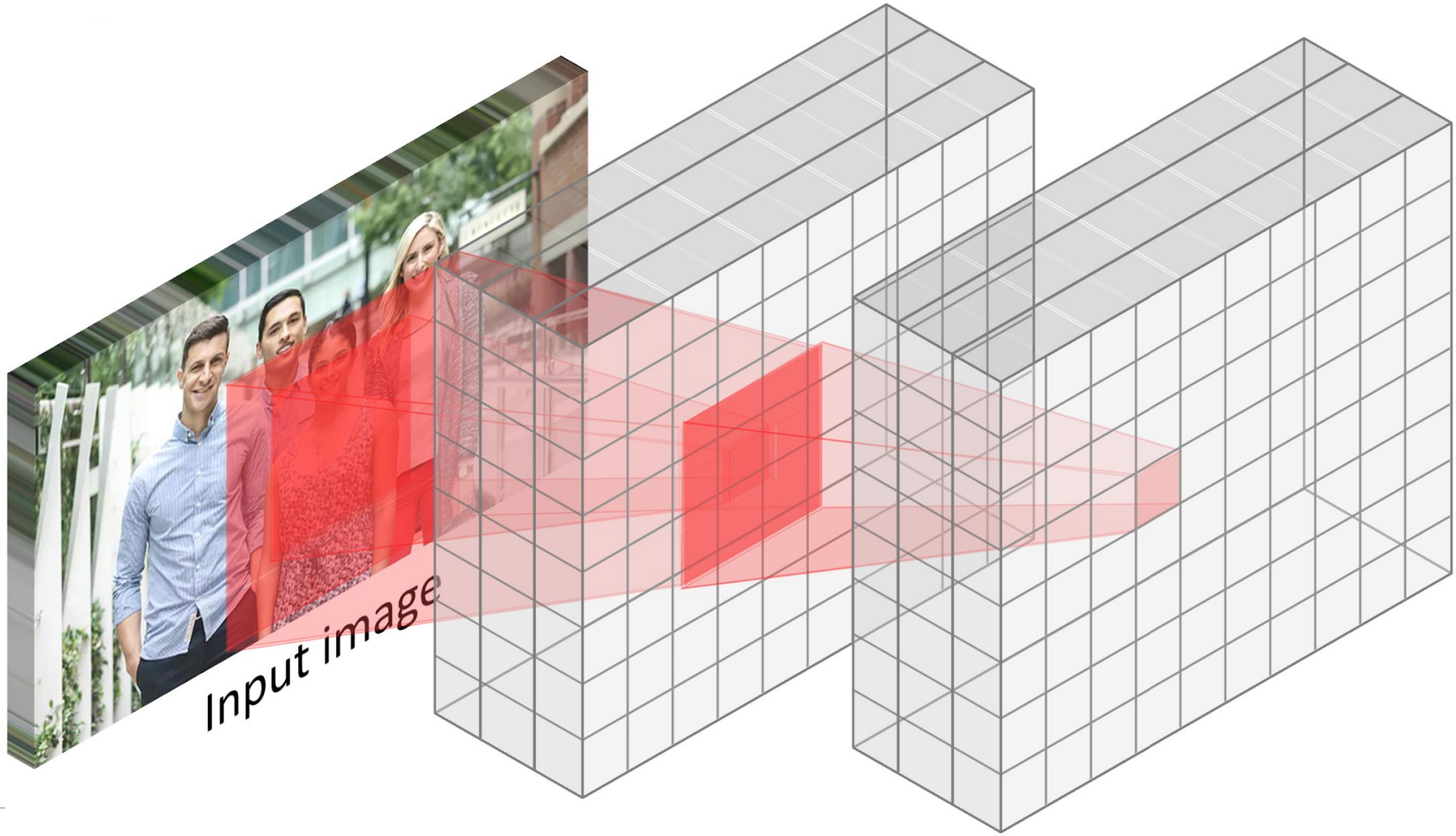


Receptive Field



Receptive field refers to the area of the input image that is used by a particular neuron or feature map in the network.

Receptive Field



Receptive field refers to the area of the input image that is used by a particular neuron or feature map in the network.

Receptive fields (Advantages)

- Capturing global features
- Improved recognition accuracy
- Improved generalization

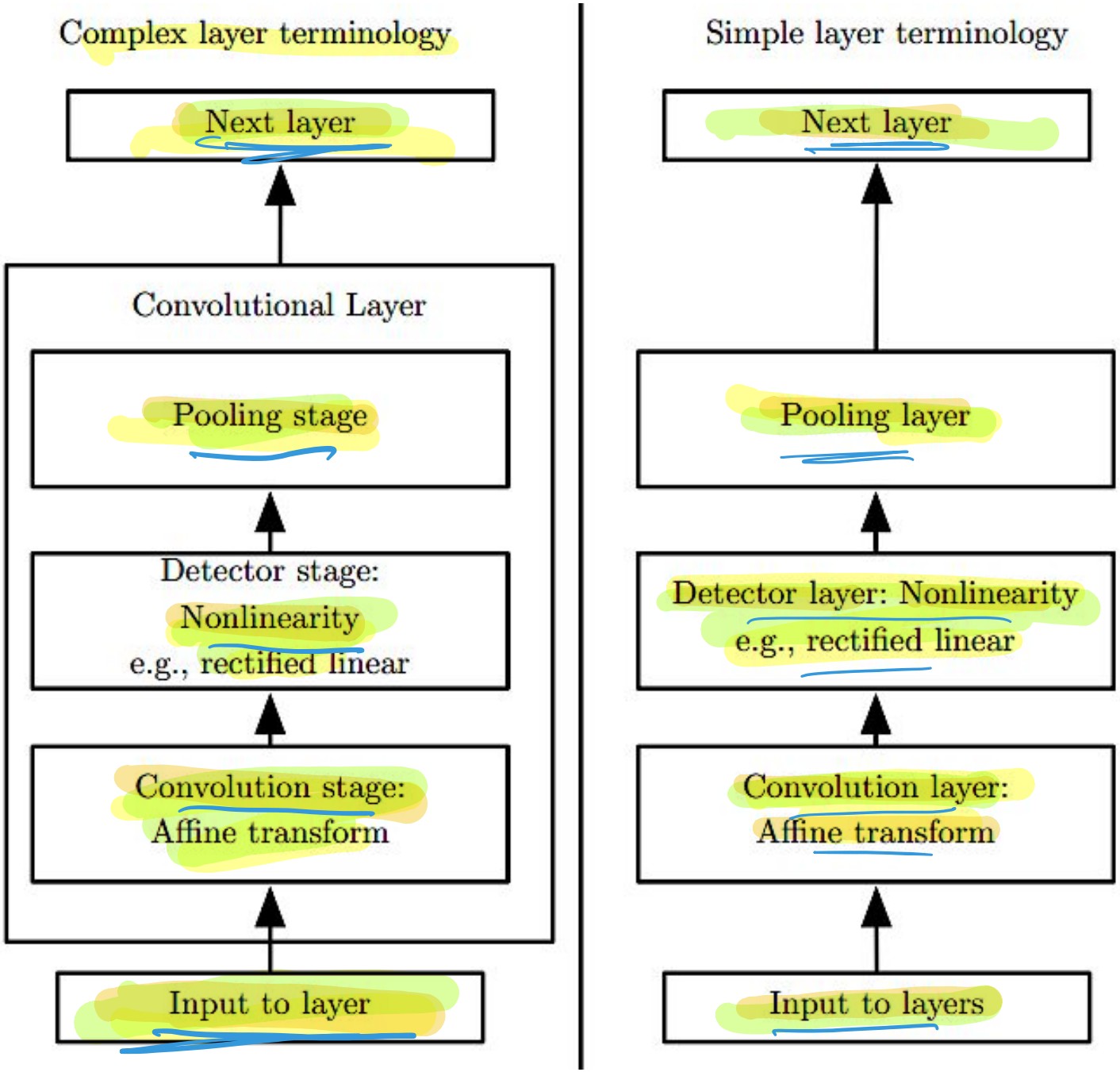
Receptive fields (Disadvantages)

- Increased computational cost
- Reduced localization accuracy
- Limited context modeling

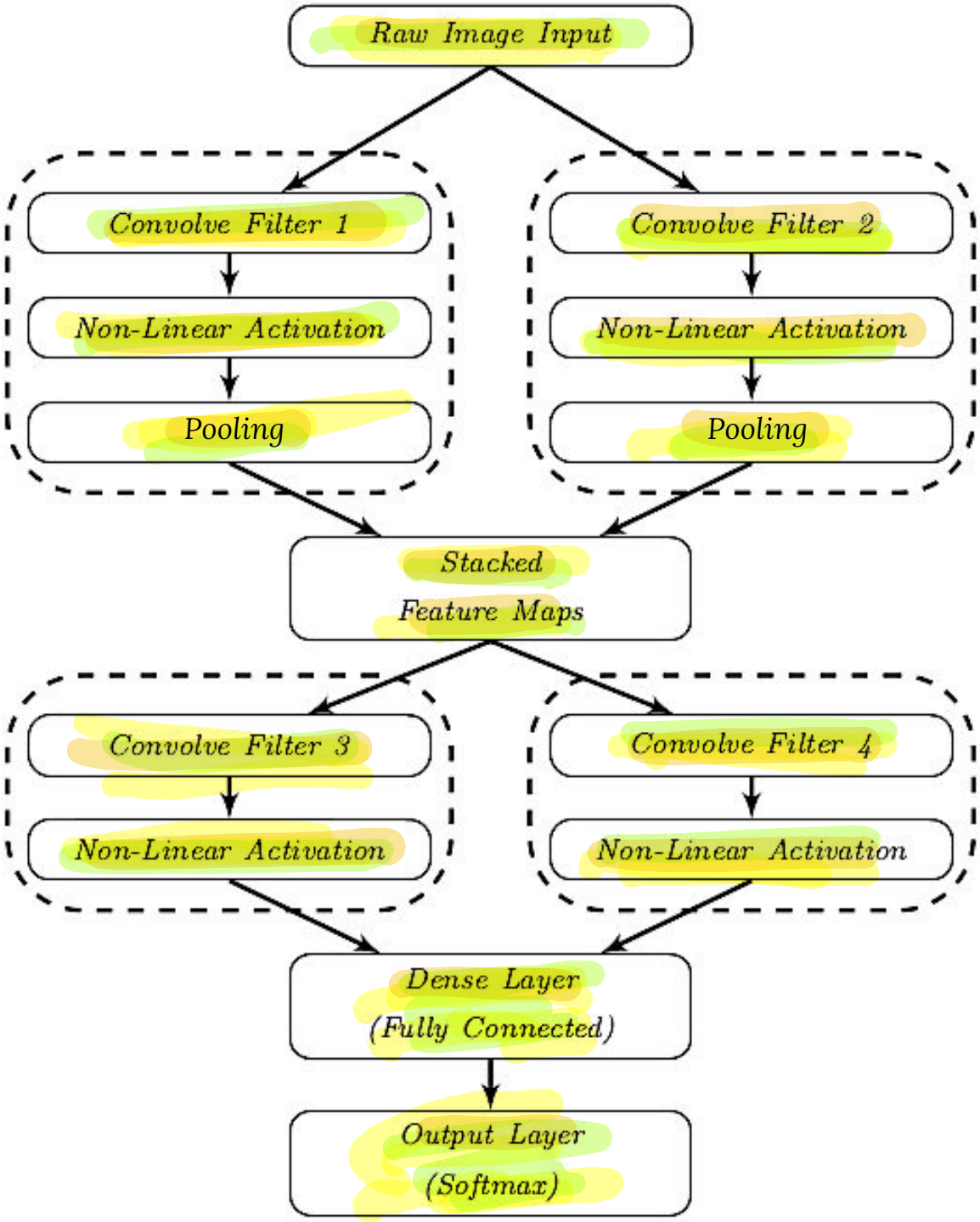
→ missing finer level of details

Convolutional Neural Networks (CNNs)

Components of CNNs



Convolution Neural Networks



Schematic of typical sequences of layers in CNNs

Feature extraction using CNNs

- CNN take advantage of the **spatial structure** for feature extraction

Feature extraction using CNNs

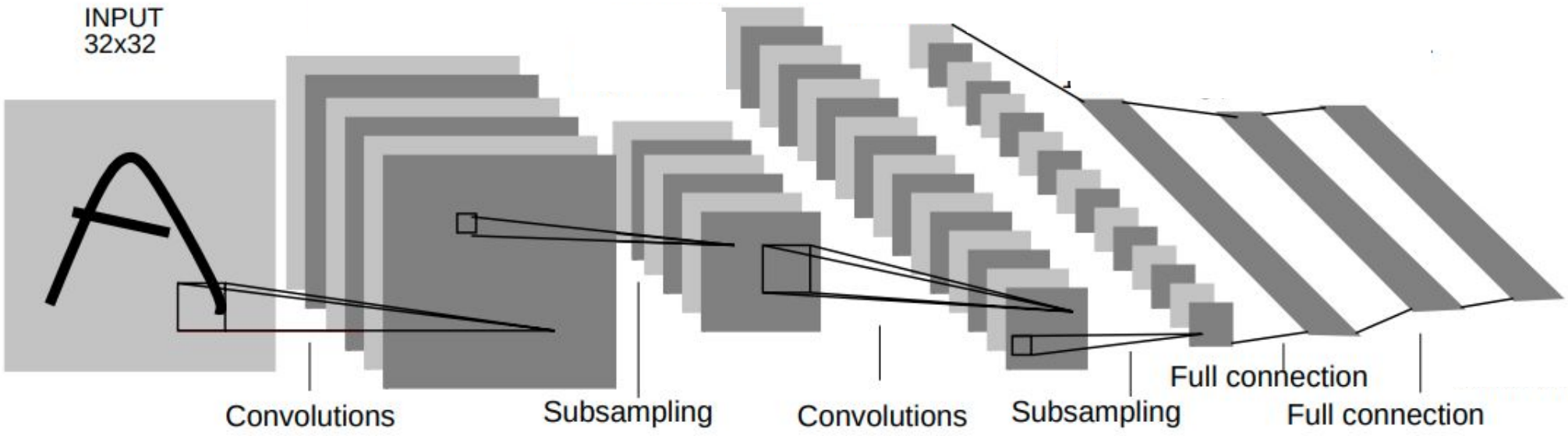
- CNN take advantage of the **spatial structure** for feature extraction
- Learn **hierarchical representations**: starting with low-level features (edges) and progressing to higher-level features (shape).

Feature extraction using CNNs

- CNN take advantage of the **spatial structure** for feature extraction
- Learn **hierarchical representations**: starting with low-level features (edges) and progressing to higher-level features (shape).
- Learn **complex representations:** convolutional layers can be easily stacked to create deeper networks.

LeNet

LeNet


INPUT
32x32
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection

LeNet

- LeNet is a convolutional neural network architecture designed by Yann LeCun in 1998.
- LeNet uses a gradient-based learning algorithm (backpropagation) for training.
- LeNet was designed specifically for handwritten digit recognition, and was trained on the MNIST dataset of handwritten digits.
- LeNet is also adapted and extended for other image recognition tasks.

LeNet

Digit recognition: 10 classes

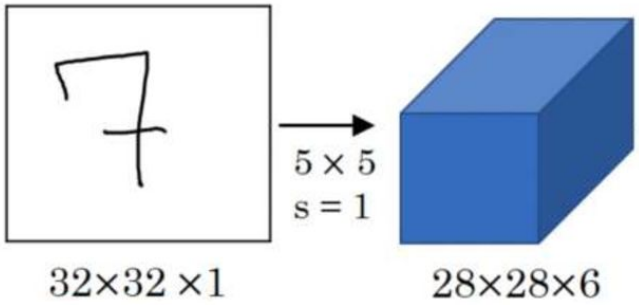
32×32 ×1

Input: 32×32 grayscale images

This one: Labeled as class "7"

LeNet

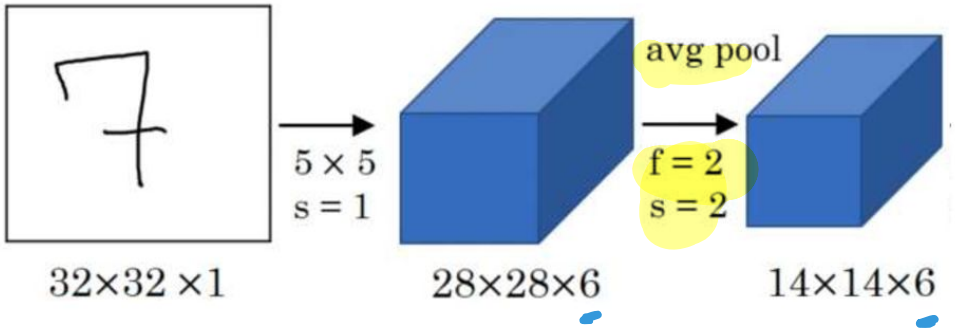
Digit recognition: 10 classes



» Valid convolution: size shrinks

LeNet

Digit recognition: 10 classes



At that time average pooling was used, now max pooling is much more common

LeNet

- **Convolutional layer 1**: This layer applies a set of filters to the input image to extract features such as edges and corners.

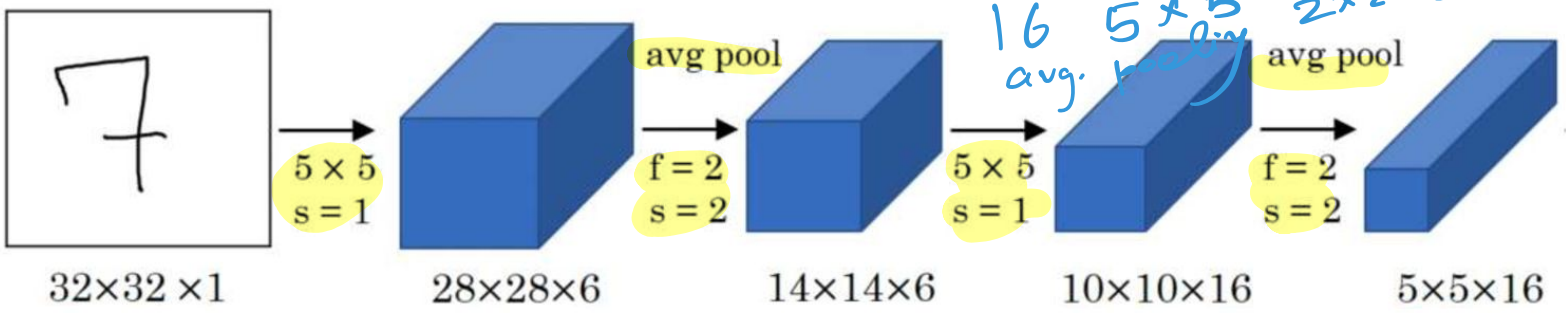
LeNet

- **Convolutional layer 1**: This layer applies a set of filters to the input image to extract features such as edges and corners.

- **Subsampling layer 1:**
  - Reduces the spatial size of the feature maps, while retaining the most important features.
  - Reduce the computational complexity.

LeNet

Digit recognition: 10 classes



- The second layer is another convolutional layer with 16 feature maps, followed by another pooling layer.

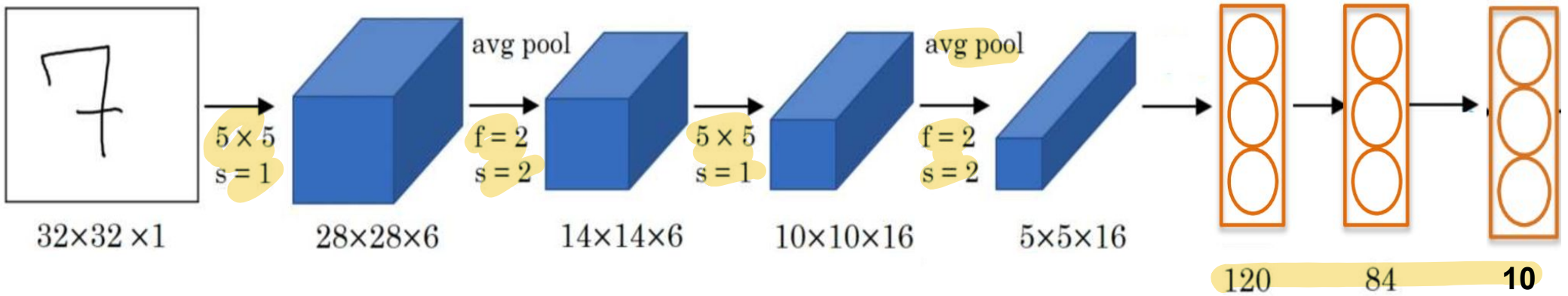
LeNet

- **Convolutional layer 2**: Allowing the network to learn more complex features.

- **Subsampling layer 2:** Reduces the size of the feature maps, while retaining the most important features.

LeNet

Digit recognition: 10 classes

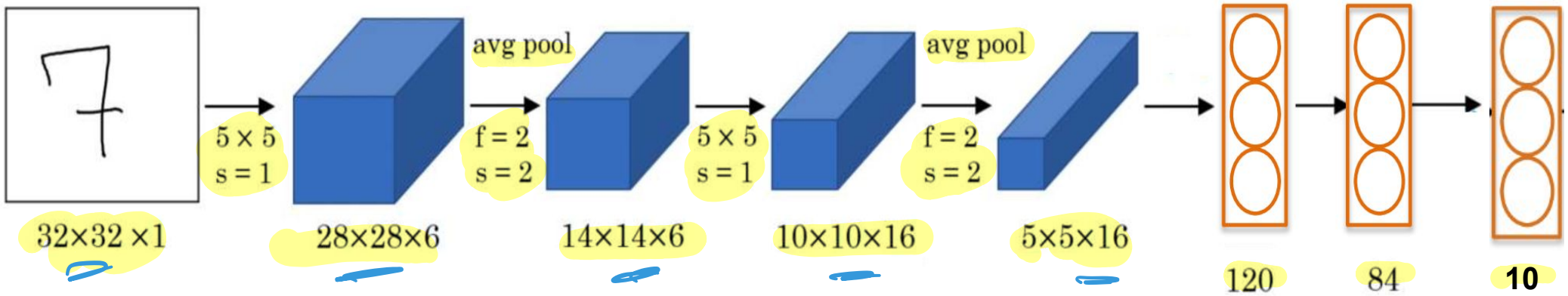


LeNet

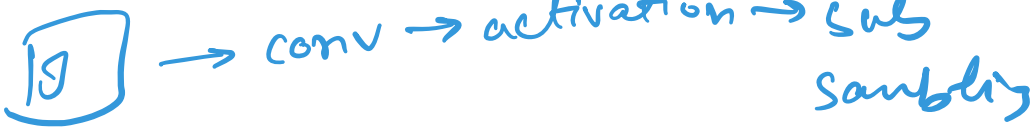
- **Fully connected layer 1 and 2:**
  - The third layer is a fully connected layer with 120 units, followed by a fourth layer with 84 units, both using a '*relu*' (*updated*).
  - Last layers learns non-linear combinations of the features extracted by the convolutional layers.

LeNet

Digit recognition: 10 classes



- **Output layer:** Fully connected layers with 10 units each, using a softmax activation function to produce the final output probabilities.



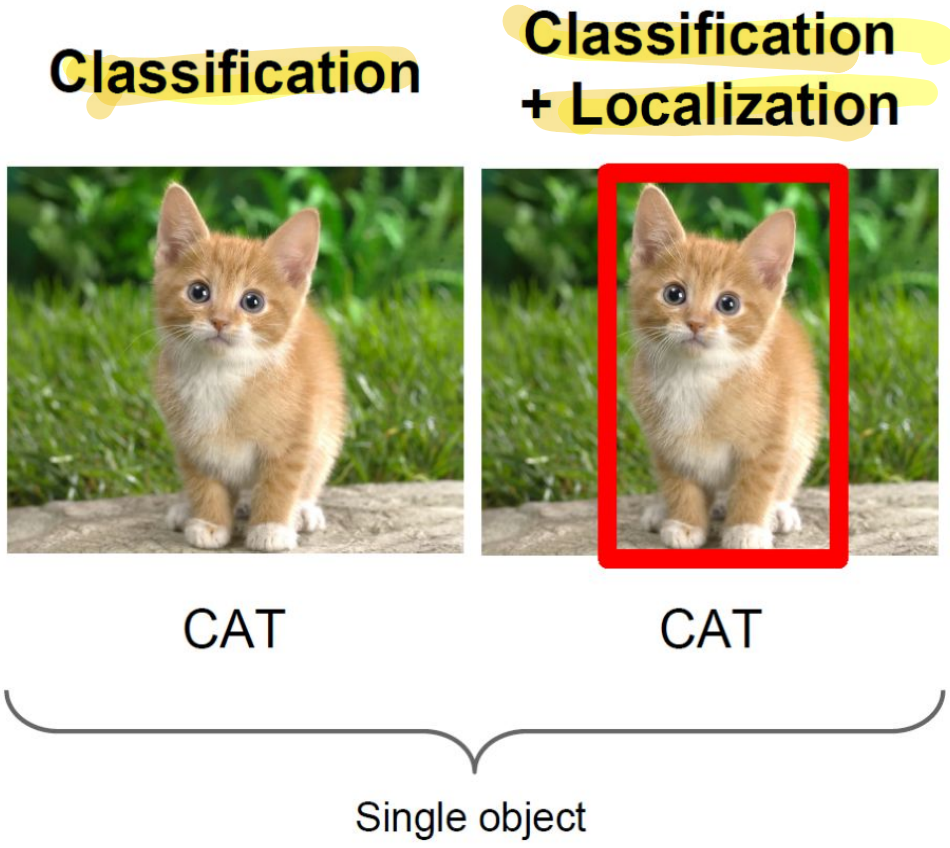
```python
from keras.models import Sequential
from keras.layers import Conv2D, AveragePooling2D, Flatten, Dense

# Input layer
inputs = Input(shape=(32, 32, 1))

# Convolutional layer 1
conv1 = Conv2D(filters=6, kernel_size=(5, 5), strides=(1, 1), activation='relu',
padding='same')(inputs)

# Average pooling layer 1
pool1 = AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid')(conv1)

# Convolutional layer 2
conv2 = Conv2D(filters=16, kernel_size=(5, 5), strides=(1, 1), activation='relu',
padding='valid')(pool1)

# Average pooling layer 2
pool2 = AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid')(conv2)

# Flatten layer
flatten = Flatten()(pool2)

# Fully connected layer 1
fc1 = Dense(units=120, activation='relu')(flatten)
# Fully connected layer 2
fc2 = Dense(units=84, activation='relu')(fc1)

# Output layer
outputs = Dense(units=10, activation='softmax')(fc2)
```

32 x 32 x 1

↓ 5x5x6 p5=1

LeNet

Advantages of Convolutional Networks



Advantages of Convolutional Networks

LeNet → MNIST

5×5 ↓ 6 filter

Avg pool f = 2 s = 2

5×5 ↓ 16 filter

Avg pool f = 2 s = 2

Classification

Classification + Localization

Object Detection

CAT

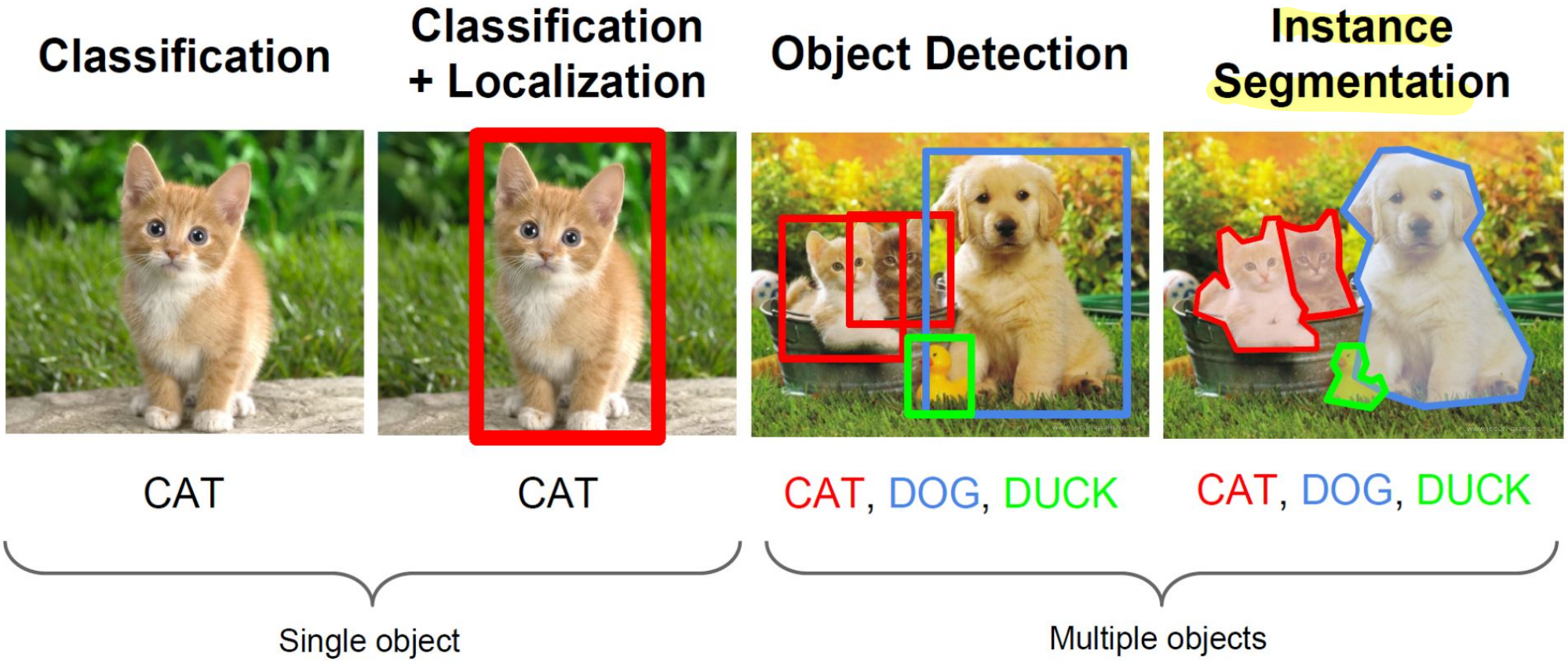
CAT

CAT, DOG, DUCK

Single object

Multiple objects

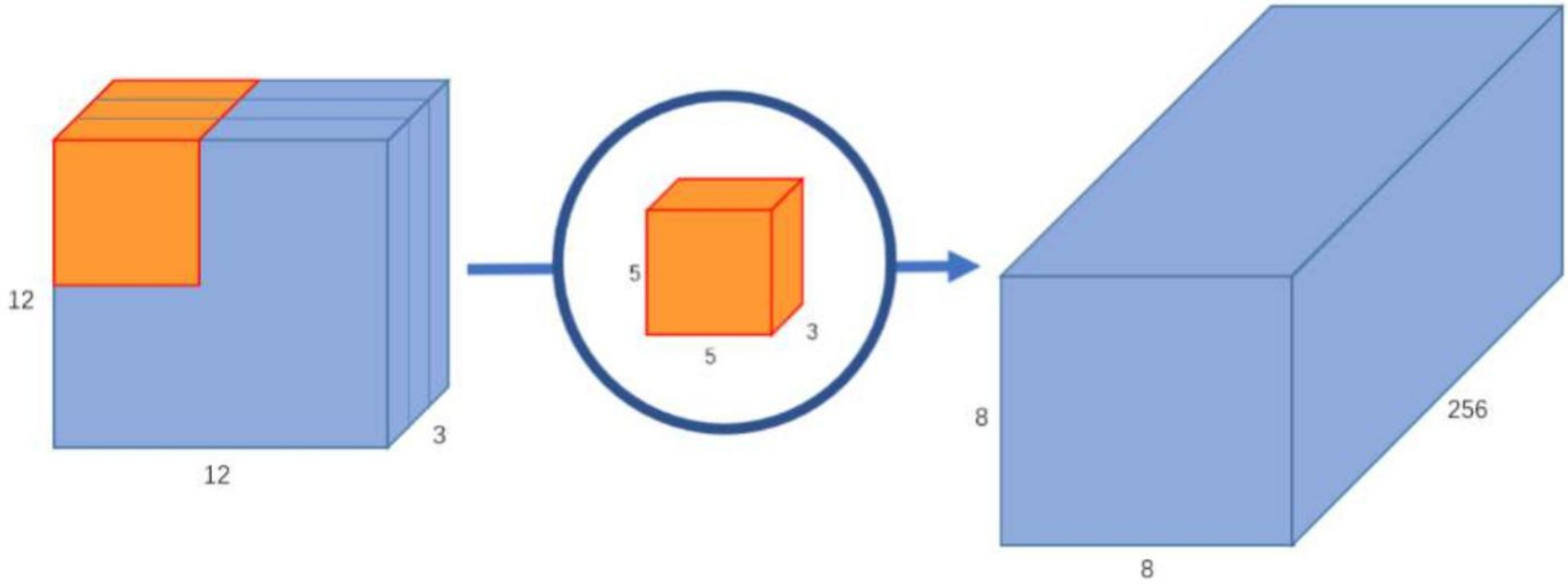
Advantages of Convolutional Networks



Special Convolution (Depth-wise separable convolutions)

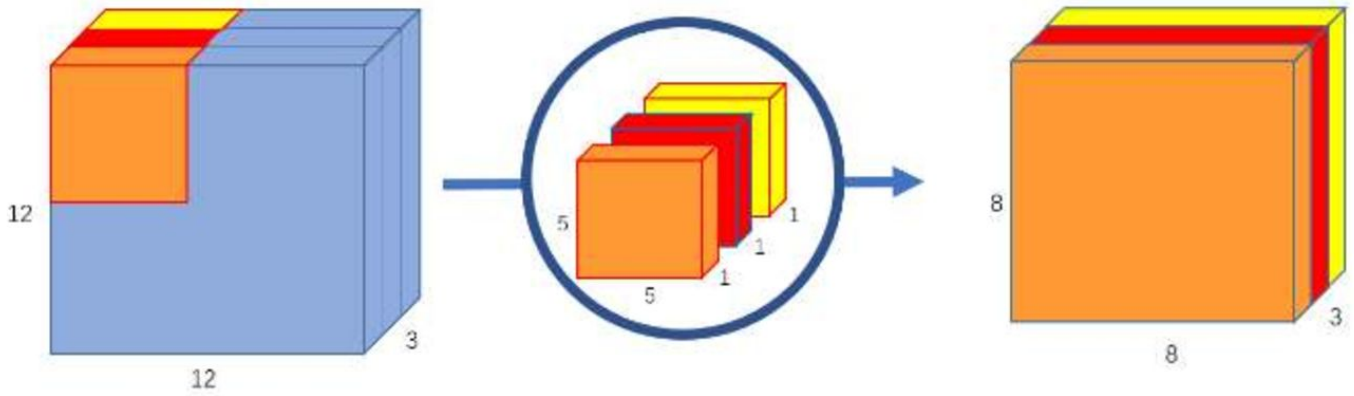
Normal convolutions

Normal convolution acts on all channels



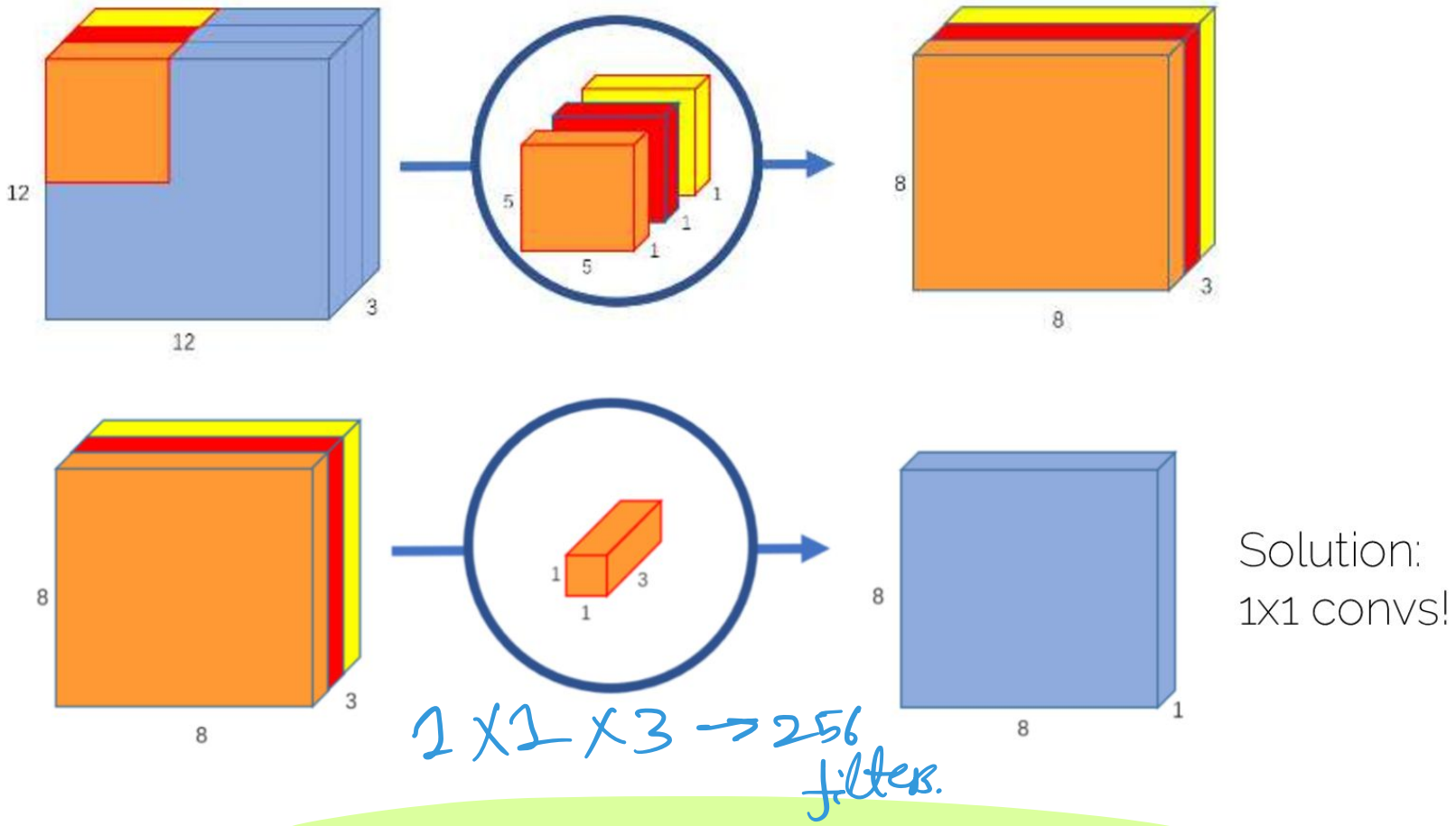

```
# Input layer for an image of size 12x12 with 3 channels
inputs = Input(shape=(12, 12, 3))

# Convolutional layer with 256 filters of size 5x5
conv = Conv2D(filters=256, kernel_size=(5, 5), padding='valid')(inputs)
```

Depth-wise separable convolutions



1. Apply a separate filter to each input channel to produce a set of intermediate feature maps.

Depth-wise separable convolutions



2. Use a 1x1 convolution to combine the intermediate feature maps into the final output.