# Lecture 3

# Convolutional Neural Networks

*Minh-Tan Pham, Matthieu Le Lain*

**BUT2 INFO, 2023-2024**

**IUT de Vannes, Université Bretagne Sud**

*minh-tan.pham@univ-ubs.fr*

# Planning

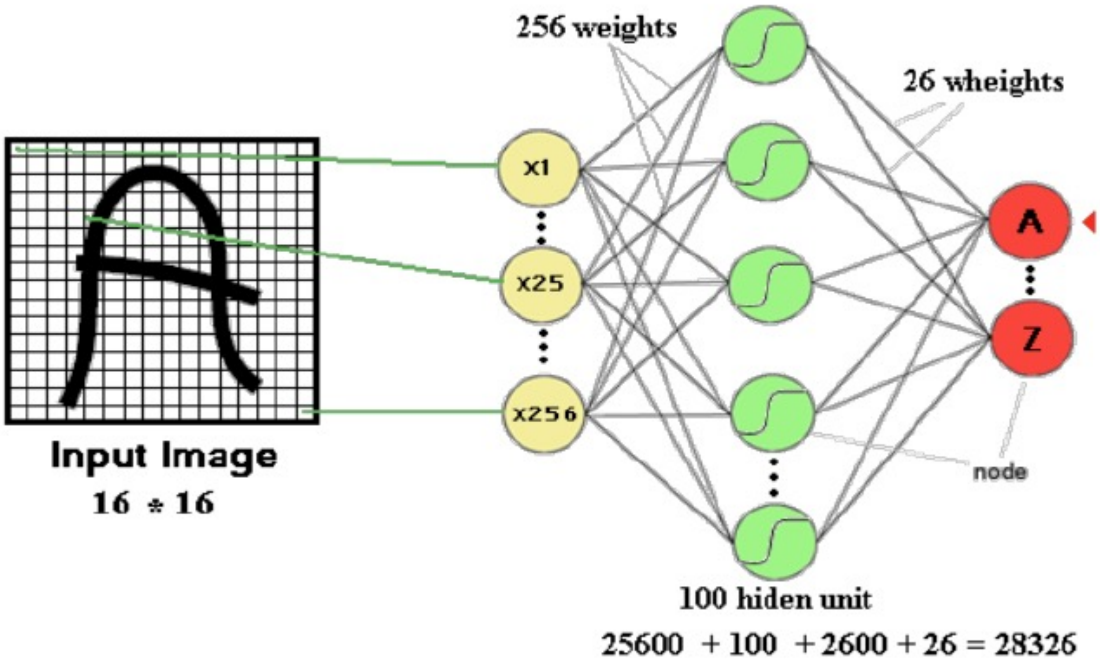
- Why CNNs ?
- Image Convolution
- Pooling Layer
- Typical CNNs
- Lab Assignments

# Why CNNs ?

# Why CNNs ?

## Neural nets - Limitations

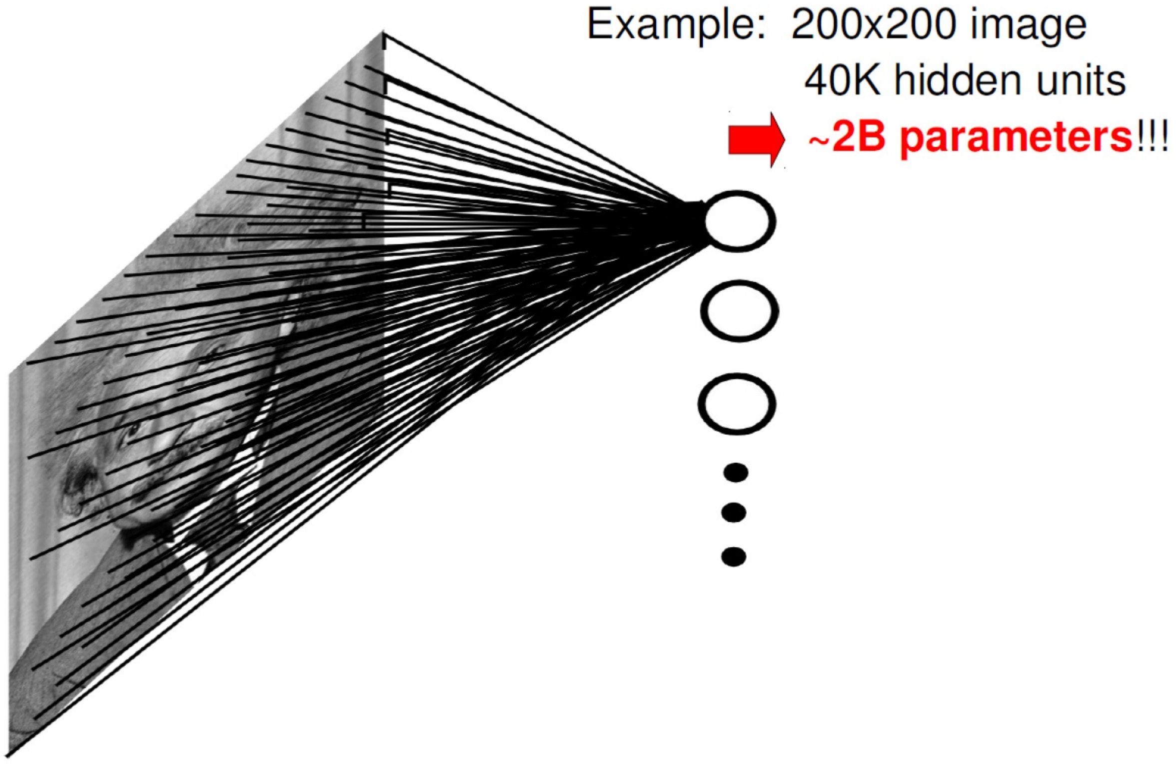
- Neural networks = fully-connected networks
- Ignore the input data structure (spatial relationship and topology)
- Require too many parameters → more time, disk space, more training data



# Why CNNs ?

## Neural nets - Limitations

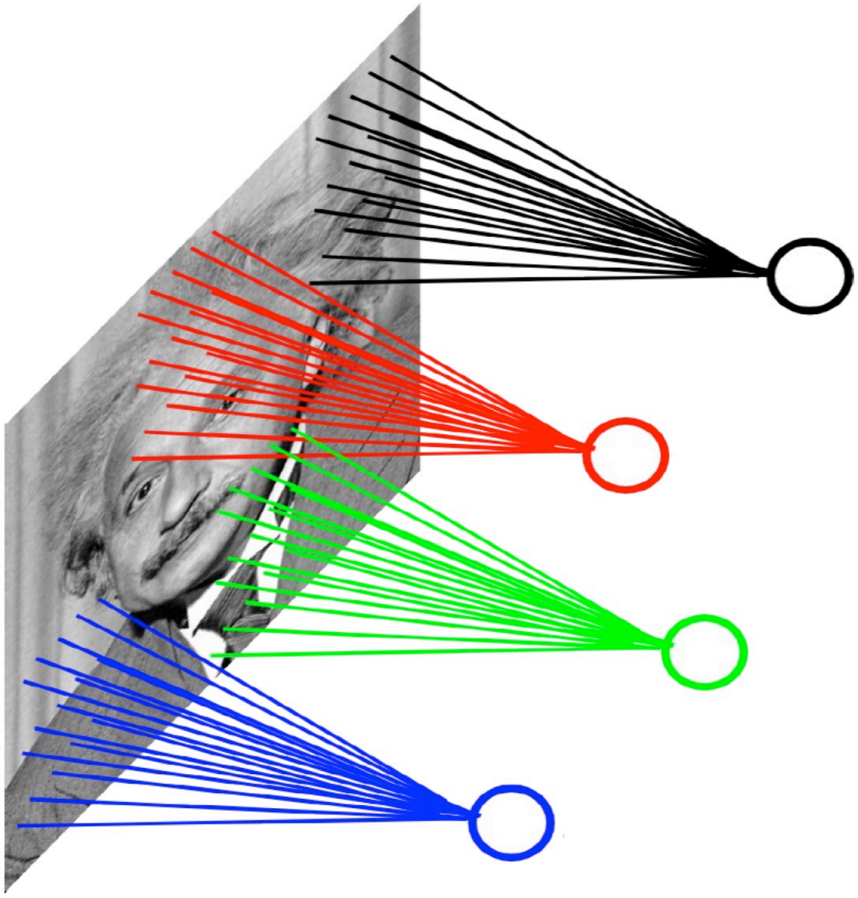
- Neural networks = fully-connected networks
- Ignore the input data structure (spatial relationship and topology)
- Require too many parameters → more time, disk space, more training data



# Why CNNs ?

## Solution: local connectivity

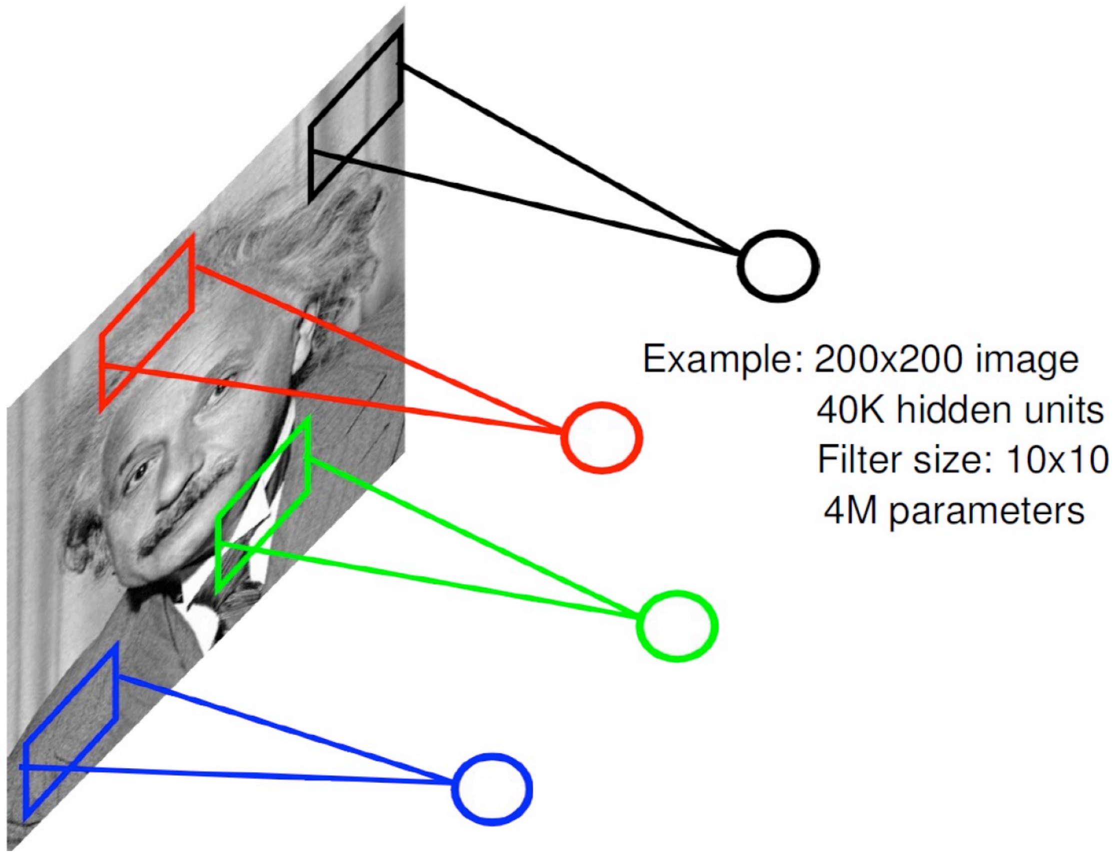
- Sparse connectivity ➔ weights only connected to local patch



# Why CNNs ?

## Solution: local connectivity

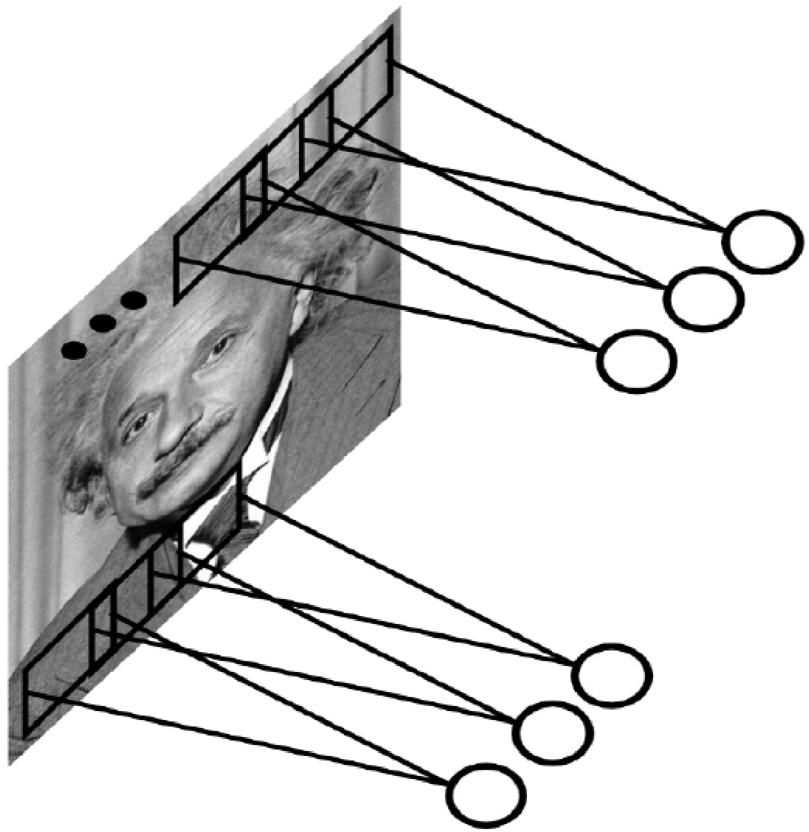
- Sparse connectivity ➔ weights only connected to local patch
- Preserve the spatial topology (keep spatial information in 2D)



# Why CNNs ?

## Solution: local connectivity

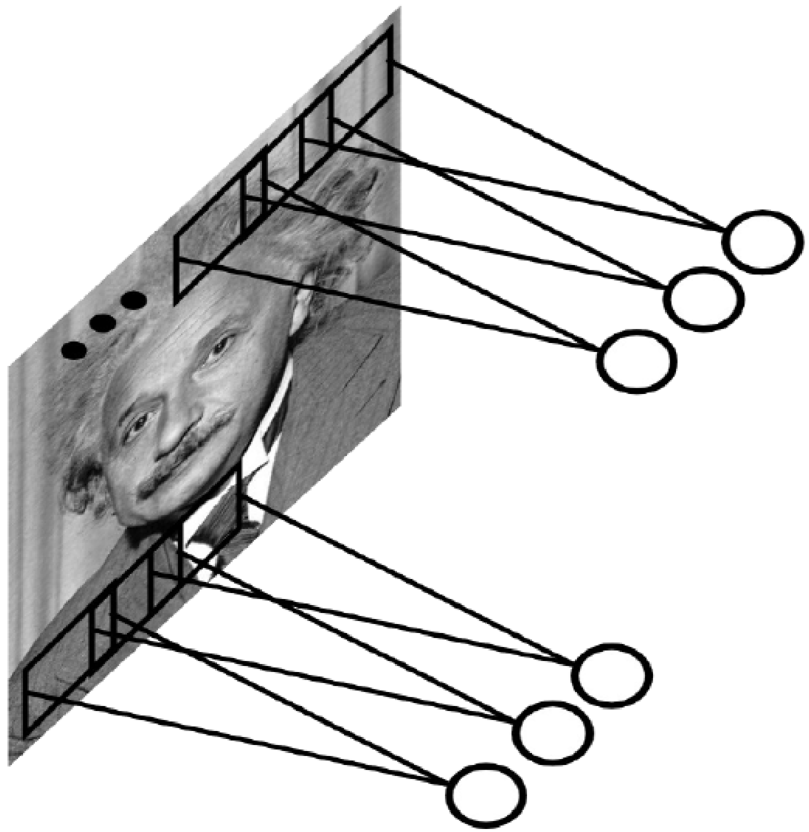
- Sparse connectivity ➔ weights only connected to local patch
- Preserve the spatial topology (keep spatial information in 2D)
- Share the same parameters across different locations ➔ reduce the number of parameters



# Why CNNs ?

## Convolutional layer

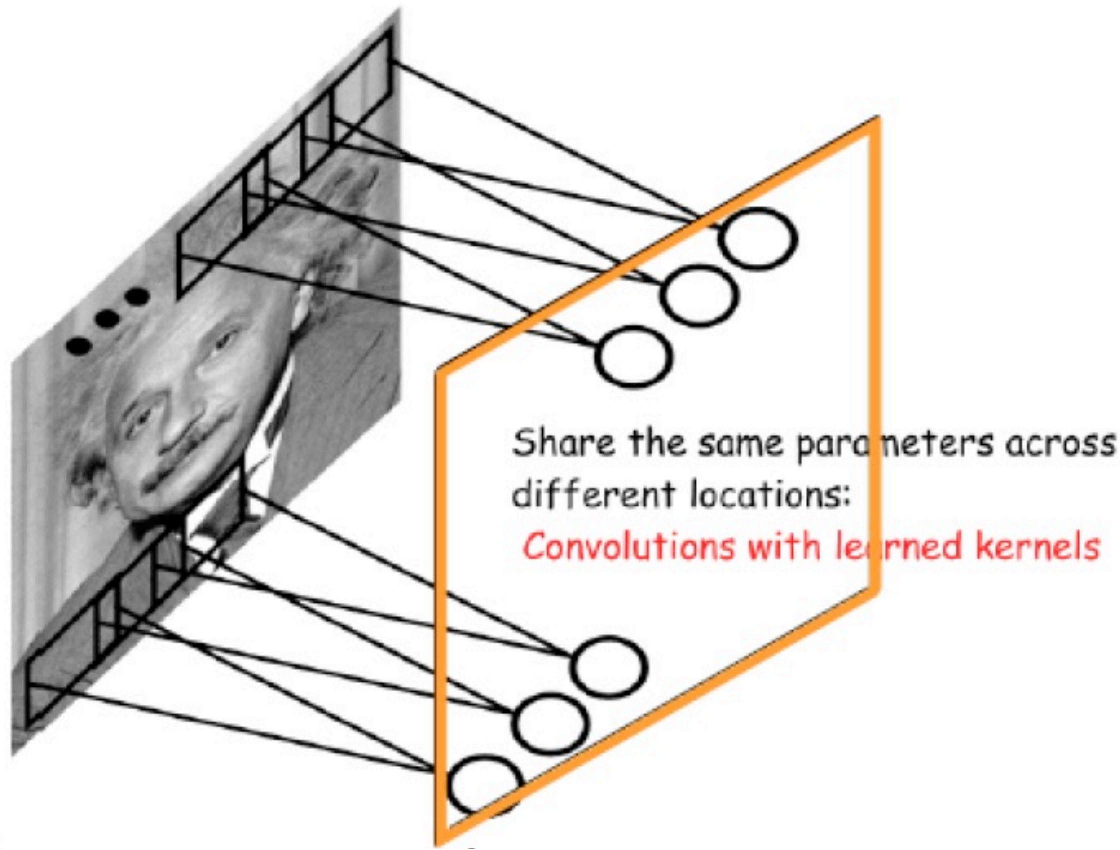
- Connect each hidden unit (neuron) to a small input patch (locally) ➔ filter or kernel



# Why CNNs ?

## Convolutional layer

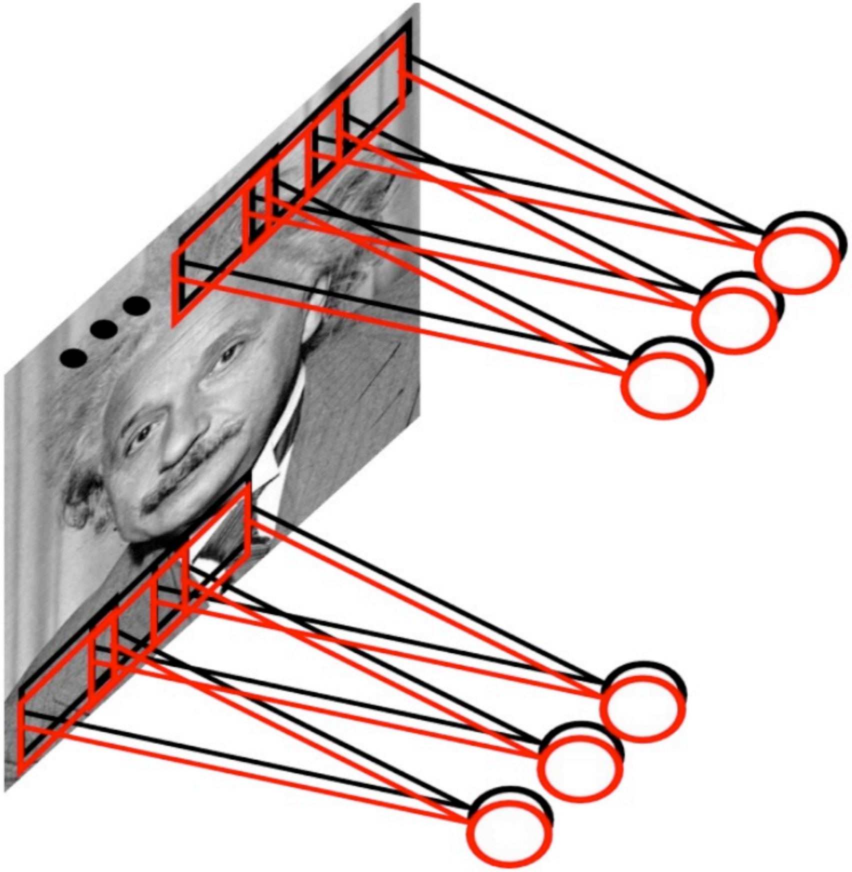
- Connect each hidden unit (neuron) to a small input patch (locally) ➔ filter or kernel
- Share the same weights across spatial space



# Why CNNs ?

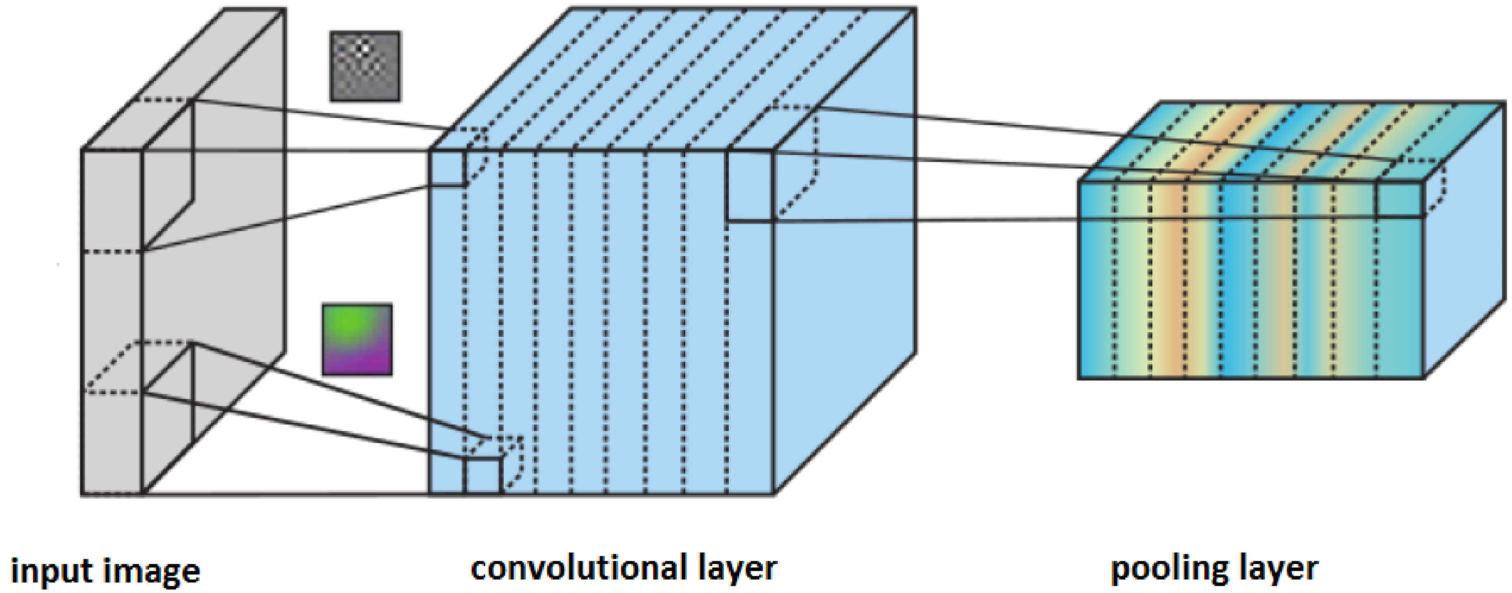
## Convolutional layer

- Connect each hidden unit (neuron) to a small input patch (locally) ➔ filter or kernel
- Share the same weights across spatial space
- Learn multiple fiters



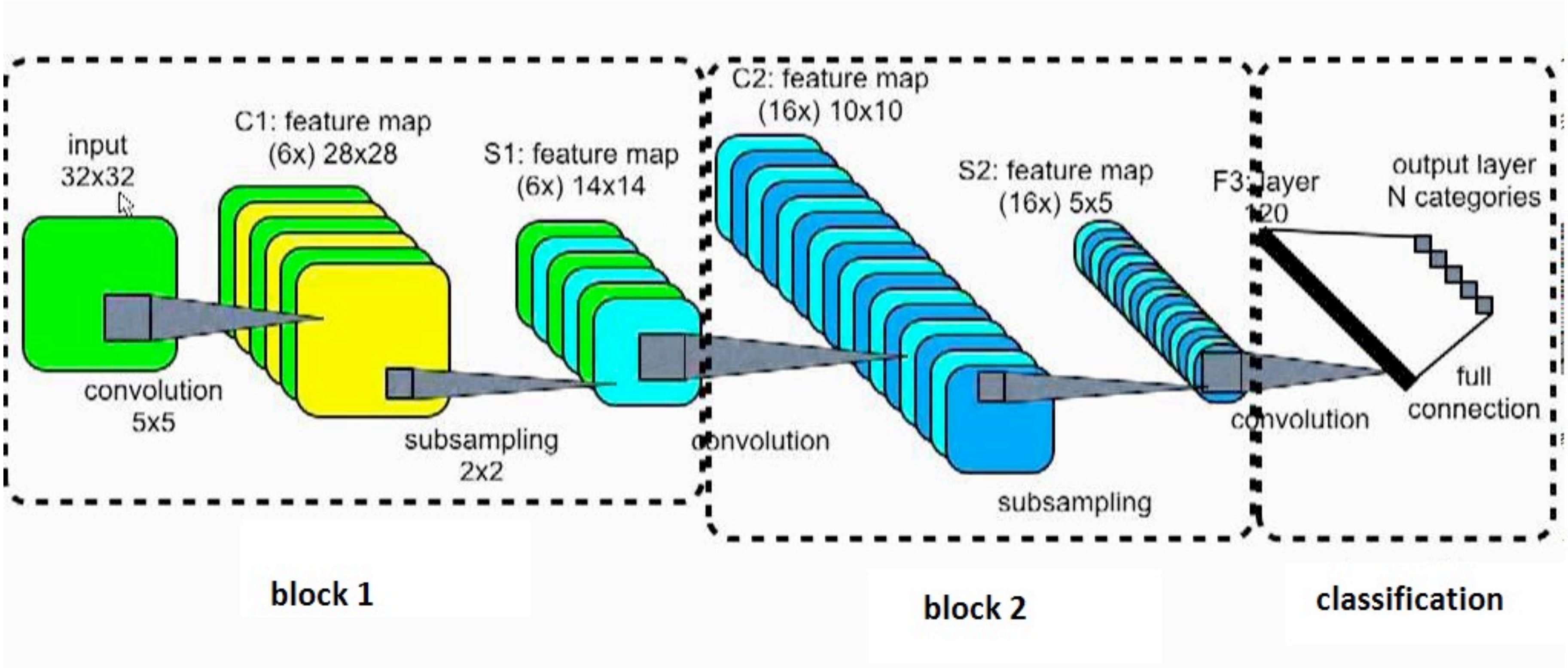
# Convolutional neural networks

- Replace the fully-connected layer by the convolutional layer
- Combine with other layers: non-linear activation, pooling (sub-sampling), normalization, drop out, etc. ) ➔ an elementary block
- For example: a common used block = convolution + ReLu + Pooling



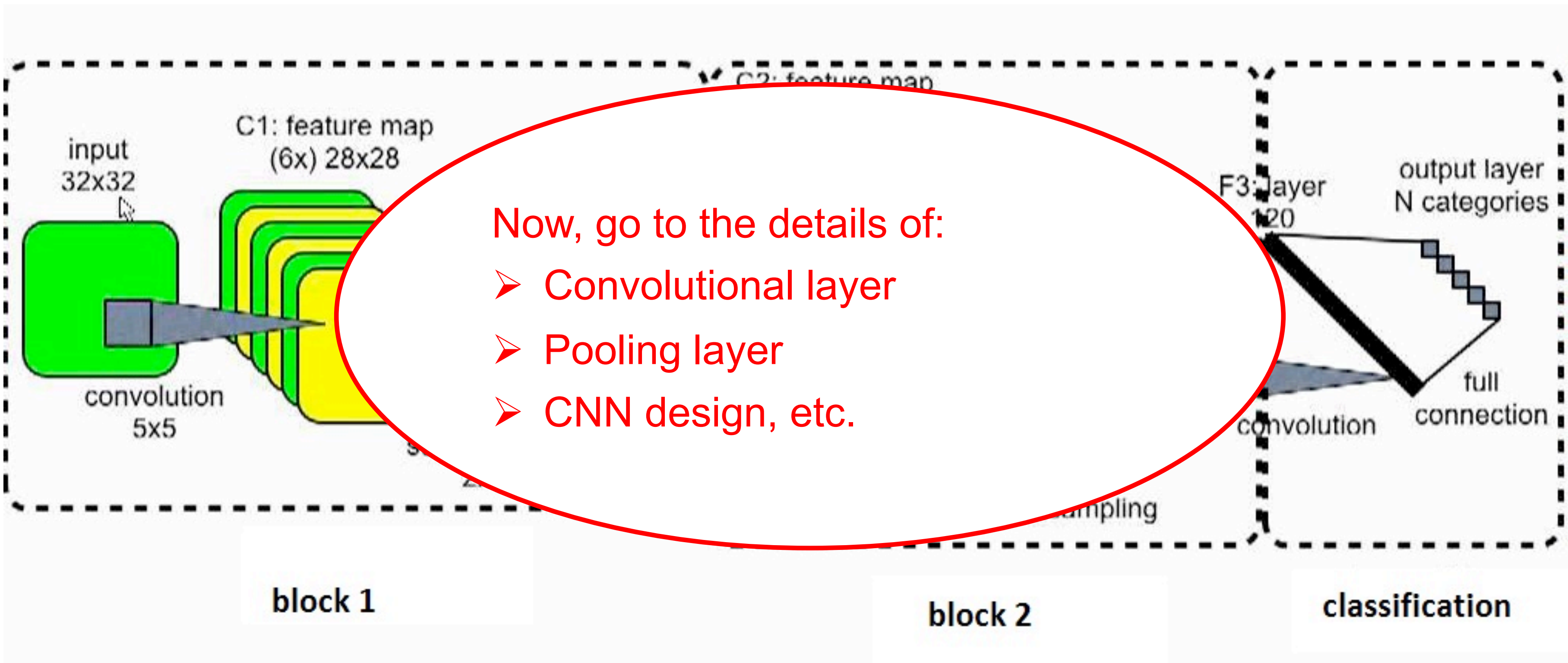
# Convolutional neural networks

- Replace the fully-connected layer by the convolutional layer
- Combine with other layers: non-linear activation, pooling (sub-sampling), normalization, drop out, etc. ) ➔ an elementary block
- For example: a common used block = convolution + ReLu + Pooling
- Stack several block ➔ **CNNs**



# Convolutional neural networks

- Replace the fully-connected layer by the convolutional layer
- Combine with other layers: non-linear activation, pooling (sub-sampling), normalization, drop out, etc. ) ➔ an elementary block
- For example: a common used block = convolution + ReLu + Pooling
- Stack several block ➔ **CNNs**




Now, go to the details of:
- Convolutional layer
- Pooling layer
- CNN design, etc.

# Convolutional neural networks

- Before we go, let's imagine !!!!

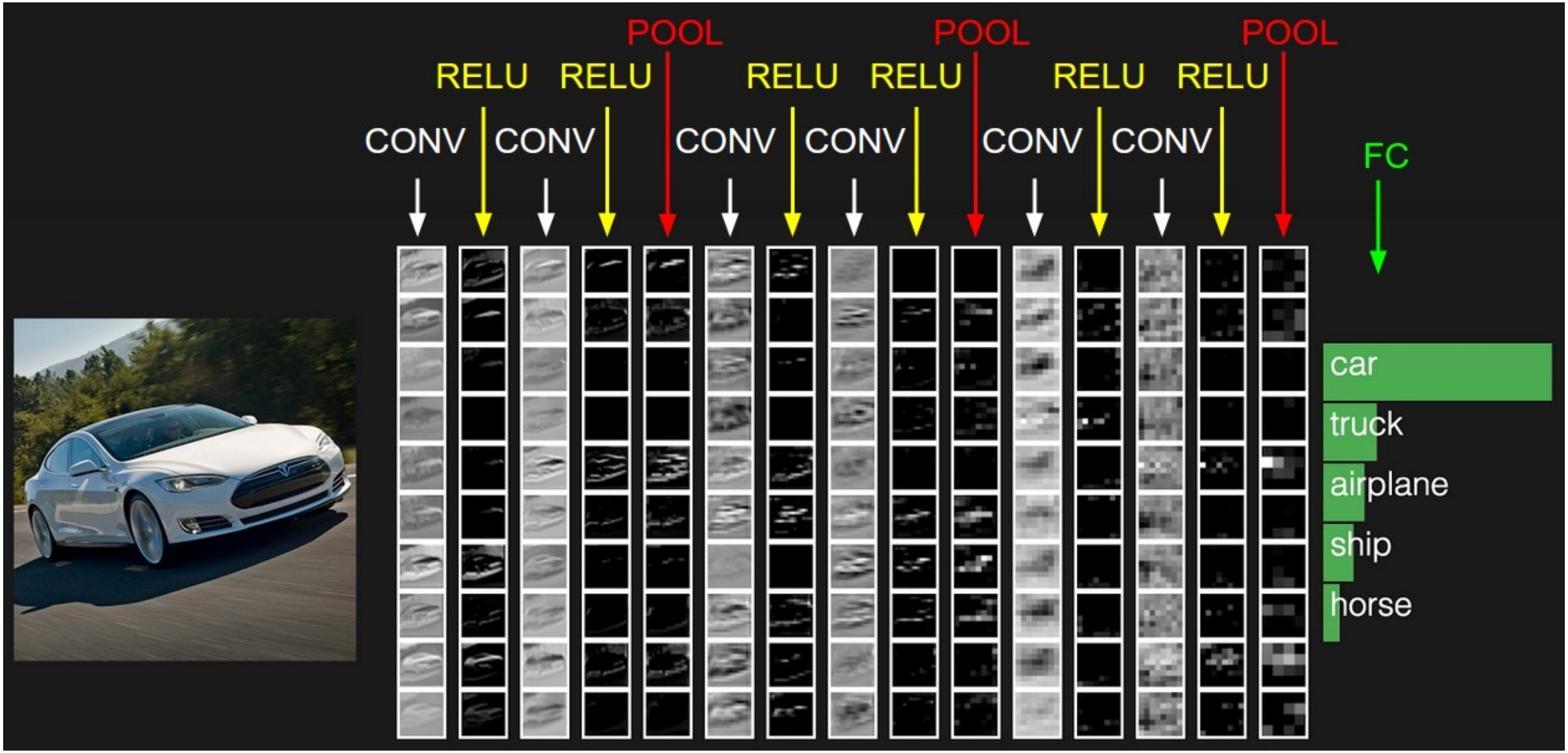



Image credits: cs231n

# Image Convolution

# Image Convolution - Grayscale image

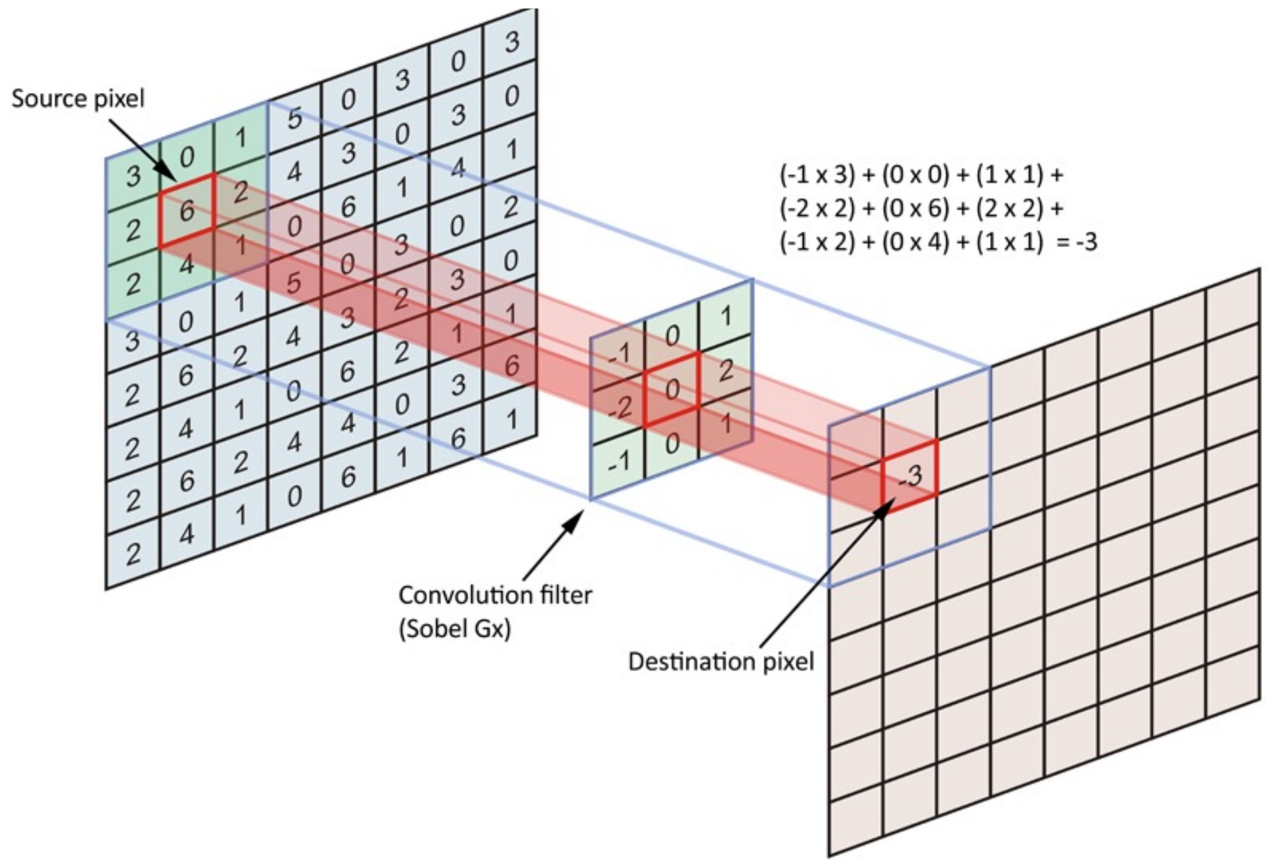
- 2D convolution on image **f** with a filter **h** of size $d{\times}d$

$$f'(i,j) = (f \star h)(i,j) = \sum_{n=-\frac{d-1}{2}}^{\frac{d-1}{2}} \sum_{m=-\frac{d-1}{2}}^{\frac{d-1}{2}} f(i-n, m-j)h(n,m)$$

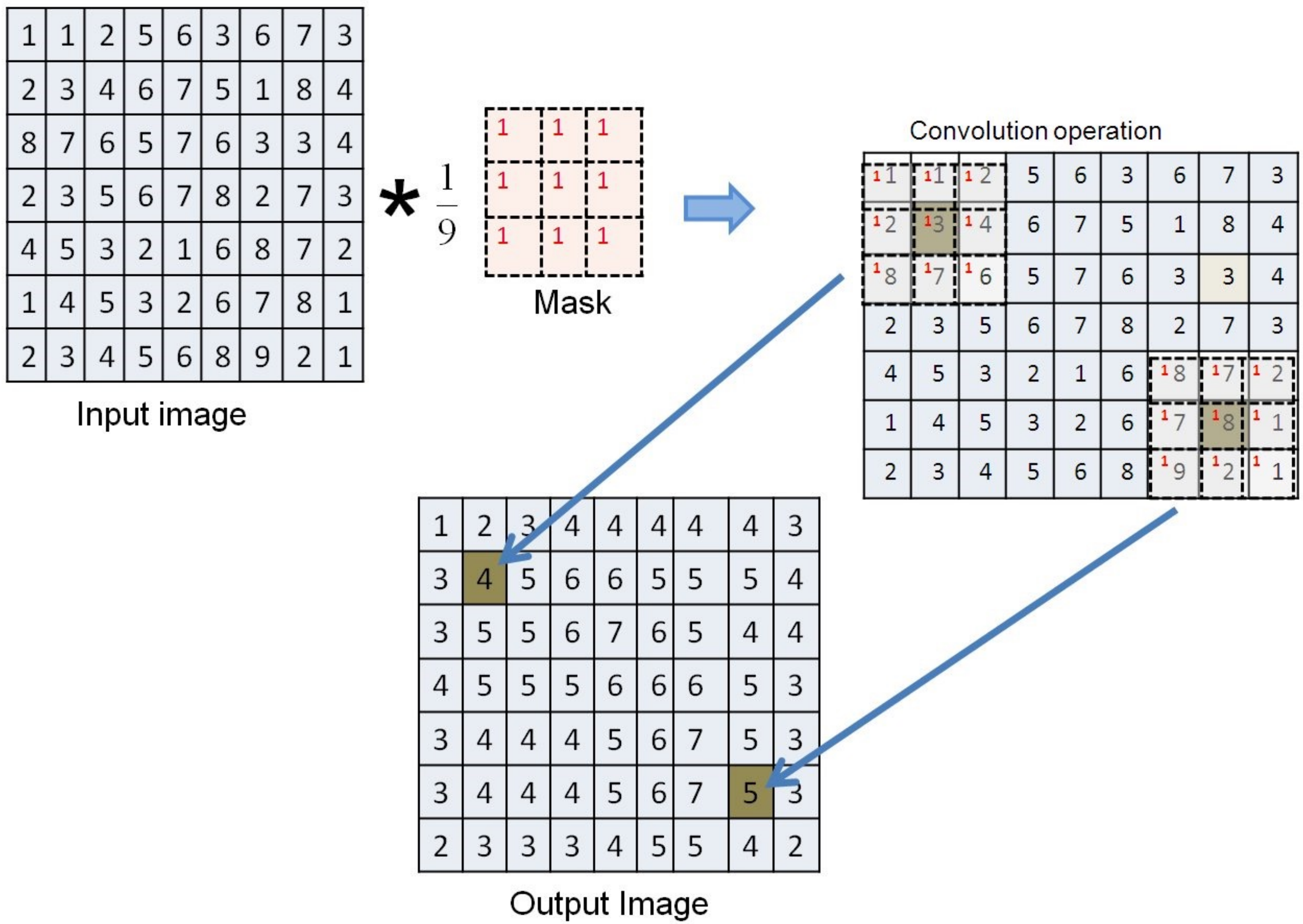
- <u>In practice:</u>

→ Center the filter **h** at each pixel location

→ Compute the weighted sum



# Image Convolution - Grayscale image



# Image Convolution - Example

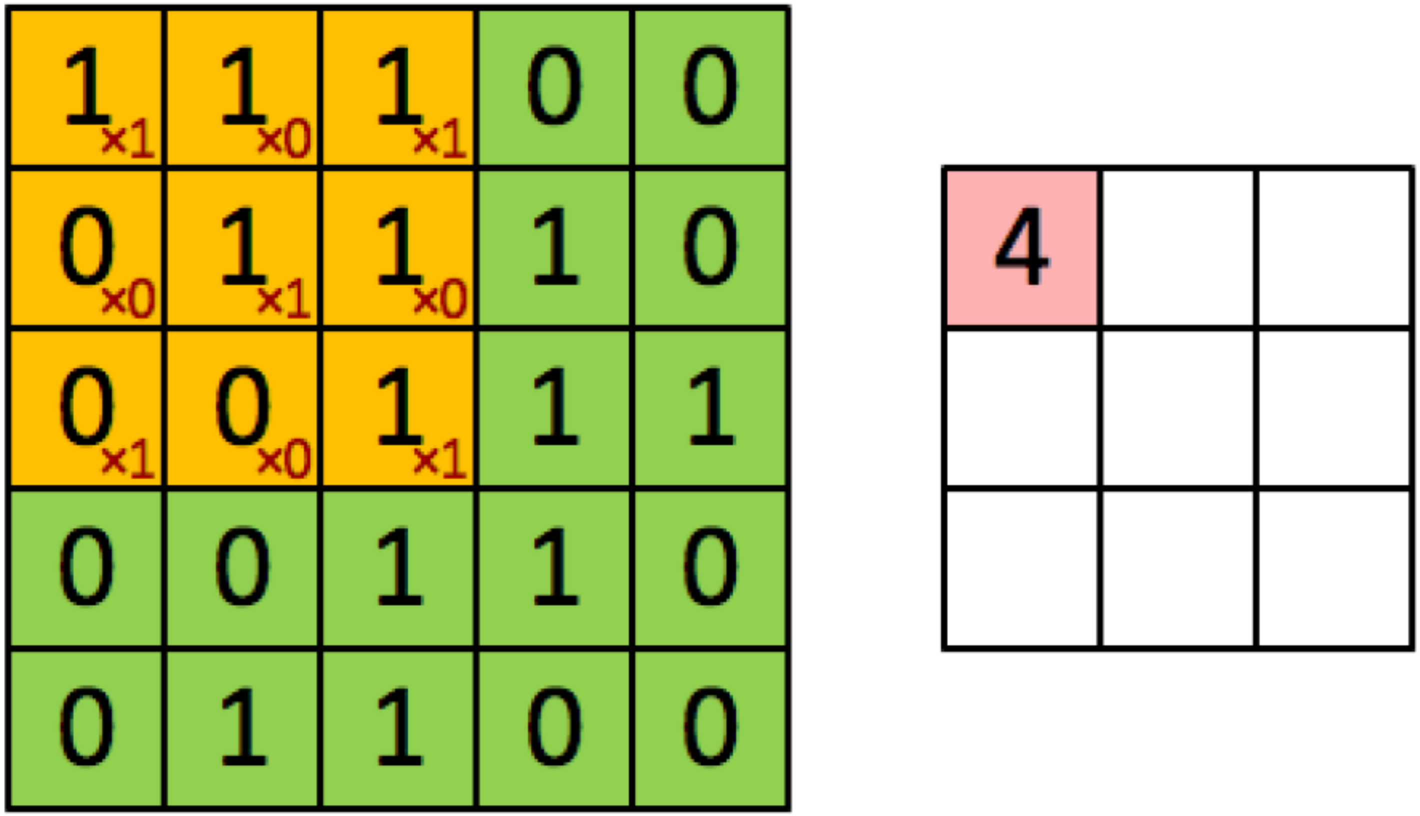
| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Image

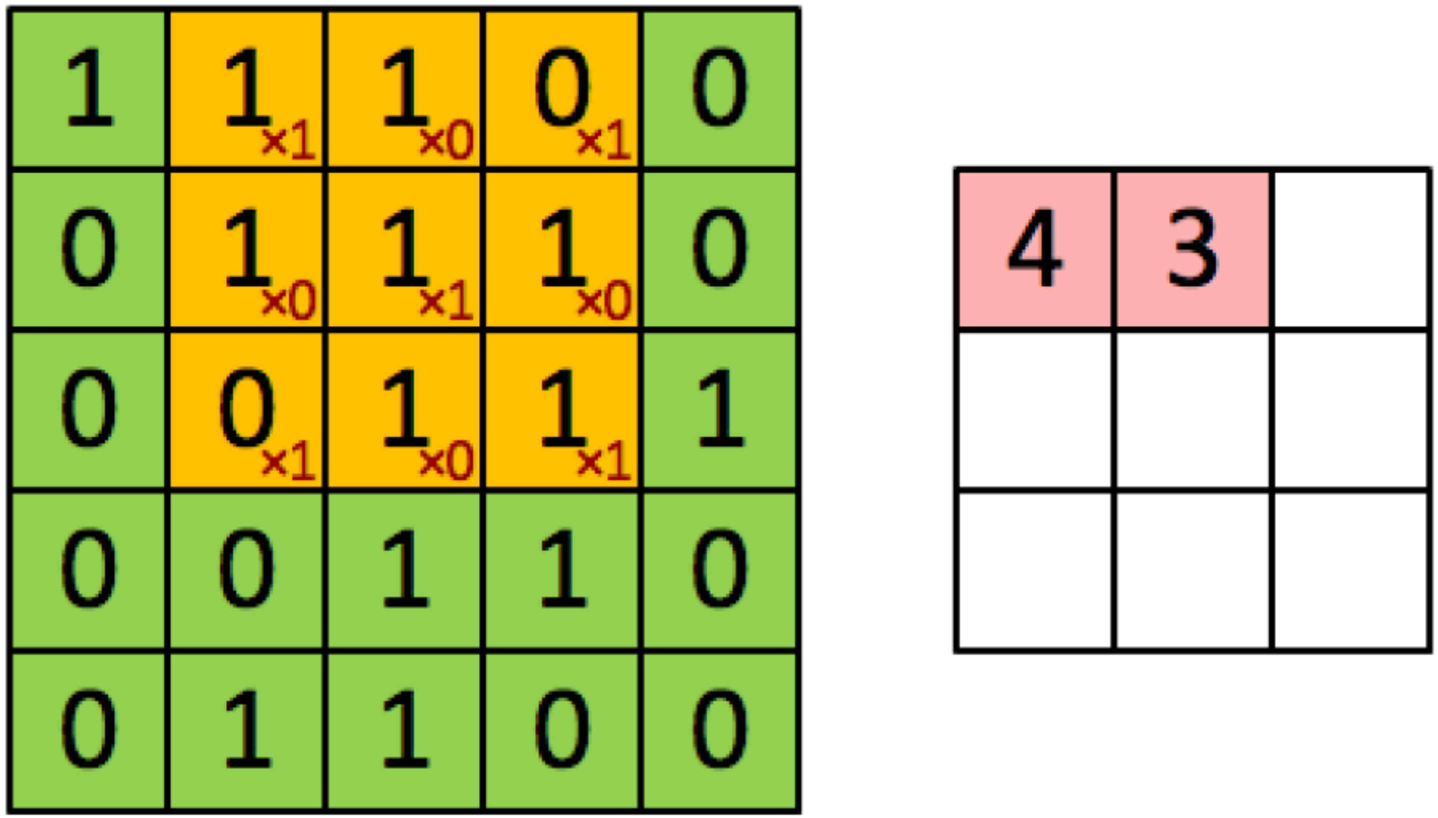
| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

**Filter/kernel**

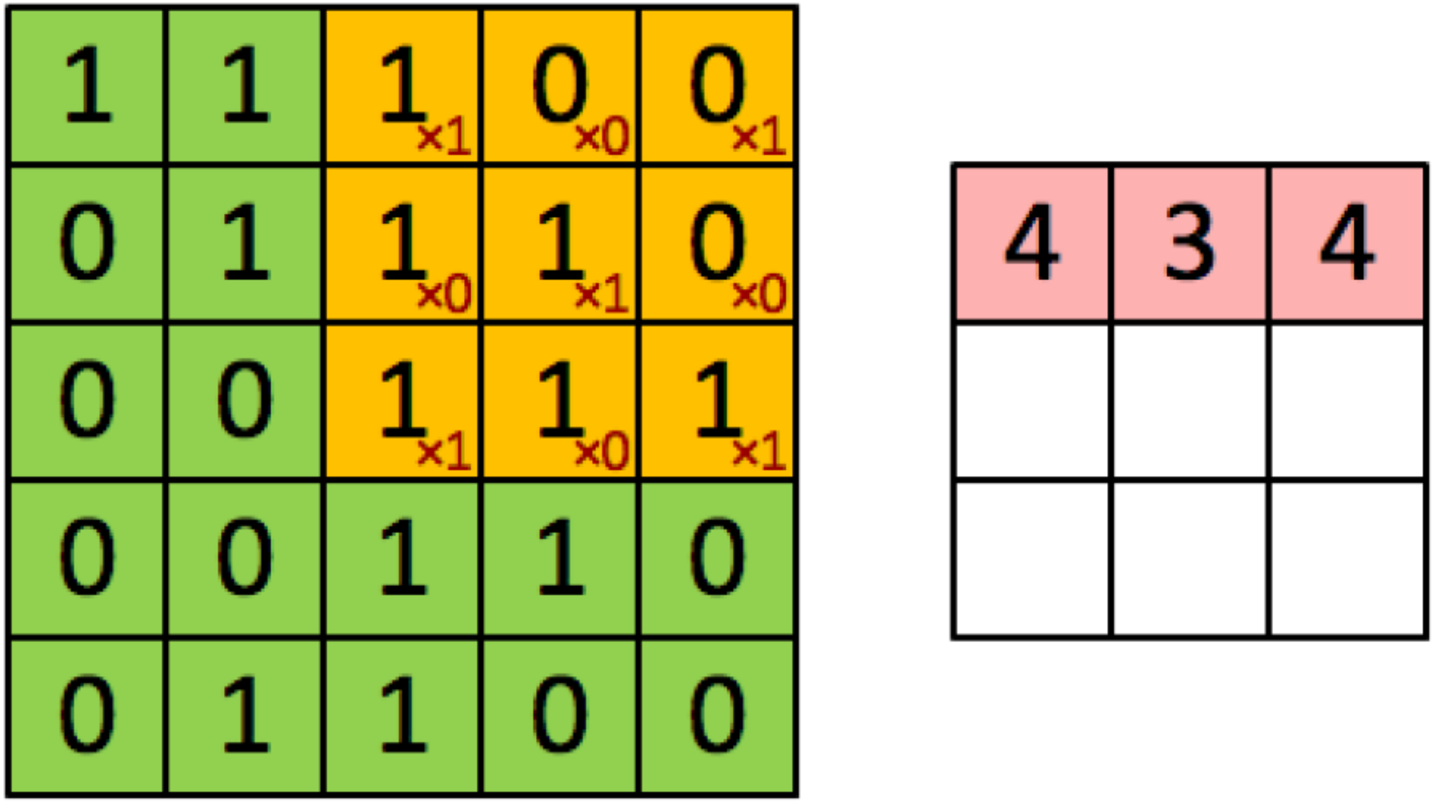
# Image Convolution - Example



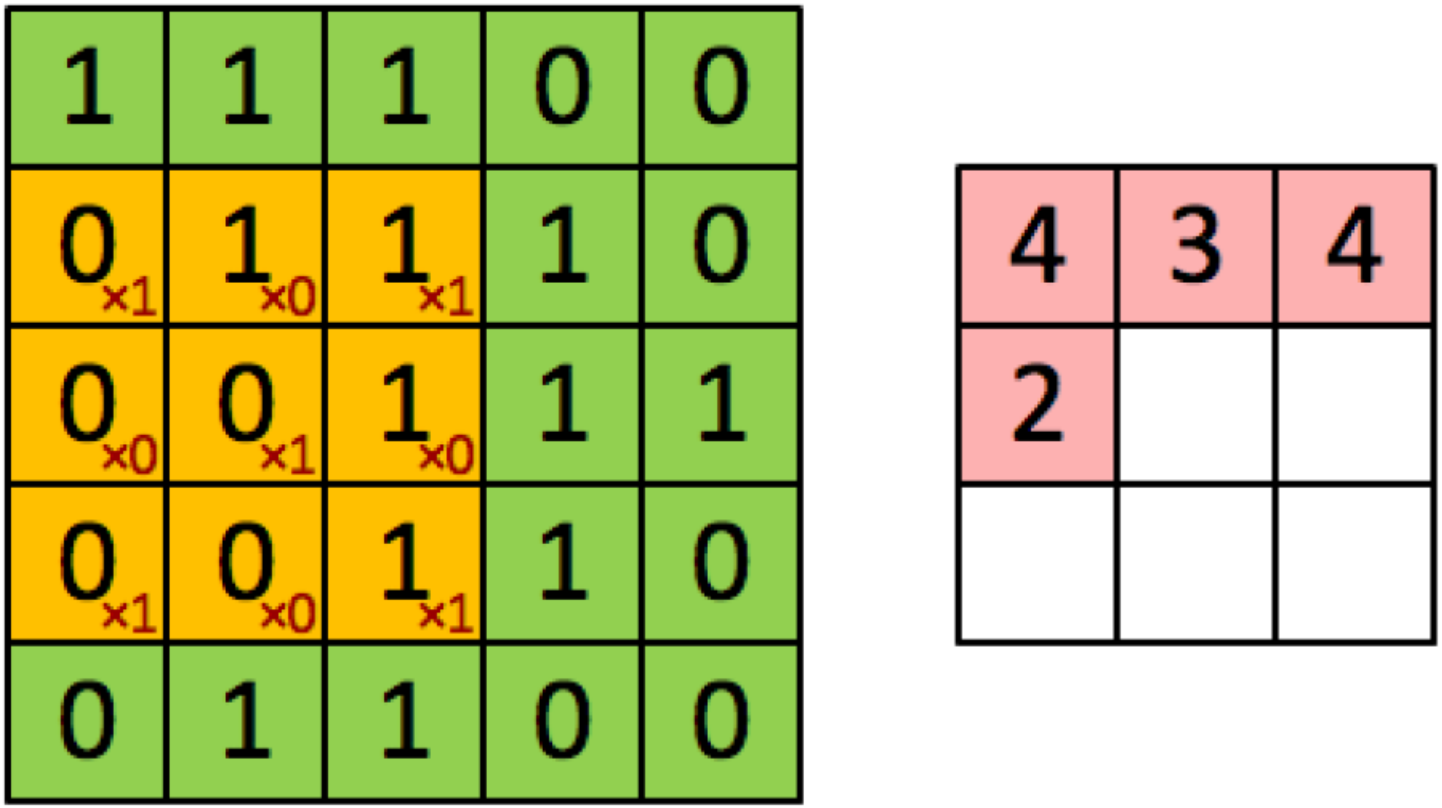
# Image Convolution - Example



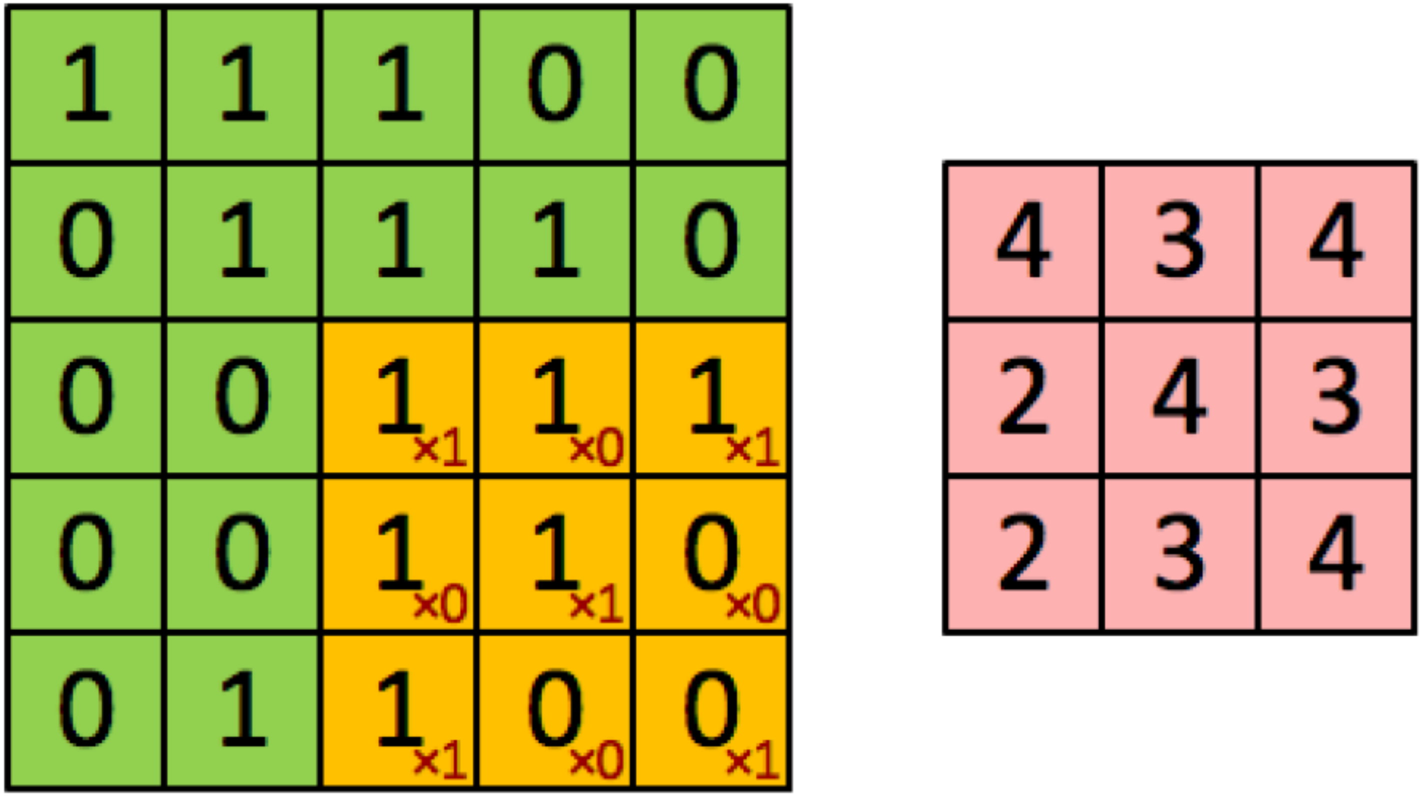
# Image Convolution - Example



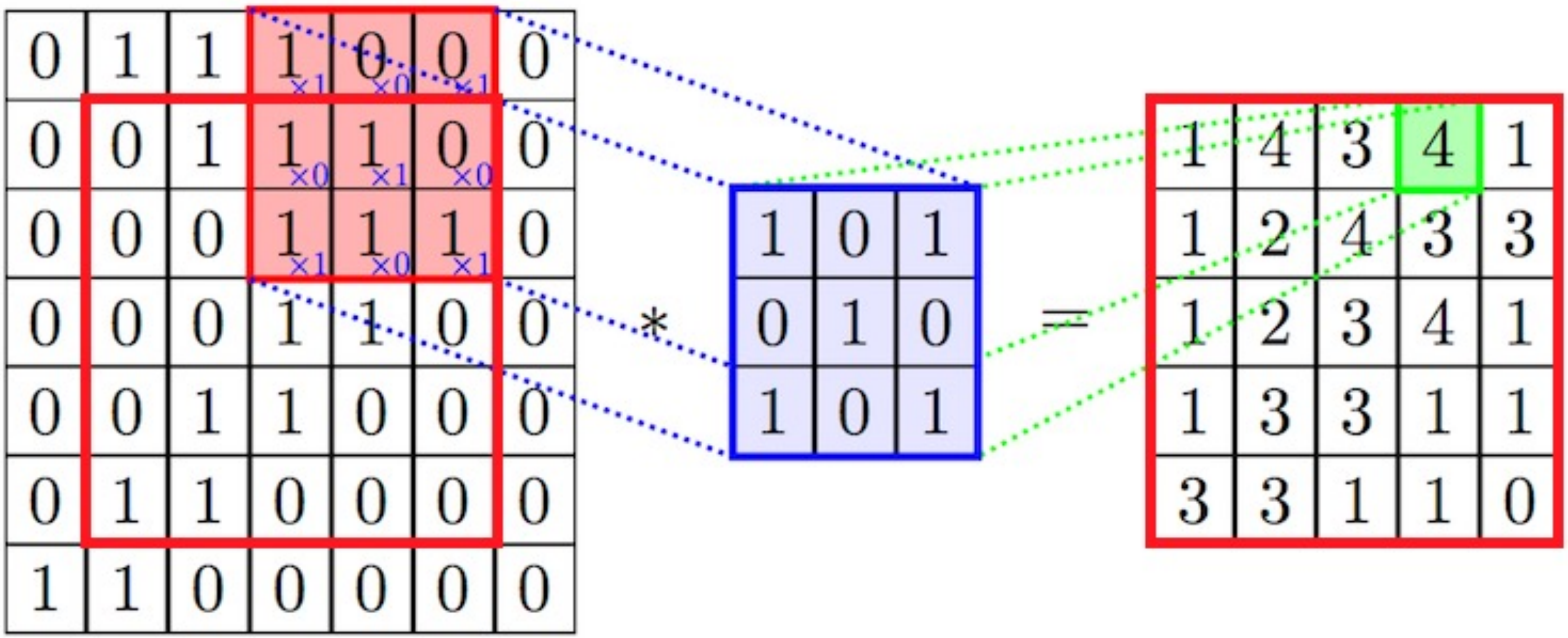
# Image Convolution - Example



# Image Convolution - Example



# Image Convolution - Example




<u>In general:</u>

$$\boldsymbol{W_{f'} = W_f - W_h + 1}$$

$$\boldsymbol{H_{f'} = H_f - H_h + 1}$$

# Image Convolution - padding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 5 | 6 | 3 | 6 | 7 | 3 |
| 2 | 3 | 4 | 6 | 7 | 5 | 1 | 8 | 4 |
| 8 | 7 | 6 | 5 | 7 | 6 | 3 | 3 | 4 |
| 2 | 3 | 5 | 6 | 7 | 8 | 2 | 7 | 3 |
| 4 | 5 | 3 | 2 | 1 | 6 | 8 | 7 | 2 |
| 1 | 4 | 5 | 3 | 2 | 6 | 7 | 8 | 1 |
| 2 | 3 | 4 | 5 | 6 | 8 | 9 | 2 | 1 |

Input Image

$$* \quad \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

filter

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 2 | 5 | 6 | 3 | 6 | 7 | 3 | 0 | 0 |
| 0 | 0 | 2 | 3 | 4 | 6 | 7 | 5 | 1 | 8 | 4 | 0 | 0 |
| 0 | 0 | 8 | 7 | 6 | 5 | 7 | 6 | 3 | 3 | 4 | 0 | 0 |
| 0 | 0 | 2 | 3 | 5 | 6 | 7 | 8 | 2 | 7 | 3 | 0 | 0 |
| 0 | 0 | 4 | 5 | 3 | 2 | 1 | 6 | 8 | 7 | 2 | 0 | 0 |
| 0 | 0 | 1 | 4 | 5 | 3 | 2 | 6 | 7 | 8 | 1 | 0 | 0 |
| 0 | 0 | 2 | 3 | 4 | 5 | 6 | 8 | 9 | 2 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Zero Padding(option1)

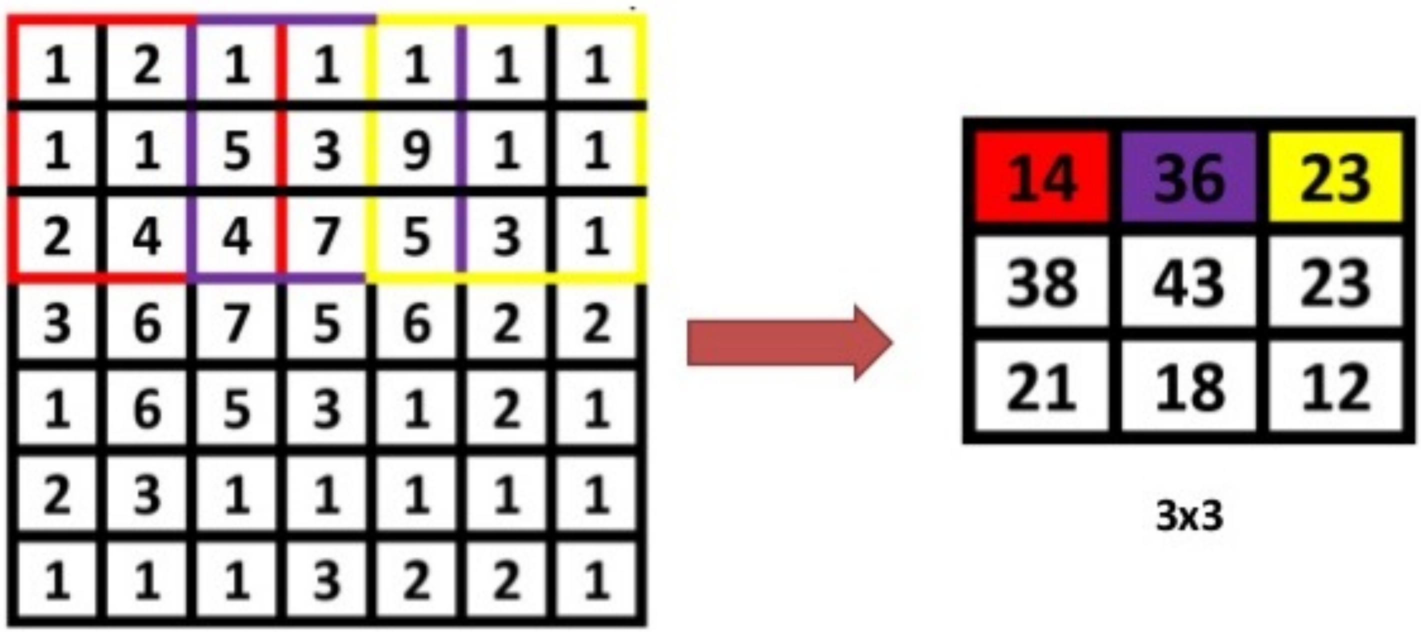
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 3 | 4 | 6 | 7 | 5 | 1 | 8 | 4 | 4 | 4 |
| 1 | 1 | 1 | 1 | 2 | 5 | 6 | 3 | 6 | 7 | 3 | 3 | 3 |
| 1 | 1 | 1 | 1 | 2 | 5 | 6 | 3 | 6 | 7 | 3 | 3 | 7 |
| 3 | 2 | 2 | 3 | 4 | 6 | 7 | 5 | 1 | 8 | 4 | 4 | 8 |
| 7 | 8 | 8 | 7 | 6 | 5 | 7 | 6 | 3 | 3 | 4 | 4 | 3 |
| 3 | 2 | 2 | 3 | 5 | 6 | 7 | 8 | 2 | 7 | 3 | 3 | 7 |
| 5 | 4 | 4 | 5 | 3 | 2 | 1 | 6 | 8 | 7 | 2 | 2 | 7 |
| 4 | 1 | 1 | 4 | 5 | 3 | 2 | 6 | 7 | 8 | 1 | 1 | 8 |
| 3 | 2 | 2 | 3 | 4 | 5 | 6 | 8 | 9 | 2 | 1 | 1 | 2 |
| 3 | 2 | 2 | 3 | 4 | 5 | 6 | 8 | 9 | 2 | 1 | 1 | 2 |
| 3 | 1 | 1 | 4 | 5 | 3 | 2 | 6 | 7 | 8 | 1 | 1 | 2 |

Repetition of border pixels(option 2

➔ Output has the same size as input !

# Image Convolution - stride

Example : stride of 2

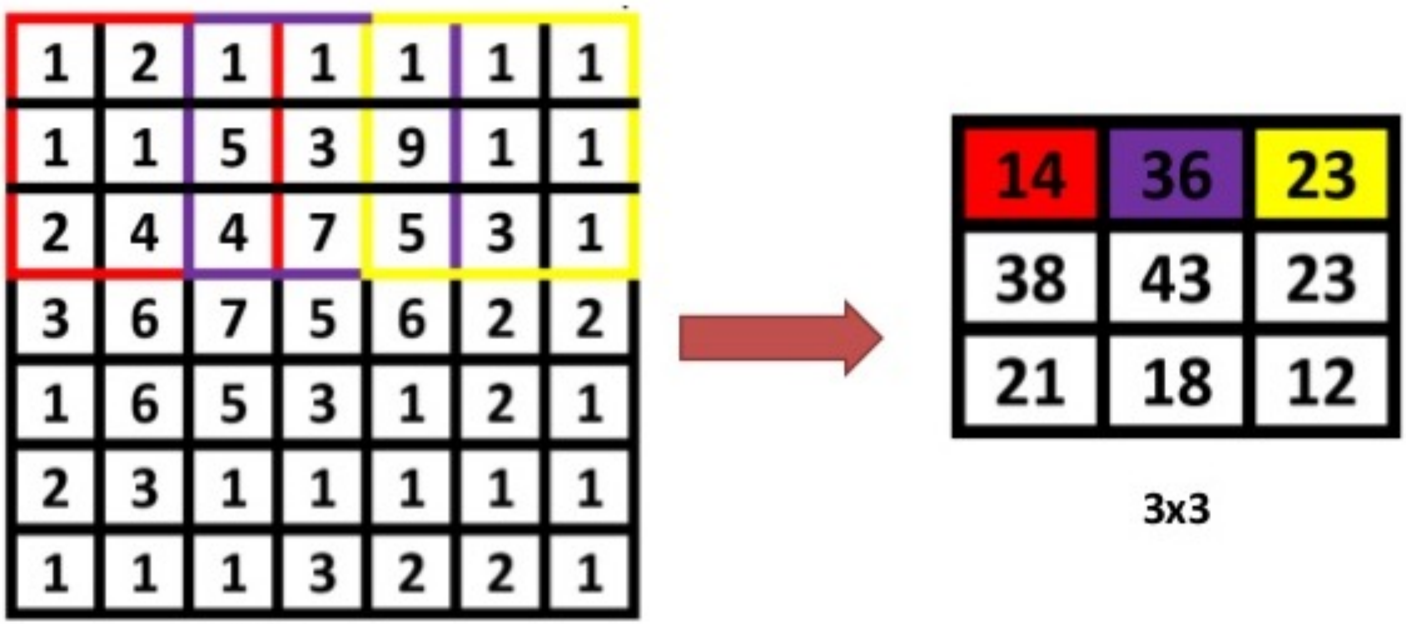



7x7 Image

3x3

# Image Convolution – output size

$$W_{output} = \frac{(W_{input} + 2 \times padding) - W_{filter}}{stride} + 1$$

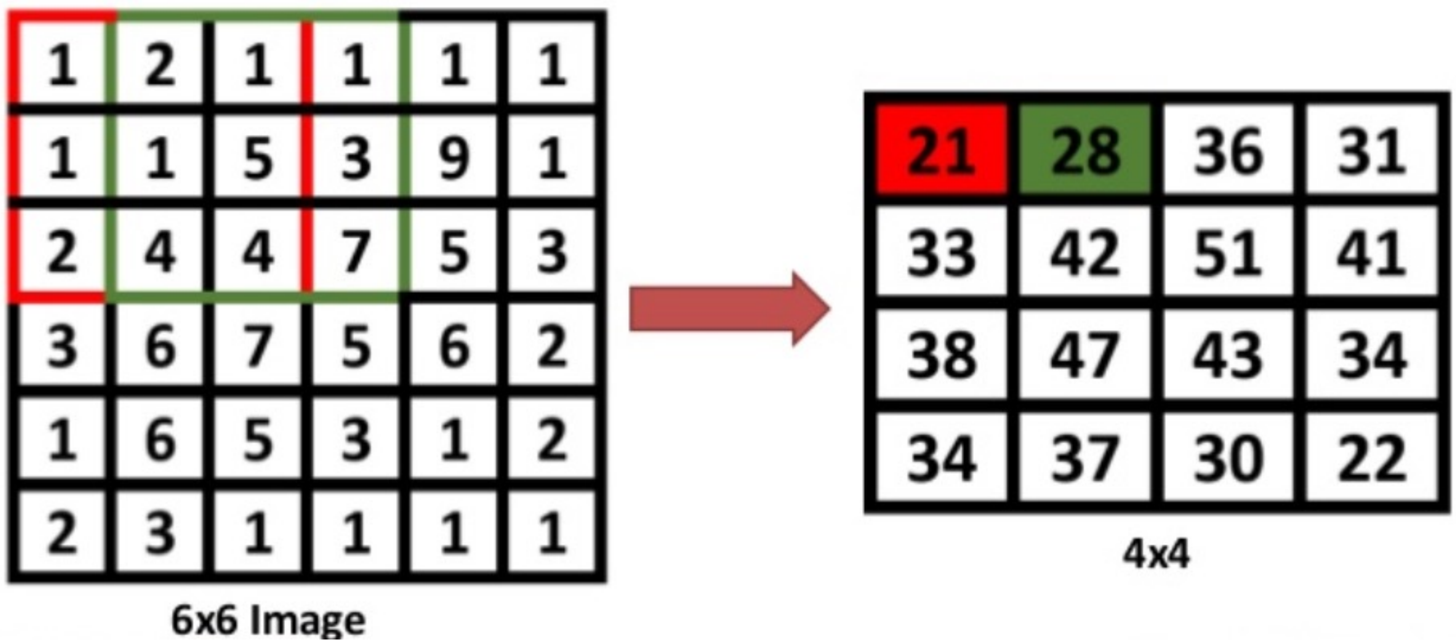



7x7 Image

3x3

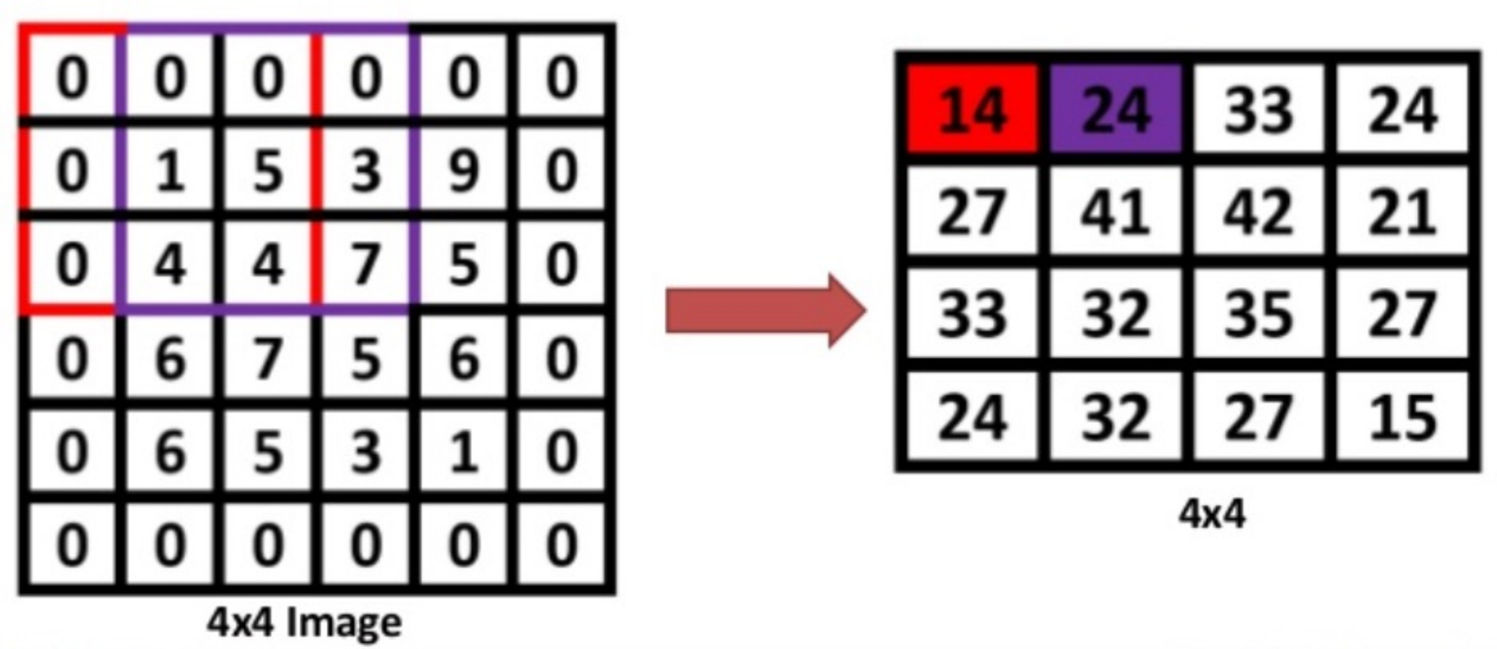
# Image Convolution - output size

$$W_{output} = \frac{(W_{input} + 2 \times padding) - W_{filter}}{stride} + 1$$



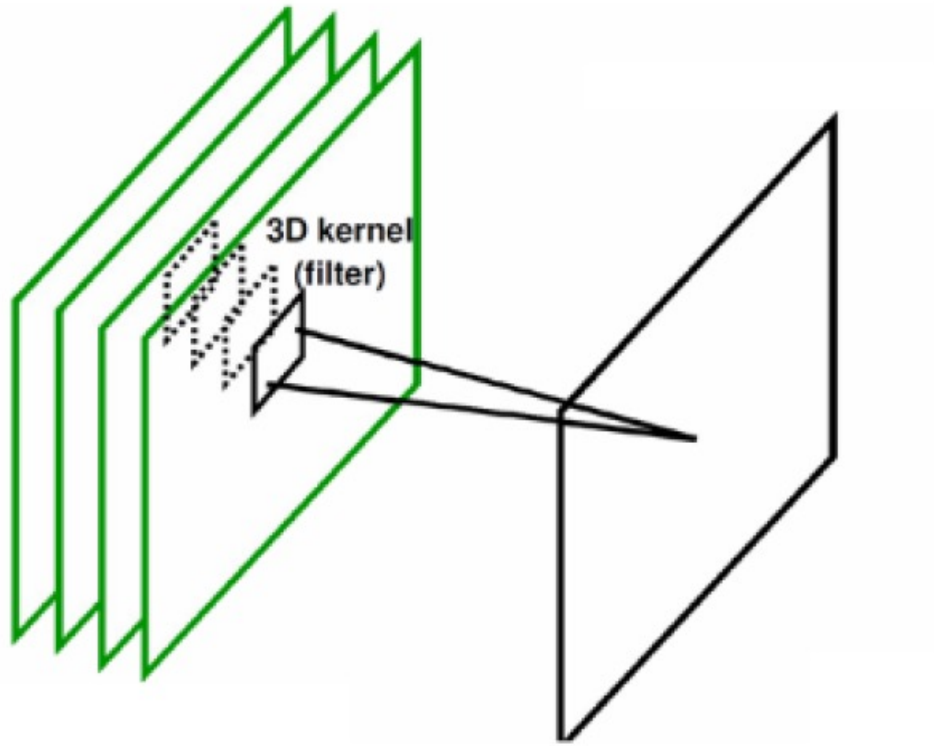
# Image Convolution - output size

$$W_{output} = \frac{(W_{input} + 2 \times padding) - W_{filter}}{stride} + 1$$



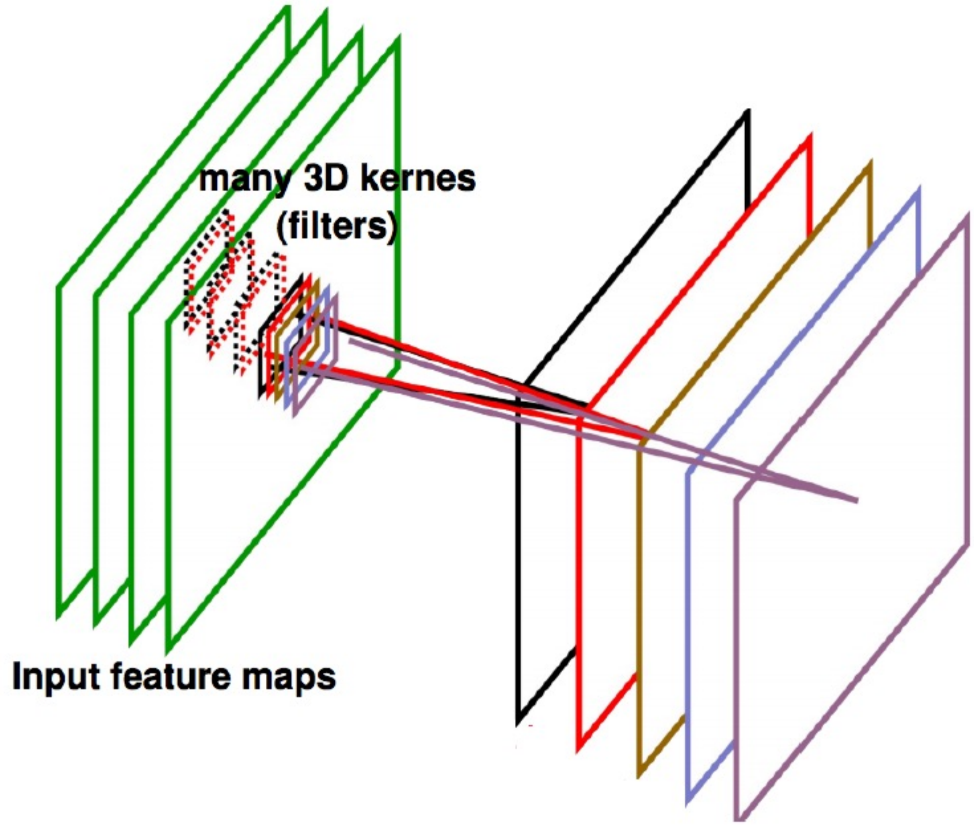
# Convolution: multi-channel image

- Each filter has the size of $d \times d \times D$ (spatial and depth)
- Common color image: D=3 (3 bands RGB)
- Example of multispectral image: D=4 (RGB+IR)



# Convolution: multi-channel image

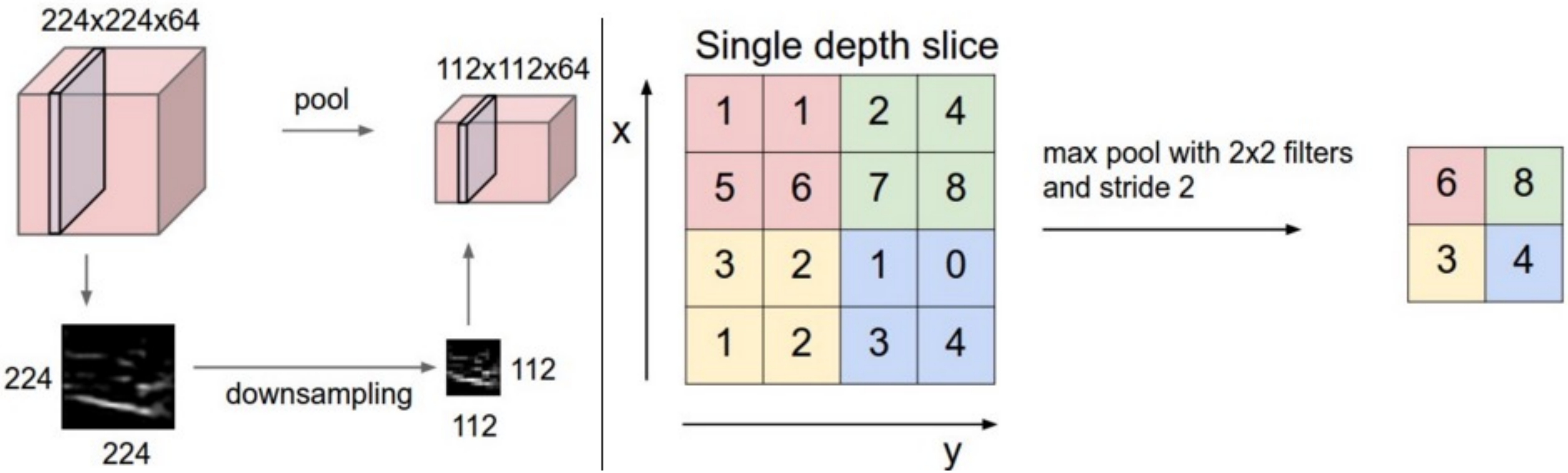
- Each filter has the size of $d \times d \times D$ (spatial and depth)
- Common color image: D=3 (3 bands RGB)
- Example of multispectral image: D=4 (RGB+IR)
- Use multiple filters ➔ multiple output filtered images



# Pooling Layer

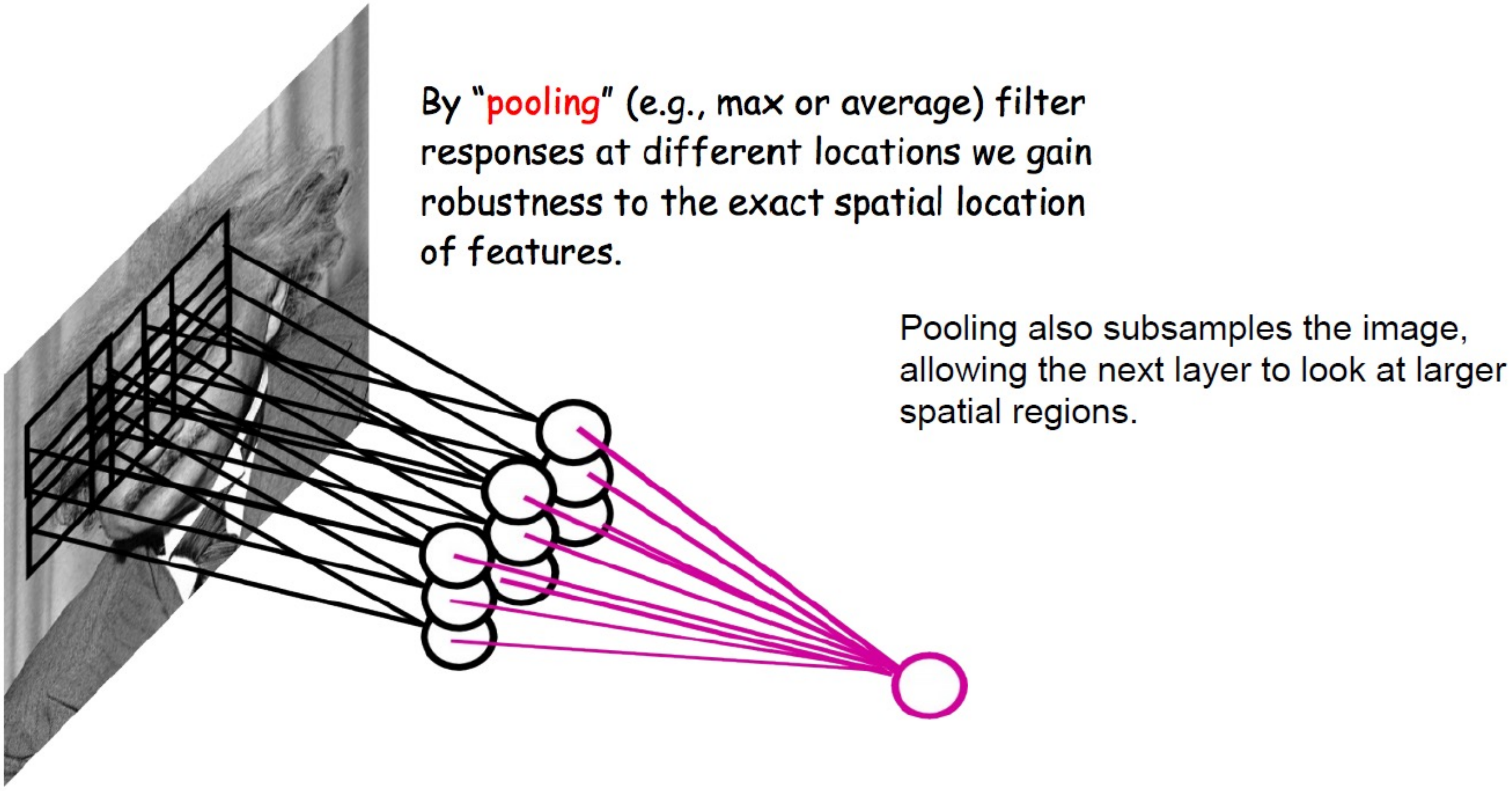
# Pooling layer

- spatial down-sampling (sub-sampling) ! reduce spatial size
- reduce number of parameters and computation time
- increase shift/translation invariance
- average pooling or max-pooling (mostly used)



# Pooling layer

- spatial down-sampling (sub-sampling) ! reduce spatial size
- reduce number of parameters and computation time
- increase shift/translation invariance
- average pooling or max-pooling (mostly used)



# Typical CNNs

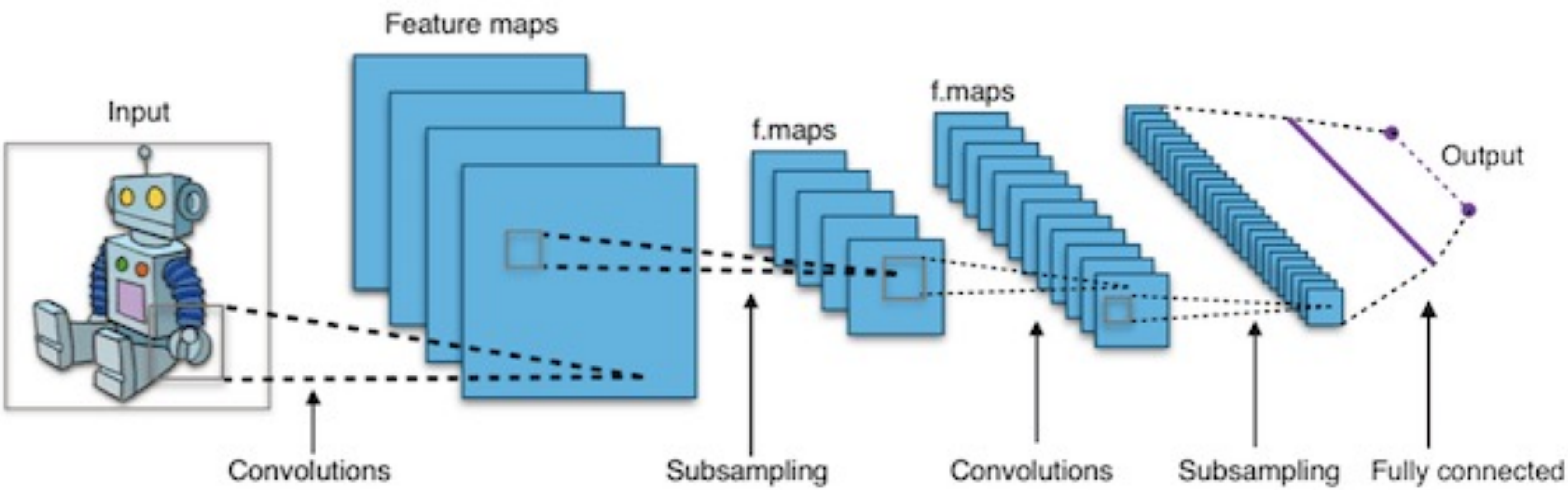
# Typical CNN architecture

- A typical CNN architecture includes:

  **Input image → [Conv - ReLu] → (Pool) → [Conv - ReLu] → (Pool)**

  **→ ... → FC (fully-connected) → Softmax (classification) → Output**

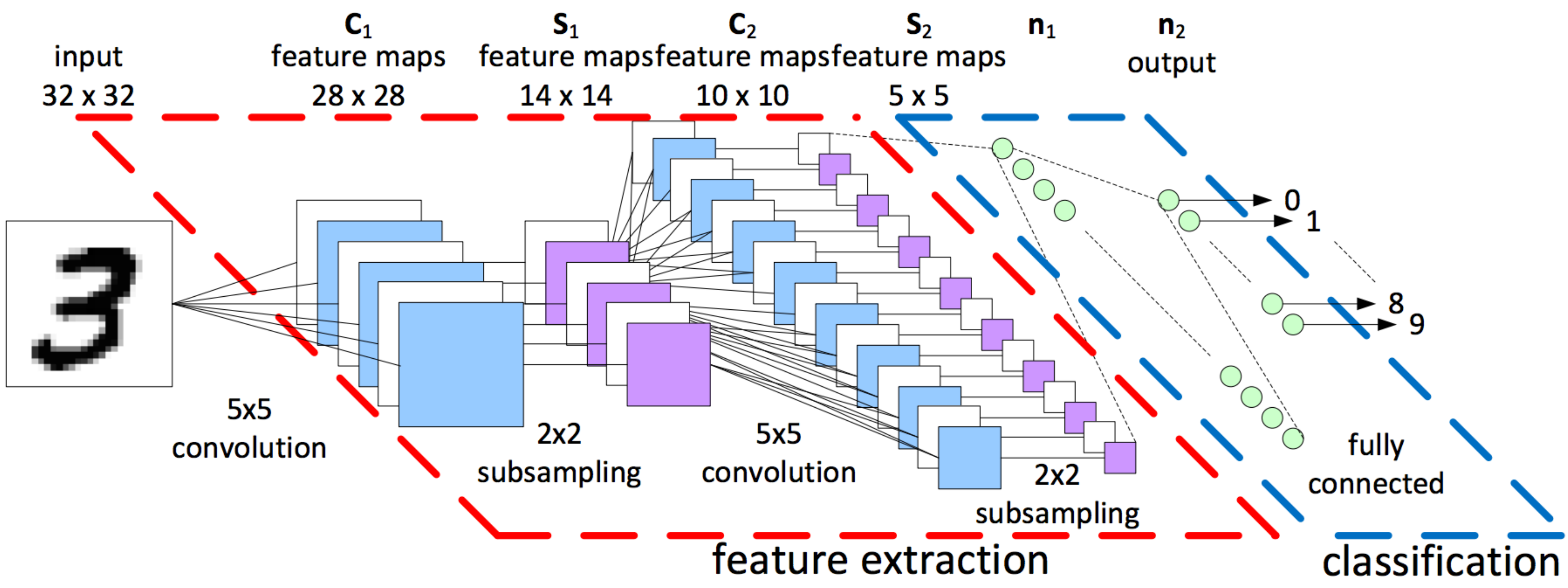
- Optional (for practical use): stride, normalization, dropout, etc.



# Typical CNN architecture

- A typical CNN architecture includes:

**Input image → [Conv - ReLu] → (Pool) → [Conv - ReLu] → (Pool) → ... → FC (fully-connected) → Softmax (classification) → Output**



# Typical CNN architecture

- Let look back to the example

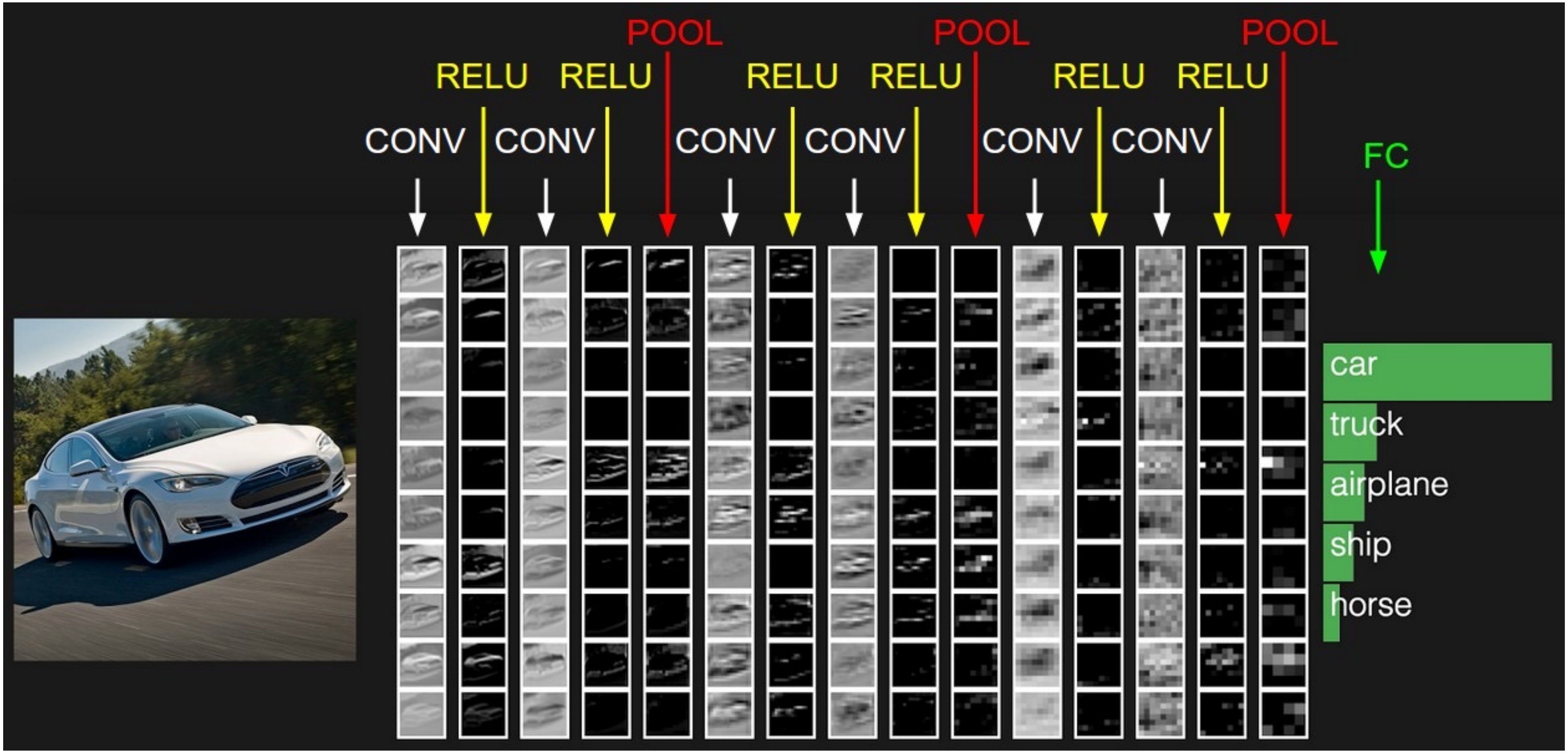
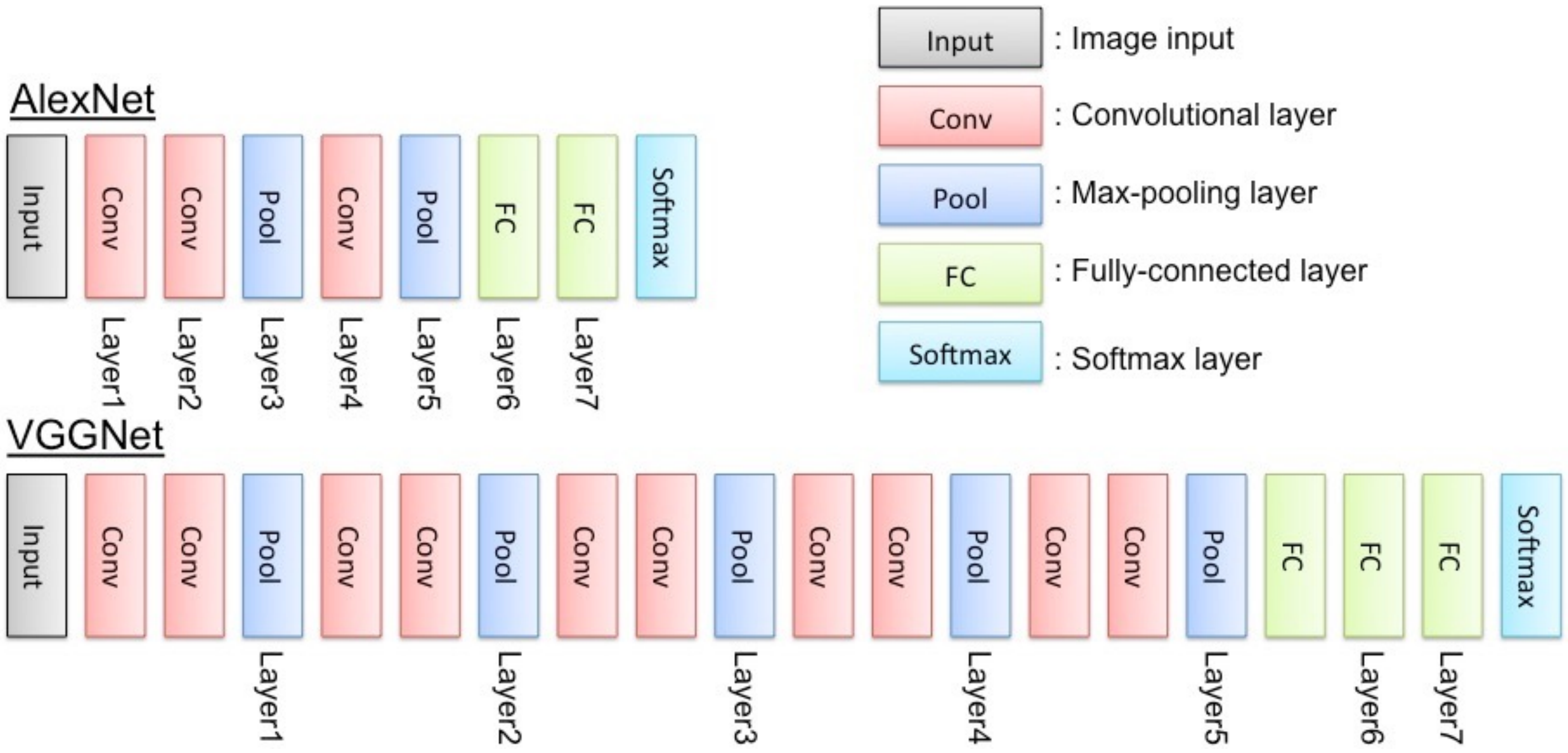
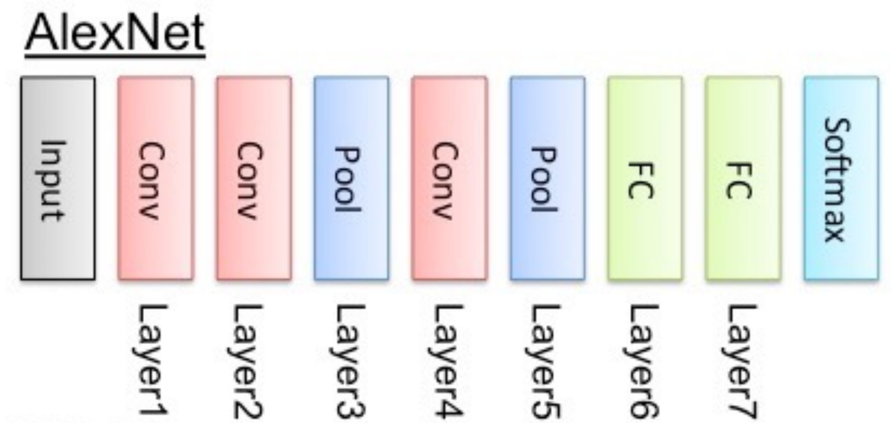


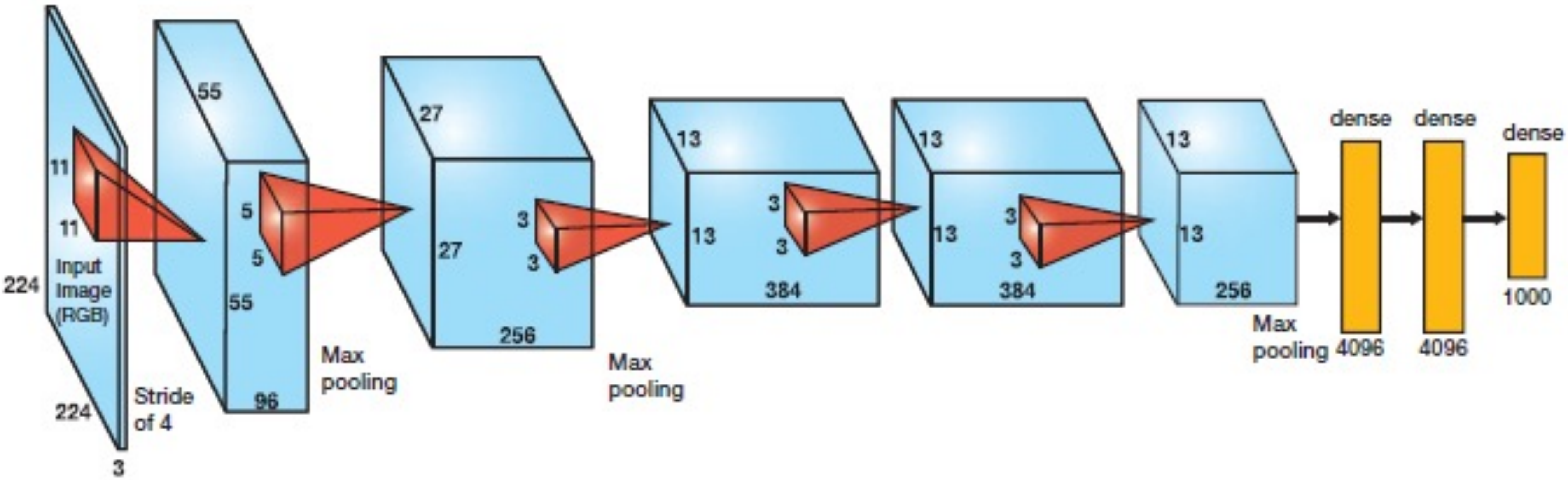

Image credits: [cs231n]

# Typical CNN architecture



# Typical CNN architecture



# Lab assignment

<u>**Practical lab:**</u> **Training convolutional neural networks for digit recognition (MNIST) and for color object detection (CIFAR10) using Pytorch**

- **Download, complete and submit** the notebook from Moodle **(R3A13_lab2_assign.ipynb)**

- Note that you need to submit **your compiled notebook with all outputs**

# References and Sources

- **Convolutional neural networks for visual recognition** - Stanford
  https://cs231n.github.io/
- **Neural networks and deep learning** (free online book)
  http://neuralnetworksanddeeplearning.com/index.html
- **Machine learning course** – Oxford
  https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/
- **Neural network course** - Hugo Larochelle
  https://info.usherbrooke.ca/hlarochelle/neural_networks/content.html
- **Deep learning course** – François Fleuret
  https://fleuret.org/dlc/