# CoE202
# Fundamentals of Artificial intelligence
# <Big Data Analysis and Machine Learning>

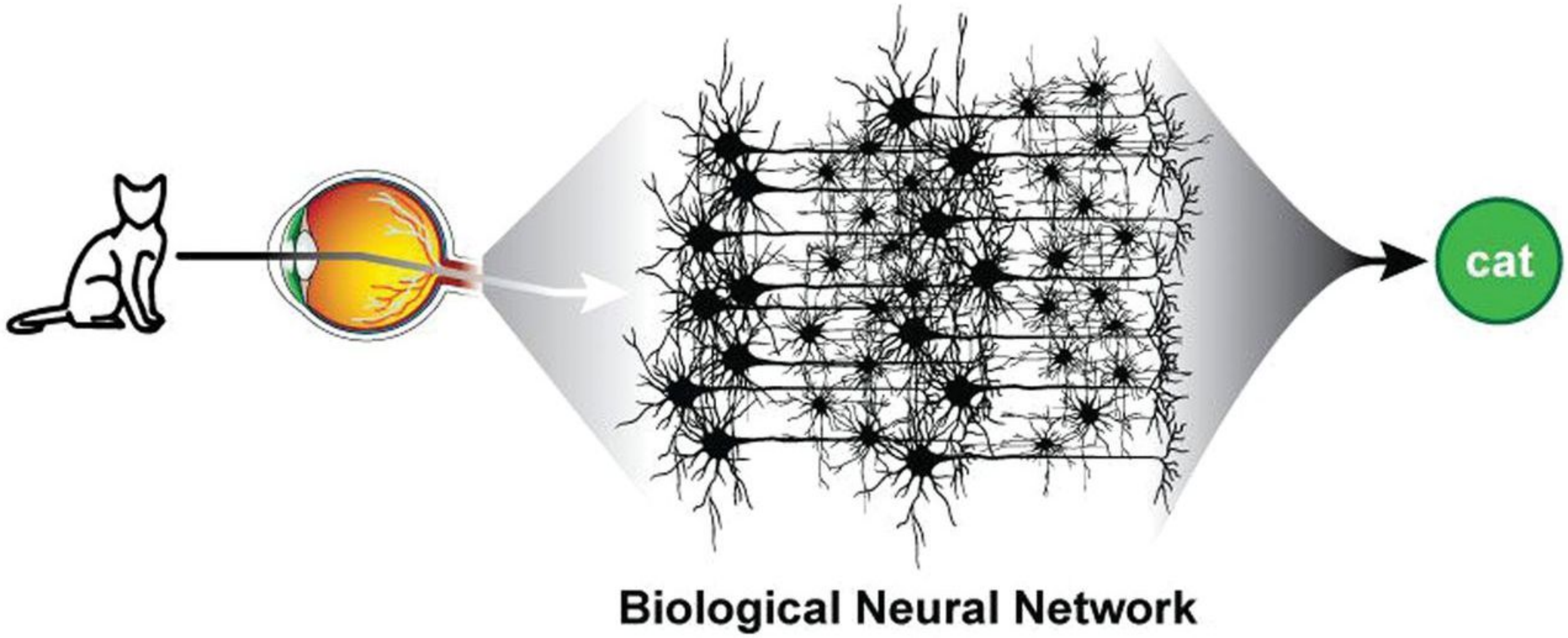
## Convolutional Neural Network

Prof. Young-Gyu Yoon
School of EE, KAIST

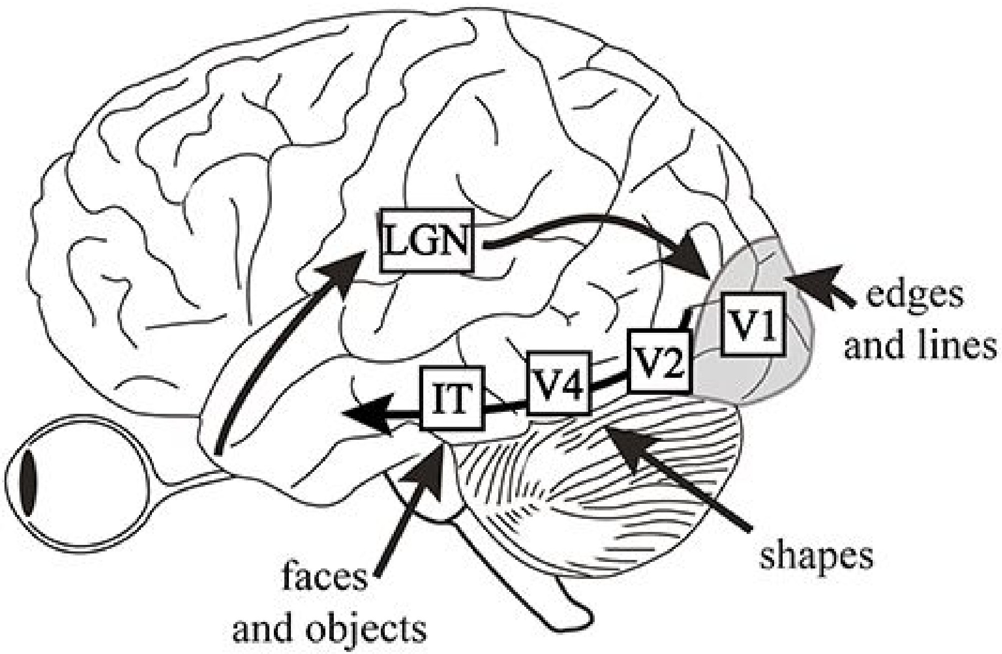
# Contents

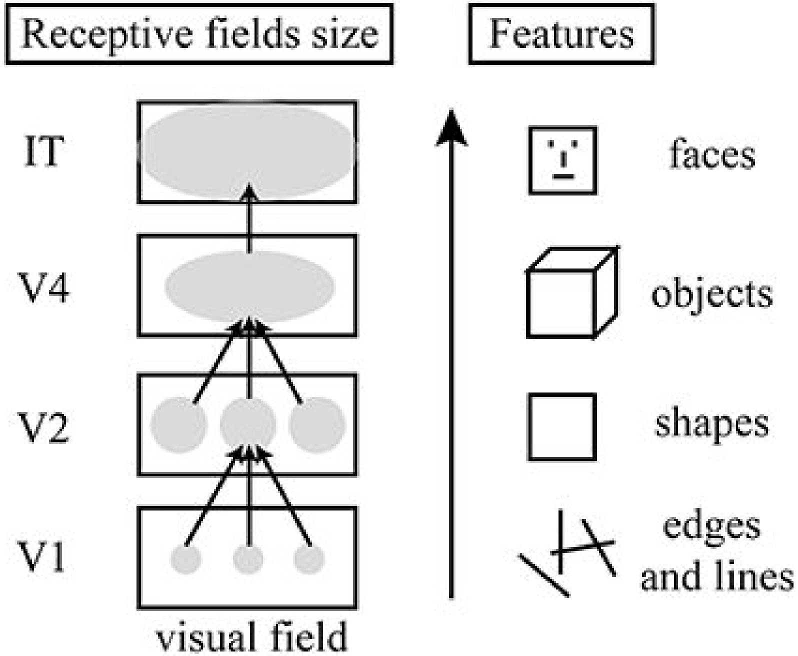
- Recap
  - Deep learning libraries
  - What we (human) do and what library does
  - Task formulation
  - NN design flow
- Biological vision
- Convolution operation
- ConvNet as a special case of NN
- Building blocks of ConvNet

# Biological vision information processing



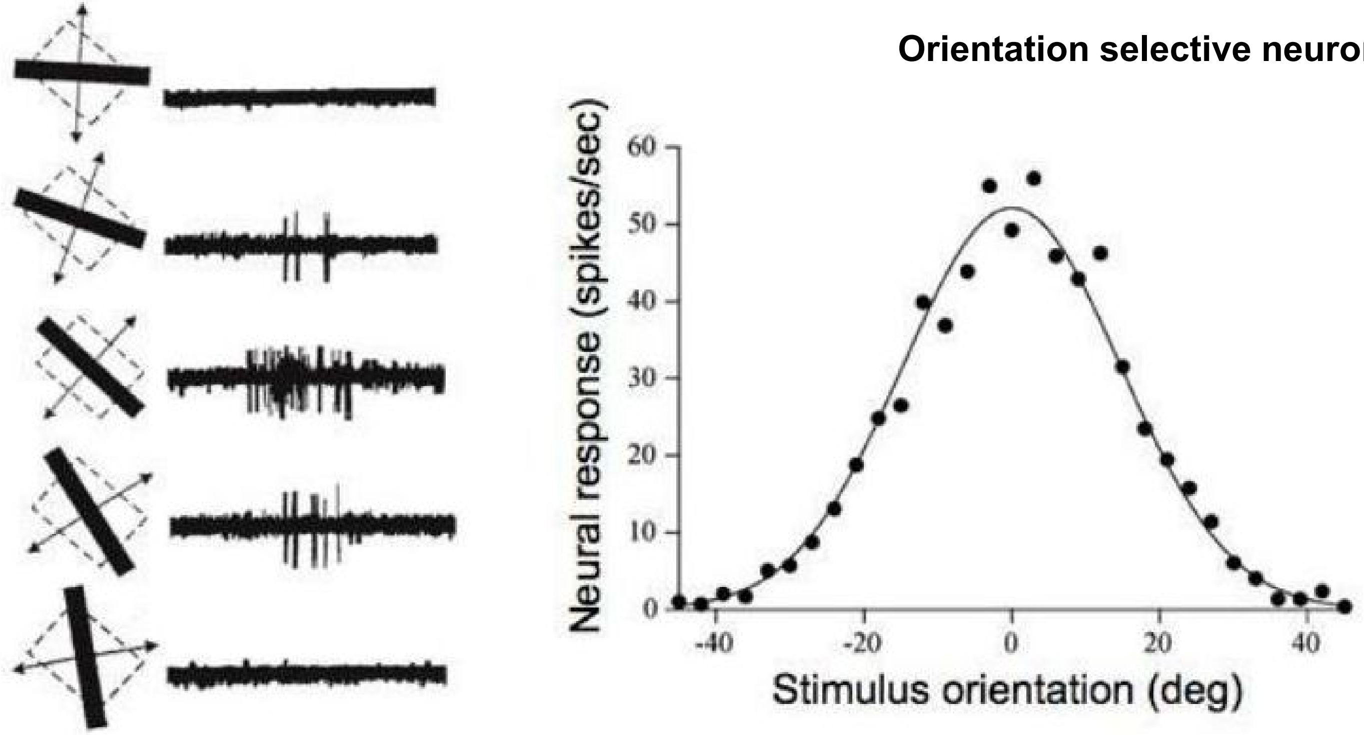
# Biological vision information processing



# Biological vision information processing

**Orientation selective neurons in V1**



# Convolution?

- **Mathematical operation on two functions ($x, h$) to produce a third function ($y$)**
- **Discrete-time convolution**

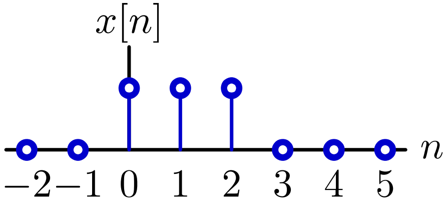
$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m]$$

- **Continuous-time convolution**

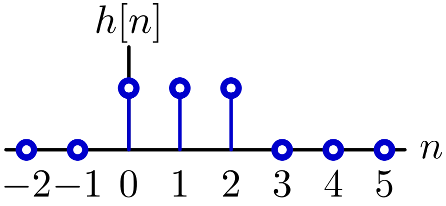
$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$

# Convolution (DT): step-by-step calculation

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

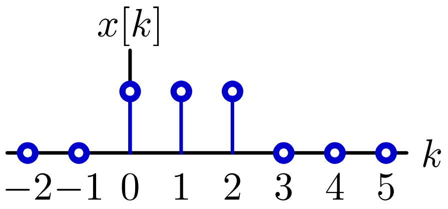



$*$

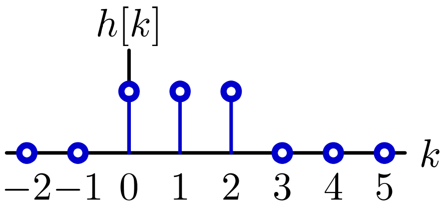


# Convolution (DT): step-by-step calculation

$$y[{\color{red}0}] = \sum_{k=-\infty}^{\infty} x[k]h[{\color{red}0} - k]$$

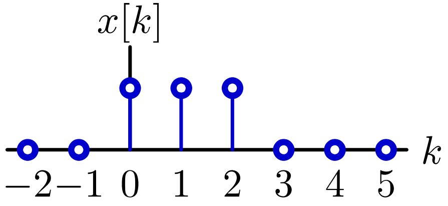



$*$



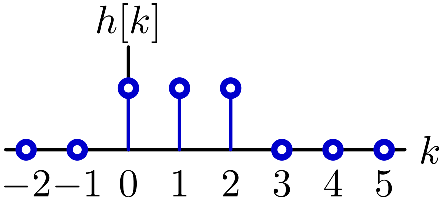
# Convolution (DT): step-by-step calculation

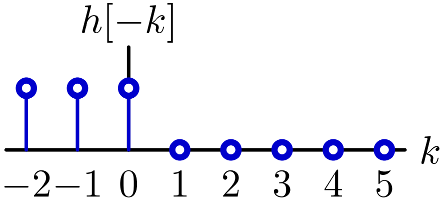
$$y[0] = \sum_{k=-\infty}^{\infty} x[k]h[0 - k]$$




flip

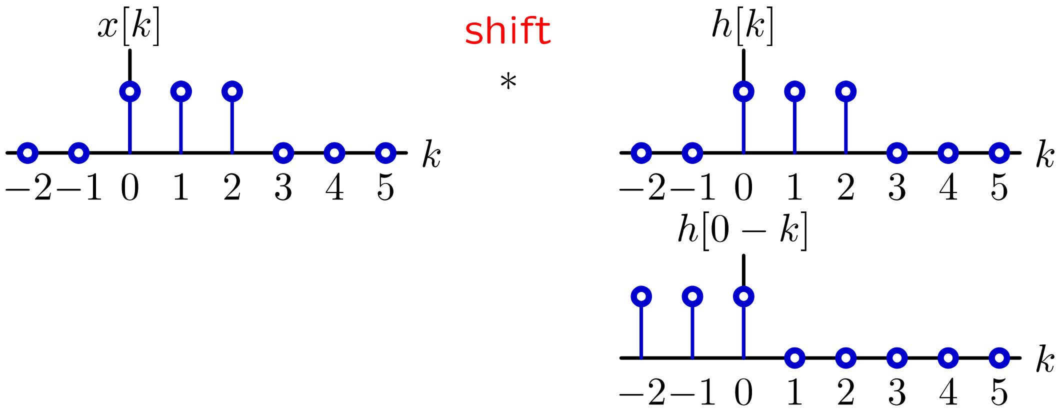
$*$



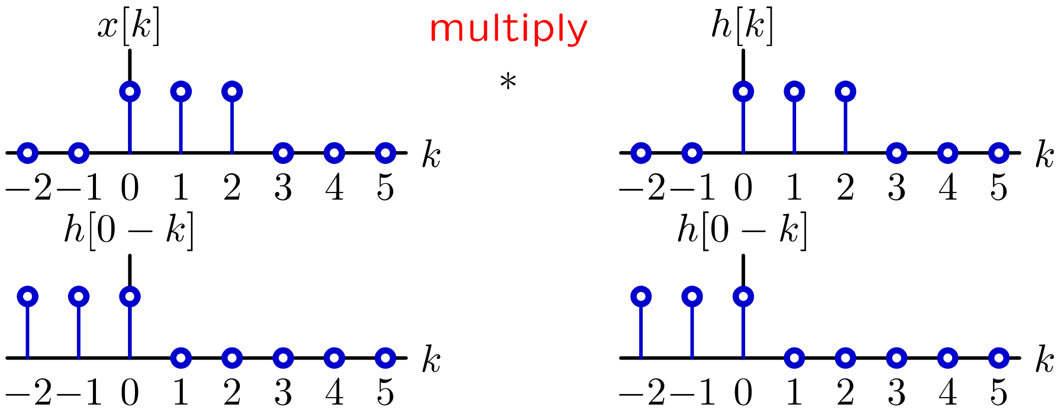
# Convolution (DT): step-by-step calculation

$$y[0] = \sum_{k=-\infty}^{\infty} x[k]h[0-k]$$



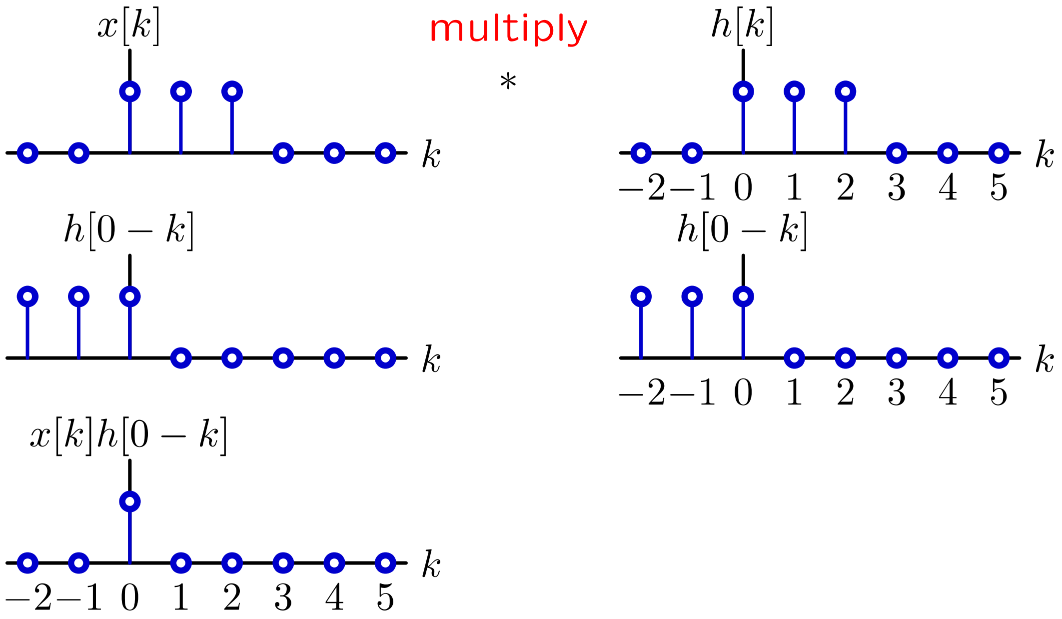
# Convolution (DT): step-by-step calculation

$$y[0] = \sum_{k=-\infty}^{\infty} x[k]h[0-k]$$



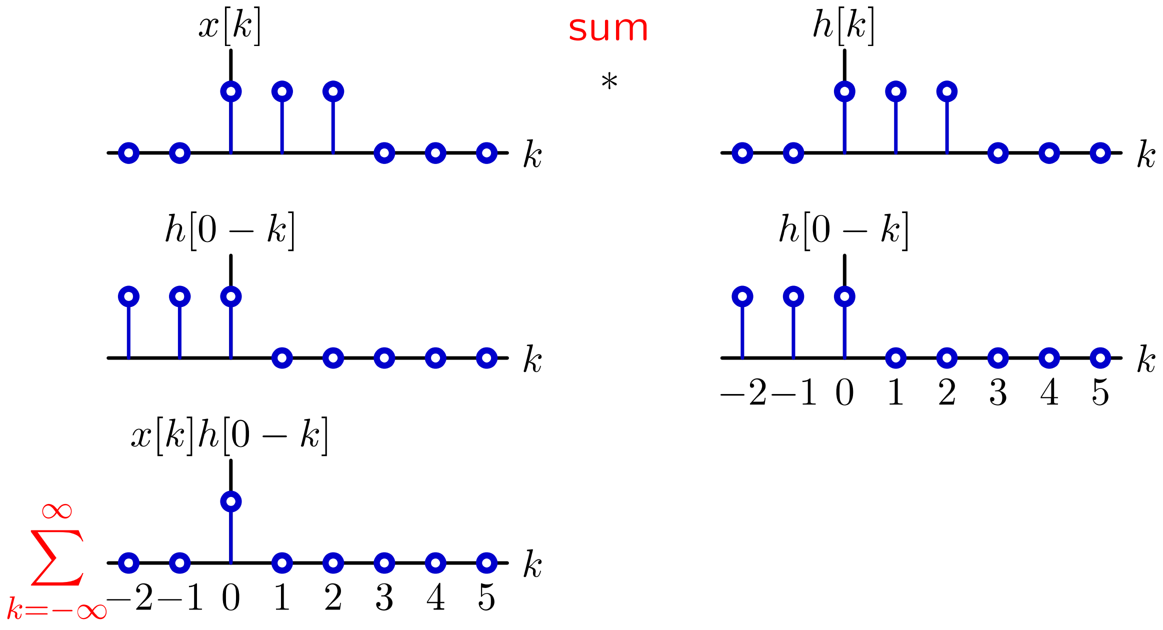
# Convolution (DT): step-by-step calculation

$$y[0] = \sum_{k=-\infty}^{\infty} x[k]h[0-k]$$



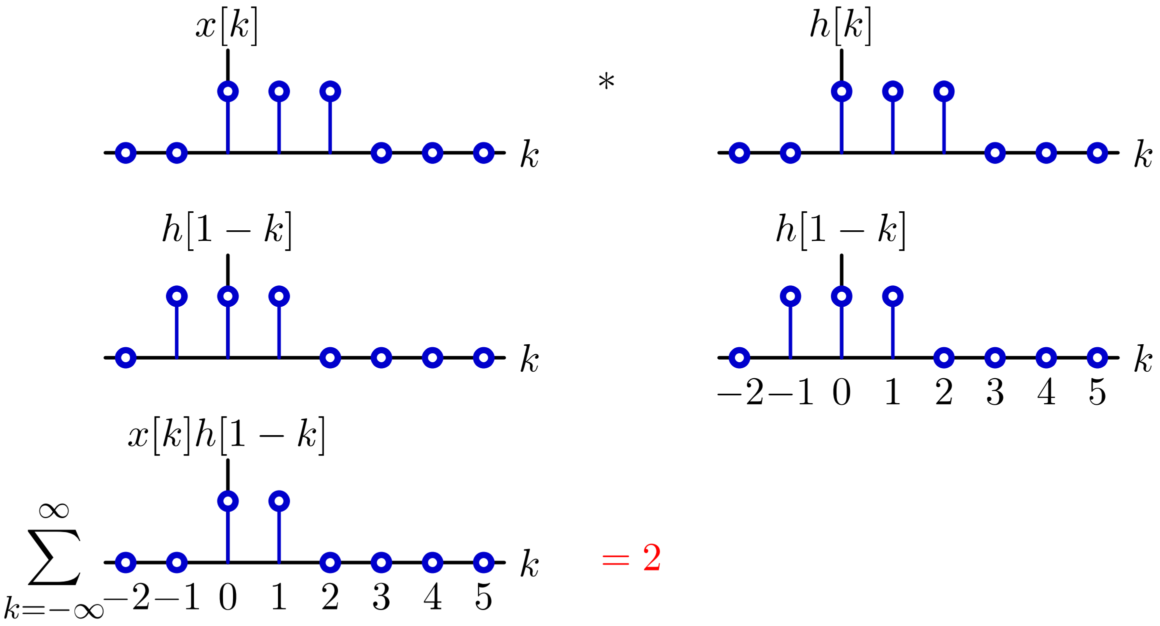
# Convolution (DT): step-by-step calculation

$$y[0] = \sum_{k=-\infty}^{\infty} x[k]h[0-k]$$



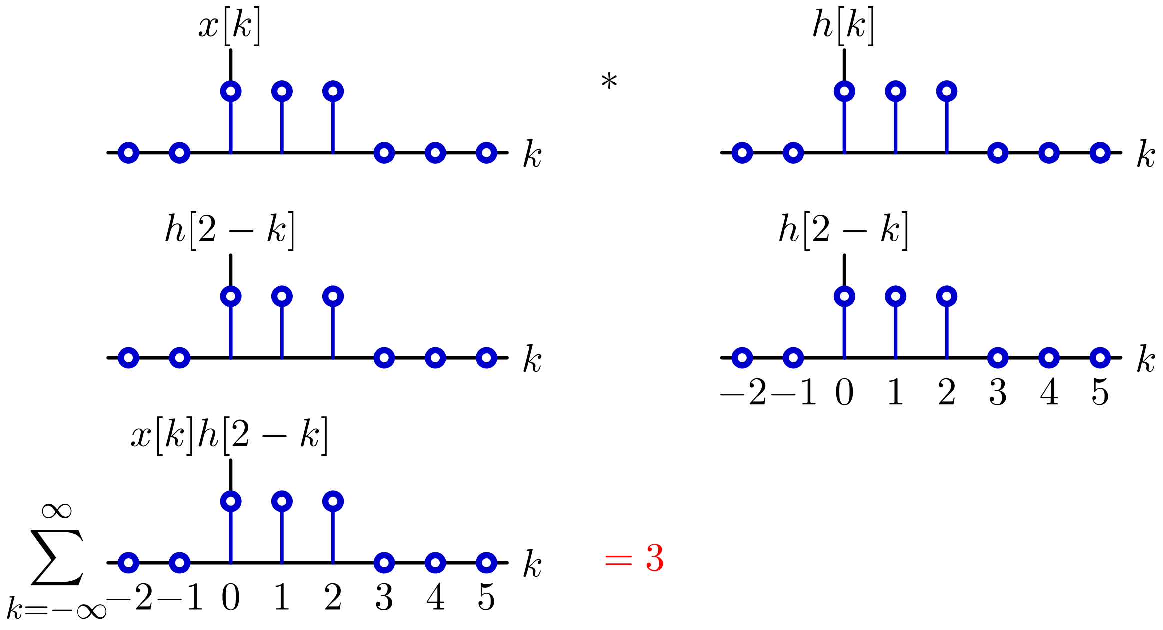
# Convolution (DT): step-by-step calculation

$$y[\color{red}{1}] = \sum_{k=-\infty}^{\infty} x[k]h[1-k]$$



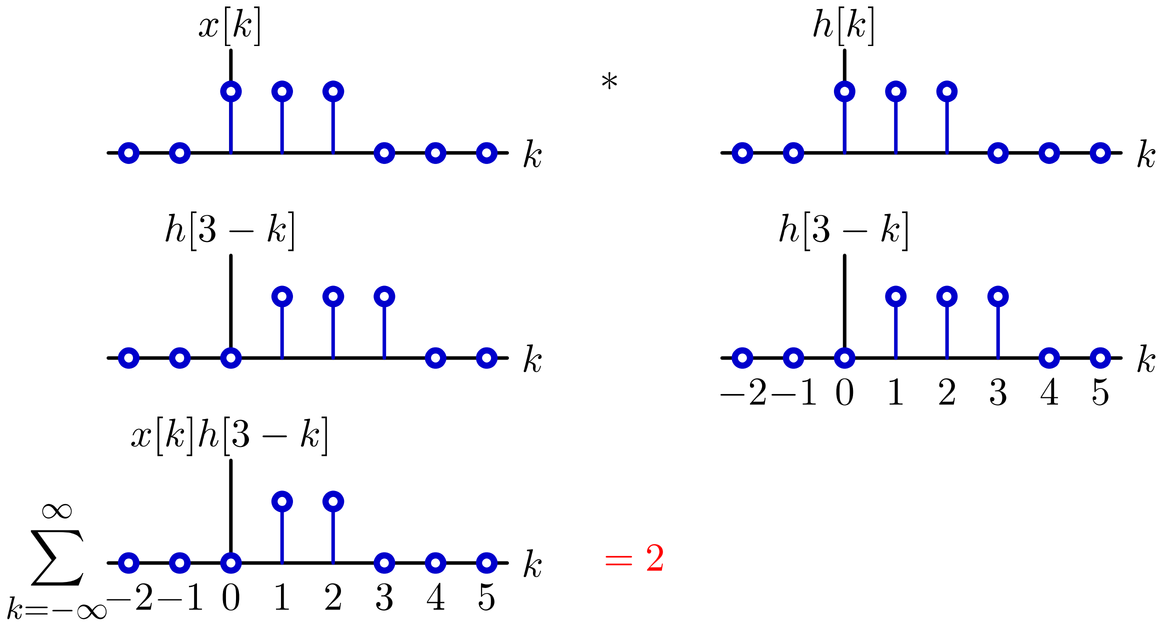
# Convolution (DT): step-by-step calculation

$$y[2] = \sum_{k=-\infty}^{\infty} x[k]h[2-k]$$

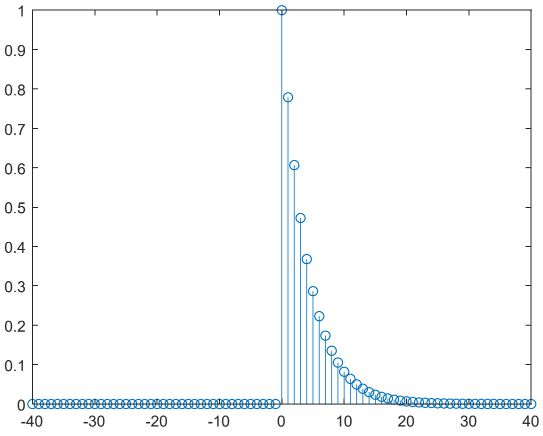


# Convolution (DT): step-by-step calculation

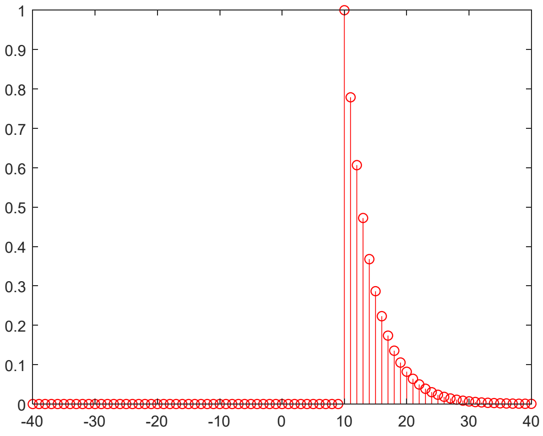
$$y[\color{red}{3}] = \sum_{k=-\infty}^{\infty} x[k]h[3-k]$$



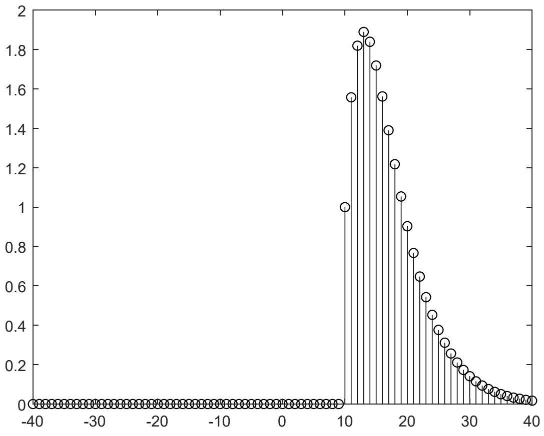
# Convolution (DT): example




$x[n]$

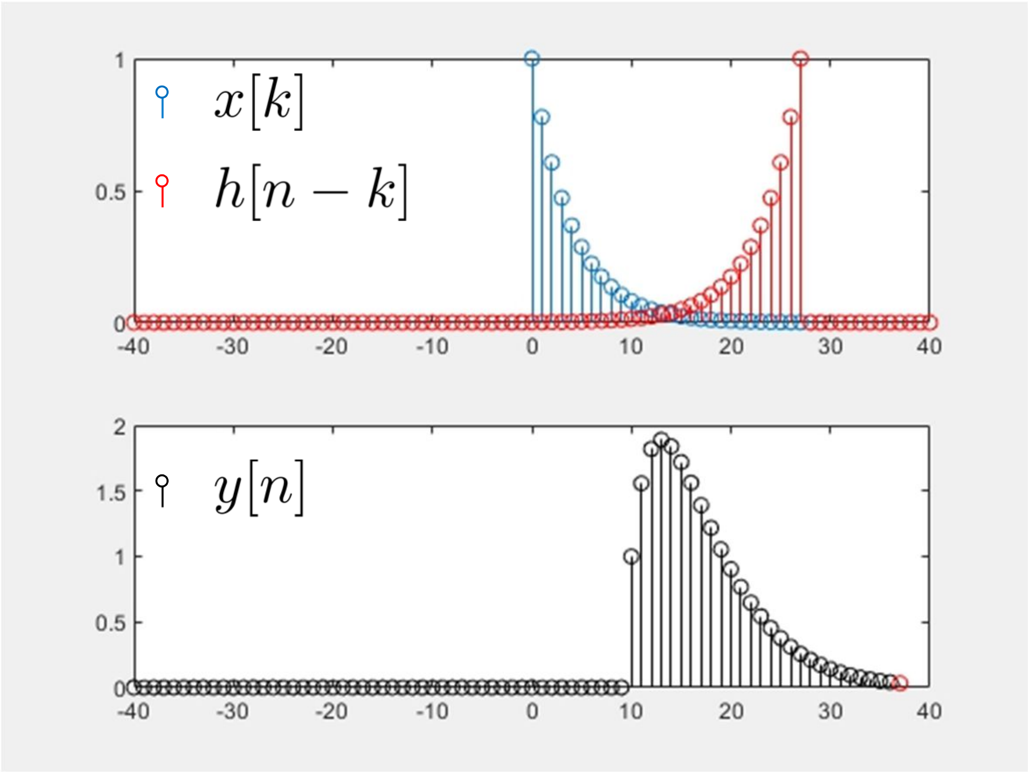



$h[n]$




$y[n]$

# Convolution (DT): example



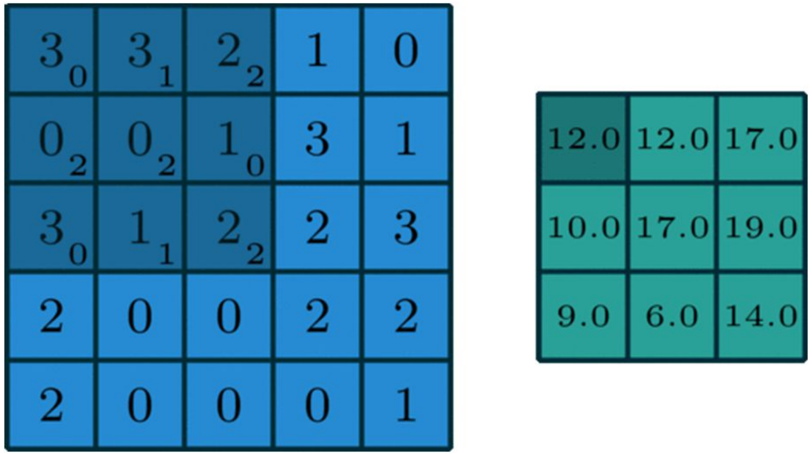

- $h[k]$ → flip → $h[-k]$
- $h[-k]$ → shift by $n$ → $h[n-k]$

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m]$$

# 2-D convolution

- 2-D convolution

$$y[m,n] = x[m,n] * h[m,n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x[k,l]h[m-k,n-l]$$



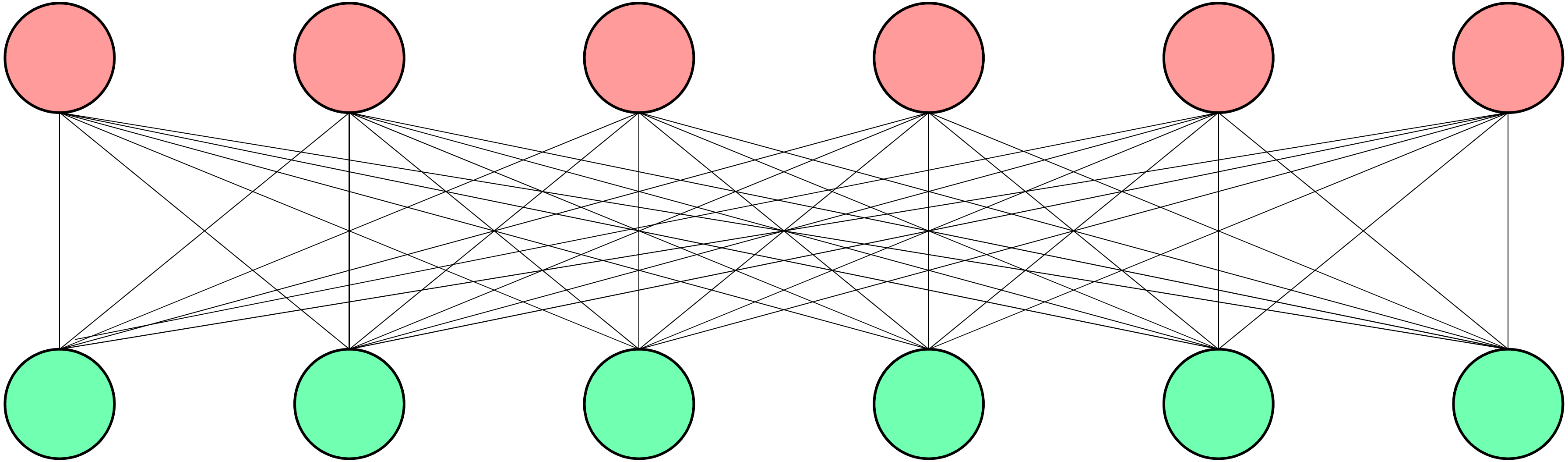
# Convolution for image processing

*

=

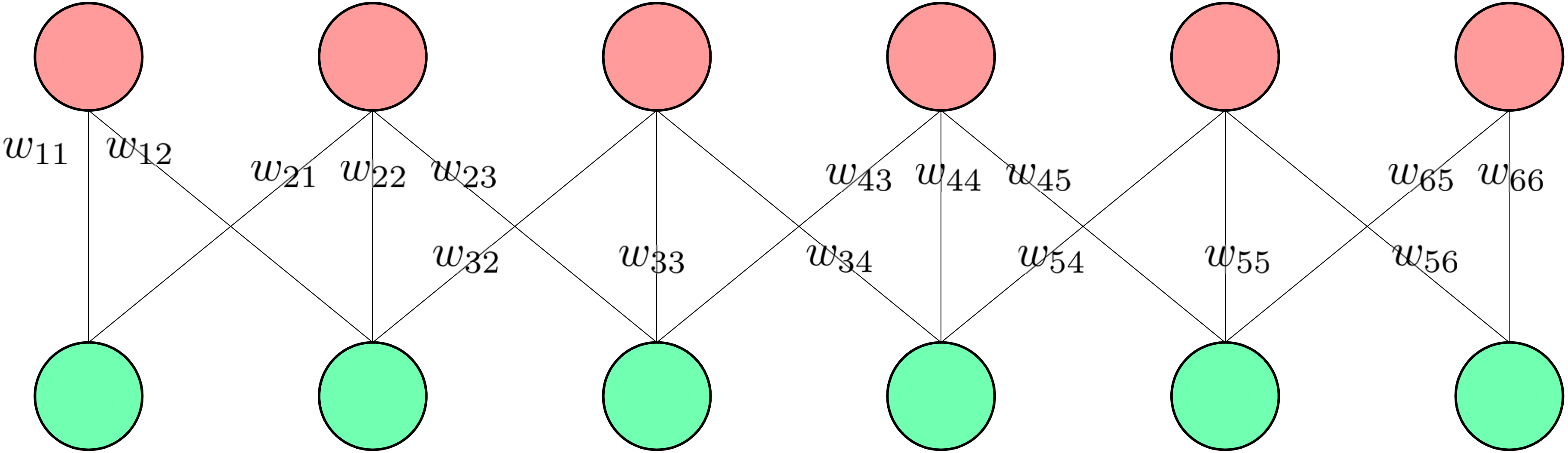
- **Convolution is the heart of image processing**

# Convolutional Neural Network?



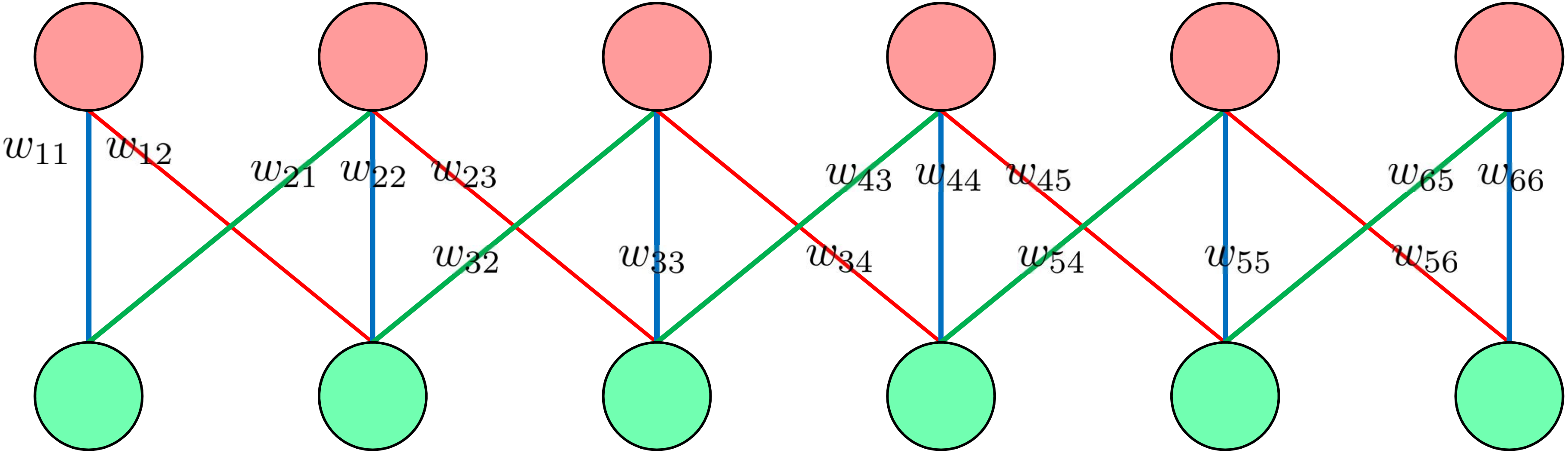
- This is just a layer of a neural network
  - Lots of weights
  - All weights are independent

# Convolutional Neural Network?




- This is (still) just a layer of a neural network
  - Some weights are set to zero (smaller number of weights)

# Convolutional Neural Network?



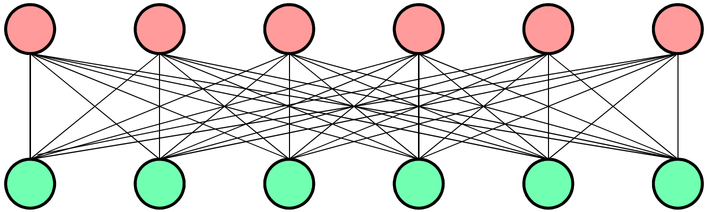

- This is (still) just a layer of a neural network
  - Some weights have shared value (only three weights)

$w_{11} = w_{22} = w_{33} = w_{44} = w_{55} = w_{66}$

$w_{12} = w_{23} = w_{34} = w_{45} = w_{56}$

$w_{21} = w_{32} = w_{43} = w_{54} = w_{65}$

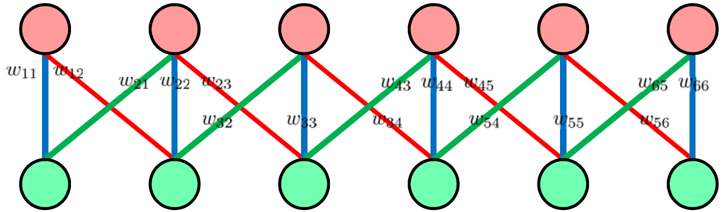
# Convolutional Neural Network?



$Y = h(WX + B)$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = h(\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} & w_{51} & w_{61} \\ w_{12} & w_{22} & w_{32} & w_{42} & w_{52} & w_{62} \\ w_{13} & w_{23} & w_{33} & w_{43} & w_{53} & w_{63} \\ w_{14} & w_{24} & w_{34} & w_{44} & w_{54} & w_{64} \\ w_{15} & w_{25} & w_{35} & w_{45} & w_{55} & w_{65} \\ w_{16} & w_{26} & w_{36} & w_{46} & w_{56} & w_{66} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix})$$

neural network




$Y = h(WX + B)$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = h(\begin{bmatrix} w_b & w_g & 0 & 0 & 0 & 0 \\ w_r & w_b & w_g & 0 & 0 & 0 \\ 0 & w_r & w_b & w_g & 0 & 0 \\ 0 & 0 & w_r & w_b & w_g & 0 \\ 0 & 0 & 0 & w_r & w_b & w_g \\ 0 & 0 & 0 & 0 & w_r & w_b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix})$$

convolutional neural network

# Convolutional Neural Network

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = h(\begin{bmatrix} w_b & w_g & 0 & 0 & 0 & 0 \\ w_r & w_b & w_g & 0 & 0 & 0 \\ 0 & w_r & w_b & w_g & 0 & 0 \\ 0 & 0 & w_r & w_b & w_g & 0 \\ 0 & 0 & 0 & w_r & w_b & w_g \\ 0 & 0 & 0 & 0 & w_r & w_b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix})$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = h(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} * \begin{bmatrix} w_r \\ w_b \\ w_g \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix})$$

$$y_n = h(x_n * w_n + b_n) = h(\sum x_m w_{n-m} + b_n)$$

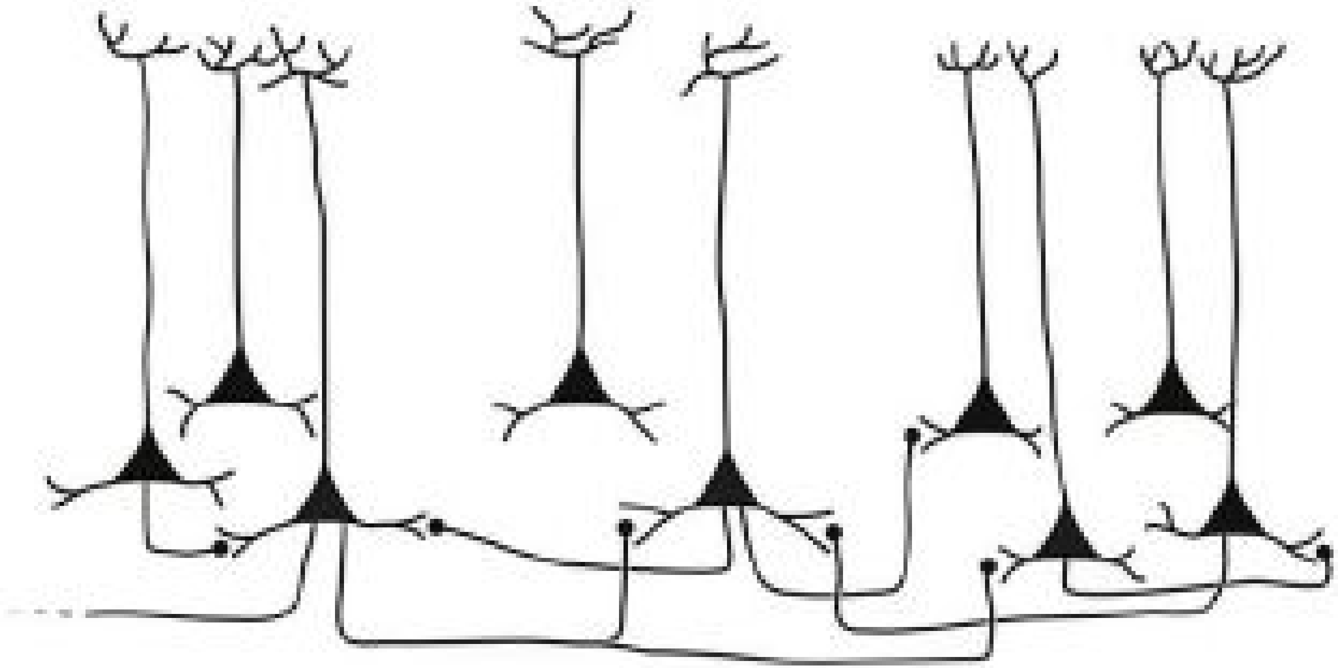
- Matrix multiplication is replace by convolution
- Convolutional neural network (ConvNet or CNN) can be thought of as a special case of neural network
  - Lots of weights are zero (only “local” operations allowed)
  - Lots of weights are shared

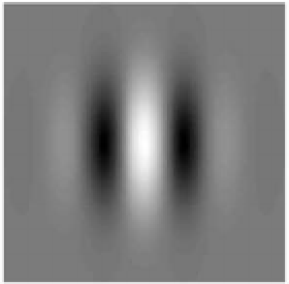
# Why ConvNet?

- Again, ConvNet is just a special case of neural network
- **Local operation**
  - Optimized for processing spatial information (i.e., image)
- **Parameter sharing**
  - Smaller number of weights (or higher capacity with same number of parameters)
- Local operation + parameter sharing
  - Optimized for processing translationally-invariant structures of the image

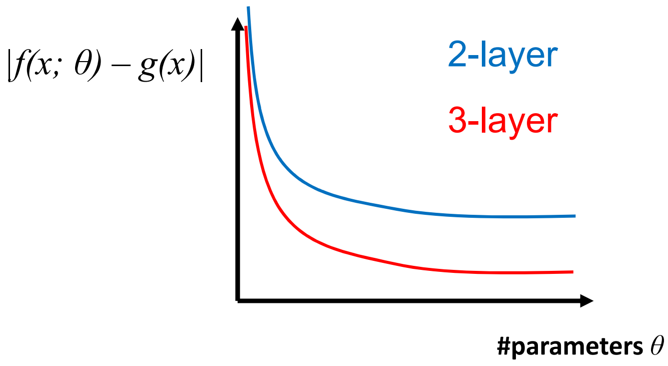
- In a nutshell, ConvNet is a specialized neural network structure suited for image processing

# Local connection & parameter sharing?

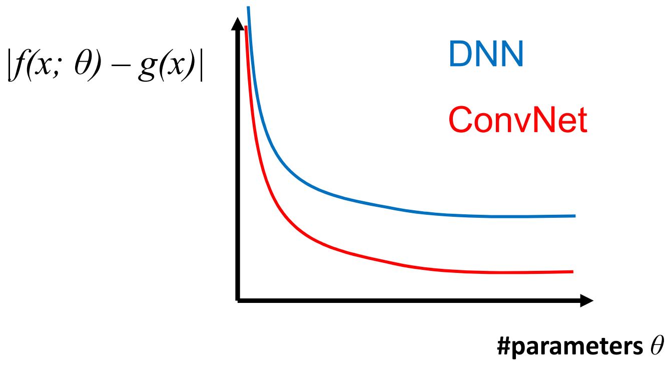




# Revisit: Neural Network Design




- We want to use different forms of parameterized function $f(x;\ \theta)$ depending on our target function $g(x)$ (which we do not know)
- Some families of parameterized functions $f(x;\ \theta)$ can approximate $g(x)$ with a smaller error with smaller number of parameters than others
- …and this of course depends on $g(x)$ (task-dependent)




- For most $g(x)$ from image processing tasks, it is like the left plot

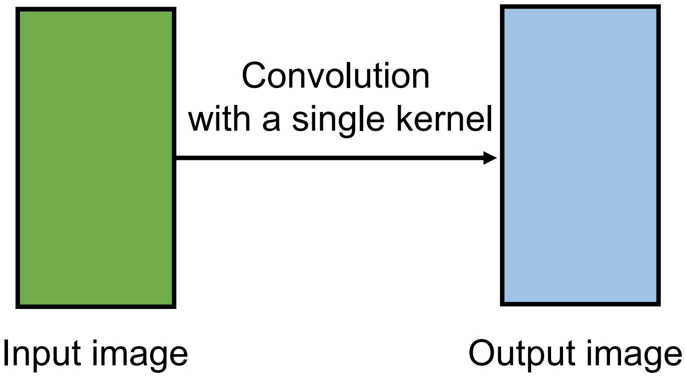
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = h\left( \begin{bmatrix} w_b & w_g & 0 & 0 & 0 & 0 \\ w_r & w_b & w_g & 0 & 0 & 0 \\ 0 & w_r & w_b & w_g & 0 & 0 \\ 0 & 0 & w_r & w_b & w_g & 0 \\ 0 & 0 & 0 & w_r & w_b & w_g \\ 0 & 0 & 0 & 0 & w_r & w_b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} \right)$$

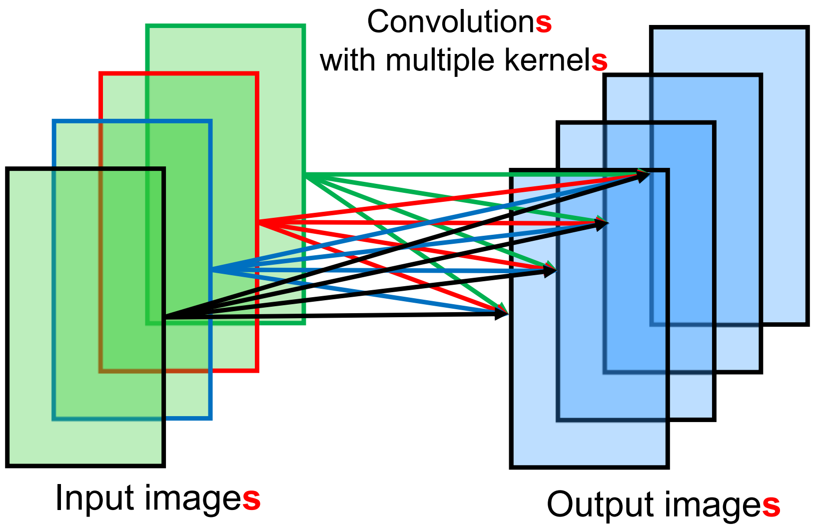
- We are essentially trying to approximate a function. If we can reduce the degree of freedom of our parameterized function by adding constraints without compromising its capability to approximate the "target function," then it would be desired!

# Convolutional Neural Network with Channels

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = h(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} * \vec{w} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix})$$

$$\begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \\ y_{16} \end{bmatrix} = h(\begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix} * \overrightarrow{w_{11}} + \begin{bmatrix} x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{25} \\ x_{26} \end{bmatrix} * \overrightarrow{w_{12}} + \cdots \begin{bmatrix} x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \\ x_{45} \\ x_{46} \end{bmatrix} * \overrightarrow{w_{14}} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{14} \\ b_{15} \\ b_{16} \end{bmatrix}) \quad \cdots \quad \begin{bmatrix} y_{41} \\ y_{42} \\ y_{43} \\ y_{44} \\ y_{45} \\ y_{46} \end{bmatrix} = h(\begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{bmatrix} * \overrightarrow{w_{41}} + \begin{bmatrix} x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{25} \\ x_{26} \end{bmatrix} * \overrightarrow{w_{42}} + \cdots \begin{bmatrix} x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \\ x_{45} \\ x_{46} \end{bmatrix} * \overrightarrow{w_{44}} + \begin{bmatrix} b_{41} \\ b_{42} \\ b_{43} \\ b_{44} \\ b_{45} \\ b_{46} \end{bmatrix})$$



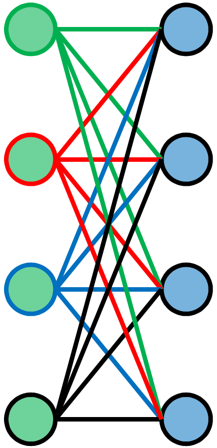




**(single layer of) ConvNet as a special case of DNN**

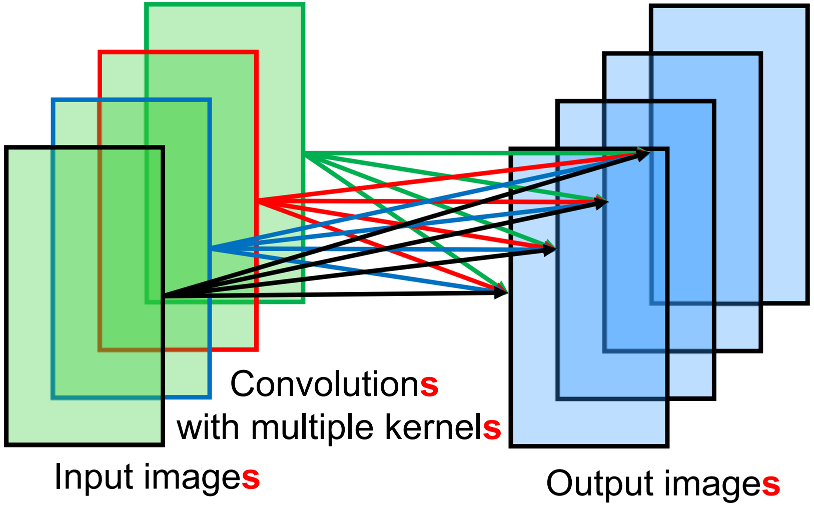
**(single layer of) ConvNet with channels**

# Convolutional Neural Network with Channels

- Earlier, we said **ConvNet can be thought of as a special case of DNN**
- We can also interpret a **ConvNet as a "generalized" form of DNN**
- Single scalar value in a node → an array
- Single scalar weight for multiplication of an edge → an array for convolution



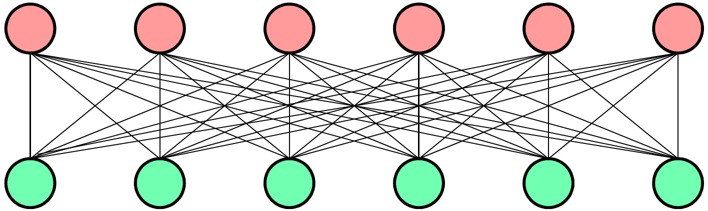
(single layer of) Deep neural network

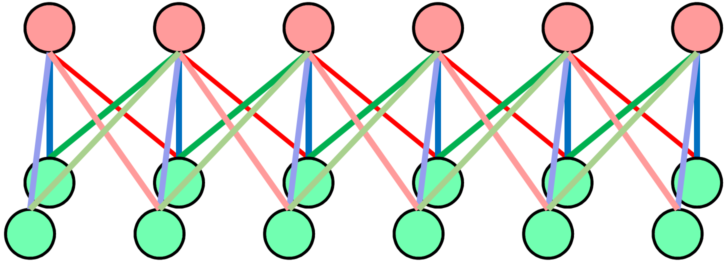



(single layer of) ConvNet with channels

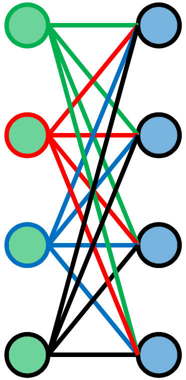
# Convolutional Neural Network with Channels

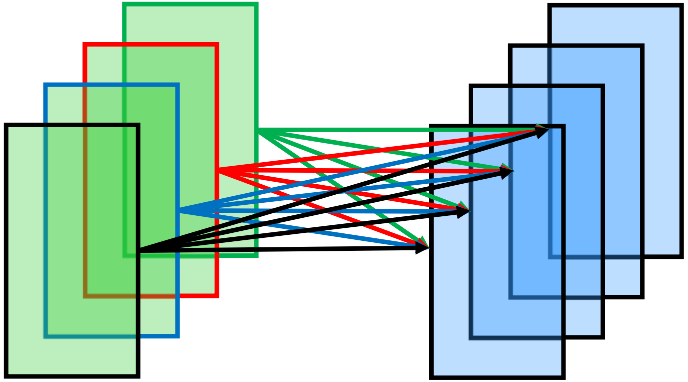
- **ConvNet as a special form of DNN**





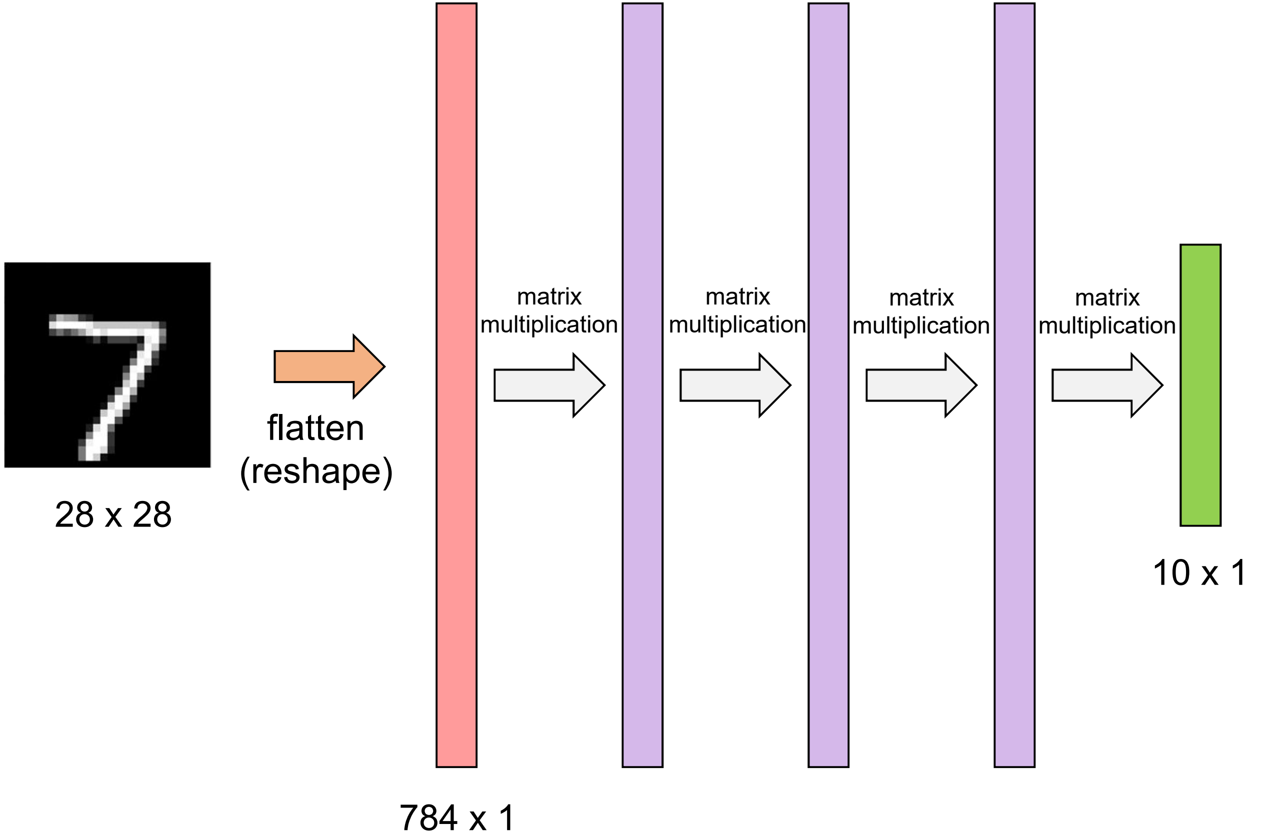
- **ConvNet as a generalized form of DNN**



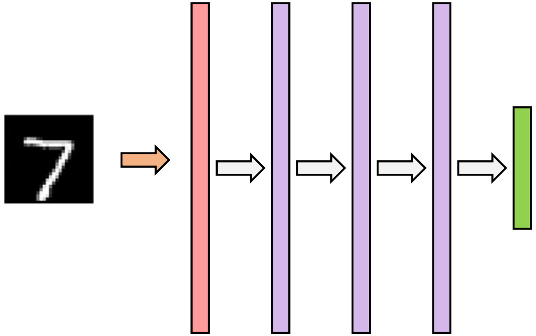


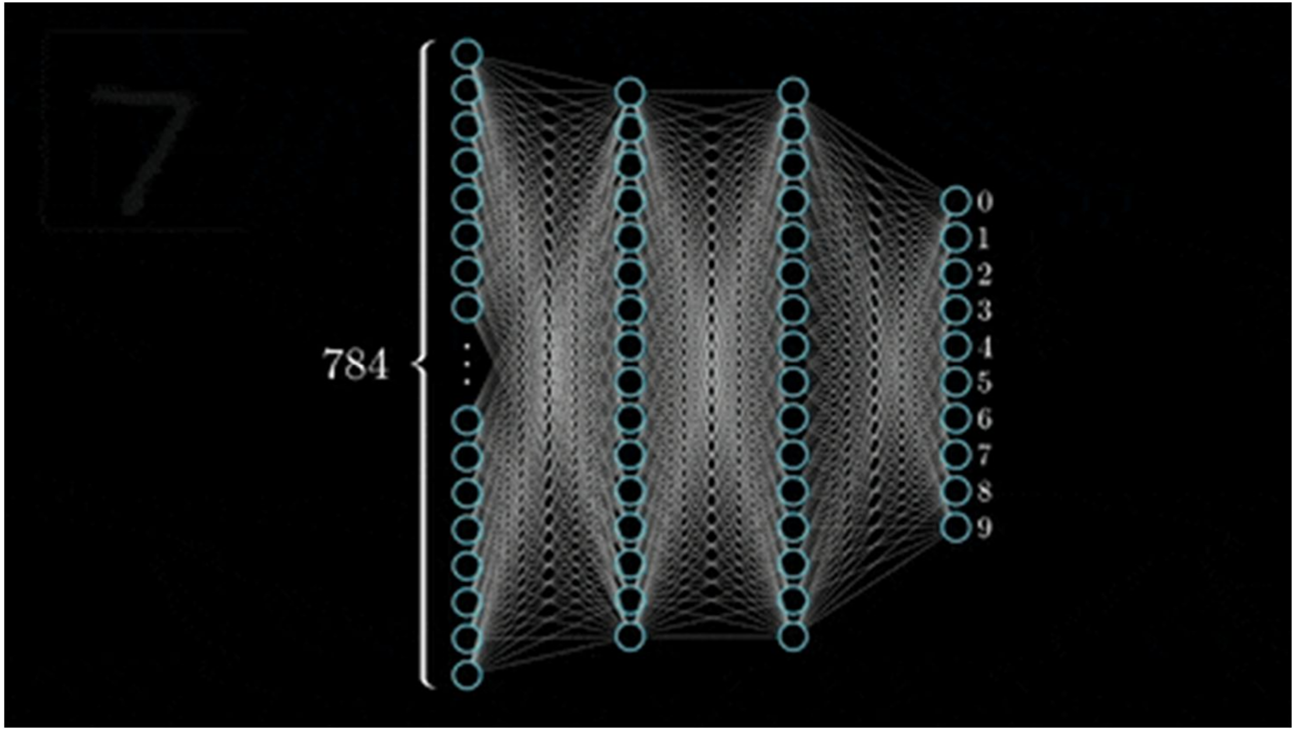
**'A is a special form of B' and 'A is a generalized form of B' at the same time?**

# MNIST: NN vs. ConvNet

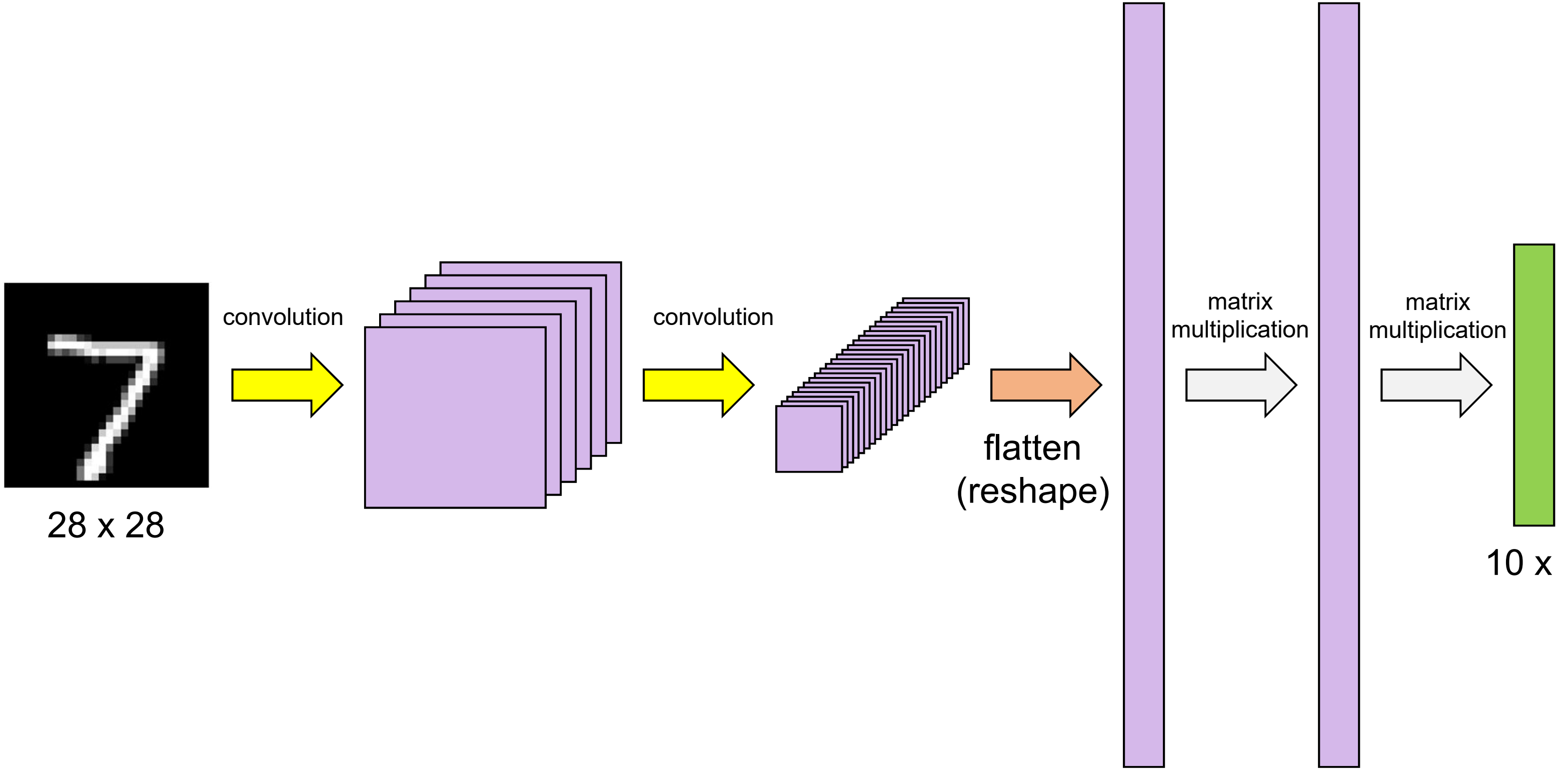


# MNIST: NN vs. ConvNet

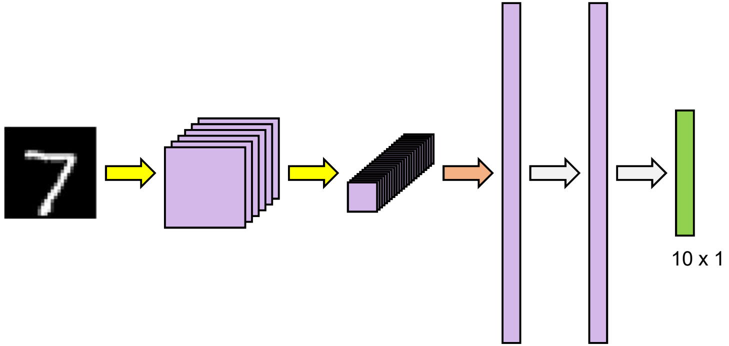


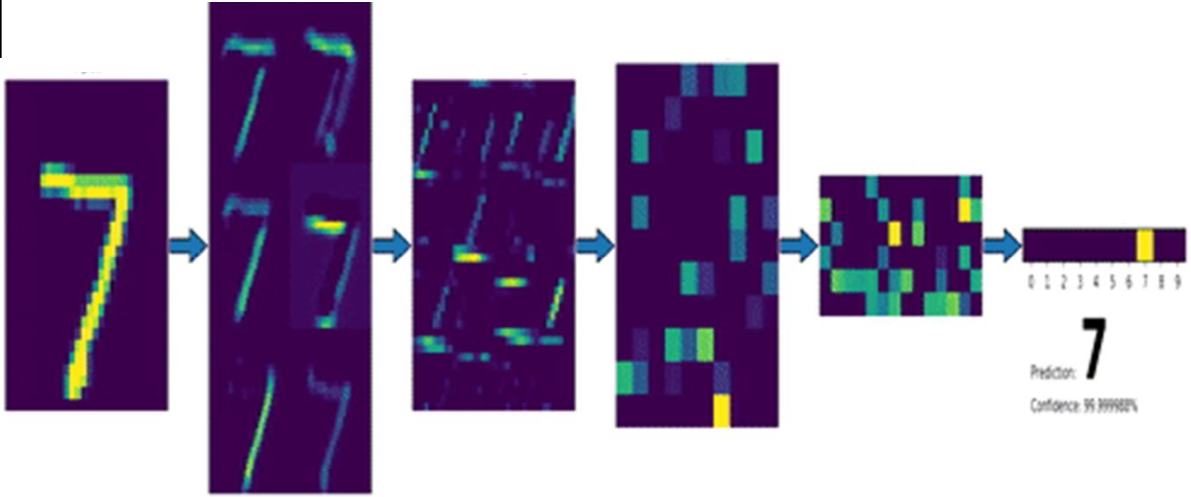


# MNIST: NN vs. ConvNet

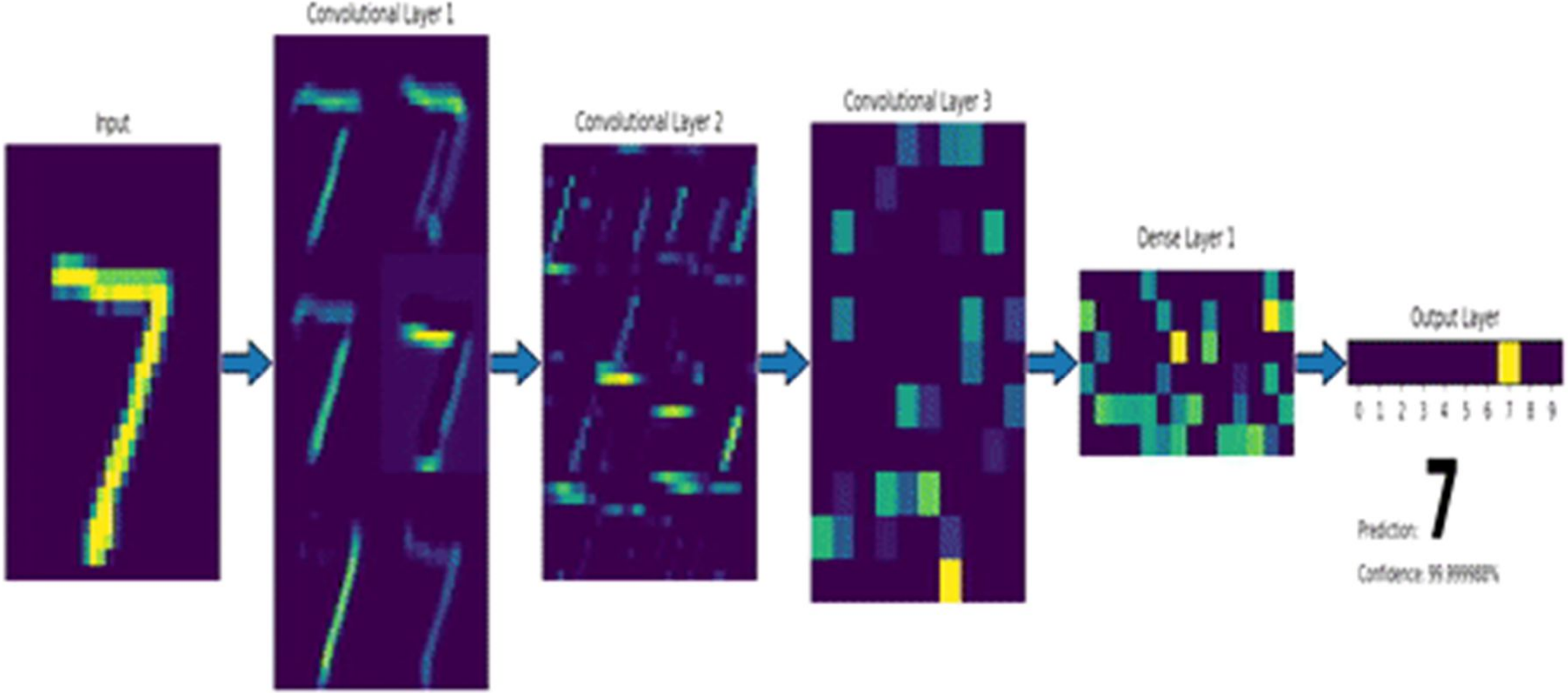


# MNIST: NN vs. ConvNet

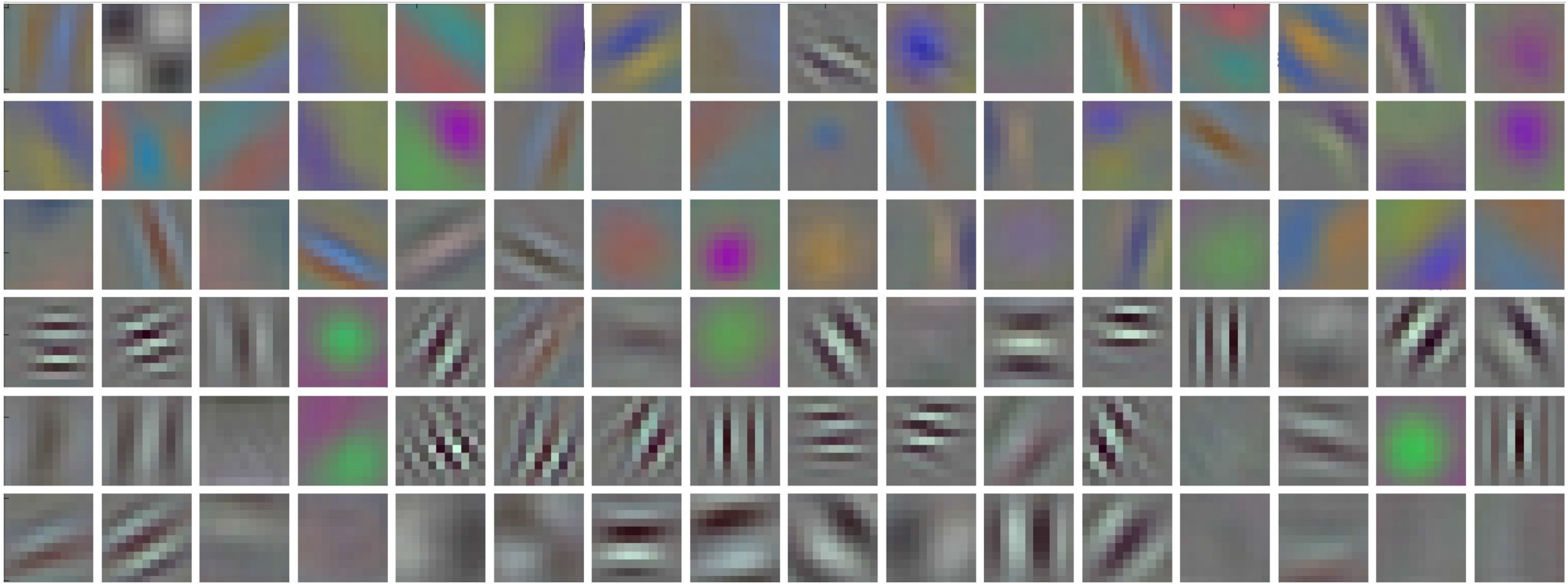


# Visualizing the signals “inside” the network

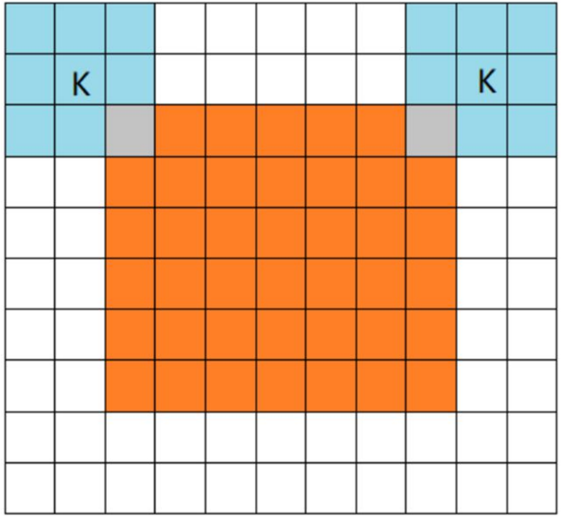


# Visualizing the parameters “inside” the network



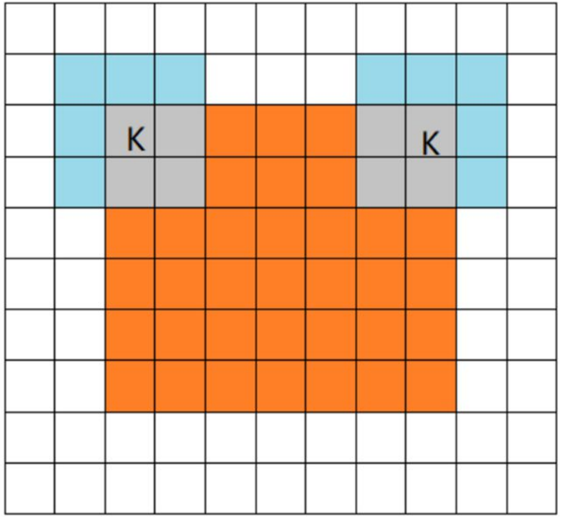
- Weights of the first layer of a ConvNet (after training)

# Buliding blocks: Convolution



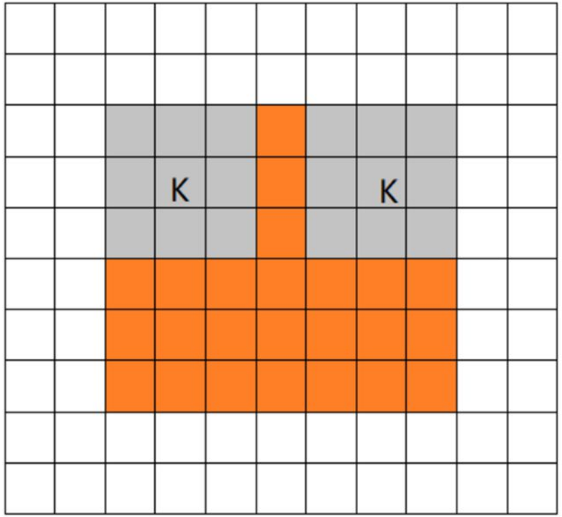

**Full convolution**

Input size ≤ output size




**Same convolution**

Input size = output size

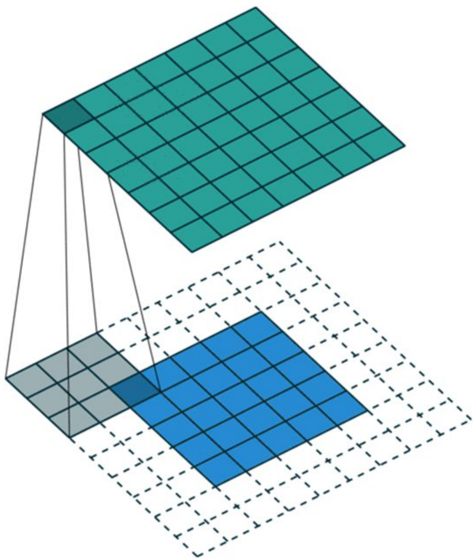



**Valid convolution**

Input size ≥ output size

Different “amount” of zero-padding before the operation

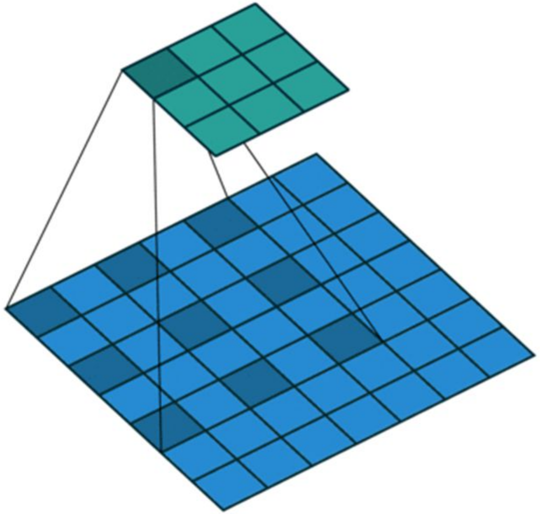
# Buliding blocks: Convolution



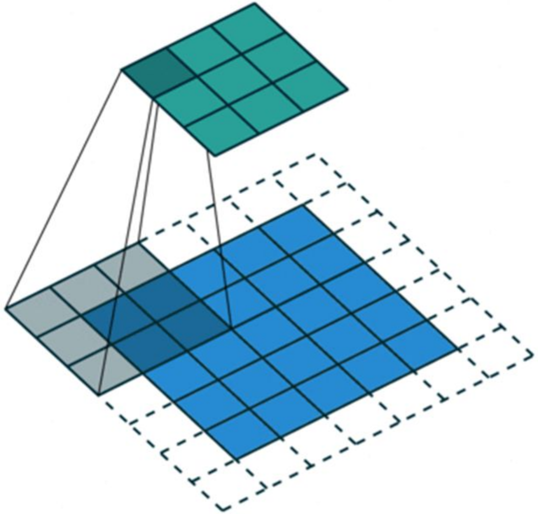
**Full convolution** **Same convolution** **Valid convolution**

Different “amount” of zero-padding before the operation

# Buliding blocks: Convolution

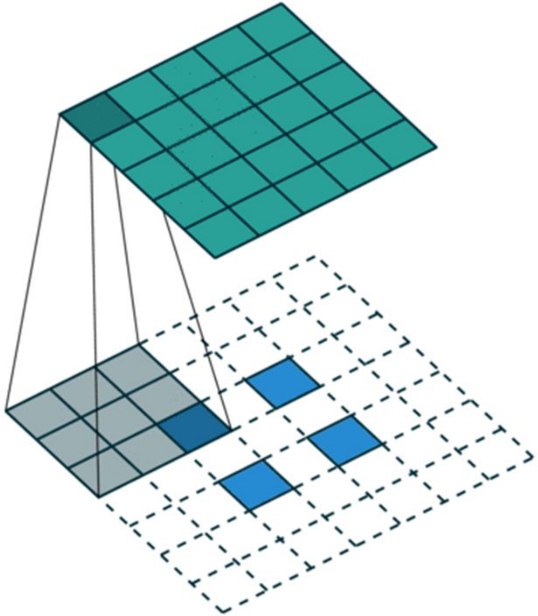


dilated convolution



Convolution with stride-2

More variations

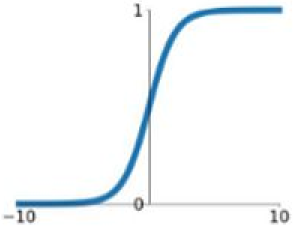


Transposed convolution
(some people call it
deconvolution or up-convolution)

# Buliding blocks: Activation

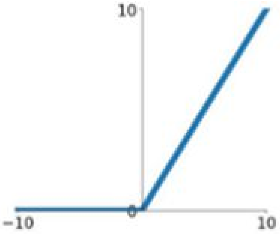
**Sigmoid**

$\sigma(x) = \frac{1}{1+e^{-x}}$



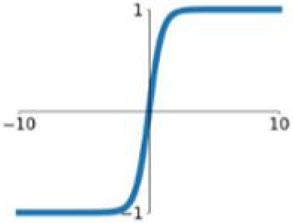

**ReLU**

$\max(0, x)$



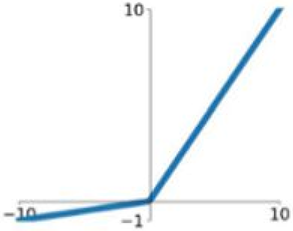

**tanh**

$\tanh(x)$



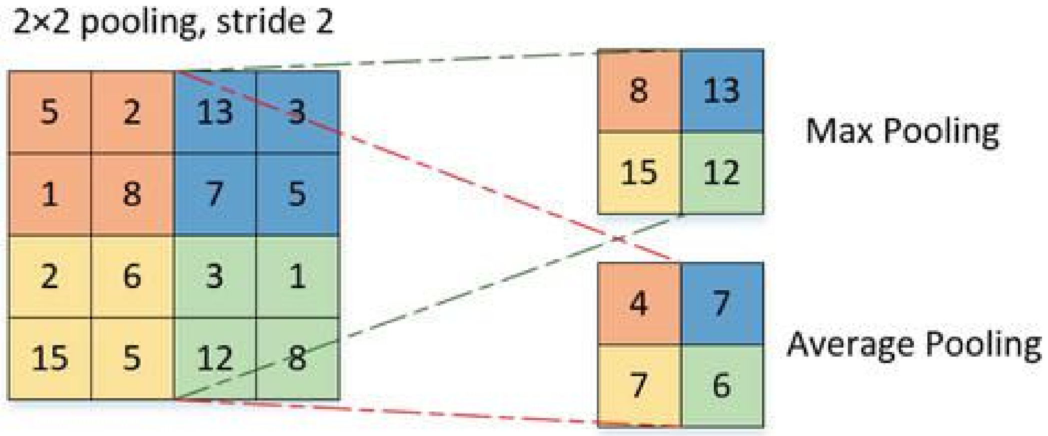

**Leaky ReLU**

$\max(\ \alpha x\ , x)$



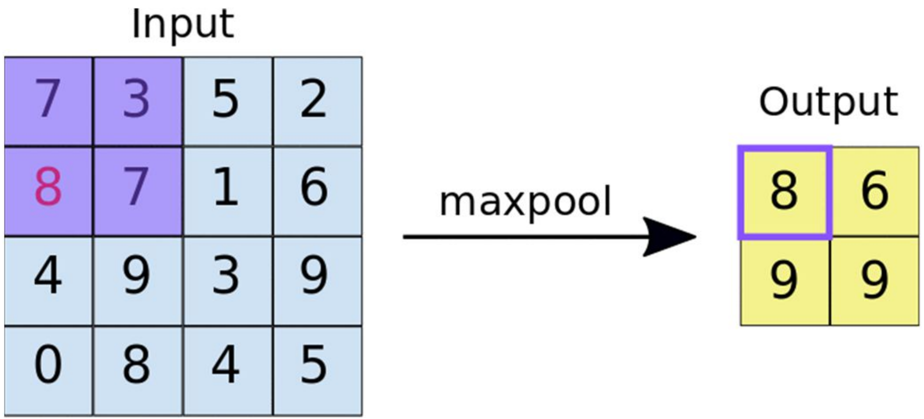

- ReLU is the most popular choice these days

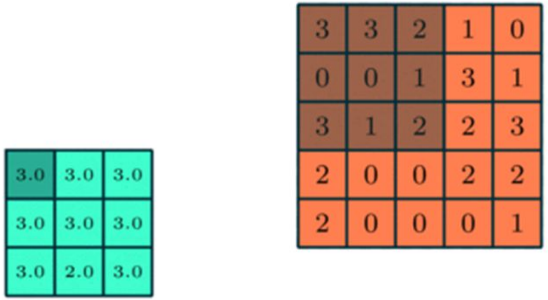
# Buliding blocks: pooling




- A **Pooling layer** allows us to efficiently (no parameter, simple computation) decrease the input size
- **Max pooling** is the most popular choice, but it depends on tasks and applications

# Buliding blocks: pooling



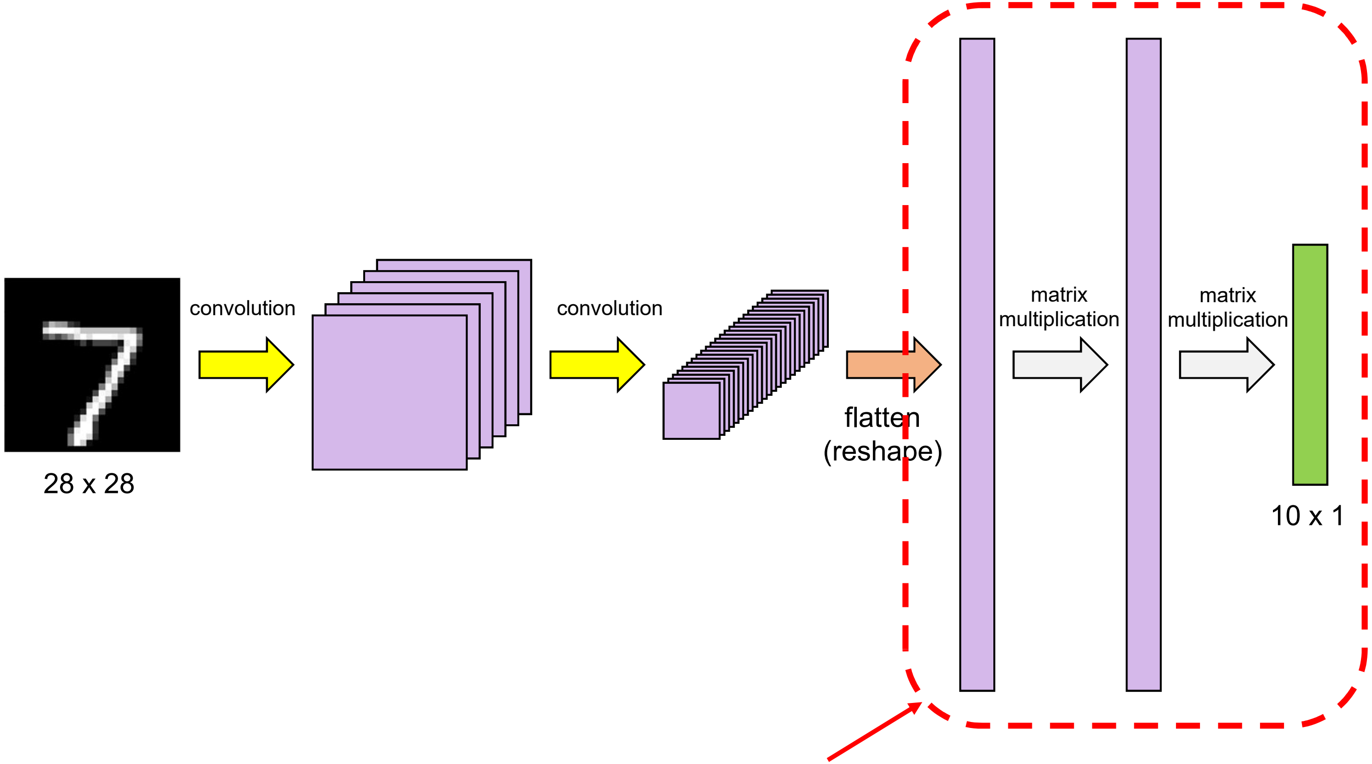




**2x2 max pooling with stride 2**

**3x3 max pooling with stride 1**

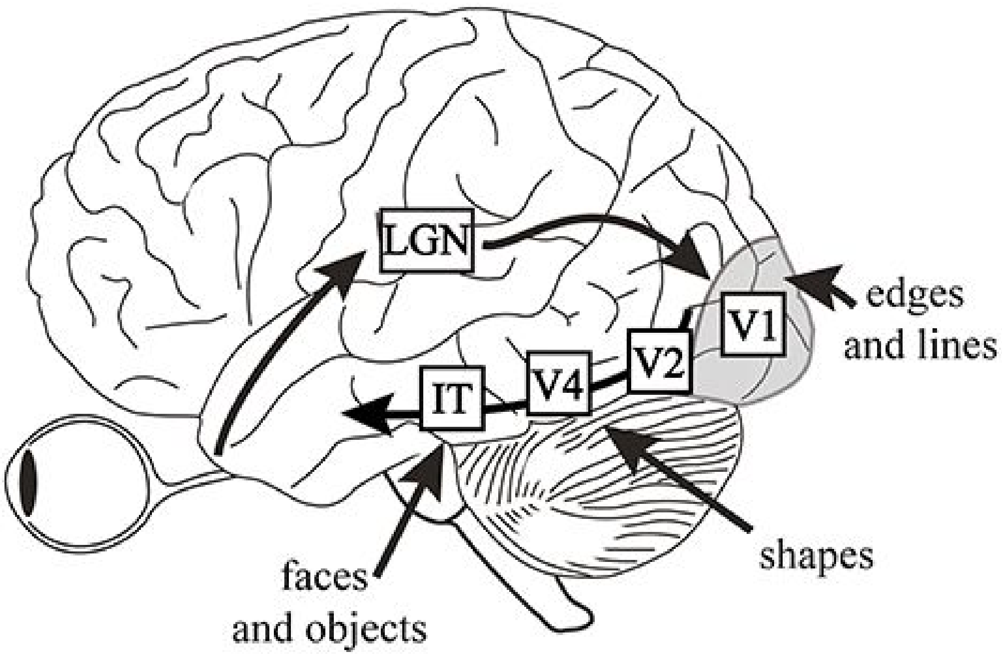
Pooling window size does not have to be the same as stride
(just like the convolution kernel size does not have to be the same as the stride)

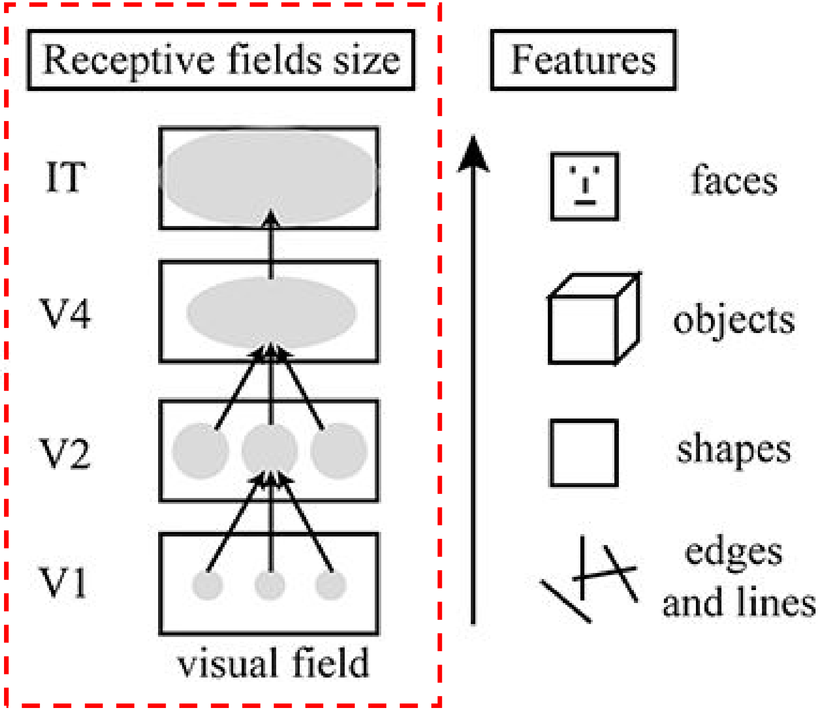
# Buliding blocks: fully-connected layer




Fully-connected layer is just a different name for
these simple neural network layers (to distinguish from convolution layers)

# Why pooling?


LGN
V1
V2
V4
IT
edges
and lines
shapes
faces
and objects

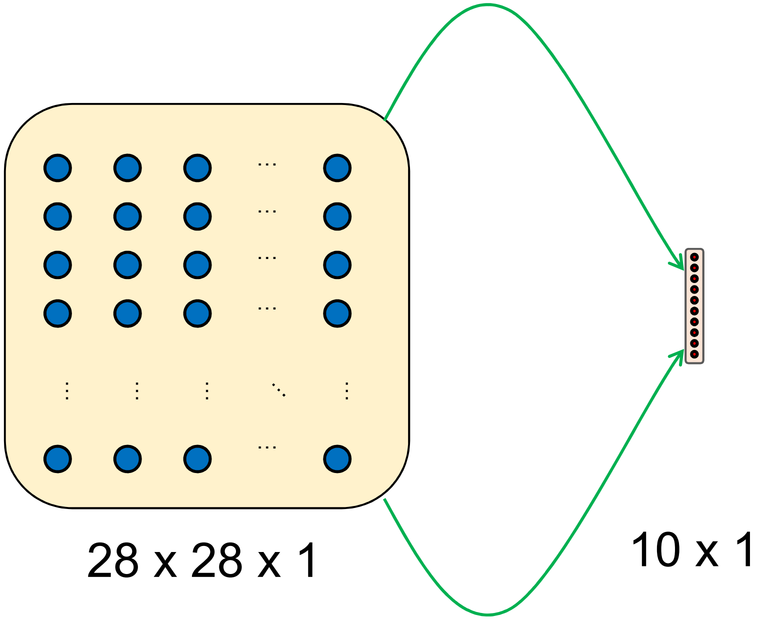

Receptive fields size
IT
V4
V2
V1
visual field
Features
faces
objects
shapes
edges
and lines

# Why pooling?

- **Parameter-free**
  - Oftentimes, we want to decrease the size of the input
  - Pooling is a deterministic without any trainable parameters → can effectively decrease the size without having to worry about overfitting

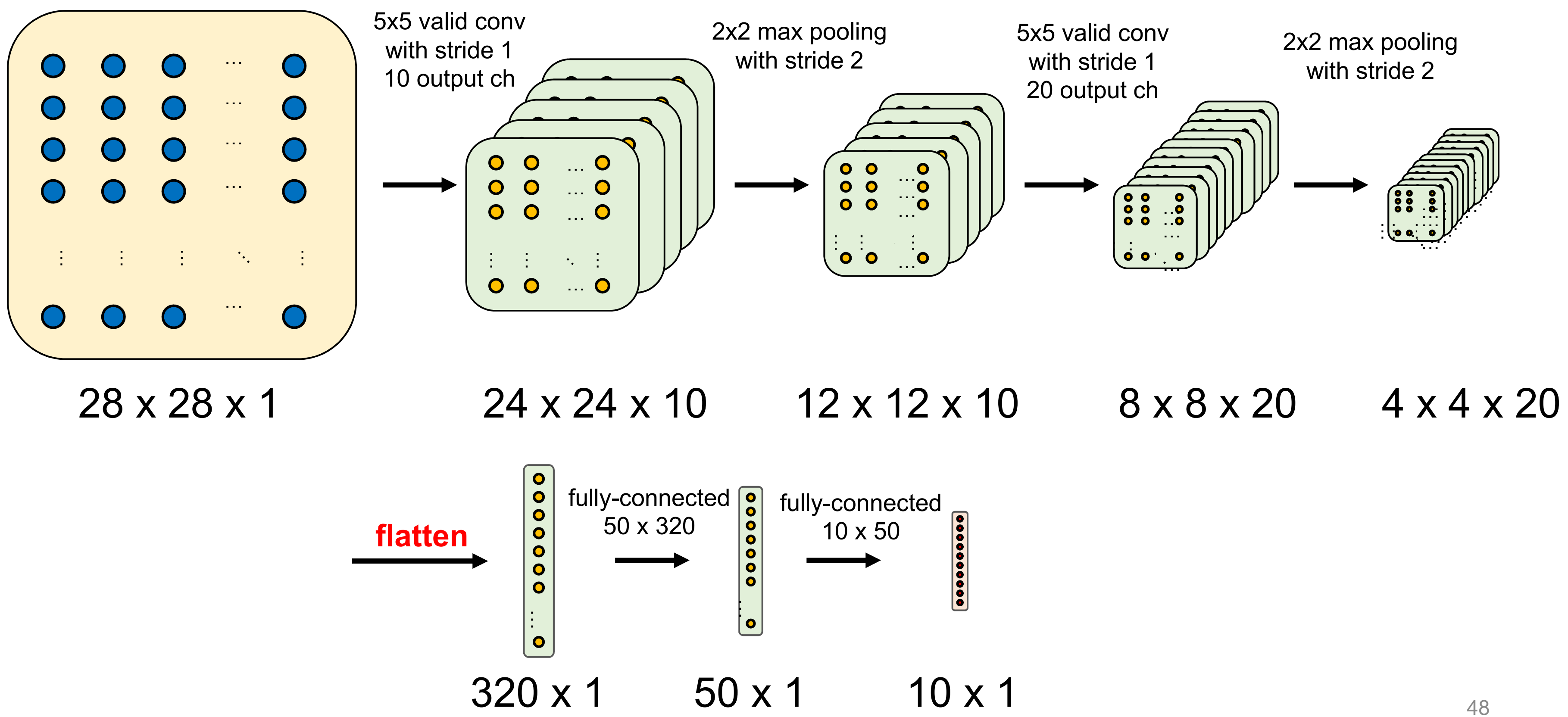
- **Increase of receptive field**
  - Imagine that we only have convolution layers with a fixed kernel size
  - Then, the pixels within only certain (and short) distance can interact inside the network
    - This is not good for “high-level” feature extraction

# ConvNet design: Ok, let's put together

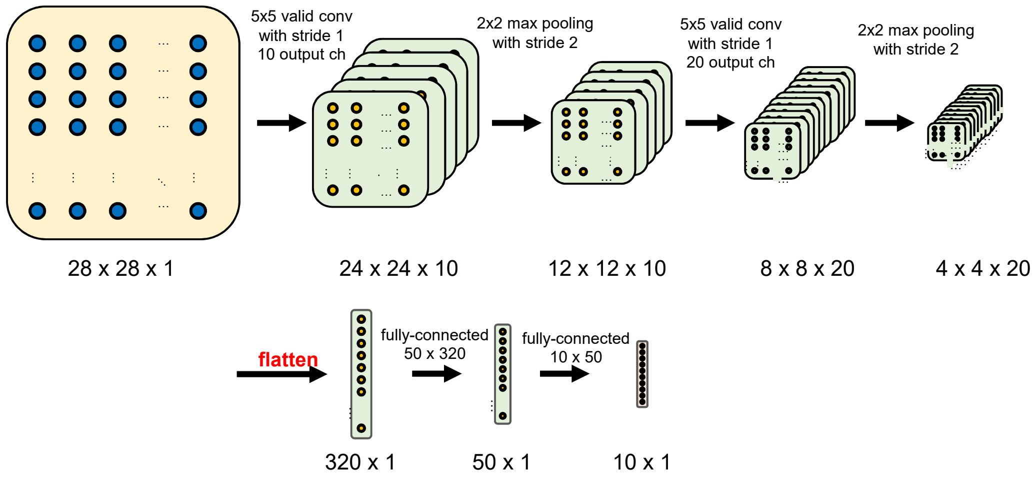



For MNIST classification,
We need to design a convolutional neural network
that takes 28x28x1 array as the input
and generates 10x1 array as the output
(**design constraint**)

# ConvNet design: Ok, let's put together



# ConvNet design: PyTorch Implementation




```
class MNIST_classifier(nn.Module):
    def __init__(self):
        super(CNN_classifier, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, 5, 1)
        self.conv2 = nn.Conv2d(10, 20, 5, 1)
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*20)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

# Summary

- Convolution operation
- ConvNet is very similar to biological neural network for vision processing
- ConvNet can be thought of as either a special case NN or a generalized NN
- Building blocks of ConvNet
  - Convolution (many types)
  - Activation
  - Pooling
  - Fully connected layer

# References

- Website
  - CS231n course website: https://cs231n.github.io/