Medical Diagnostics

Tutorial 7: CNN

Spring 2025

Faculty of Biotechnology and Food Engineering
Technion Israel Institute of Technology

TA: Mattan Hoory

Table of contents

1 Intro

משאל המרצה והמתרגל

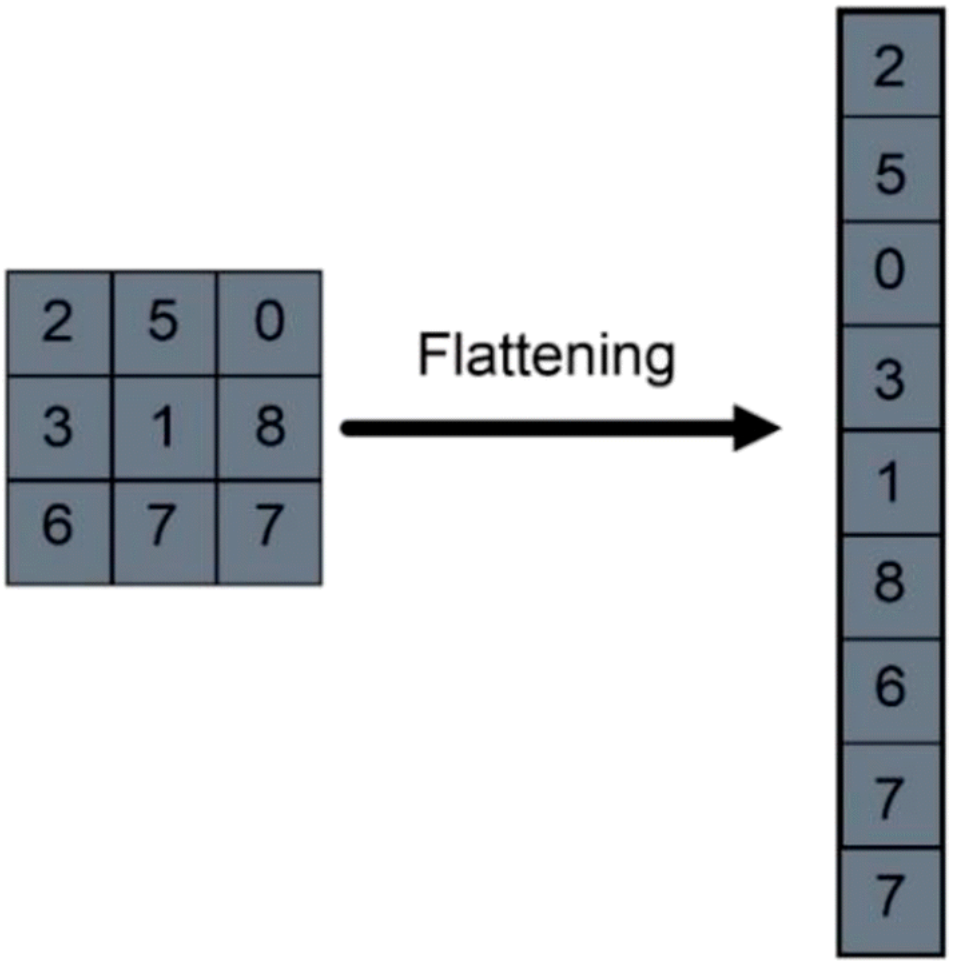
Open now!

2 Tutorial 2 recap

MLP + Backpropagation

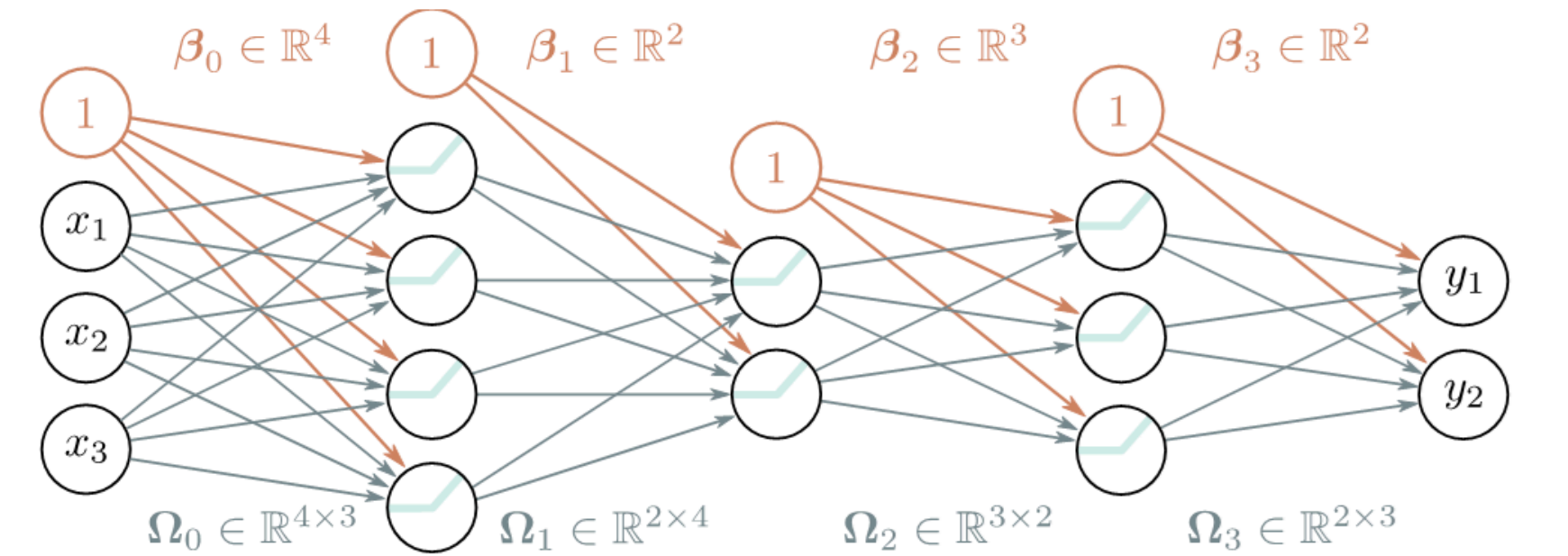
3.1 Representing images as tensors

Flattening a 1D image



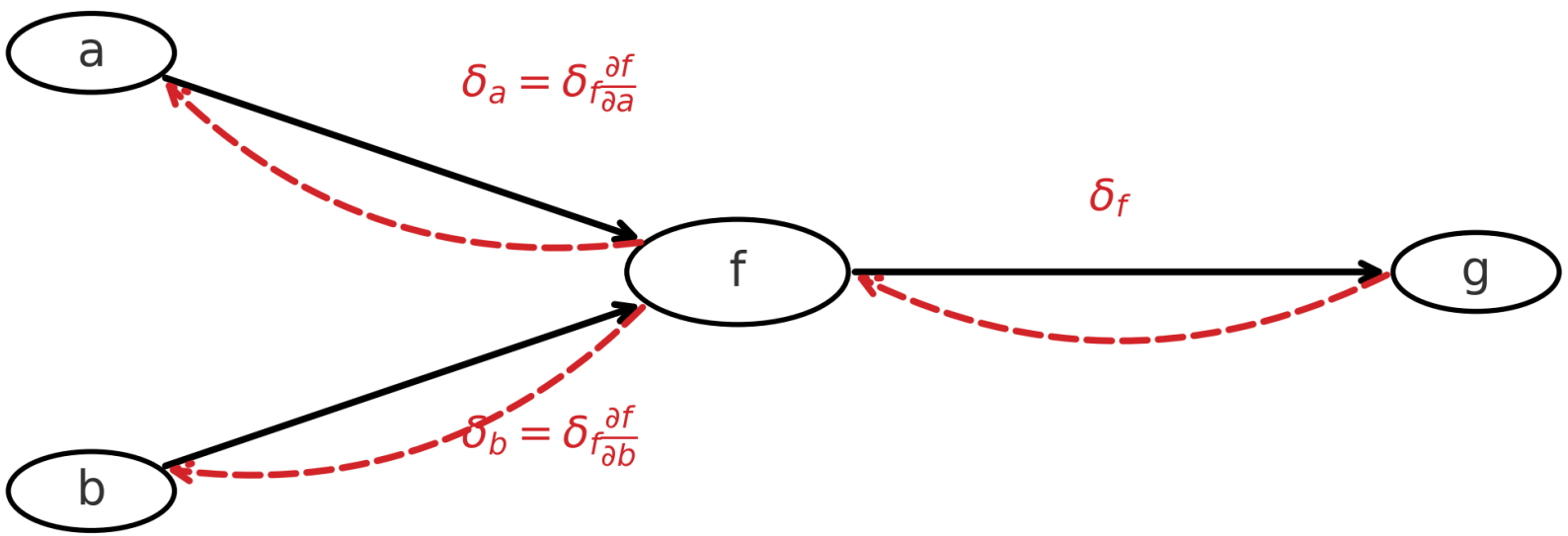
2.1 MLP

- Example: classify if an image is “5” or “6”



2.1 Backpropagation

- Used to compute gradients and update weights in neural networks.
- Example (MSE): $\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$



2.1 Training MLPs

- Almost a fully working code :-)

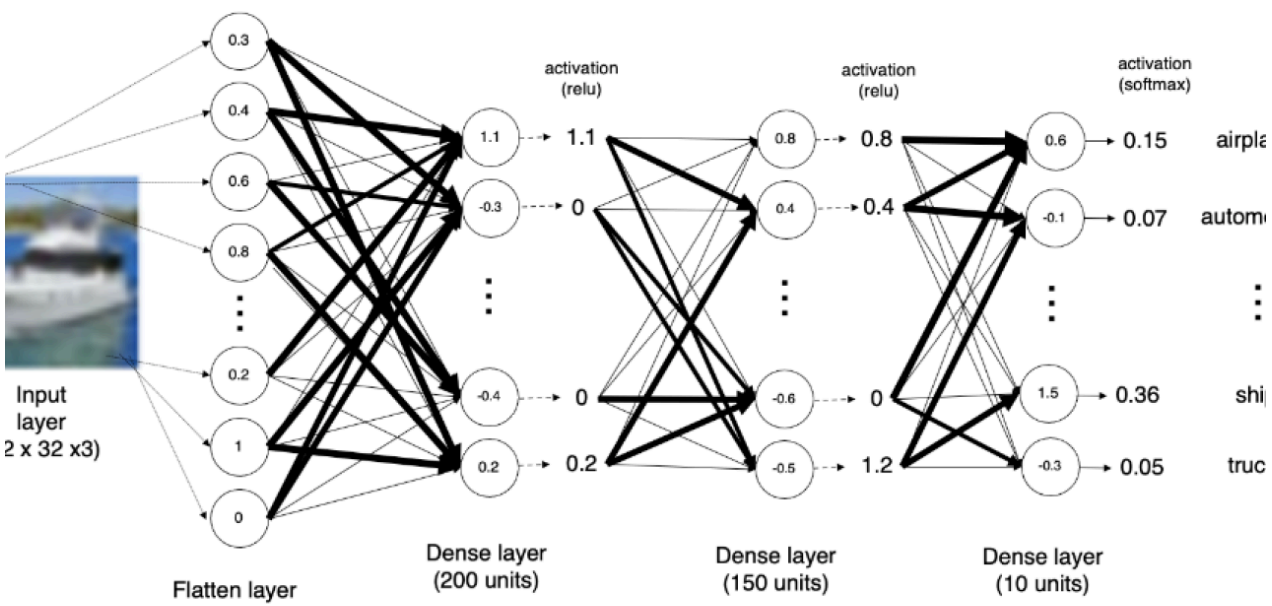
```
import SGD
import MSE

def model(X):
  ...
  return Y

def train(model, data, num_epochs=10):
    for epoch in range(num_epochs):
        for X, Y in data:
            SGD.zero_grad()
            Y_hat = model(X)
            loss = MSE(Y_hat, Y)
            loss.backward()
            SGD.step()
```

3 Convolutional Neural Networks (CNN)

Slides based on 236781 Deep learning course at the Technion (Dr. Haim Baskin)

3.2 Images and MLP

- MLPs (Multi-Layer Perceptrons) are fully connected networks
- Each neuron in an MLP is connected to every pixel in the input image
- MLPs struggle with high-dimensional data like images due to the large number of parameters - Nearby pixels statistically related
  - 224x224 RGB image = 150,528 dimensions



3.3 Convolution in 1D

Convolution* in 1D - Input vector x:

$$\mathbf{x} = [x_1, x_2, \ldots, x_I]$$

- Output is weighted sum of neighbors:

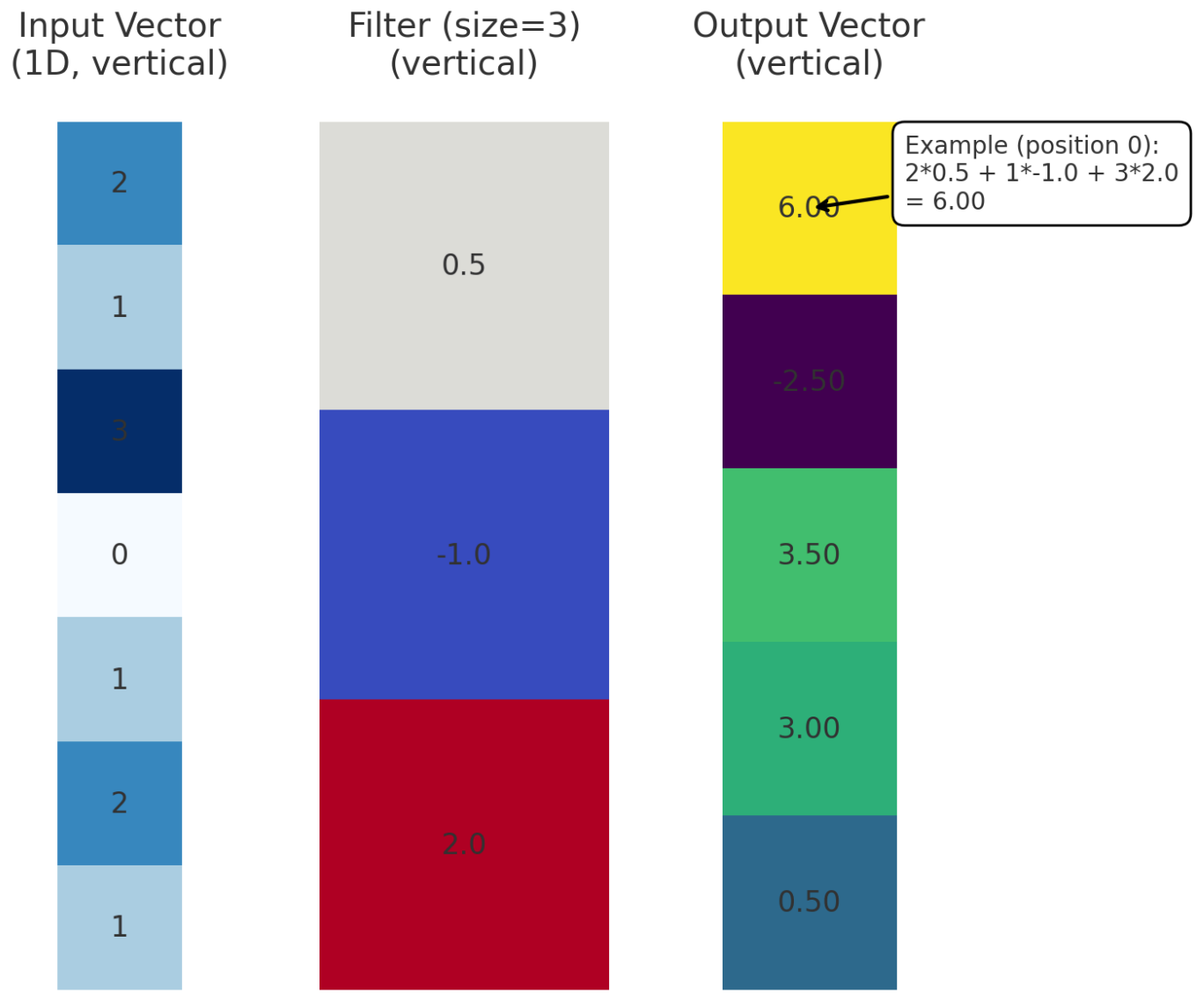
$$z_i = \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}$$

- Convolutional kernel or filter:

$$\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^T$$

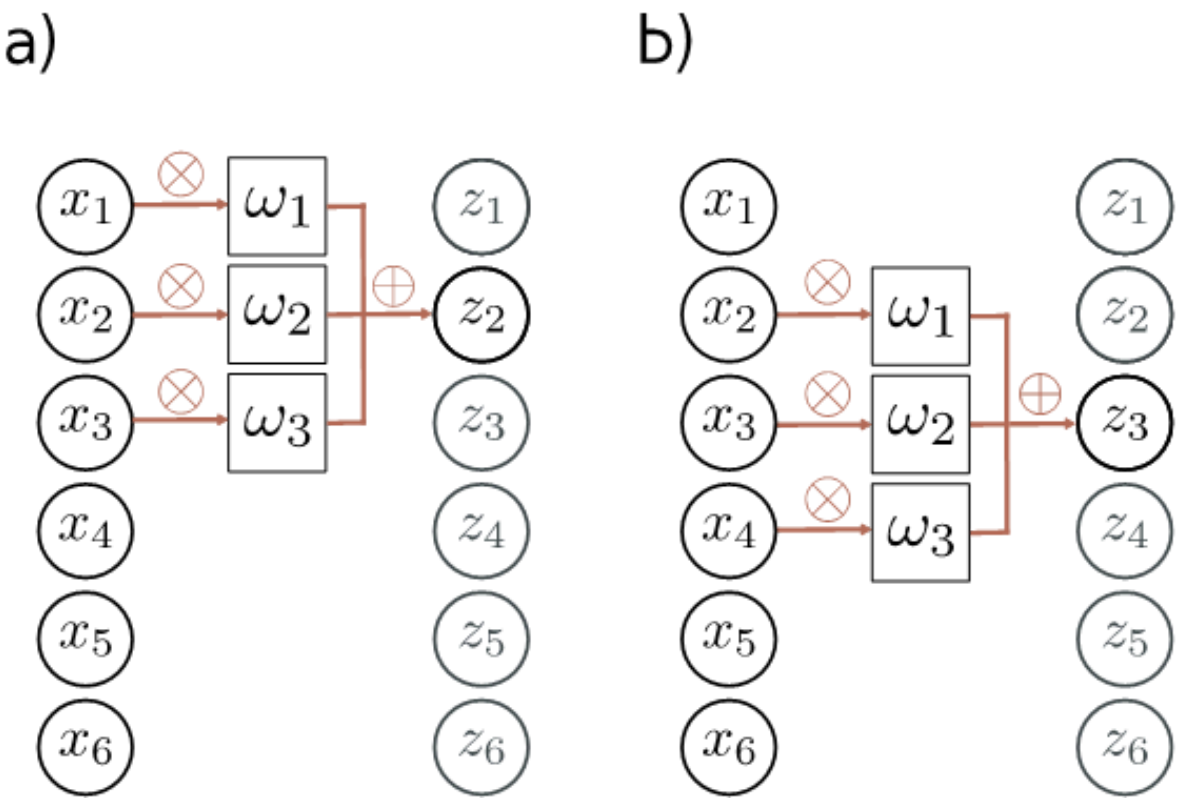
Kernel size $= 3$

Convoulution in 1D example



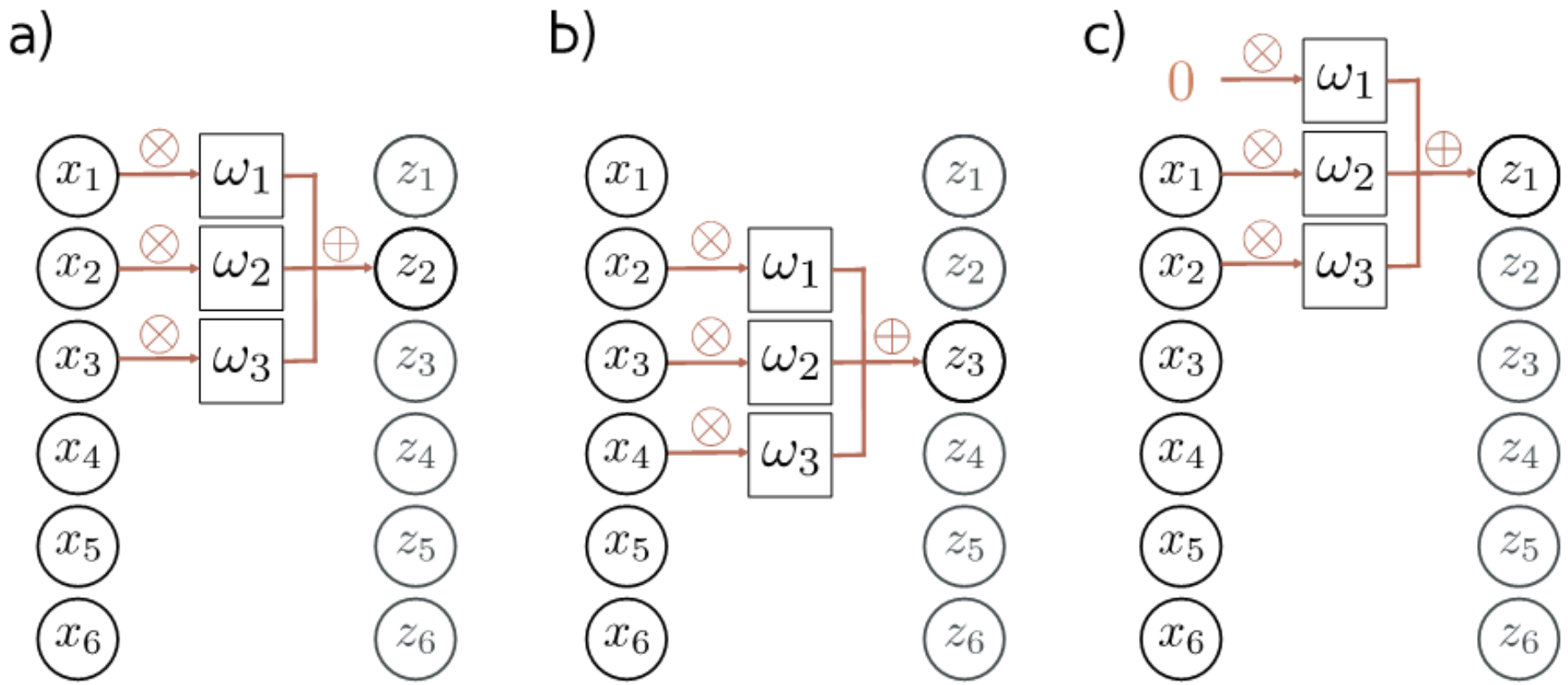
Convolution in 1D (cont.)

- **Parameter sharing**: The same filter is applied across the entire input, reducing the number of parameters.



Zeros padding

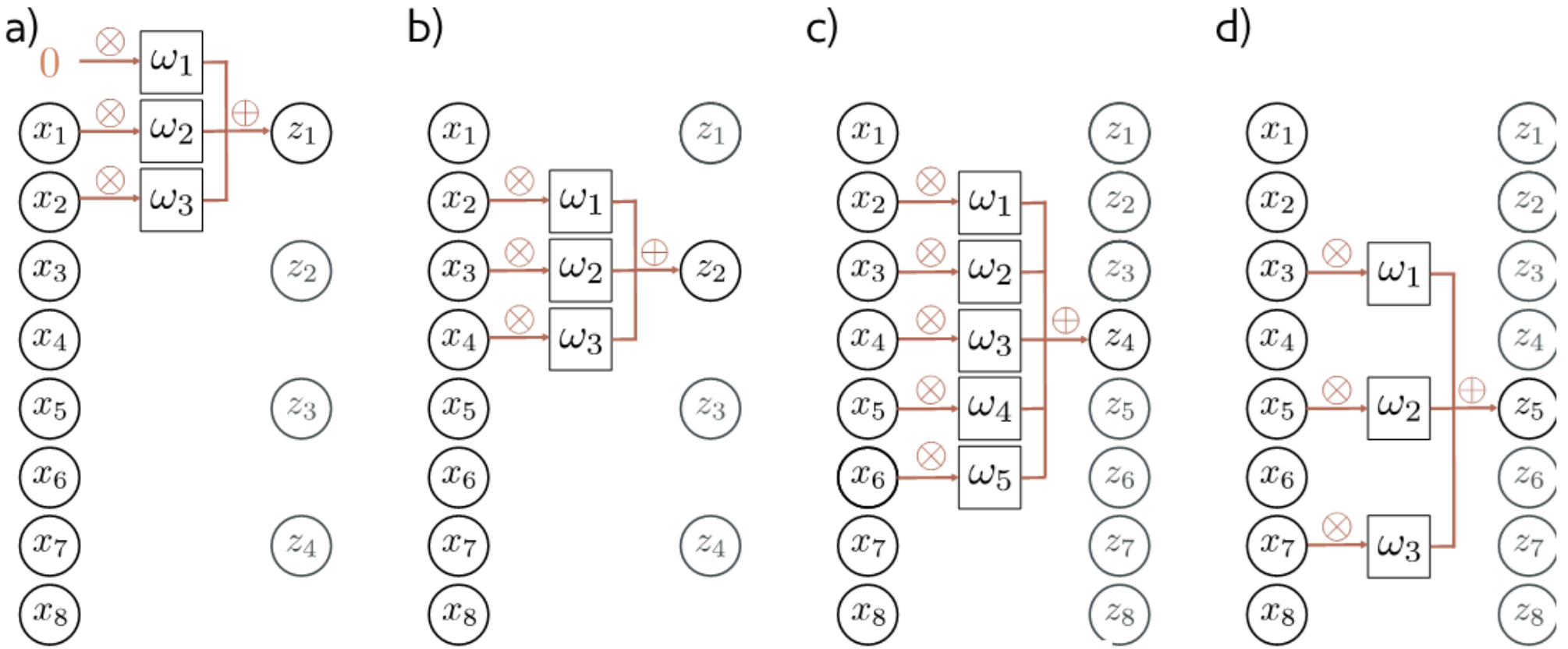
- To maintain the same output size as input (not always), we can add zeros around the input vector.



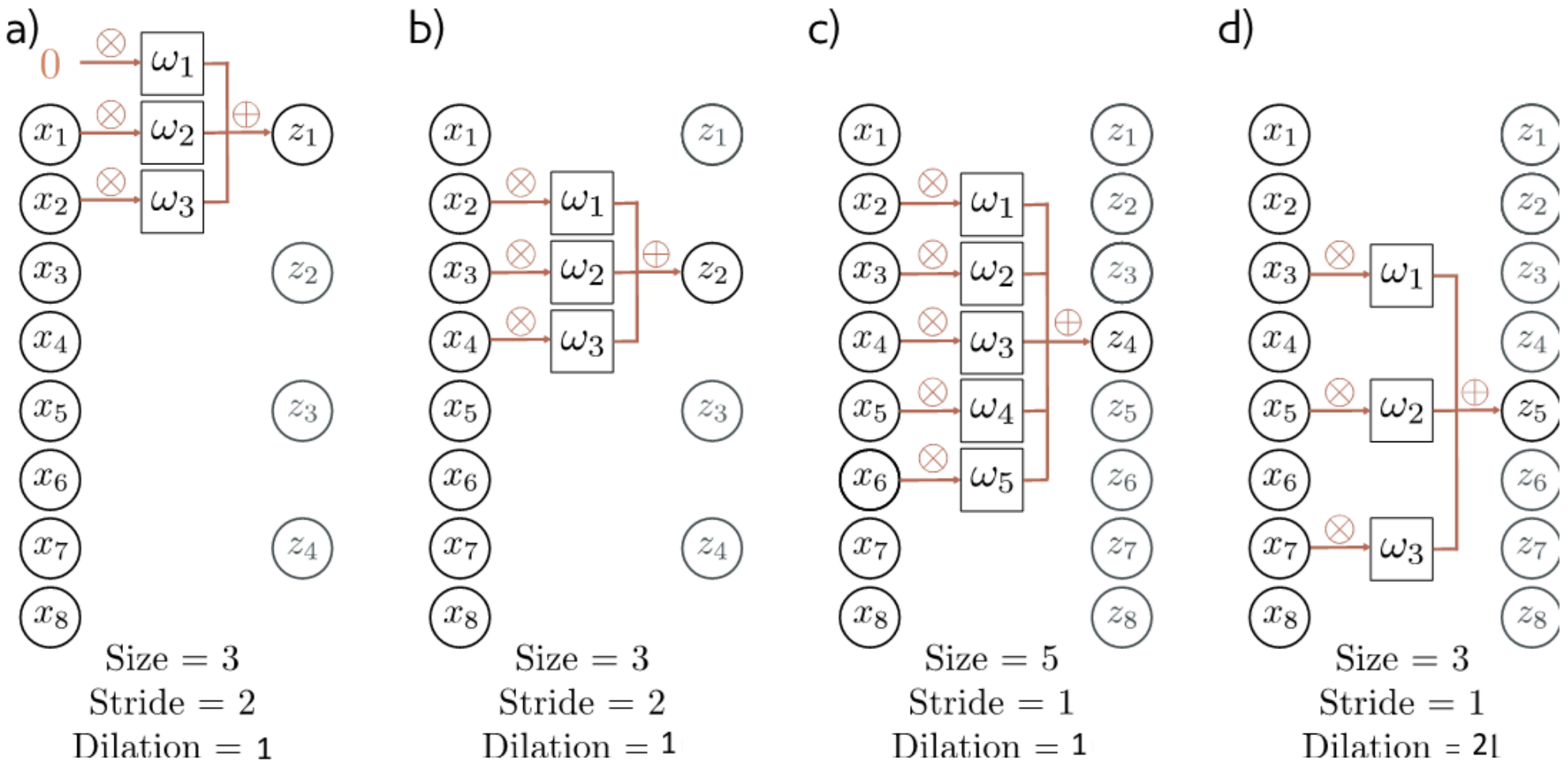
Properties of convolutional filters

- **Stride**: shift by k positions for each output
  - Decreases size of output relative to input
- **Kernel size**: weight a different number of inputs for each output
  - Combine information from a larger area
  - But kernel size 5 uses 5 parameters
- **Dilation**: insert zeros between weights
  - Combine information from a larger area
  - Fewer parameters

Stride, Kernel Size, and Dilation



Stride, Kernel Size, and Dilation



Fully connected vs. convolutional layers

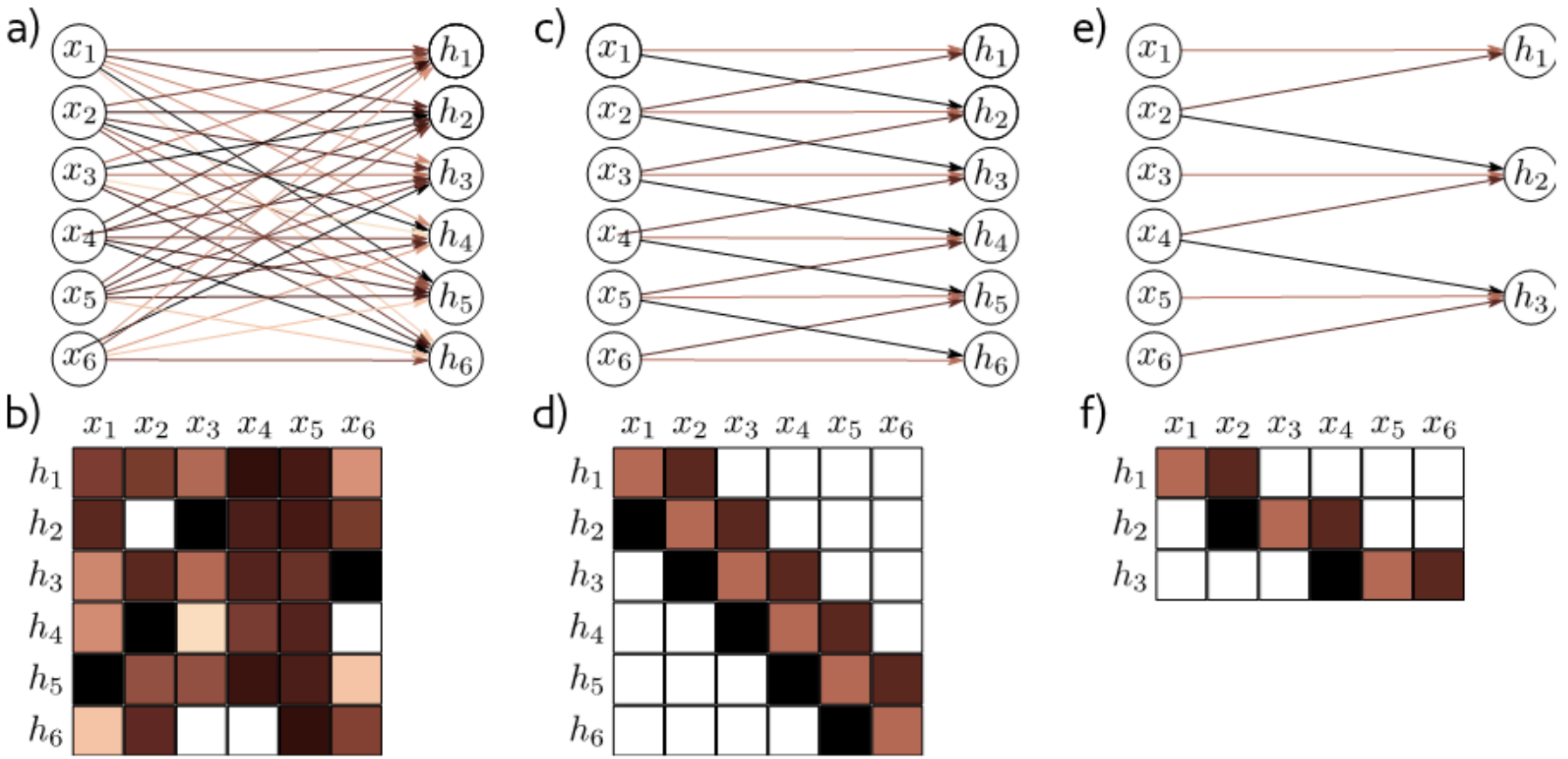
- Fully connected layers connect every input to every output, leading to a large number of parameters.
- Convolutional layers use filters (kernels) to scan the input, sharing weights across space and reducing the number of parameters. Convolutional network:

$$
\begin{aligned}
h_i &= \mathrm{a}\left[\beta + \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}\right] \\
&= \mathrm{a}\left[\beta + \sum_{j=1}^{3} \omega_j x_{i+j-2}\right]
\end{aligned}
$$

Fully connected network:

$$
h_i = \mathrm{a}\left[\beta_i + \sum_{j=1}^{D} \omega_{ij} x_j\right]
$$

Special case of fully-connected network



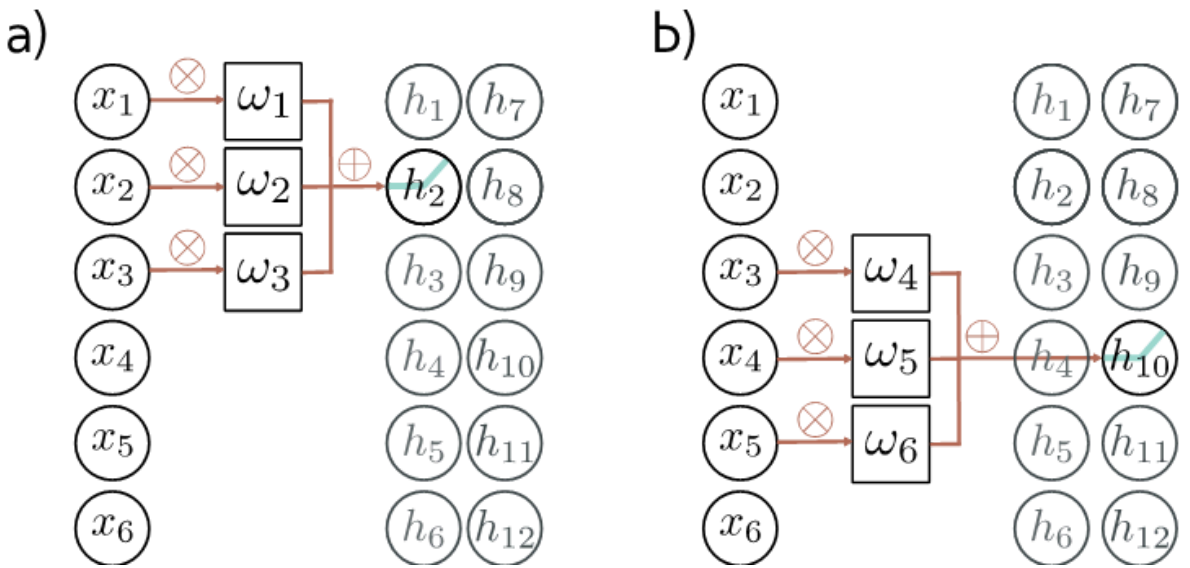
Fully connected network

Convolution, size 3, stride 1, dilation 1, zero padding

Convolution, size 3, stride 2, dilation 1, zero padding

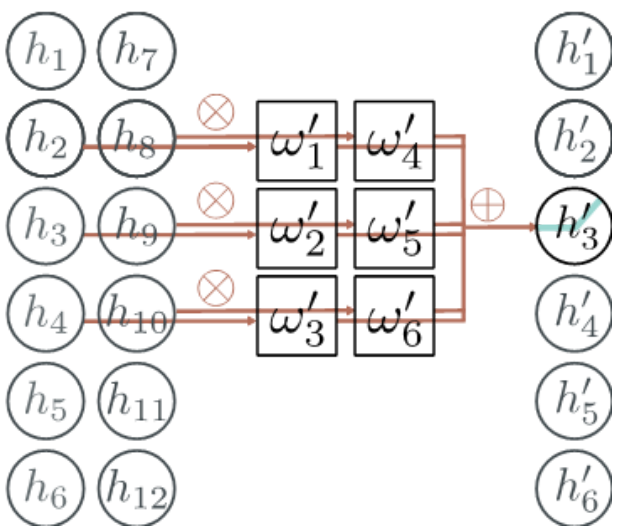
Two output channels

- Convolutional layers can have multiple output channels (feature maps).
- Each channel is produced by a different filter.
- Prevents information loss and allows the network to learn different features.



Two input channels, one output channel

- Convolutional layers can also handle multiple input channels (e.g., RGB images).
- Each input channel is convolved with its own filter, and the results are summed to produce the output channel.
- This allows the network to learn features from different input channels.



How many paramters?

How many parameters? - If there are C_i input channels and kernel size K

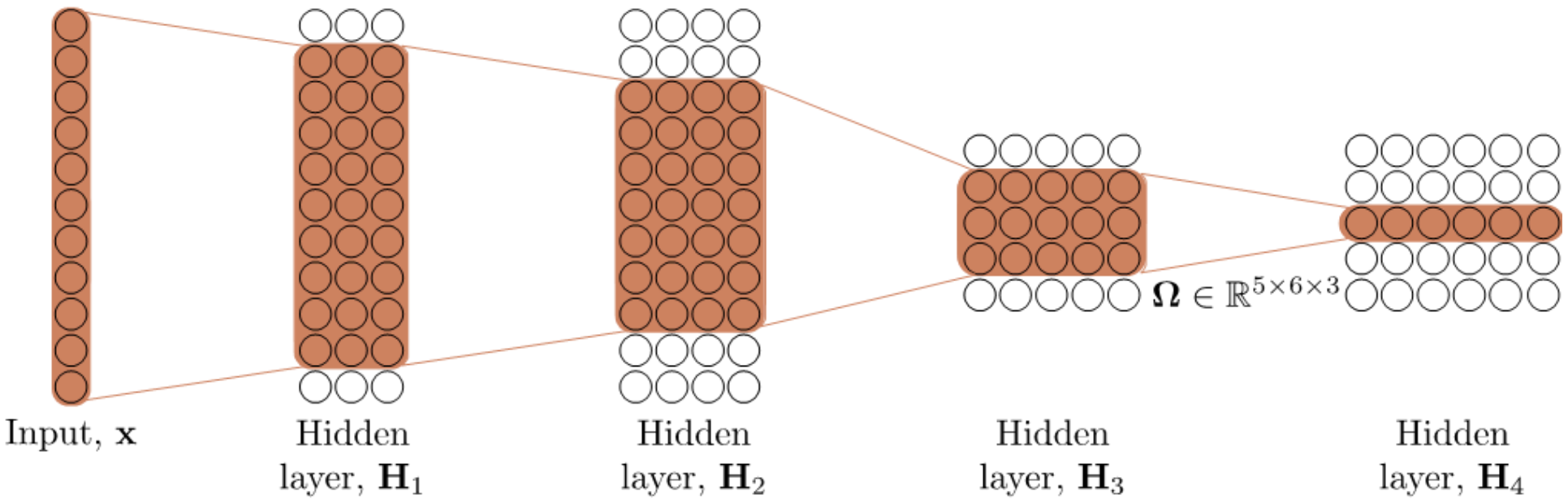
$$\boldsymbol{\Omega} \in \mathbb{R}^{C_i \times K} \quad \boldsymbol{\beta} \in \mathbb{R}$$

- If there are C_i input channels and C_o output channels

$$\boldsymbol{\Omega} \in \mathbb{R}^{C_i \times C_o \times K} \quad \boldsymbol{\beta} \in \mathbb{R}^{C_o}$$

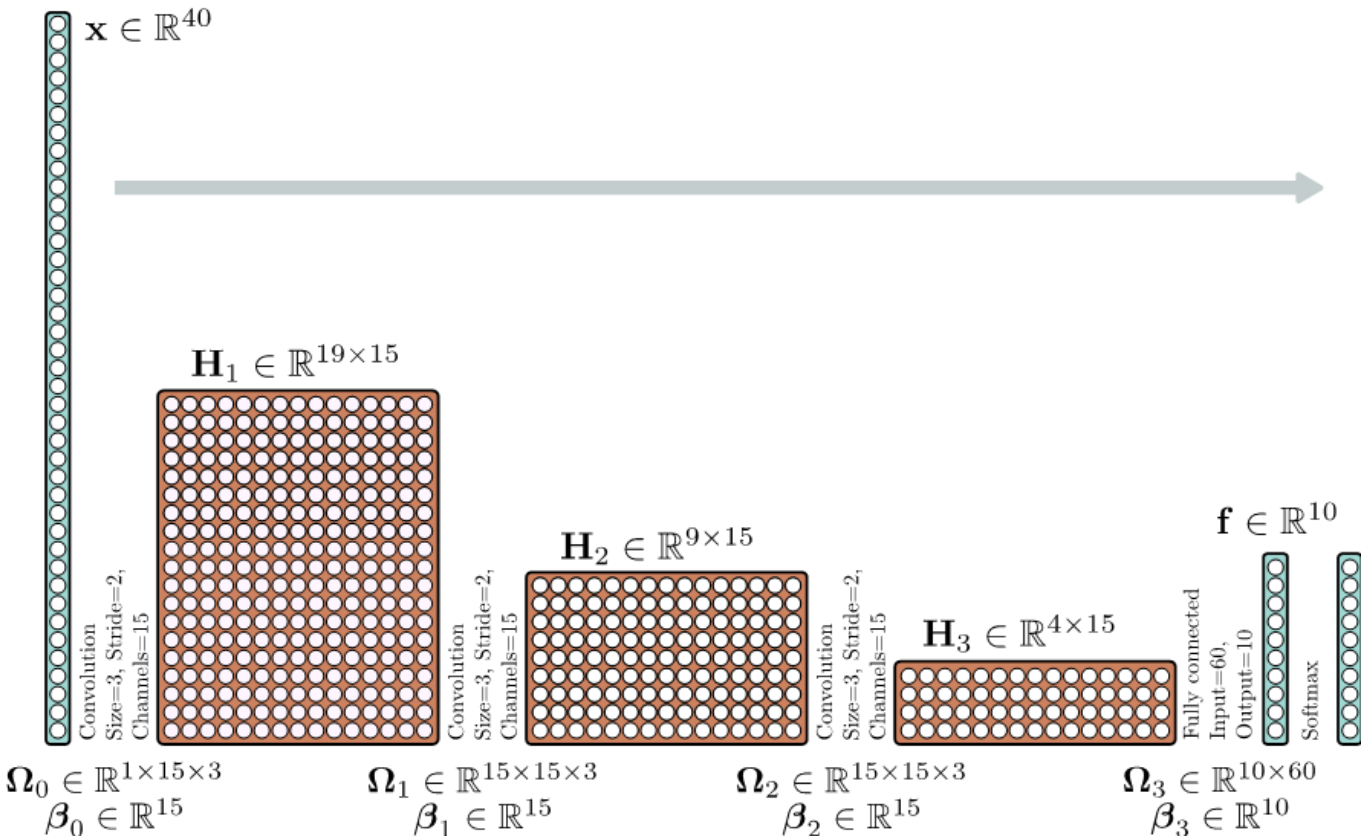
Receptive fields

- The receptive field of a neuron is the region of the input that affects its output.

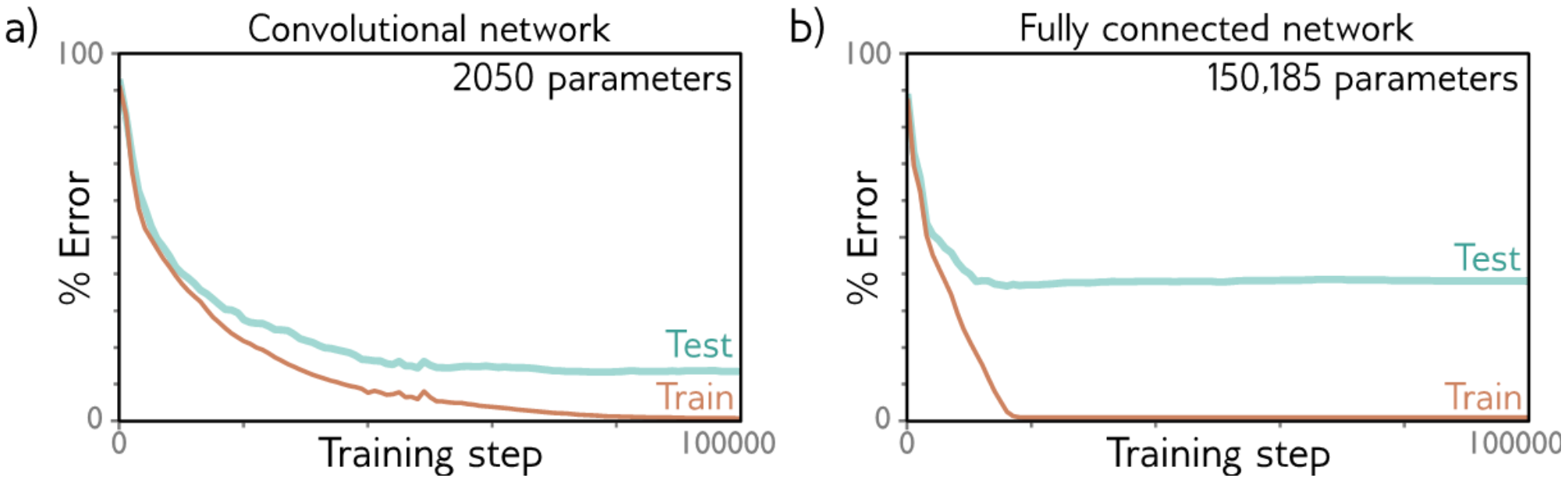


Classification using CNNs

- Input: flattened image
- Output: probabilities for each class

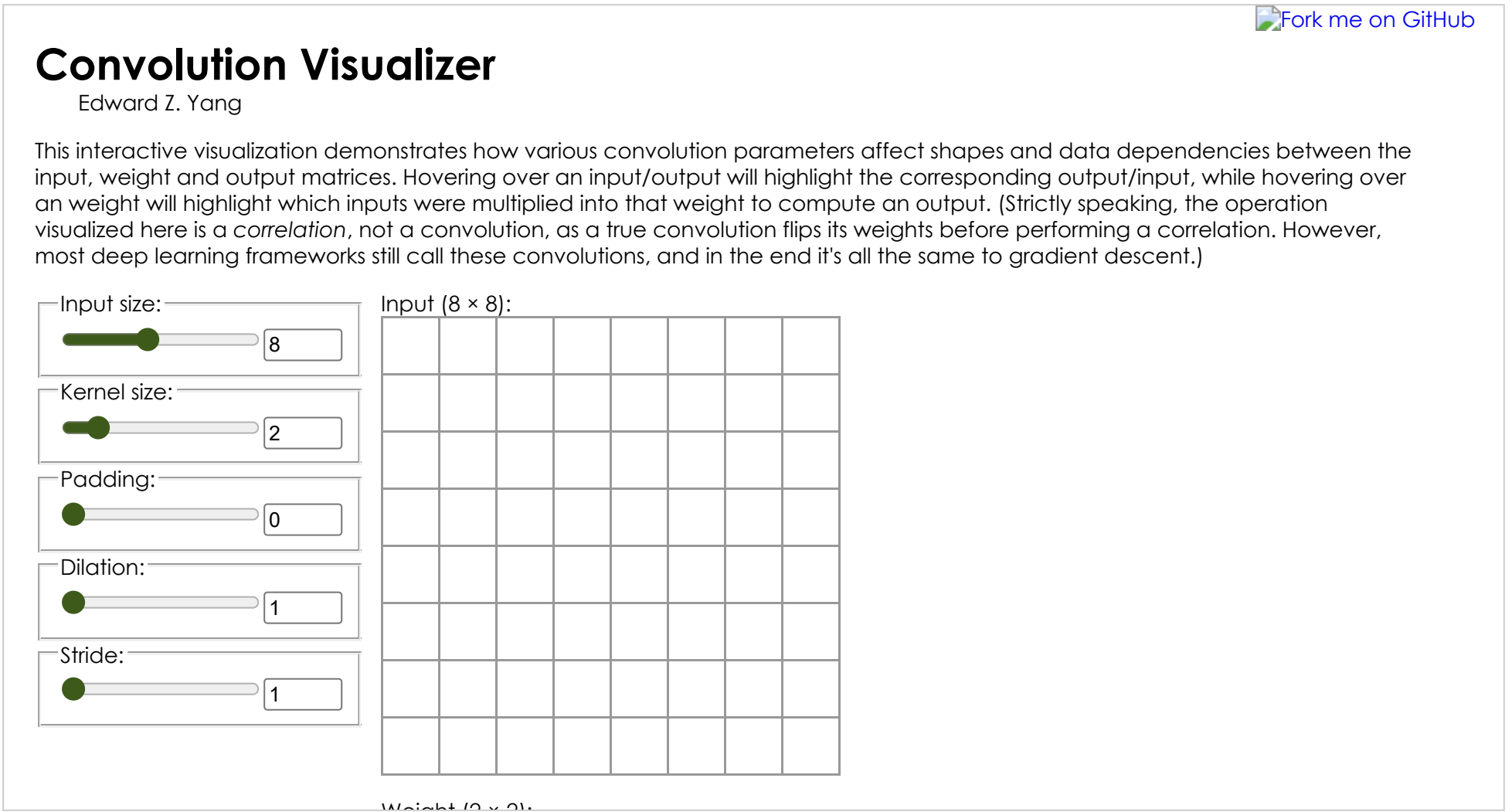


Results of classification using CNNs

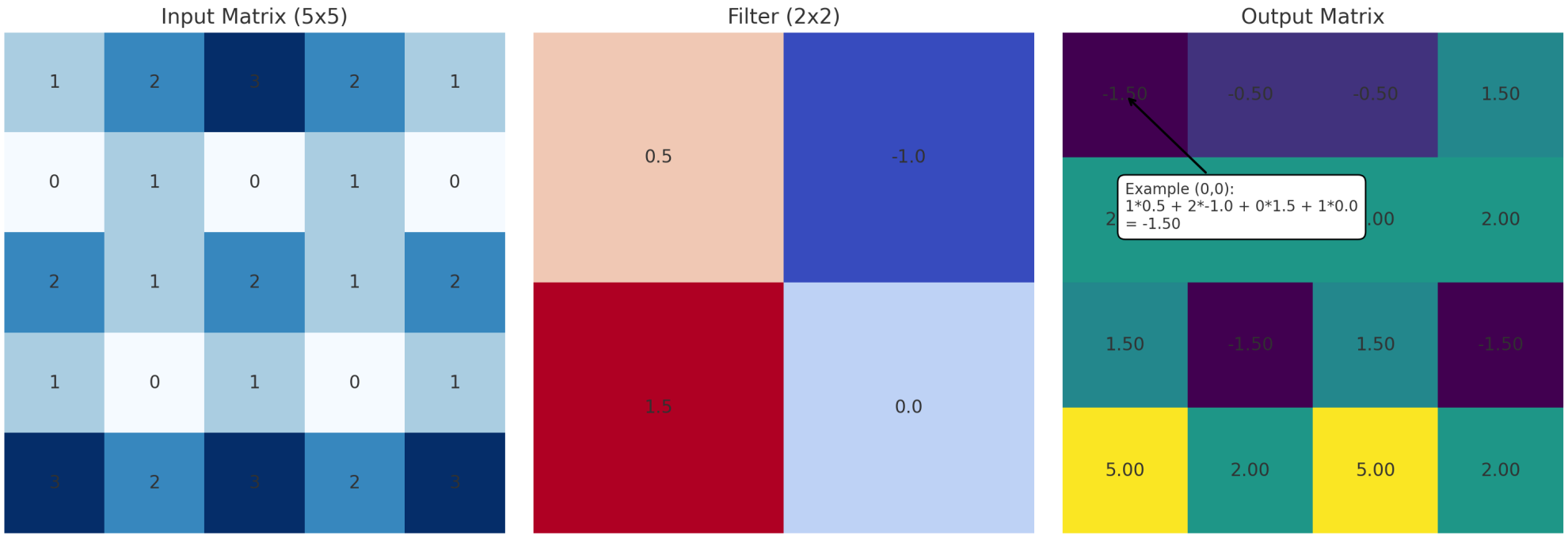


2D Convolution

- Input image is a 2D matrix, and the filter is also a 2D $K \times K$ matrix.



Convoulution in 2D example



Number of Parameters in 2D Convolution

- If there are C_i input channels and kernel size $\mathrm{K} \times \mathrm{K}$

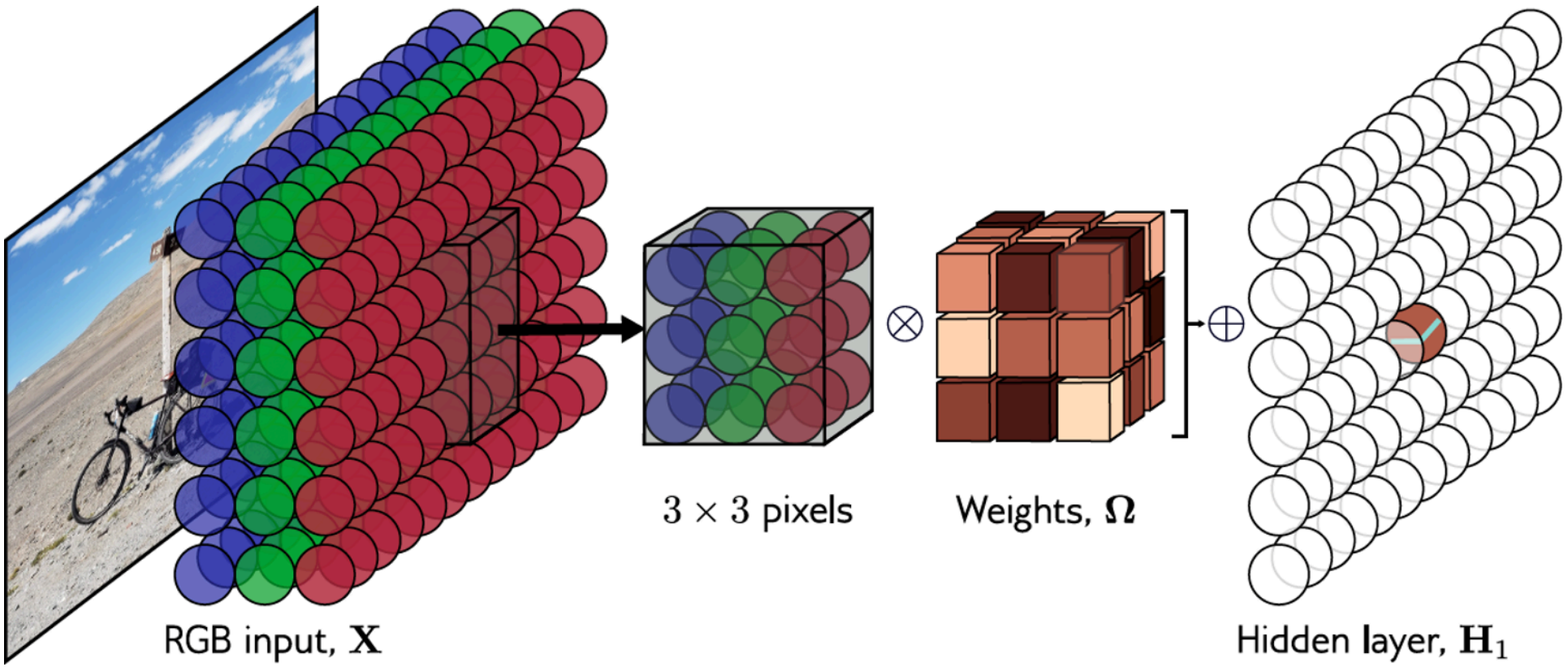
$$\boldsymbol{\omega} \in \mathbb{R}^{C_i \times K \times K} \quad \boldsymbol{\beta} \in \mathbb{R}$$

- If there are C_i input channels and C_o output channels

$$\boldsymbol{\omega} \in \mathbb{R}^{C_i \times C_o \times K \times K} \quad \boldsymbol{\beta} \in \mathbb{R}^{C_o}$$

Channel in 2D Convolution

- Each channel in a 2D convolution corresponds to a different feature map.

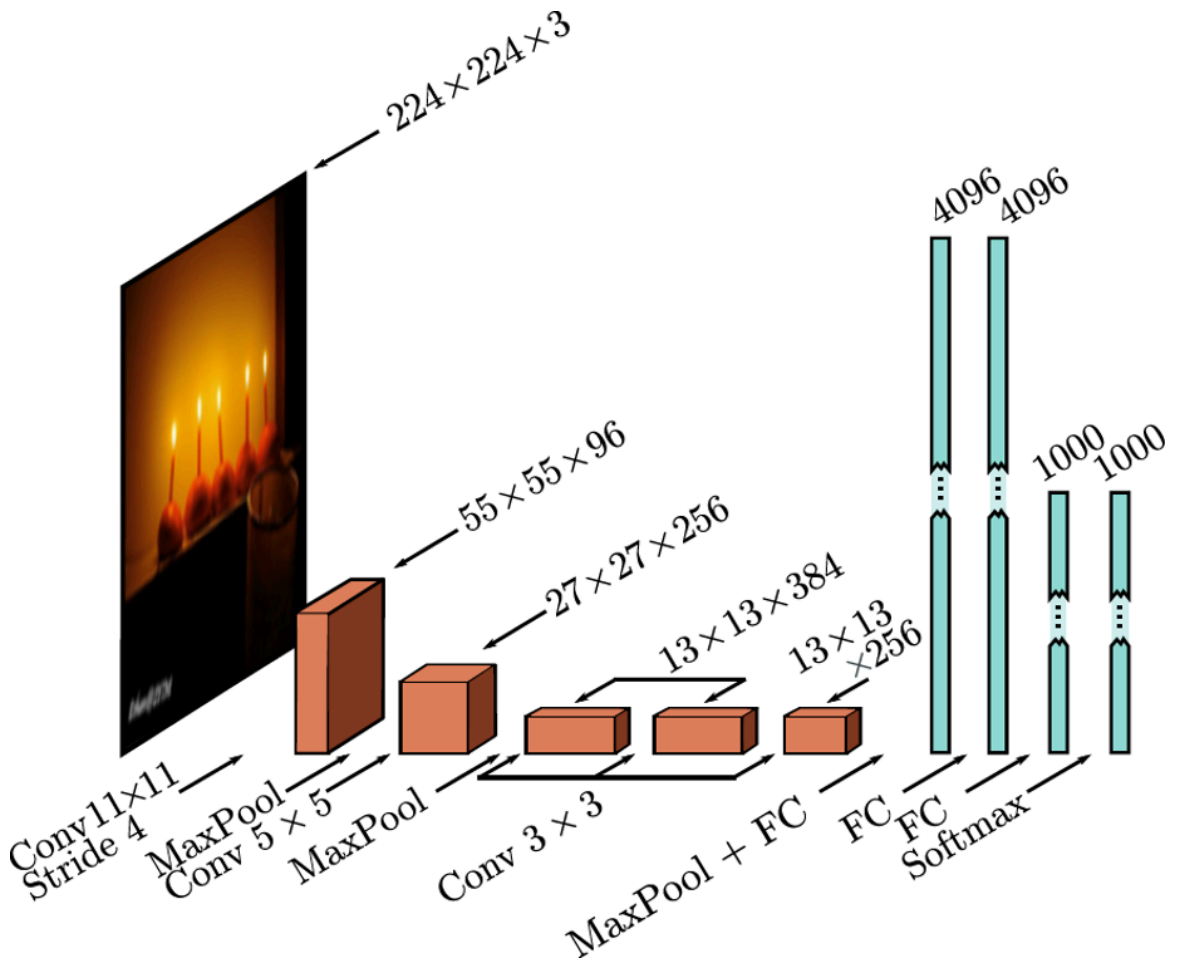


Pooling

- Pooling layers reduce the spatial dimensions of the input, retaining important features while reducing computational complexity.
- No learnable parameters.

AlexNet (2012)

- One of the first successful CNN architectures



But what does the CNN “see”?

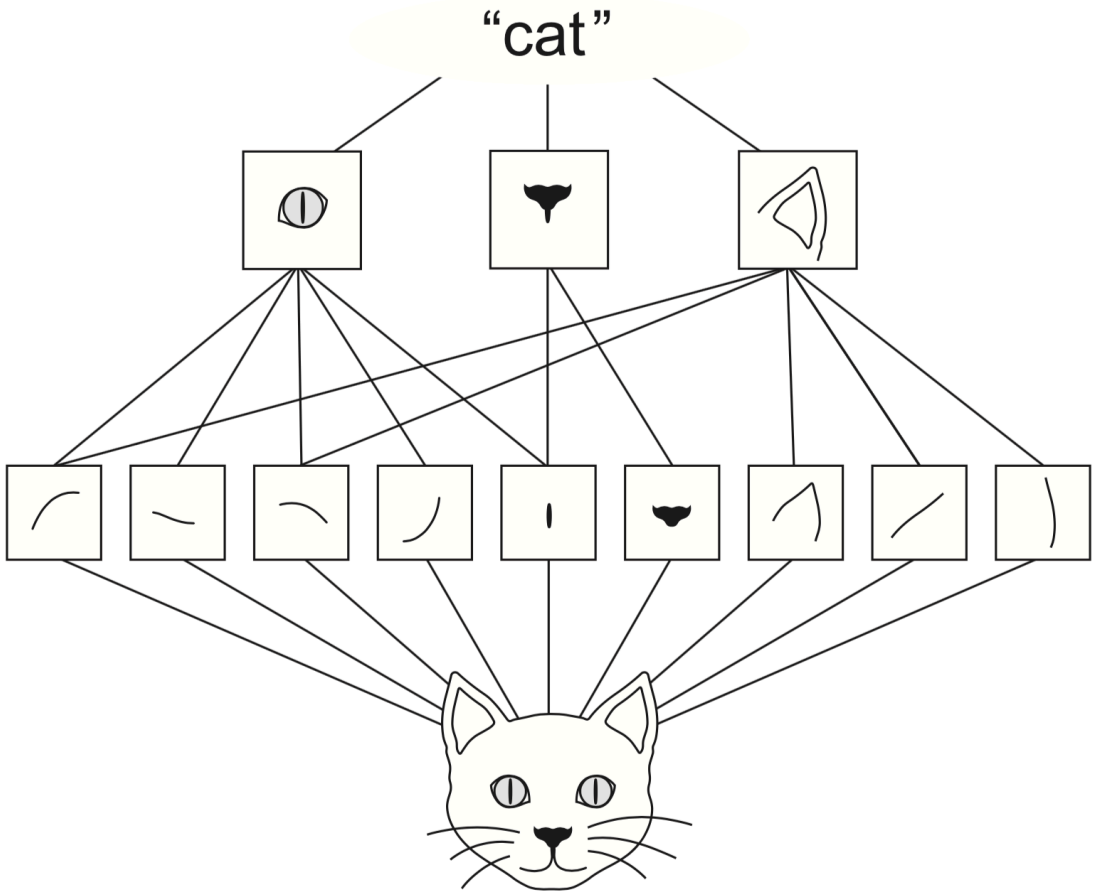
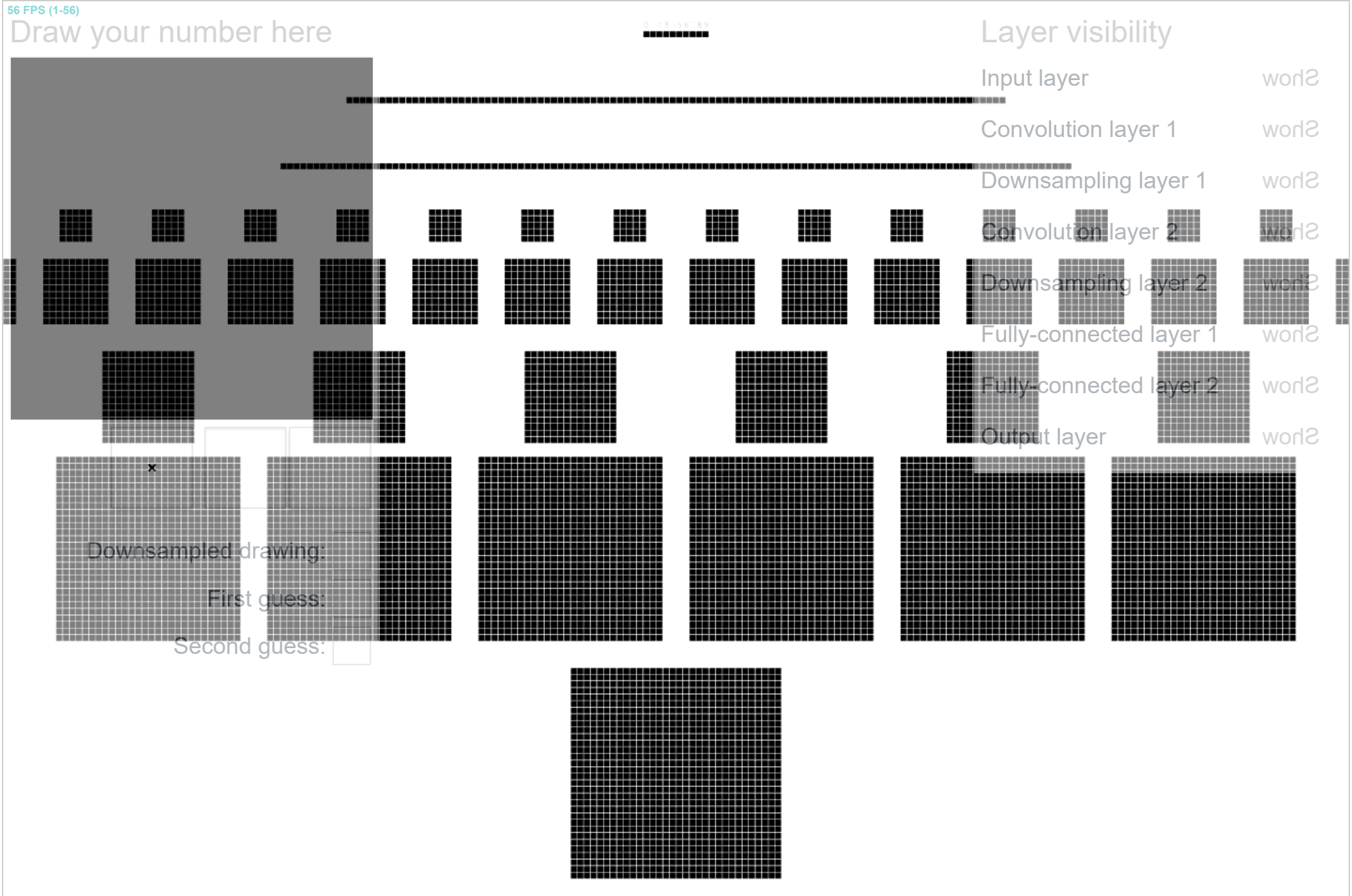


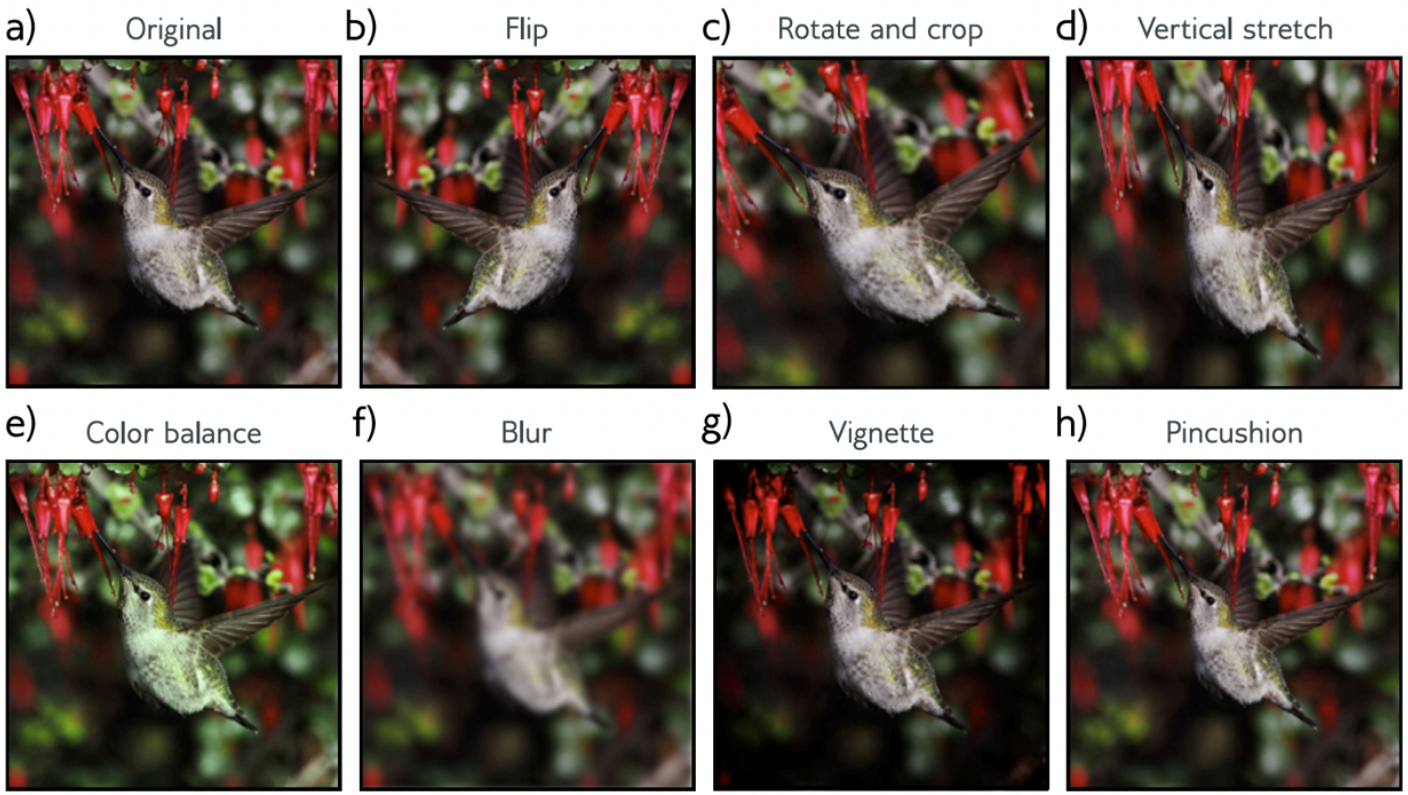
Figure 5.2 The visual world forms a spatial hierarchy of visual modules: hyperlocal edges combine into local objects such as eyes or ears, which combine into high-level concepts such as “cat.”

But what does the CNN really “see”?


56 FPS (1-56)
Draw your number here
Layer visibility
Input layer
Convolution layer 1
Downsampling layer 1
Convolution layer 2
Downsampling layer 2
Fully-connected layer 1
Fully-connected layer 2
Output layer
Downsampled drawing:
First guess:
Second guess:

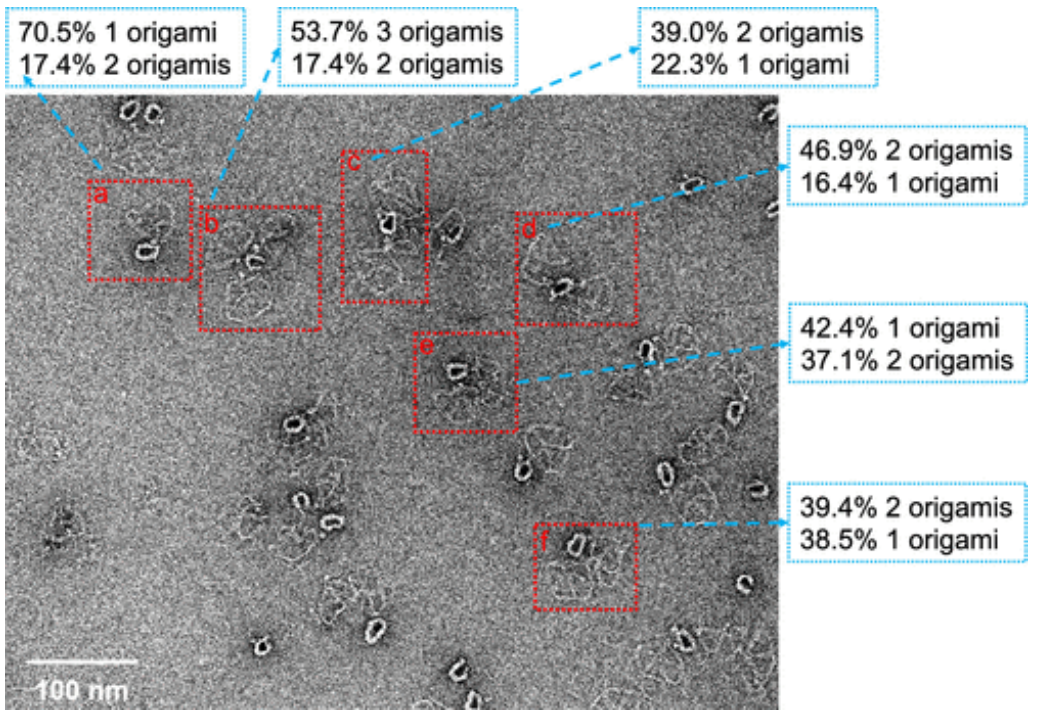
Data augmentation

- Data augmentation is a technique to artificially increase the size of the training dataset by applying transformations to the original images.
- Common transformations include rotation, scaling, flipping, and cropping.



Bounding boxes (VGG16) example

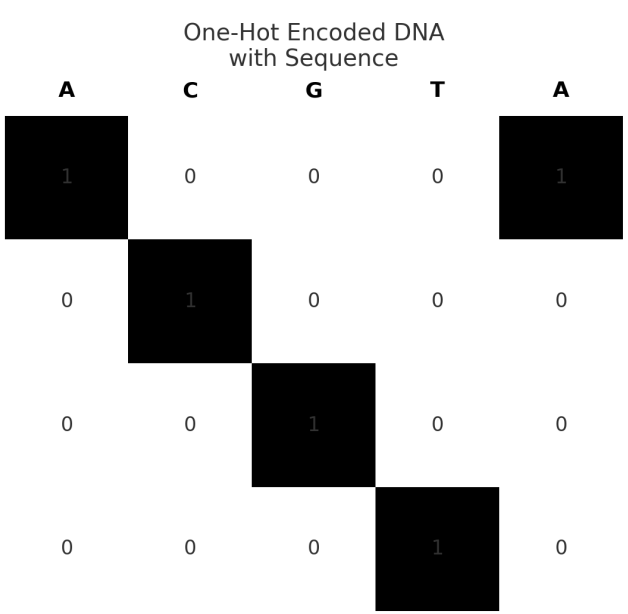
- Goal: characterize the ligation number of DNA origami nanostructures in transmission electron microscopy (TEM) images

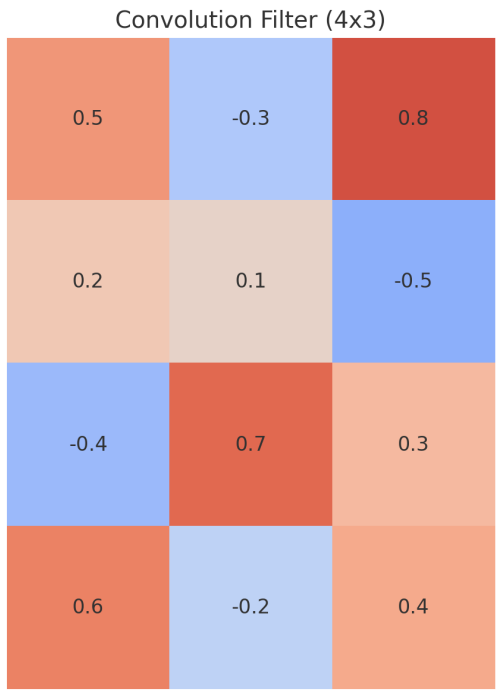


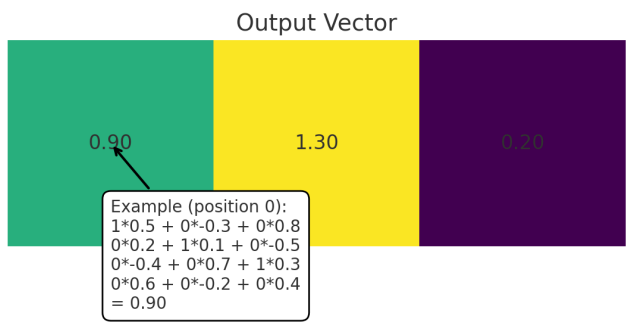
[1] Characterizing DNA Origami Nanostructures in TEM Images Using Convolutional Neural Networks Xingfei Wei, Qiankun Mo, Chi Chen, Mark Bathe, and Rigoberto Hernandez Journal of Chemical Information and Modeling Article ASAP DOI: 10.1021/acs.jcim.5c00330

One-Hot Encoding for DNA Sequences

- One-hot encoding is a technique to represent categorical variables as binary vectors.
- Each category is represented by a vector with a single 1 and the rest 0s.
- For DNA sequences, each nucleotide (A, T, C, G) is represented by a unique vector.







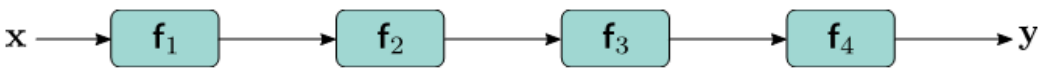
3 Residual Connections

3.1 Residual Connections

- Residual connections allow gradients to flow more easily through the network.

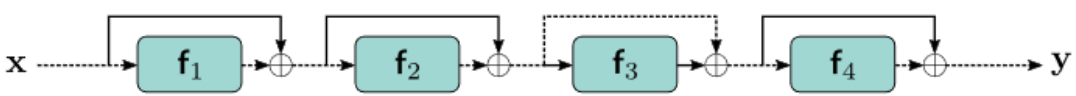
Regular network:

$$\begin{aligned}\mathbf{h}_1 &= \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1] \\ \mathbf{h}_2 &= \mathbf{f}_2[\mathbf{h}_1, \boldsymbol{\phi}_2] \\ \mathbf{h}_3 &= \mathbf{f}_3[\mathbf{h}_2, \boldsymbol{\phi}_3] \\ \mathbf{y} &= \mathbf{f}_4[\mathbf{h}_3, \boldsymbol{\phi}_4]\end{aligned}$$

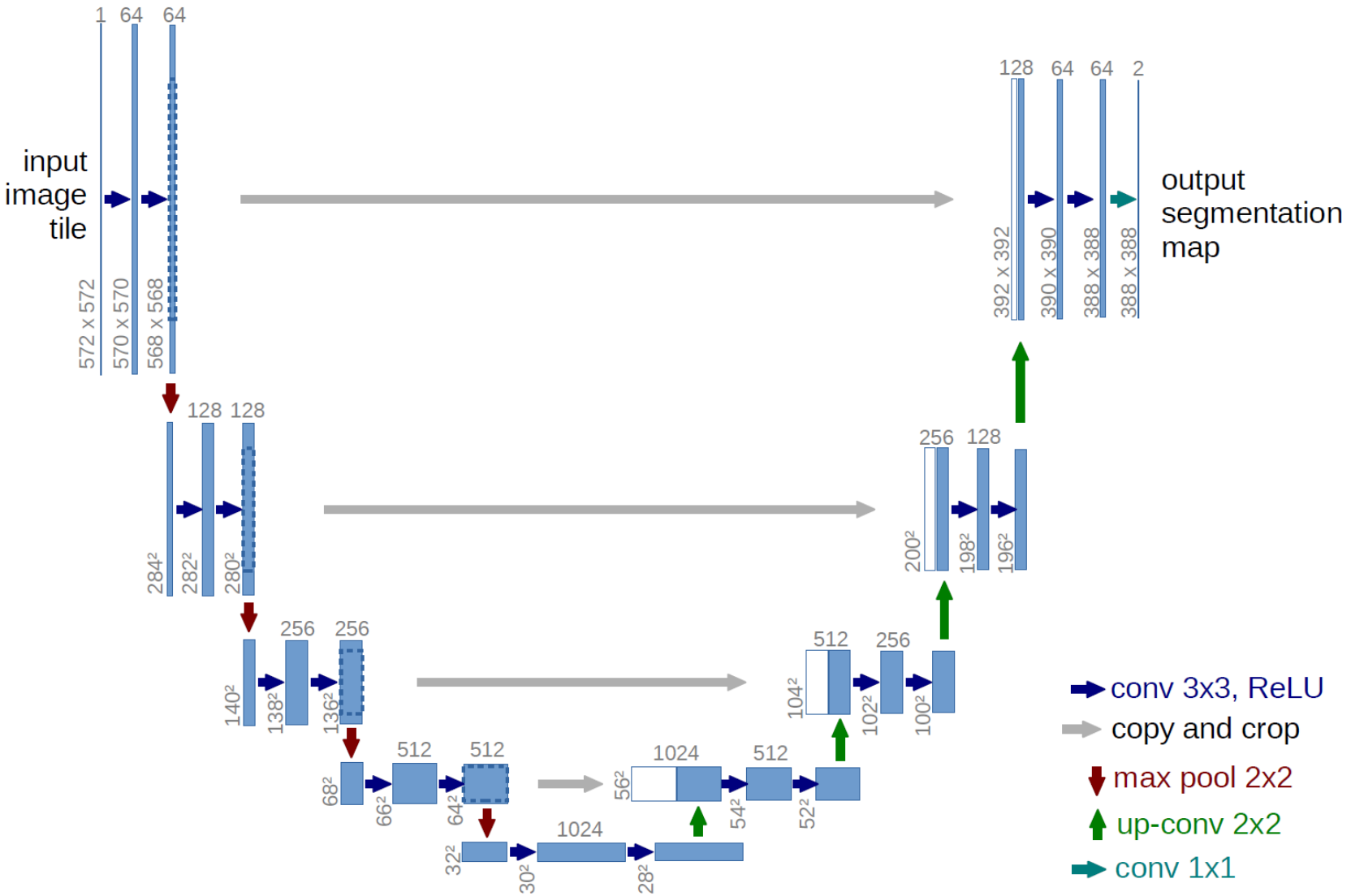


Residual network (2016):

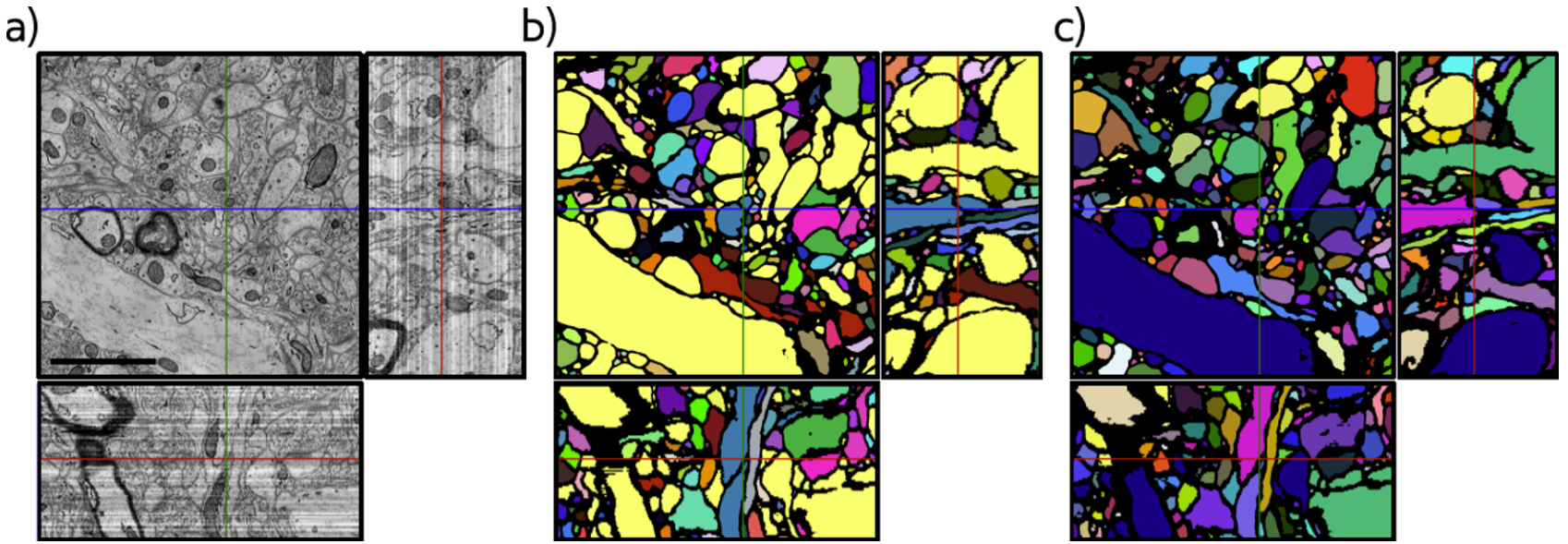
$$\begin{aligned}\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1] \\ \mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \boldsymbol{\phi}_2] \\ \mathbf{h}_3 &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \boldsymbol{\phi}_3] \\ \mathbf{y} &= \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \boldsymbol{\phi}_4]\end{aligned}$$



3.2 U-Net

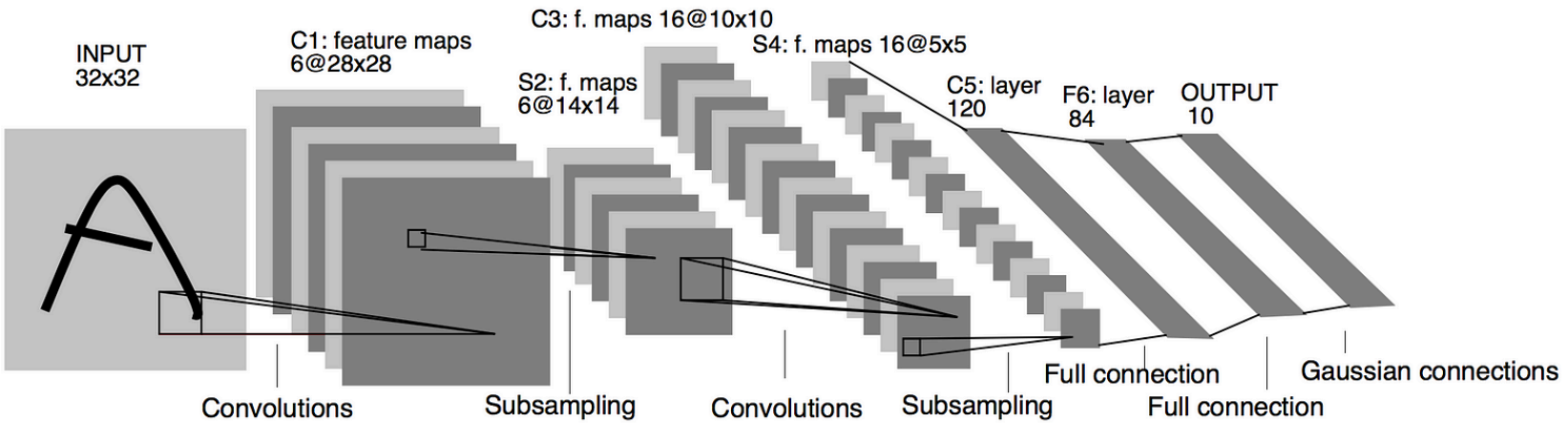

input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

3.3 U-Net: Results



4 Code

Lenet (LeCun, 1998)




```python
class LeNet(nn.Module):
    def __init__(self, in_channels=3):
        super().__init__()
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(in_channels, out_channels=6, kernel_size=5),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=
            nn.ReLU()
```

Training Lenet (LeCun, 1998)

- Almost a fully working code
- Almost identical to the MLP code from the previous tutorial

```
def train_lenet(model, data, loss_fn, optimizer, num_epochs=10):
    for epoch in range(num_epochs):
        for images, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```