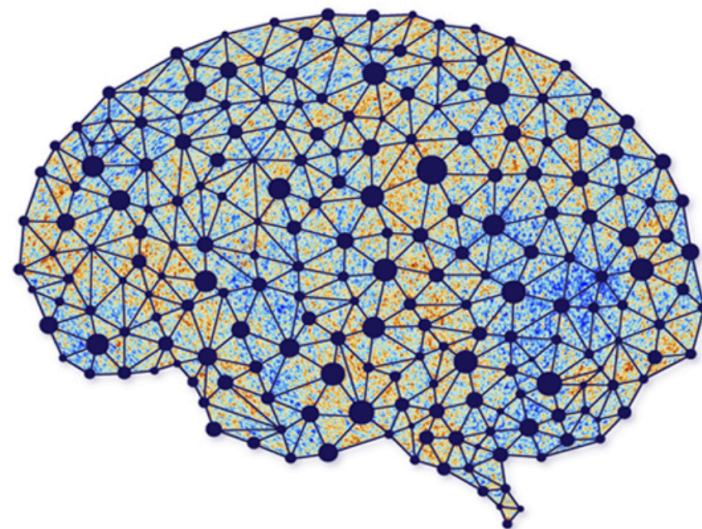


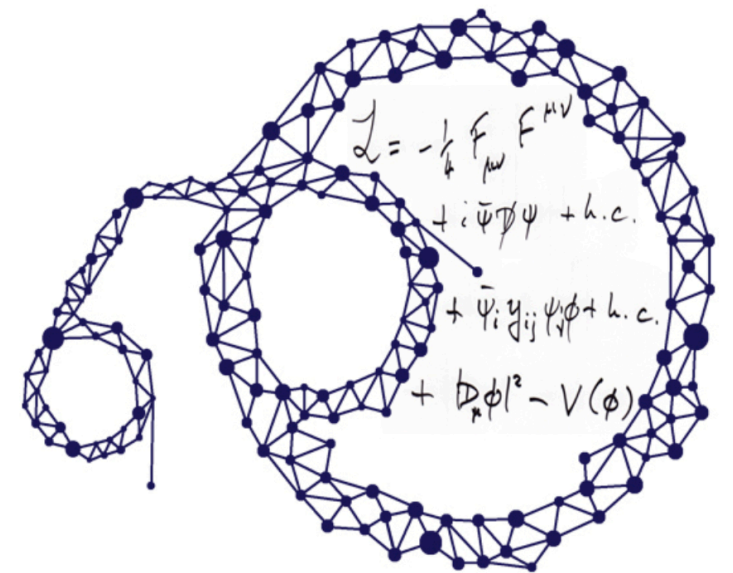
Physics 361 - Machine Learning in Physics

Lecture 9 – Convolutional Neural Networks

Feb. 19th 2025



AI
∩
Universe



Moritz Münchmeyer

Final project

- You will write **a paper on an application of machine learning to physics** of your choice. Your paper needs to contain a computational analysis, which generally will mean applying a machine learning method to some data set.
- You can **work alone or in groups of up to four people. For larger groups we will expect a little bit more total effort.**
- The paper should be **5 to 10 pages** and contain the following:
 - A short review of at least one research paper related to your topic. This is to encourage you to learn how to browse the literature.
 - A description of the data set you will be working with and its properties.
 - A brief description of the machine learning method you will use. Don't re-explain basics such as how CNNs work, rather describe the detailed properties of your approach.
 - Train the model and put the results in your paper. Explore some variations such as different hyper parameters.
 - Describe successes and problems in your analysis.
- If you are already doing research in physics or a related field, you can write the paper on this topic if you wish.

Final project

- You can **use machine learning methods either from the lecture or ones that we have not covered**. Major topics which we have not yet covered but will be covering in the next weeks are Generative models (GANs, Diffusion, Normalizing Flows), Simulation Based Inferences (which includes the important topic of assigning error bars to neural network estimates), and Transformers and LLMs.
- The project should take you a few days of work, spread over the rest of the semester.
- We will have an **intermediate check-in. Format TBA.**
- Your **paper will be due on Sunday May 4th at midnight.**
- We want to **know your topic by March 11th**. You can discuss your topic ideas with Yurii or with me, after the lecture, or during office hours.
- **We will have a brief ~1 slide presentation of your results in the lecture on April 29th.**

Some sources of data

- Kaggle competitions
 - <https://www.kaggle.com/search?q=physics>
 - <https://www.kaggle.com/search?q=physics+in%3Adatasets>
 - <https://www.kaggle.com/search?q=physics+in%3Acompetitions>
 - <https://www.kaggle.com/competitions/icecube-neutrinos-in-deep-ice>
- Shared Data and Algorithms for Deep Learning in Fundamental Physics
<https://github.com/erum-data-idt/pd4ml>
- <https://astronn.readthedocs.io/en/latest/galaxy10.html>
- Many others. Please look in the domain you are interested in.

Some sources of models

- Scikit learn: <https://scikit-learn.org/stable/index.html> (basics, but important)
- <https://pytorch.org/tutorials/> (includes things like GANs)
- <https://docs.pyro.ai/en/stable/> (probabilistic machine learning framework)
- <https://sbi-dev.github.io/sbi/> (library for simulation-based inference)
- Many git repositories associated with papers
- I would perform a web search on machine learning papers on a physics topic that you are interested in and get inspired by their data and model. Of course you need to simplify your approach substantially compared to a research paper.

Unit 3: Convolutional NN

Introduction

Resources:

- Based on:
 - <https://udlbook.github.io/udlbook/> (Simon Prince - Understanding Deep Learning)
 - Deeplearningbook.com

Application: Image classification

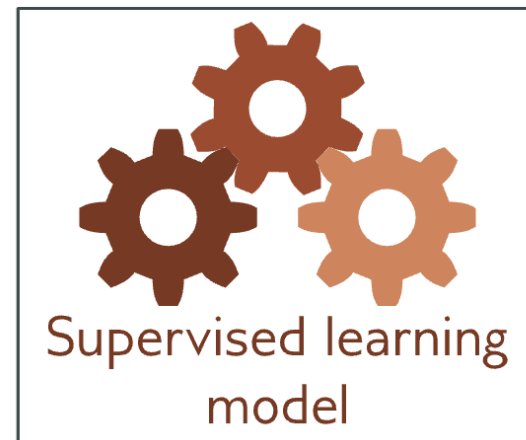
Real world input



Model
input

$\begin{bmatrix} 124 \\ 140 \\ 156 \\ 128 \\ 142 \\ 157 \\ \vdots \end{bmatrix}$

Model



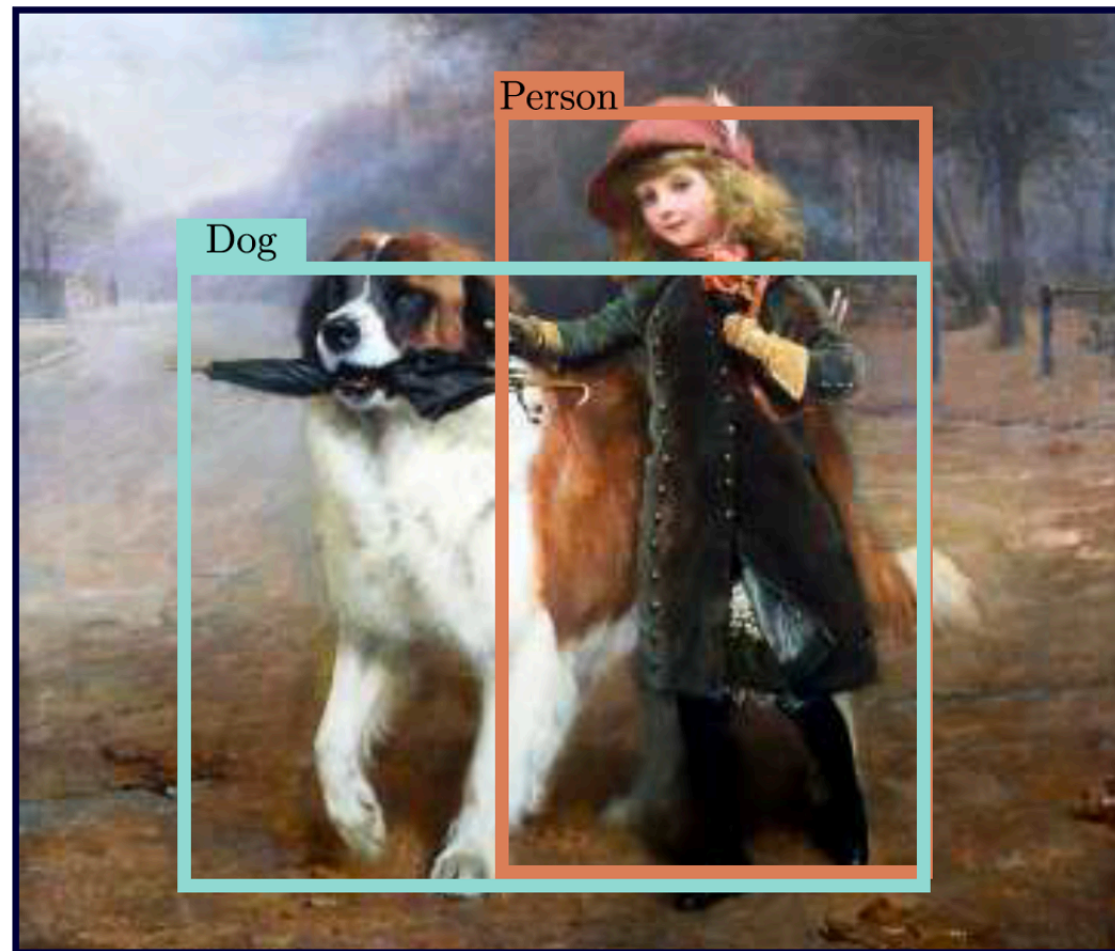
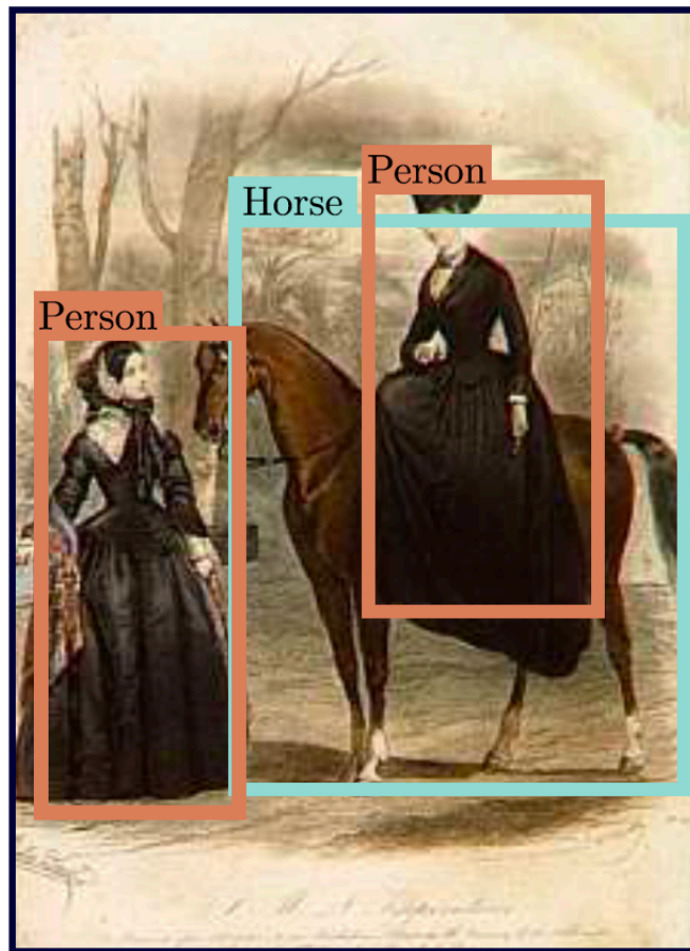
Model
output

$\begin{bmatrix} 0.00 \\ 0.00 \\ 0.01 \\ 0.89 \\ 0.05 \\ 0.00 \\ \vdots \\ 0.01 \end{bmatrix}$

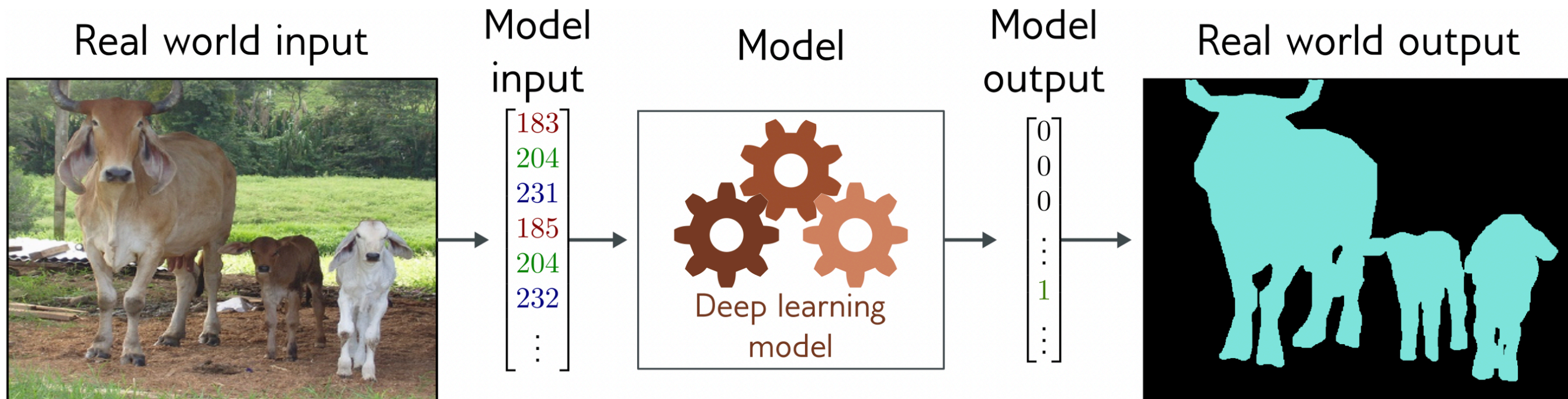
Real world output

Aardvark
Apple
Bee
Bicycle
Bridge
Clown
 \vdots

Application: Object detection



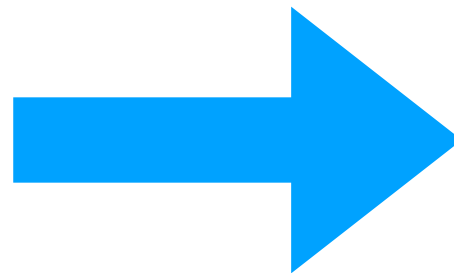
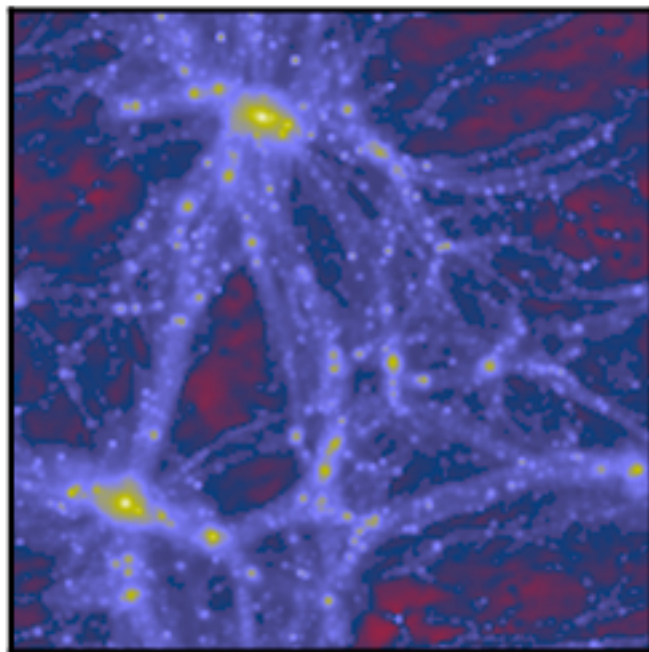
Application: Image segmentation



Application: Field-to-Field translation

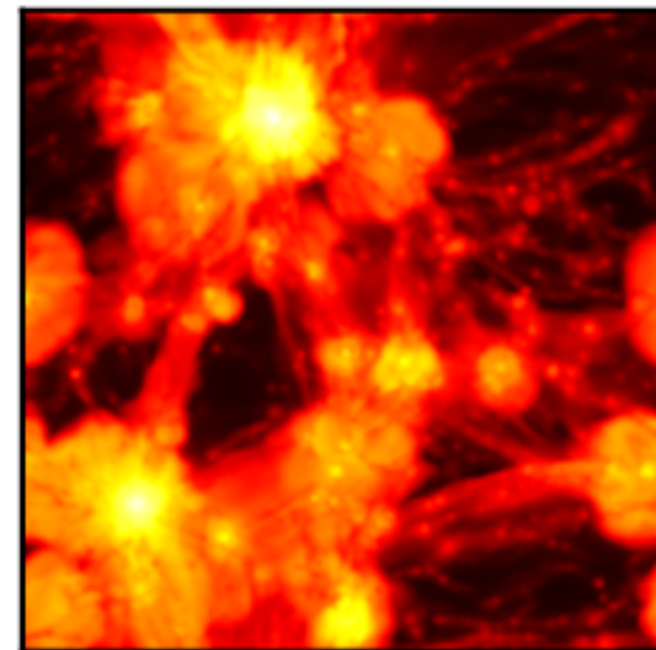
- Example from cosmology

Matter density



CNN

Gas temperature



Why not use an MLP?

- Problems with fully-connected networks

1. Size

- 224x224 RGB image = 150,528 dimensions
- Hidden layers generally larger than inputs
- One hidden layer = 150,520x150,528 weights -- 22 billion

2. Nearby pixels statistically related (there is a notion of distance that the MLP does not include).

3. Should be stable under transformations

- Don't want to re-learn appearance at different parts of image

Convolutional NN

- Convolutional networks (LeCun, 1989), also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology. Examples include time-series data, and image data
- Convolutional networks have been **tremendously successful in practical applications**. The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution.
- Convolution is a specialized kind of linear operation. Convolutional networks are **simply neural networks that use convolution in place of general matrix multiplication** in at least one of their layers.
- Parameters only look at local image patches
- Share parameters across image

Invariance

- A function $f[x]$ is **invariant** to a transformation $t[]$ if:

$$f[t[x]] = f[x]$$

i.e., the function output is the same even after the transformation is applied

e.g., Image classification

- Image has been translated, but we want our classifier to give the same result



Equivariance

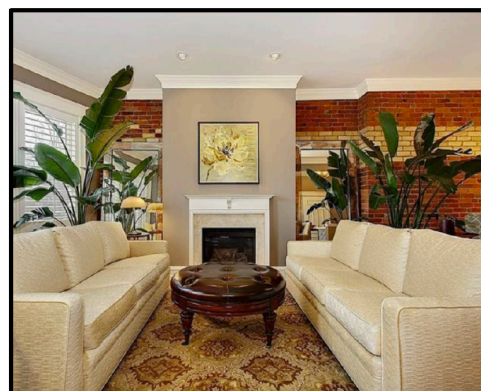
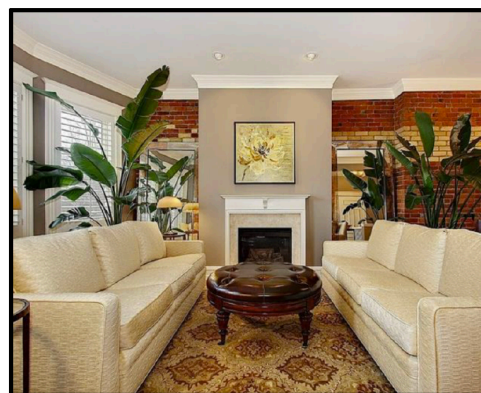
- A function $f[x]$ is **equivariant** to a transformation $t[]$ if:

$$f[t[x]] = t[f[x]]$$

i.e., the output is transformed in the same way as the input

e.g., Image segmentation

- Image has been translated and we want segmentation to translate with it



Convolutional Neural Networks

Convolutions

Convolutions in 1d

- Input vector \mathbf{x} :

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

- Output is weighted sum of neighbors:

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

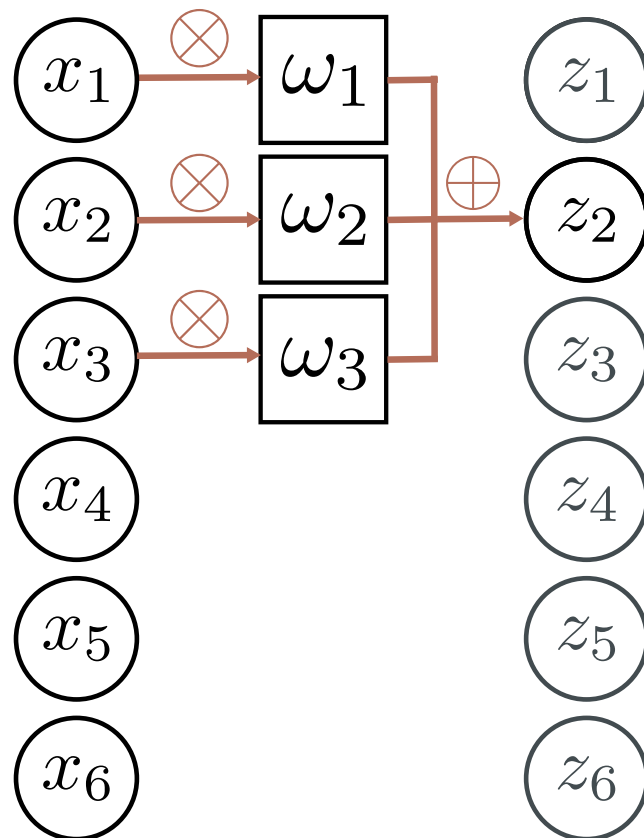
- Convolutional **kernel** or **filter**:

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

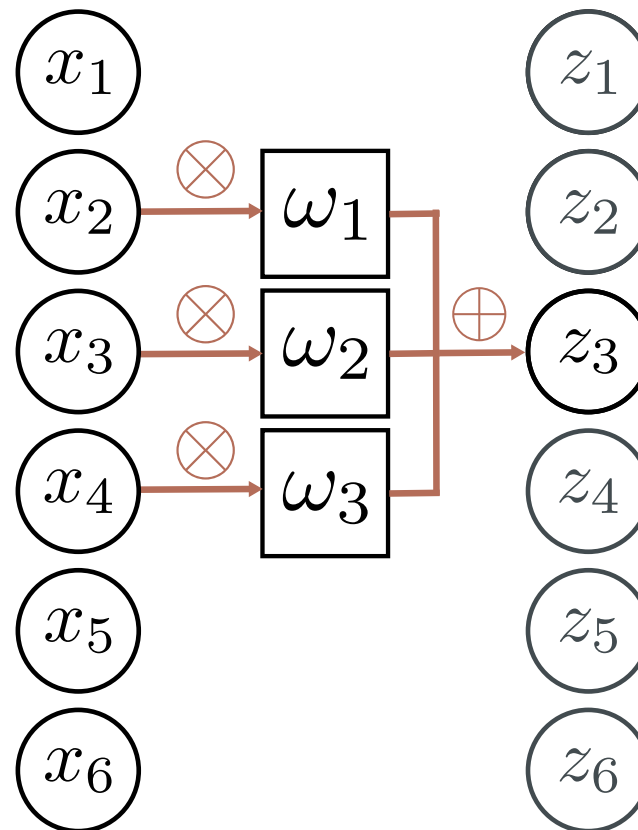
Kernel size = 3

Convolutions in 1d

a)



b)



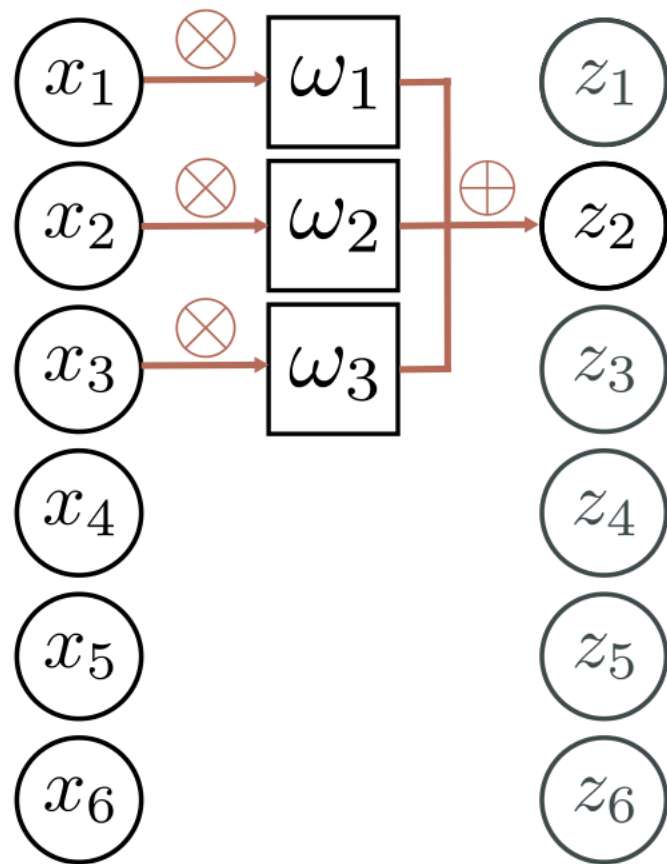
Equivariant to translation of input

$$\mathbf{f}[\mathbf{t}[\mathbf{x}]] = \mathbf{t}[\mathbf{f}[\mathbf{x}]]$$

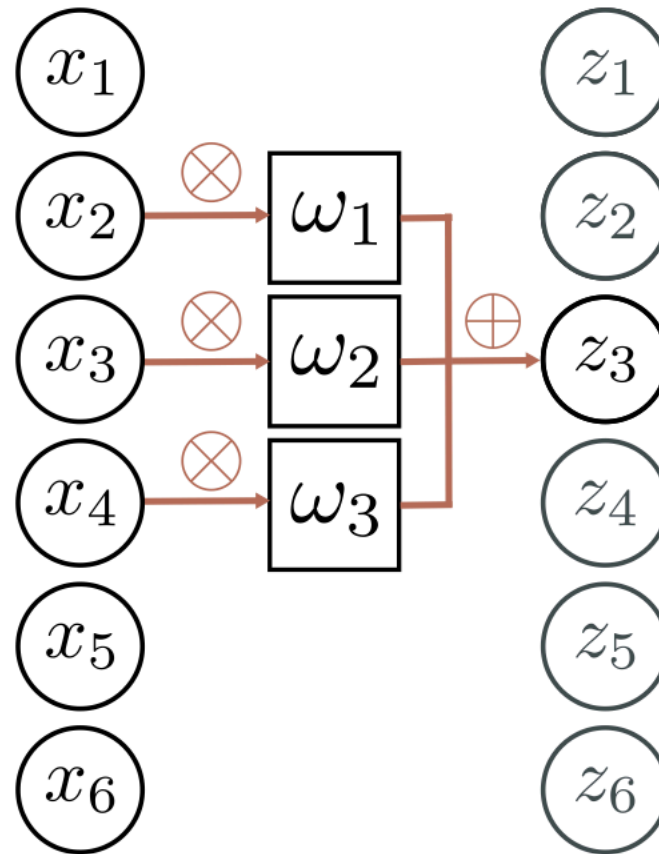
- The convolutional operation adds the weighted inputs
- Plus passes through activation (e.g. ReLU) function

What to do at the boundary? “Zero padding”

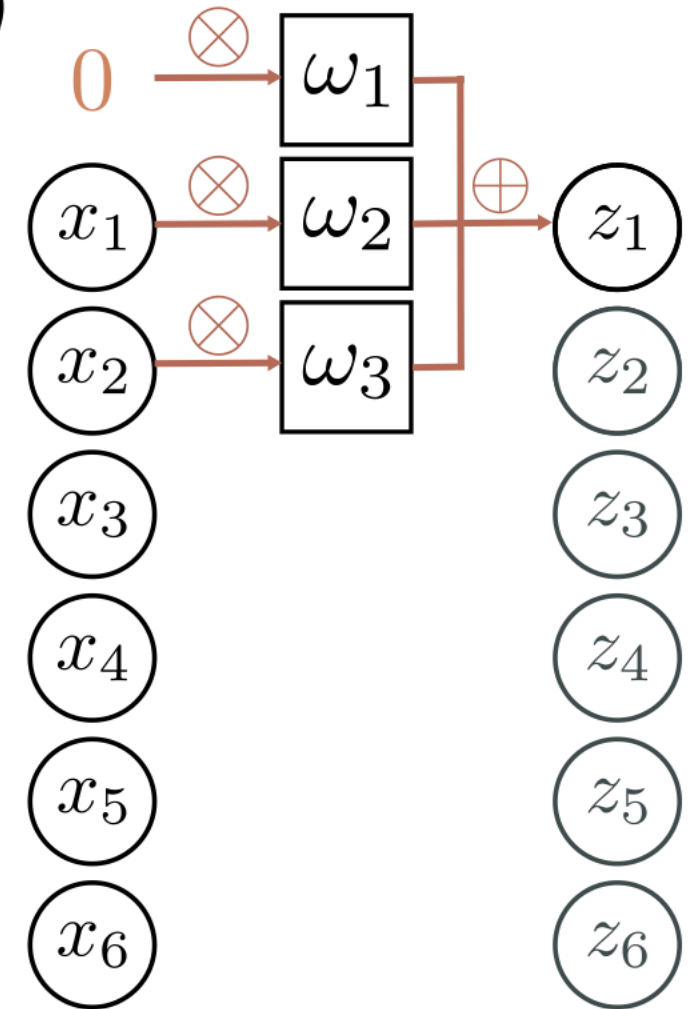
a)



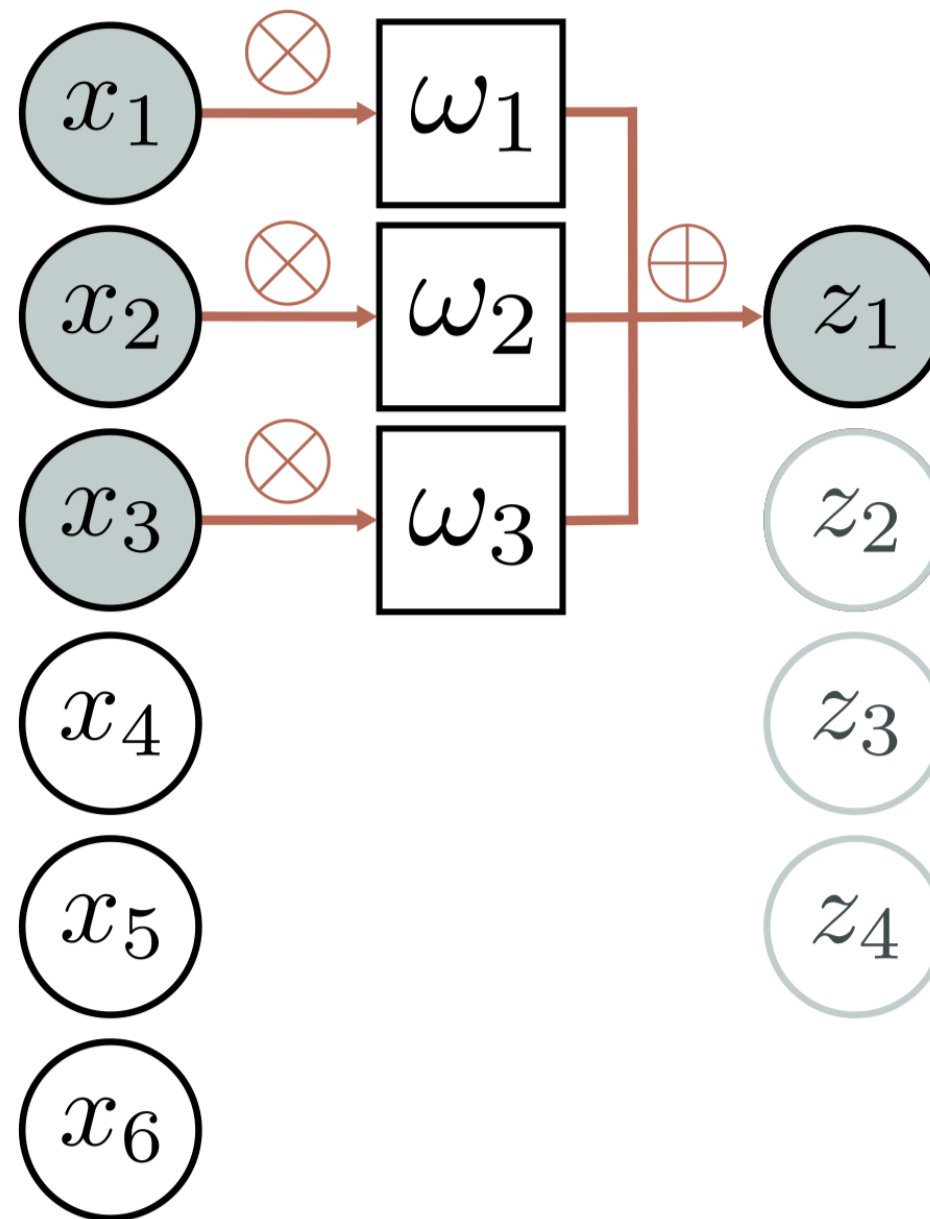
b)



c)



What to do at the boundary? “Valid” convolutions

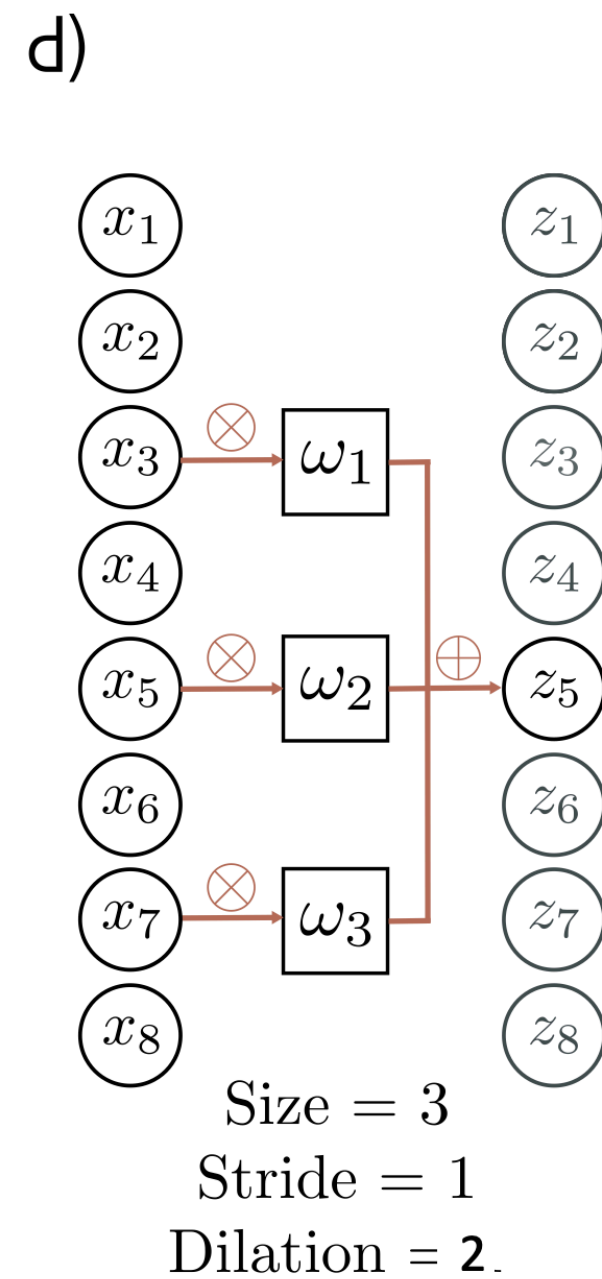
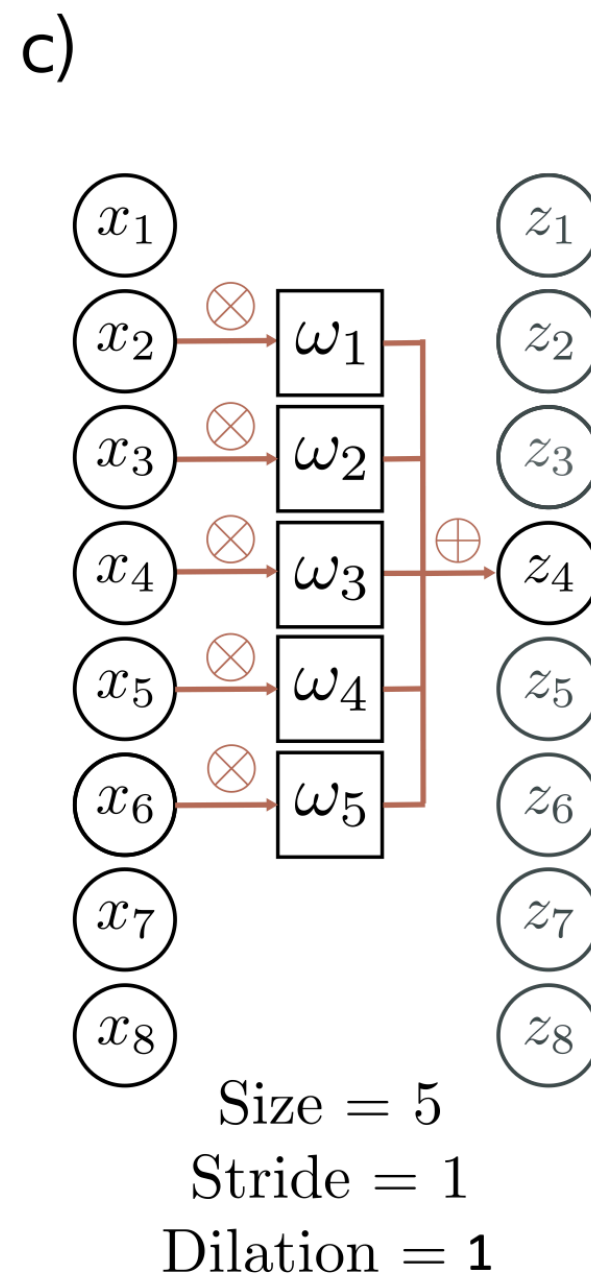
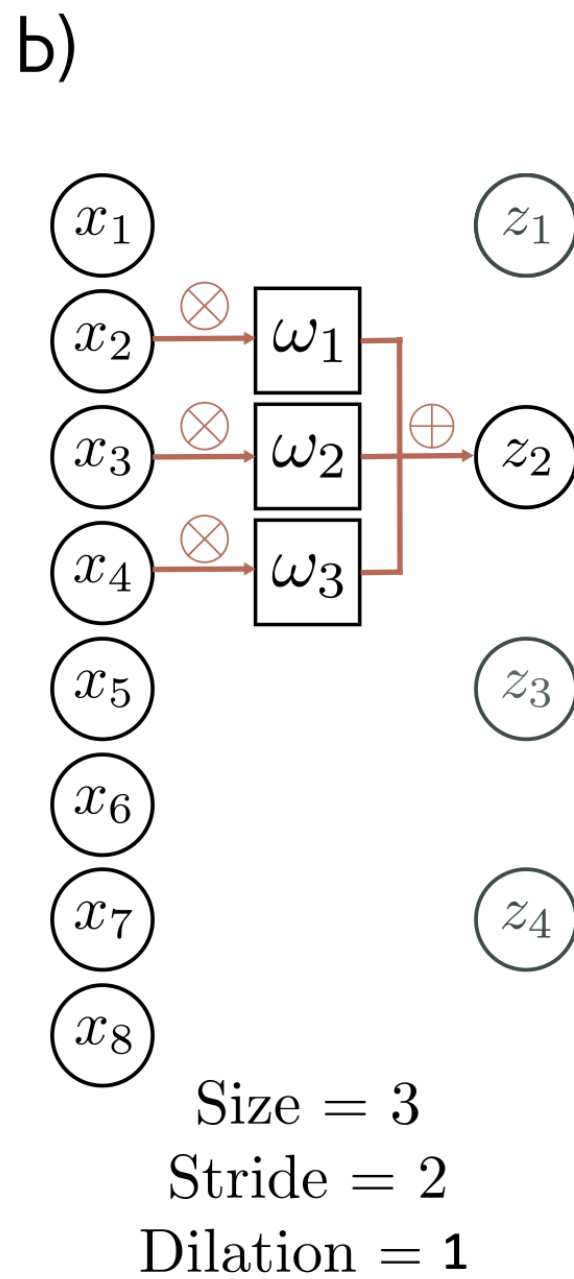
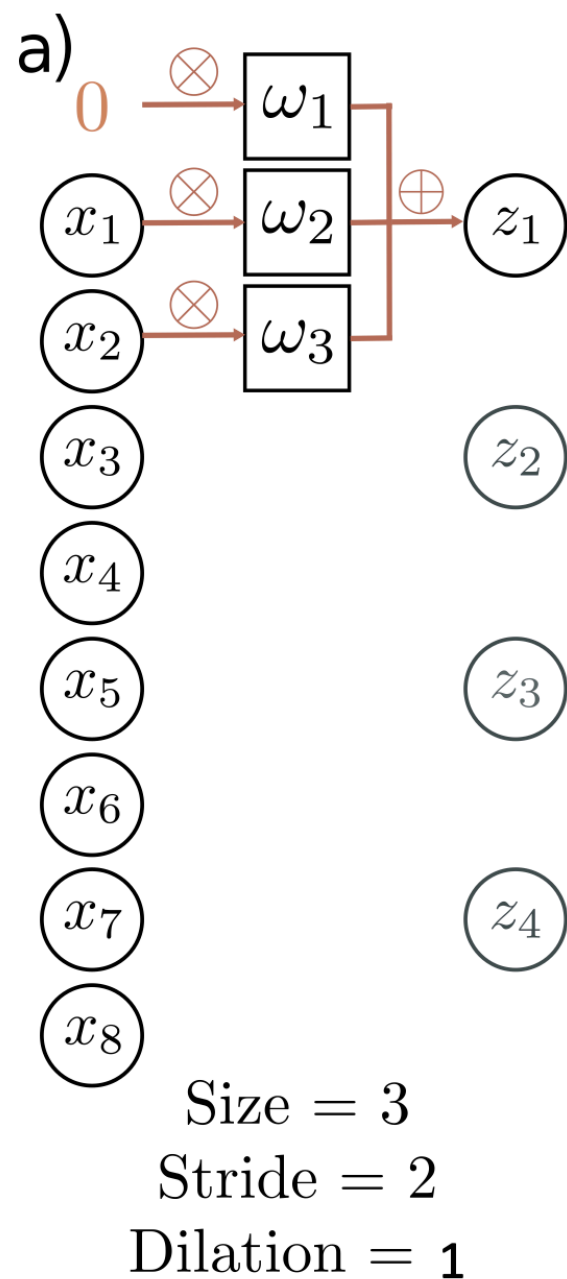


Only process positions where kernel falls in image (smaller output).

Parameters of convolutions

- **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
- **Dilated** or **atrous** convolutions = intersperse kernel values with zeros
 - Combine information from a larger area
 - Fewer parameters

Examples



CNN vs fully connected layer

Convolutional network:

$$h_i = a[\beta + \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}]$$

$$= a\left[\beta + \sum_{j=1}^3 \omega_j x_{i+j-2}\right]$$

3 weights, 1 bias

kernel weights $\omega_1, \omega_2, \omega_3$

bias β

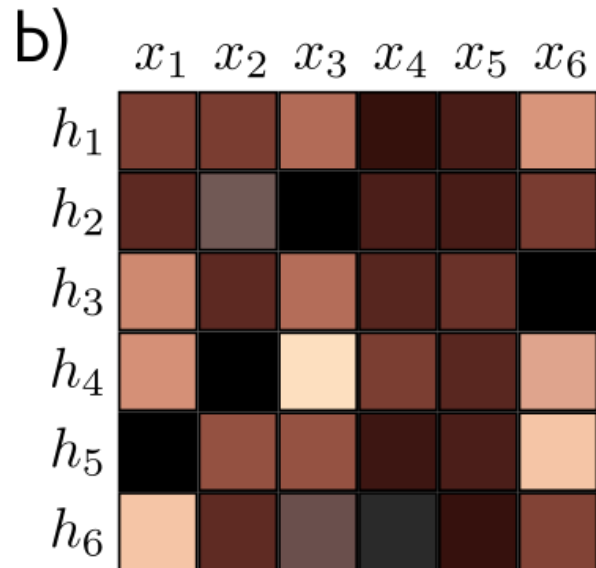
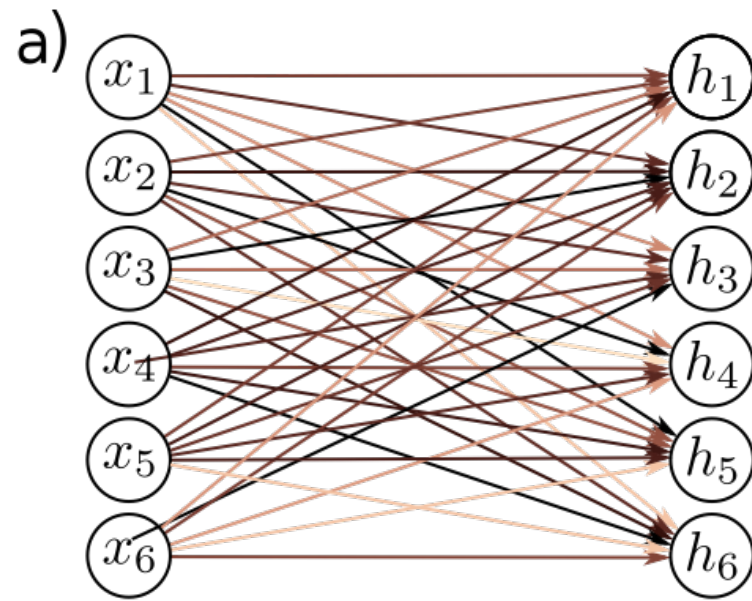
activation function $a[\bullet]$

Fully connected network:

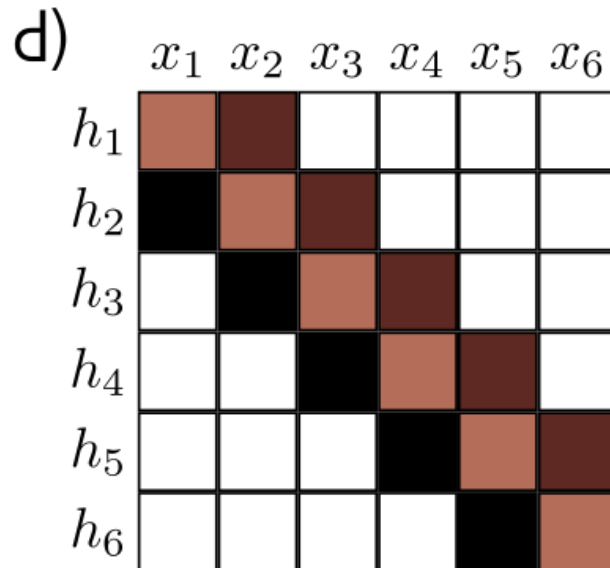
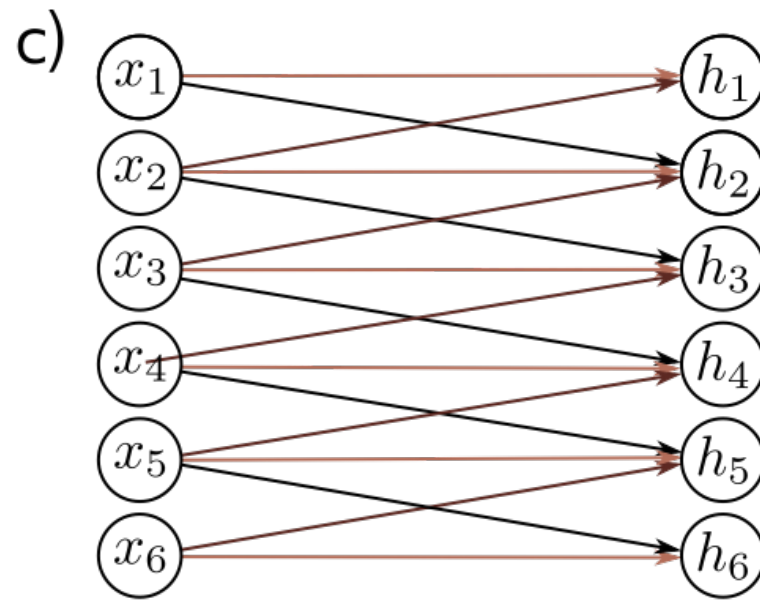
$$h_i = a\left[\beta_i + \sum_{j=1}^D \omega_{ij} x_j\right]$$

D^2 weights, D biases

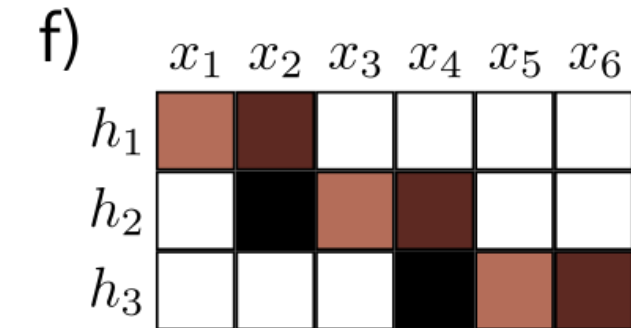
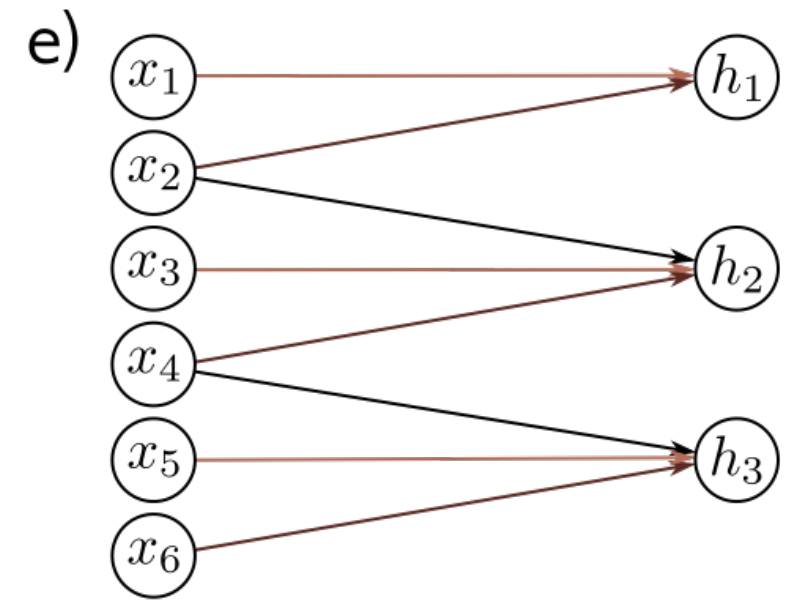
CNN vs fully connected layer



Fully connected network



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

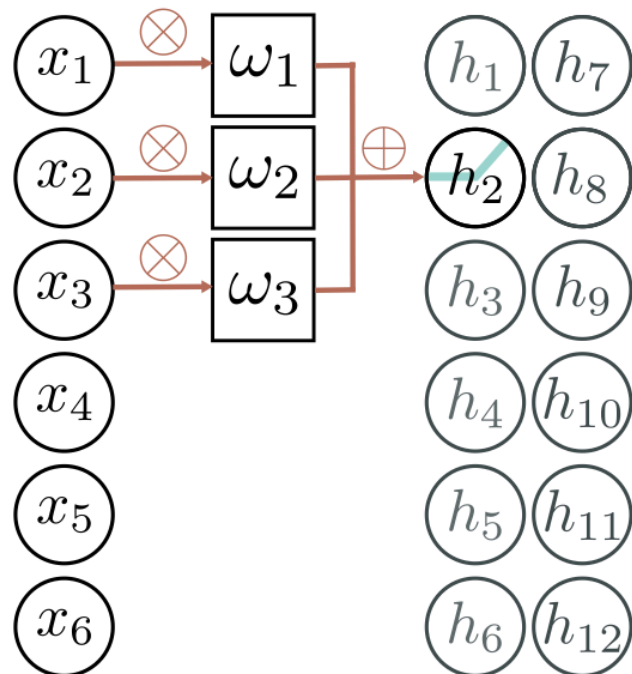


Convolution, size 3, stride 2,
dilation 1, zero padding

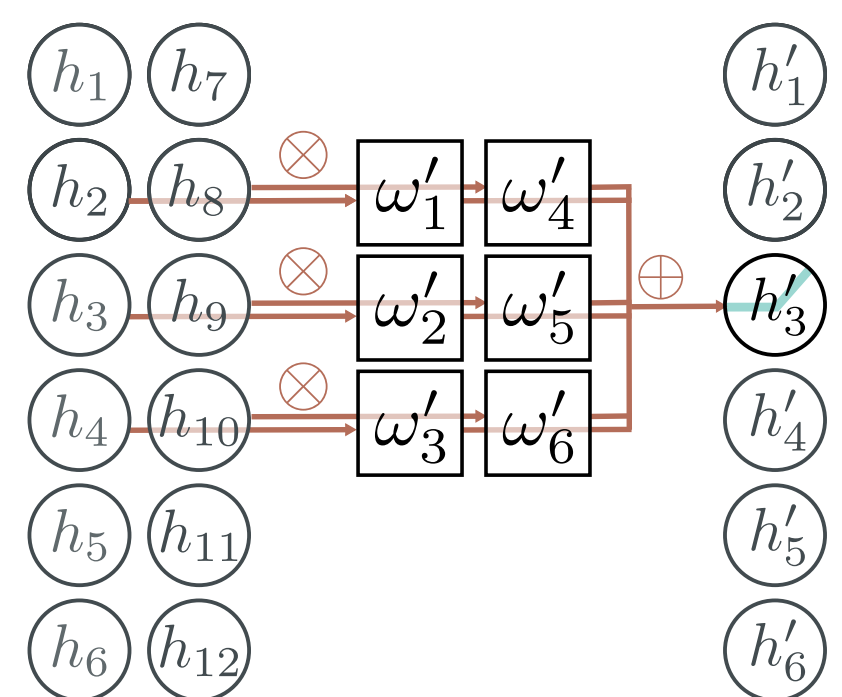
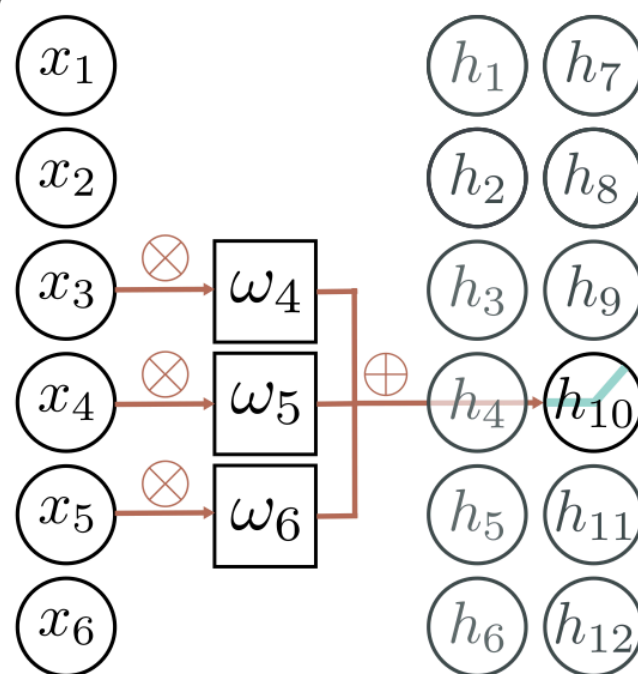
Channels

- The convolutional operation averages together the inputs
- Plus passes through ReLU function
- Has to lose information.
- Solution:
 - apply several convolutions and stack them in **channels**
 - Sometimes also called **feature maps**

a)



b)



How many parameters?

- If there are C_i input channels and kernel size K

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

$$\beta \in \mathbb{R}$$

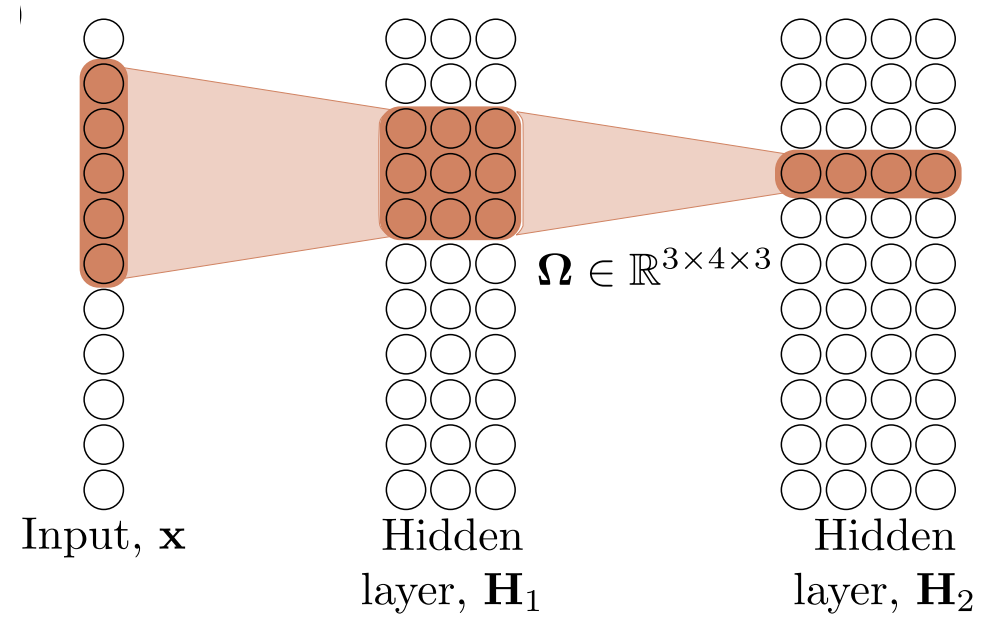
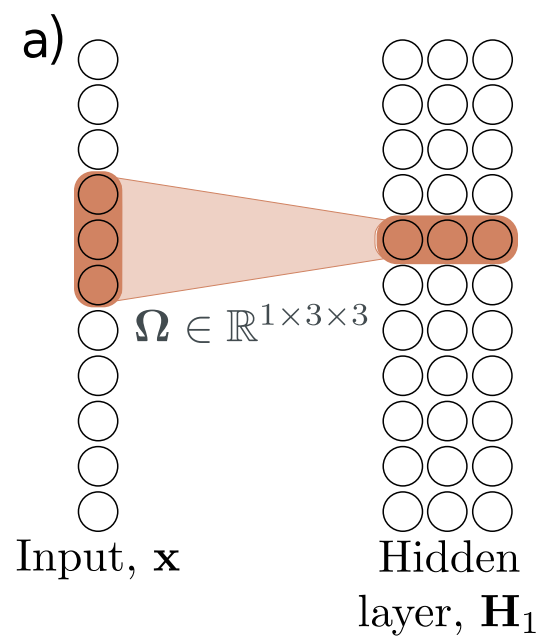
- If there are C_i input channels and C_o output channels

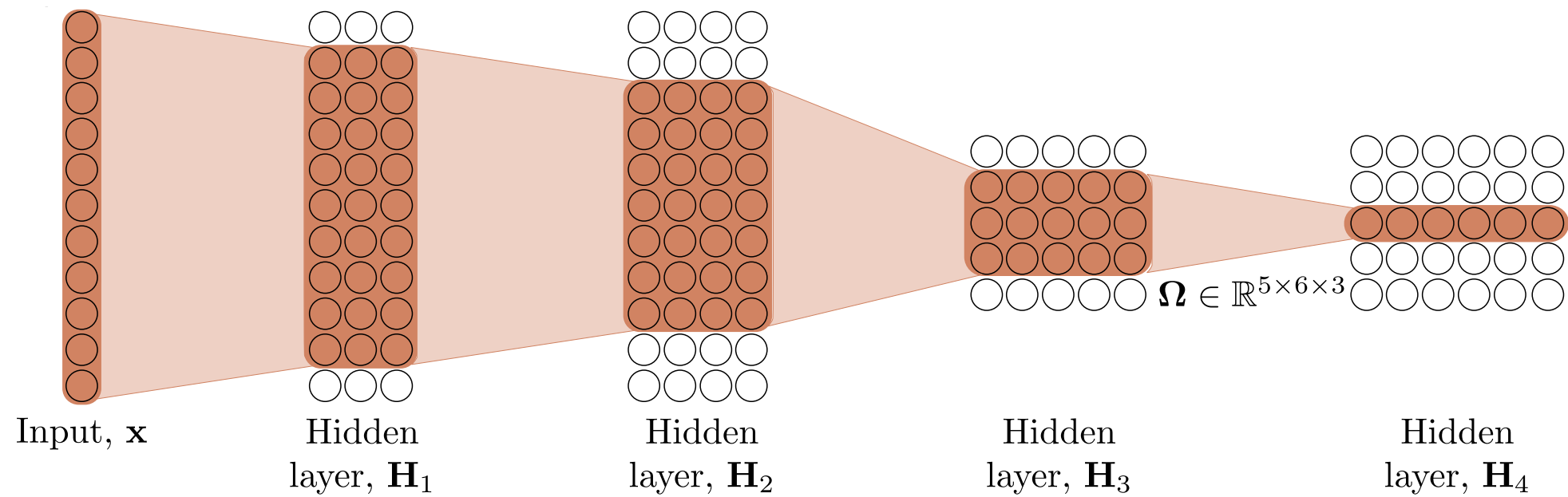
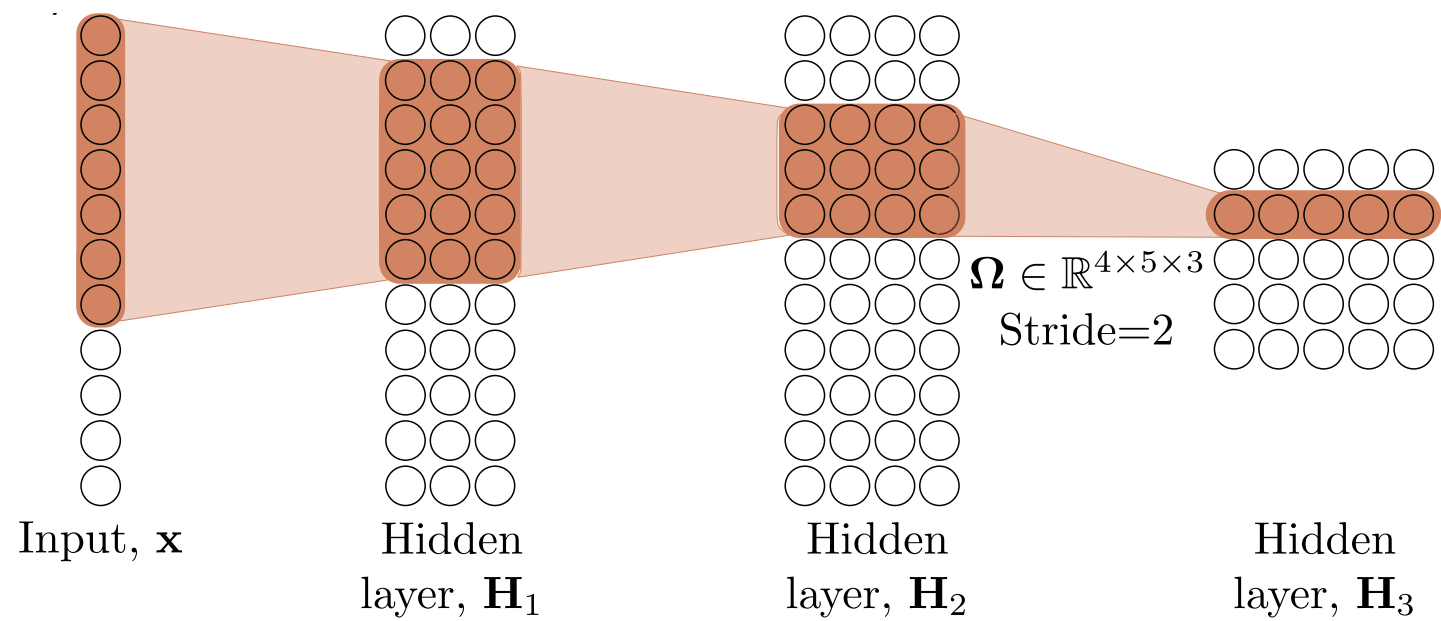
$$\Omega \in \mathbb{R}^{C_i \times C_o \times K}$$

$$\beta \in \mathbb{R}^{C_o}$$

Convolutional Neural Networks

Receptive fields in CNN

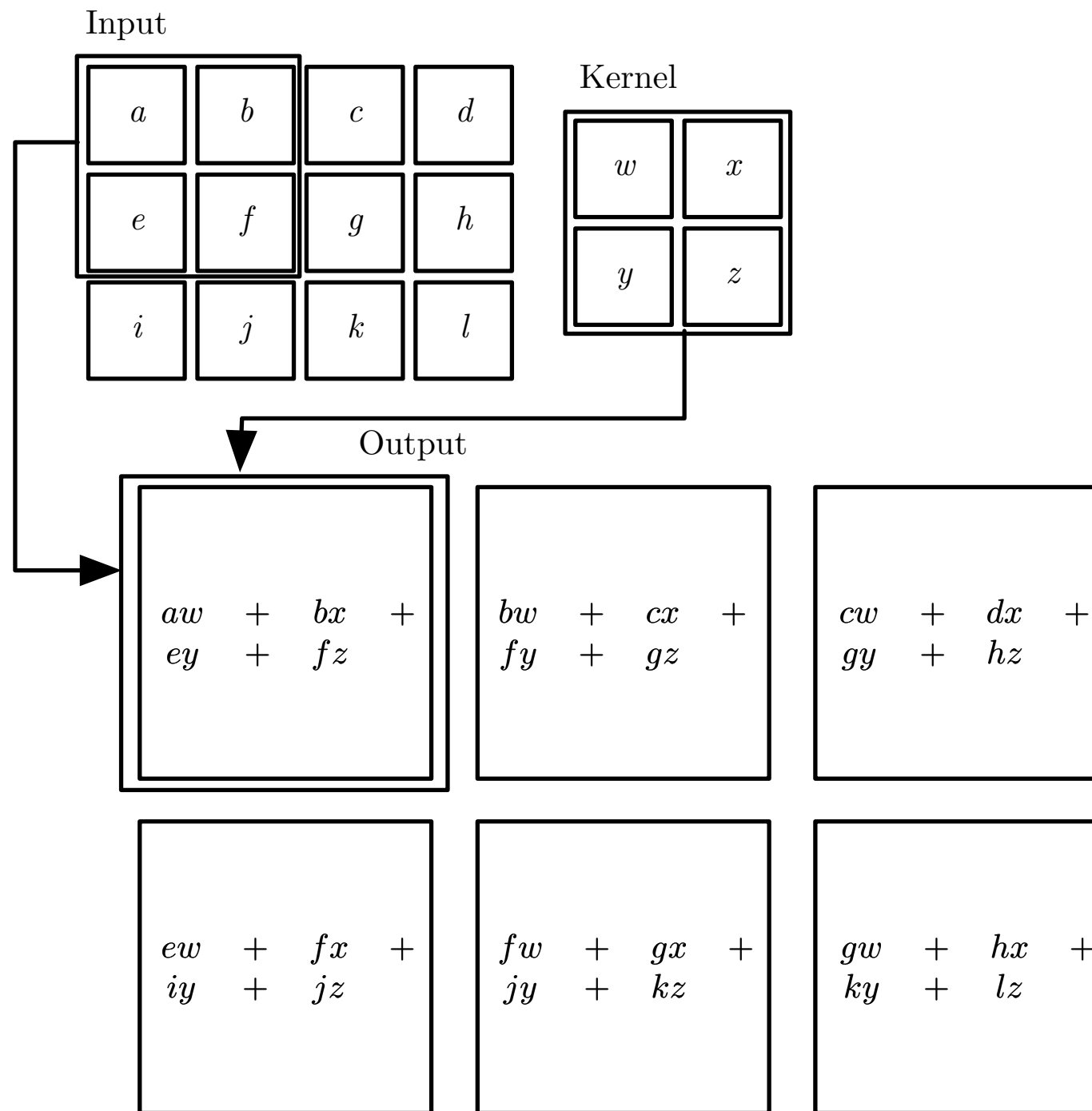




Convolutional Neural Networks

2d Convolutions

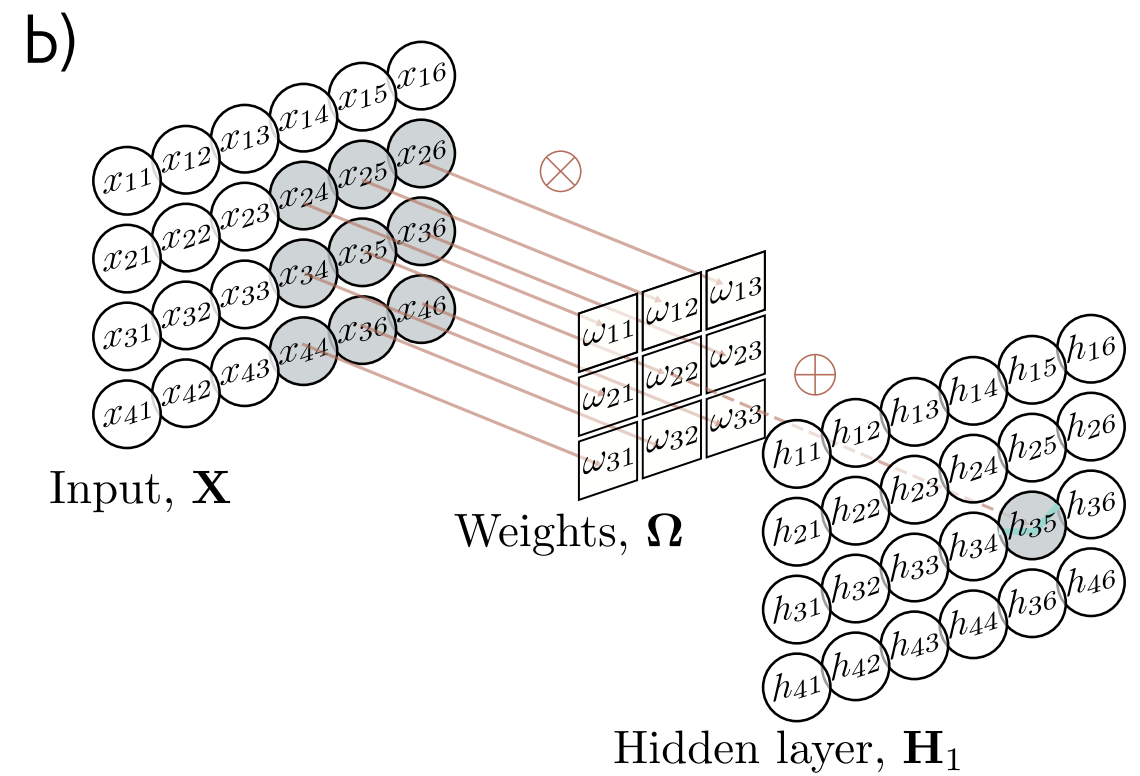
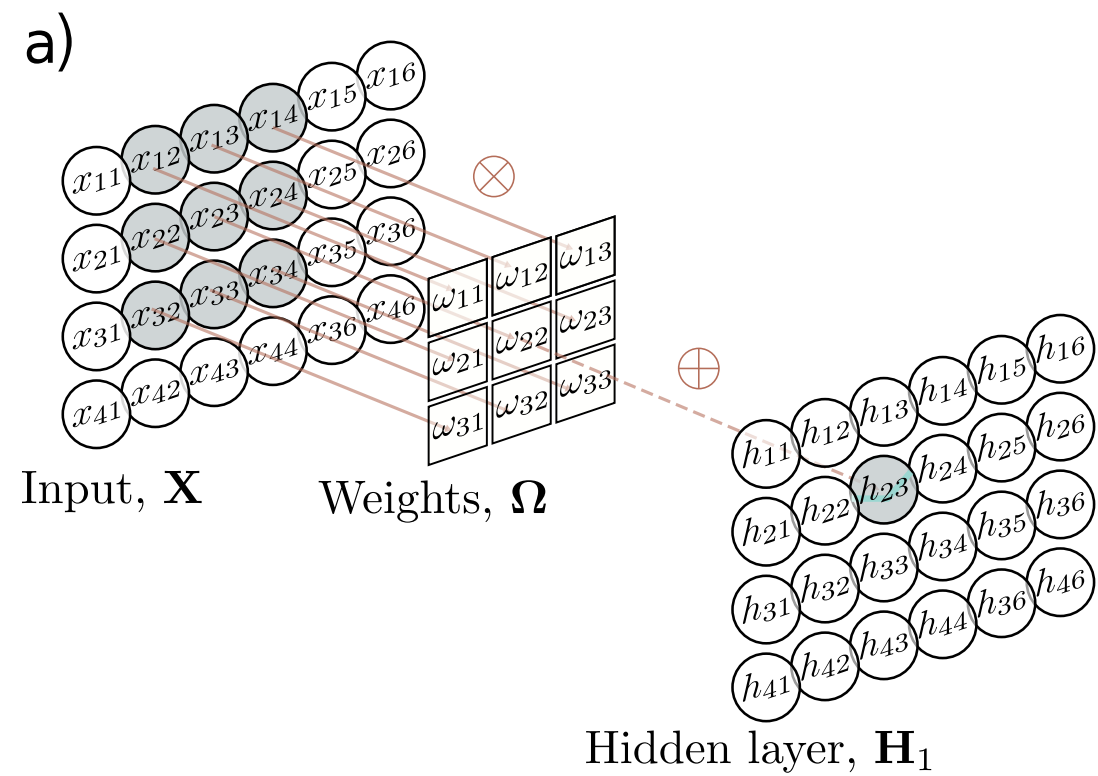
2d Convolution

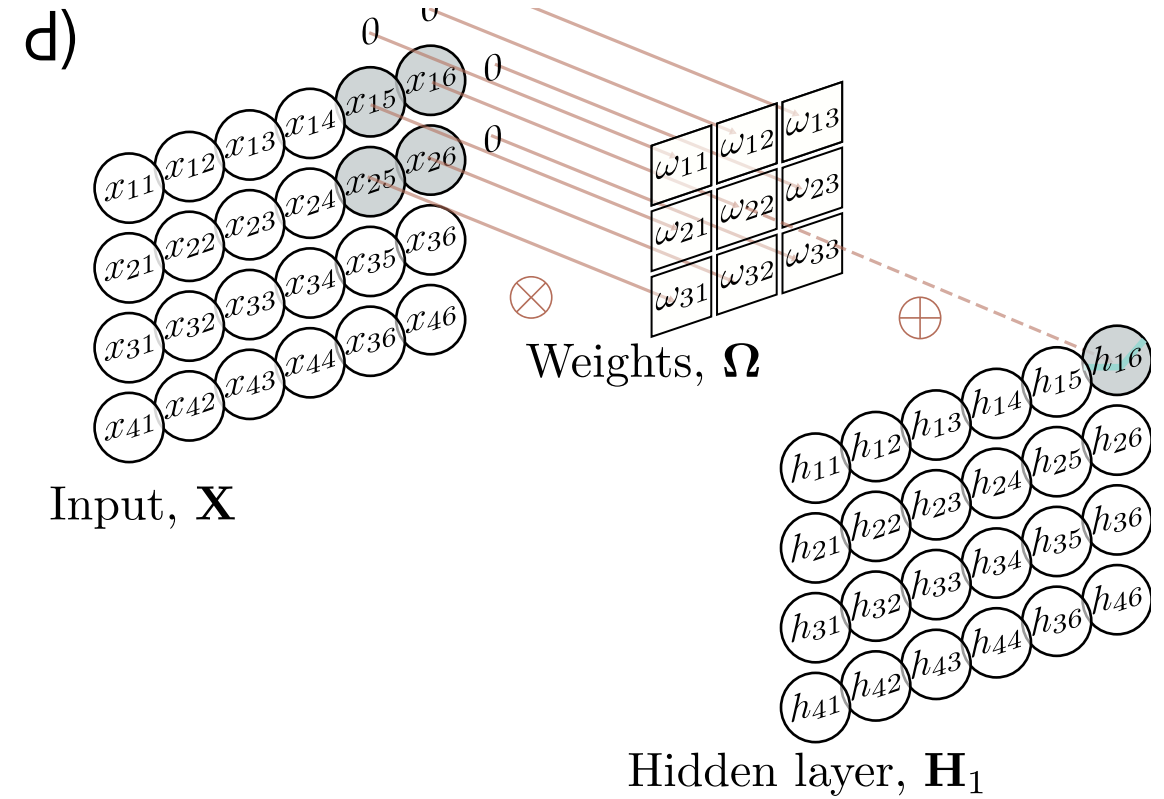
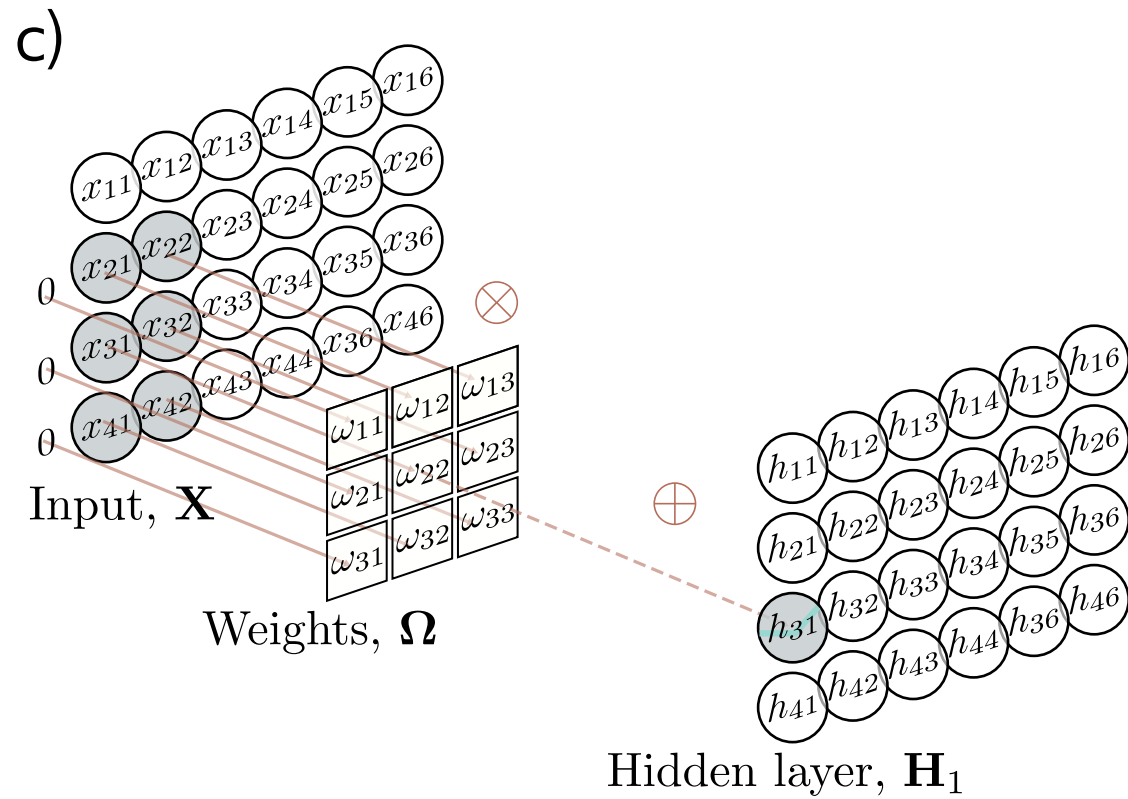


2D Convolution

- Convolution in 2D
 - Weighted sum over a $K \times K$ region
 - $K \times K$ weights
- Build into a convolutional layer by adding bias and passing through activation function
- Example for a 2d convolutional layer with 3×3 kernel K :

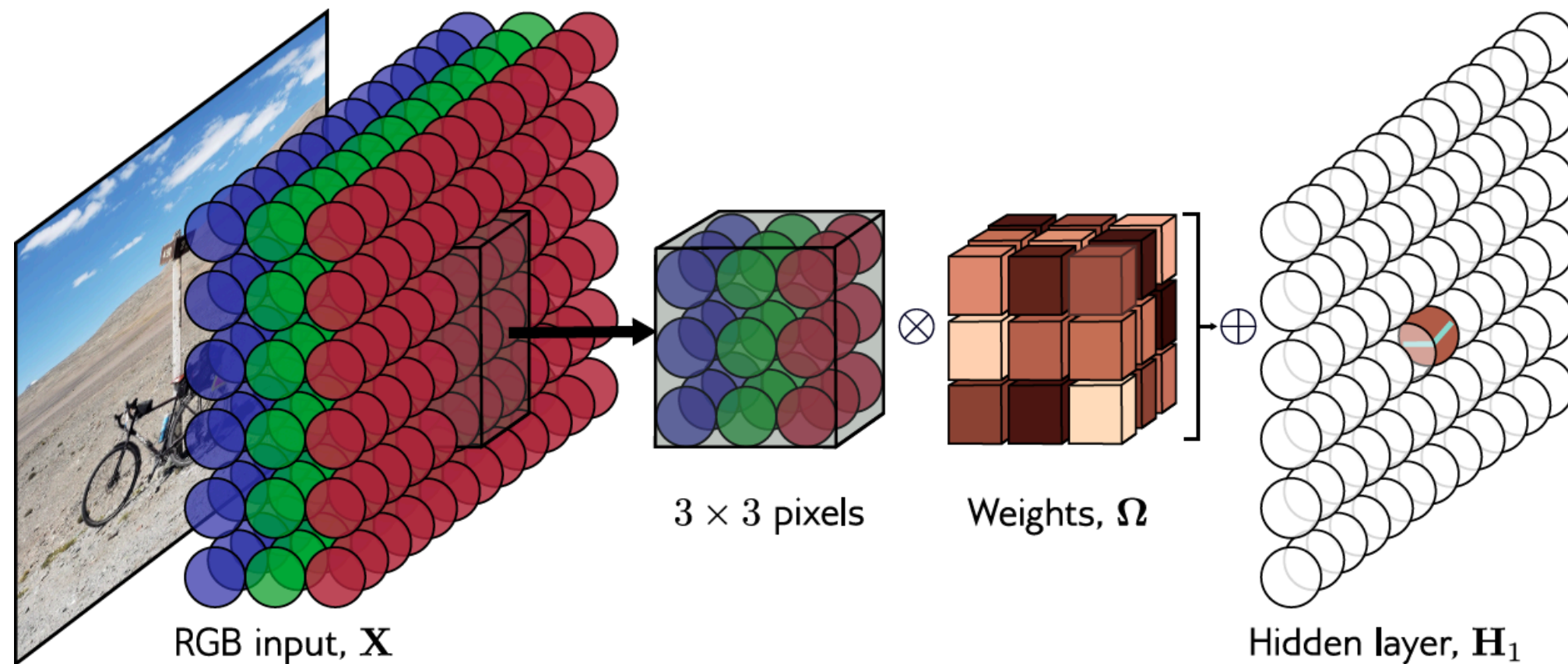
$$h_{i,j} = a \left[\beta + \sum_{m=1}^3 \sum_{n=1}^3 \omega_{m,n} x_{i+m-2,j+n-2} \right]$$





With zero-padding, positions beyond the image's edge are considered to be zero.

Channels in 2D convolutions: Example of RGB image



2D convolution applied to an image. The image is treated as a 2D input with three channels corresponding to the red, green, and blue components. With a 3×3 kernel, each pre-activation in the first hidden layer is computed by pointwise multiplying the $3 \times 3 \times 3$ kernel weights with the 3×3 RGB image patch centered at the same position, summing, and adding the bias. To calculate all the pre-activations in the hidden layer, we “slide” the kernel over the image in both horizontal and vertical directions. The output is a 2D layer of hidden units. To create multiple output channels, we would repeat this process with multiple kernels, resulting in a 3D tensor of hidden units at hidden layer H_1 .

How many parameters?

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

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

- If there are C_i input channels and C_o output channels

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

Guide to different types of 2d convolutions

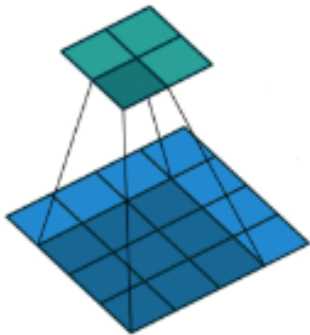
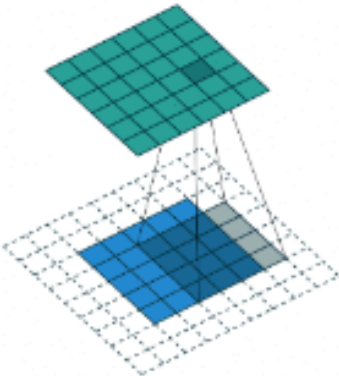
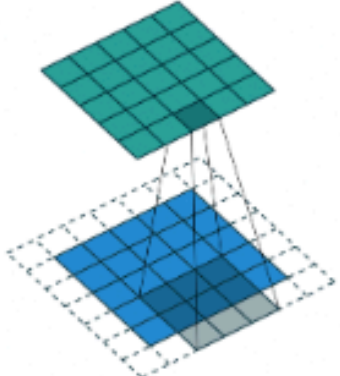
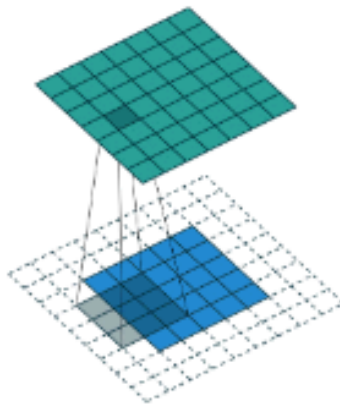
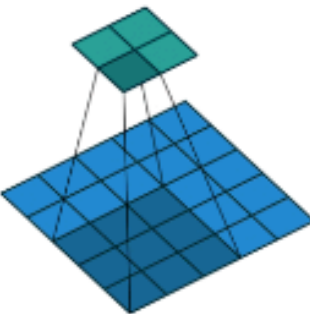
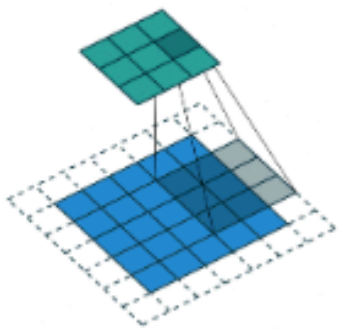
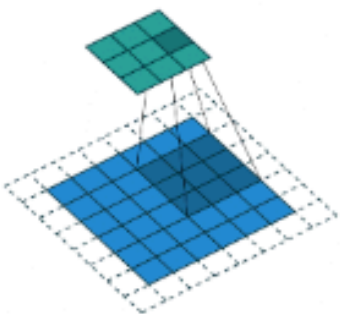
https://github.com/vdumoulin/conv_arithmetic

Famous animated guide

Kernel size, stride, dilation all work as you would expect from the 1d case.

Convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

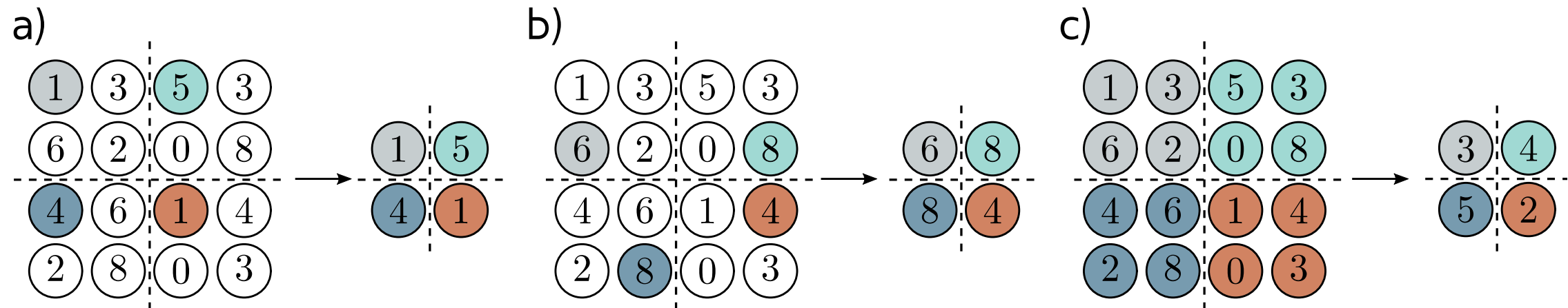
Convolutional Neural Networks

Downsampling and Upsampling

Changing tensor resolution

- CNNs often include steps that can **scale the dimension of the tensor (e.g. the 2d image) up or down**.
- **Downsampling** is useful for example for image classification, where we want the stream of information to converge to a single label.
- **Upsampling** is useful for image-to-image learning.
- There is also a method to **change the number of channels**, which is useful to reduce computation load.

Downsampling

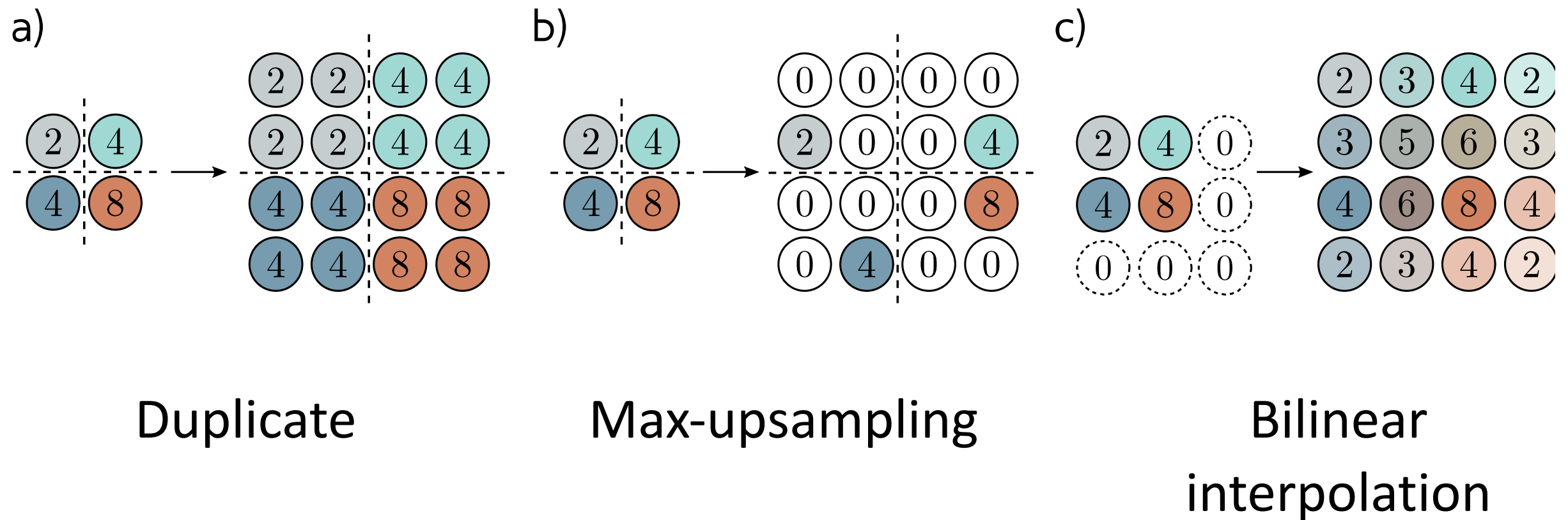


Sample every
other position
(equivalent to
stride two)

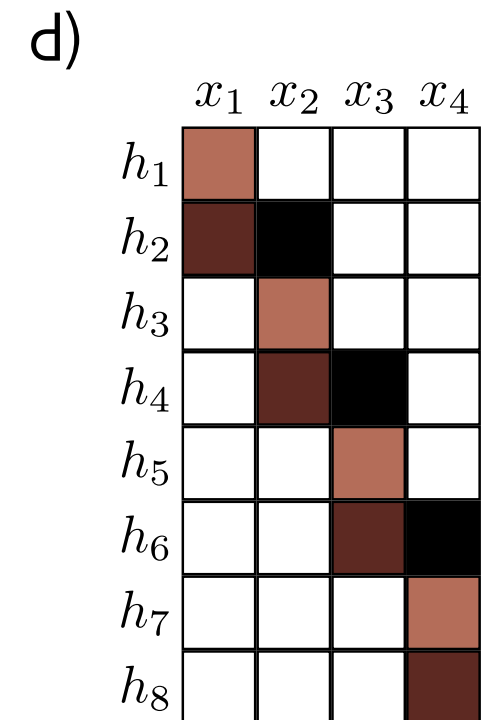
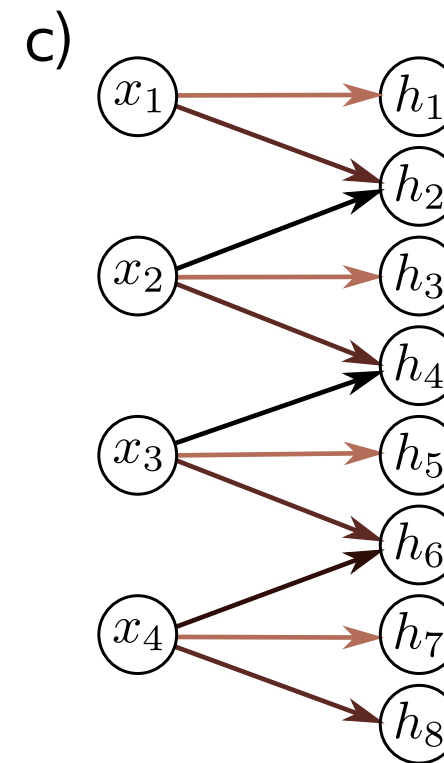
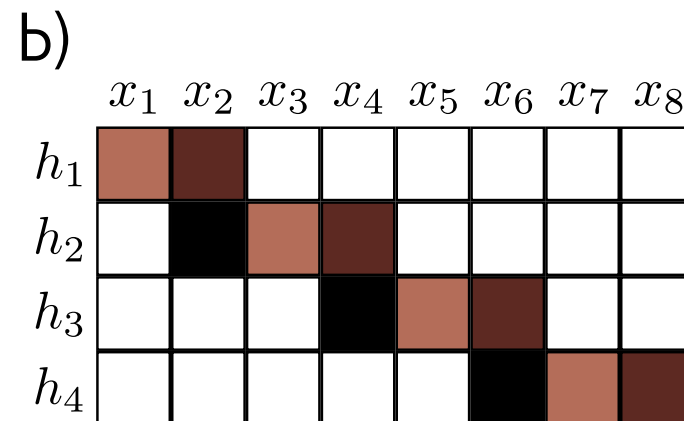
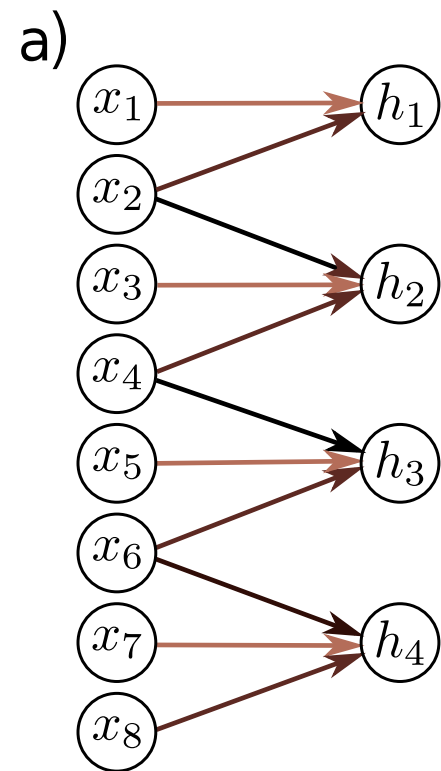
Max pooling
(partial
invariance to
translation)

Mean pooling

Upsampling



Transposed convolutions

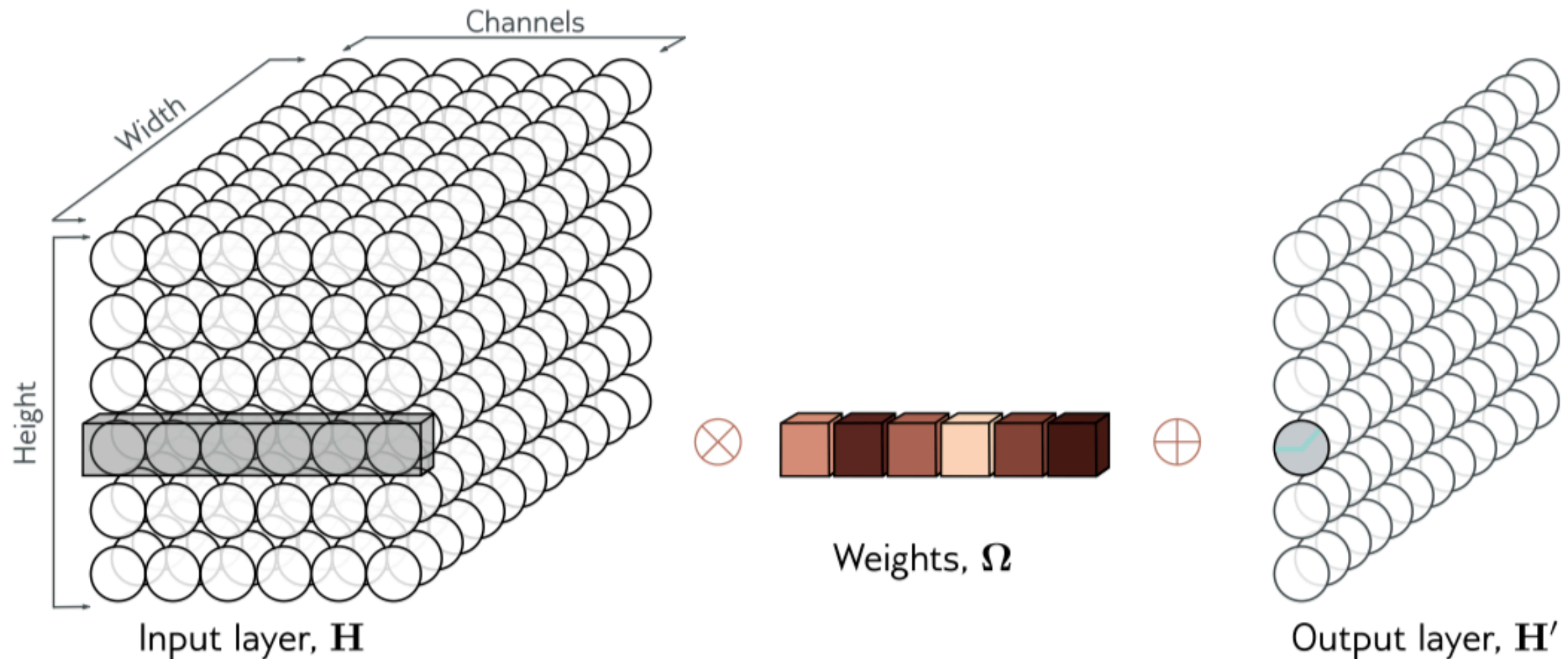


Kernel size 3, Stride 2 convolution

Transposed convolution

Transposed convolution in 1D. a) Downsampling with kernel size three, stride two, and zero-padding. Each output is a weighted sum of three inputs (arrows indicate weights). b) This can be expressed by a weight matrix (same color indicates shared weight). c) In transposed convolution, each input contributes three values to the output layer, which has twice as many outputs as inputs. d) The associated weight matrix is the transpose of that in panel (b).

1x1 convolution to change channel number

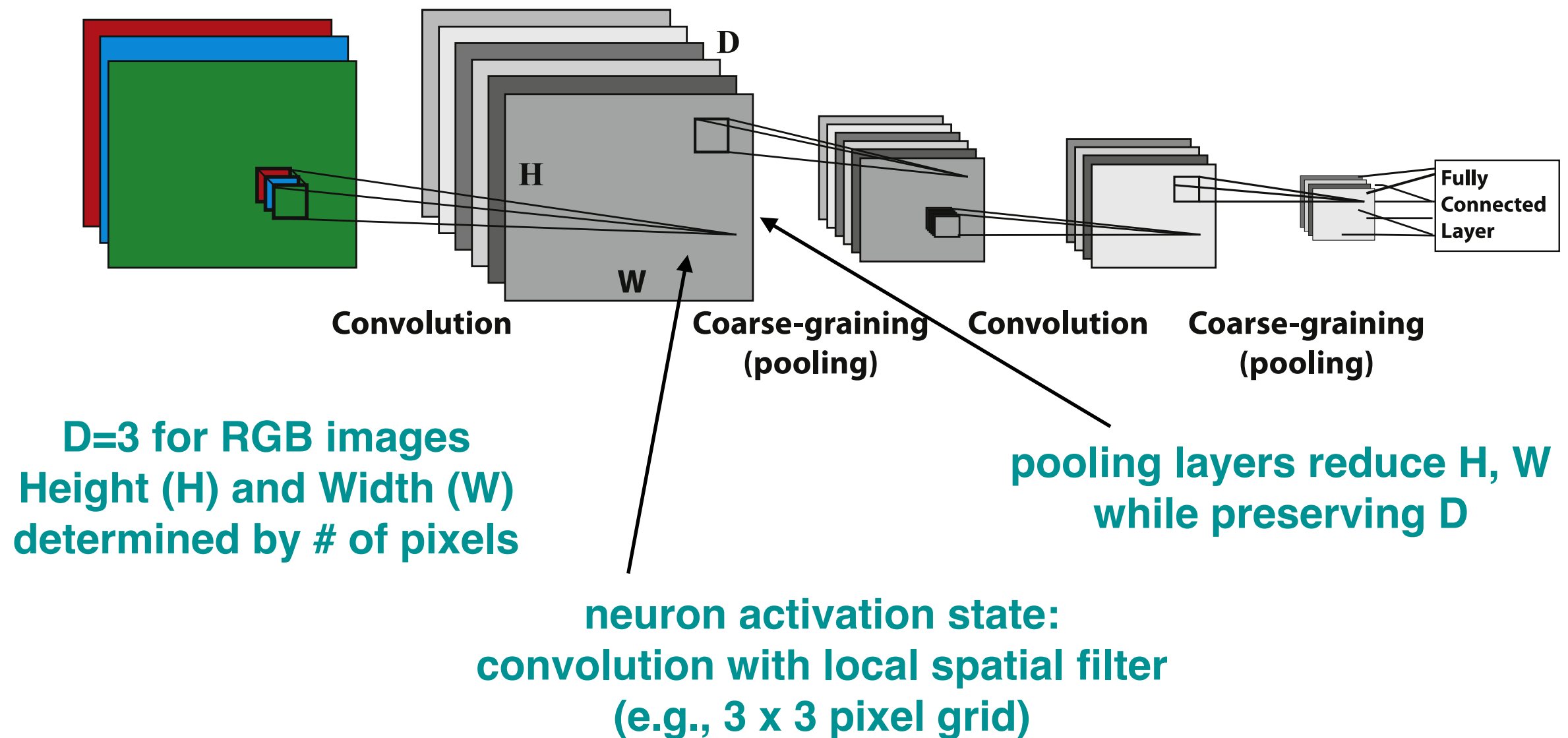


- Mixes channels
- Can change number of channels
- Equivalent to running same fully connected network at each position

Convolutional Neural Networks

Application to image
classification

Simple example architecture

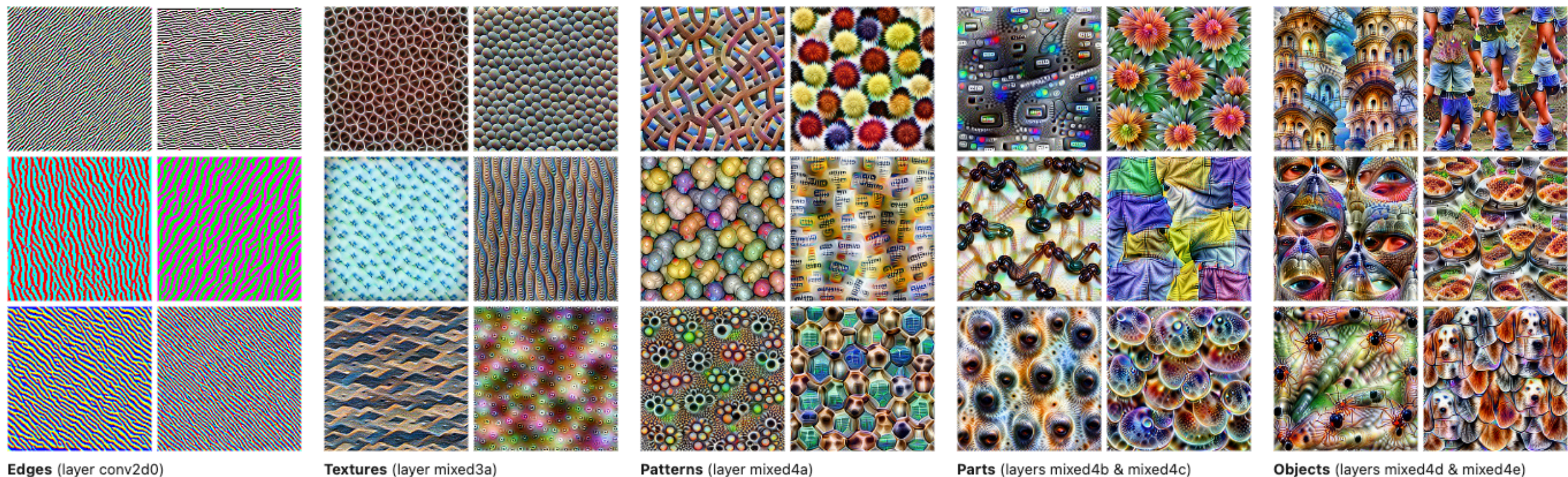


Stacking many layers

The intuition is that lower level CNN layers are sensitive to small simple features such as edges, and higher level layers become sensitive to progressively more abstract features.

Feature Visualization

How neural networks build up their understanding of images



Feature visualization allows us to see how GoogLeNet [1], trained on the ImageNet [2] dataset, builds up its understanding of images over many layers. Visualizations of all channels are available in the [appendix](#).

<https://distill.pub/2017/feature-visualization/>

Classic Benchmark Datasets

- **MNIST** database: images of digits
- **ImageNet** challenge: Much of the pioneering work on deep learning in computer vision focused on image classification using the ImageNet dataset (figure 10.15). This contains 1,281,167 training images, 50,000 validation images, and 100,000 test images, and every image is labeled as belonging to one of 1000 possible categories.
- Performance of algorithms measured on benchmark datasets.

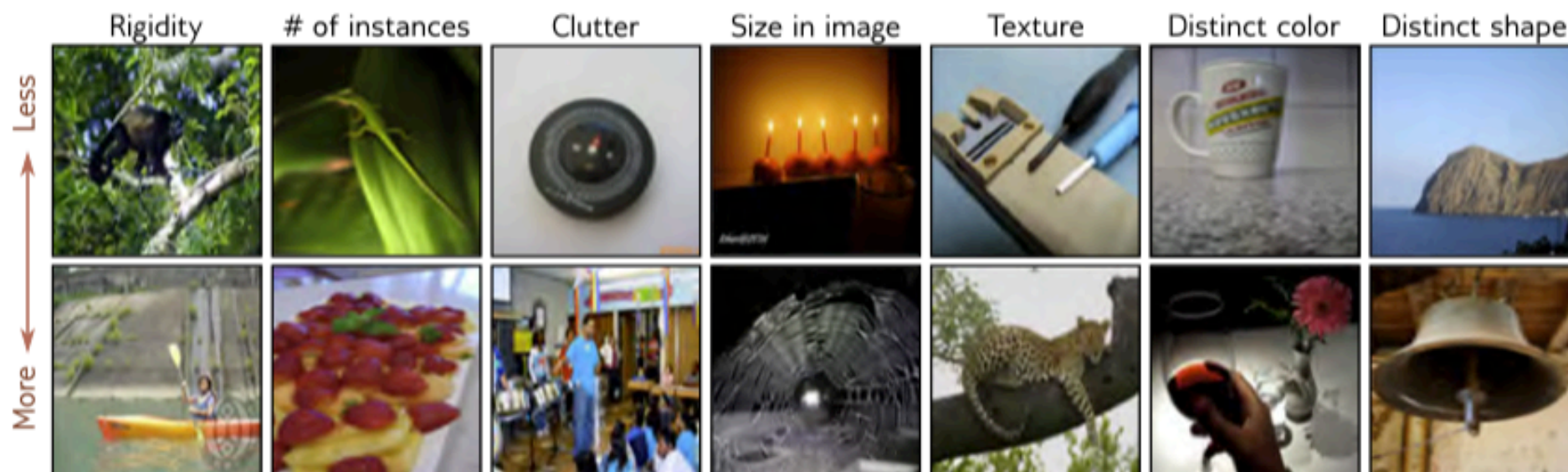
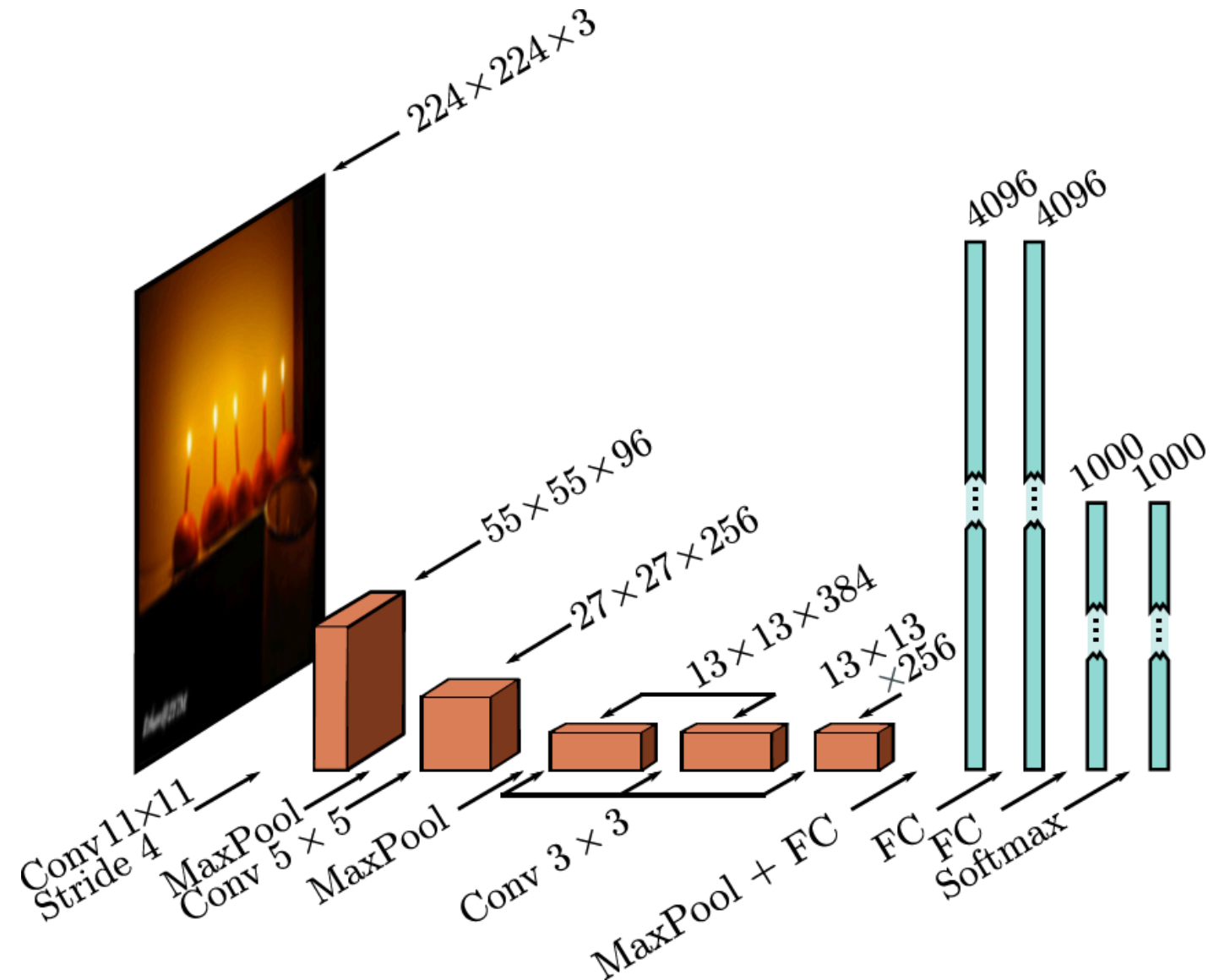


Figure 10.15 Example ImageNet classification images. The model aims to assign an input image to one of 1000 classes. This task is challenging because the images vary widely along different attributes (columns). These include rigidity (monkey < canoe), number of instances in image (lizard < strawberry), clutter (compass < steel drum), size (candle < spiderweb), texture (screwdriver < leopard), distinctiveness of color (mug < red wine), and distinctiveness of shape (headland < bell). Adapted from Russakovsky et al. (2015).

AlexNet (2012)

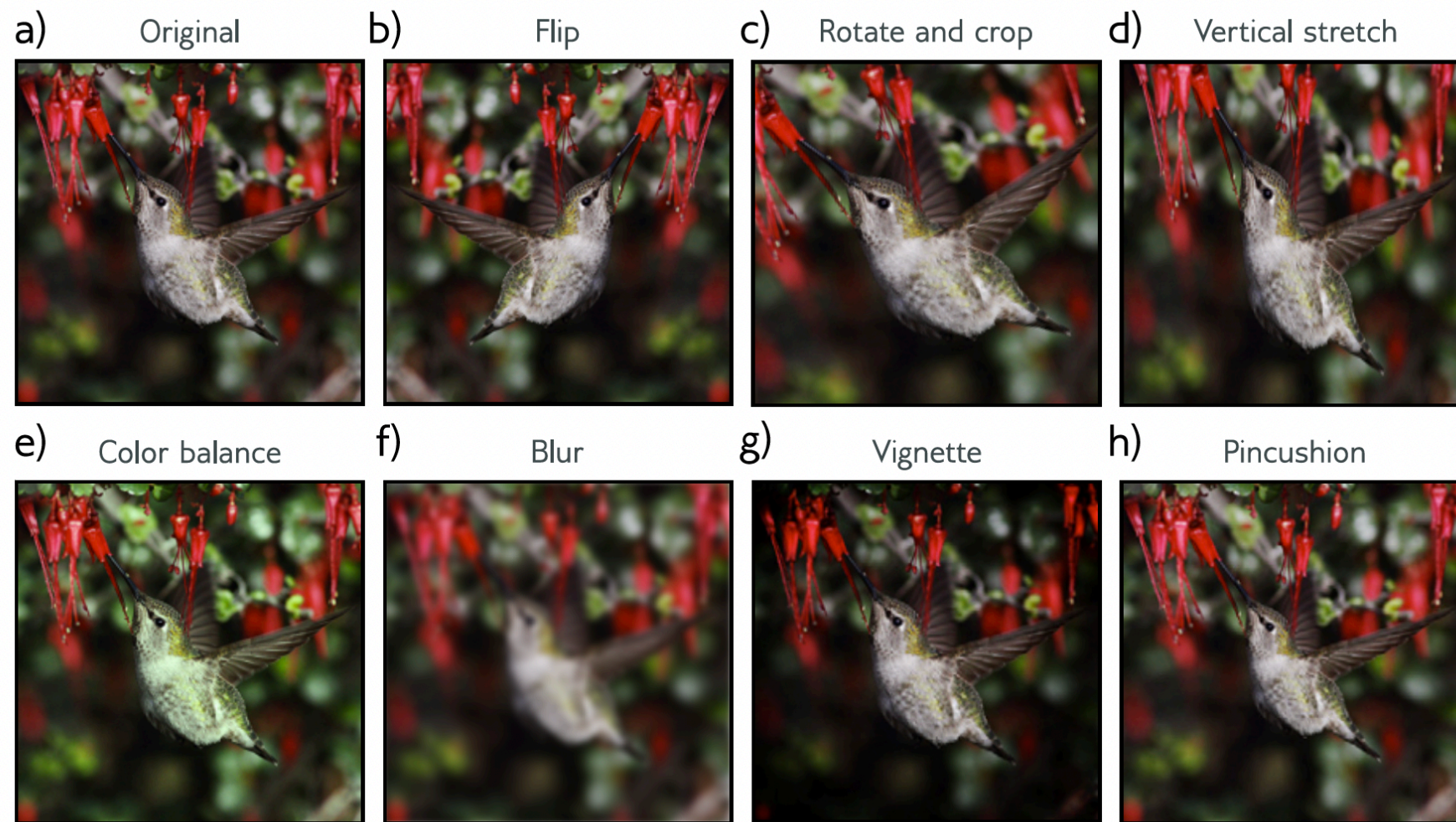
Figure 10.16 AlexNet (Krizhevsky et al., 2012). The network maps a 224×224 color image to a 1000-dimensional vector representing class probabilities. The network first convolves with 11×11 kernels and stride 4 to create 96 channels. It decreases the resolution again using a max pool operation and applies a 5×5 convolutional layer. Another max pooling layer follows, and three 3×3 convolutional layers are applied. After a final max pooling operation, the result is vectorized and passed through three fully connected (FC) layers and finally the softmax layer.



This system achieved a 16.4% top-5 error rate (proportion of times the correct label is not within the model's top 5 predicted classes) and a 38.1% top-1 error rate. At the time, this was an enormous leap forward in performance at a task considered far beyond the capabilities of contemporary methods. This result revealed the potential of deep learning and kick-started the modern era of AI research.

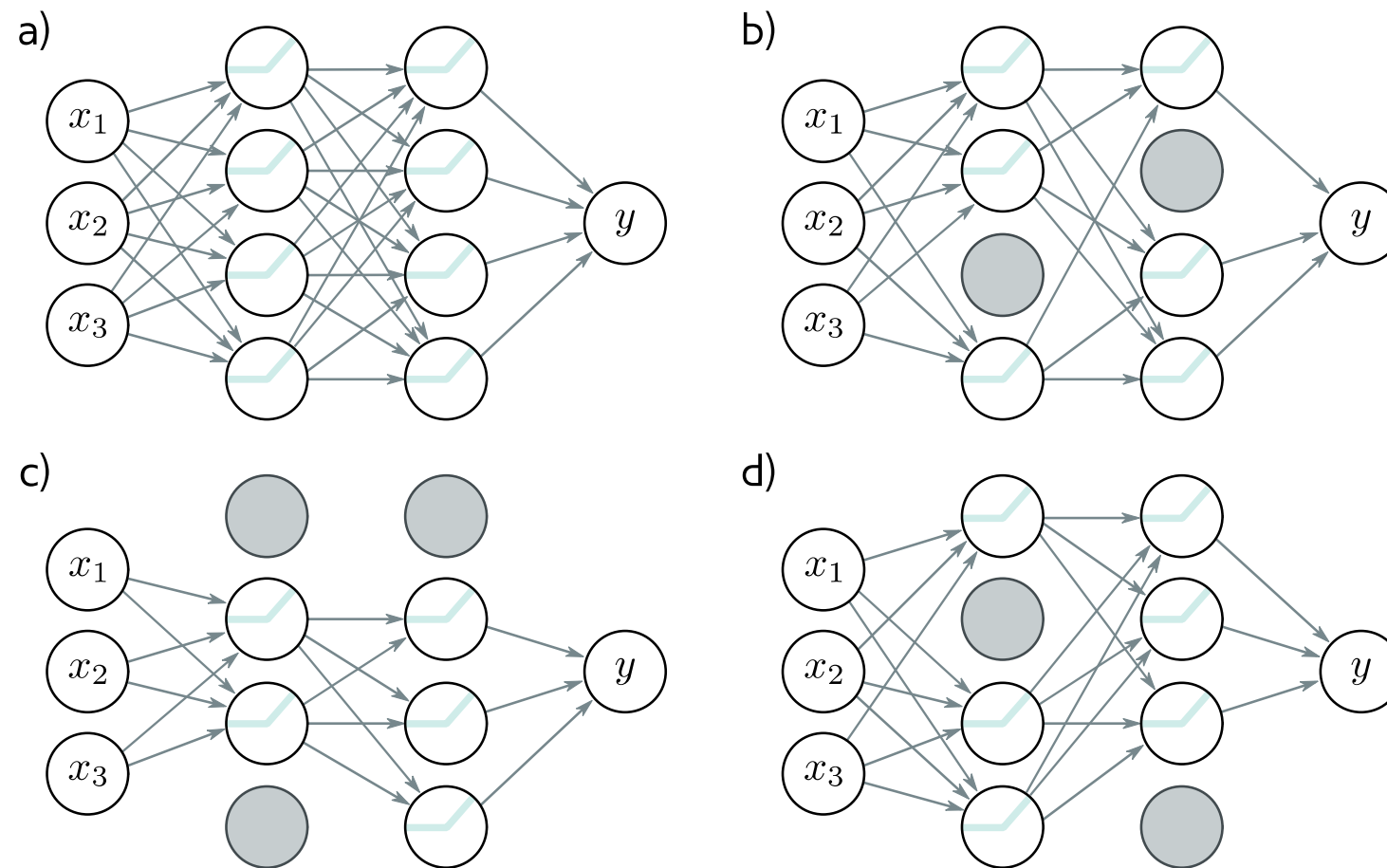
Almost all the 60 million parameters are in fully connected layers

AlexNet: Data augmentation



- Data augmentation a factor of 2048 using (i) spatial transformations and (ii) modifications of the input intensities.

AlexNet used “Dropout” in fully connected layers



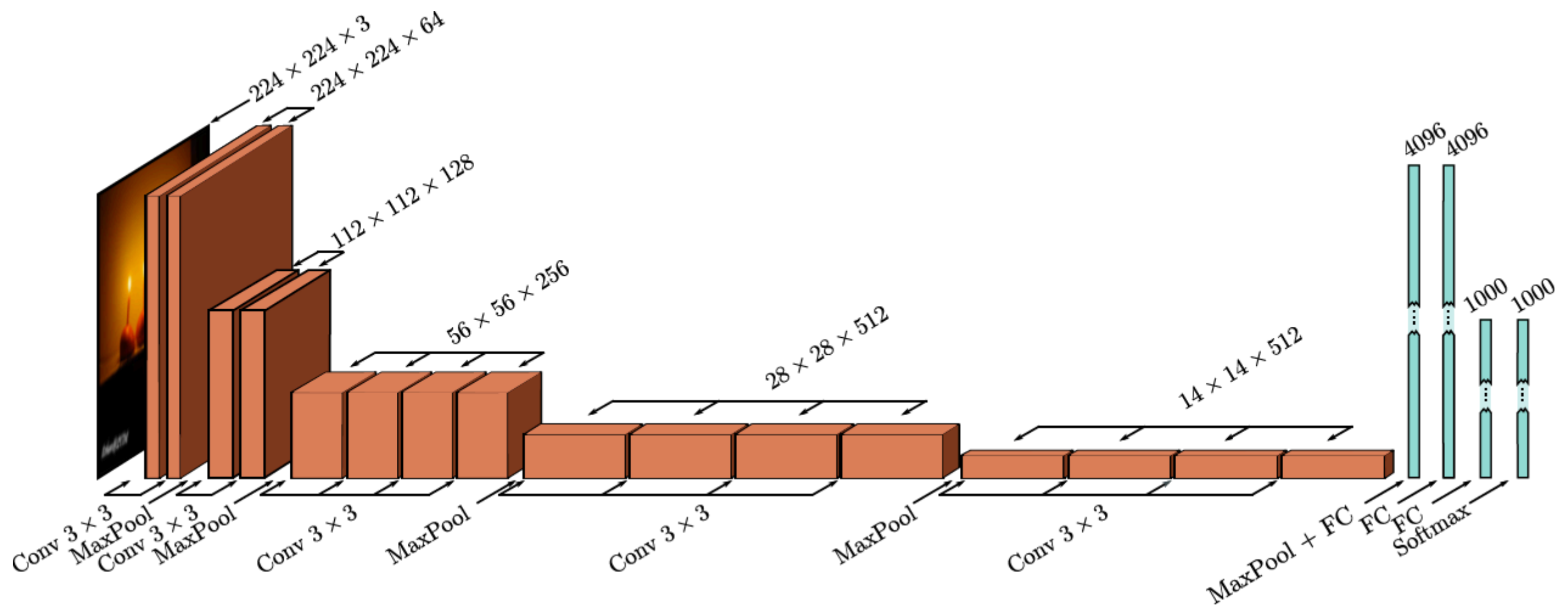
- Dropout was applied in the fully connected layers. Dropout sets a random number of weights to zero at training time, to reduce overfitting.

AlexNet: Details of the training

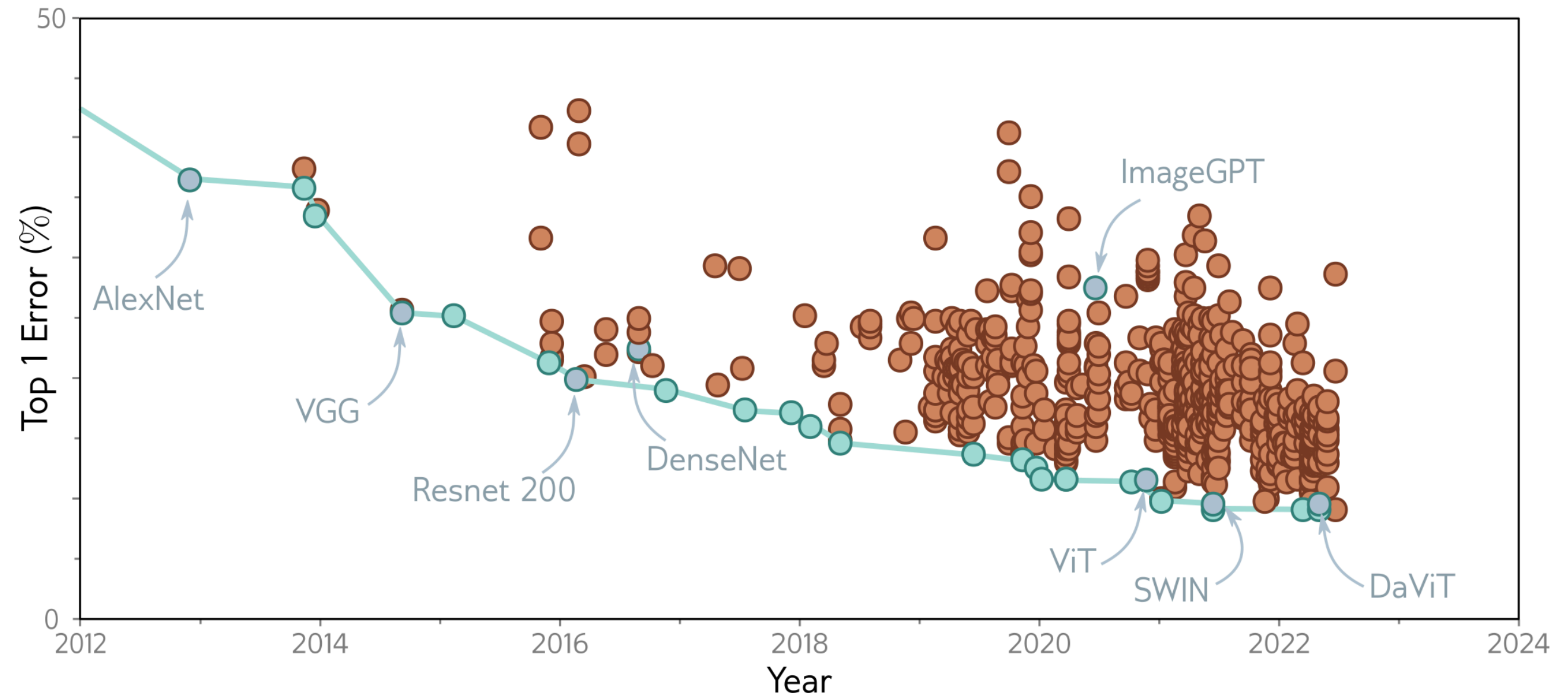
- At test time average results from five different cropped and mirrored versions of the image
- SGD with a momentum coefficient of 0.9 and batch size of 128.
- L2 (weight decay) regularizer used.
- This system achieved a 16.4% top-5 error rate and a 38.1% top-1 error rate.

VGG (2015)

- 19 hidden layers
- 144 million parameters
- 6.8% top-5 error rate, 23.7% top-1 error rate

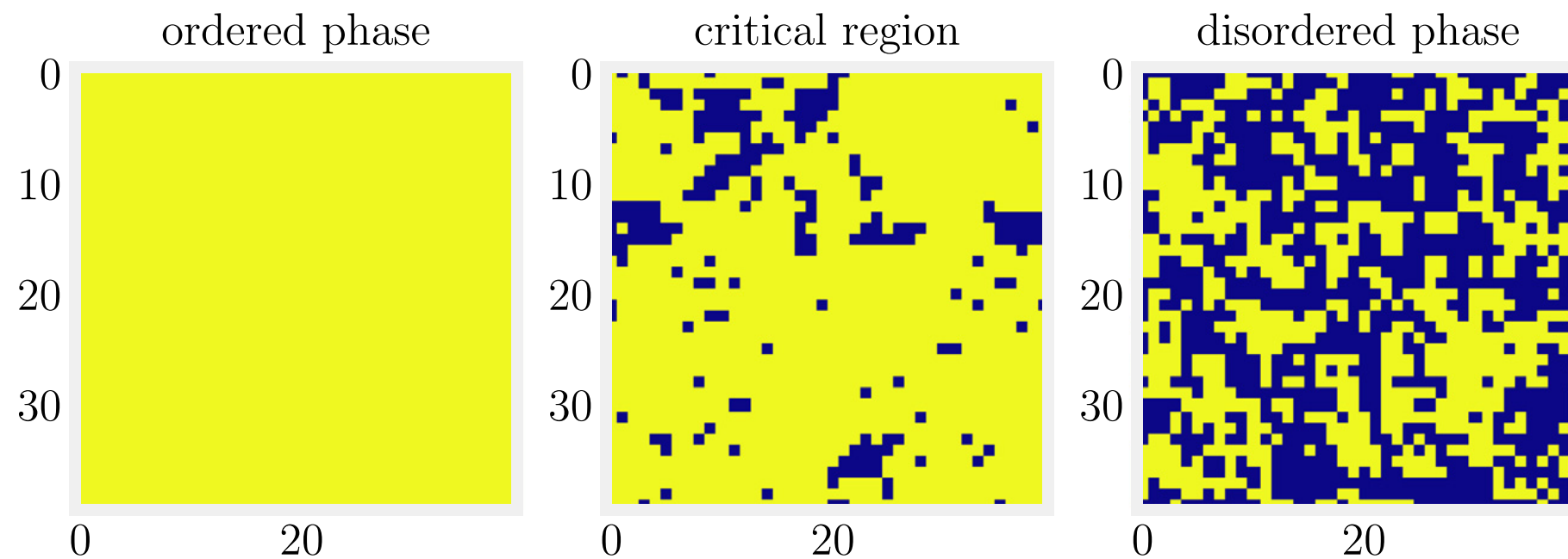


ImageNet History



Example: CNNs for Ising Model

- See Notebook 14 in review 1803.08823: Pytorch CNN (Ising).
- Learn to recognize what phase the Ising model is in.



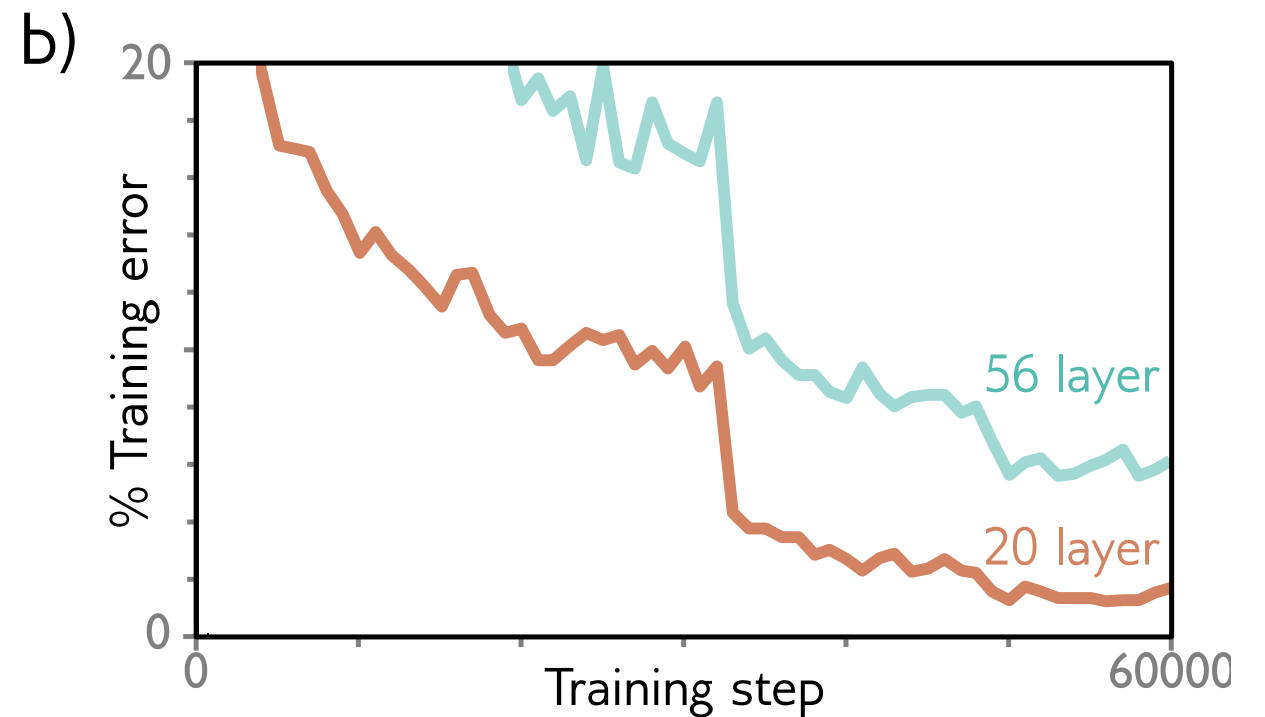
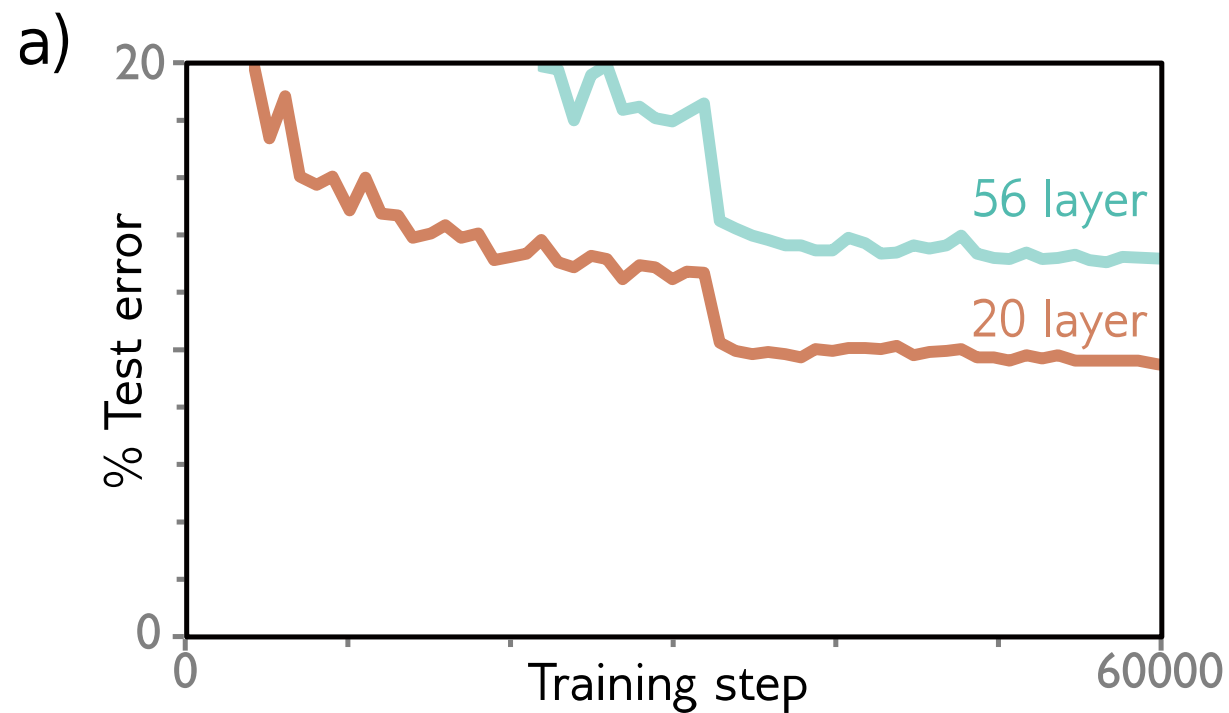
Convolutional Neural Networks

Residual networks (ResNets)

Going deeper

- Image classification performance improved as the depth of convolutional networks was extended from eight layers (AlexNet) to nineteen layers (VGG). This led to experimentation with even deeper networks. However, **performance decreased again when many more layers were added.**
- A novel idea to **overcome this problem are residual blocks.** Here, each network layer computes an additive change to the current representation instead of transforming it directly.
- **Residual blocks** allow much deeper networks to be trained, and these networks improve performance across a variety of tasks.

CIFAR Image classification for deeper networks



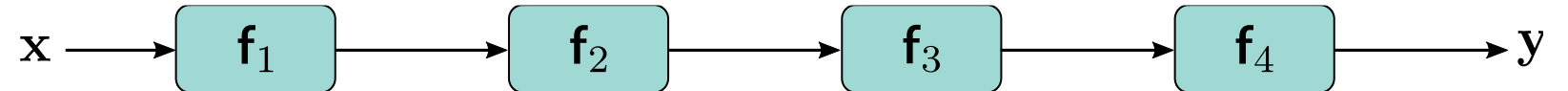
Regular network:

$$\mathbf{h}_1 = \mathbf{f}_1[\mathbf{x}, \phi_1]$$

$$\mathbf{h}_2 = \mathbf{f}_2[\mathbf{h}_1, \phi_2]$$

$$\mathbf{h}_3 = \mathbf{f}_3[\mathbf{h}_2, \phi_3]$$

$$\mathbf{y} = \mathbf{f}_4[\mathbf{h}_3, \phi_4]$$



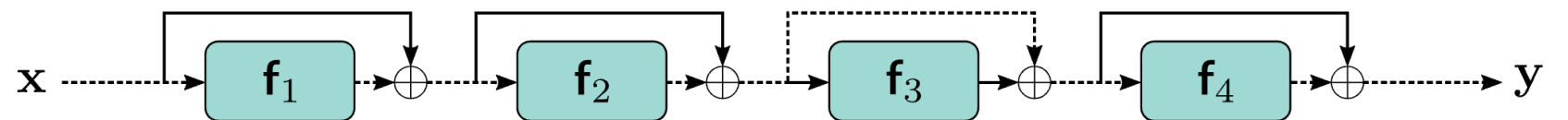
Residual network (2016):

$$\mathbf{h}_1 = \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \phi_1]$$

$$\mathbf{h}_2 = \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \phi_2]$$

$$\mathbf{h}_3 = \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \phi_3]$$

$$\mathbf{y} = \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \phi_4]$$



Course logistics

- **Reading for this lecture:**

- <https://udlbook.github.io/udlbook/> (Simon Prince - Understanding Deep Learning)
- deeplearningbook.com