Convolutional Neural Networks

NSF Workshop – Deep Learning on GPU

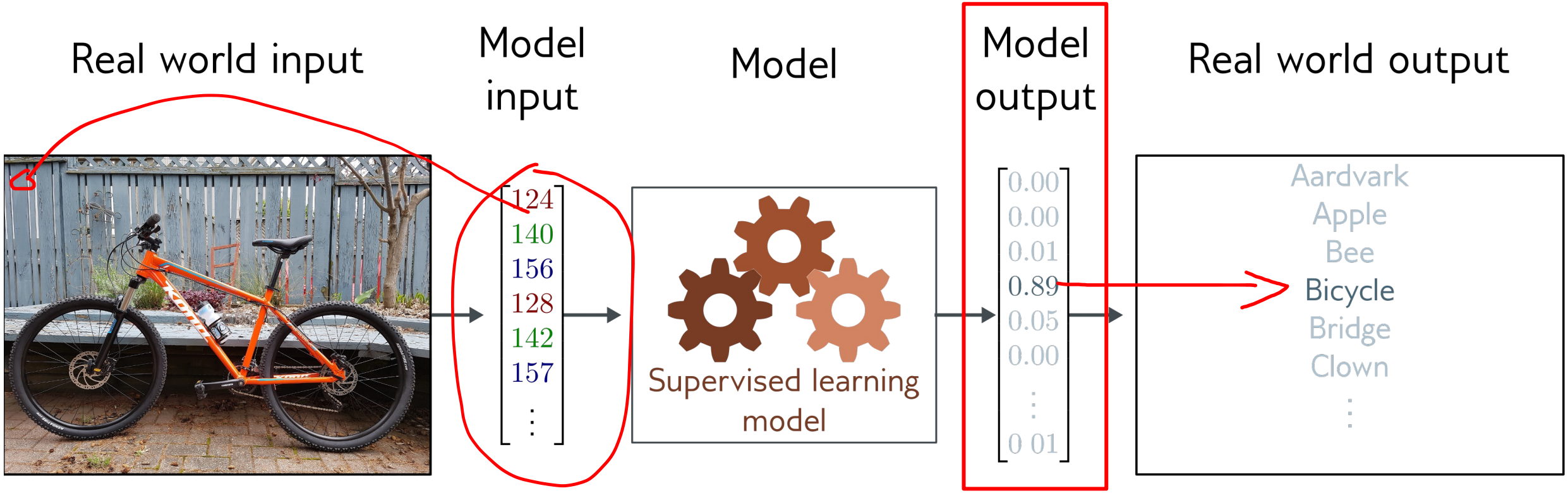
Tony T. Luo

Associate Professor of Computer Science, Missouri S&T

May 20, 2024

Some typical applications of CNN

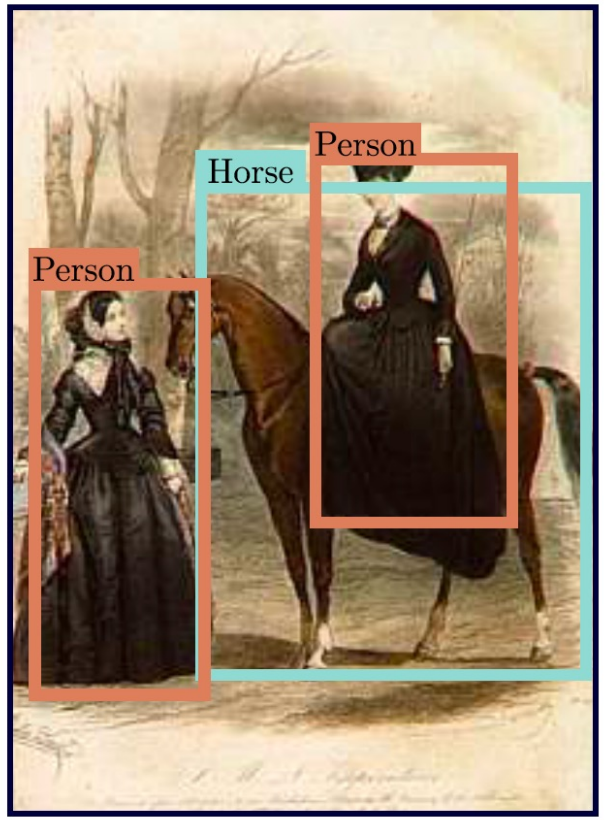
Image classification

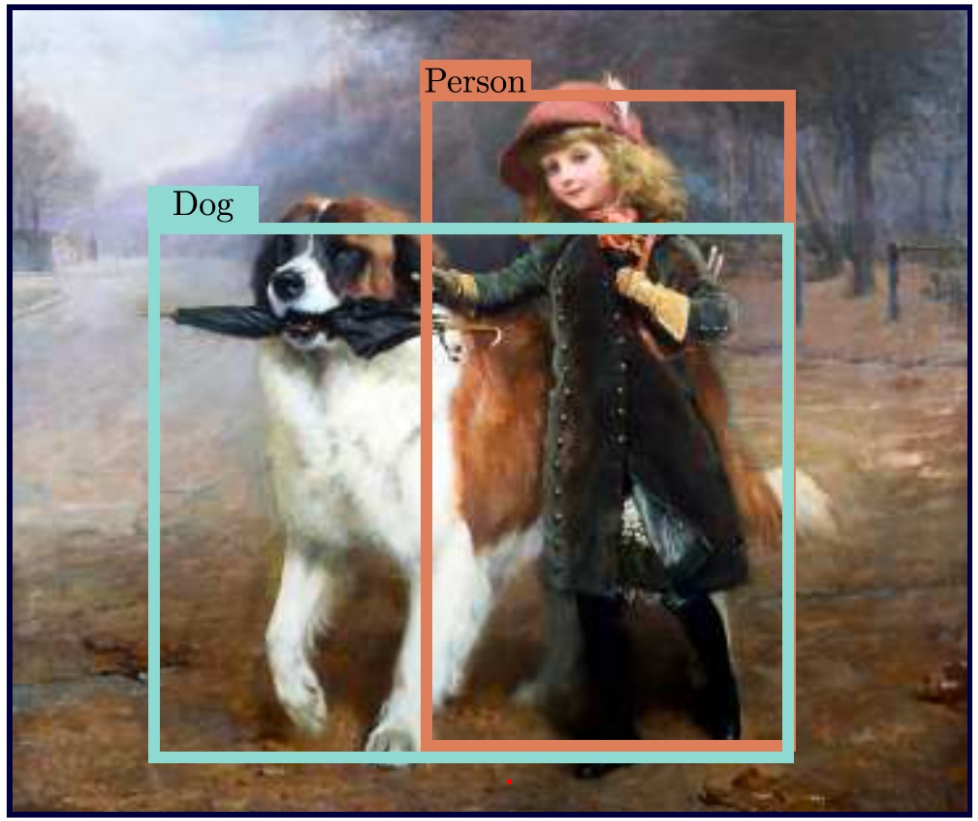


- Multiclass classification problem (discrete classes)

Object detection

Classification + Regression (coordinates of bounding boxes)





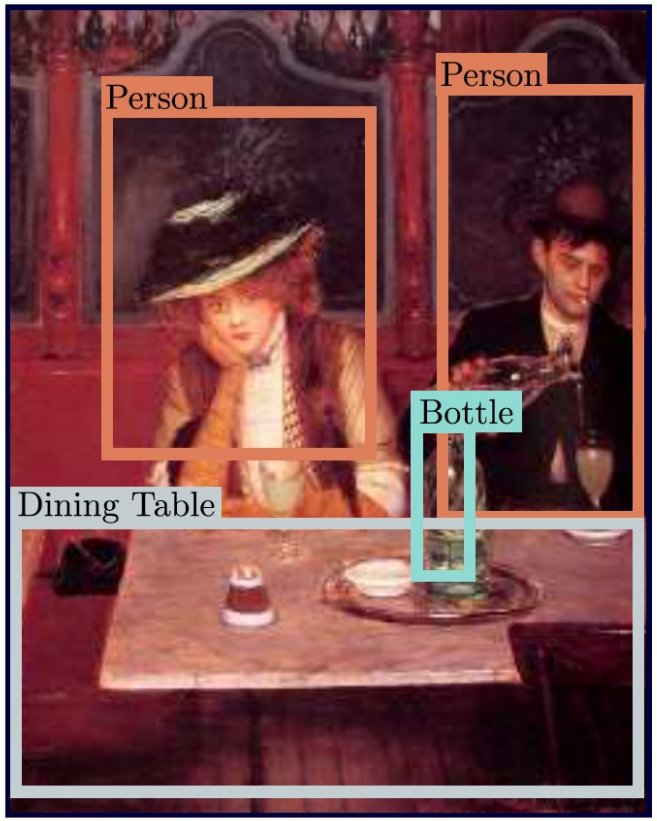
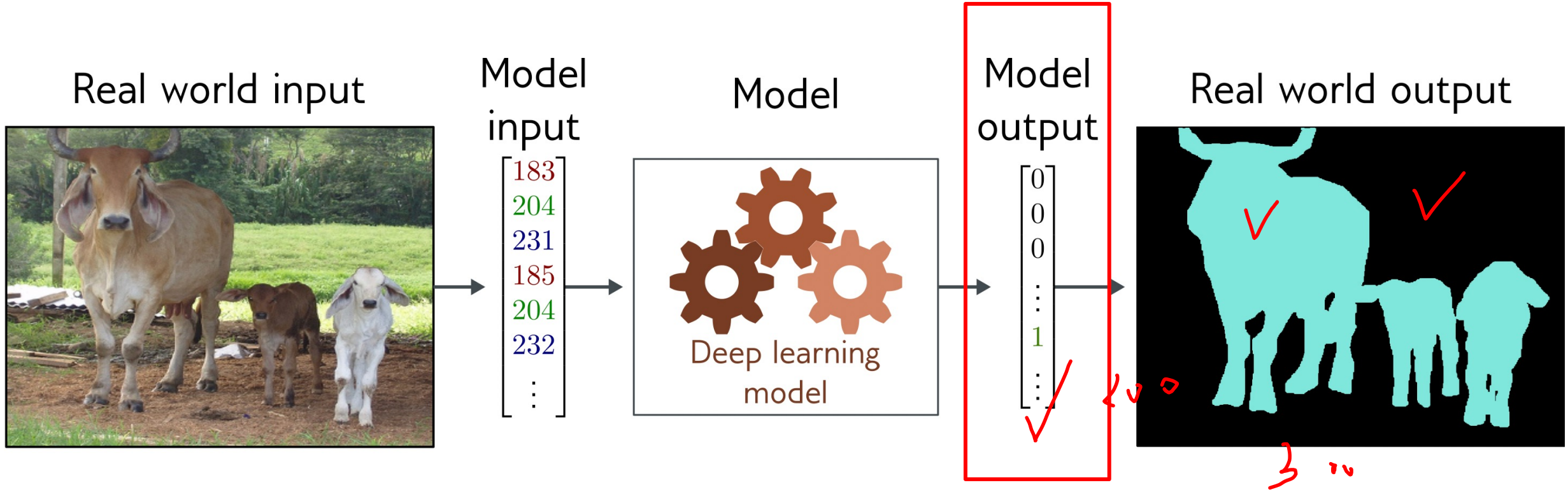


Image segmentation

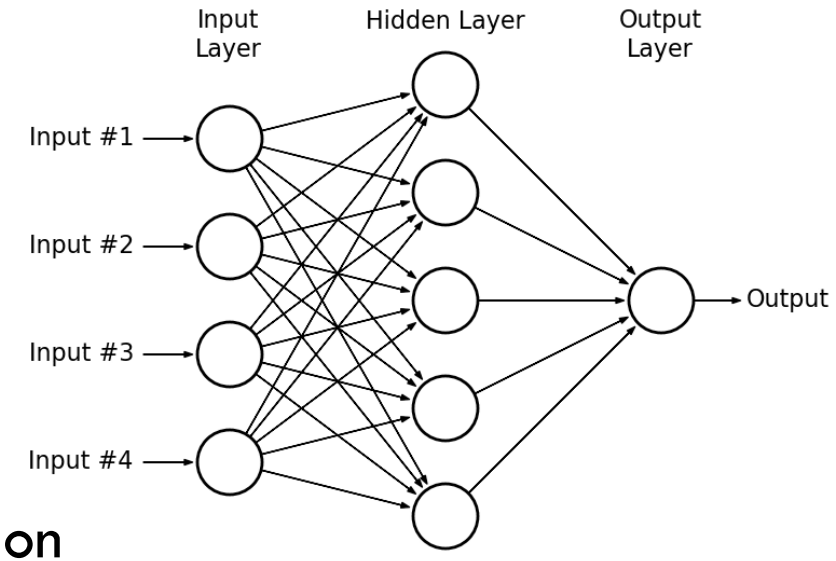


- Multivariate binary classification problem (many outputs, two discrete classes)
  - ❑ Classification of every pixel (object, non-object / background)
- Convolutional encoder-decoder network

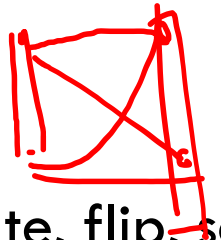
U-net

Why CNN (why not just MLP)?

Problems with MLP (all FC layers)



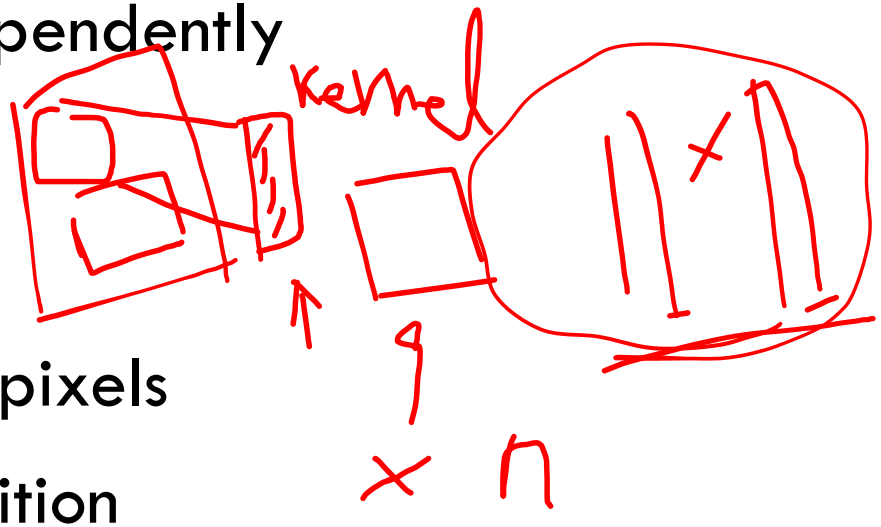
1. Two many parameters
   - 224x224 RGB image = 150,528 dimensions
   - One FC layer = 150,520 x 150,528 weights --> 22 billion
   - And hidden layers are generally"wider" than inputs
2. Spatial correlation (nearby pixels are statistically related)
   - But FC has no notion of "nearby" (proximity) and treats relationship between every pixel the same
   - Every neuron sees the same inputs



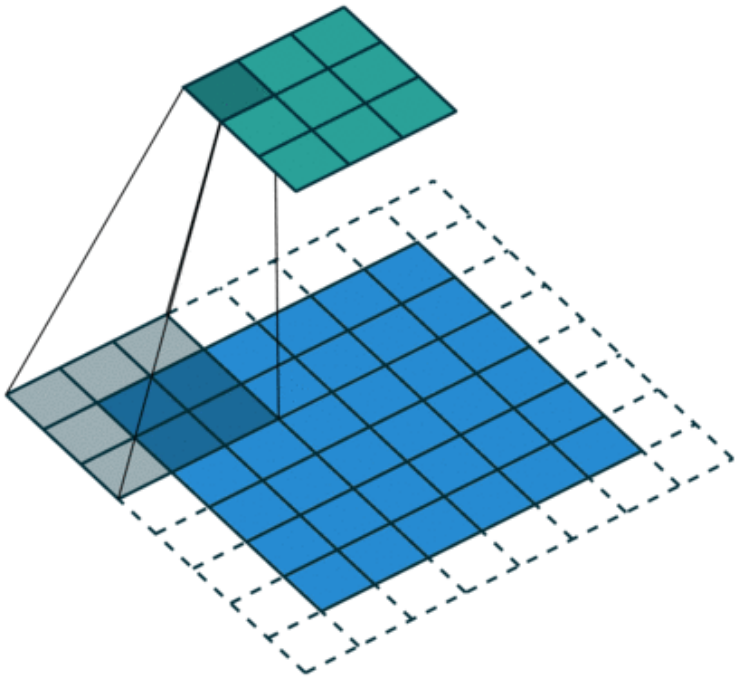
3. Geometric transformations (e.g., shift, rotate, flip, scaling) should have similar/stable output
   - But FC has to re-learn all the patterns separately at every changed position

Convolutional layers

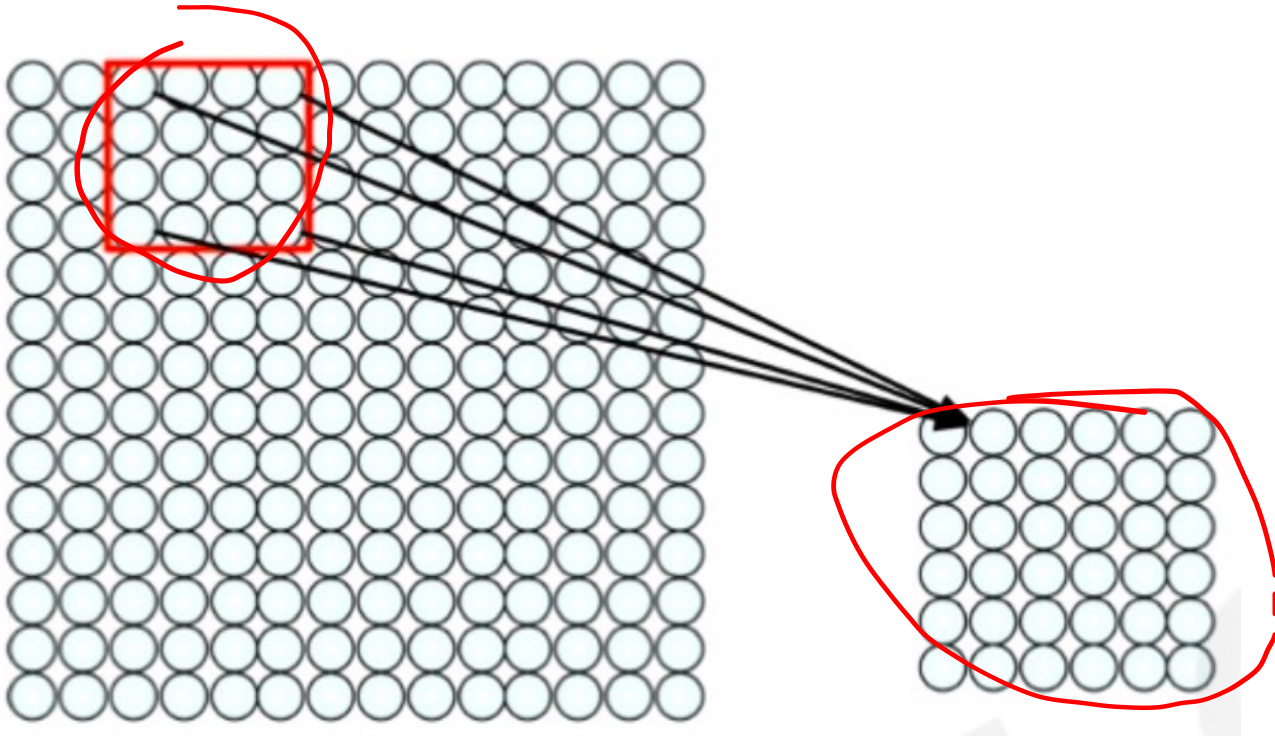
- Process each local image region/patch independently
- Share parameters across the whole image
- Fewer parameters than FC
- Exploit spatial relationship between nearby pixels
- Don't have to re-learn patterns at every position



Convolution Operation



Each neuron only sees a “patch”



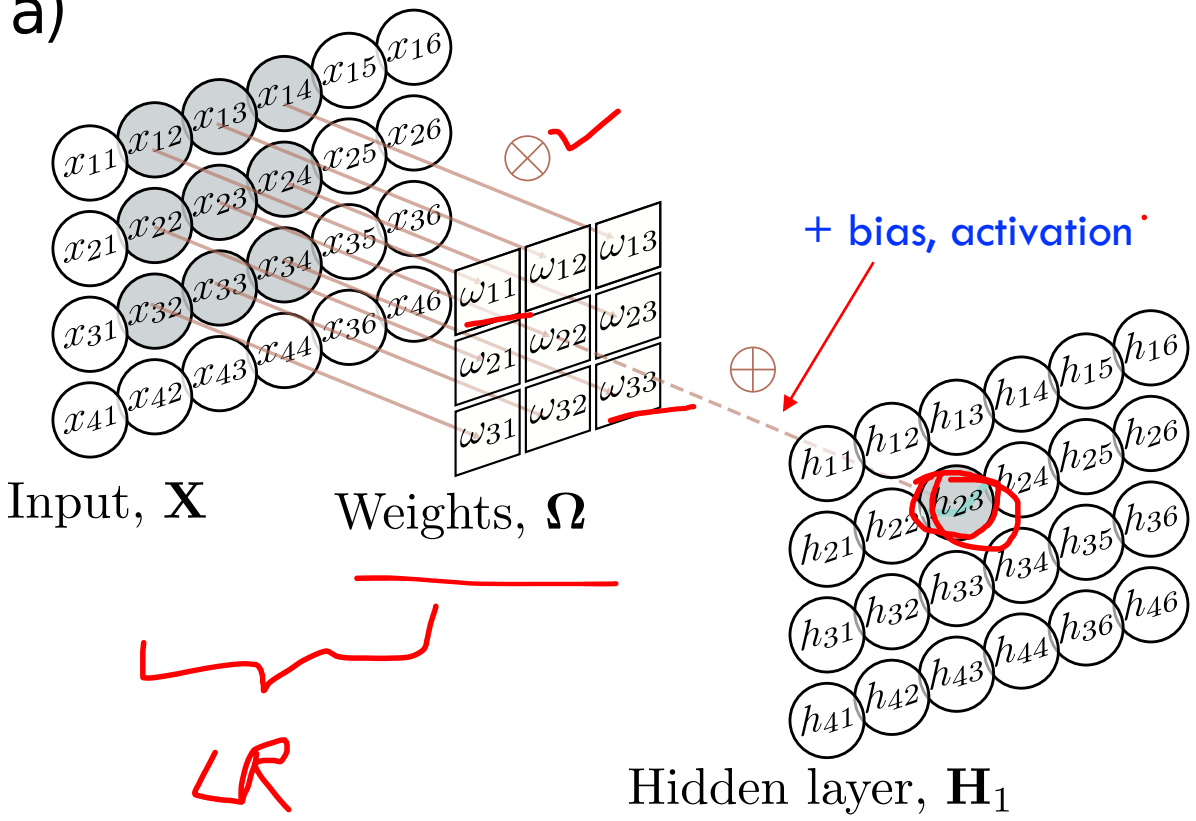
A more detailed view

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

sig

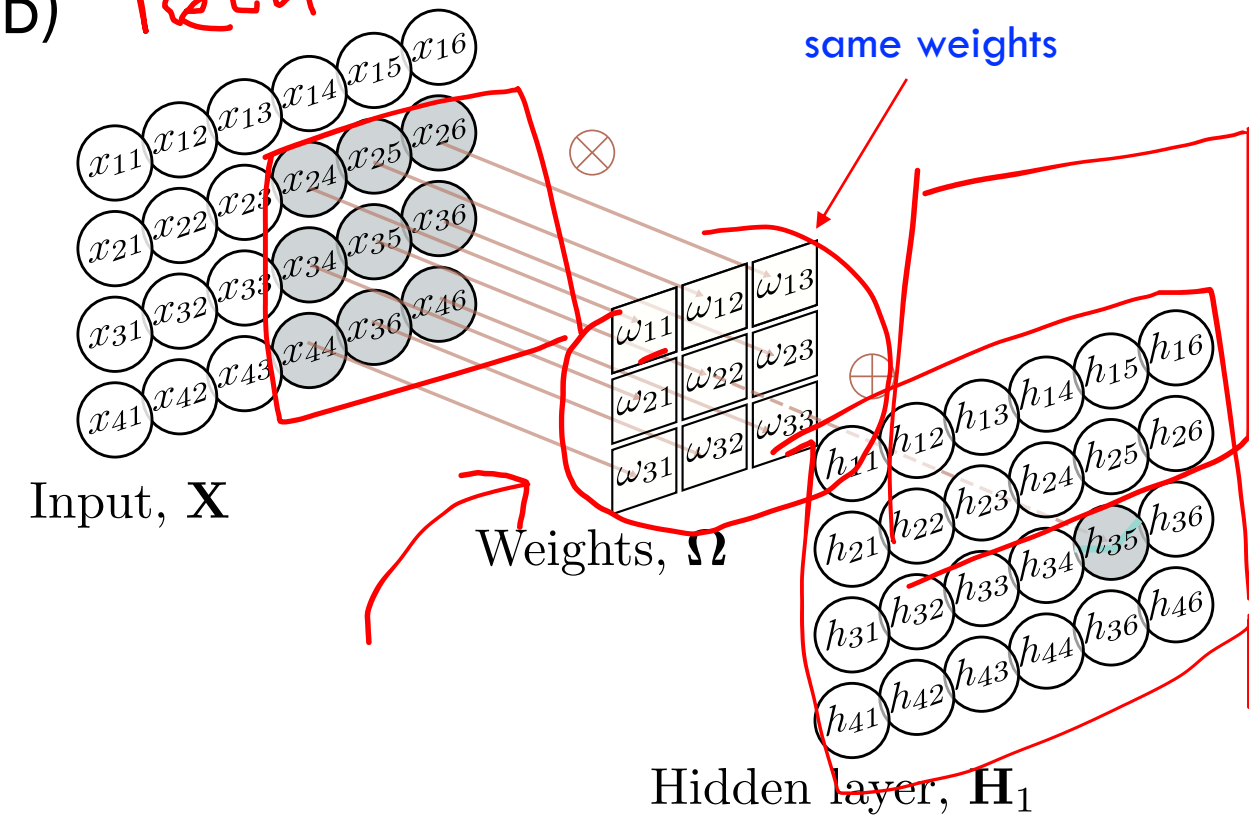
ReLU

a)



LR

b)



But there is a problem with CNN

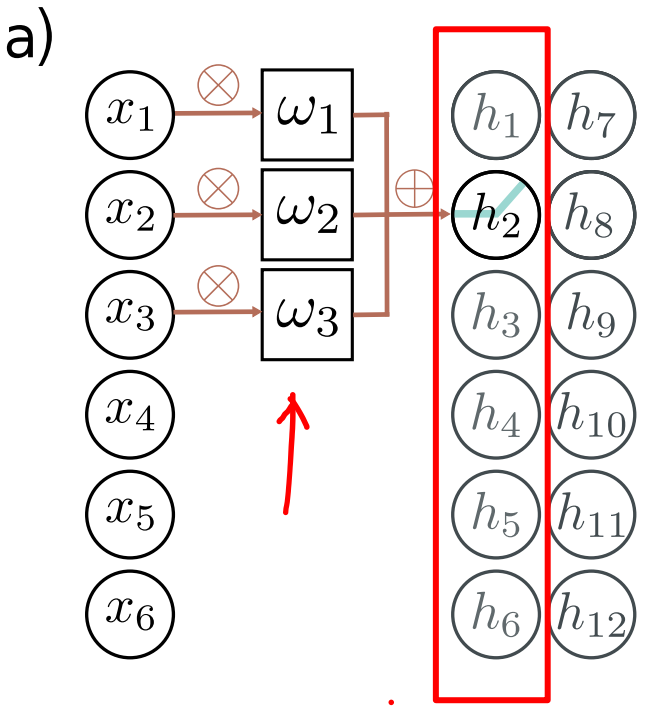
- The convolutional operation averages nearby inputs (weighted avg)
- And passing through ReLU function clips negative results
- Hence lose information

How to compensate?

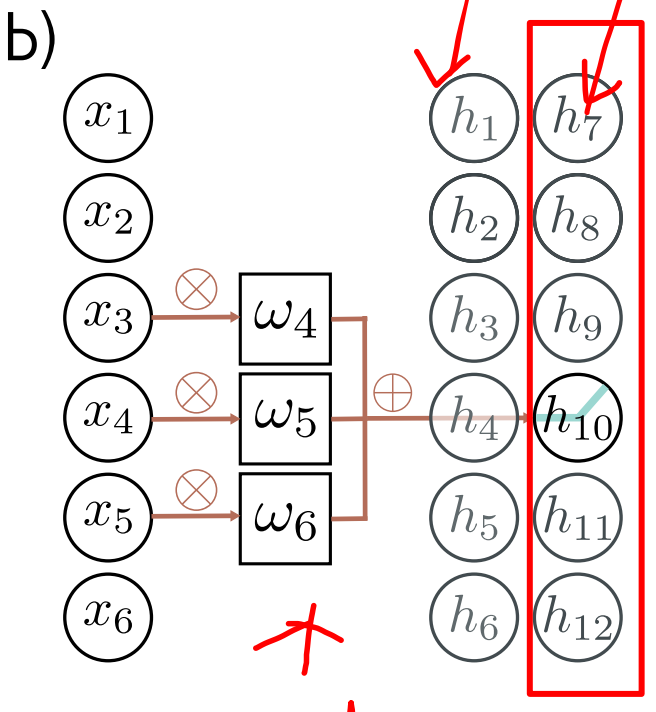
- apply several convolutions (in parallel due to independence)
- each convolution produces a set of hidden units (or hidden variables)
- stack them each as a channel, also called a feature map

Let's illustrate this idea in 1D convolution (2D is the same in principle)

Two output channels, one input channel



Conv-1, channel-1

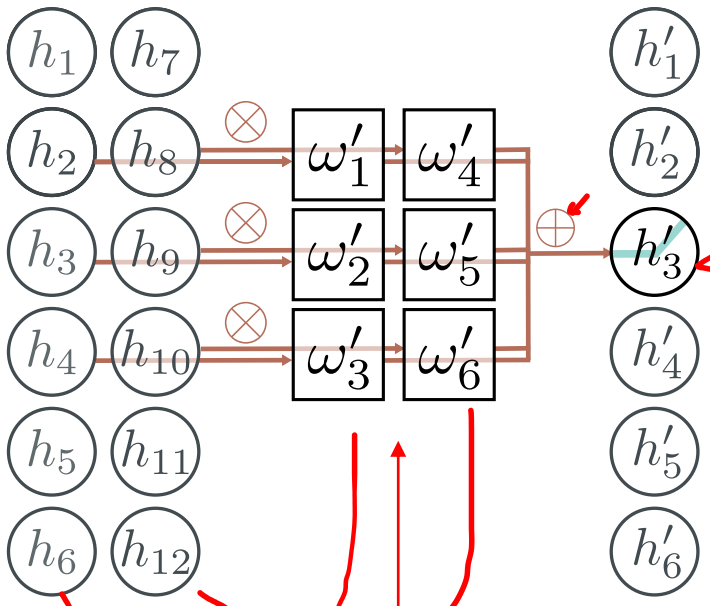


2nd

Conv-2 (note the weights are different), channel-2

Two input channels, one output channel

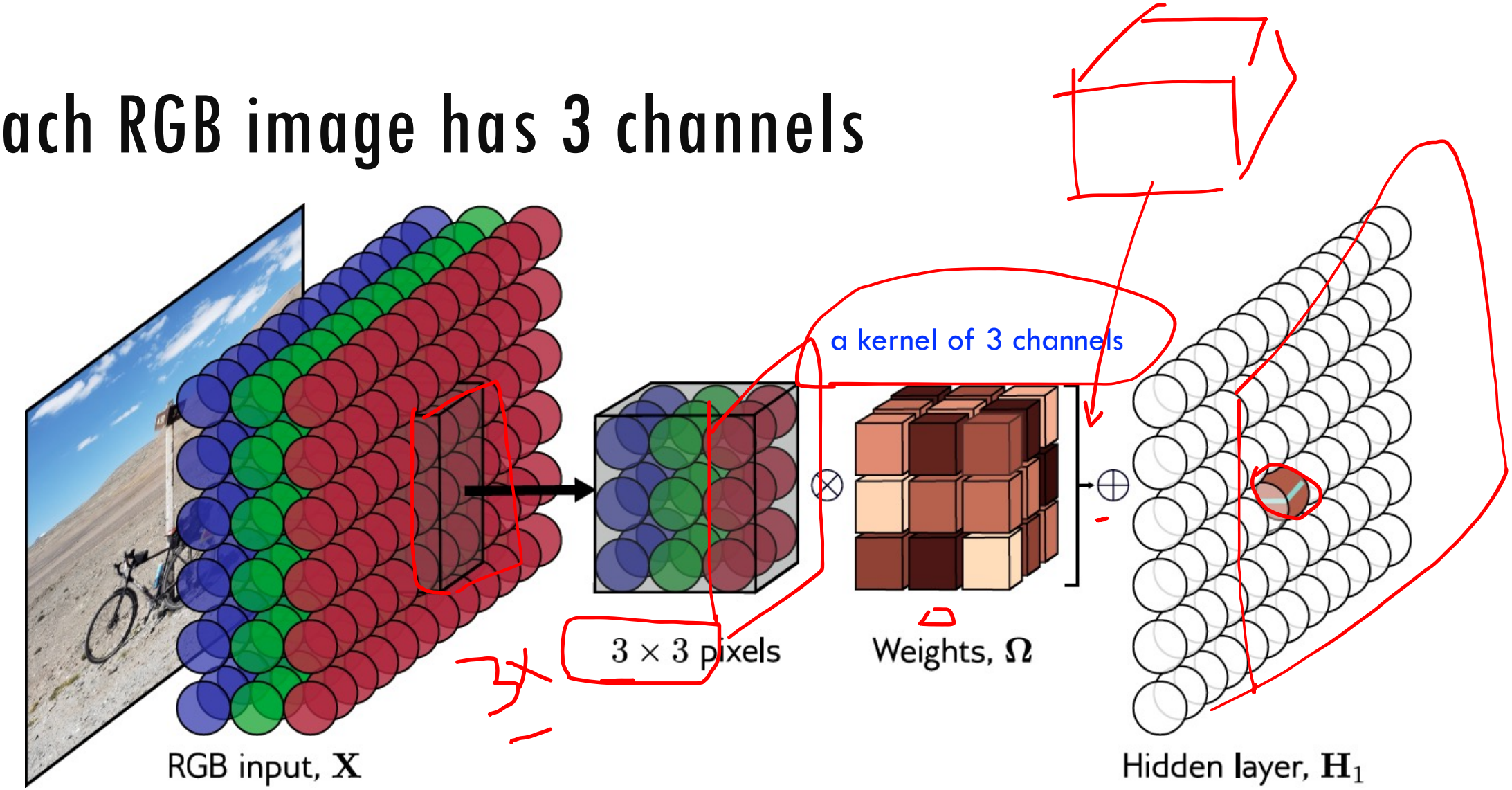
Add one more conv layer (after the two layers in the previous slide)



Each unit is computed as a simple sum of all channels (sum of weighted sums)

Now the kernel also has two "channels", each corresponding to one feature map

Each RGB image has 3 channels



Variations in convolution

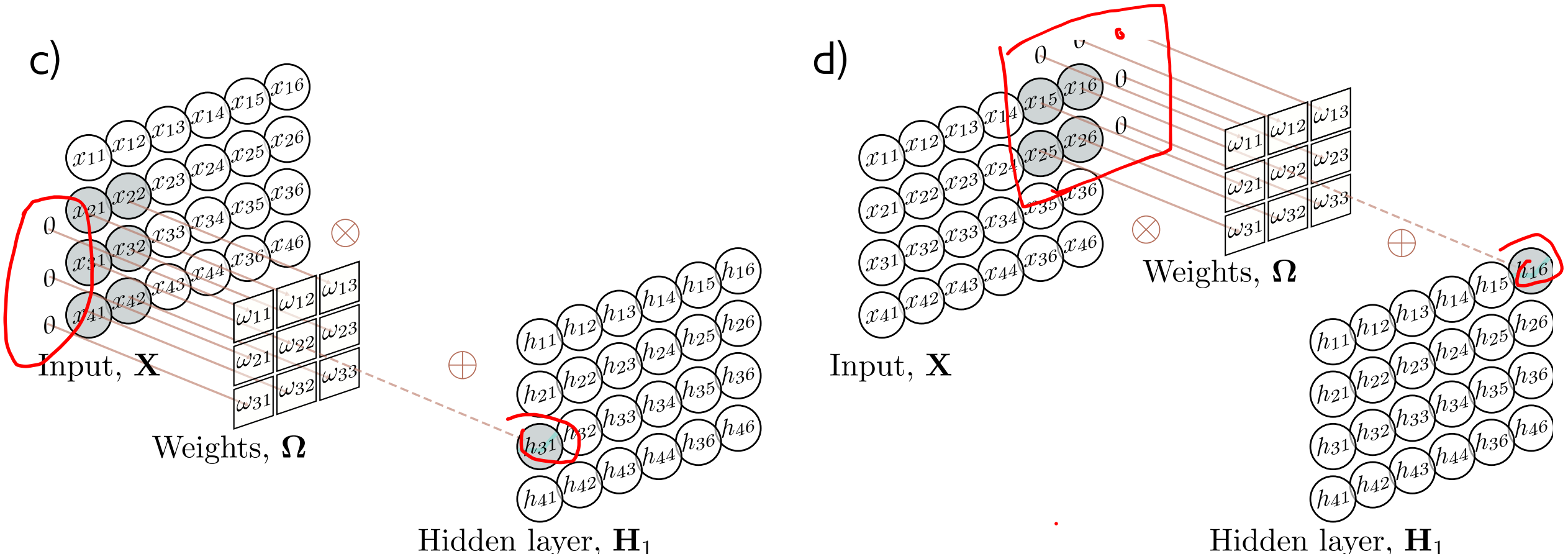
Padding ✓

Stride ✓

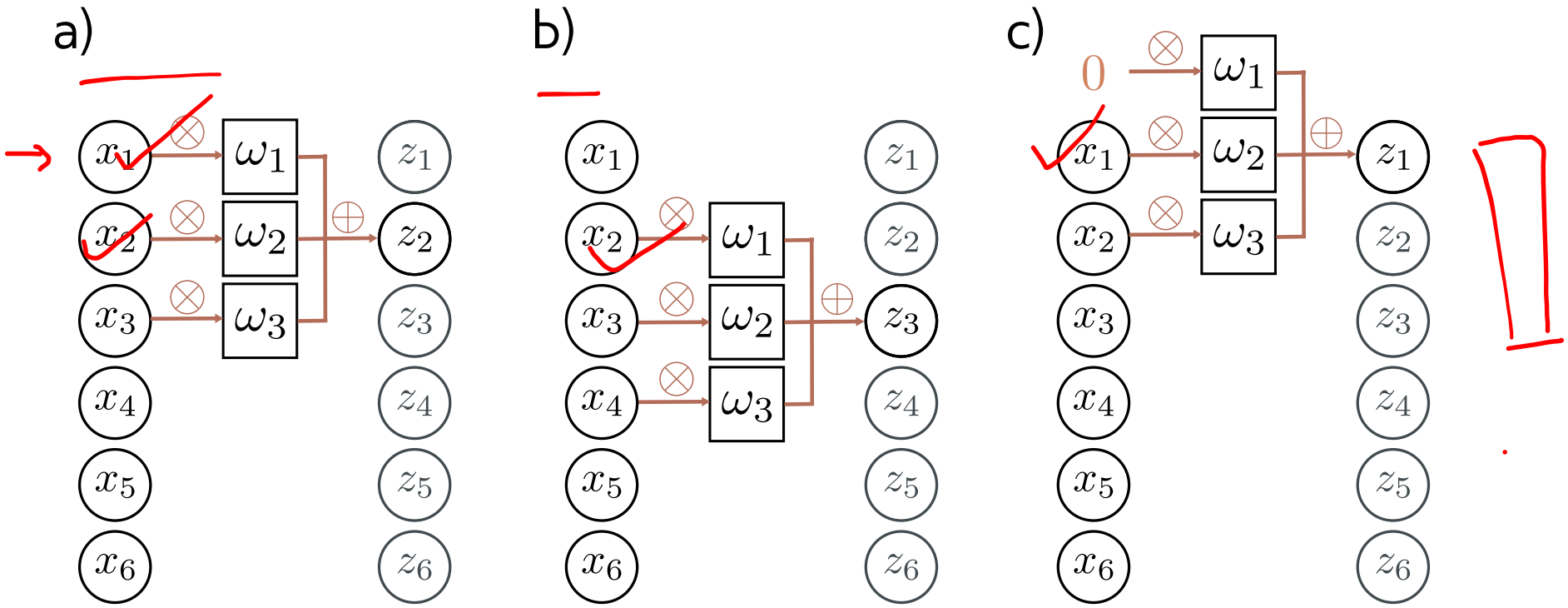
Dilation ✓

Conv2d(in, cout, sz, p, s, d)

2D Convolution with padding



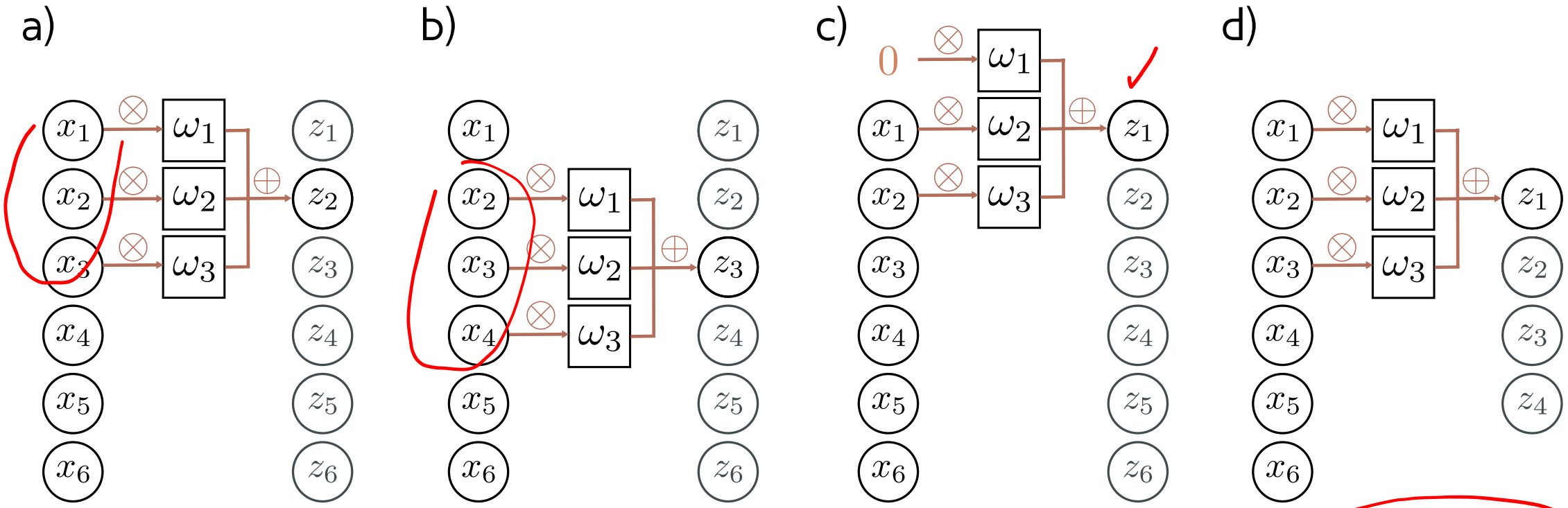
Illustrating zero padding in 1D conv



Zero padding

"Valid" convolutions (no padding)

Alternative to padding

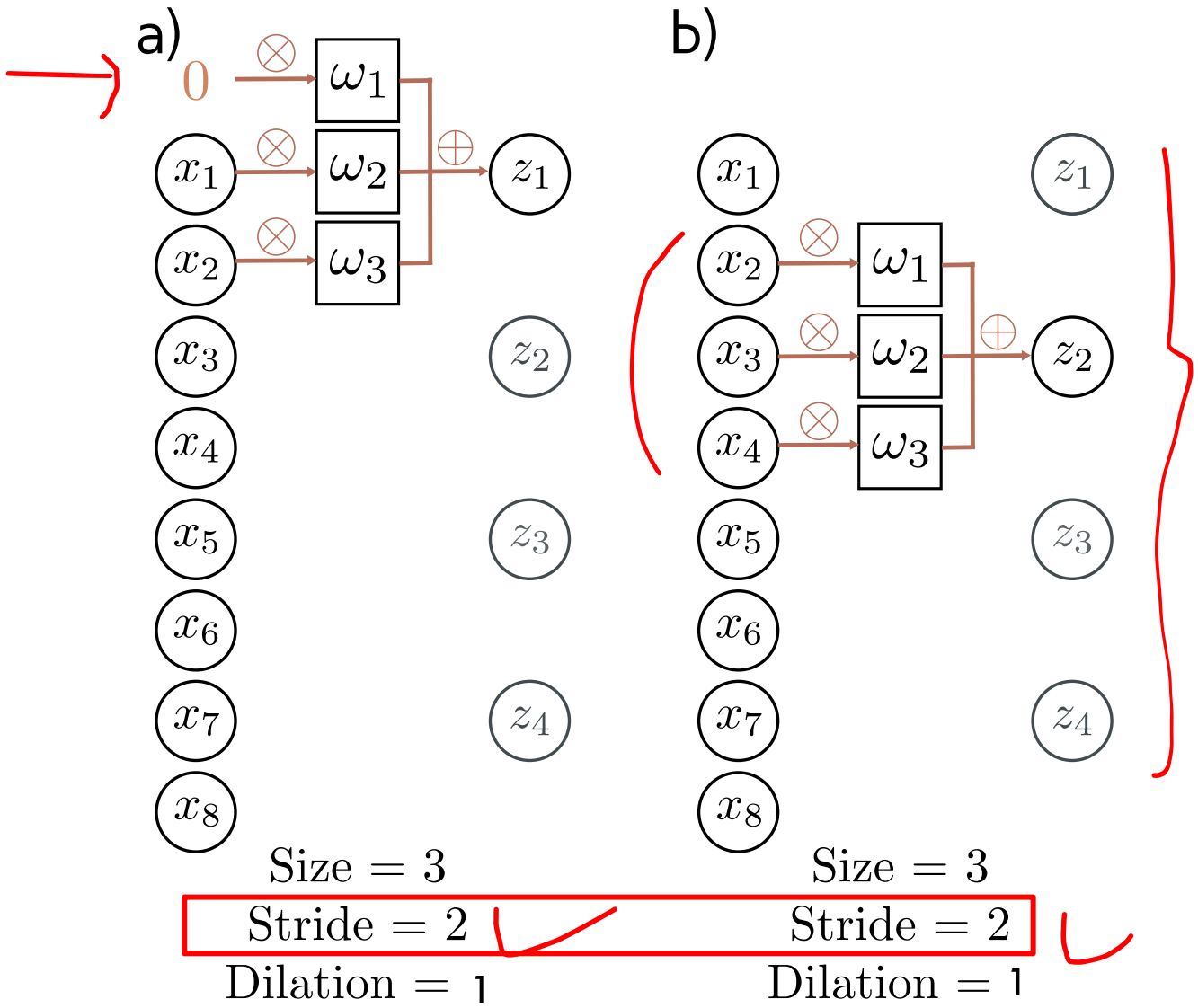


s = 1

"valid" convolution → smaller output

Stride = shift by k positions for each output

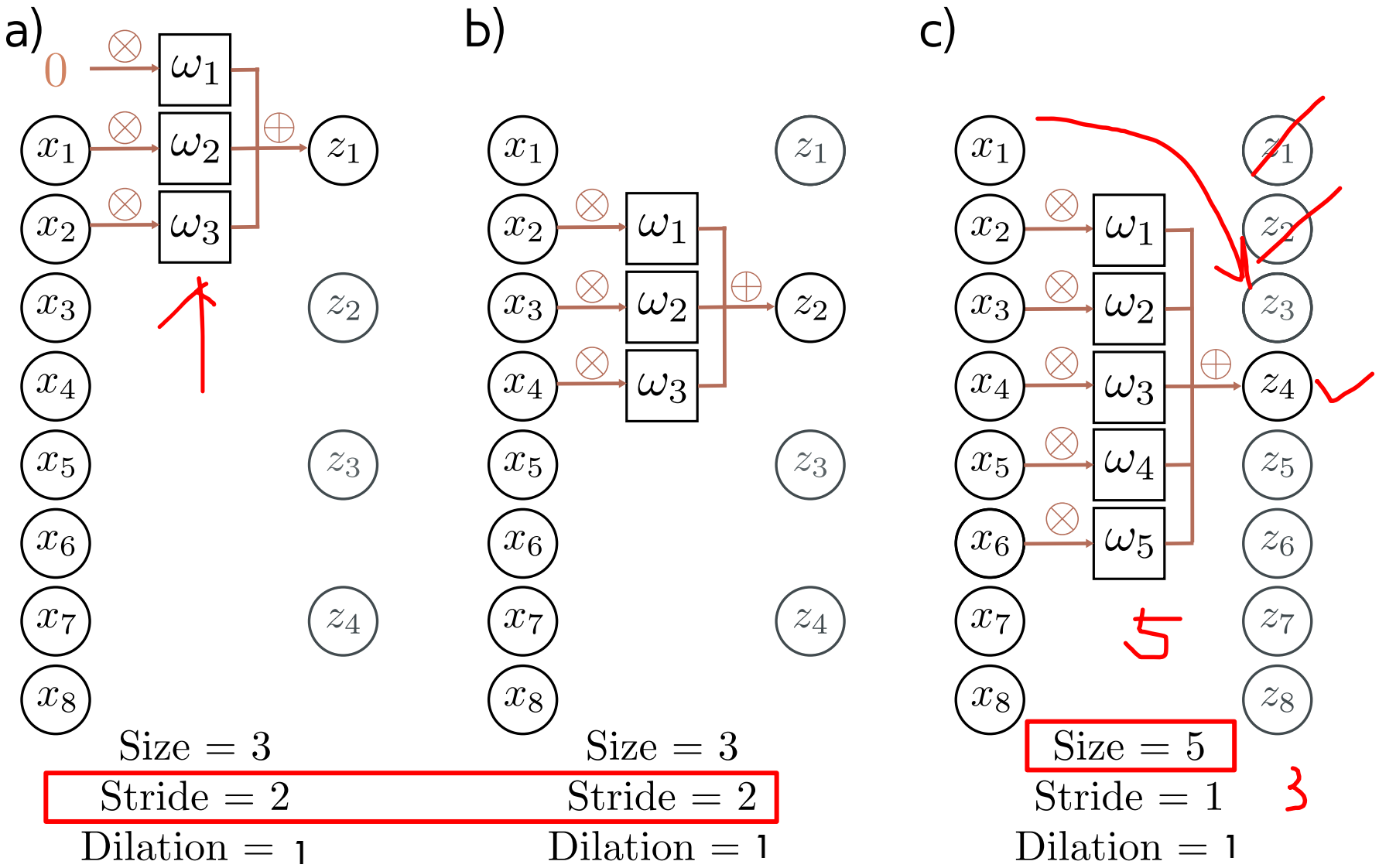
- Decreases size of output relative to input



Output size reduce to 1/s

Kernel size = weight a different number of inputs for each output

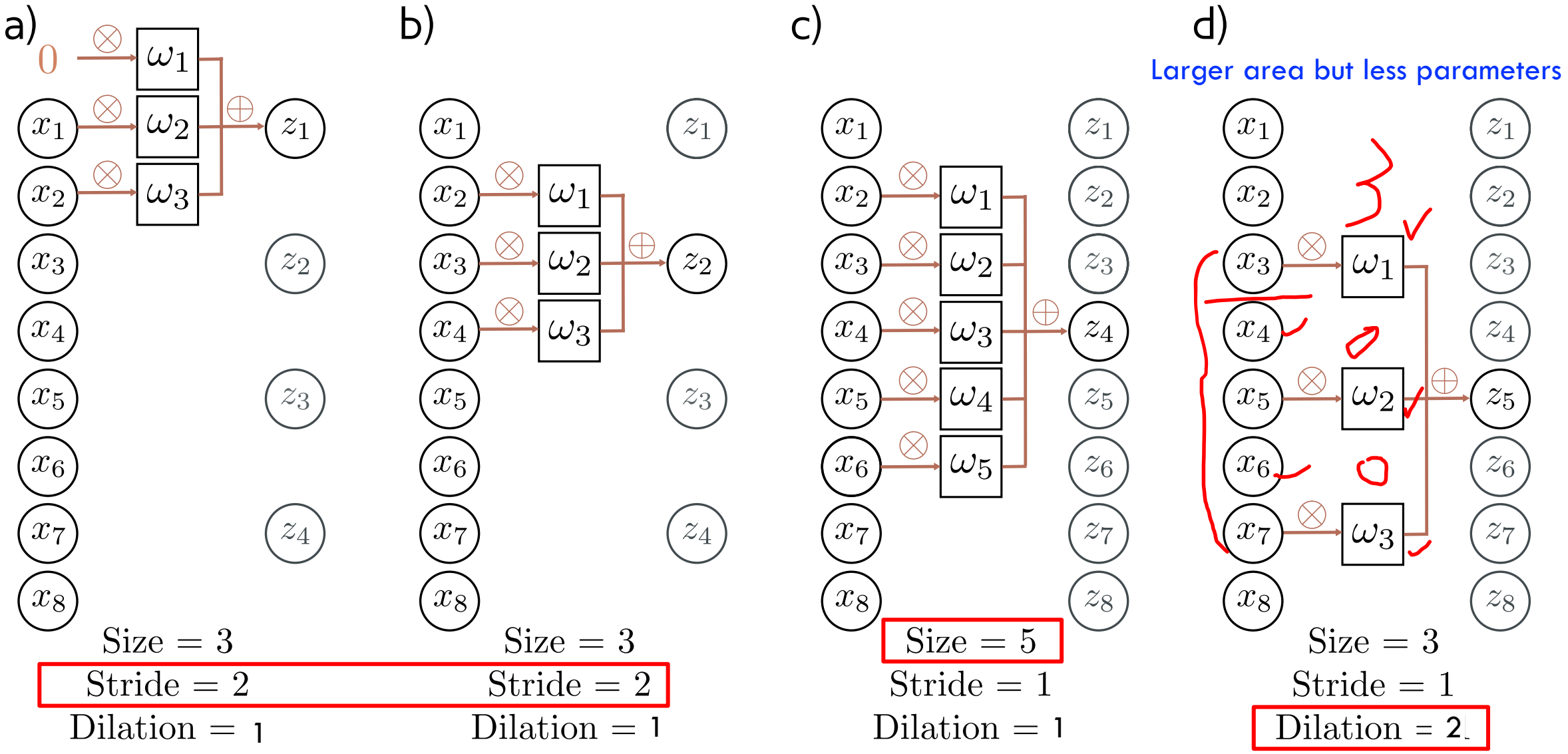
- Larger kernel combines information from a larger area, with more parameters (weights)



Typically kernel size is an odd number so that a kernel is centered around the current position.

Dilated convolutions = intersperse kernel values with zeros

- Also combine information from a larger area, but with fewer parameters than using a larger kernel



Summary so far

- Why CNN (problems with MLP)
- Convolution: patch x kernel/filter → reserve spatial correlation
- Channel to compensate information loss
- Variations:
  - Padding
  - Stride
  - Dilation

How does a complete convolution operation look like?

It involves other operations:

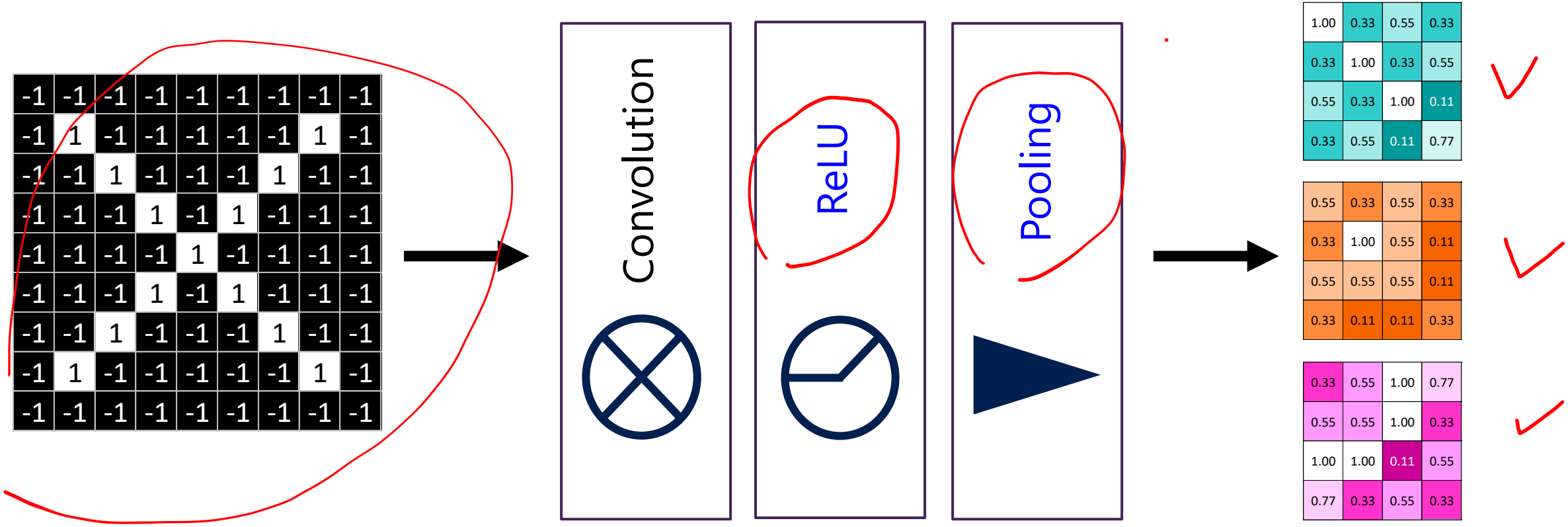
- ReLU
- Pooling

Pipeline

Add *activation function* to each convolution

- *Sigmoid, Relu, Tanh, etc.*

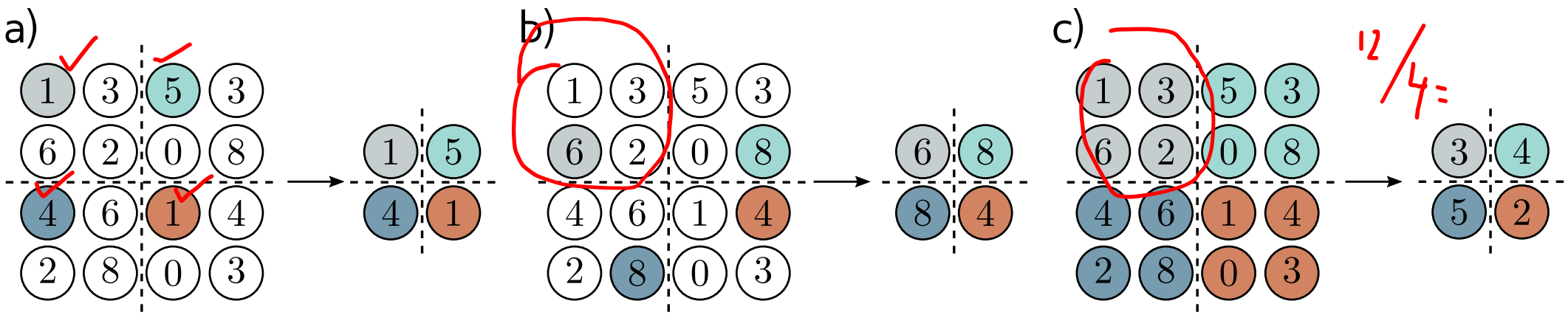
Add pooling to downsample feature map



Why do downsampling?

Reduce representation size (parameters) while keeping the receptive field size

Downsampling (by a factor of 2)



[D]

Sub-sampling
(sample every other position; fixed)
(equivalent to stride 2)

Max pooling 2 x 2
(partial invariance to translation – say input shifts by 1 pixel, many max values remain the same)

Mean pooling

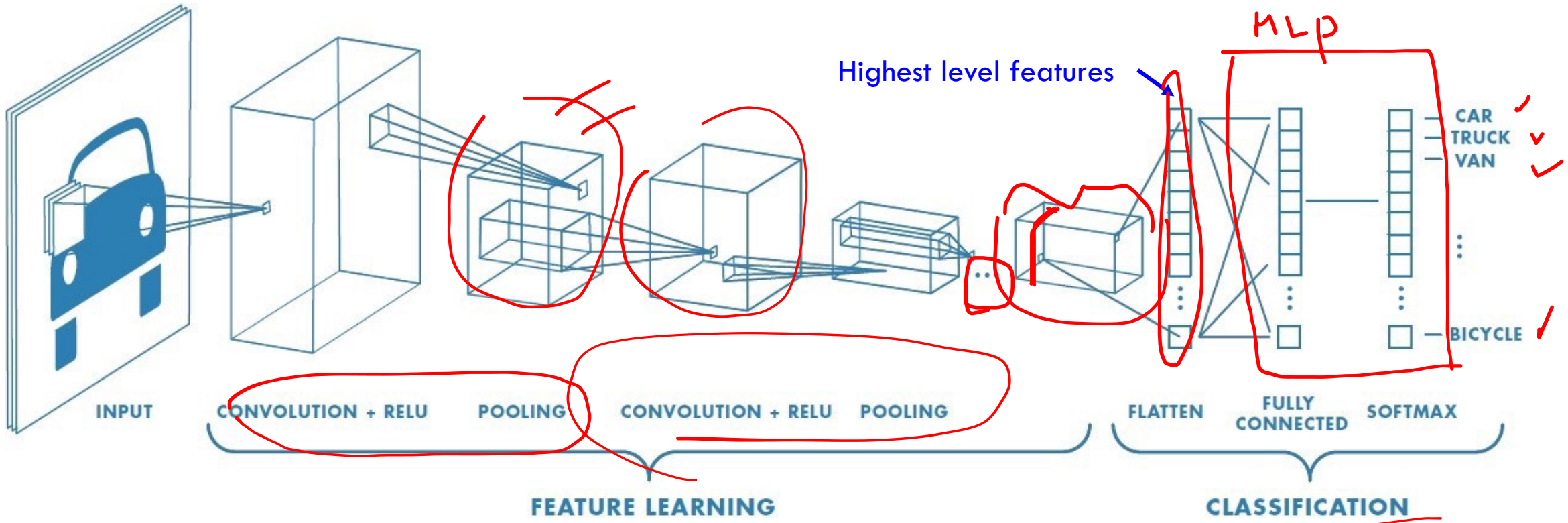
Now it's time to look at an entire CNN

all

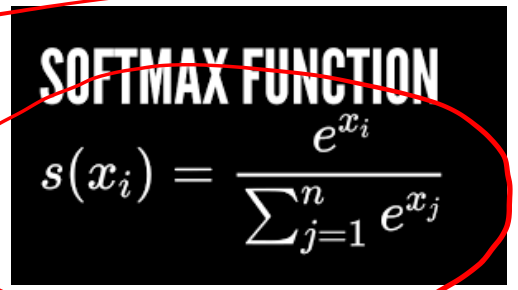
It concatenates multiple conv blocks (for feature extraction)

and then followed by a few FC layers (for final classification)

CNN: concatenates many conv layers + FC

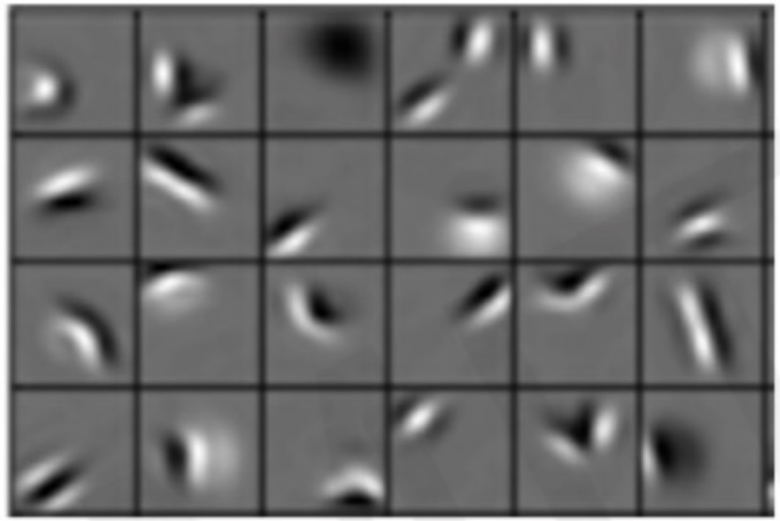


- Convolution + activation + pooling
- Convolution increases depth by <# of filters> folds
- Output of FC: scores of the input belonging to each class



Intuition of "features"

Low level features



Edges, dark spots

Mid level features

Eyes, ears, nose

High level features

Facial structure

CNN is Versatile

Image classification

Object detection

Segmentation

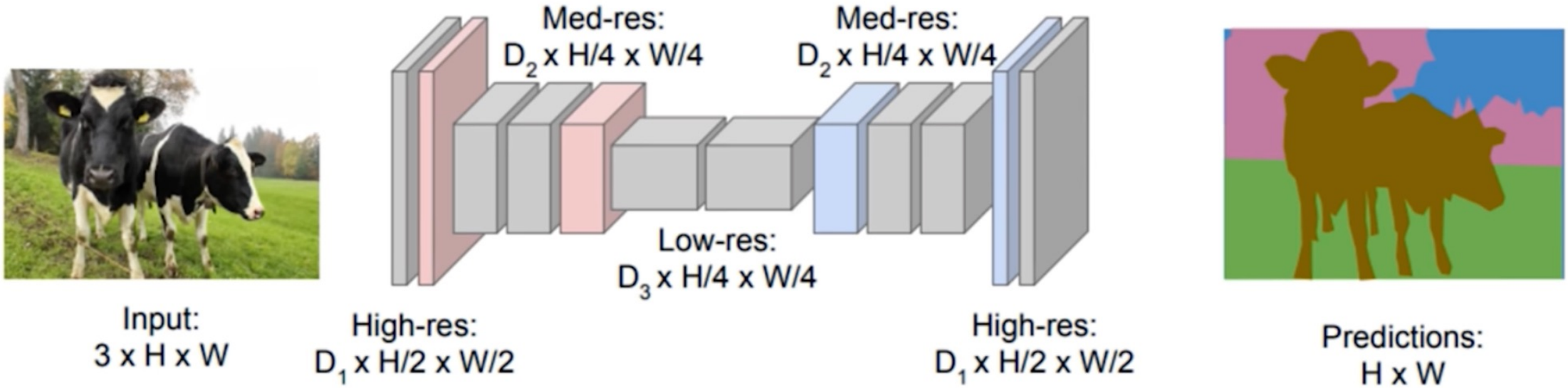
… (probabilistic control / robotics)

Object detection

Multi-OD

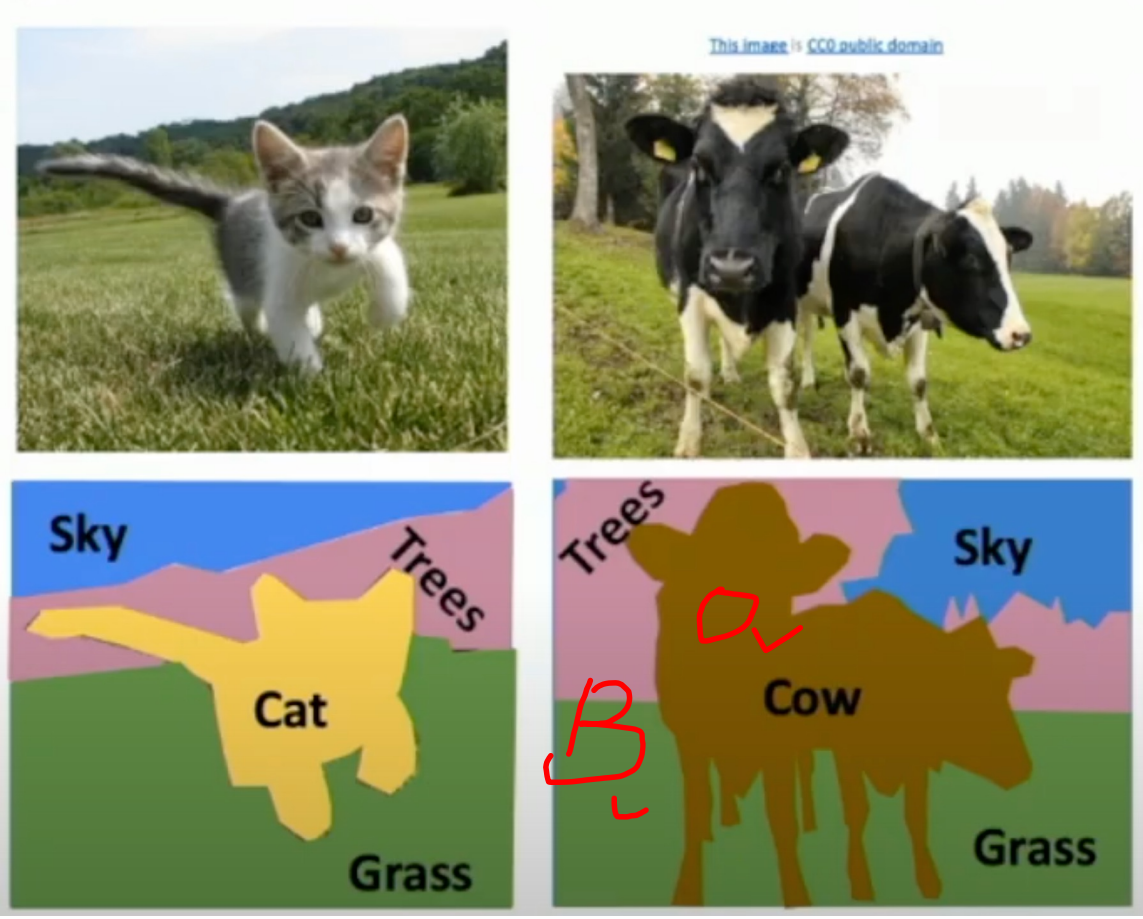
Semantic segmentation:

Assign a class label (membership) to every pixel



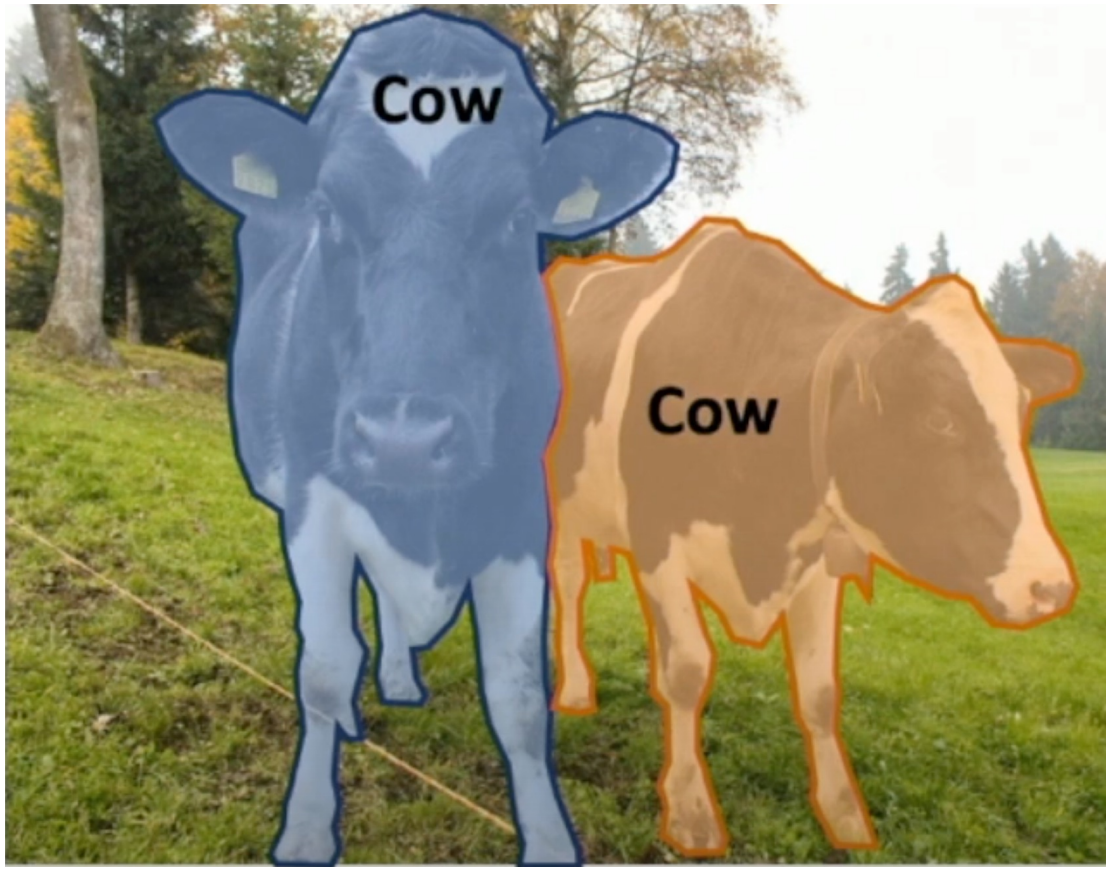
Semantic segmentation

(pixels)

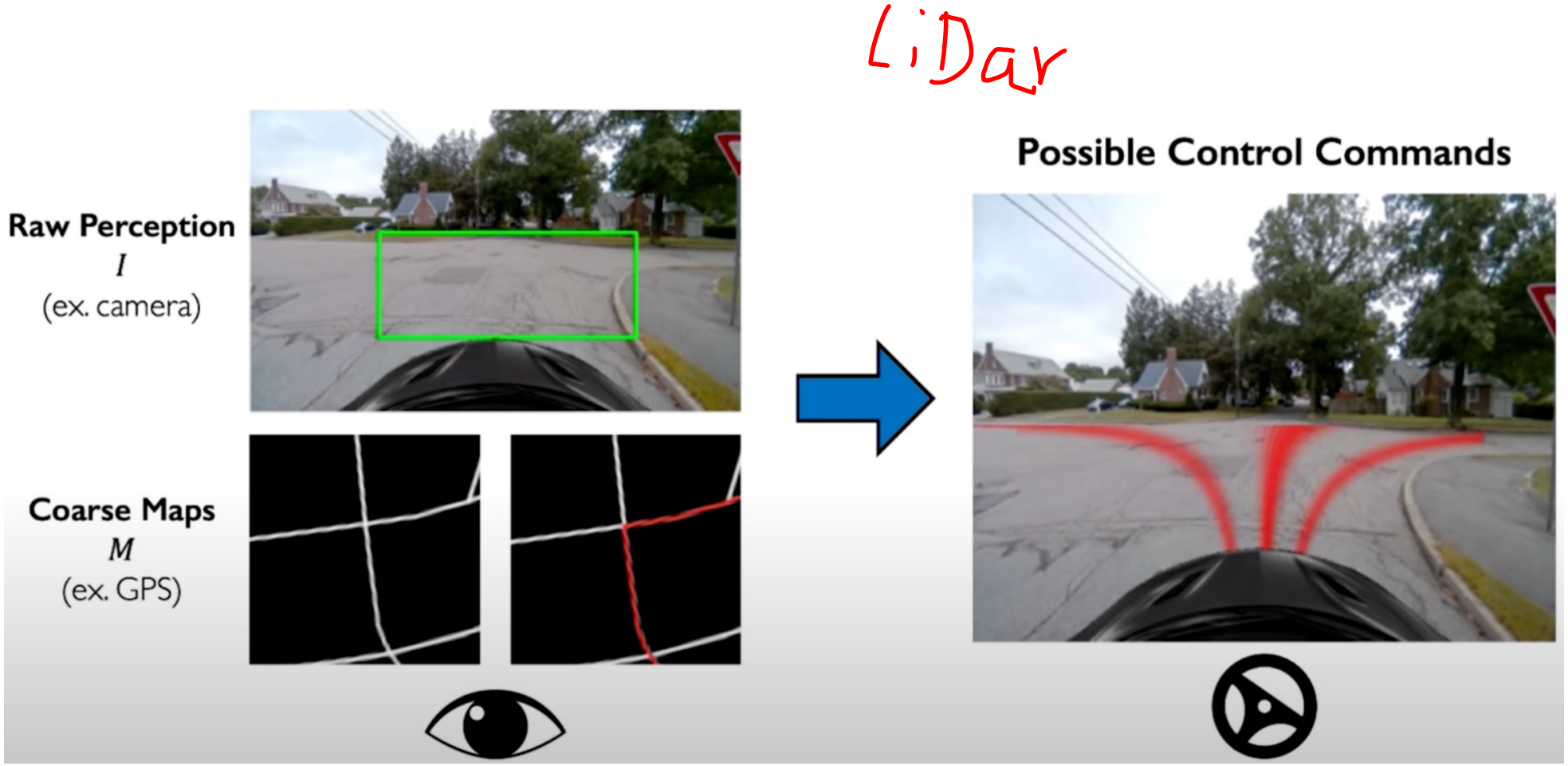


Instance segmentation

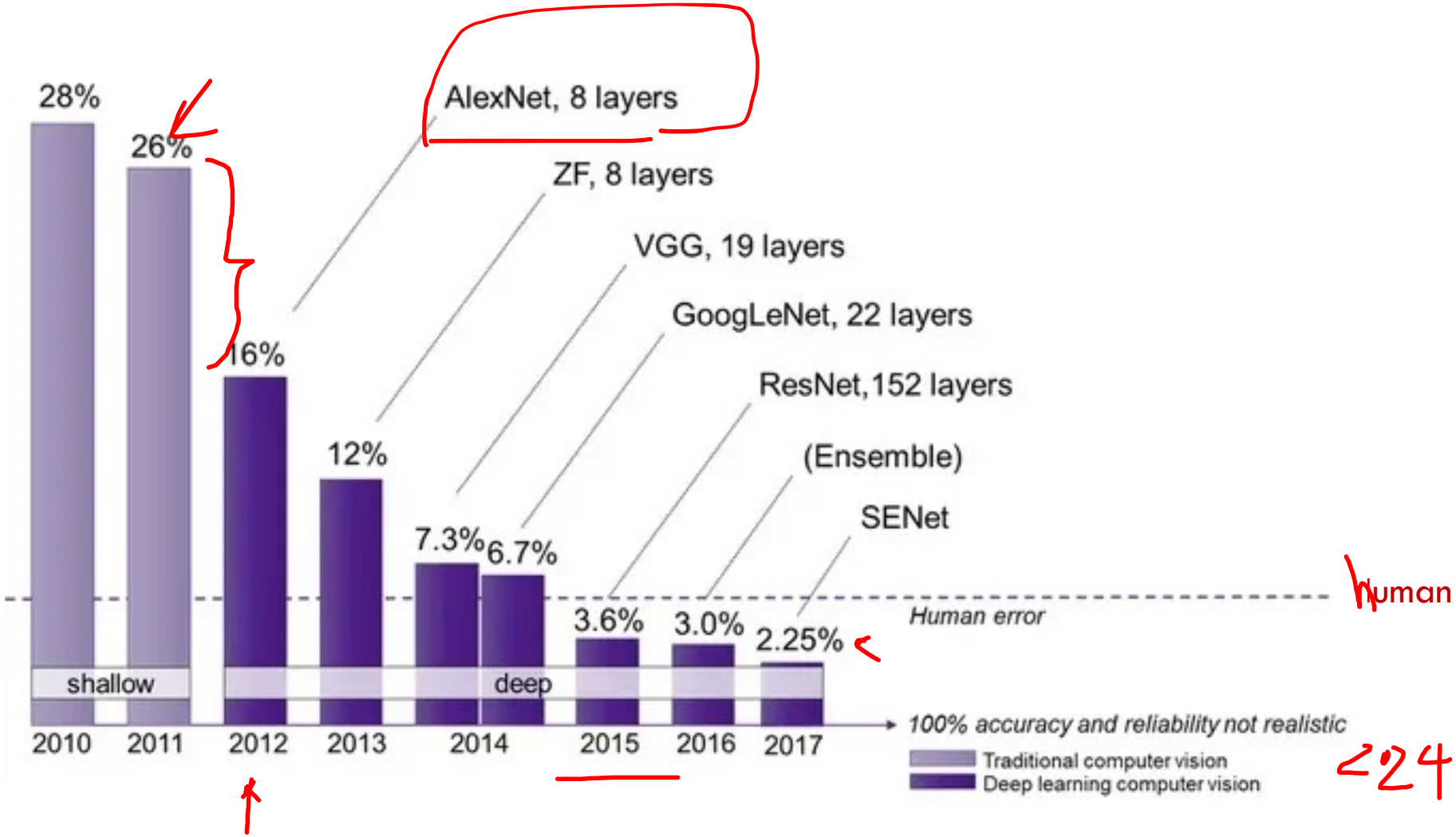
(objects)



Robotic control



Anecdote: ImageNet competition (ILSVRC)



human error is about 5.1%

224

Now let's build our own CNN in Pytorch

Code demo

Drawer slide

Output dimension = (H – F + 2P) / S + 1

W: width & height

F: filter size

P: padding

S: stride