

Advanced School in Artificial Intelligence

Introduction to Convolutional Neural Networks

Ing. Zese Riccardo
riccardo.zese@unife.it

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021



**Università
degli Studi
di Ferrara**

Outline

- Introduction to Python
- Introduction to Neural Networks
- Convolutional NN
- Recurrent NN
- Autoencoders and self supervised learning

Outline

- Introduction to Python
- Introduction to Neural Networks
- Convolutional NN
- Recurrent NN
- Autoencoders and self supervised learning

Motivation – Spatially Structured Data

- When data has spatial structure (topological structure, e.g., images) → **convolutional**
- There are relations among inputs variables → Images have intrinsic properties which could and should be taken advantage of in the models
- CNNs are a specialized kind of neural network for processing data that has a known, grid-like topology.
 - Time-series data, which can be thought of as a 1D grid taking samples at regular time intervals
 - Image data, which can be thought of as a 2D grid of pixels.

Motivation - Images



Image data has important structures:

- Topology of pixels
 - Variables (i.e. pixels) have a significant natural (spatial component) "topology". This makes images different, for example, from the prediction of consumption of a good where the variables do not have a natural topology.
- Translation invariance
- Issues of lighting and contrast
- Knowledge of human visual system
 - CNNs are based on what we know about the structure of images and also what we know about the human visual system. The human visual system has "receptive fields" which respond to horizontal bars, vertical bars, etc.
- Nearby pixels tend to have similar values.
- Edges and shapes.
- Scale Invariance.
 - Objects may appear at different sizes in the image.


Motivation - Images

- Fully connected networks would require a vast number of parameters
 - MNIST images are small (28 x 28 pixels) and in grayscale
 - Color images are more typically at least (224 x 224) pixels x 3 color channels (RGB) = 1,101,760 values.
 - A single fully connected layer would require $(224 \times 224 \times 3)^2 = 1,101,760^2$ weights!
- From a dimensionality point of view, CNNs have been introduced in order to have much fewer parameters to tune.
 - We still have a TON of parameters, but better than tons of tons of parameters.

Motivation - Images

- Variance (in terms of bias-variance) would be too high.
- So we introduce “bias” by structuring the network to look for certain kinds of patterns.
- Idea: expect to see the same basic features anywhere in the image.
 - If we have images with a cats, they can be everywhere in the image, but we just need to identify the cats, which have all same (or similar) features.
- So don't learn different things for each location in the image (at least in the early layers).

Motivation - Images

- Features need to be “built up” 
 - Edges → shapes → relations between shapes
Cat = two eyes in certain relation to one another + cat fur texture.
Eyes = dark circle (pupil) inside another circle.
Circle = particular combination of edge detectors.
Fur = edges in certain pattern.
- Textures
- We don't input these “features”, network learns them.

What to do!



- We do not input features, but images.
- What we must do is simplify the network's job by not inputting all the pixels to fully connected layers.
- Instead, we input a **convoluted** version of that.
 - Use of **kernels**
- In its most general form, convolution is an operation on two functions of a real-valued argument

Kernels

- A **kernel** is a grid of weights **overlaid** on image, **centered** on one pixel such that
 - Each weight is multiplied with the pixel underneath it
 - Output over the centered pixel is $\sum_{p=1}^P W_p \cdot pixel_p$
- Convolution operation is typically denoted with an asterisk $x * w$
- Kernels have been used for traditional image processing techniques also before their application to NN:
 - Blur
 - Sharpen
 - Edge detection
 - Emboss

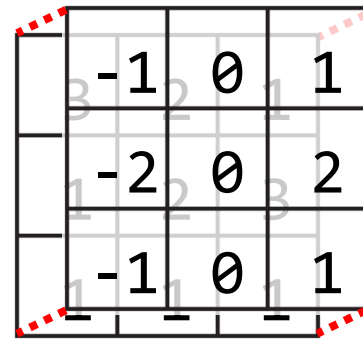
Example: Kernel 3x3

Input		
3	2	1
1	2	3
1	1	1

Kernel		
-1	0	1
-2	0	2
-1	0	1

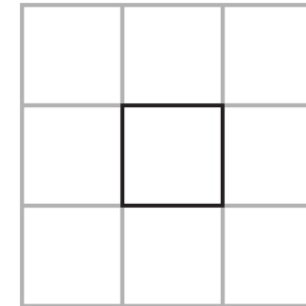
Output		

Example: Kernel 3x3



	-1	0	1	
	-2	0	2	
	-1	0	1	

Output



Example: Kernel 3x3

Input			Kernel			Output		
3	2	1	-1	0	1			
1	2	3	-2	0	2		2	
1	1	1	-1	0	1			



So the output is just the dot product of entries, and the output in this example is 2. We captured the essence of those 9 pixels.

$$\begin{aligned} &= (3 \cdot -1) + (2 \cdot 0) + (1 \cdot 1) \\ &+ (1 \cdot -2) + (2 \cdot 0) + (3 \cdot 2) \\ &+ (1 \cdot -1) + (1 \cdot 0) + (1 \cdot 1) \end{aligned}$$

$$= -3 + 1 - 2 + 6 - 1 + 1 = 2$$

Kernel as Feature Detection

- Can think of kernels as a "local feature detectors"



Vertical Line
Detector

-1	1	-1
-1	1	-1
-1	1	-1

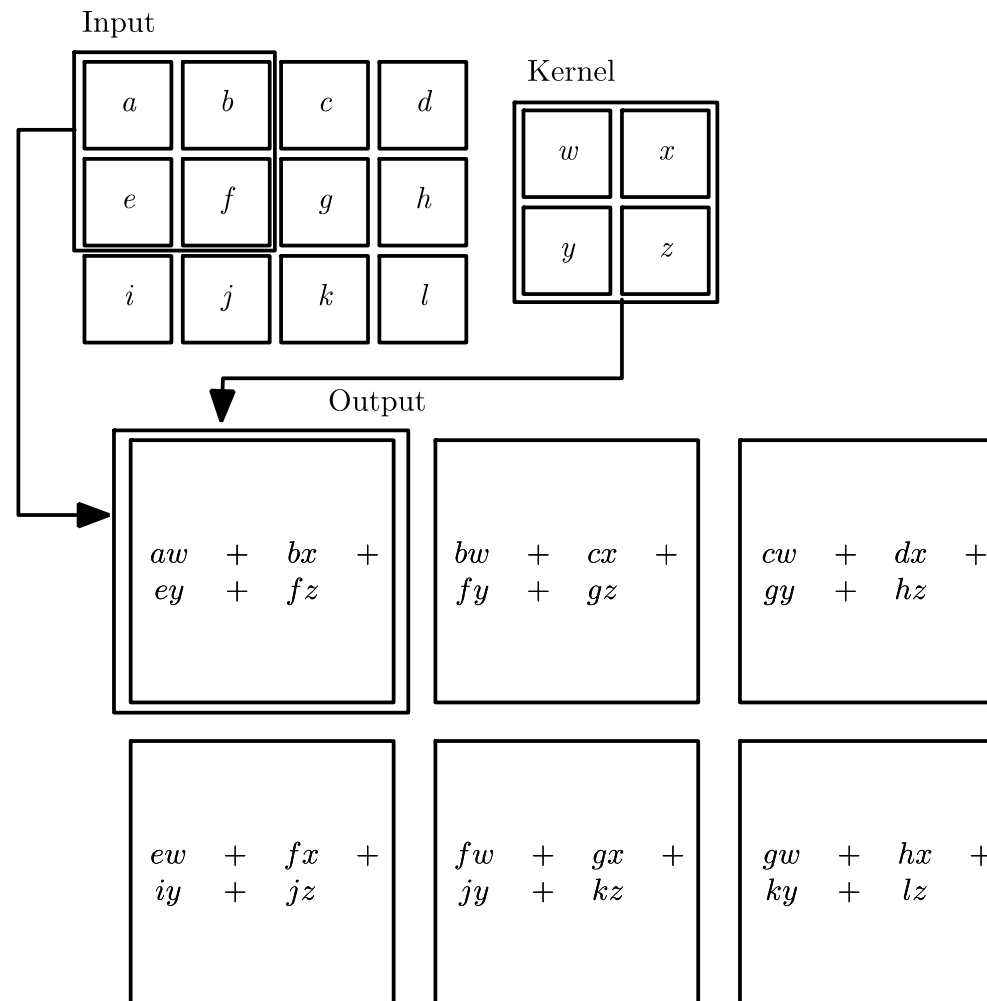
Horizontal Line
Detector

-1	-1	-1
1	1	1
-1	-1	-1

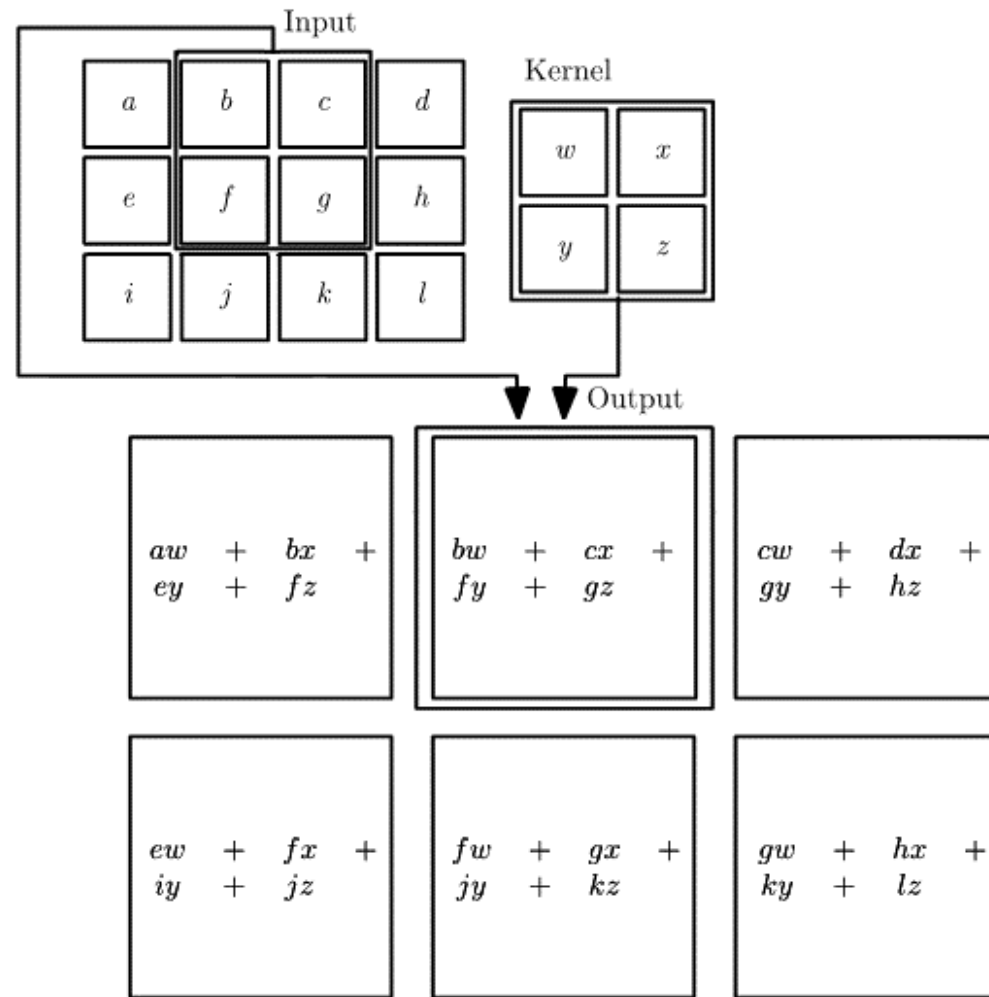
Corner
Detector

-1	-1	-1
-1	1	1
-1	1	1

Kernel



Kernel



Motivation

- Convolution is based on three important ideas that can help improve a machine learning system:
 - **Sparse interactions**
 - **Parameter sharing**
 - **Equivariant representations**

Motivation

- Convolution is based on three important ideas that can help improve a machine learning system:
 - **Sparse interactions** accomplished by making the kernel smaller than the input
 - We need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency.
 - Computing the output requires fewer operations.
 - **Parameter sharing**
 - **Equivariant representations**

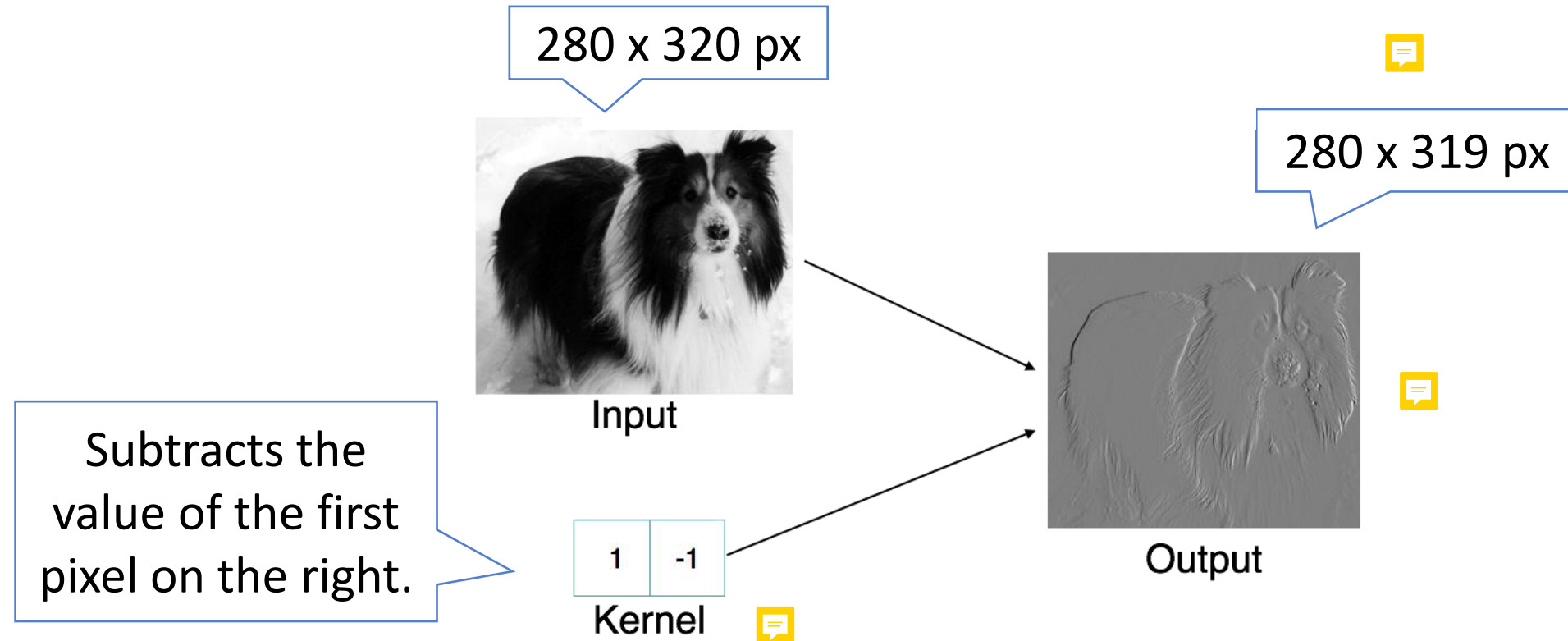
Motivation

- Convolution is based on three important ideas that can help improve a machine learning system:
 - **Sparse interactions**
 - **Parameter sharing** also said **tied weights**, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere.
 - Each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels, depending on the design decisions regarding the boundary).
 - Rather than learning a separate set of parameters for every location we learn only one set.
 - **Equivariant representations**

Motivation

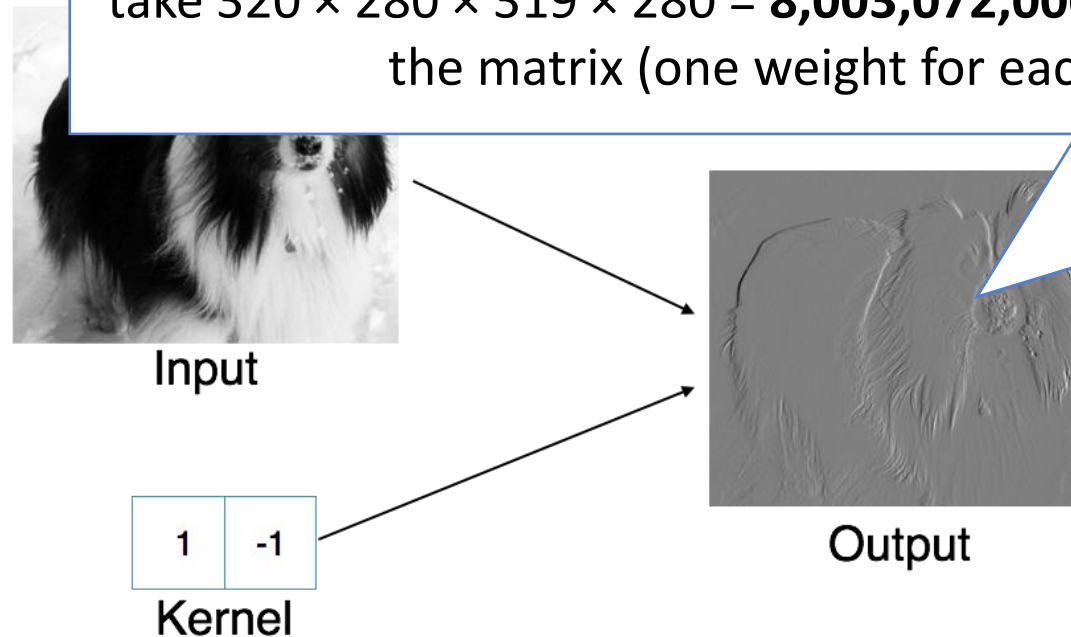
- Convolution is based on three important ideas that can help improve a machine learning system:
 - **Sparse interactions**
 - **Parameter sharing**
 - **Equivariant representations** due to the particular form of parameter sharing adopted.
 - A function is equivariant means that if the input changes, the output changes in the same way.
 - If we shift an image of e.g. 1 pixel in a direction we have two images whose convolution result is equivalent.
 - Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations.

Edge Detection by Convolution



Edge Detection by Convolution



This transformation requires $319 \times 280 \times 3 = \mathbf{267,960}$ floating point operations (two multiplications and one addition per output pixel). To describe the same transformation with a matrix multiplication would take $320 \times 280 \times 319 \times 280 = \mathbf{8,003,072,000}$ (over eight billion) entries in the matrix (one weight for each pair pixel-pixel).



Edge Detection by Convolution

This transformation requires $319 \times 280 \times 3 = \mathbf{267,960}$ floating point operations (two multiplications and one addition per output pixel). To describe the same transformation with a matrix multiplication would take $320 \times 280 \times 319 \times 280 = \mathbf{8,003,072,000}$ (over eight billion) entries in the matrix.

We keep in memory only the nonzero weights $\rightarrow 2$ for each pixel.

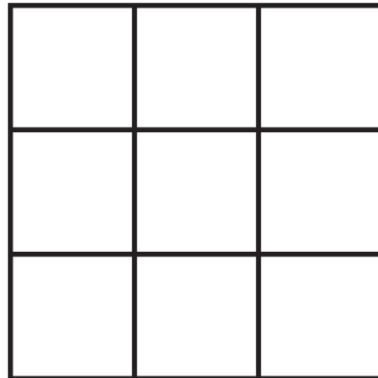


	Convolution	Dense Matrix	Sparse Matrix
Stored floats	2	$320 \times 280 \times 319 \times 280$	$2 \times 319 \times 280$
Float muls and adds	$319 \times 280 \times 3 = 267,960$	$> 16e9$	Same as convolution (267,960)

Convolution Settings

- **Grid Size (Height and Width):**
 - The number of pixels a kernel sees at once
 - Typically use odd numbers so that there is a **center** pixel
 - Kernel does not need to be square

Height: 3,
Width: 3



Height: 1,
Width: 3



Height: 3,
Width: 1



Convolution Settings

- **Padding**
 - Using Kernels directly, there will be an **edge effect**.
 - Pixels near the edge will not be used as **center pixels** since there are not enough surrounding pixels.
 - Padding adds extra pixels around the frame.
 - So every pixel of the original image will be a center pixel as the kernel moves across the image.
 - Added pixels are typically of value zero (**zero-padding**).

Without Padding

We calculate the upper left entry of the output by multiplying the kernel against the upper left 3x3 patch of the input (and summing).

1	2	0	3	1
1	0	0	2	2
2	1	2	1	1
0	0	1	0	0
1	2	1	1	1

input

-1	1	2
1	1	0
-1	-2	0

kernel

-2		

output



With Padding

0	0	0	0	0	0	0
0	1	2	0	3	1	0
0	1	0	0	2	2	0
0	2	1	2	1	1	0
0	0	0	1	0	0	0
0	1	2	1	1	1	0
0	0	0	0	0	0	0

input

-1	1	2
1	1	0
-1	-2	0

kernel

-1				

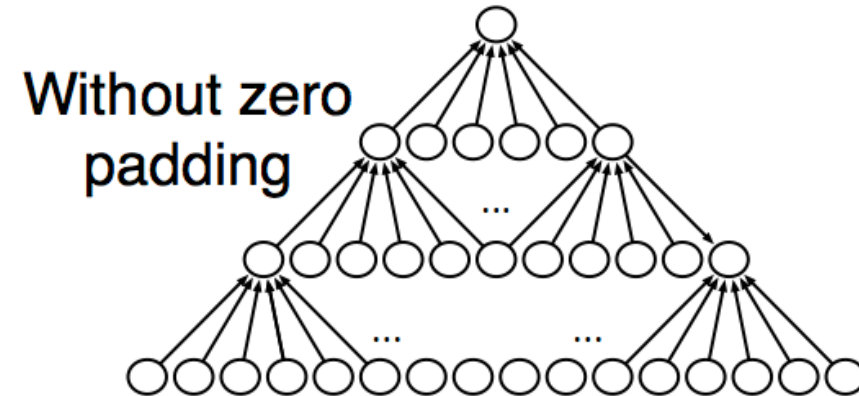
output

Note the different size. If there is no padding, the output will be smaller than the input since we **lose** a little at the boundary.

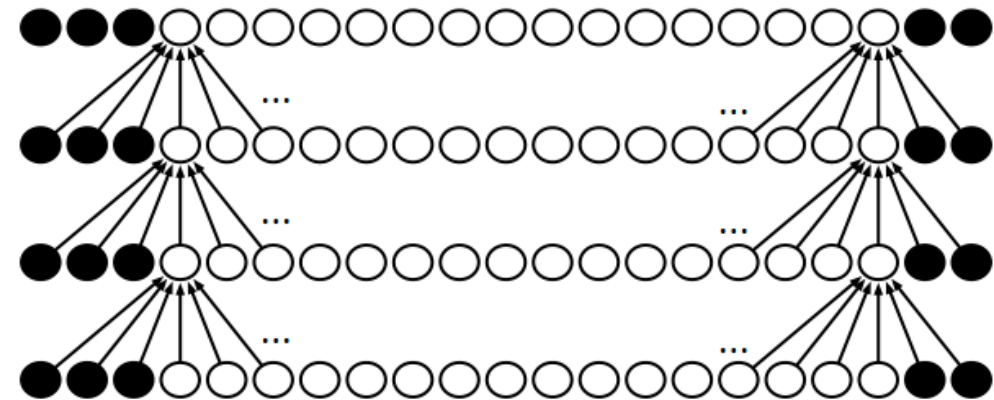
Convolution and Padding

Without padding, the width of the representation shrinks by one pixel less than the kernel width at each layer.

Without zero padding, we are forced to choose between shrinking the spatial extent of the network rapidly and using small kernels (to reduce the speed of shrinking) both significantly limiting the expressive power of the network.

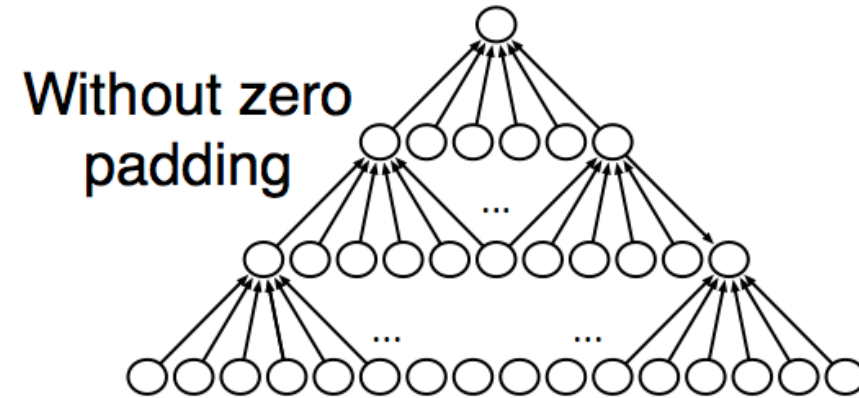


With zero padding

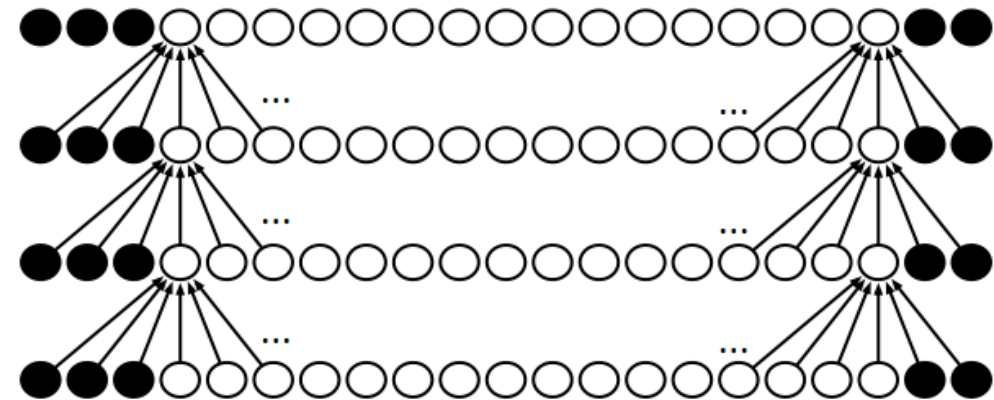


Convolution and Padding

Zero padding the input allows the control of the kernel width and the size of the output independently.



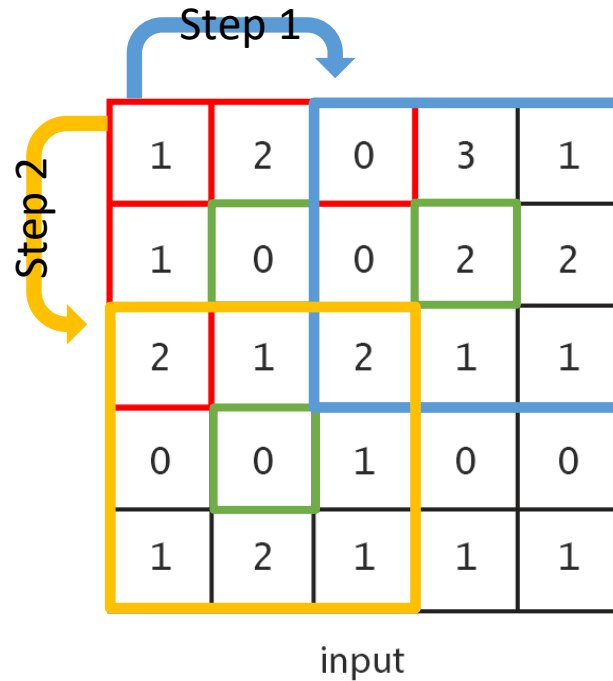
With zero padding



Convolution Settings

- **Stride**
 - The **step size** as the kernel moves across the image
 - Can be different for vertical and horizontal steps (but usually is the same value)
 - When stride is greater than 1, it **scales down** the output dimension
→ **Smaller stride = higher resolution of output image**

Stride 2 – No Padding



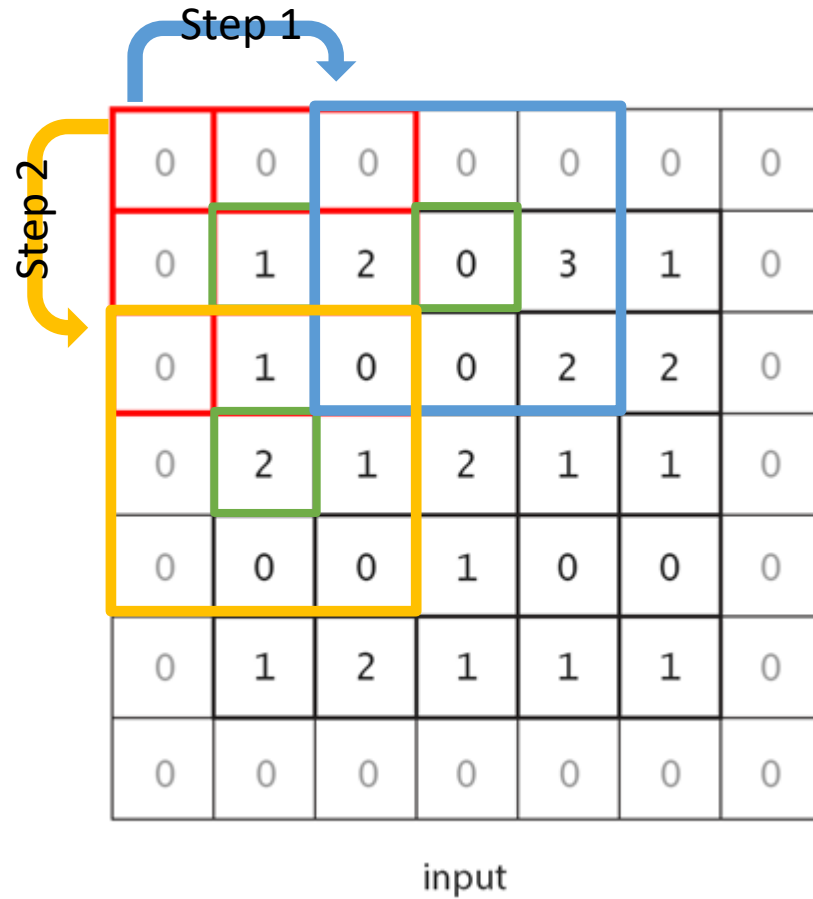
-1	1	2
1	1	0
-1	-2	0

kernel

-2	3
0	

output

Stride 2 - With Padding



We can have strides and padding at the same time too.
Notice how the output size depends on these parameters.

-1	1	2
1	1	0
-1	-2	0

kernel

-1	2	
3		

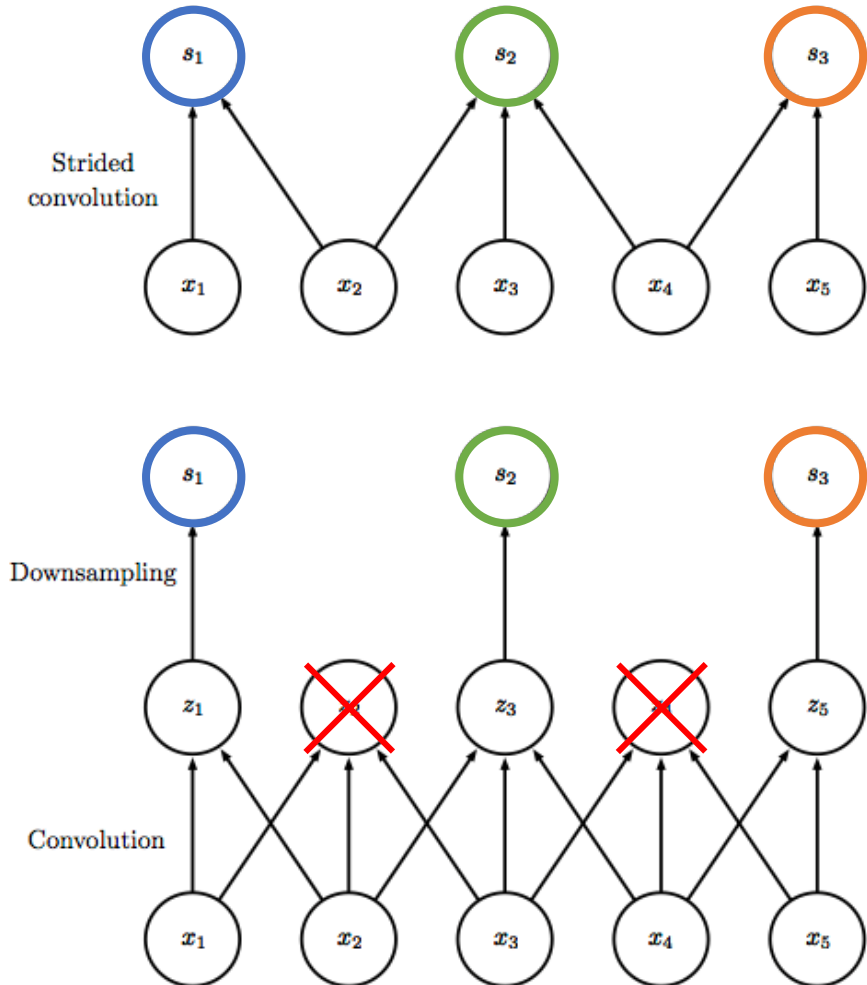
output

Convolution with Stride

- Stride acts like **downsampling** but more efficient

Convolution with a stride greater than one pixel is **mathematically equivalent** to convolution with unit stride followed by downsampling.

Obviously, the two-step approach involving downsampling is **computationally wasteful**, because it computes many values that are then discarded.



Convolutional Settings

- **Depth**
 - In images, we often have multiple numbers associated with each pixel location.
 - These numbers are referred to as **channels**
 - RGB image – 3 channels
 - CMYK – 4 channels
 - The number of channels is referred to as the **depth**
 - So the kernel itself will have a depth the same size as the number of input channels.
 - Example: a 5x5 kernel on an RGB image
 - There will be $5 \times 5 \times 3 = 75$ weights

Convolutional Settings




- **Depth**

- The kernel will look at the entire depth of the input.
- On first layer, the depth will be just RGB or CMYK, but for later layers the input depth may be quite high, so the kernels will have a lot of weights.
 - The output from the layer will also have a depth
- The networks typically train many different kernels
- Each kernel outputs a single number at each pixel location
 - So if there are 10 kernels in a layer, the output of that layer will have depth 10.

Pooling

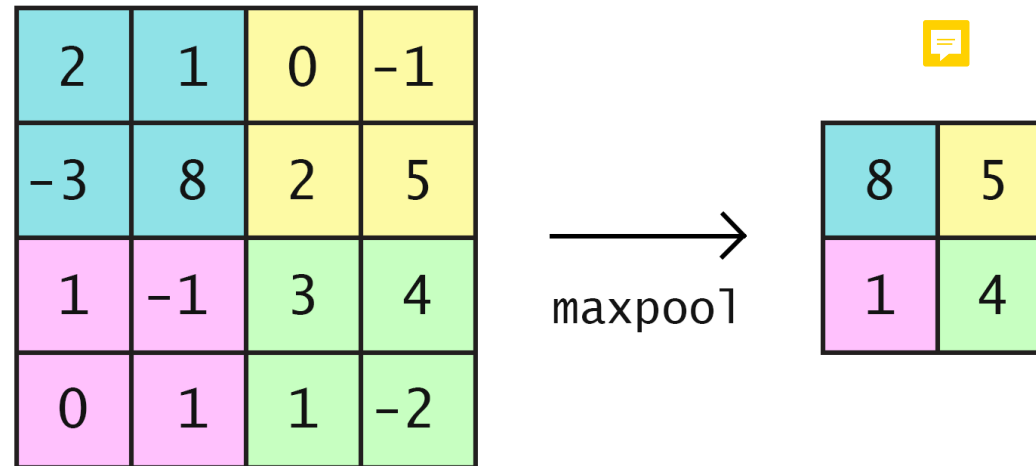
- If an image is very large, the network would have a huge number of numbers to store.
- Idea: Reduce the image size by mapping a patch of pixels to a single value.
- Shrinks the dimensions of the image.
- Does not have parameters or weights to learn, though there are different types of pooling operations.

Pooling

- A typical layer of a convolutional network consists of three stages:
 1. The layer performs several convolutions in parallel to produce a set of linear activations.
 2. Each linear activation is run through a nonlinear activation function, such as ReLU. This stage is sometimes called the **detector** stage.
 3. We use a pooling function to modify the output of the layer further. 
- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.

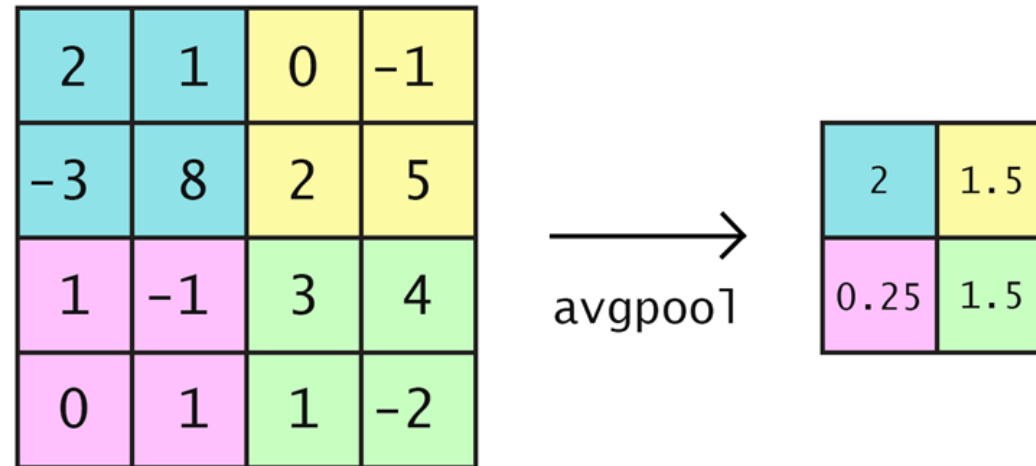
Pooling: Max Pooling

- For each distinct patch, represent it by the **maximum**.
- 2x2 max pooling shown below



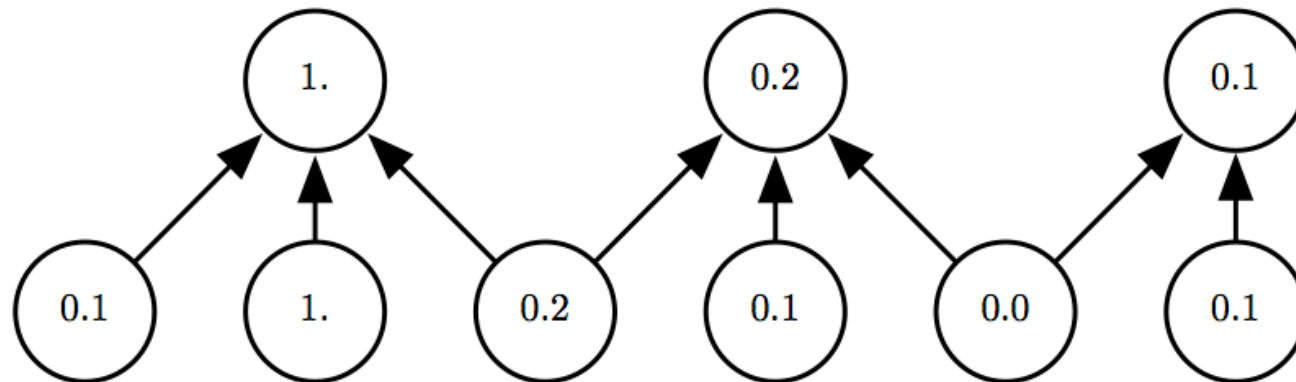
Pooling: Average Pooling

- For each distinct patch, represent it by the **average**.
- 2x2 max pooling shown below



Pooling with Stride

- Combining pooling with stride of more than 1 unit reduces the representation size by a factor depending of the stride.
 - Given a stride of k , the next layer has roughly k times fewer inputs to process → This improves the computational efficiency of the network.



Pooling



- For many tasks, pooling is essential for handling inputs of varying size, because the input to the classification layer must have a fixed size.
- By varying the size of the stride between pooling regions we can reduce every image to the size wanted by the network.
 - The resulting input will be a summary of the original image as each input “pixel” contains information about a set of neighbouring pixels (max value, average, etc.).