Introduction to Convolutional Neural Networks

Motivation – Image Data

- So far, the structure of our neural network treats all inputs interchangeably.
- No relationships between the individual inputs
- Just an ordered set of variables

- We want to incorporate domain knowledge into the architecture of a Neural Network.

Motivation

- Image data has important structures, such as;

- ”Topology” of pixels
- Translation invariance
- Issues of lighting and contrast
- Knowledge of human visual system
- Nearby pixels tend to have similar values
- Edges and shapes
- Scale Invariance – objects may appear at different sizes in the image.

Motivation – Image Data

- Fully connected would require a vast number of parameters
- MNIST images are small (32 x 32 pixels) and in grayscale
- Color images are more typically at least (200 x 200) pixels x 3 color channels (RGB) = 120,000 values.
- A single fully connected layer would require $(200\text{x}200\text{x}3)^2$ = 14,400,000,000 weights!
- 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

Motivation

- Features need to be “built up”
- Edges -> shapes -> relations between shapes
- Textures

- 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.

CNN Animations

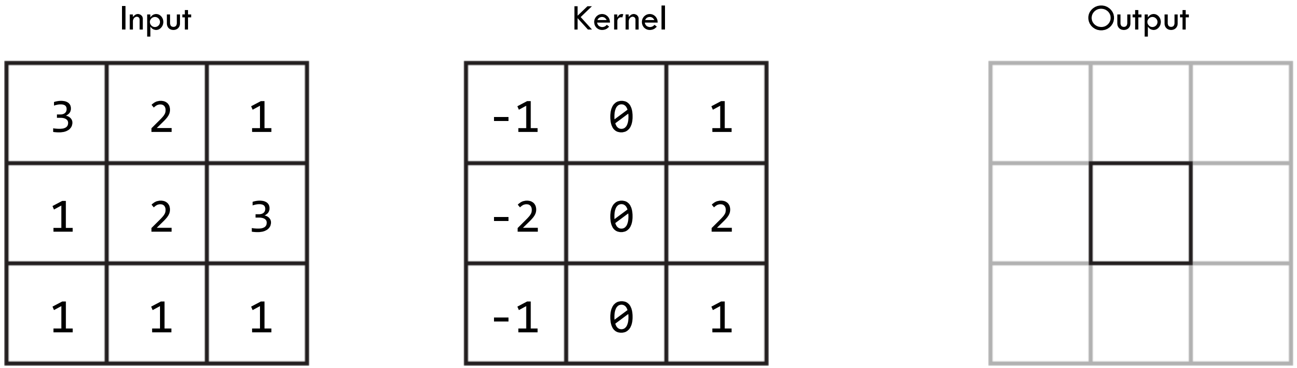
https://adamharley.com/nn_vis/cnn/3d.html

https://hannibunny.github.io/mlbook/neuralnetworks/convolutionDemos.html

Kernels

- A *kernel* is a grid of weights "overlaid" on image, centered on one pixel
- Each weight multiplied with pixel underneath it
- Output over the centered pixel is $\sum_{p=1}^{P} W_p \cdot pixel_p$
- Used for traditional image processing techniques:
  - Blur
  - Sharpen
  - Edge detection
  - Emboss

Kernel: 3x3 Example



Input

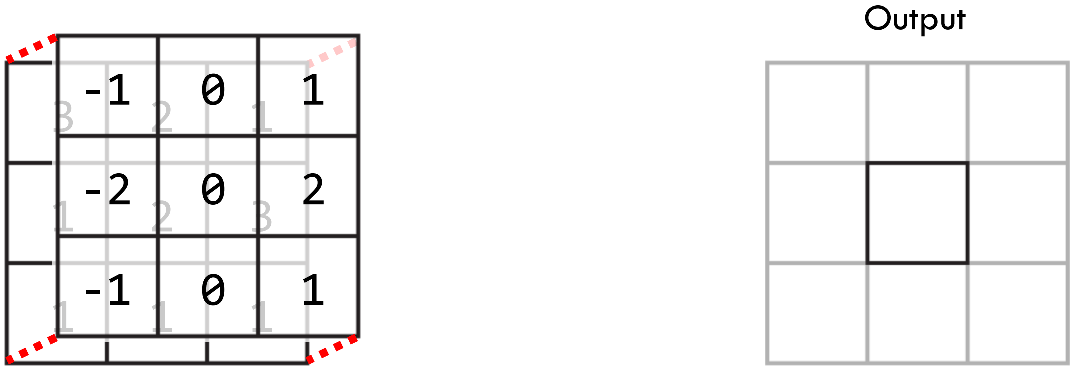
3	2	1
1	2	3
1	1	1

Kernel

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

Output

Kernel: 3x3 Example



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

3	2	1
1	2	3
1	1	1

Kernel: 3x3 Example

Input

3	2	1
1	2	3
1	1	1

Kernel

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

Output

	2	

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

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

Kernels as Feature Detectors

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

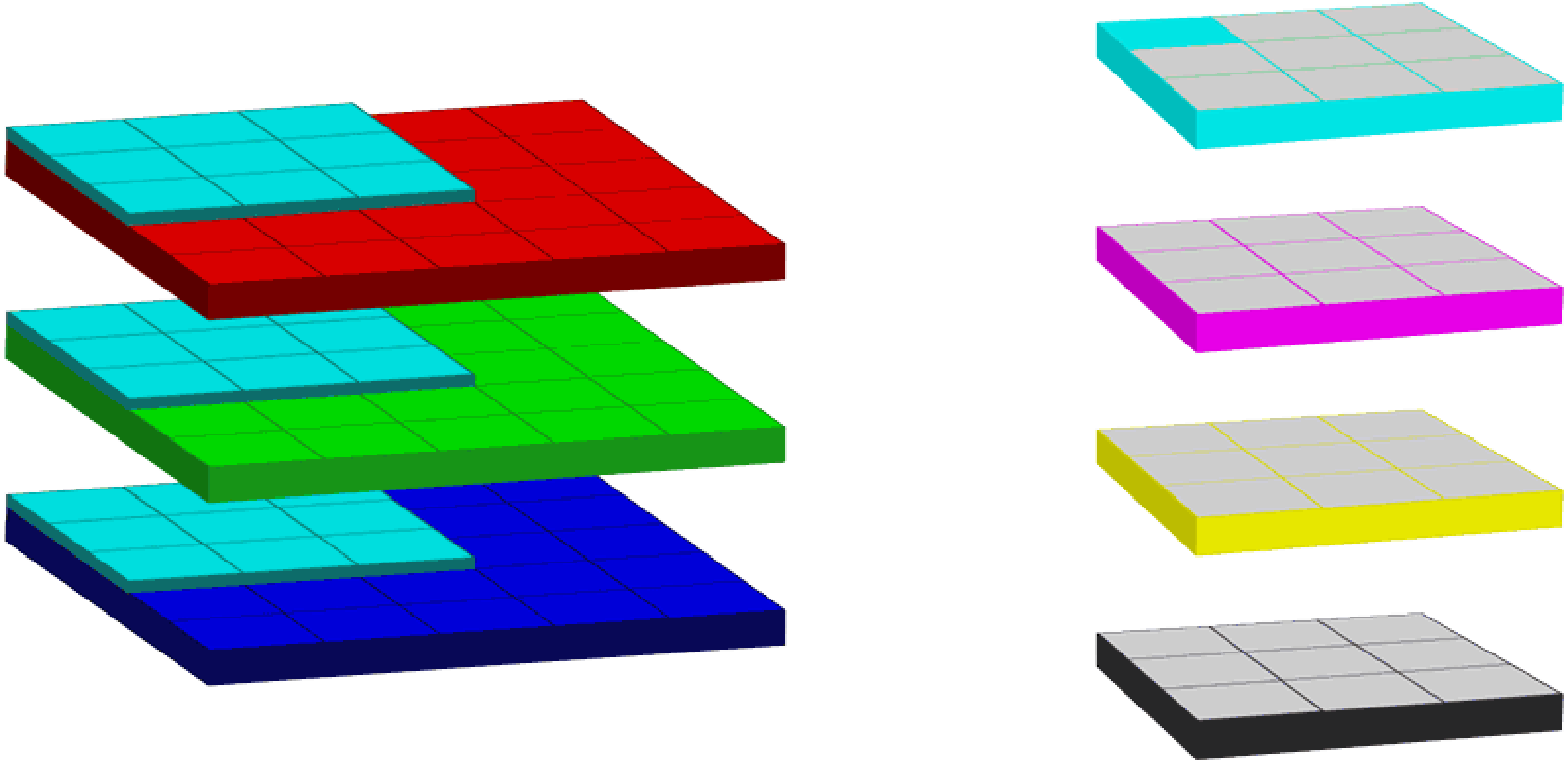
-1	-1	-1
-1	1	1
-1	1	1

Convolutional Neural Nets

Primary Ideas behind Convolutional Neural Networks:

- Let the Neural Network learn which kernels are most useful
- Use same set of kernels across entire image (translation invariance)
- Reduces number of parameters and “variance” (from bias-variance point of view)

Convolutions

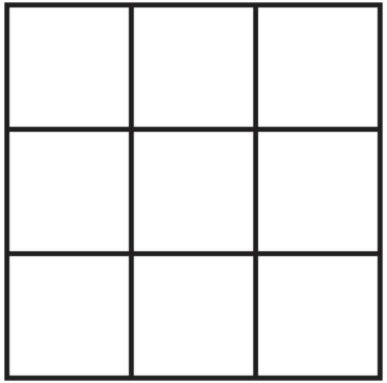


Convolution Settings – Grid Size

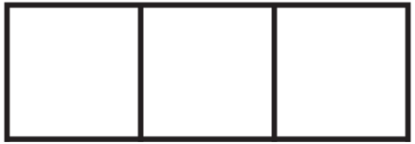
Grid Size (Height and Width):

- 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

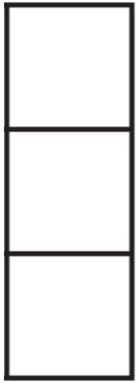
Height: 3, Width: 3



Height: 1, Width: 3



Height: 3, Width: 1

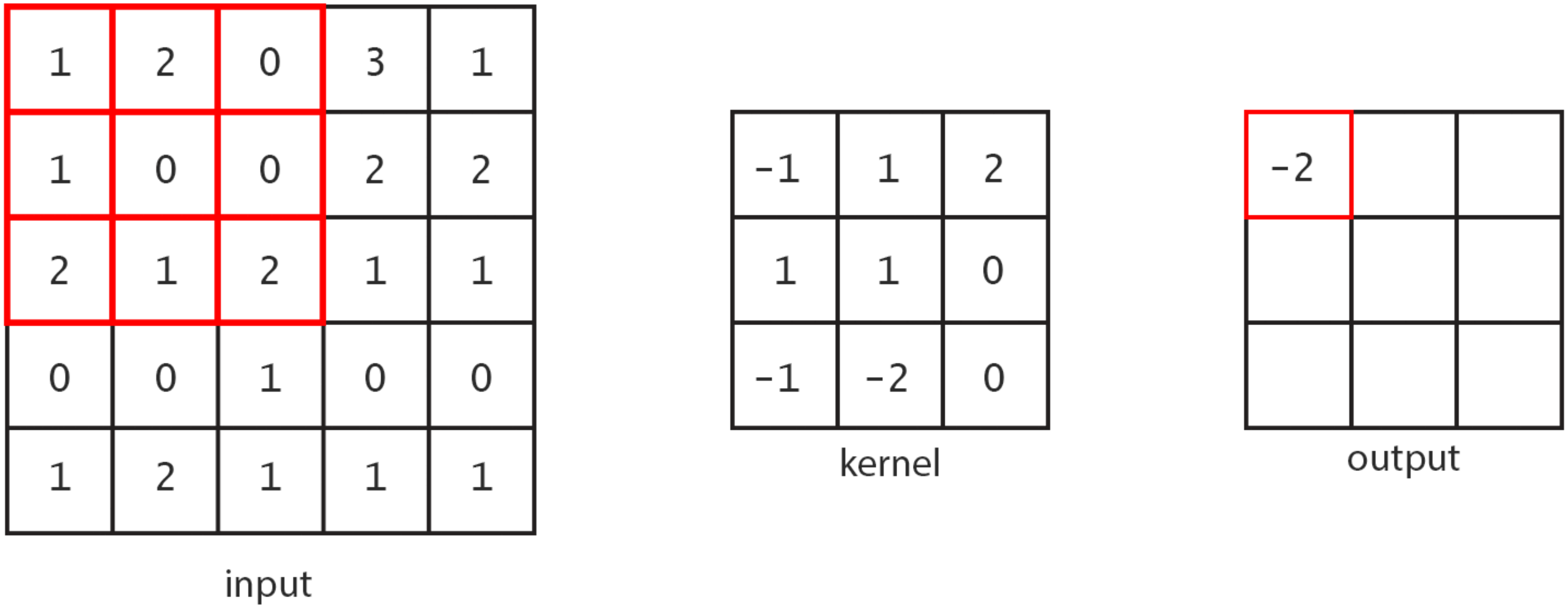


Convolution Settings - Padding

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



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

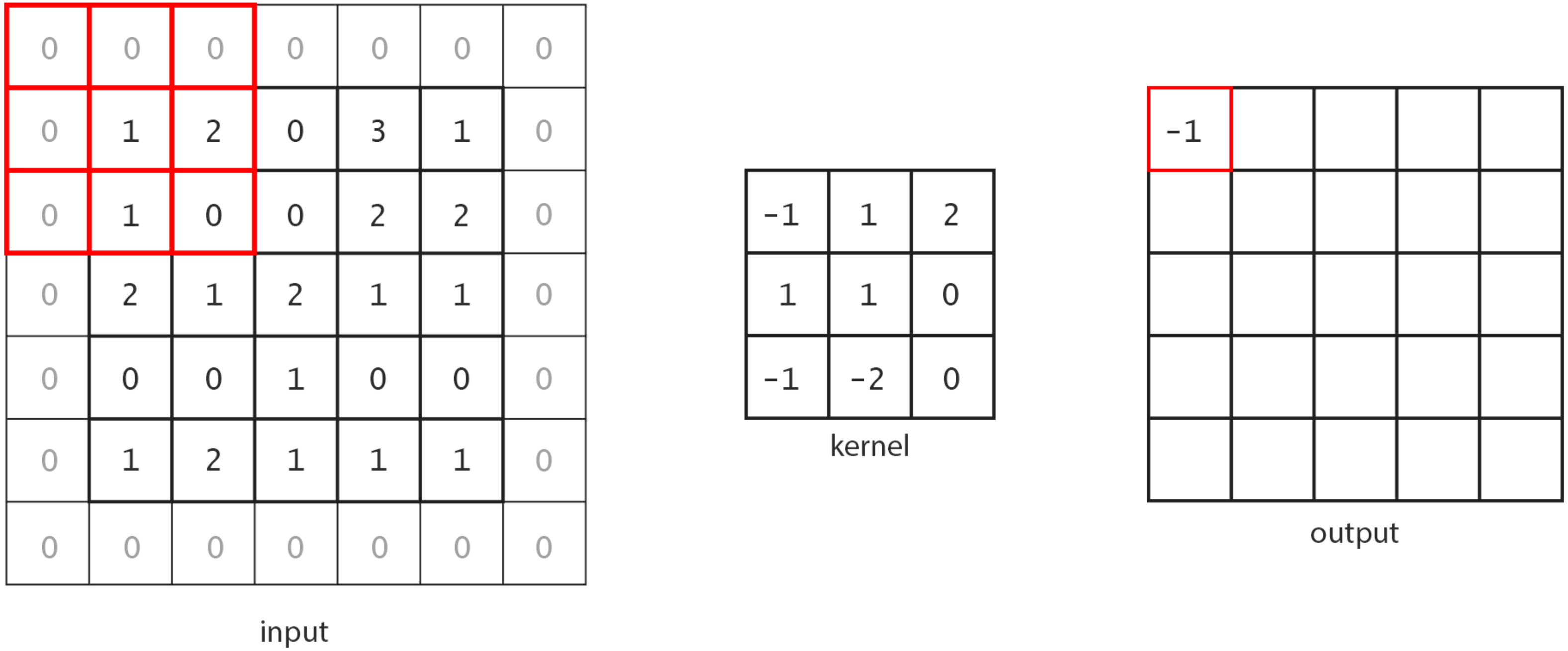
-1	1	2
1	1	0
-1	-2	0

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				

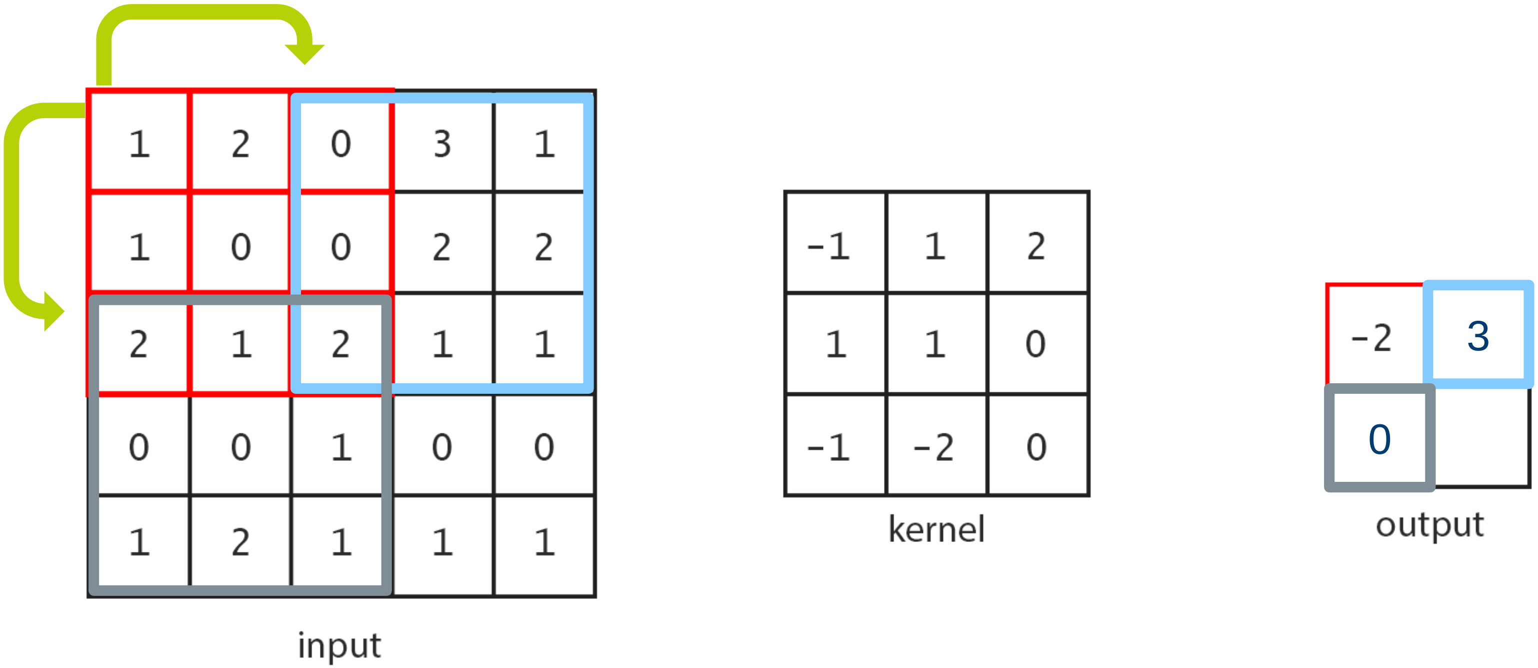
output

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

Stride 2 Example – No Padding



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

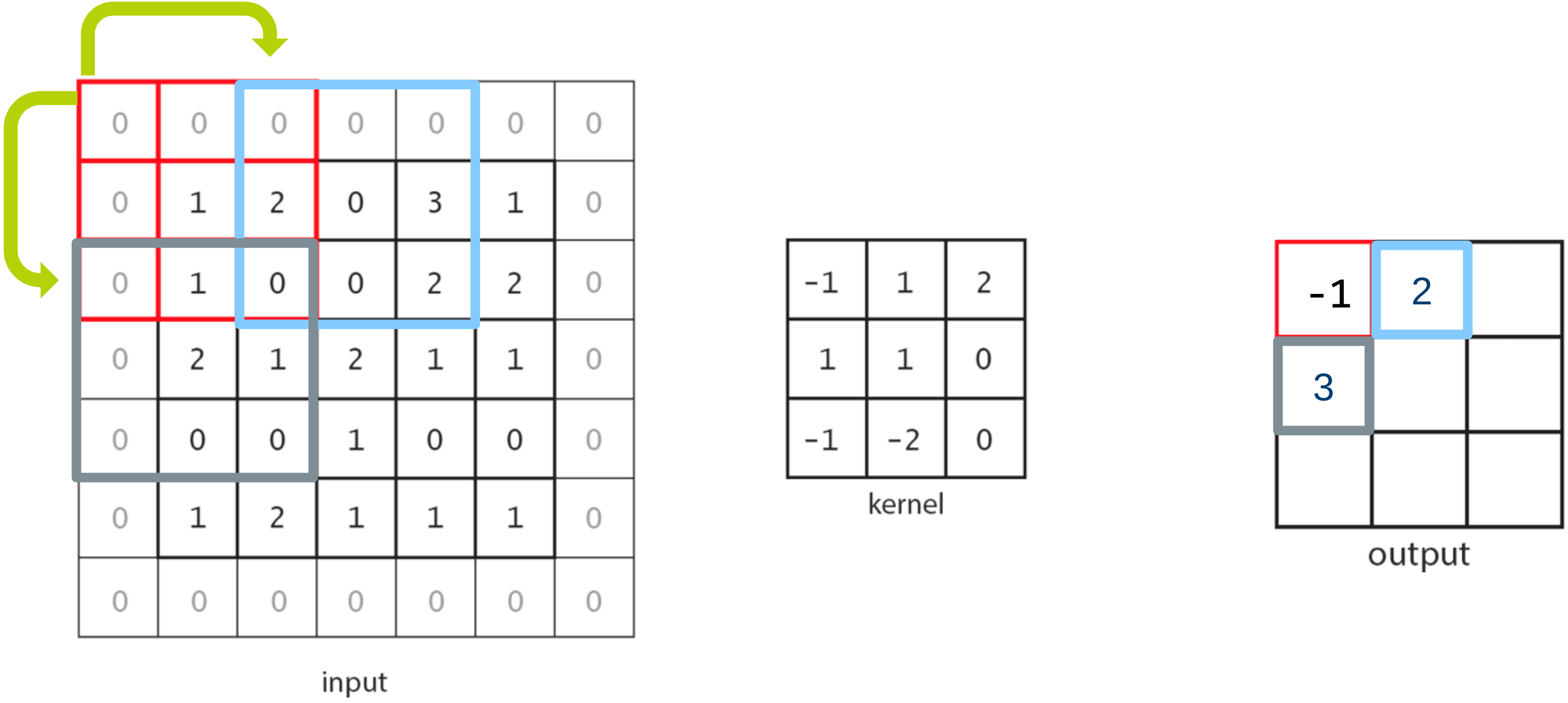
-1	1	2
1	1	0
-1	-2	0

kernel

-2	3
0	

output

Stride 2 Example – 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	2	
3		

output

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 5x5x3 = 75 weights

Convolutional Settings - Depth

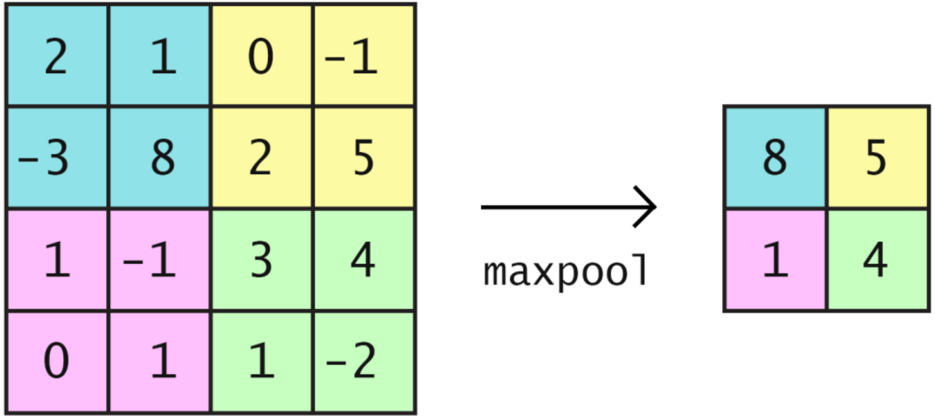
- The output from the layer will also have a depth
- The networks typically train many different kernels
- Each kernel outputs a single number at each pixel location
- So if there are 10 kernels in a layer, the output of that layer will have depth 10.

Pooling

- 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, though there are different types of pooling operations.

Pooling: Max-pool

- For each distinct patch, represent it by the maximum
- 2x2 maxpool shown below

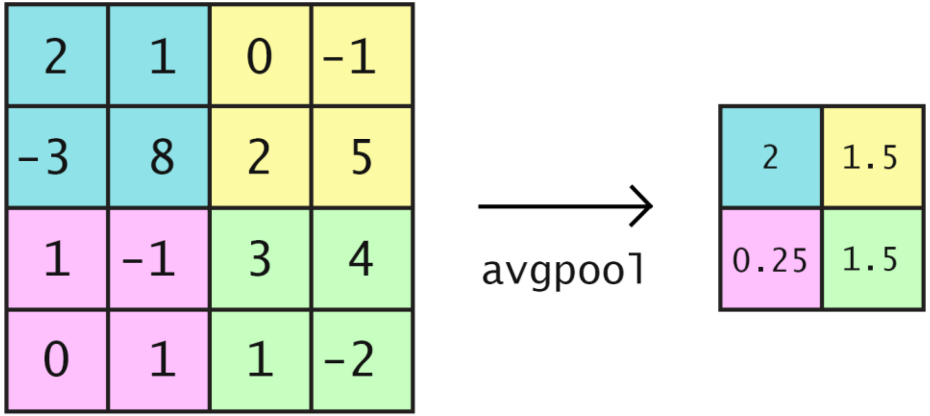


2	1	0	-1
-3	8	2	5
1	-1	3	4
0	1	1	-2

8	5
1	4

Pooling: Average-pool

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



2	1	0	-1
-3	8	2	5
1	-1	3	4
0	1	1	-2

2	1.5
0.25	1.5