Lecture 15: Computer Vision (Part 1)

CIS 4190/5190

Spring 2025

Agenda

- **Computer vision**
  - Prior to deep learning
  - Convolutional layers
  - Convolutional neural networks
  - Feature visualization

Images as 2D Arrays

- Grayscale image is a 2D array of pixel values

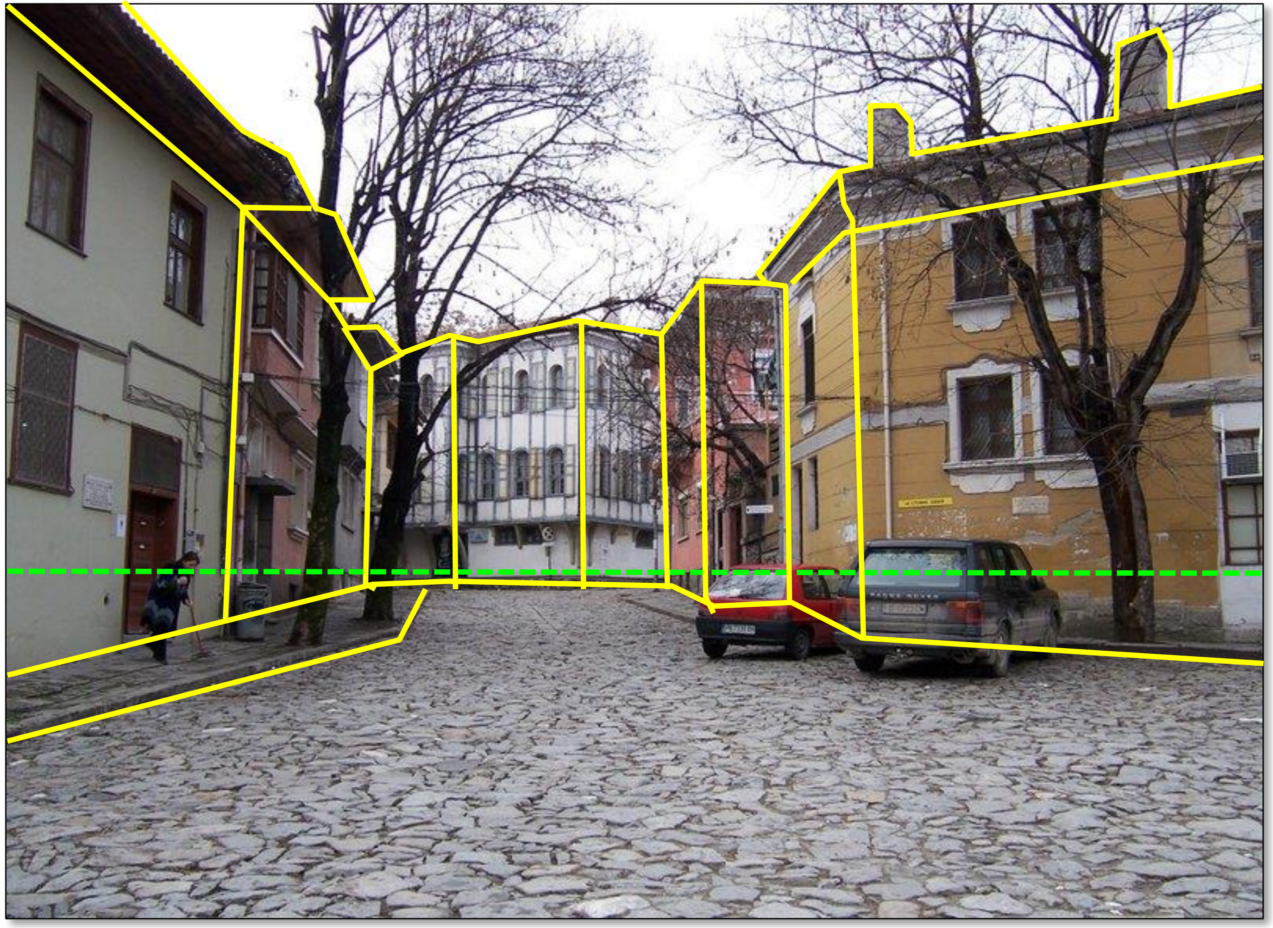
- Color images are 3D array
  - 3rd dimension is color (e.g., RGB)
  - Called "channels"

0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

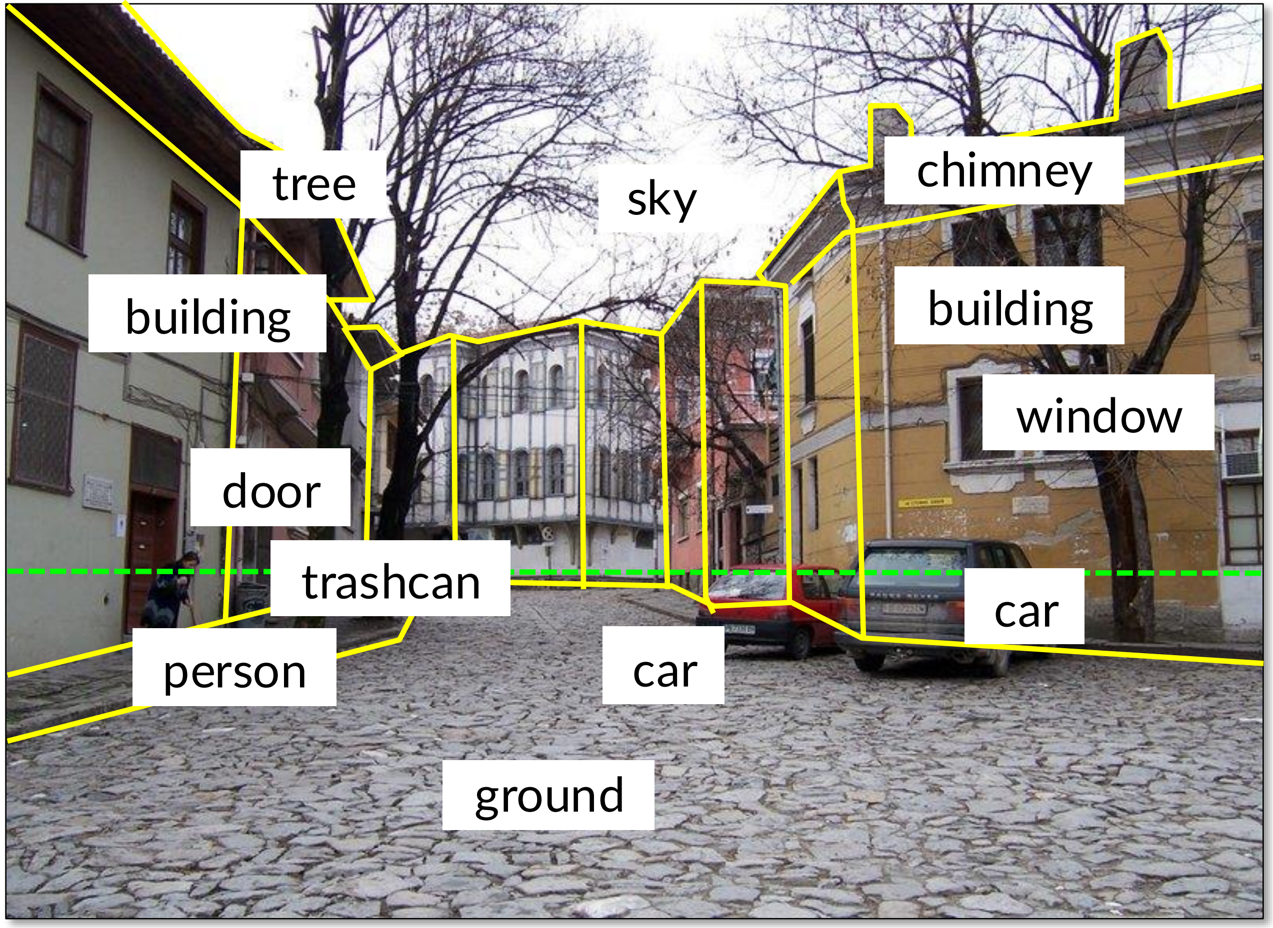
Source: S. Narasimhan, S. Lazebnik

Structure in Images

Structure in Images



Structure in Images



Outdoor scene

City

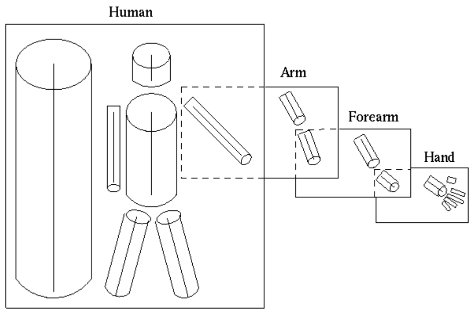
European

History of Computer Vision

- **Deceptively challenging task**
  - In the 1960s, Marvin Minsky assigned some undergrads to program a computer to use a camera to identify objects in a scene
  - Half a century later, we are still working on it

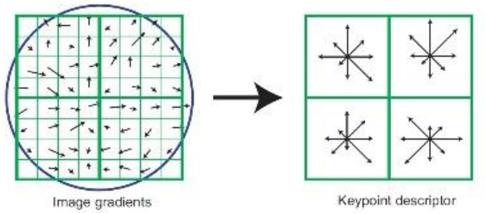
- **Moravec's paradox**
  - Motor and perception skills require enormous computational resources
  - Largely unconscious, biasing our intuition
  - Likely innate to some degree

History of Computer Vision



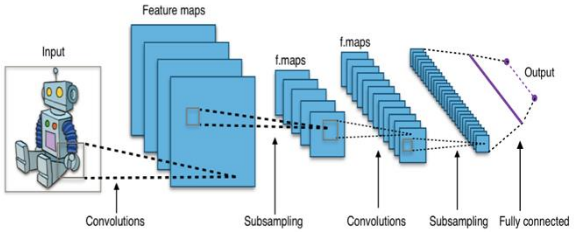
Very **old: 60's – Mid 90's**

Image → hand-def. features → hand-def. classifier



Old: Mid 90's – 2012

Image → hand-def. features → learned classifier

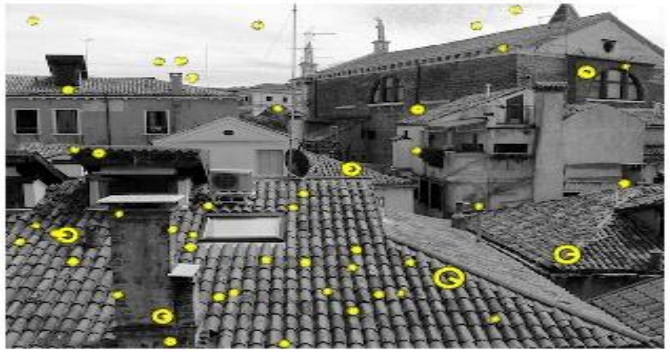


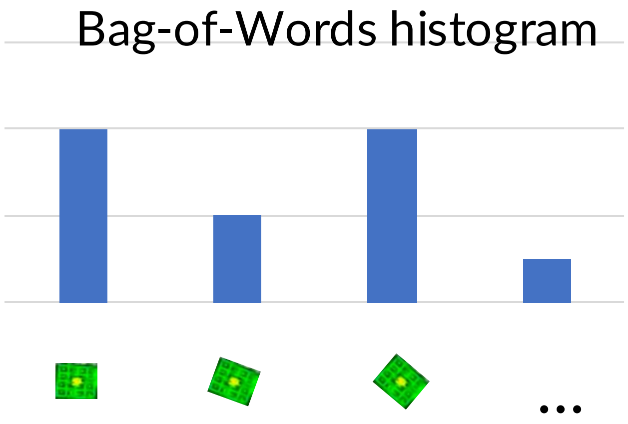
Current: 2012 – Present

Image → jointly learned features + classifier

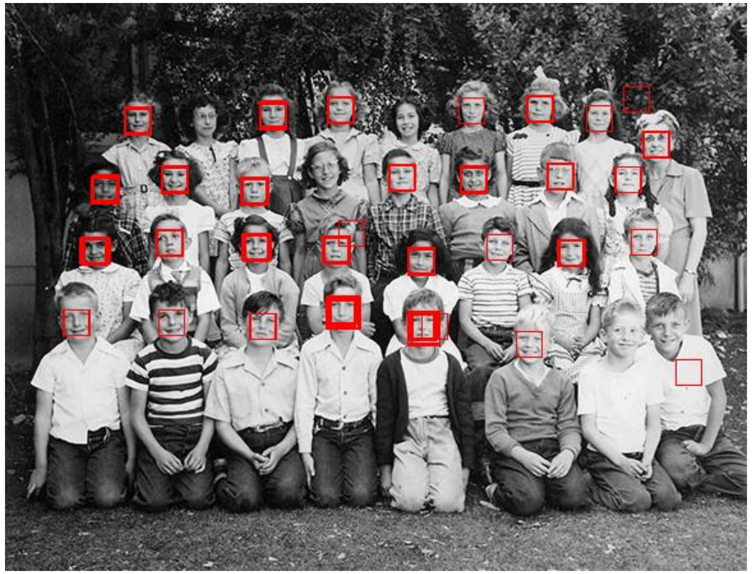
Prior to Deep Learning

- **Step 1:** Find “pixels of interest”
  - E.g., corner points or “difference of gaussians”
- **Step 2:** Compute features at these points
  - E.g., “SIFT”, “HOG”, “SURF”, etc.
- **Step 3:** Convert to feature vector via statistics of features such as histograms
  - E.g., “Bag of Words”, “Spatial Pyramids”, etc.
- **Step 4:** Use standard ML algorithm



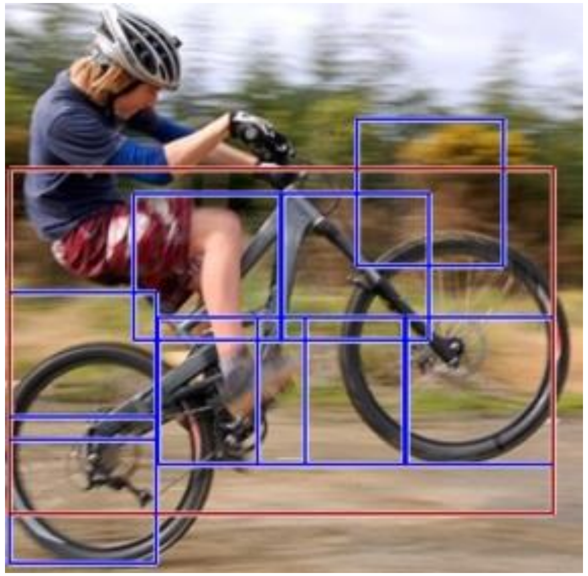


Prior to Deep Learning



https://github.com/alexdemartos/ViolaAndJones

Viola-Jones face detector
(with AdaBoost!)
~2000



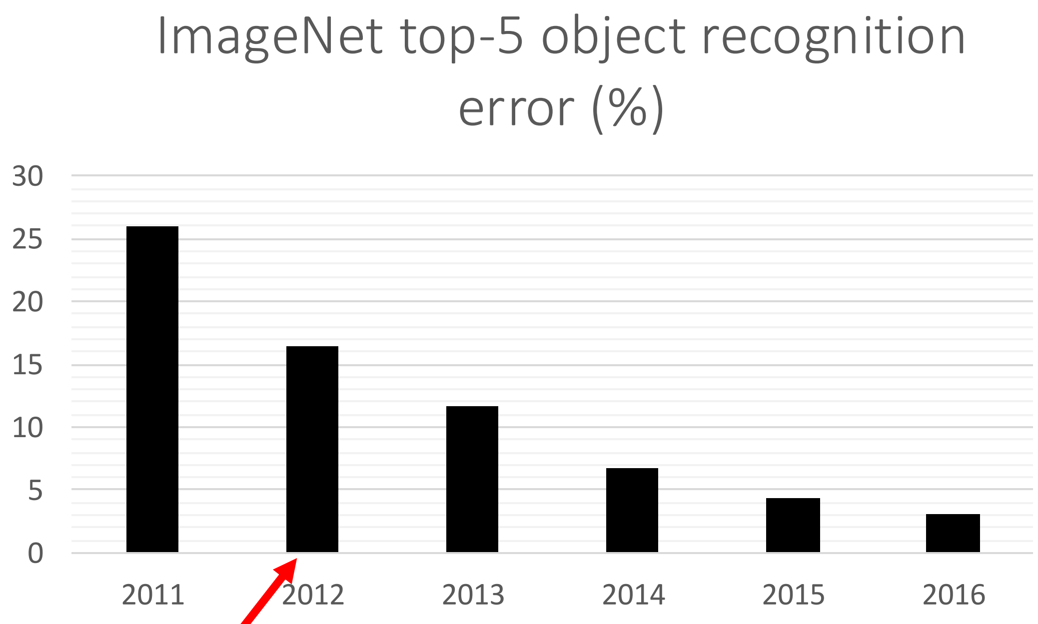
Deformable Parts Model
object detection
(with linear classifiers!)
~2010

GIST
Scene retrieval
(with nearest neighbors!)
~2006

See libraries such as VLFeat and OpenCV

Impact of Deep Learning

ImageNet 1000-object category recognition challenge



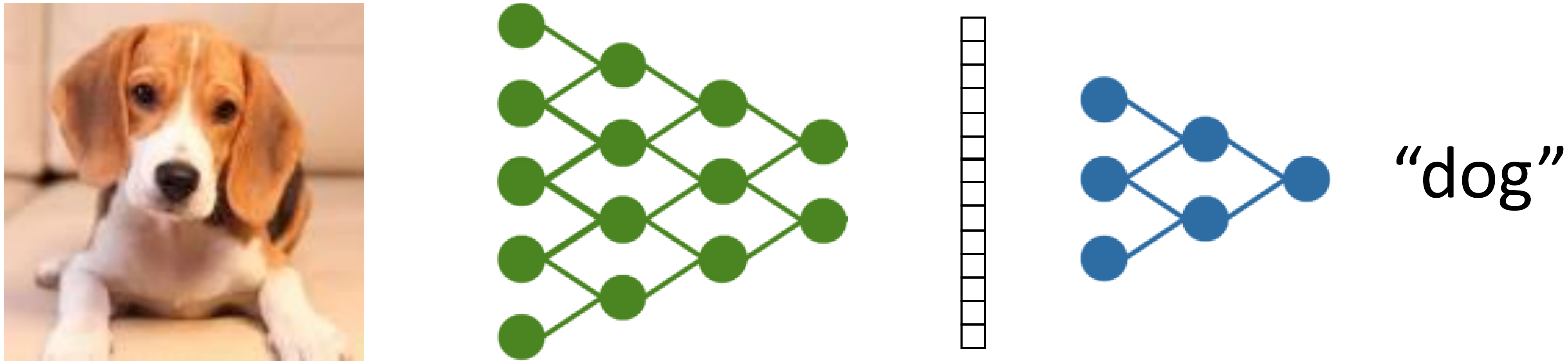
Deep learning breakthrough

Agenda

- **Neural networks**
  - Hyperparameter tuning
  - Implementation

- **Computer vision**
  - Prior to deep learning
  - Convolutional & pooling layers
  - Convolutional neural networks

Representation Learning

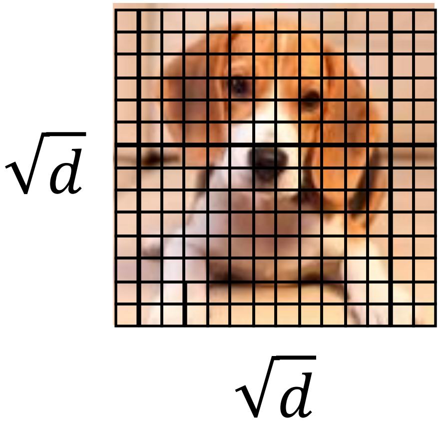


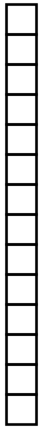
image

d-length
“feature vector” x

Representing Images as Inputs

- **Naïve strategy**
  - Feed image to neural network as a vector of pixels



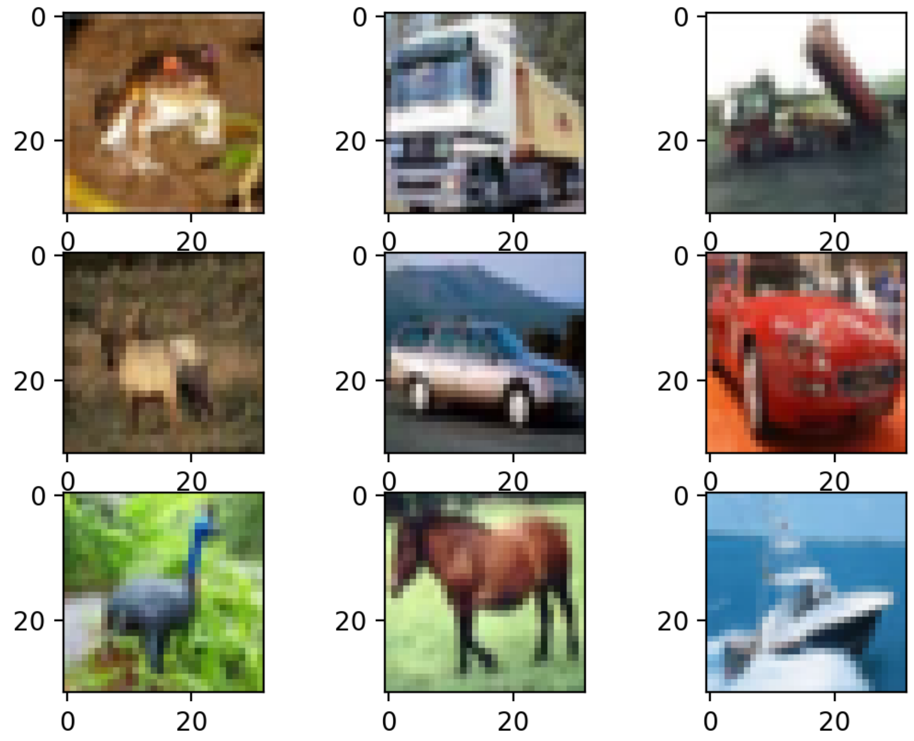


image

d-length
feature $\boldsymbol{x}$

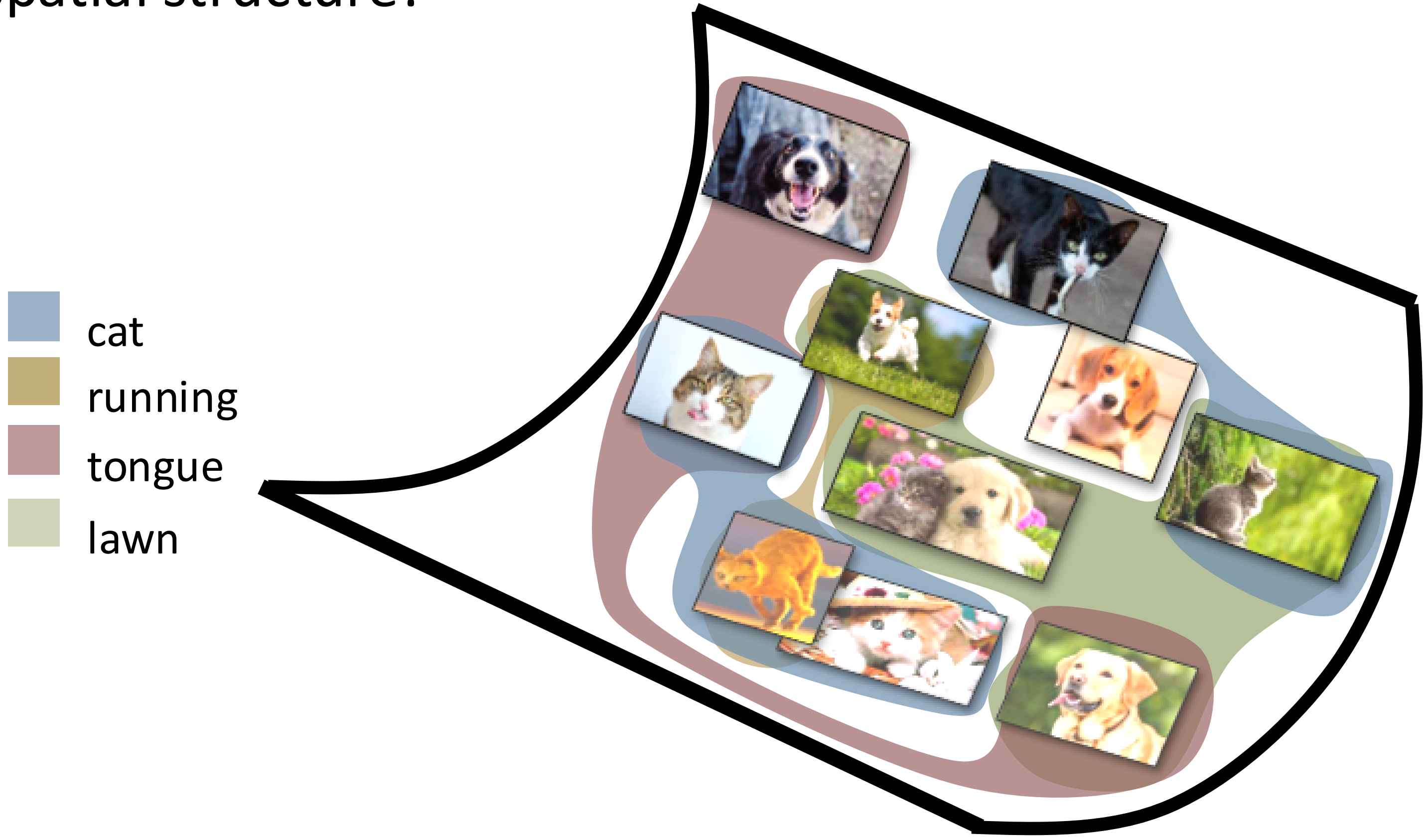
Representing Images as Inputs

- **Shortcomings**
  - Very high dimensional! $32 \times 32 \times 3 = 3072$ dimensions



Representing Images as Inputs

- **Shortcomings**
  - Ignores spatial structure!

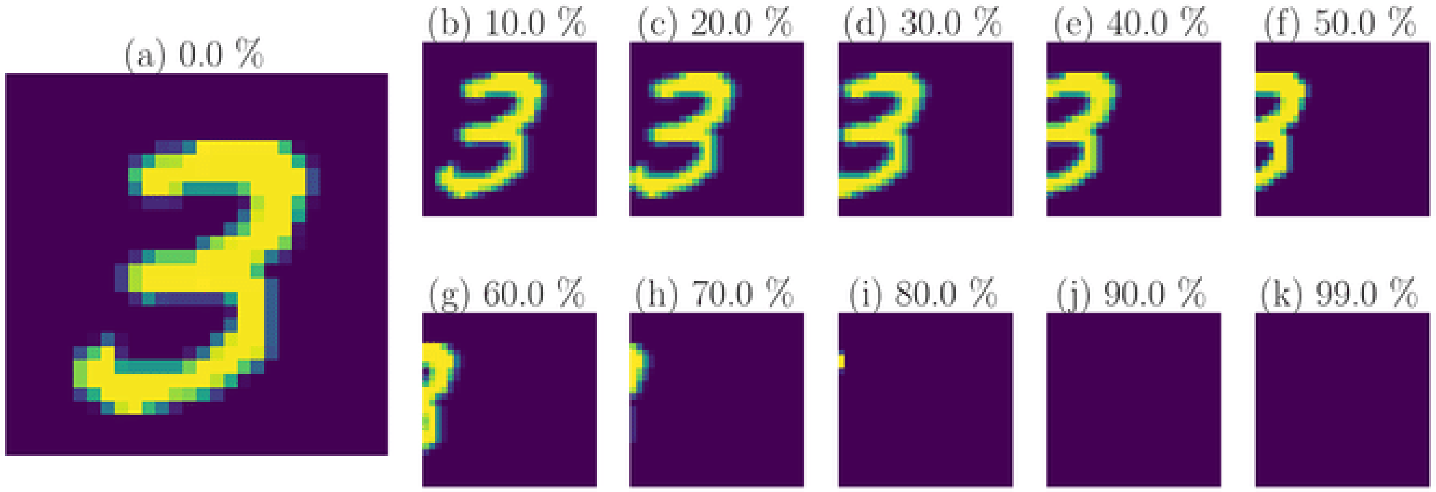


Structure in Images

- **2D image structure**
  - Location associations and spatial neighborhoods are meaningful
  - So far, we can shuffle the features without changing the problem (e.g., $\beta^\top x$)
  - Not true for images!

Structure in Images

- **Translation invariance**
  - Consider image classification (e.g., labels are cat, dog, etc.)
  - **Invariance:** If we translate an image, it does not change the category label



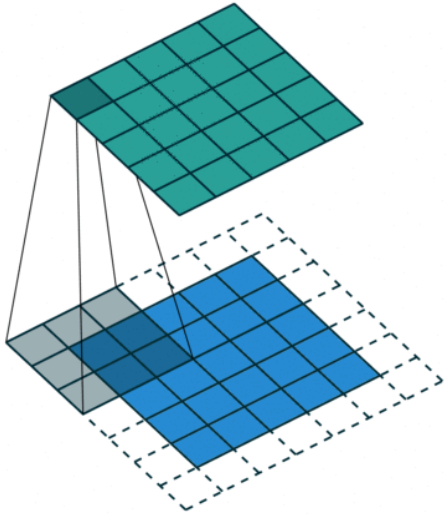
Source: Ott et al., Learning in the machine: To share or not to share?

Structure in Images

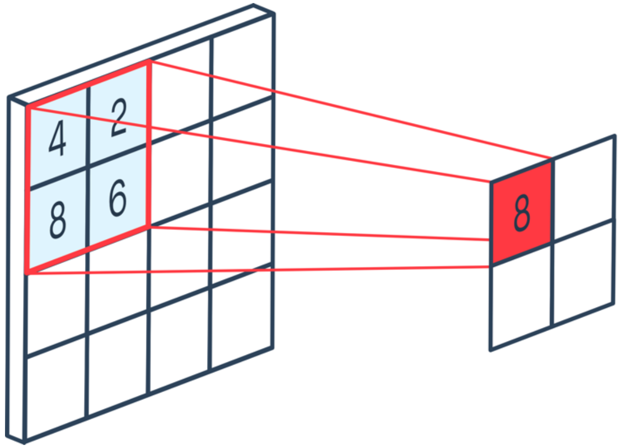
- **Translation equivariance**
  - Consider object detection (e.g., find the position of the cat in an image)
  - **Equivariance:** If we translate an image, the the object is translated similarly

Structure in Images

- Use layers that capture structure



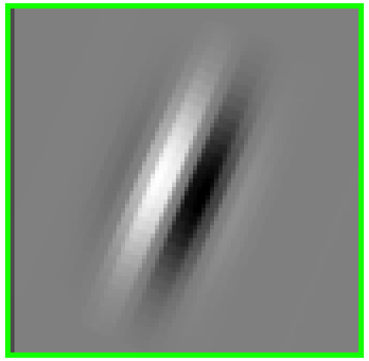
Convolution layers
(Capture equivariance)



Pooling layers
(Capture invariance)

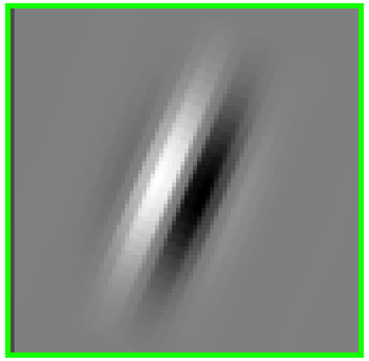
https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d
https://peltarion.com/static/2d_max_pooling_pa1.png

Convolution Filters



graphic credit: S. Lazebnik

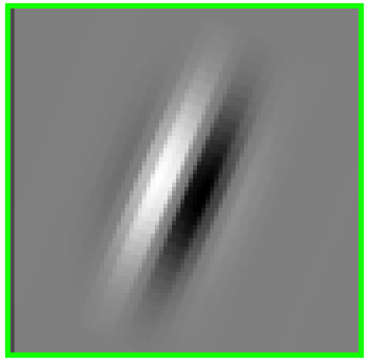
Convolution Filters



$$\text{output}[0,0] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau, \gamma] \cdot \text{image}[0+\tau, 0+\gamma]$$

graphic credit: S. Lazebnik

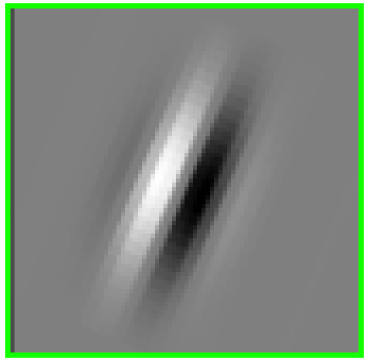
Convolution Filters



$$\text{output}[0,1] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau, \gamma] \cdot \text{image}[0+\tau, 1+\gamma]$$

graphic credit: S. Lazebnik

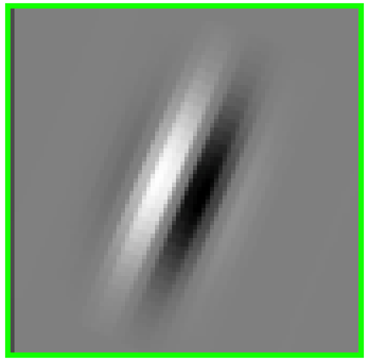
Convolution Filters



$$\text{output}[0,2] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau,\gamma] \cdot \text{image}[0+\tau, 2+\gamma]$$

graphic credit: S. Lazebnik

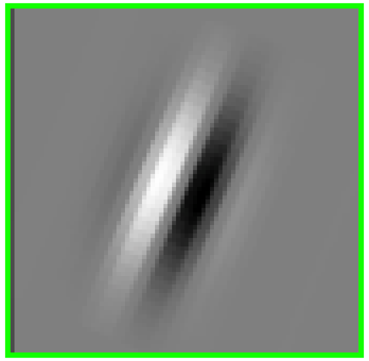
Convolution Filters

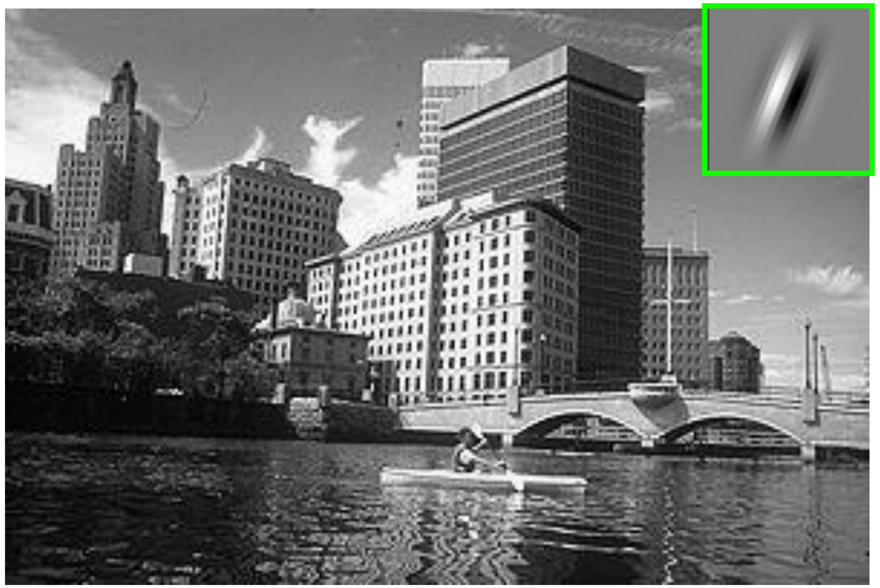


$$\text{output}[i,j] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau,\gamma] \cdot \text{image}[i+\tau, j+\gamma]$$

graphic credit: S. Lazebnik

Convolution Filters

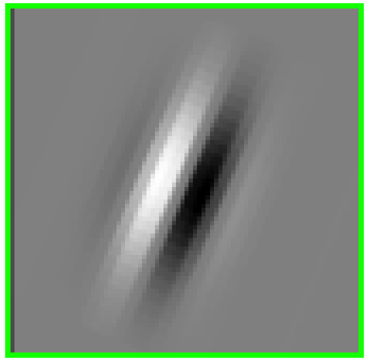


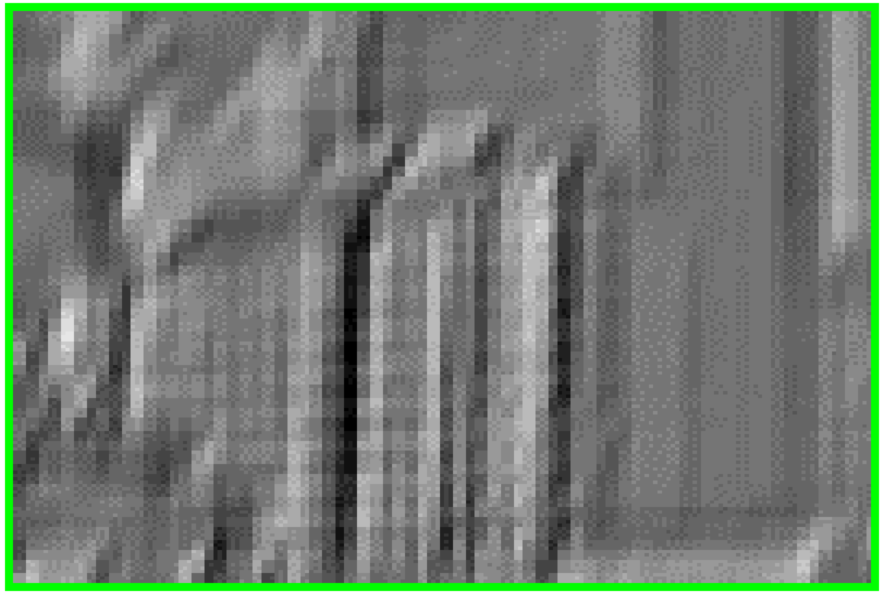


$$\text{output}[i,j] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau,\gamma] \cdot \text{image}[i+\tau, j+\gamma]$$

graphic credit: S. Lazebnik

Convolution Filters

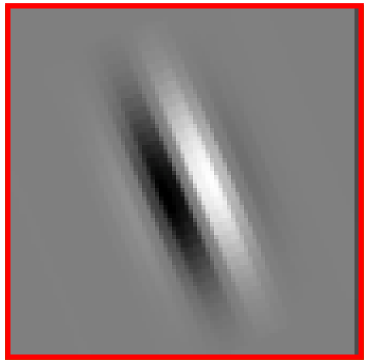




$$\text{output}[i, j] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau, \gamma] \cdot \text{image}[i + \tau, j + \gamma]$$

graphic credit: S. Lazebnik

Convolution Filters



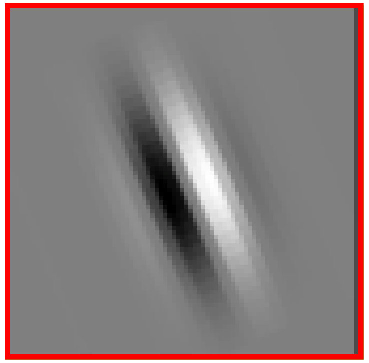
graphic credit: S. Lazebnik

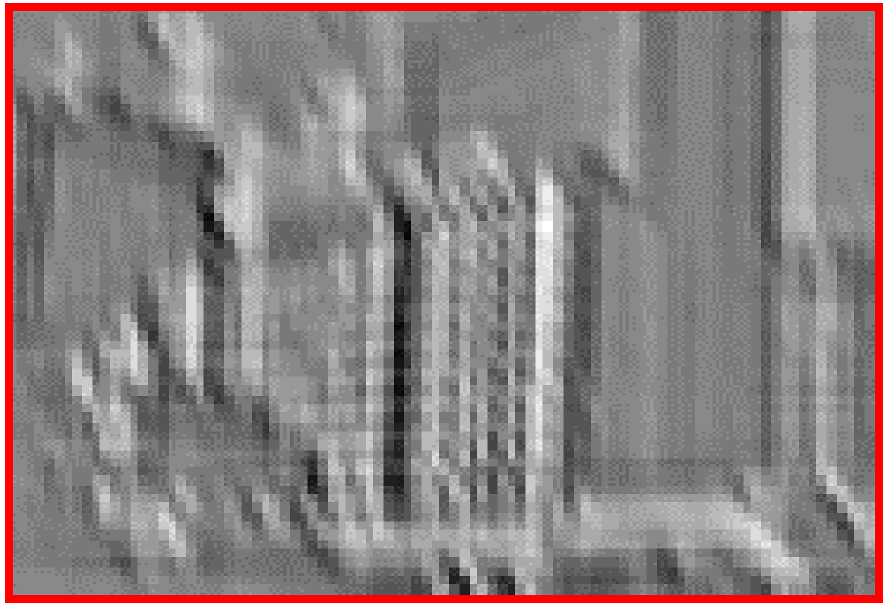
Convolution Filters

$$\text{output}[i,j] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau,\gamma] \cdot \text{image}[i+\tau, j+\gamma]$$

graphic credit: S. Lazebnik

Convolution Filters





$$\text{output}[i,j] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau,\gamma] \cdot \text{image}[i+\tau, j+\gamma]$$

graphic credit: S. Lazebnik

1D Convolution Filters

- **Given:**
  - 1D sequence $x$is 1D
  - 1D **kernel** k

- Convolution is the following:

$$y[t] = \sum_{\tau=0}^{|k|-1} k[\tau] \cdot x[t+\tau]$$

- Technically **cross-correlation**

1D Convolution Filters

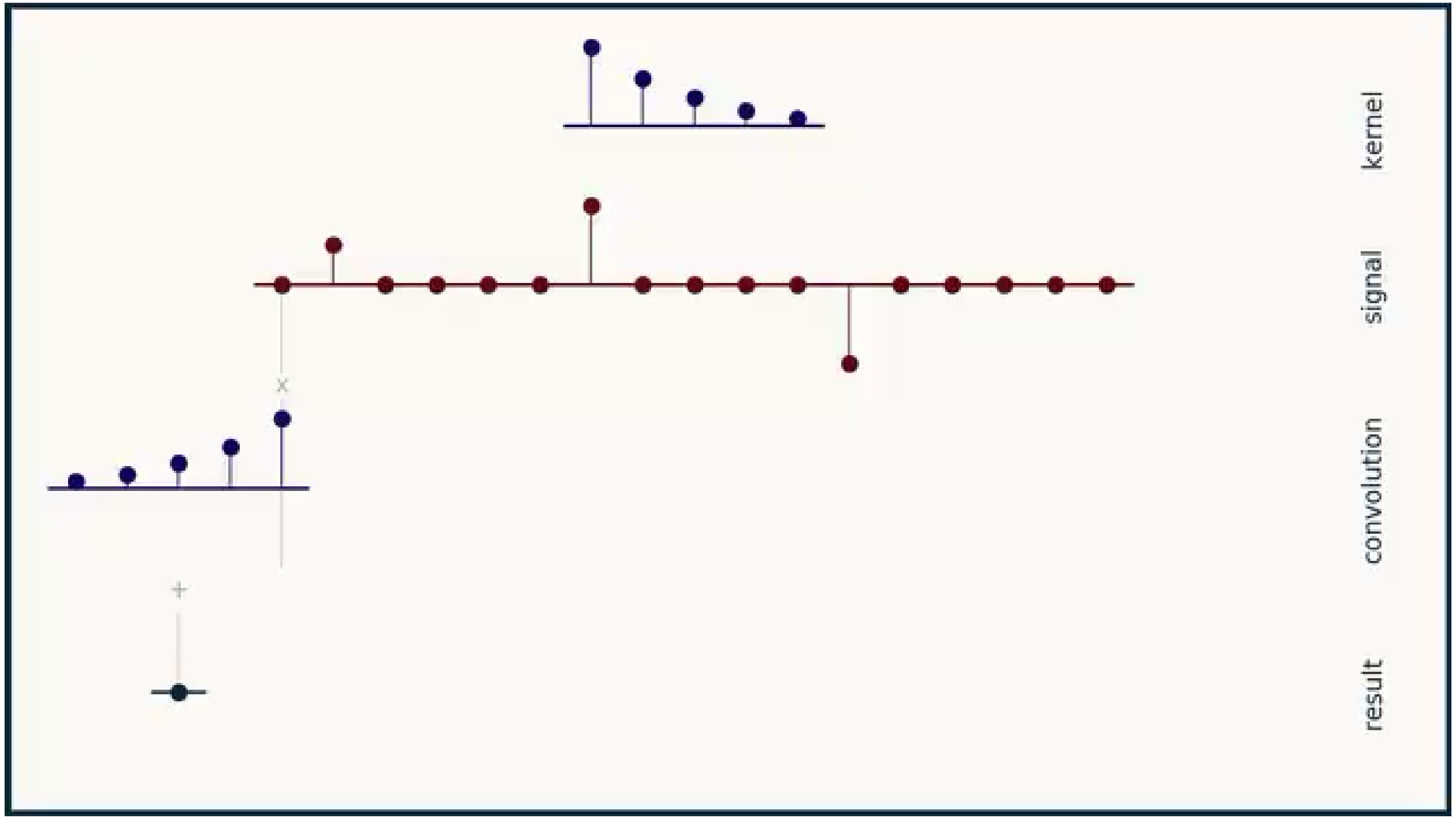
- **Example:**
  - $x = [25000, 28000, 30000, 21000, 18000, \ldots]$
  - $k = [-1, 1, -1]$

- **Convolution:**

$$y[t] = \sum_{\tau=0}^{|k|-1} k[\tau] \cdot x[t+\tau]$$

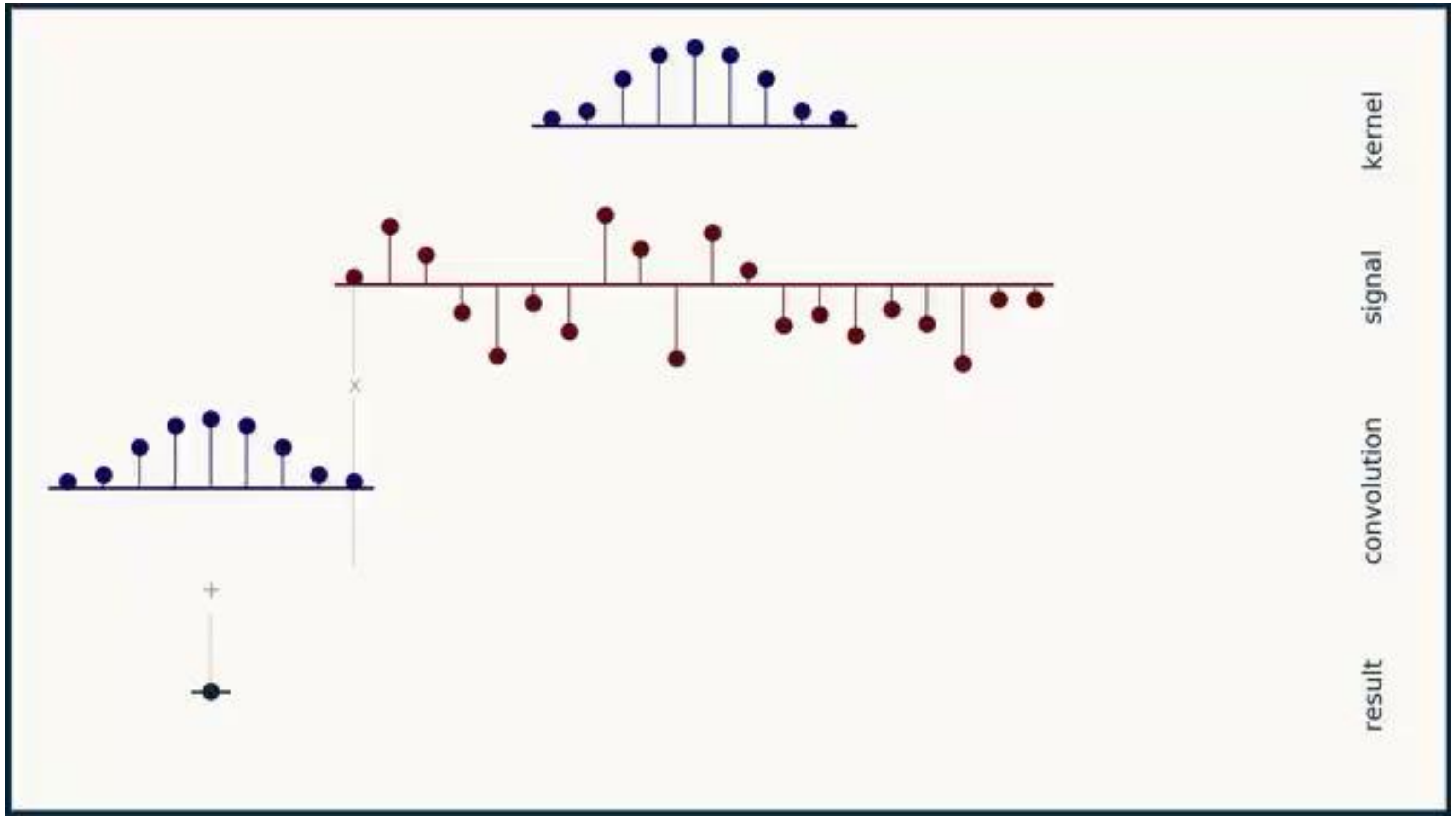
$$y[0] = k[0]x[0] + k[1]x[1] + k[2]x[2] = -25000 + 28000 - 30000$$
$$y[1] = k[0]x[1] + k[1]x[2] + k[2]x[3] = -28000 + 30000 - 21000$$
$$y[2] = k[0]x[2] + k[1]x[3] + k[2]x[4] = -30000 + 21000 - 18000$$

1D Convolution Filters



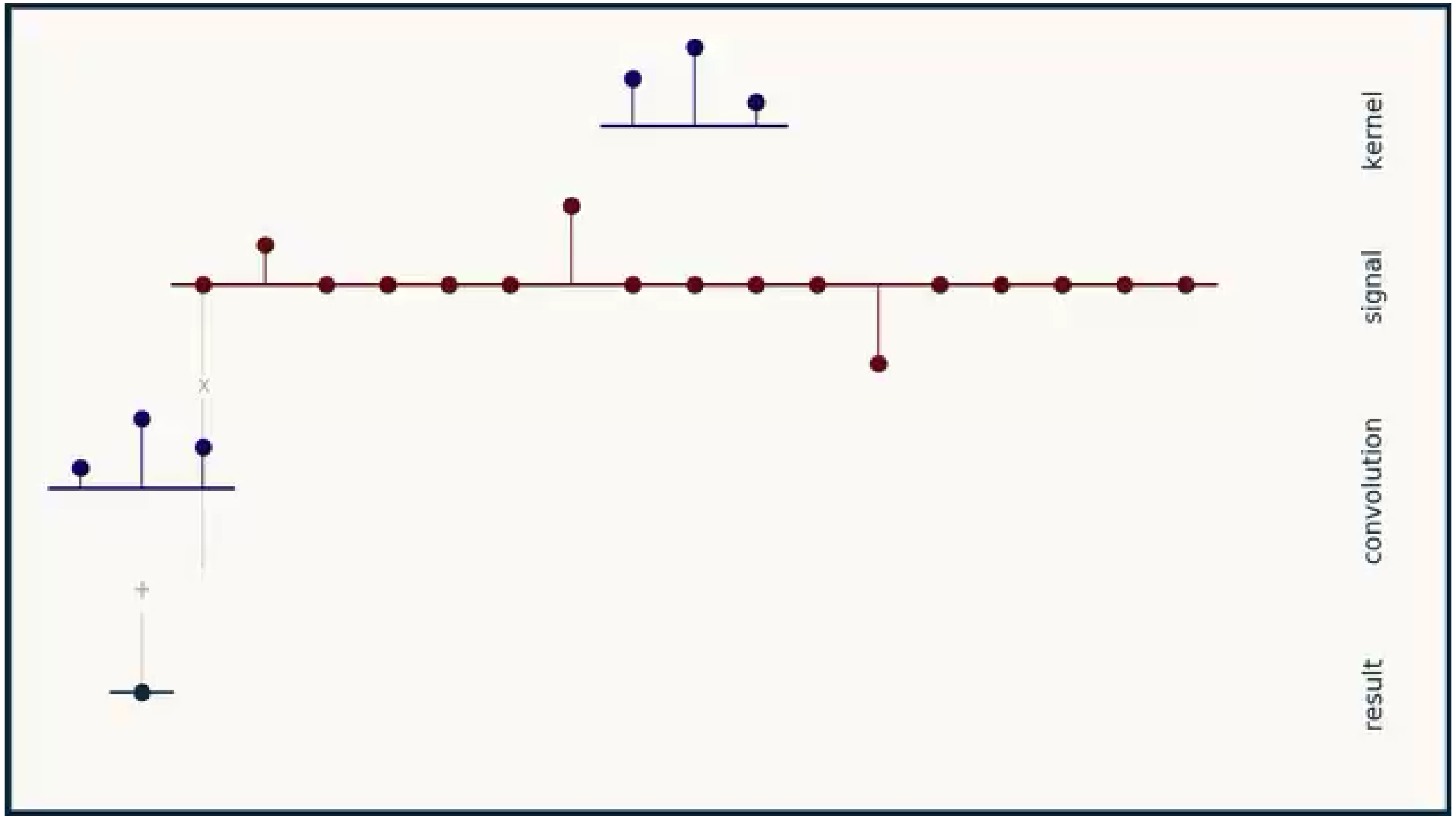
https://gitlab.com/brohrer/

1D Convolution Filters



https://gitlab.com/brohrer/

1D Convolution Filters



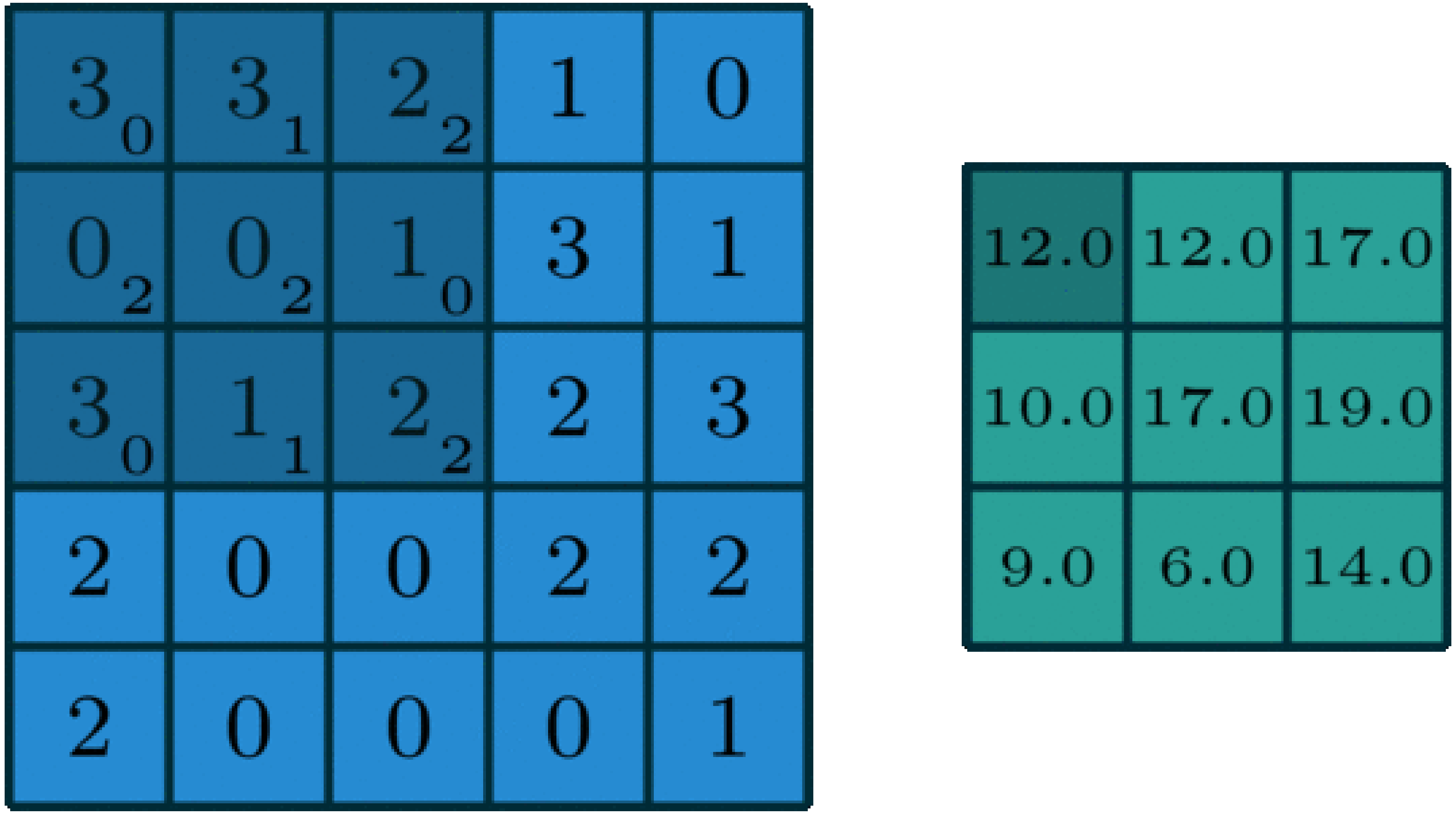
2D Convolution Filters

- **Given:**
  - A 2D input x
  - A 2D $h \times w$ kernel k

- The 2D convolution is:

$$y[s,t] = \sum_{\tau=0}^{h-1} \sum_{\gamma=0}^{w-1} k[\tau,\gamma] \cdot x[s+\tau, t+\gamma]$$

2D Convolution Filters



2D Convolution Filters

- Historically (until late 1980s), kernel parameters were handcrafted
  - E.g., “edge detectors”

2D Convolution Filters

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

Vertical lines

-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines

Example Edge Detection Kernels

Result of Convolution with Horizontal Kernel

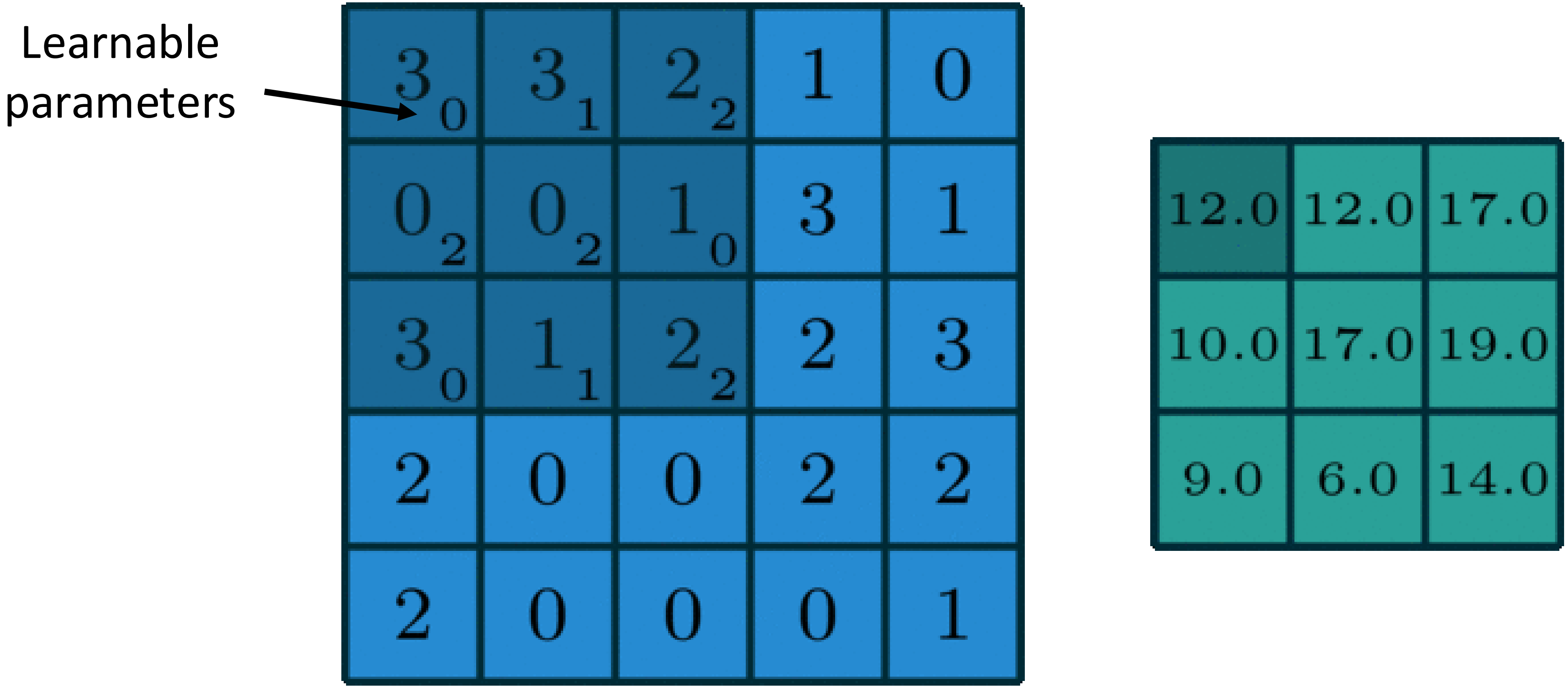
https://aishack.in/tutorials/image-convolution-examples/

2D Convolution Filters

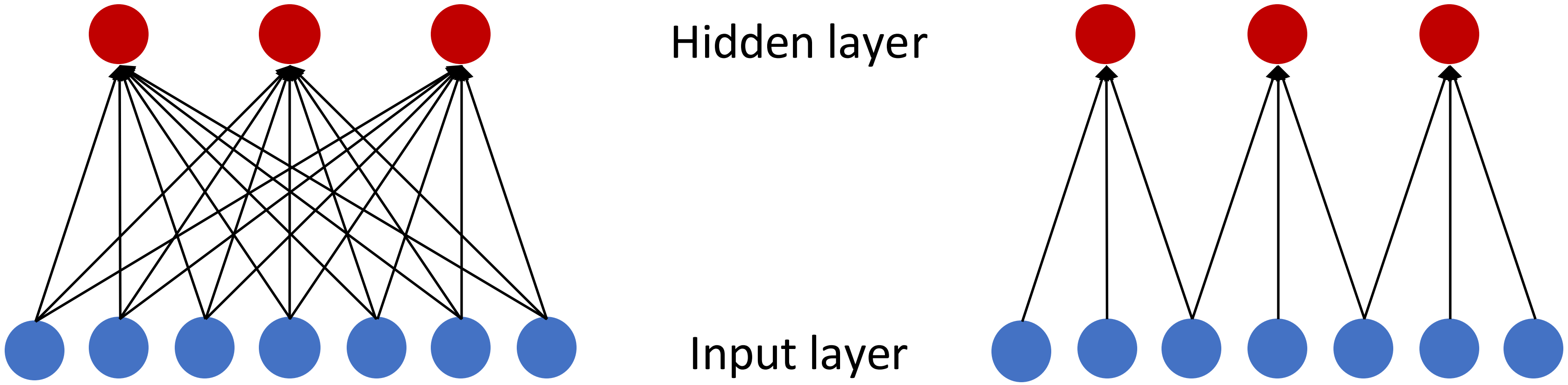
- Historically (until late 1980s), kernel parameters were handcrafted
  - E.g., “edge detectors”

- In convolutional neural networks, they are learned
  - Essentially a linear layer with fewer “connections”
  - Backpropagate as usual!

Convolution Layers



Convolution Layers

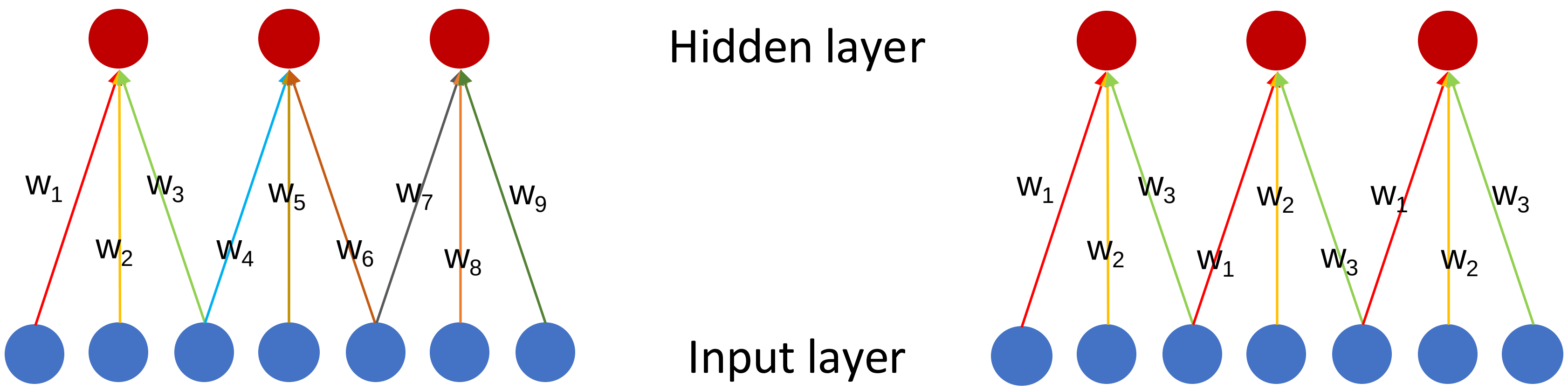


Fully connected
(3 input × 7 output = 21 parameters)

Locally connected
(3 input × 3 output = 9 parameters)

Slide credit: Jia-Bin Huang

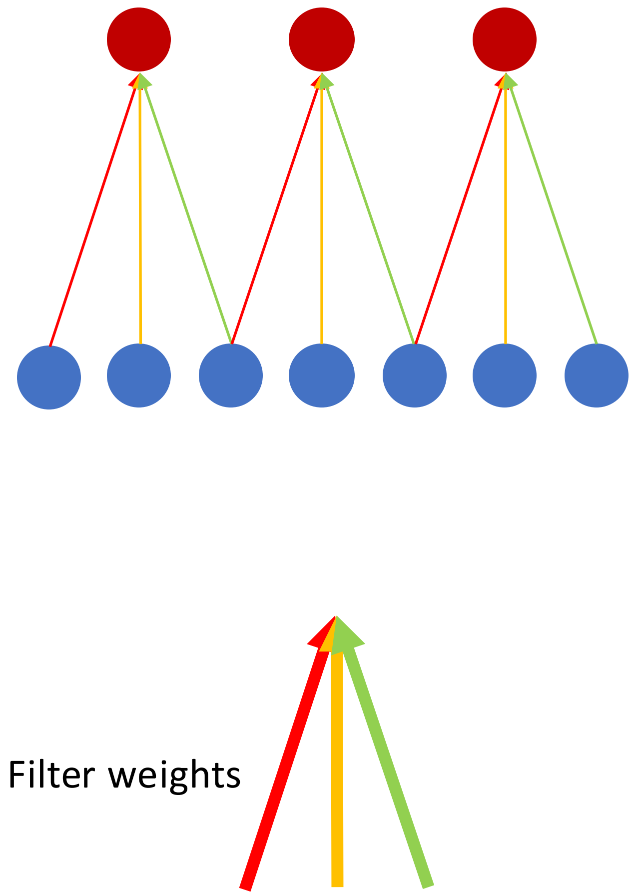
Convolution Layers



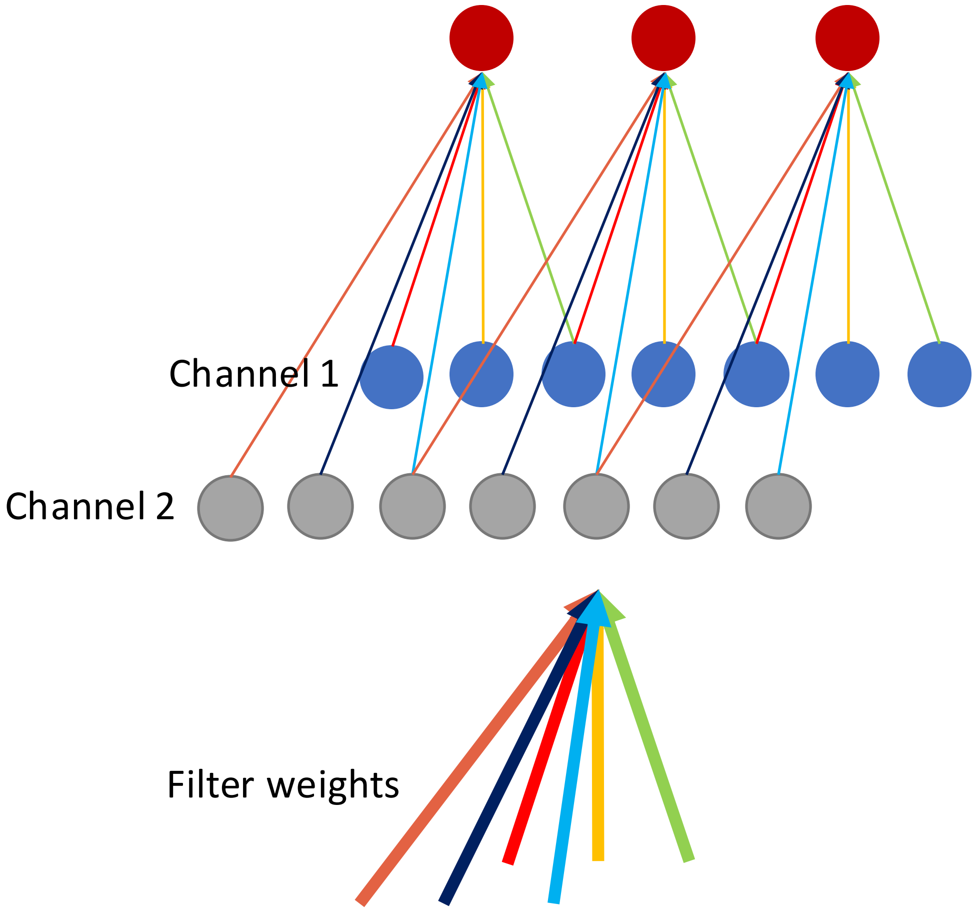
Without weight sharing
(3 input × 3 output = 9 parameters)

With weight sharing
(3 parameters)

Convolution Layers



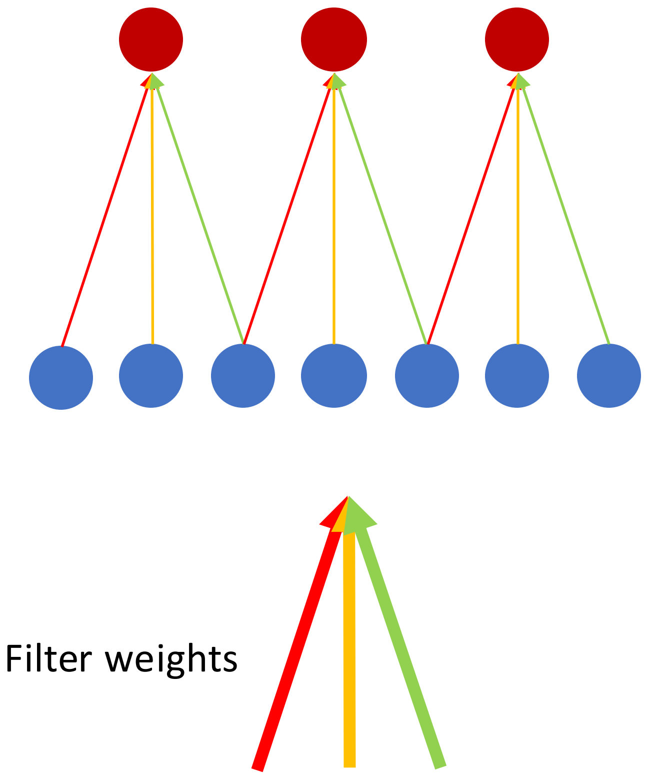
Single input channel



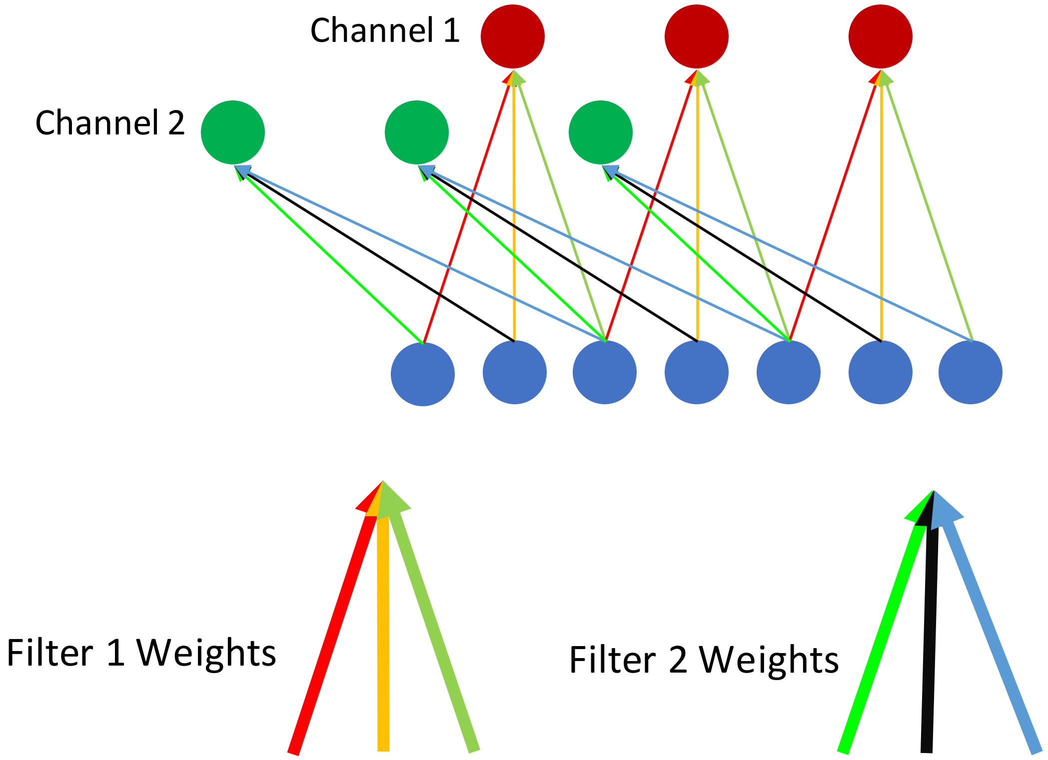
Multiple input channels

Slide credit: Jia-Bin Huang

Convolution Layers



Single output map



Multiple output maps

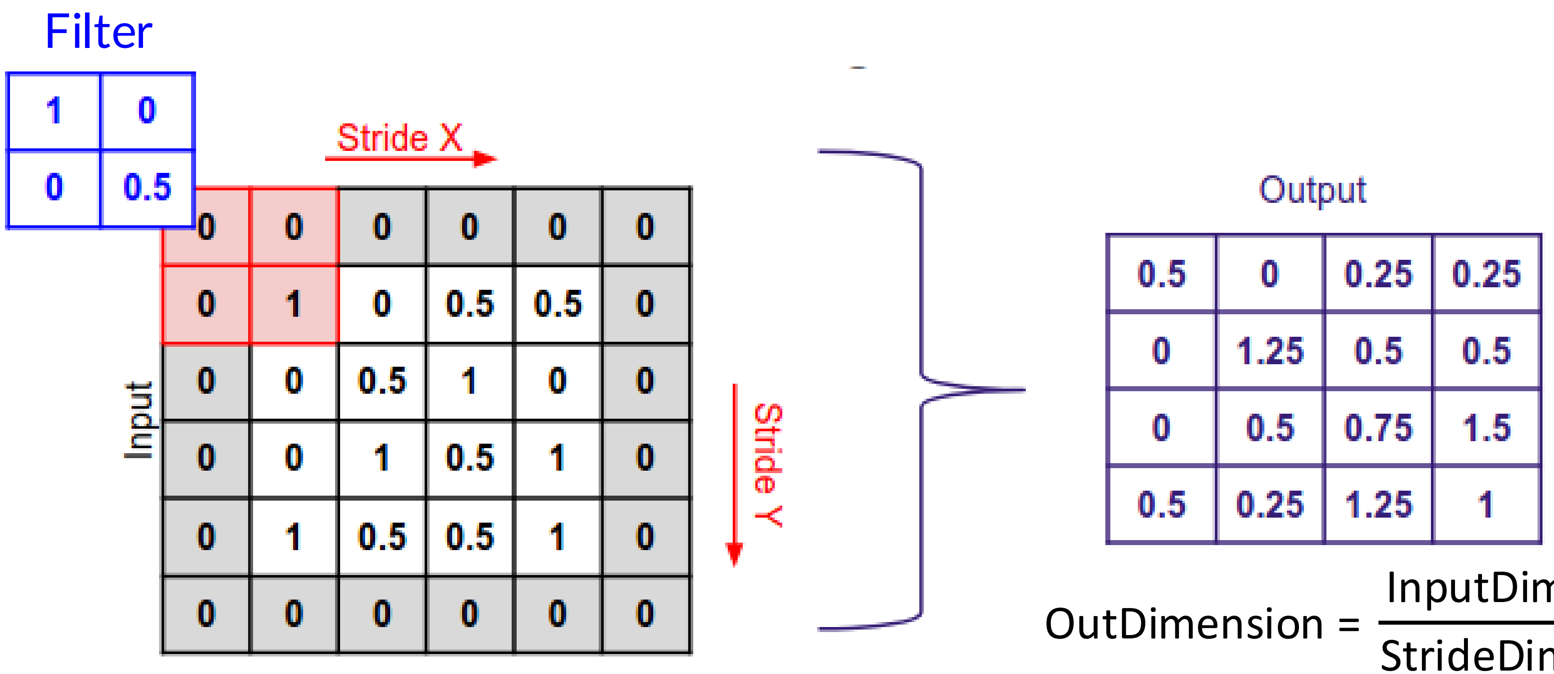
Slide credit: Jia-Bin Huang

Convolution Layers

- **Summary**
  - Local connectivity
  - Weight sharing
  - Handling multiple input/output channels
  - Retains location associations

Convolution Layer Parameters

- **Stride:** How many pixels to skip (if any)
  - **Default:** Stride of 1 (no skipping)

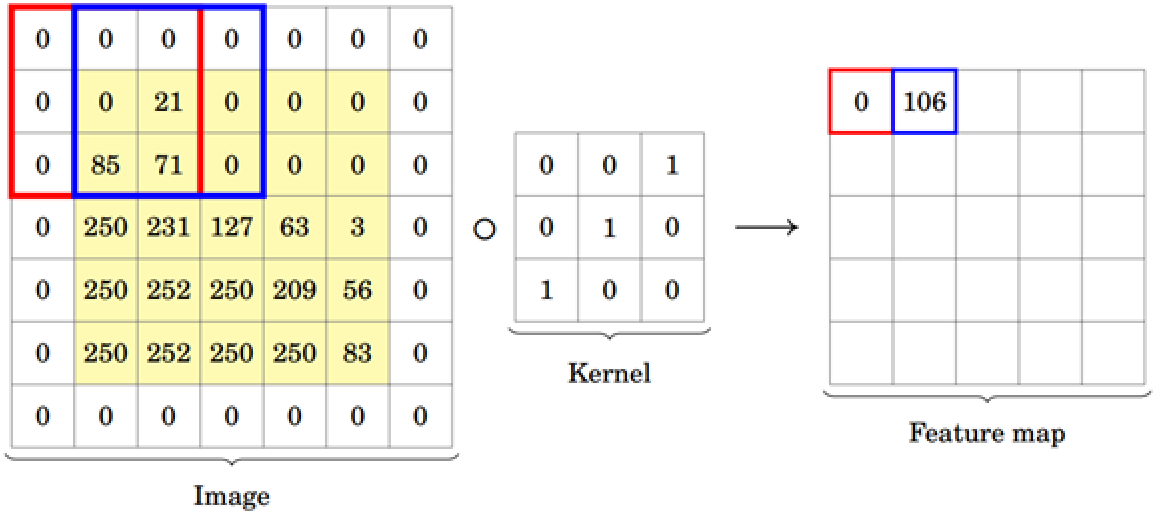


$$\text{OutDimension} = \frac{\text{InputDimension}}{\text{StrideDimension}}$$

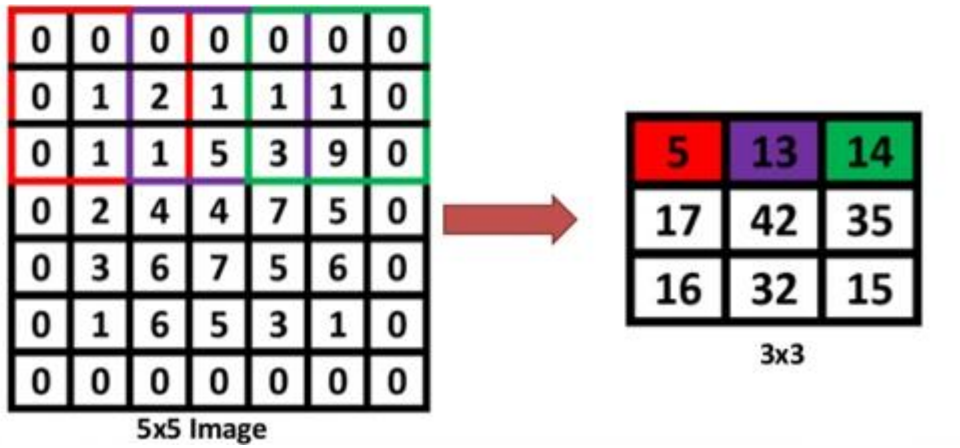
https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7

Convolution Layer Parameters

- **Padding:** Add zeros to edges of image to capture ends
  - **Default:** No padding



stride = 1, zero-padding = 1



stride = 2, zero-padding = 1

https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7

Convolution Layer Parameters

- **Summary:** Hyperparameters
  - Kernel size
  - Stride
  - Amount of zero-padding
  - Output channels

- Together, these determine the relationship between the input tensor shape and the output tensor shape

- Typically, also use a single bias term for each convolution filter

Convolution Layers

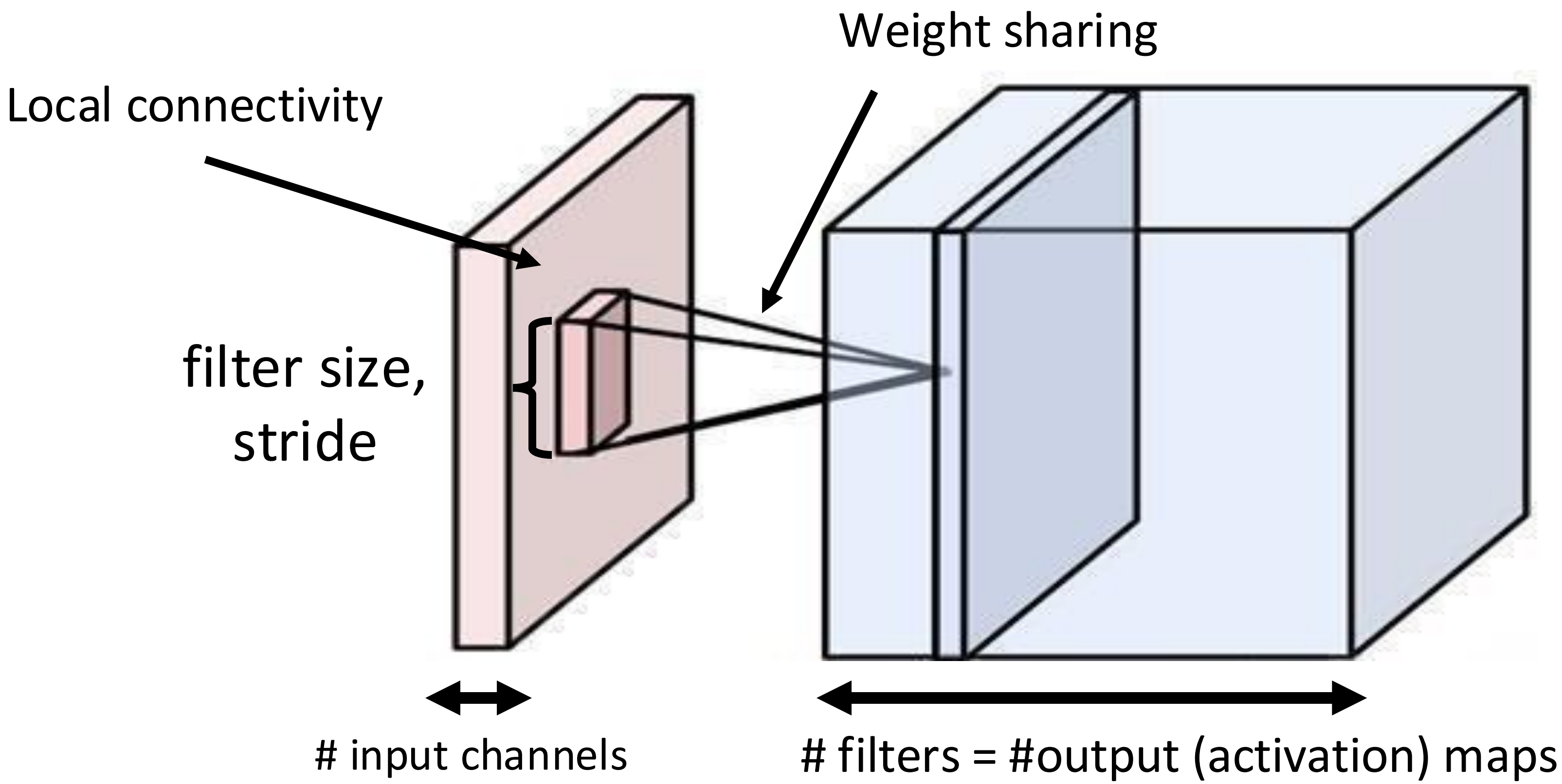
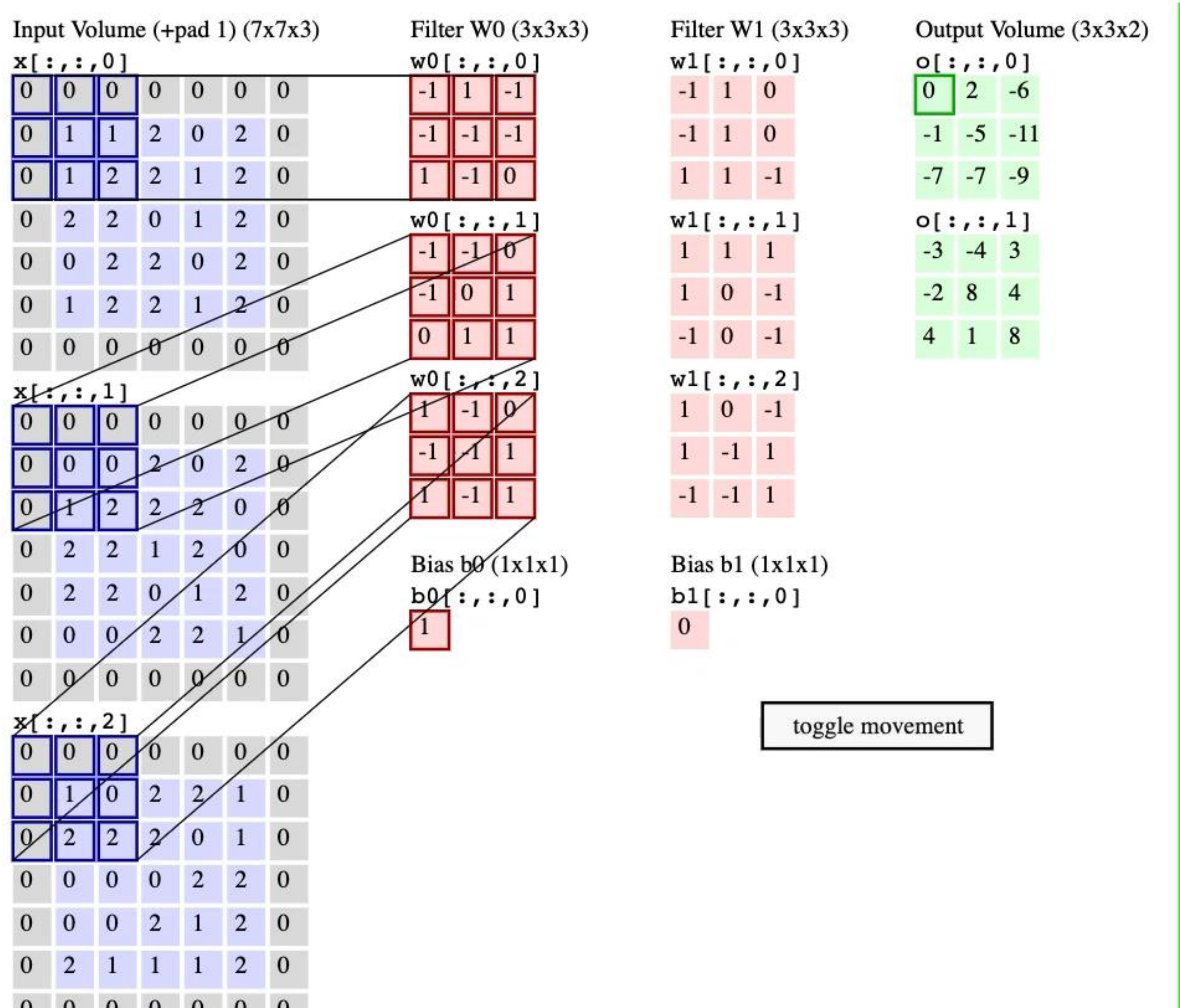


Image credit: A. Karpathy Slide credit: Jia-Bin Huang

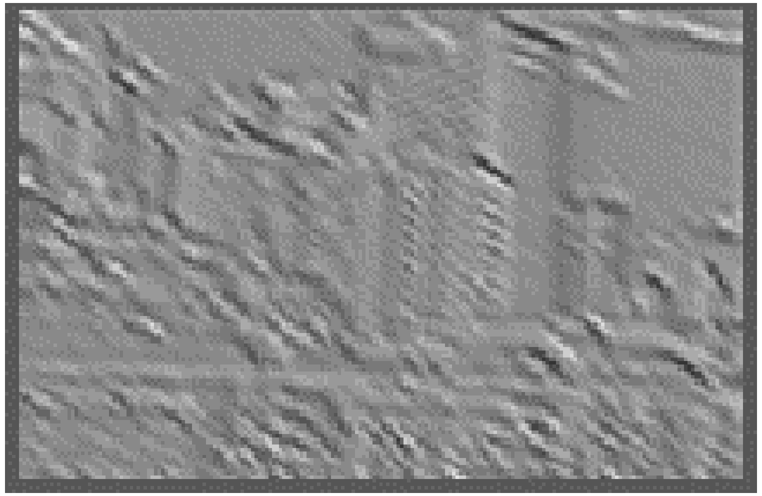
Example

- Kernel size 3, stride 2, padding 1
- 3 input channels
  - Hence kernel size 3×3×3
- 2 output channels
  - Hence 2 kernels
- Total # of parameters:
  - $(3 \times 3 \times 3 + 1) \times 2 \ = \ 56$

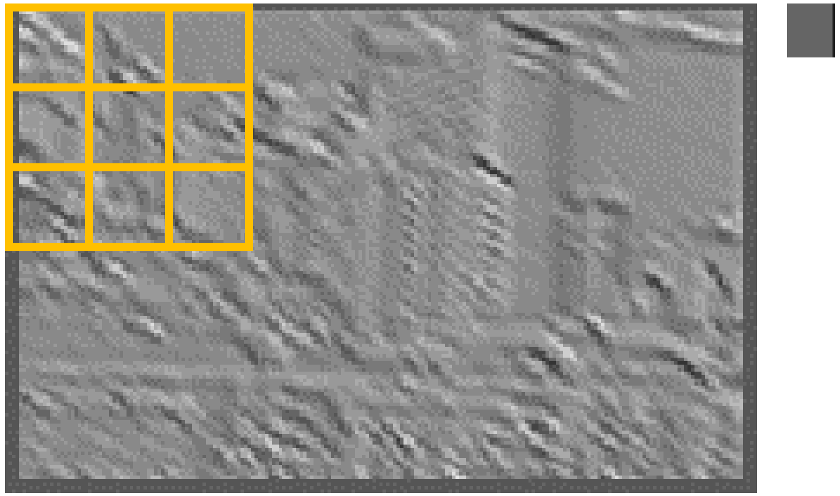


http://cs231n.github.io/convolutional-networks/

Pooling Layers

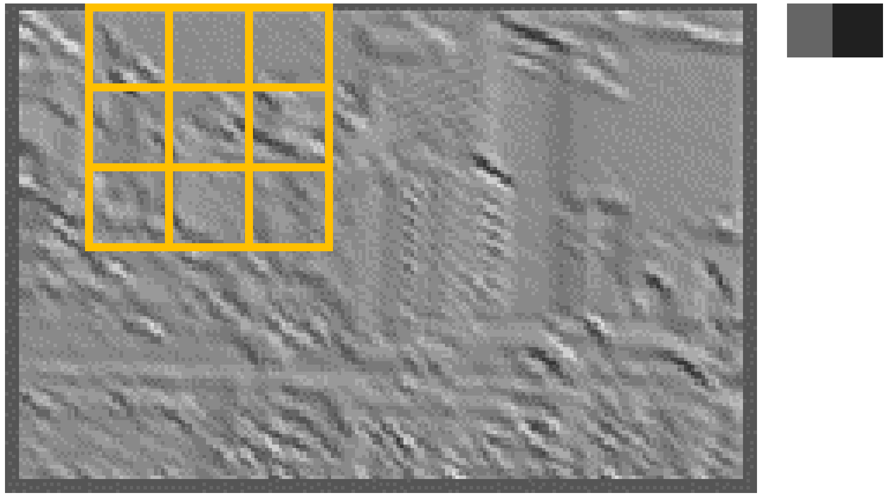


Pooling Layers



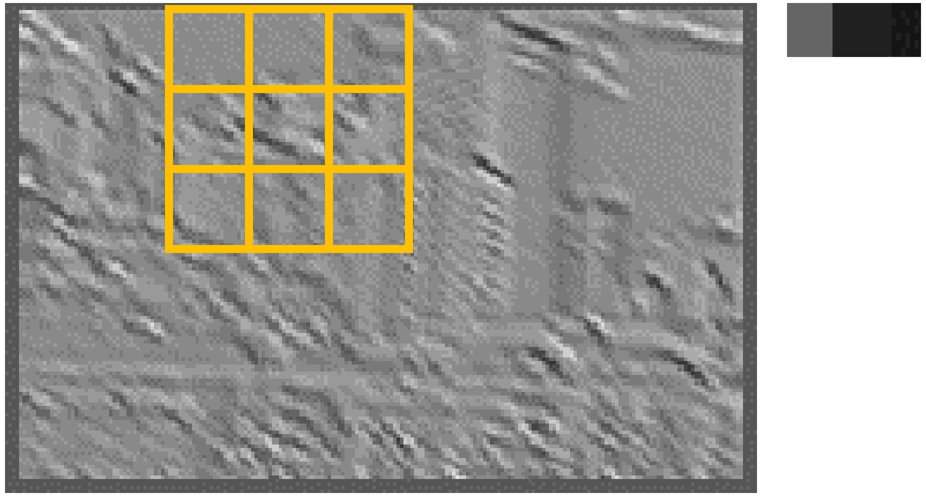
$$\text{output}[0,0] = \max_{0\le\tau<k}\ \max_{0\le\gamma<k}\ \text{image}[0+\tau, 0+\gamma]$$

Pooling Layers



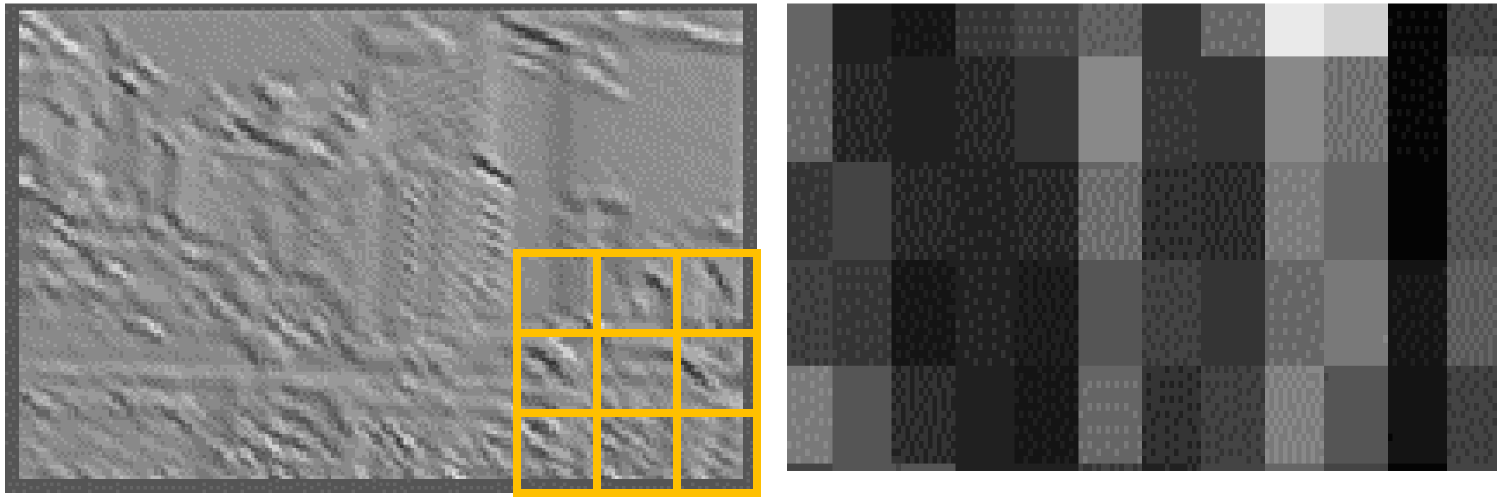
$$\text{output}[0,1] = \max_{0 \le \tau < k} \max_{0 \le \gamma < k} \text{image}[0+\tau, 1+\gamma]$$

Pooling Layers



$$\text{output}[0,2] = \max_{0 \le \tau < k} \max_{0 \le \gamma < k} \text{image}[0+\tau, 2+\gamma]$$

Pooling Layers



$$\text{output}[i, j] = \max_{0 \le \tau < k} \max_{0 \le \gamma < k} \text{image}[i + \tau, j + \gamma]$$

Pooling Layers

- **Summary:** Hyperparameters
  - Kernel size
  - Stride (usually >1)
  - Amount of zero-padding
  - Pooling function (almost always "max")

- Together, these determine the relationship between the input tensor shape and the output tensor shape

- **Note:** Unlike convolution, pooling operates on channels separately
  - Thus, n input channels → n output channels

Summary: Convolution vs. Pooling

- **Convolution layers:** Translation equivariant
  - If object is translated, convolution output is translated by same amount
  - Produce “image-shaped” features that retain associations with input pixels
- **Pooling layers:** Translation invariant
  - Binning to make outputs insensitive to translation
  - Also reduces dimensionality
- Combined in modern architectures
  - Convolution to construct equivariant features
  - Pooling to enable invariance