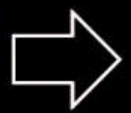
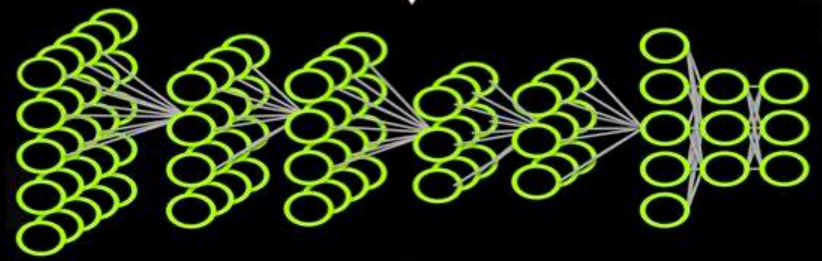
Deep Learning for Self Driving Cars

Neural Networks for Vision

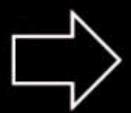
DEEP LEARNING APPROACH

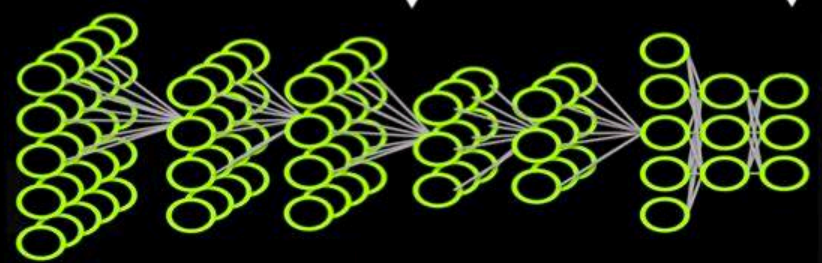
Train:





Deploy:

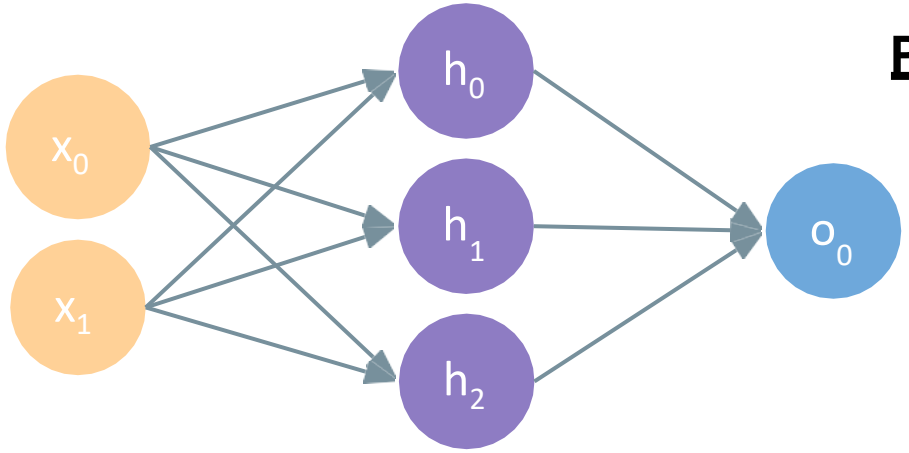




Total Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



Predicted

[
0.05
0.02
0.96
0.35
]

Actual

[
1
0
1
1
]

$$\text{total loss} := J(\theta) = \frac{1}{N}\sum_{i}^{N} loss(f(x^{(i)};\theta), y^{(i)}))$$

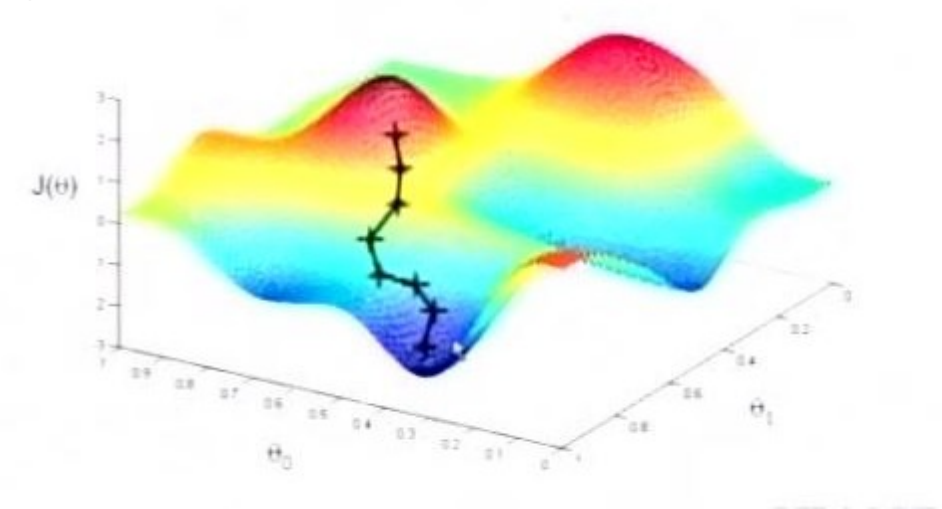
Predicted Actual

Why is it Stochastic Gradient Descent?

- Initialize θ randomly
- For N Epochs
  - For each training example (x, y):
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update θ with update rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

Only an estimate of true gradient!

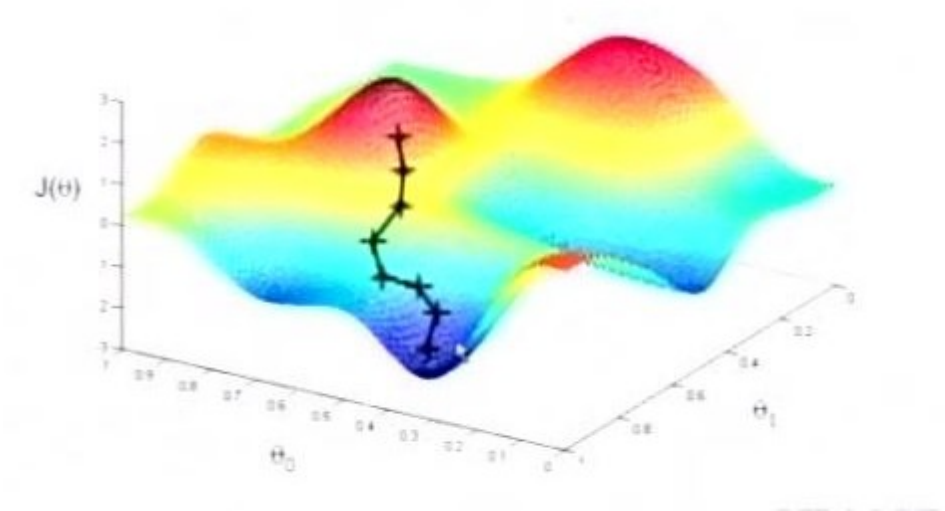


(Stolen from N. Locascio)

Minibatches Reduce Gradient Variance

- Initialize θ randomly
- For N Epochs
  - For each training **batch {(x0, y0), … , (x_B, y_B)}:**
    - Compute Loss Gradient: $$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B}\sum_{i}^{B}\frac{\partial J_i(\theta)}{\partial \theta}$$
    - Update θ with update rule: $$\theta := \theta - \eta\frac{\partial J(\theta)}{\partial \theta}$$

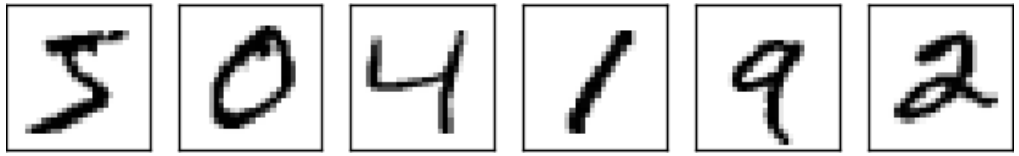
More accurate estimate!



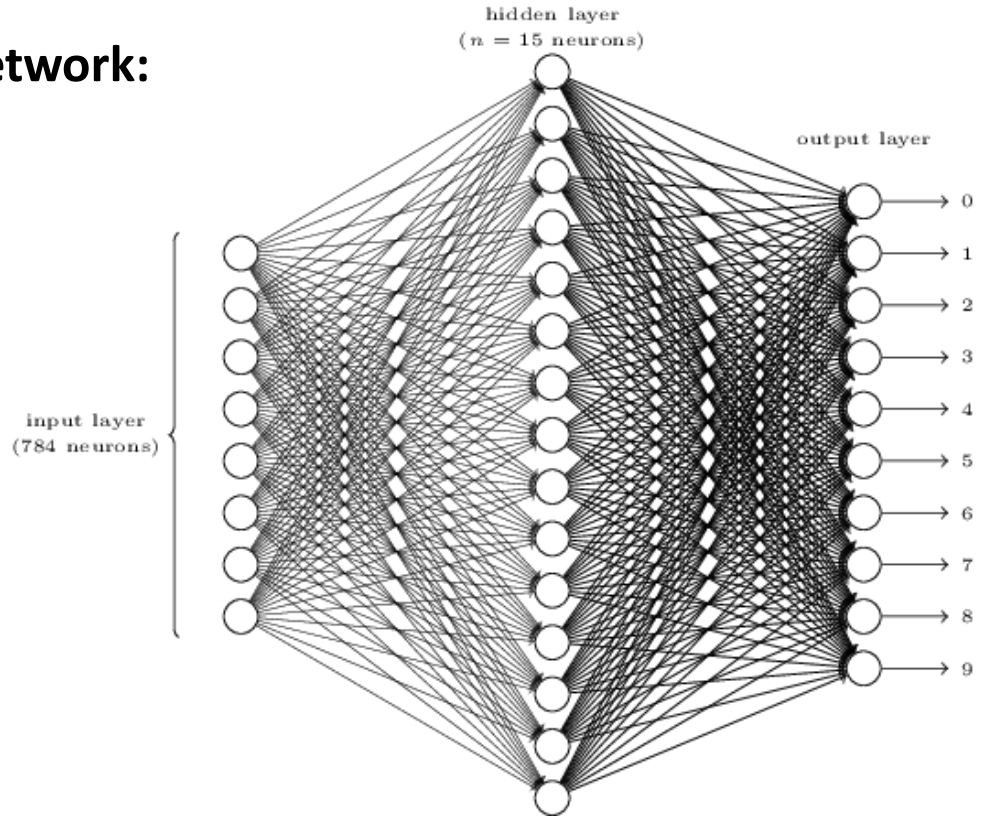
(Stolen from N. Locascio)

Classify and Image of a Number

Input:
(28x28)

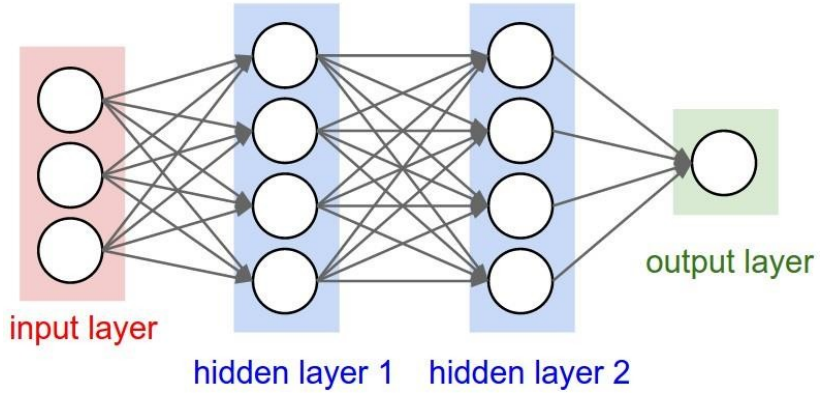


Network:

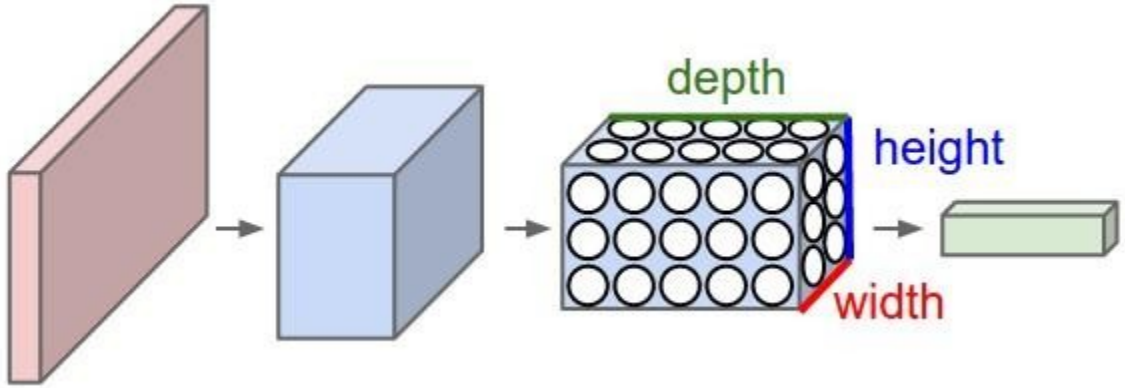


Convolutional Neural Networks

Regular neural network (fully connected):



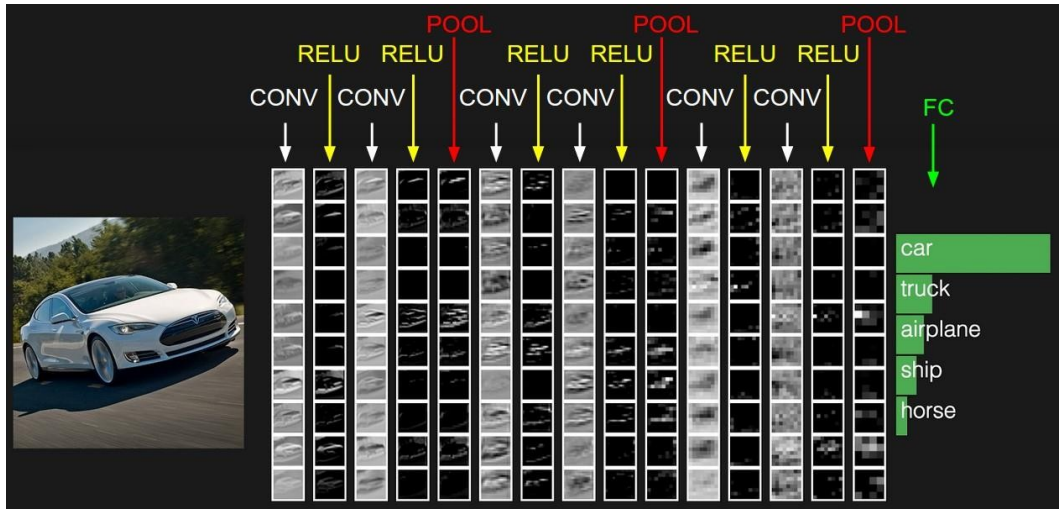
Convolutional neural network:



Each layer takes a 3d volume, produces 3d volume with some smooth function that may or may not have parameters.

Convolutional Neural Networks: Layers

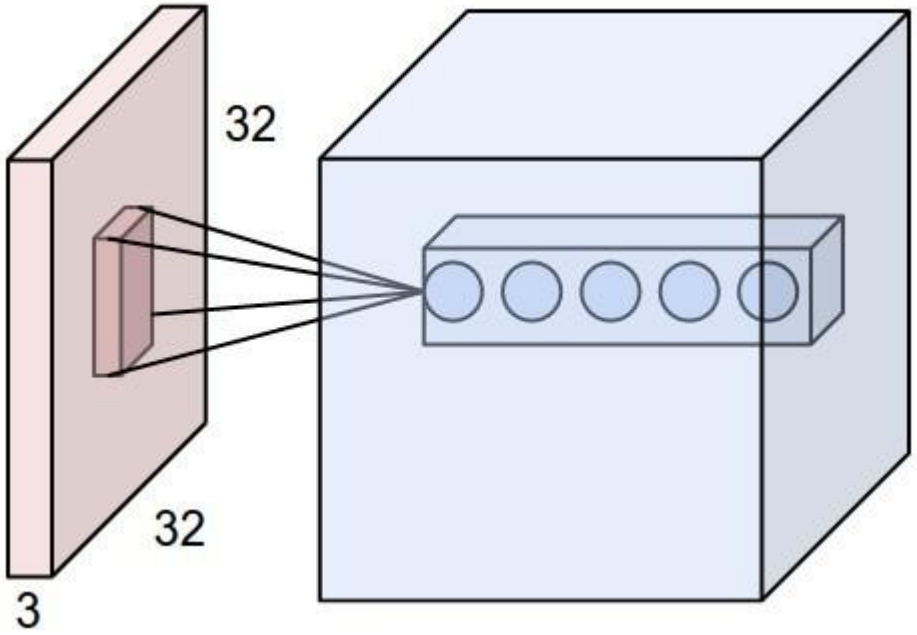
- **INPUT** [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- **CONV** layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- **RELU** layer will apply an elementwise activation function, such as the *max(0,x)* thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- **POOL** layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- **FC** (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

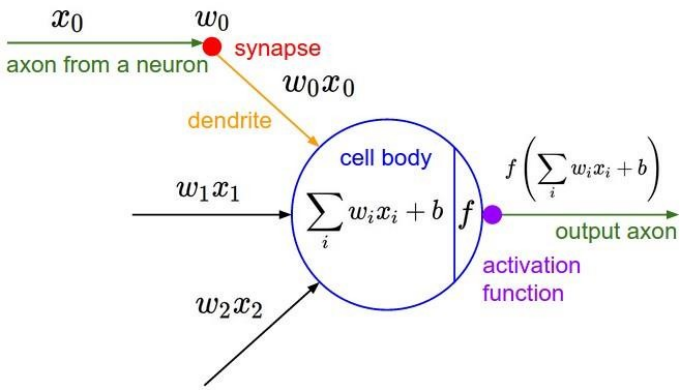


Layers **highlighted in blue** have learnable parameters.

Slides stolen L
Fridman

Dealing with Images: Local Connectivity



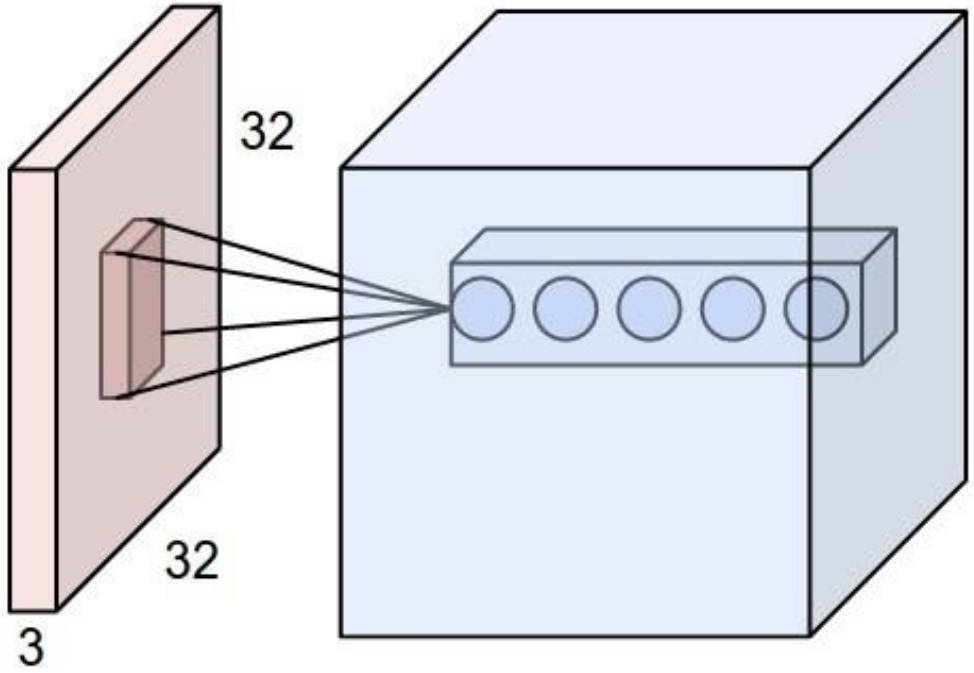


Same neuron. Just more focused (narrow “receptive field”).

The parameters on a each filter are spatially “shared”

(if a feature is useful in one place, it’s useful elsewhere)

ConvNets: Spatial Arrangement of Output Volume



- **Depth:** number of filters
- **Stride:** filter step size (when we “slide” it)
- **Padding:** zero-pad the input

Convolution

- Let *f* be the image and *g* be the kernel. The output of convolving *f* with *g* is denoted by *f* * *g*.

$$(f * g)[n,m] = \sum_{k,l} f[k,l]\, g[n-k, m-l]$$

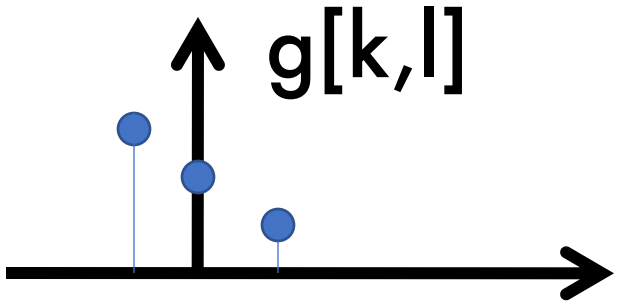
- Weighted product of f(k,l) by g(-(k,l))) computed at different locations n,m

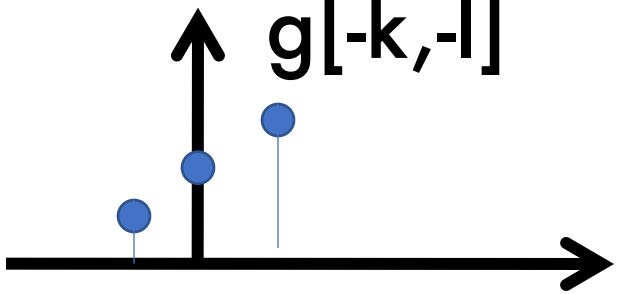
- MATLAB: conv2 vs. filter2 (also imfilter)

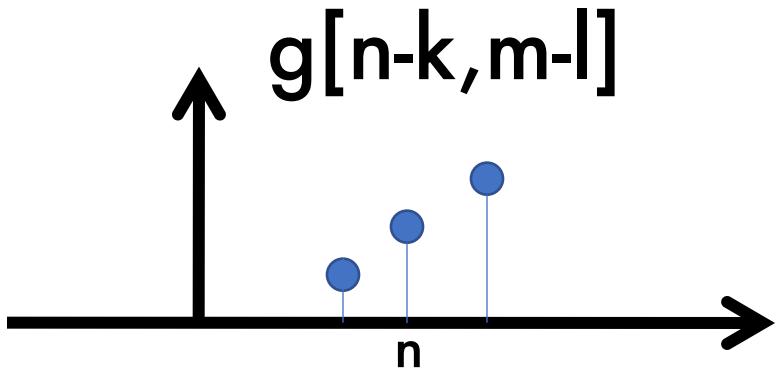
Discrete convolution

$$(f * g)[n,m] = \sum_{k,l} f[k,l]\, g[n-k, m-l]$$

- Fold g[k,l] about origin to form g[−k,−l]
- Shift the folded results by n,m to form g[n − k,m − l]
- Multiply g[n − k,m − l] by f[k, l]
- Sum over all k,l
- Repeat for every n,m



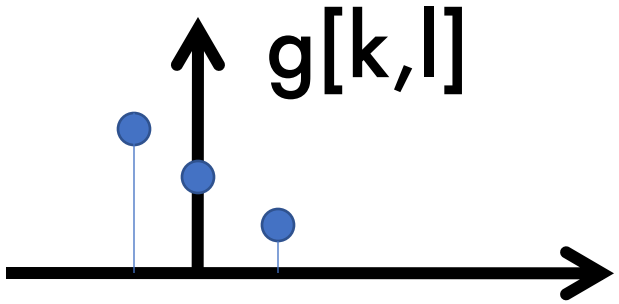


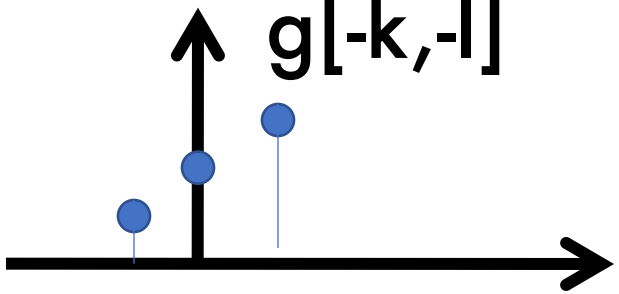


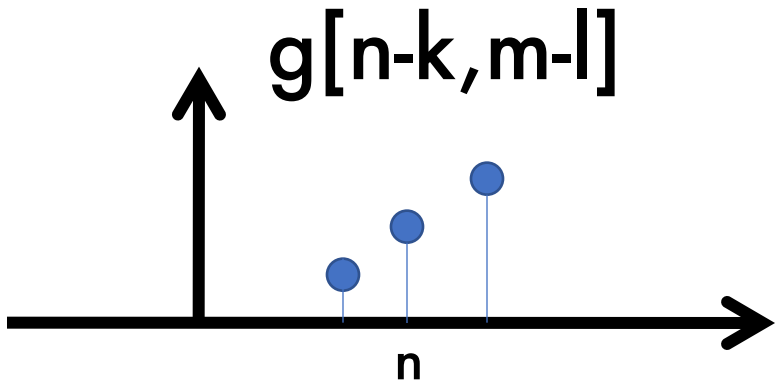
Discrete convolution

$$(f * g)[n,m] = \sum_{k,l} f[k,l]\, g[n-k, m-l]$$

- Fold g[k,l] about origin to form g[−k,−l]
- Shift the folded results by n,m to form g[n − k,m − l]
- Multiply g[n − k,m − l] by f[k, l]
- Sum over all k,l
- Repeat for every n,m



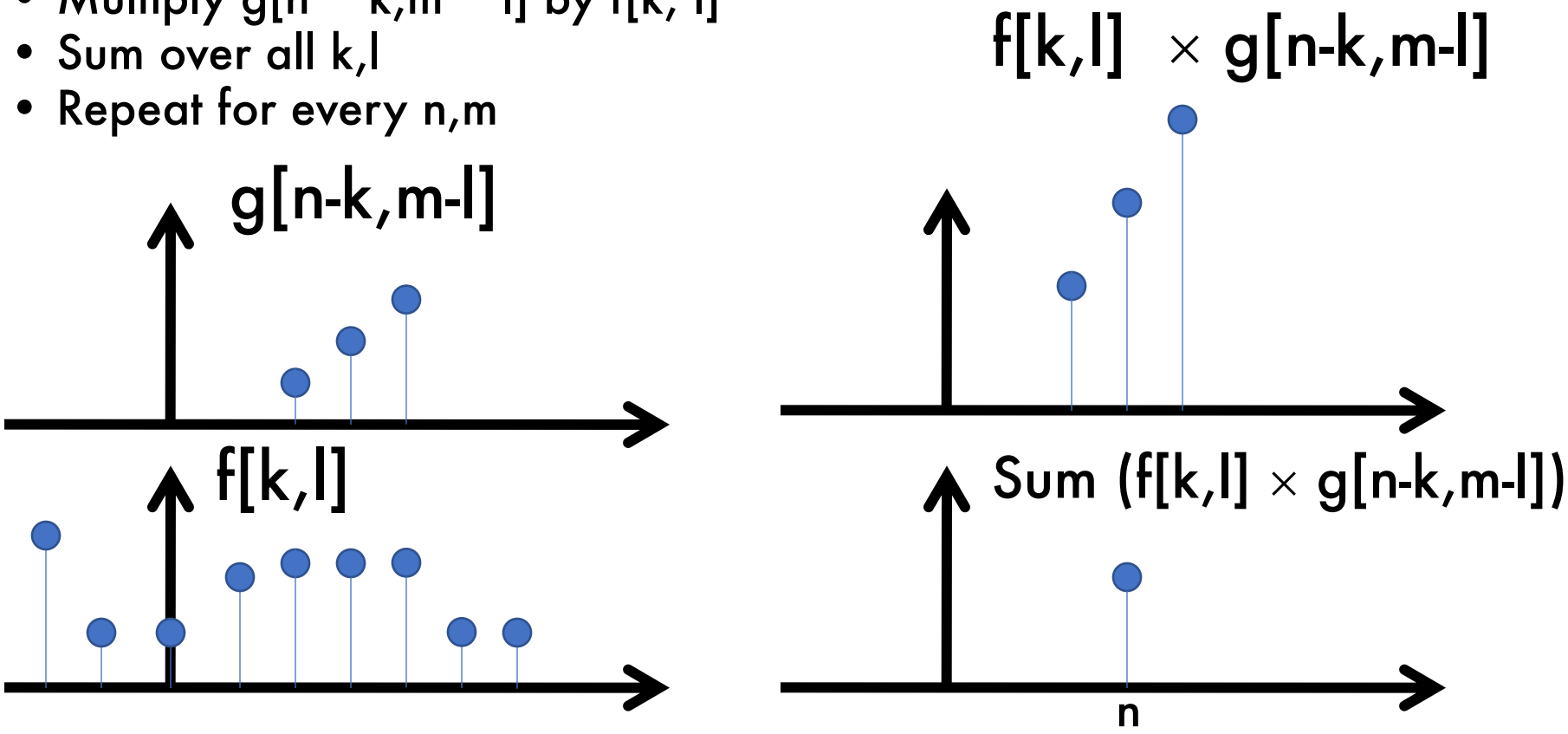




Discrete convolution

$$(f * g)[n,m] = \sum_{k,l} f[k,l]\, g[n-k, m-l]$$

- Fold g[k,l] about origin to form g[−k,−l]
- Shift the folded results by n,m to form g[n − k,m − l]
- Multiply g[n − k,m − l] by f[k, l]
- Sum over all k,l
- Repeat for every n,m

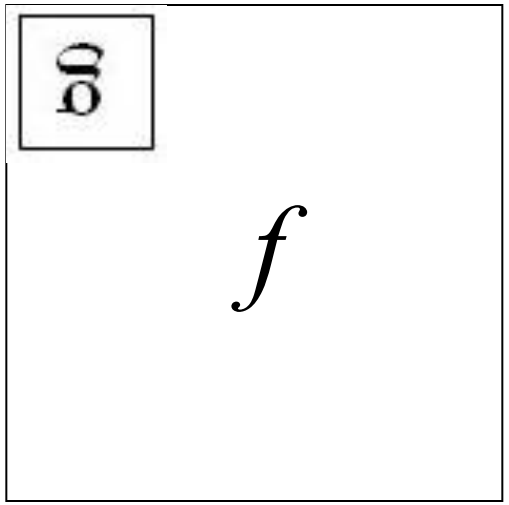


Convolution

- Let *f* be the image and *g* be the kernel. The output of convolving *f* with *g* is denoted by *f* * *g*.

$$(f * g)[n,m] = \sum_{k,l} f[k,l]\, g[n-k, m-l]$$

Weighted product of f(k,l) by g(-(k,l))) computed at different locations n,m



Box filter

- Kernel k with positive entries, that sum to 1.
- Notice: all weights are equal

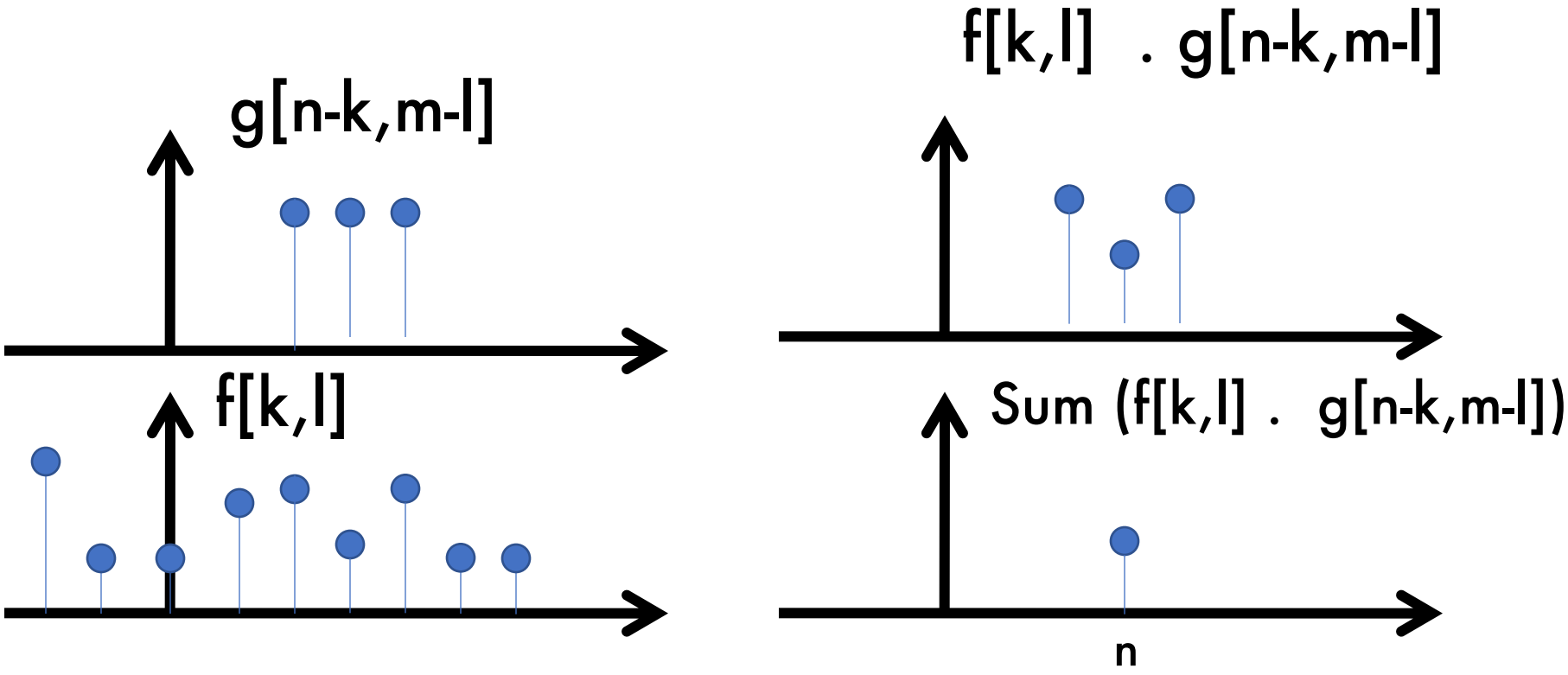
$g[\cdot,\cdot]$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Slide credit: David Lowe (UBC)

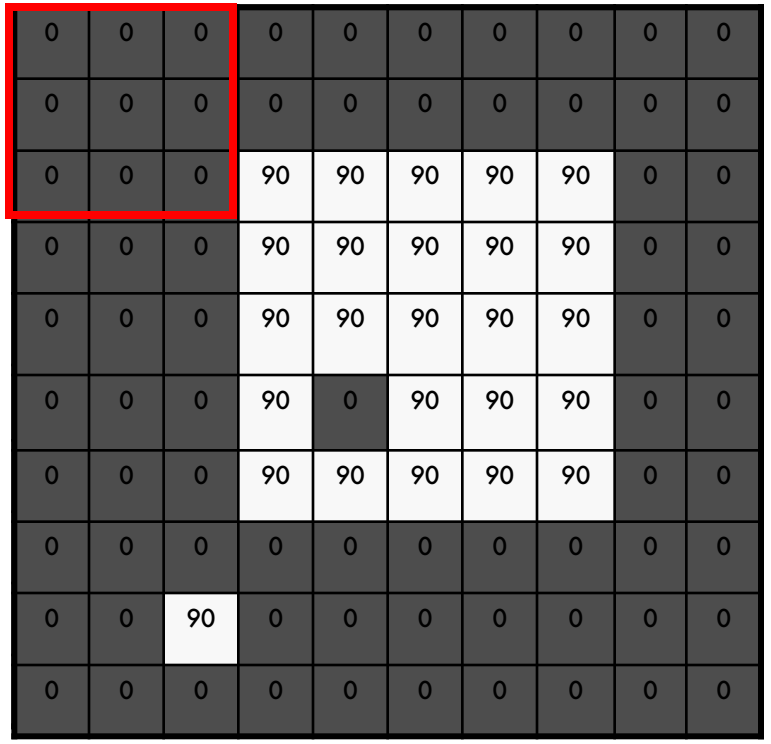
Discrete convolution

- Fold g[n,m] about origin to form g[−k,−l]
- Shift the folded results by n,m to form g[n − k,m − l]
- Multiply g[n − k,m − l] by f[k, l]
- Sum over all k,l
- Repeat for every n,m

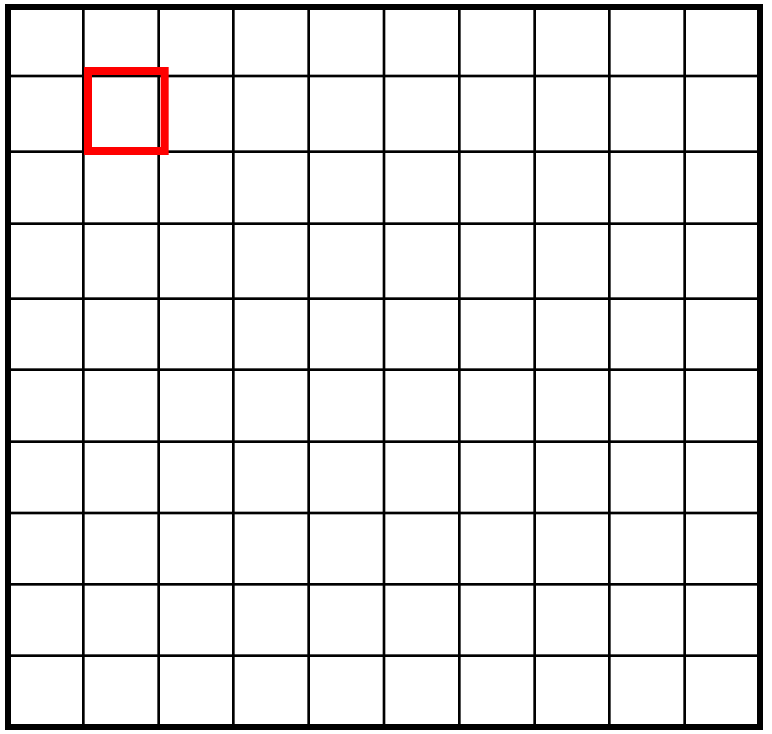


Box filter

$F[x, y]$



$G[x, y]$

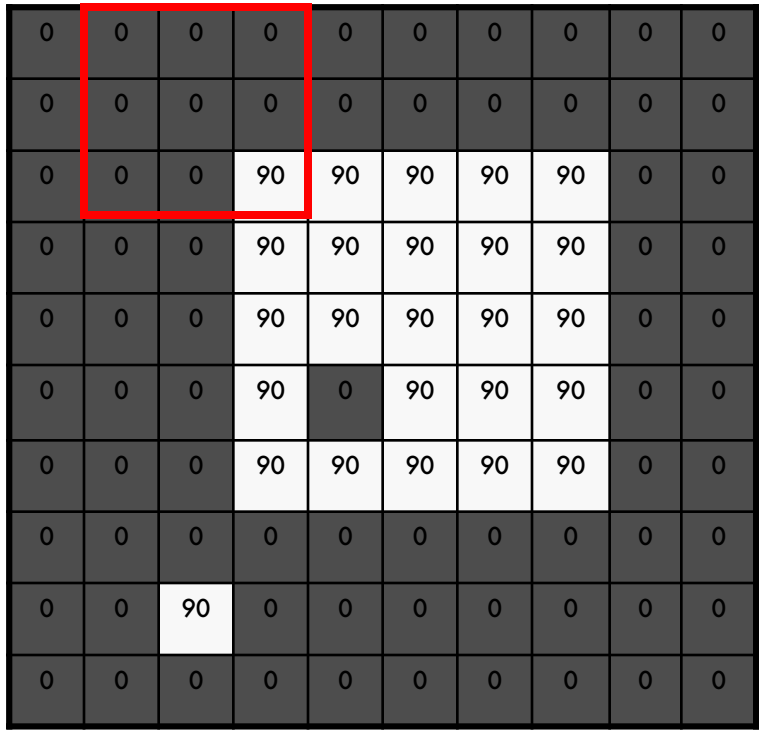


$$(f * g)[m,n] = \sum_{k,l} f[k,l]\, g[m-k, n-l]$$

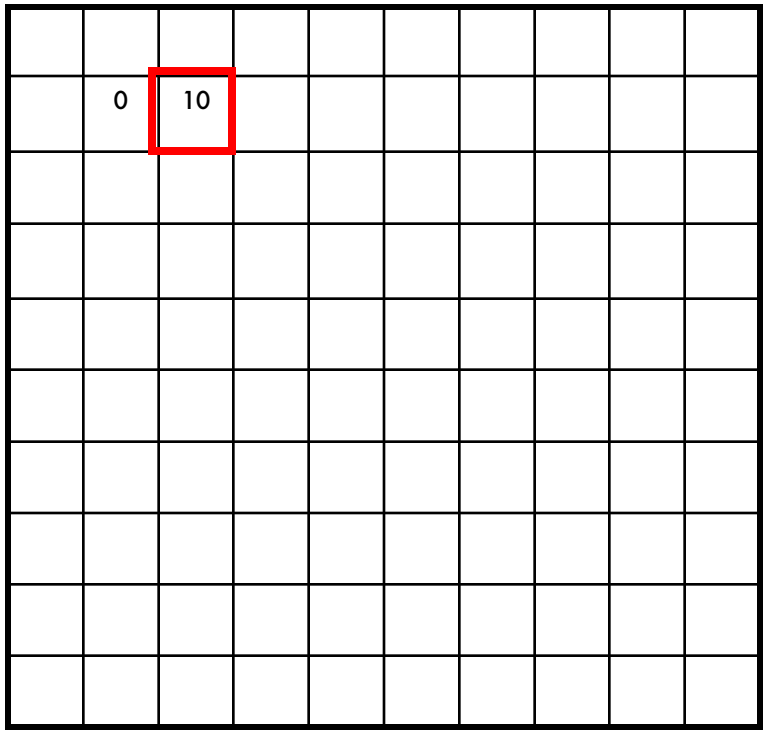
Source: S. Seitz

Box filter

$F[x, y]$



$G[x, y]$

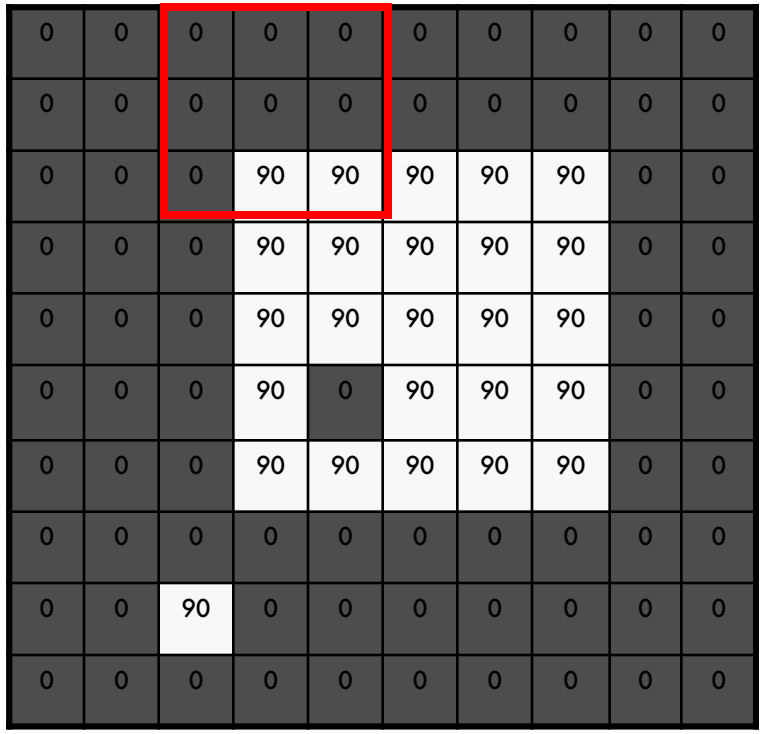


$$(f * g)[m, n] = \sum_{k,l} f[k, l]\, g[m - k, n - l]$$

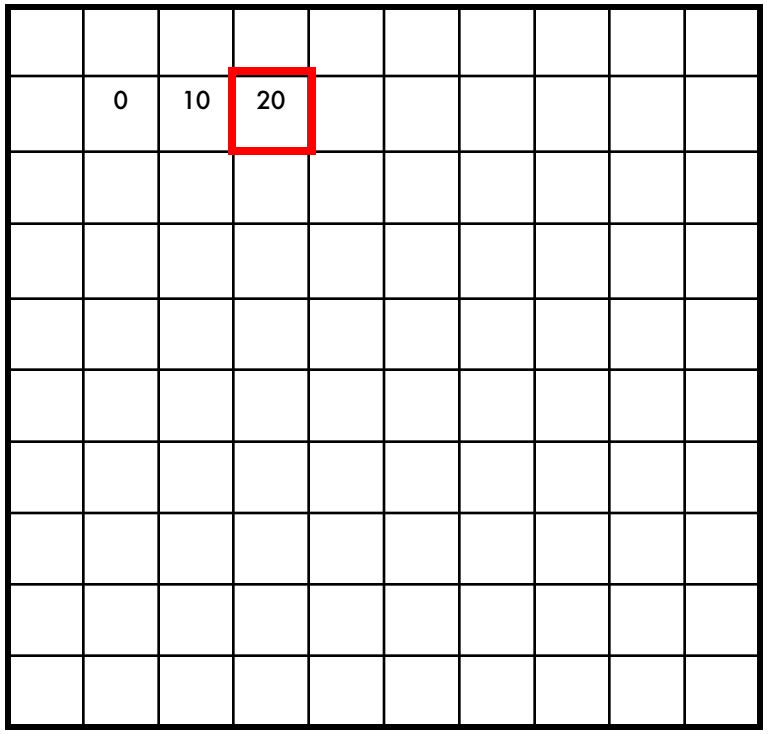
Source: S. Seitz

Box filter

$F[x, y]$



$G[x, y]$

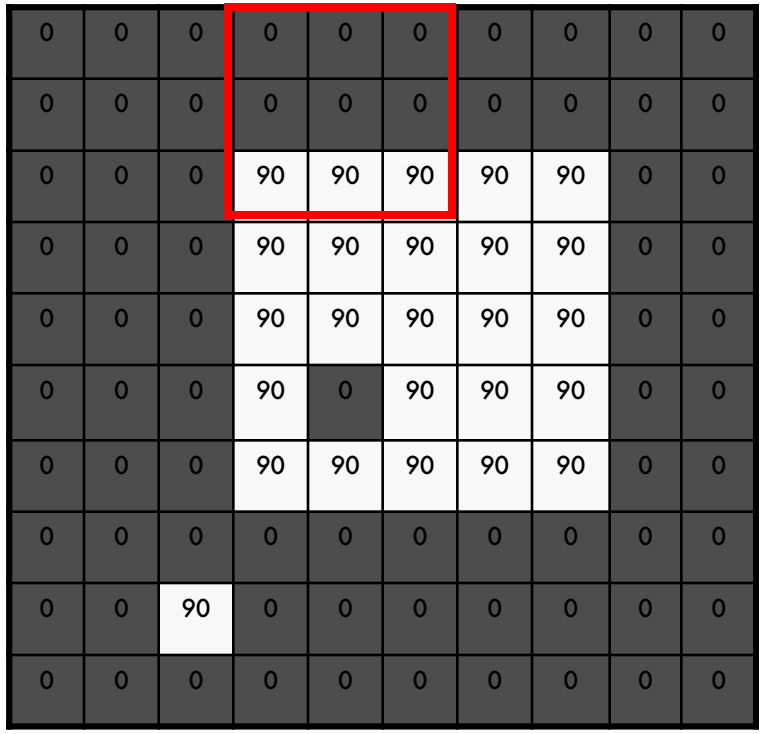


$$(f * g)[m,n] = \sum_{k,l} f[k,l]\, g[m-k, n-l]$$

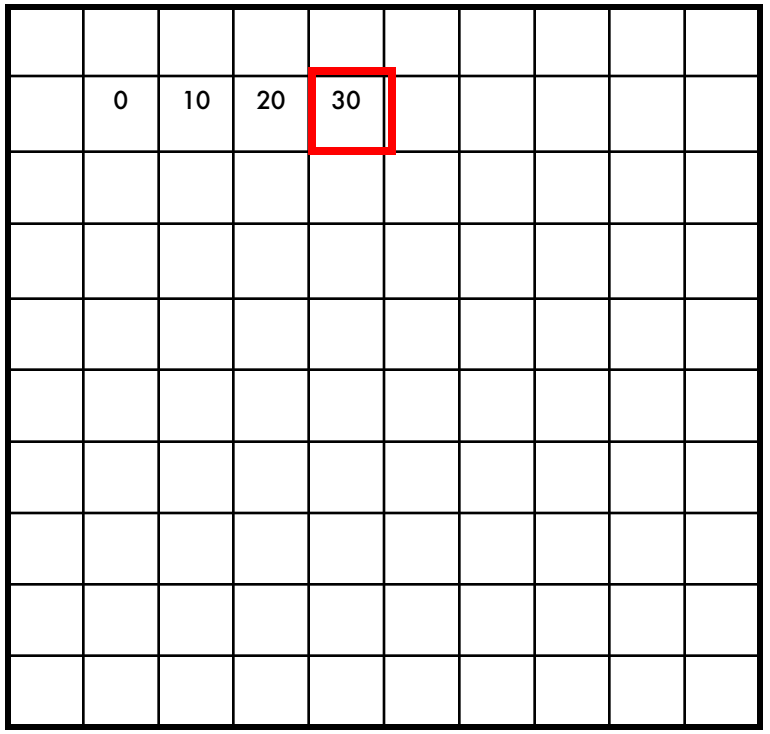
Source: S. Seitz

Box filter

$F[x, y]$



$G[x, y]$

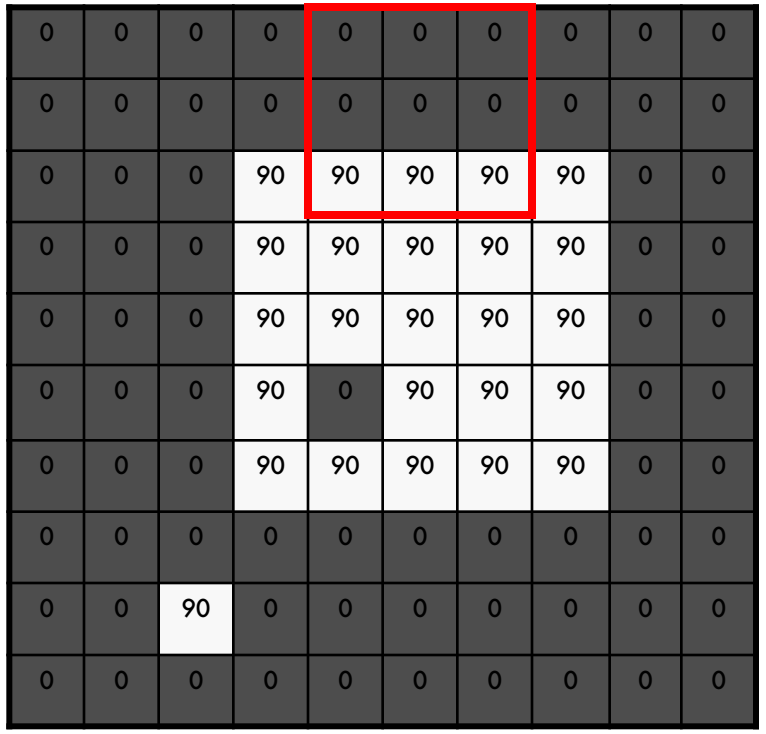


$$(f * g)[m, n] = \sum_{k,l} f[k, l]\, g[m - k, n - l]$$

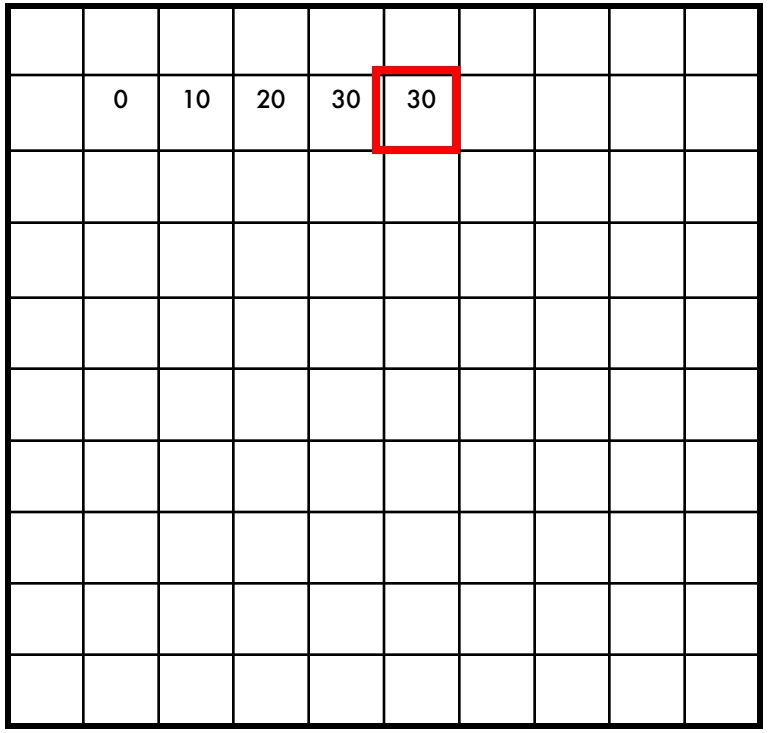
Source: S. Seitz

Box filter

$F[x, y]$



$G[x, y]$



$$(f * g)[m,n] = \sum_{k,l} f[k,l]\, g[m-k, n-l]$$

Source: S. Seitz

Box filter

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

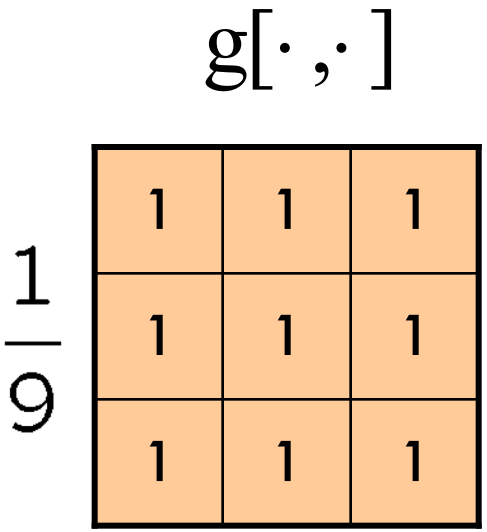
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$$(f * g)[m, n] = \sum_{k,l} f[k, l]\, g[m - k, n - l]$$

Source: S. Seitz

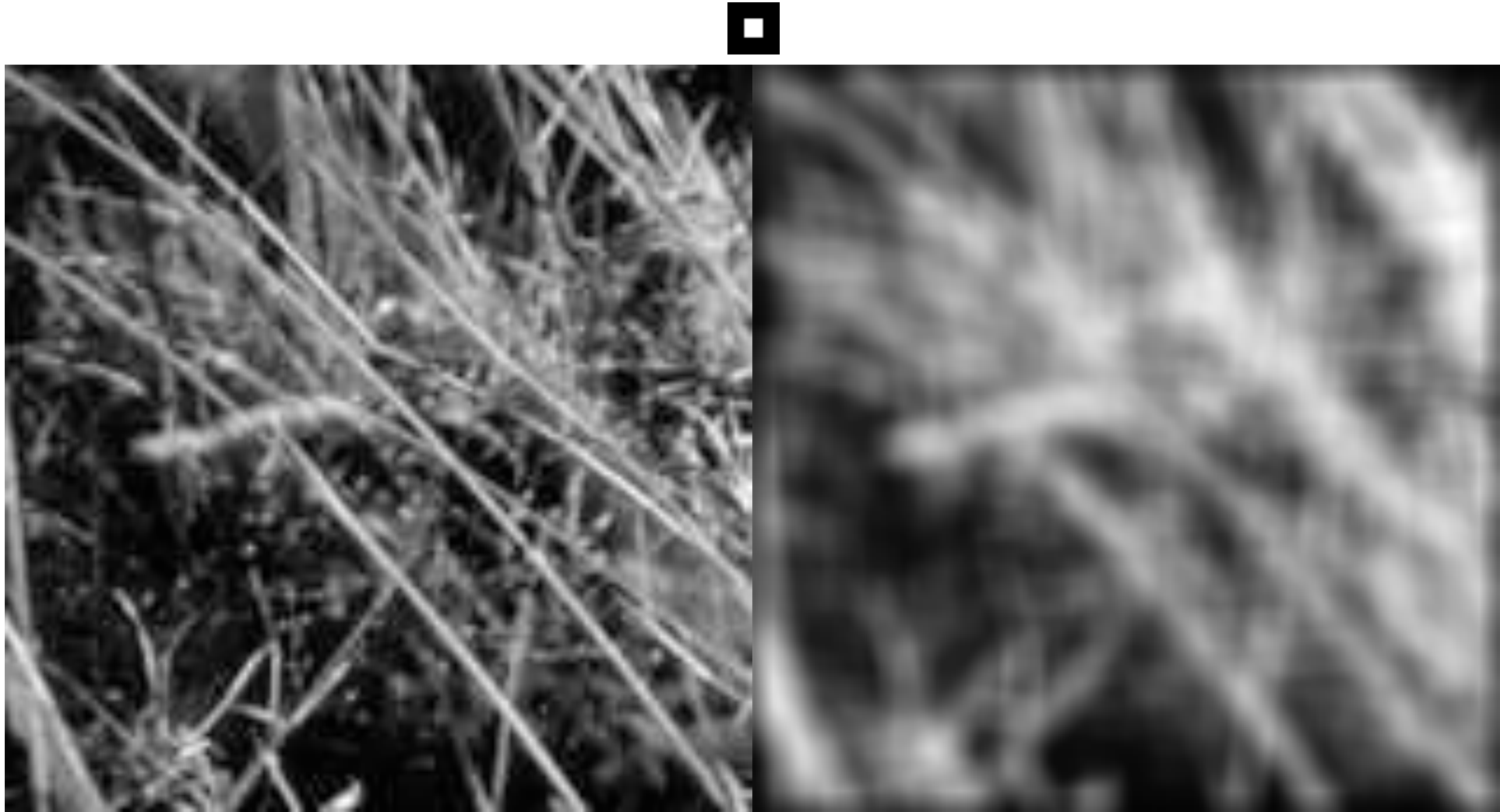
Box filter

- Replaces each pixel with an average of its neighborhood.
- Achieve smoothing effect (remove sharp features)



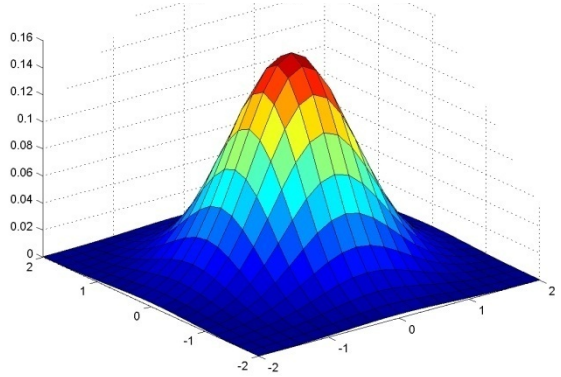
Slide credit: David Lowe (UBC)

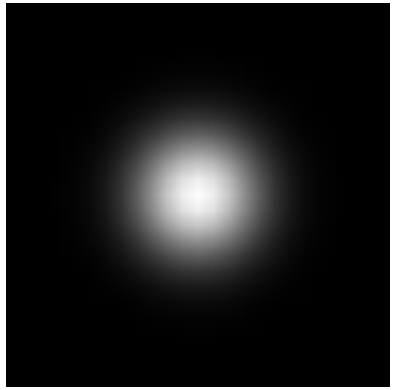
Example: Smoothing with a box filter



Smoothing with a Gaussian

- Weight contributions of neighboring pixels by nearness





0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

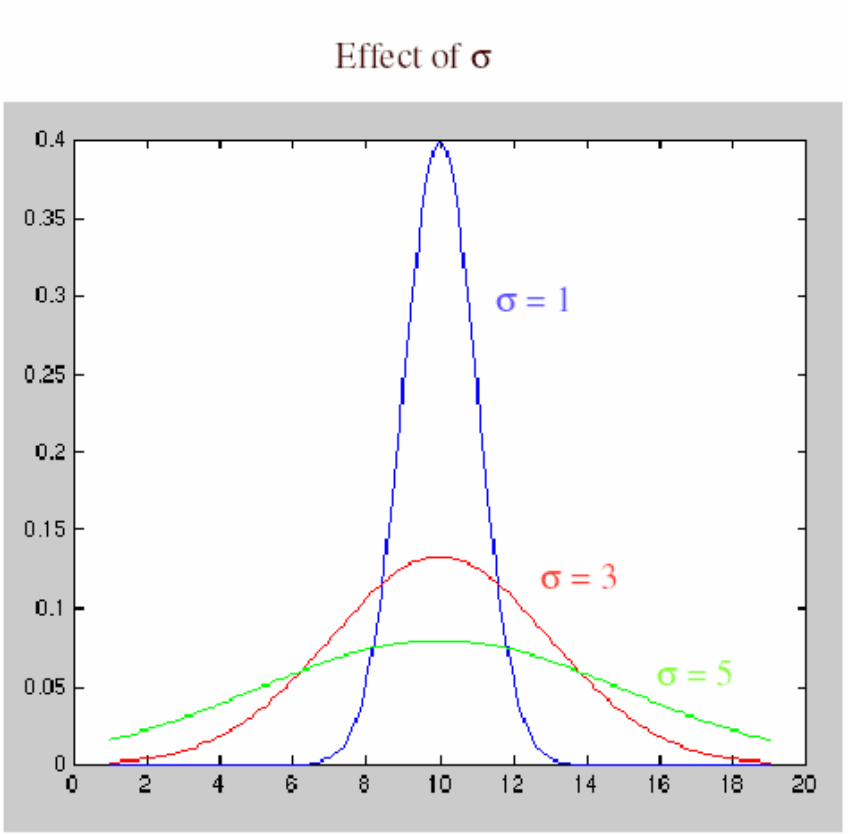
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

- Constant factor at front makes volume sum to 1 (can be ignored, as we should normalize weights to sum to 1 in any case).

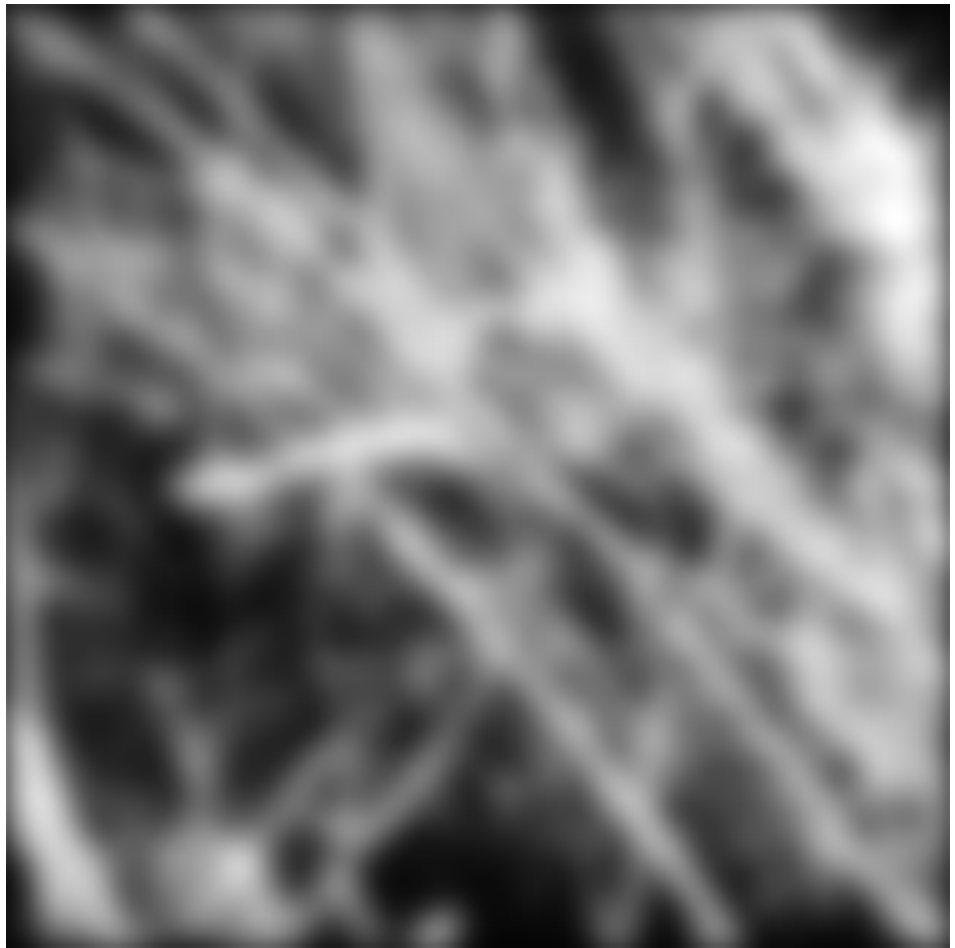
Slide credit: Christopher Rasmussen

Choosing kernel width

- Rule of thumb: set filter half-width to about 3 σ



Smoothing with a Gaussian



Gaussian noise

- Mathematical model: sum of many independent factors
- Assumption: independent, zero-mean noise

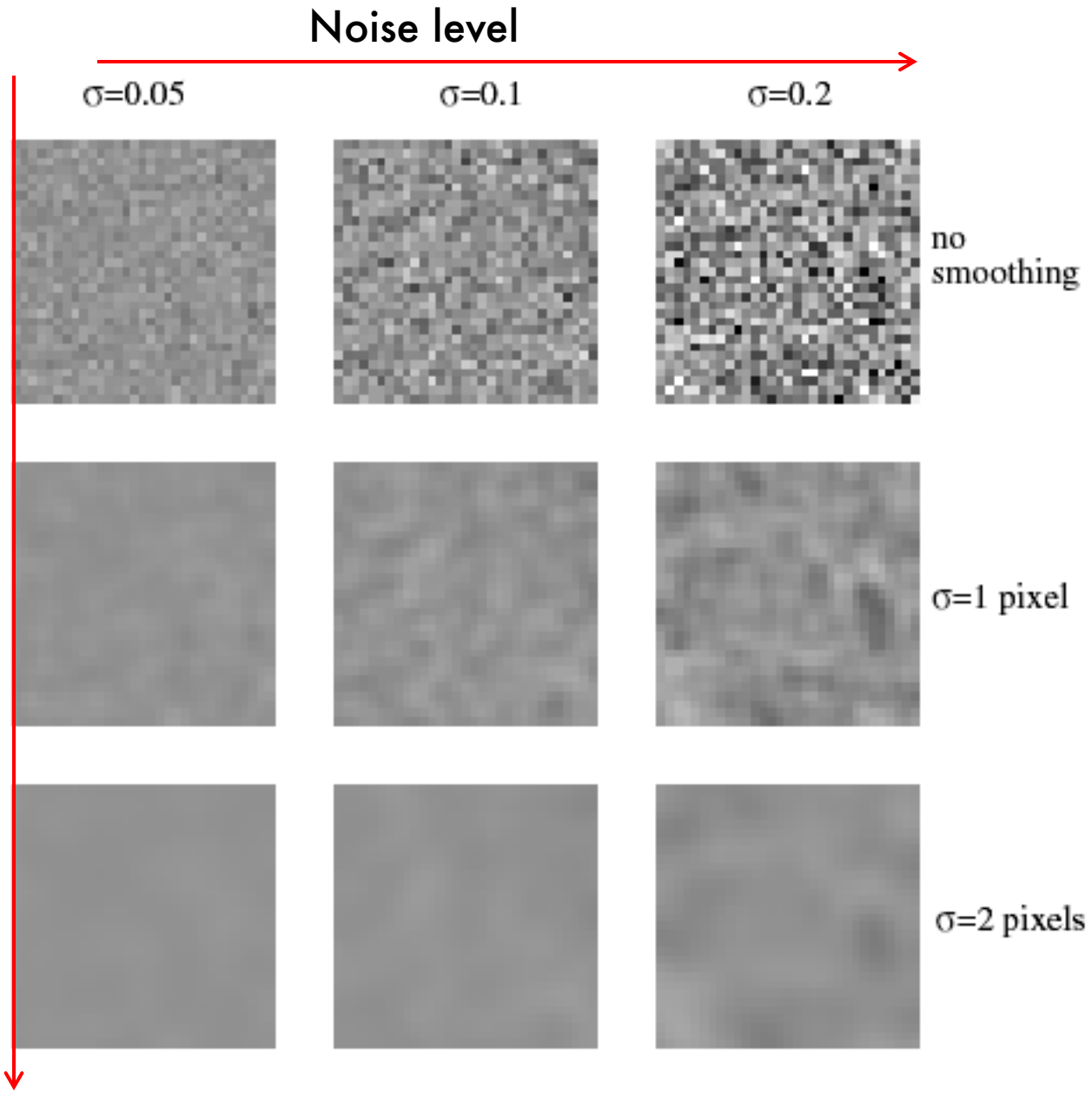
Image
Noise

$$f(x,y) = \overbrace{\bar{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

Source: K. Grauman

Smoothing with a Gaussian



Smoothing reduces pixel noise:

Each row shows smoothing with Gaussians of different width; each column shows different amounts of Gaussian noise.

Gaussian filters

- Remove “high-frequency” components from the image (low-pass filter)

- Convolution with self is another Gaussian
  - F *G (σ) * G (σ) = F * G (sqrt(σ))

- *Separable* kernel
  - Factors into product of two 1D Gaussians: $G(x,y) = G(x)\,G(y)$
  - Why this is useful?
    can convolve all rows, then all columns

Source: K. Grauman

Separability of the Gaussian filter

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{x^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{y^2}{2\sigma^2}}\right)$$

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

Source: D. Lowe

Convolution: Properties

- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal

- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$

- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$

- Scalars factor out: $ka * b = a * kb = k(a * b)$

- Identity: unit impulse e =

0	0	0
0	1	0
0	0	0

  $a * e = a$

Convolution: Properties

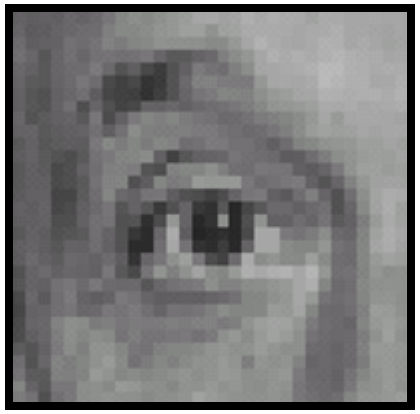
- **Linearity:** filter($f_1 + f_2$) = filter(f_1) + filter(f_2)

- **Shift invariance:** filter (shift (f)) = shift (filter (f))

  (same behavior regardless of pixel location)

- Theoretical result: any linear shift-invariant operator can be represented as a convolution

Other examples of convolution



Original

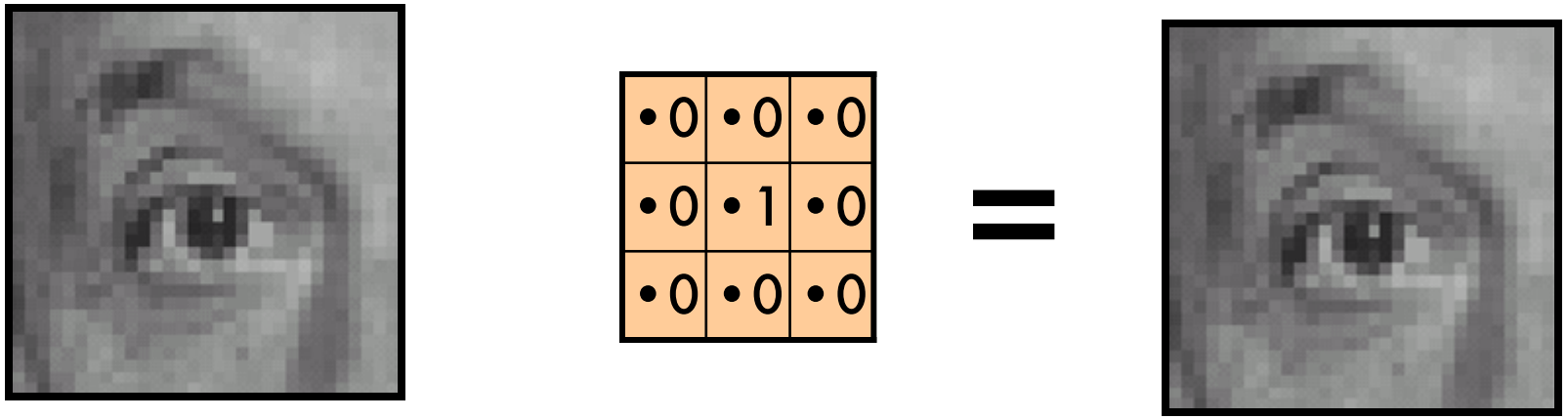
•0	•0	•0
•0	•1	•0
•0	•0	•0

=

?

Source: D. Lowe

Other examples of convolution

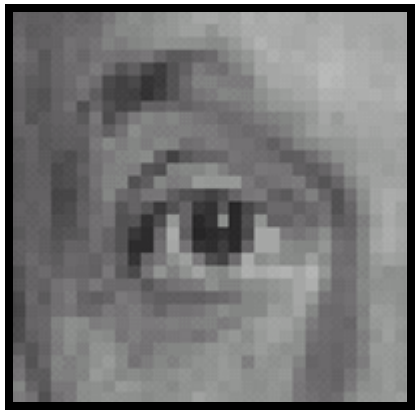


Original

Filtered
(no change)

Source: D. Lowe

Other examples of convolution



Original

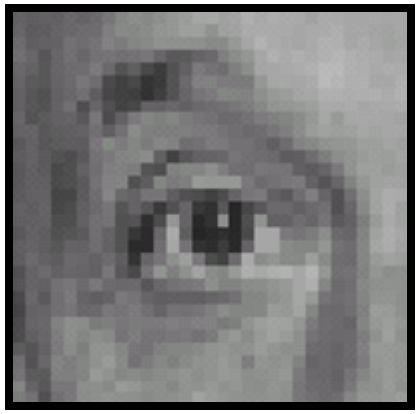
• 0	• 0	• 0
• 0	• 0	• 1
• 0	• 0	• 0

=

?

Source: D. Lowe

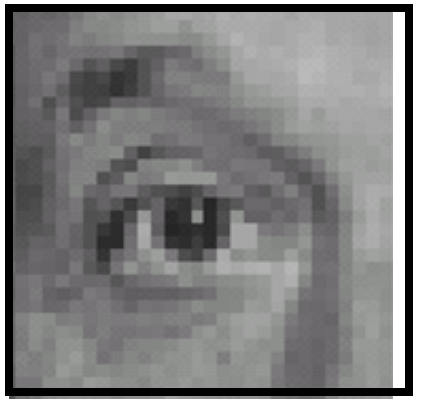
Practice with linear filters



Original

• 0	• 0	• 0
• 0	• 0	• 1
• 0	• 0	• 0

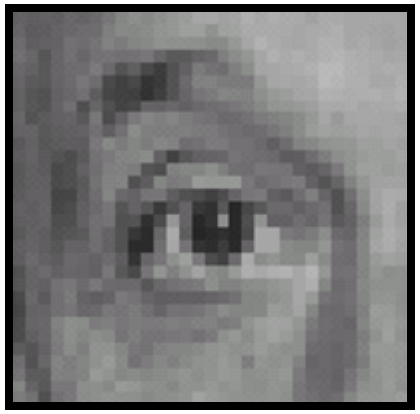
=



Shifted left
By 1 pixel

Source: D. Lowe

Other examples of convolution

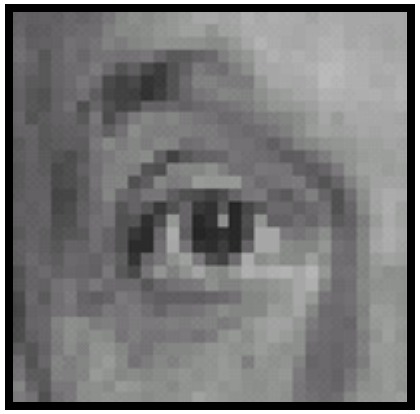


Original

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \; ?$$

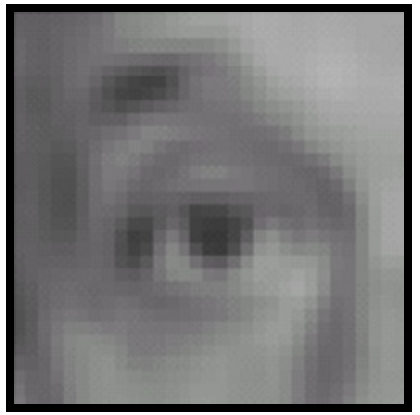
Source: D. Lowe

Other examples of convolution



Original

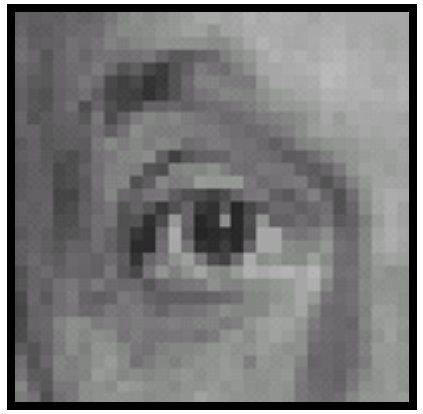
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$



Blur (with a box filter)

Source: D. Lowe

Other examples of convolution



Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

(Note that filter sums to 1)

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}\right)$$

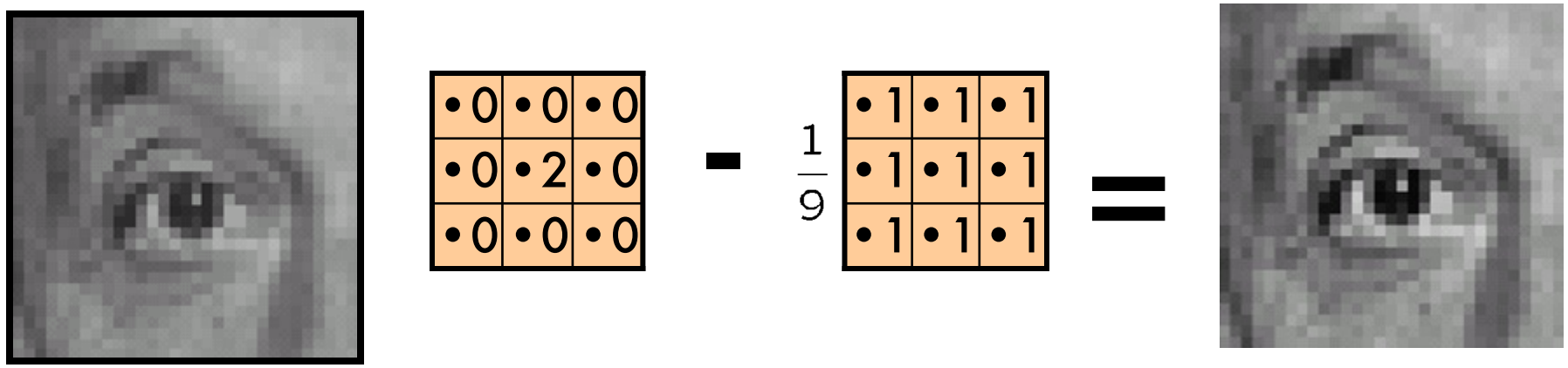
details

Source: D. Lowe

- What does blurring take away?

- Let's add it back:

Other examples of convolution



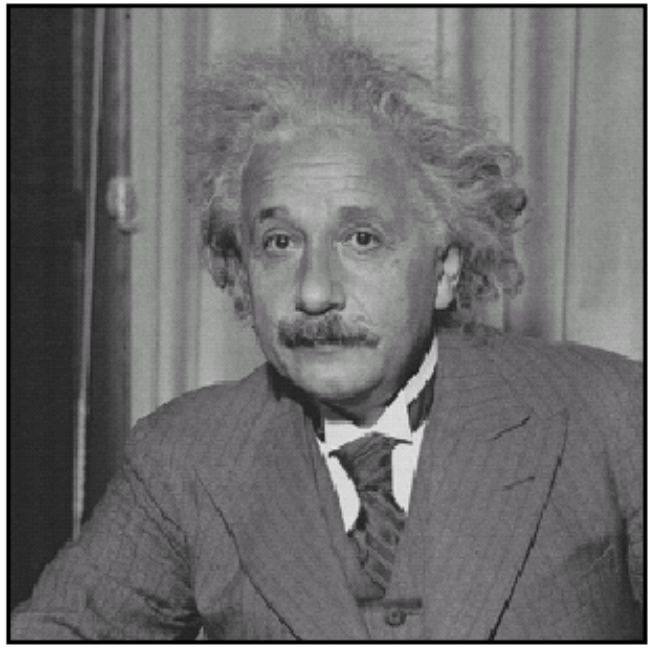
Original

Sharpening filter

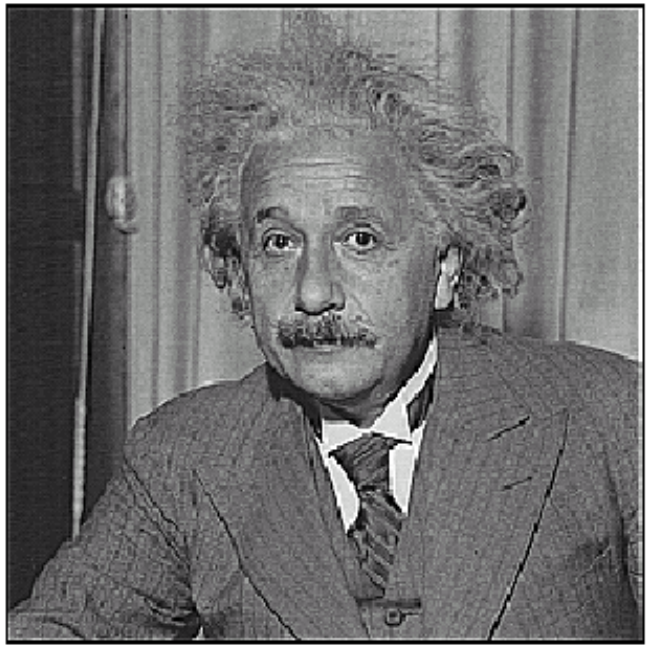
- Accentuates differences with local average

Source: D. Lowe

Sharpening



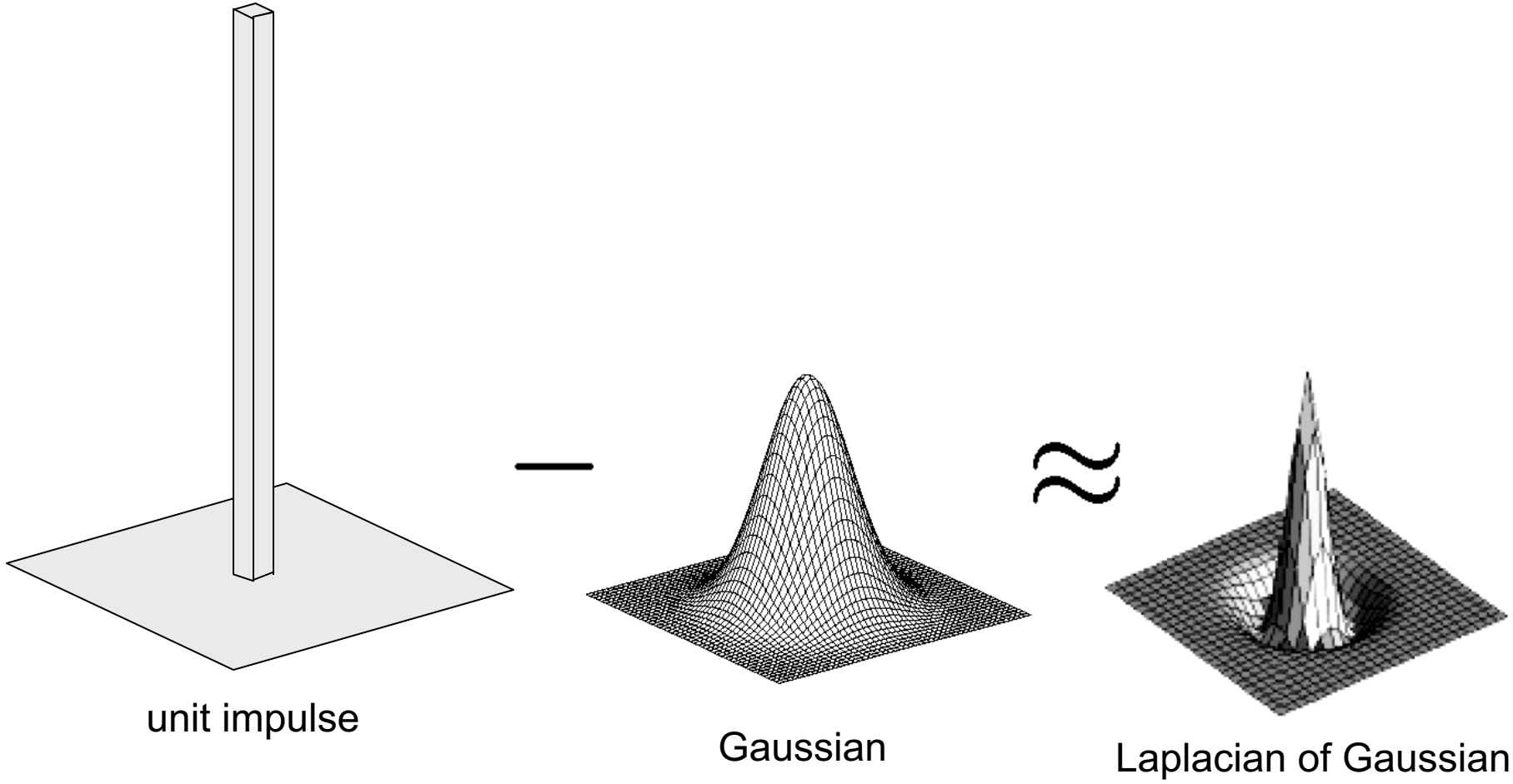
before



after

Source: D. Lowe

Sharpening



unit impulse

Gaussian

Laplacian of Gaussian

$$f * ((1 + \alpha)e - g)$$

Impulse function

Differentiation by convolution

- Recall, for 2D function, f(x,y):

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \left(\frac{f(x+\varepsilon, y)}{\varepsilon} - \frac{f(x,y)}{\varepsilon} \right)$$

- This is linear and shift invariant -> convolution

- We could approximate this as

$$\frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

2D Kernel

-1 1

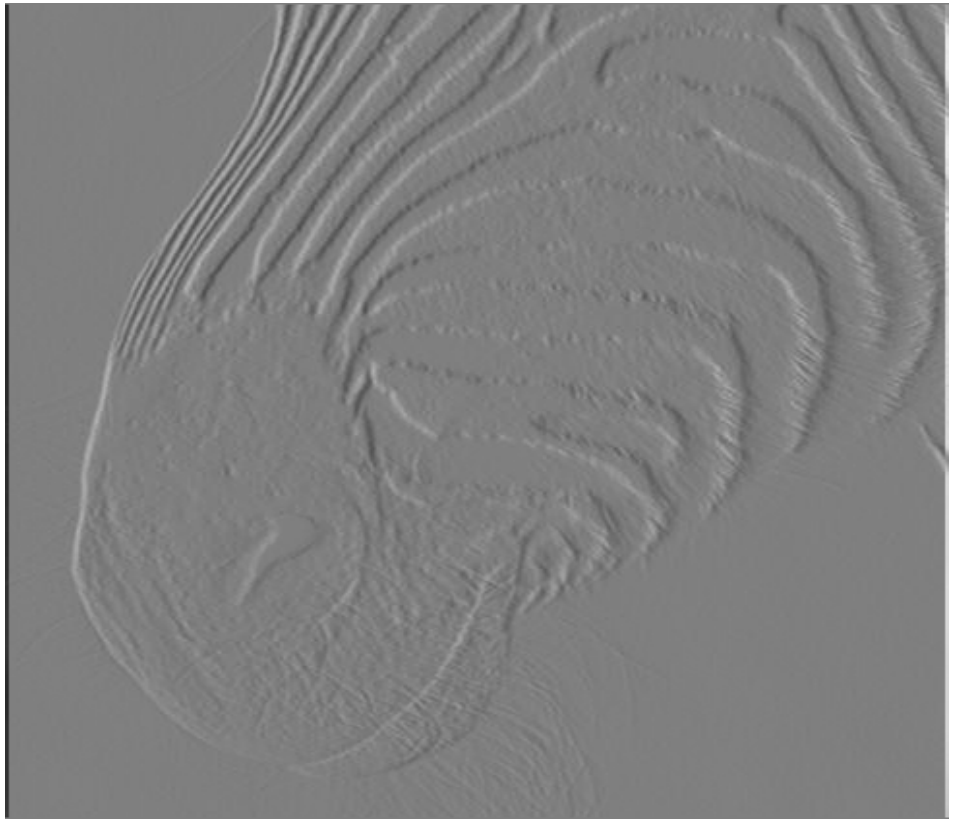
-1 1

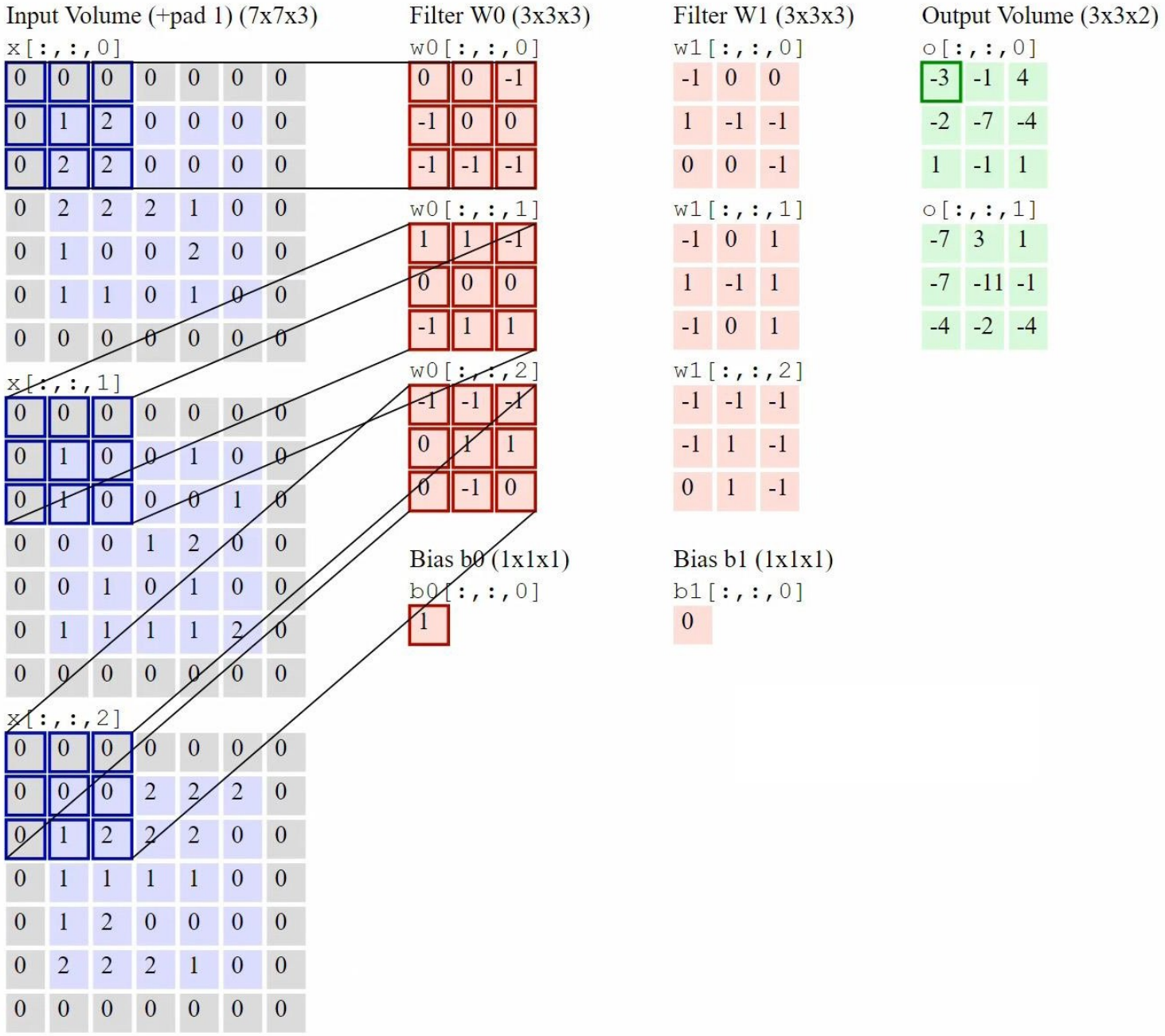
Rudimentary edge detector!

Differentiation by convolution

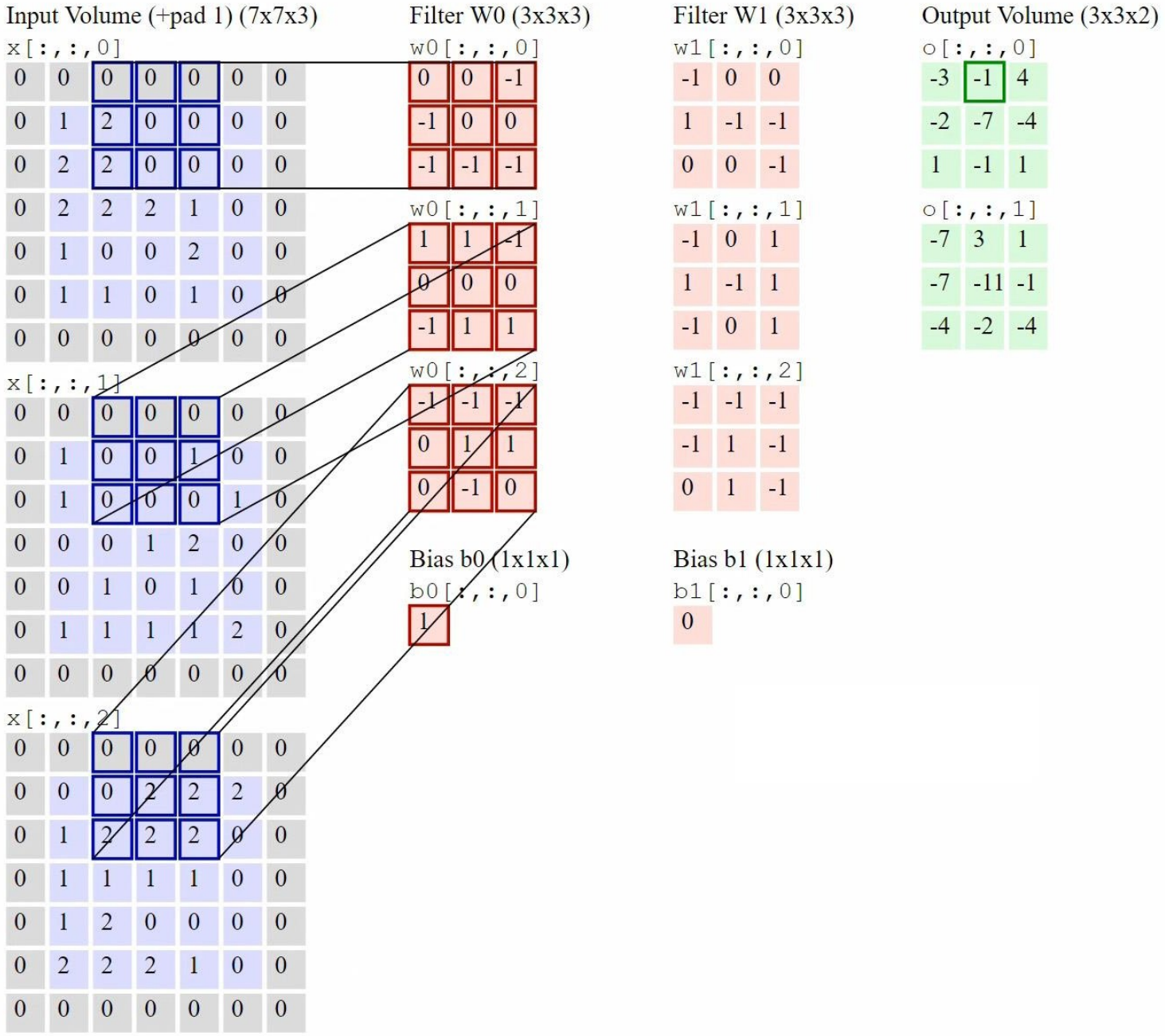
Directional Derivatives Of A Binary Image			
Original Image 			
2D Kernel 0 0 0 0 1 0 0 0 0	2D Kernel -1 0 1 -1 0 1 -1 0 1	2D Kernel -2 -1 0 1 2 -2 -1 0 1 2 -2 -1 0 1 2 -2 -1 0 1 2 -2 -1 0 1 2	2D Kernel -3 -2 -1 0 1 2 3 -3 -2 -1 0 1 2 3 -3 -2 -1 0 1 2 3 -3 -2 -1 0 1 2 3 -3 -2 -1 0 1 2 3 -3 -2 -1 0 1 2 3 -3 -2 -1 0 1 2 3

Differentiation by convolution

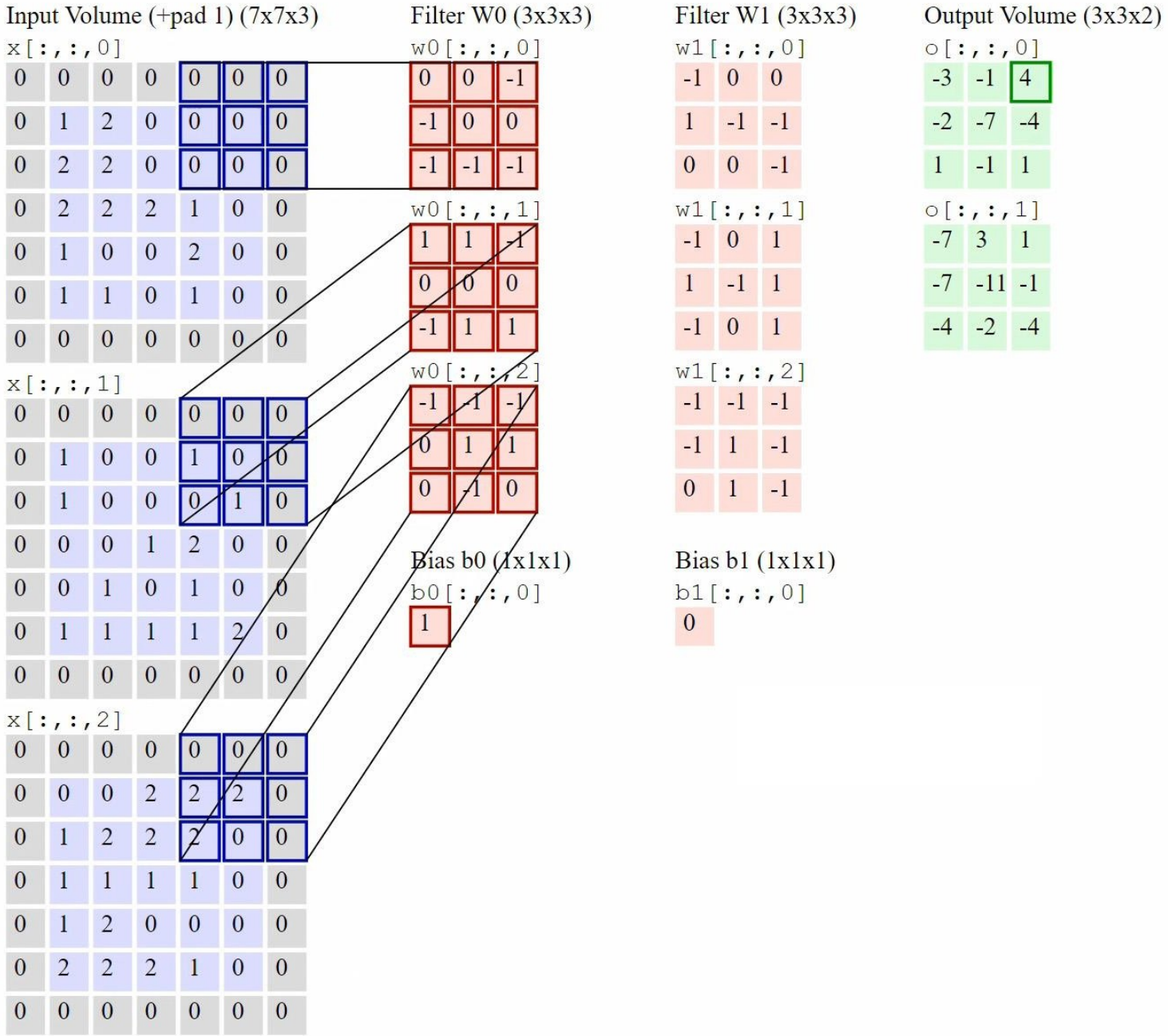




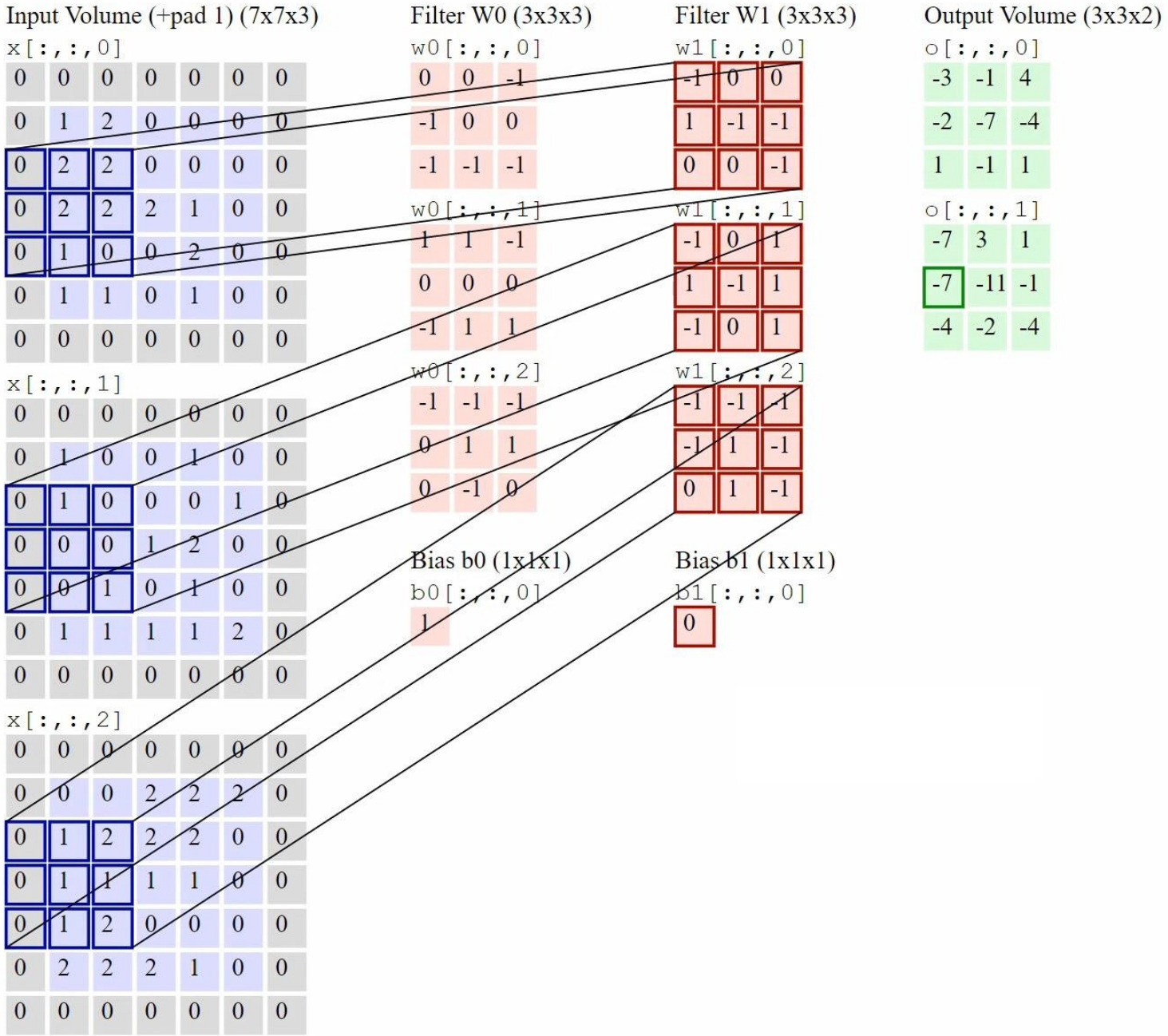
Slides stolen L
Fridman



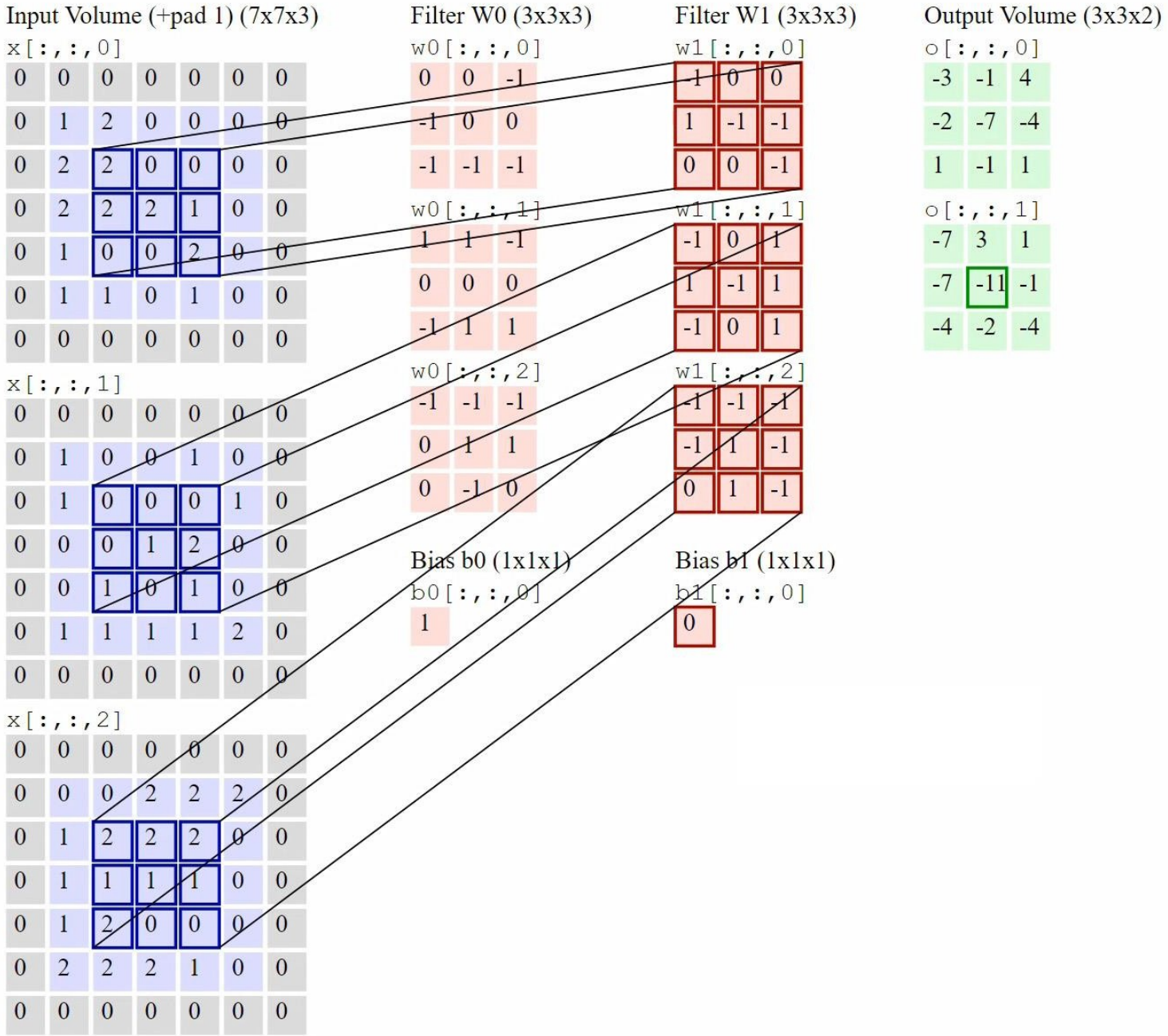
Slides stolen L
Fridman



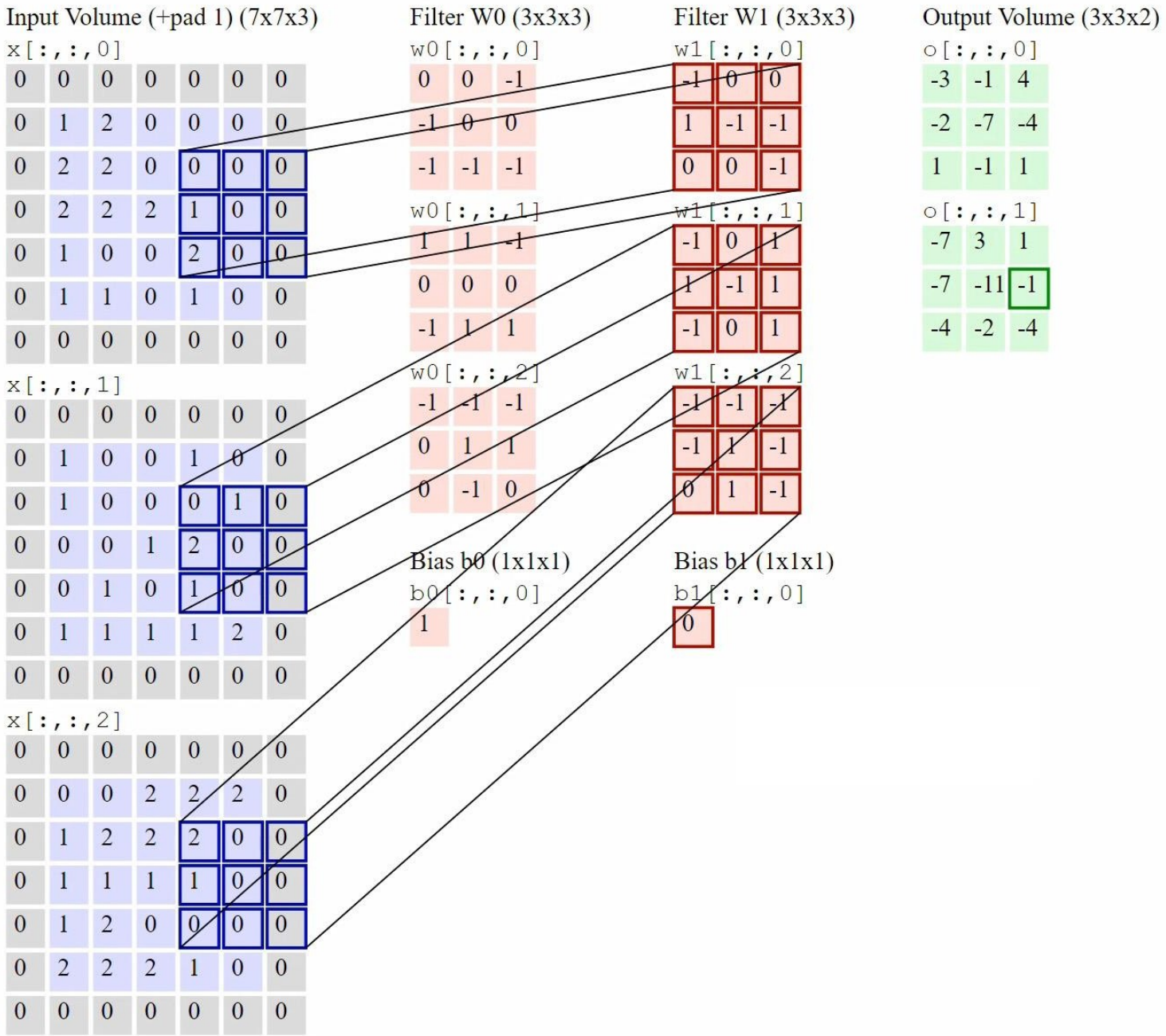
Slides stolen L
Fridman



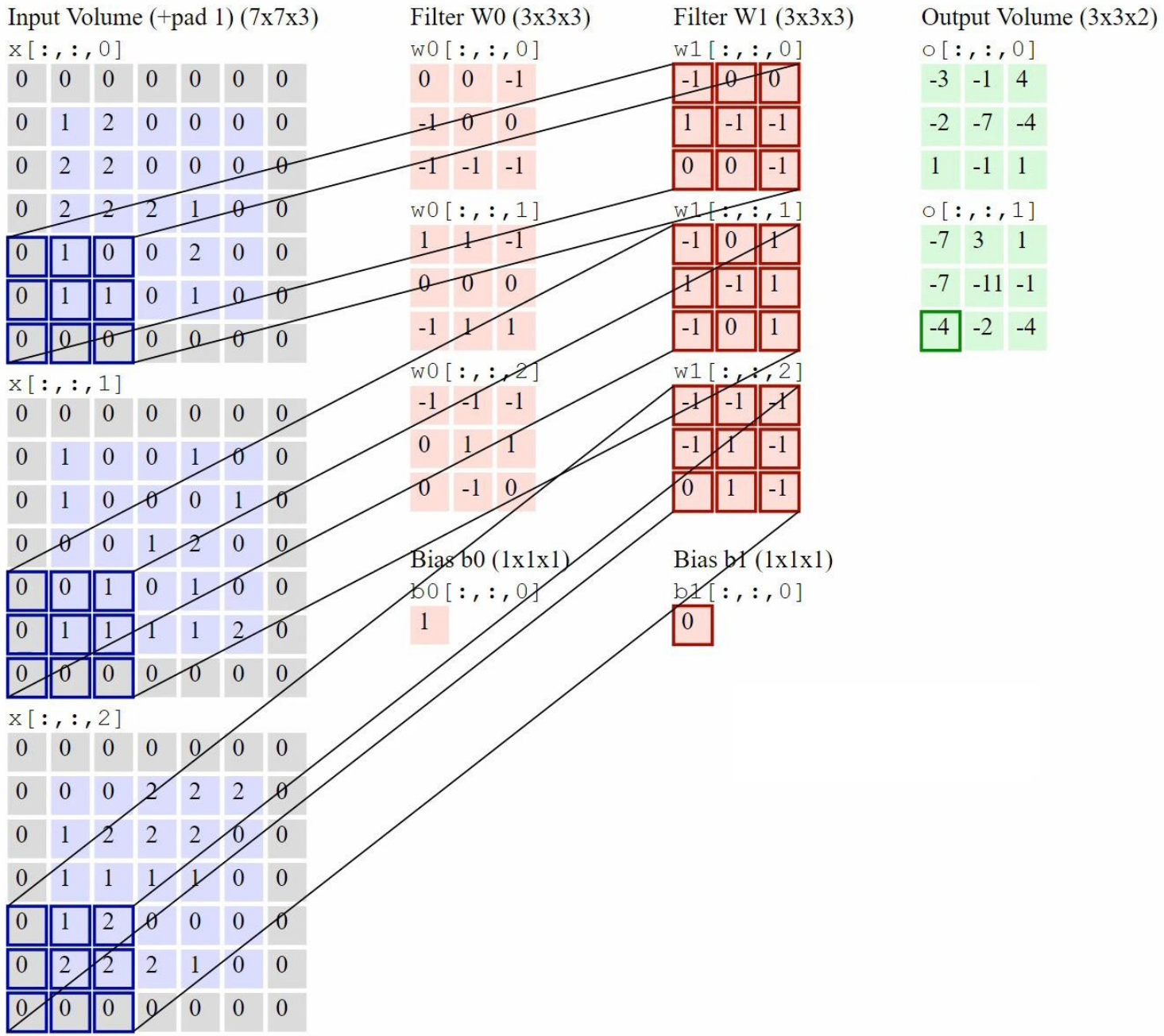
Slides stolen L
Fridman



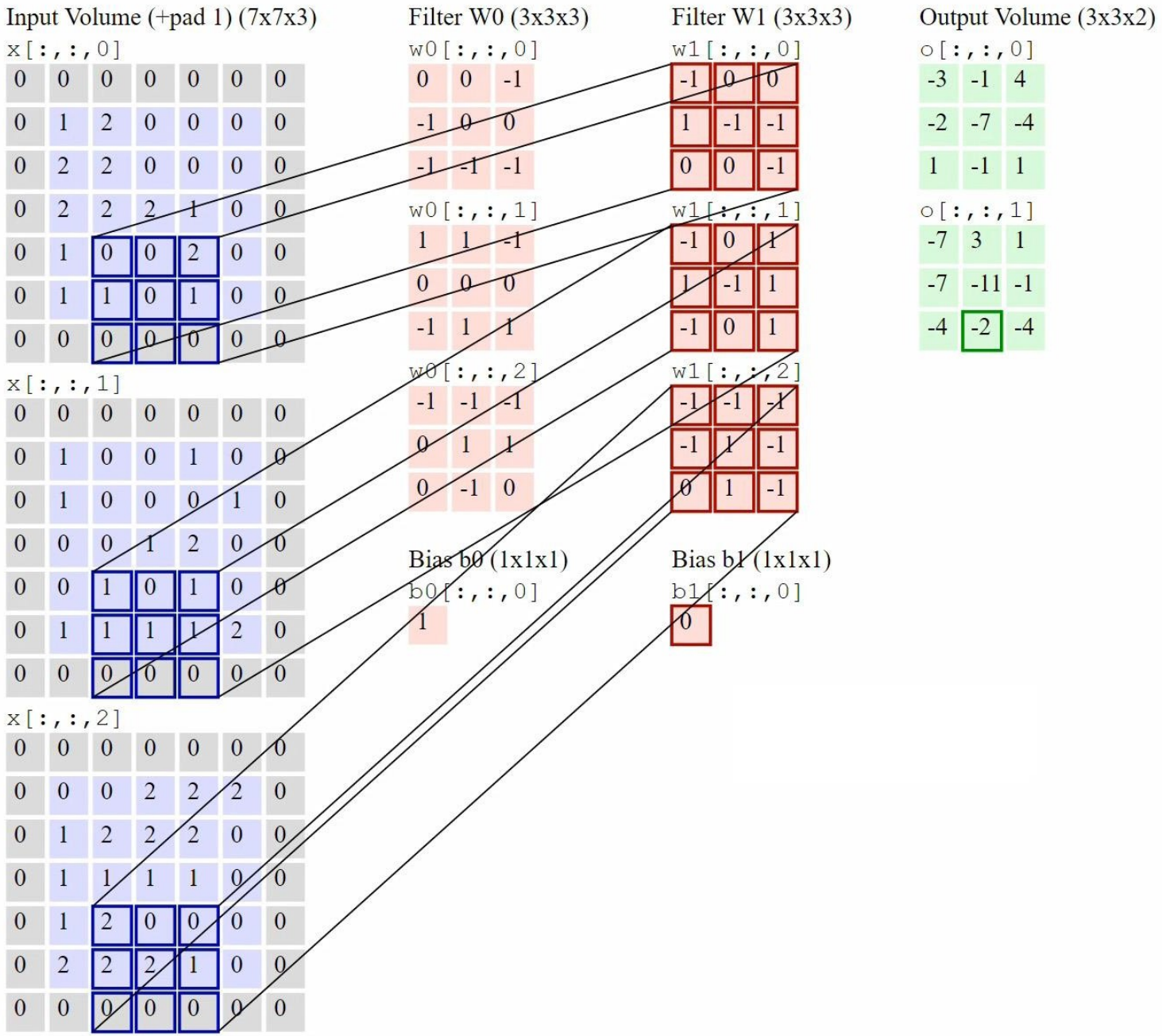
Slides stolen L
Fridman



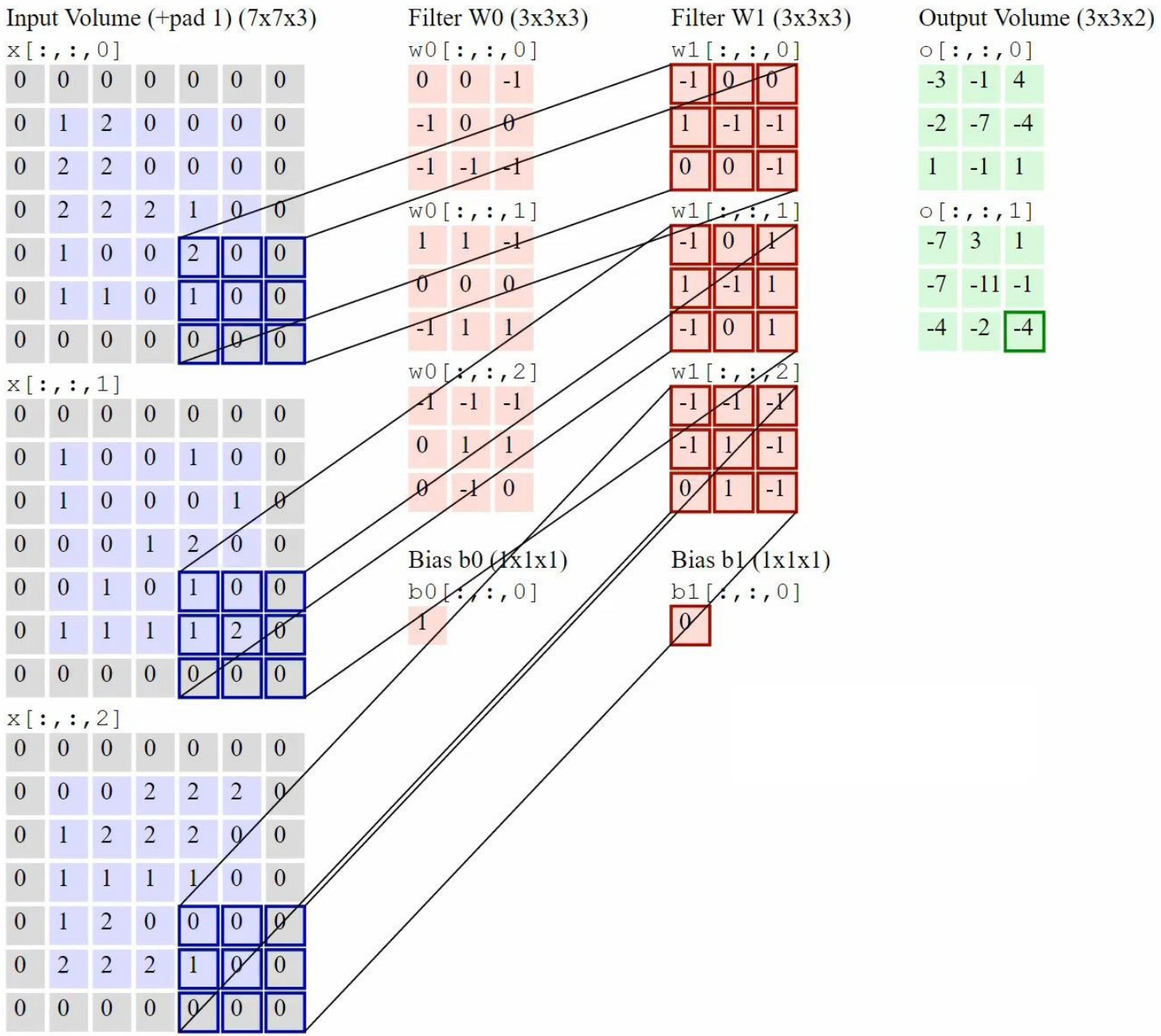
Slides stolen L
Fridman



Slides stolen L
Fridman

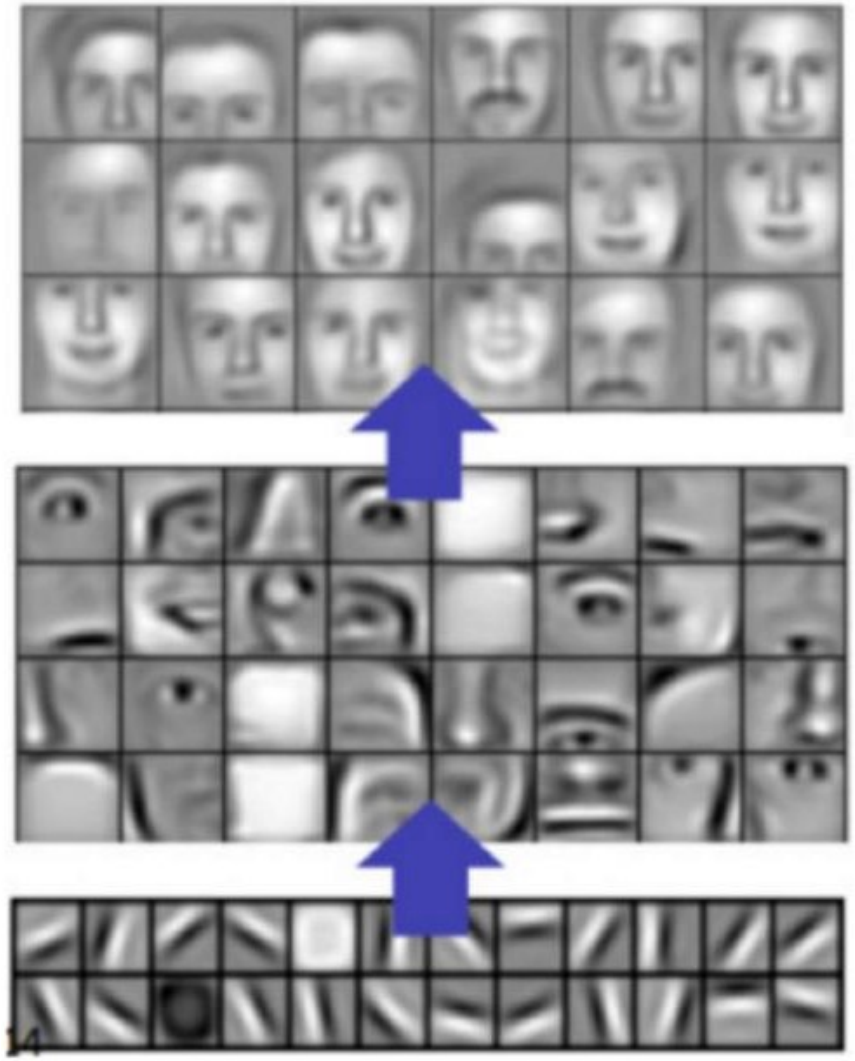


Slides stolen L
Fridman



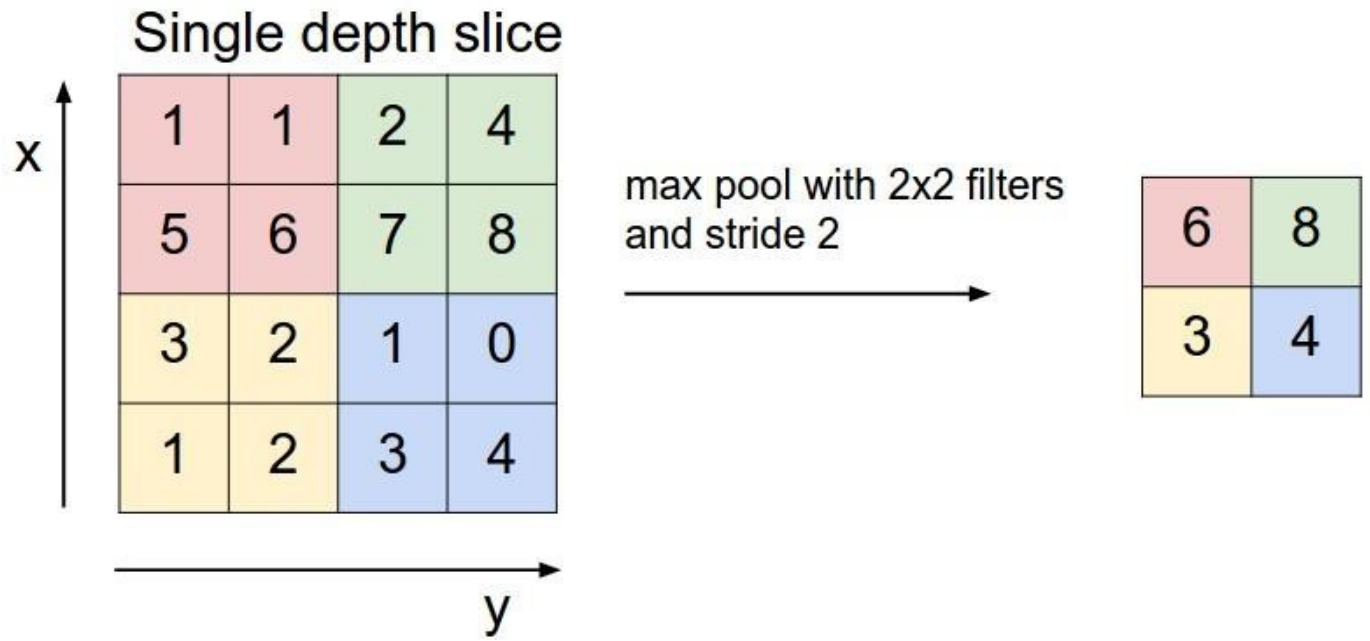
Slides stolen L
Fridman

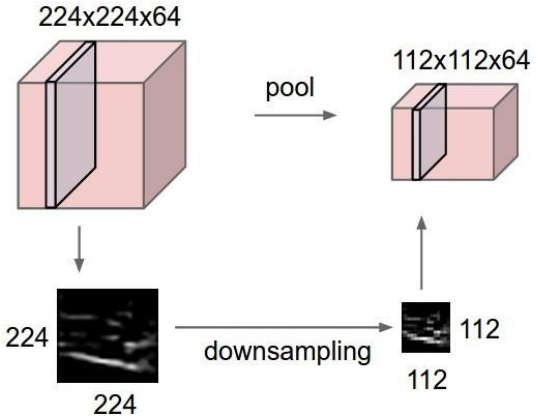
Convolution: Representation Learning



Slides Stolen L. Fridman:

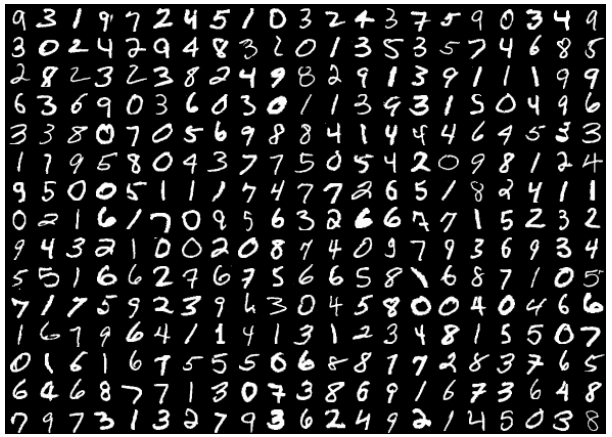
ConvNets: Pooling



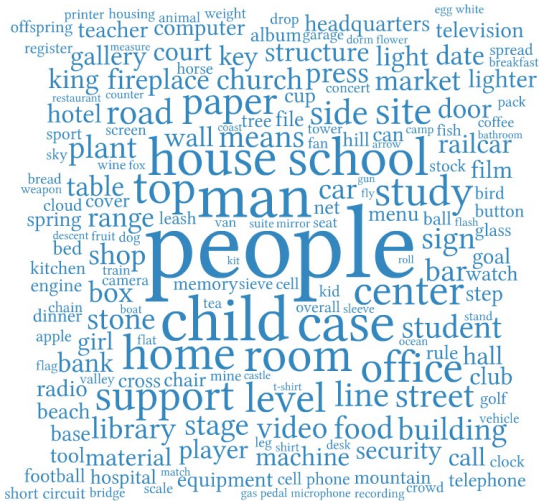


Slides Stolen L. Fridman:

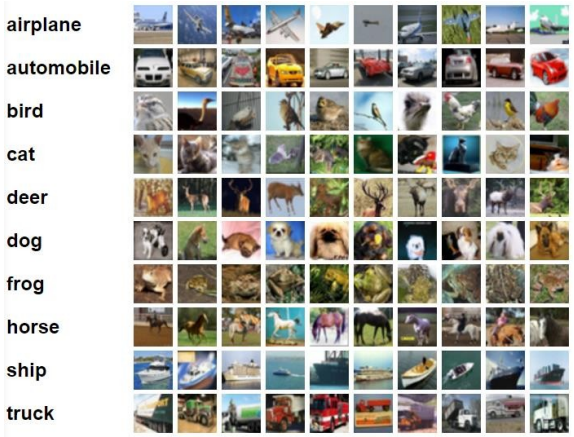
Famous Computer Vision Datasets



MNIST: handwritten digits



ImageNet: WordNet hierarchy

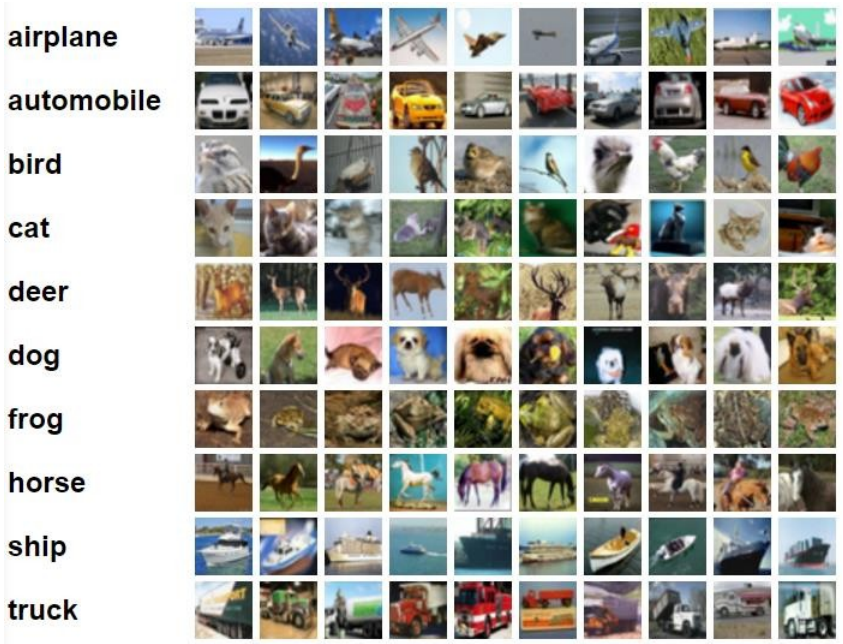


CIFAR-10(0): tiny images

Places: natural scenes

(Slides stolen from L Fridman)

Let’s Build an Image Classifier for CIFAR-10



test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

→ 456

Slides stolen L Fridman

Let’s Build an Image Classifier for CIFAR-10

test image					training image					pixel-wise absolute value differences				
56	32	10	18	-	10	20	24	17	=	46	12	14	1	→ 456
90	23	128	133		8	10	89	100		82	13	39	33	
24	26	178	200		12	16	178	170		12	10	0	30	
2	0	255	220		4	32	233	112		2	32	22	108	

Accuracy

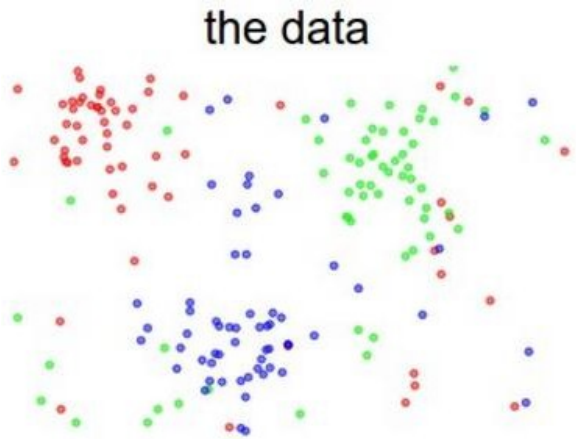
Random: **10%**

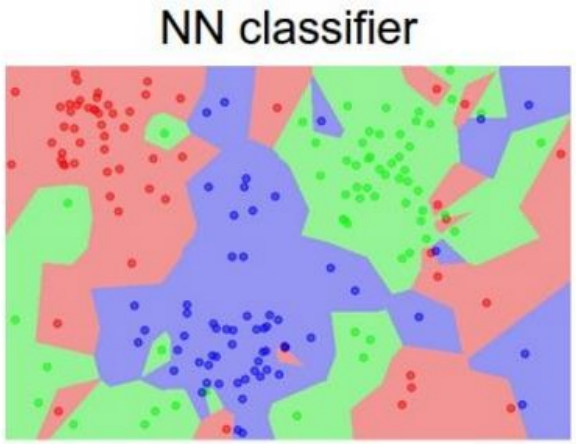
Our image-diff (with L1): **38.6%**

Our image-diff (with L2): **35.4%**

Slides stolen L Fridman

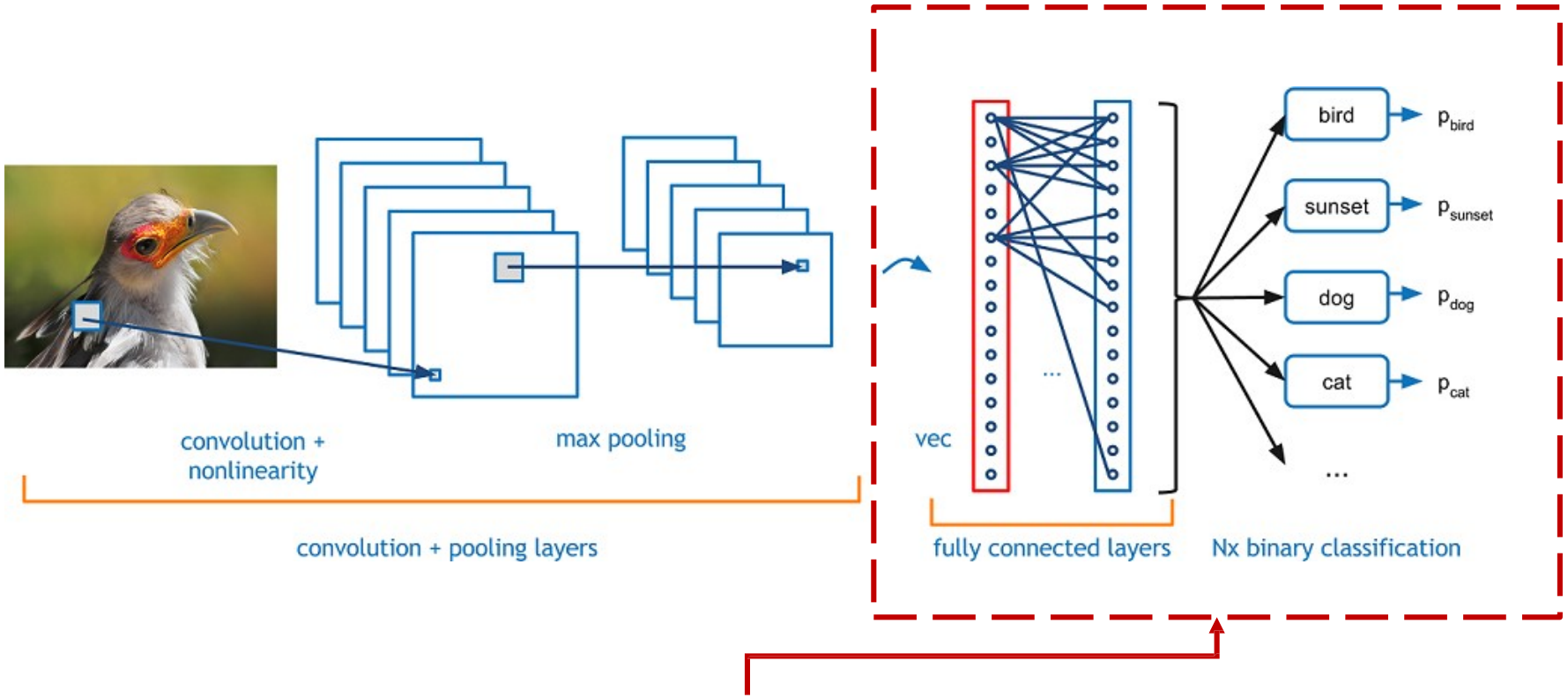
K-Nearest Neighbors: Generalizing the Image-Diff Classifier





- Accuracy
- Random: 10%
- Training and testing on the same data: 35.4%
- 7-Nearest Neighbors: ~30%
- Human: ~94%
- …
- Convolutional Neural Networks: ~95%

Same Architecture, Many Applications



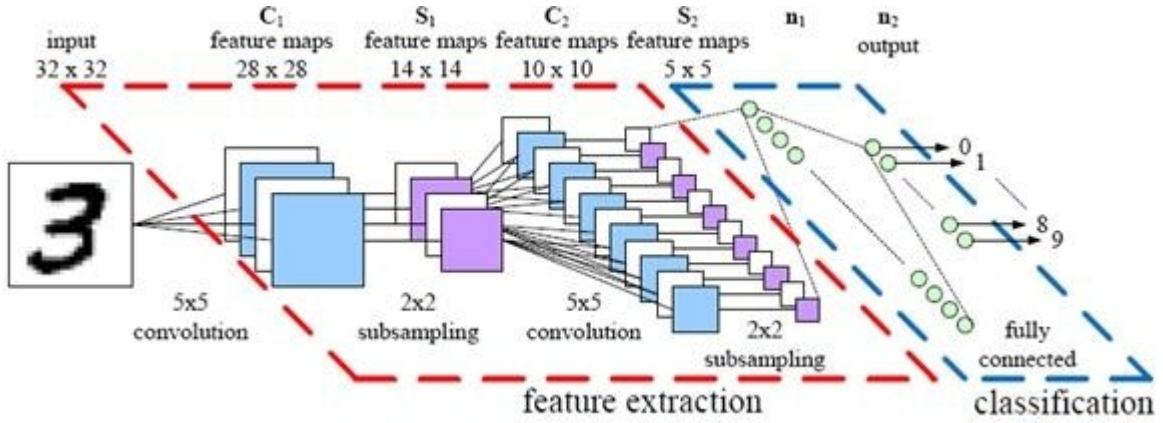
This part might look different for:

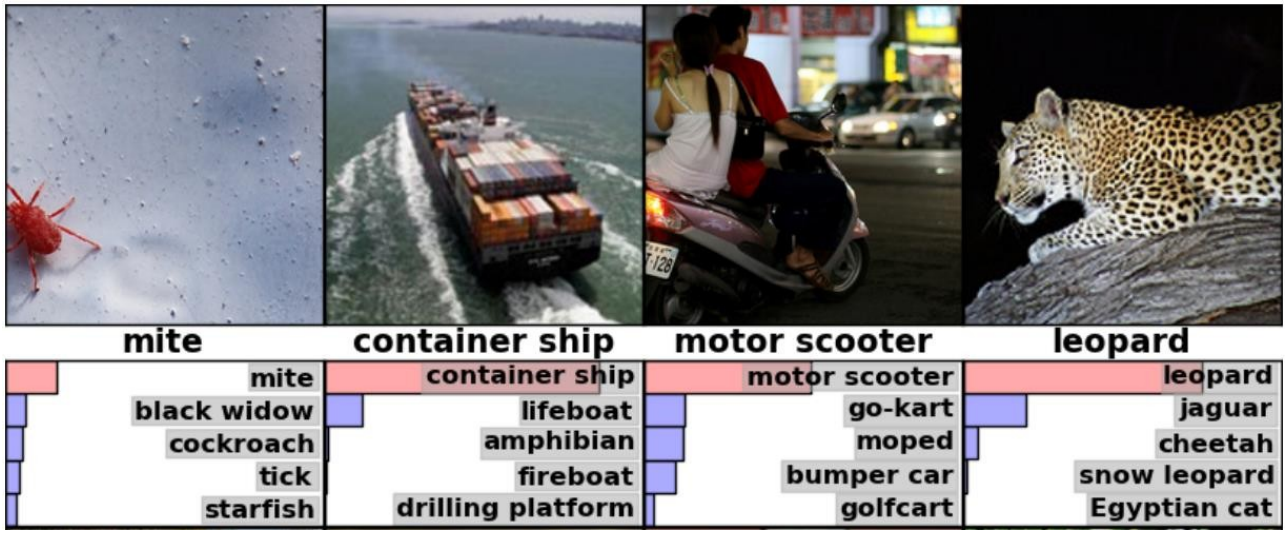
- Different image classification domains
- Image captioning with recurrent neural networks
- Image object localization with bounding box
- Image segmentation with fully convolutional networks
- Image segmentation with deconvolution layers

Slides Stolen L. Fridman:

Object Recognition

Case Study: ImageNet





Slides Stolen L. Fridman:

What is ImageNet?

- **ImageNet:** dataset of 14+ million images (21,841 categories)
  - Links to images not images
- Let's take the high level category of **fruit** as an example:
  - Total 188,000 images of fruit
  - There are 1206 Granny Smith apples:

What is ImageNet?

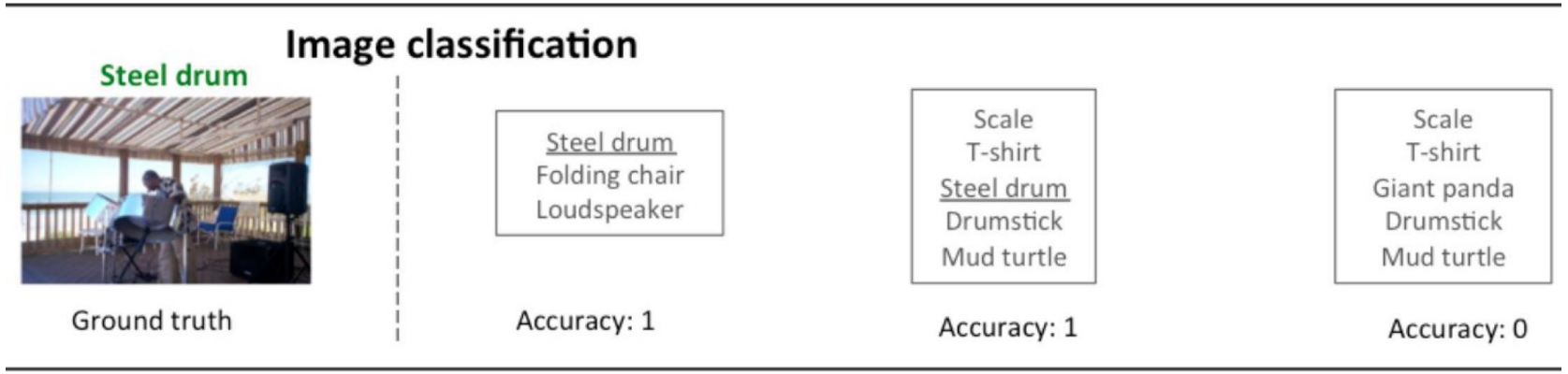
Dataset →
- **ImageNet:** dataset of 14+ million images

Competition →
- **ILSVRC:** ImageNet Large Scale Visual Recognition Challenge

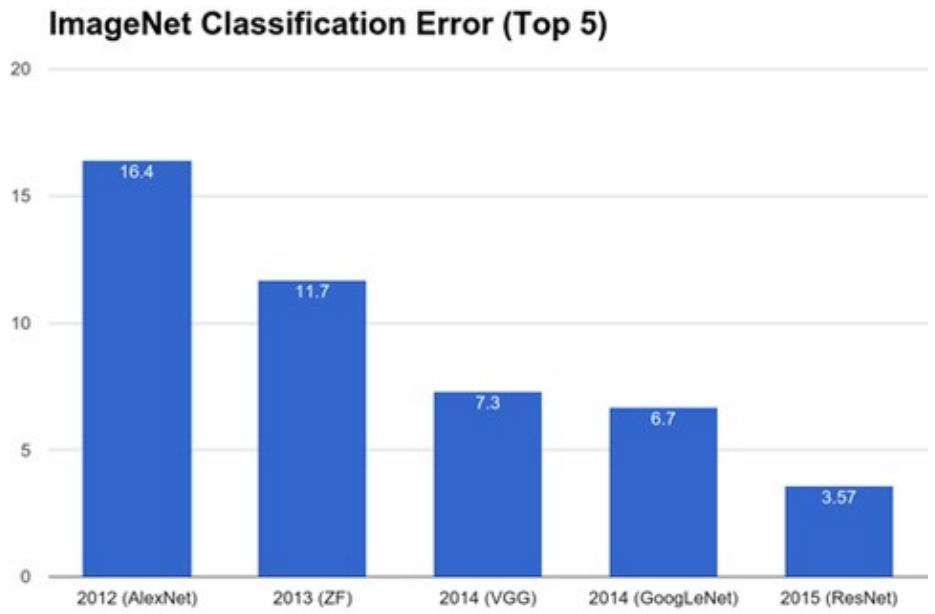
Networks →
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2015)
- CUImage (2016)

ILSVRC Challenge Evaluation for Classification

- Top 5 error rate:
  - You get 5 guesses to get the correct label

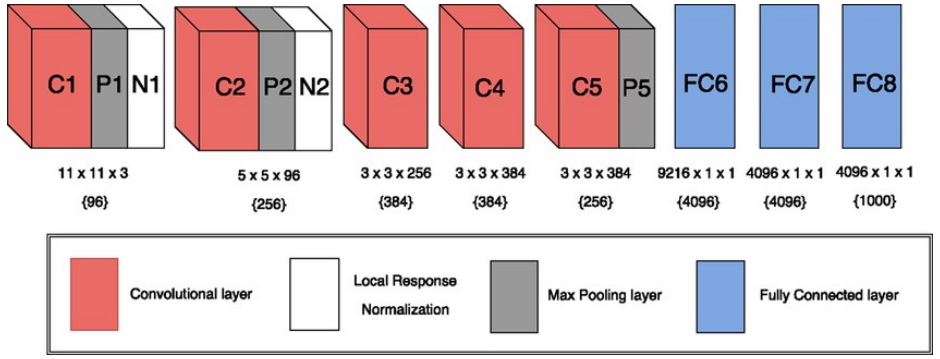


- ~20% reduction in accuracy for Top 1 vs Top 5
  - Example: In 2012 AlexNet achieved
- Human annotation is a binary task: “apple” or “not apple”



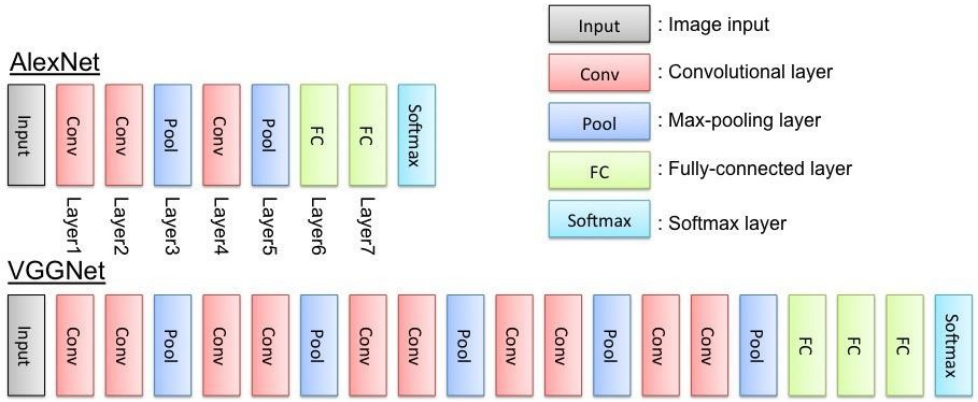
- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

Slides Stolen L. Fridman:



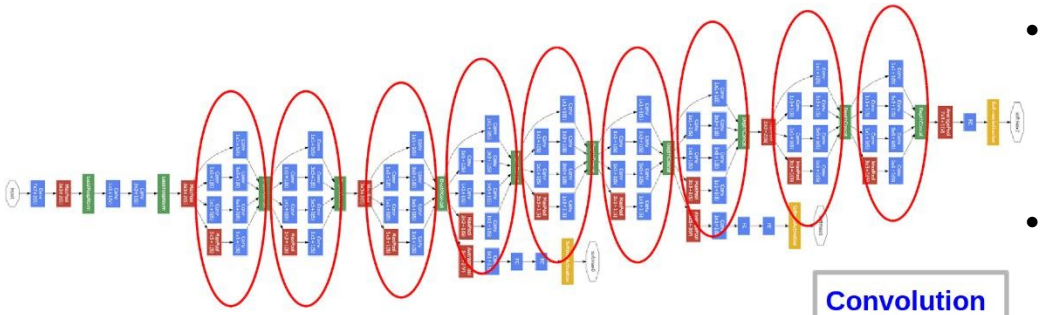
Krizhevsky et al. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

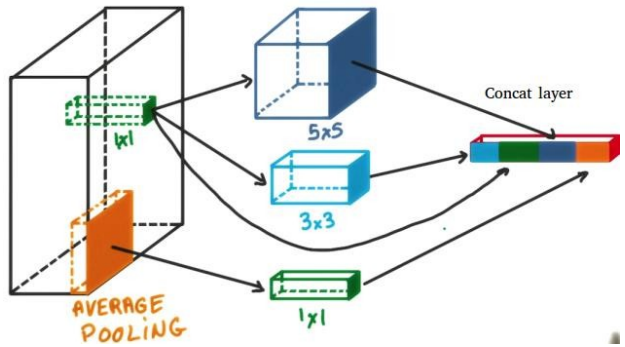
- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

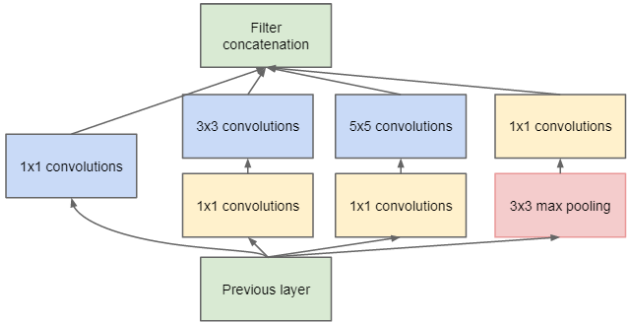


Simonyan et al. "Very deep convolutional networks for large-scale image recognition." 2014.

- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

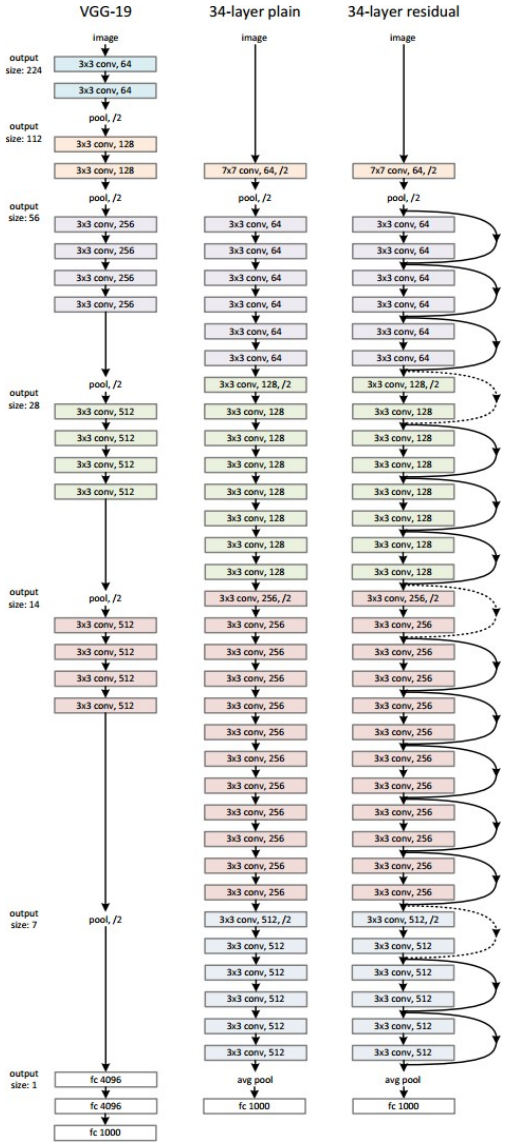


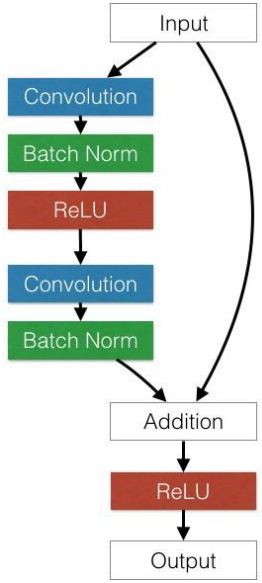




Szegedy et al. "Going deeper with convolutions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

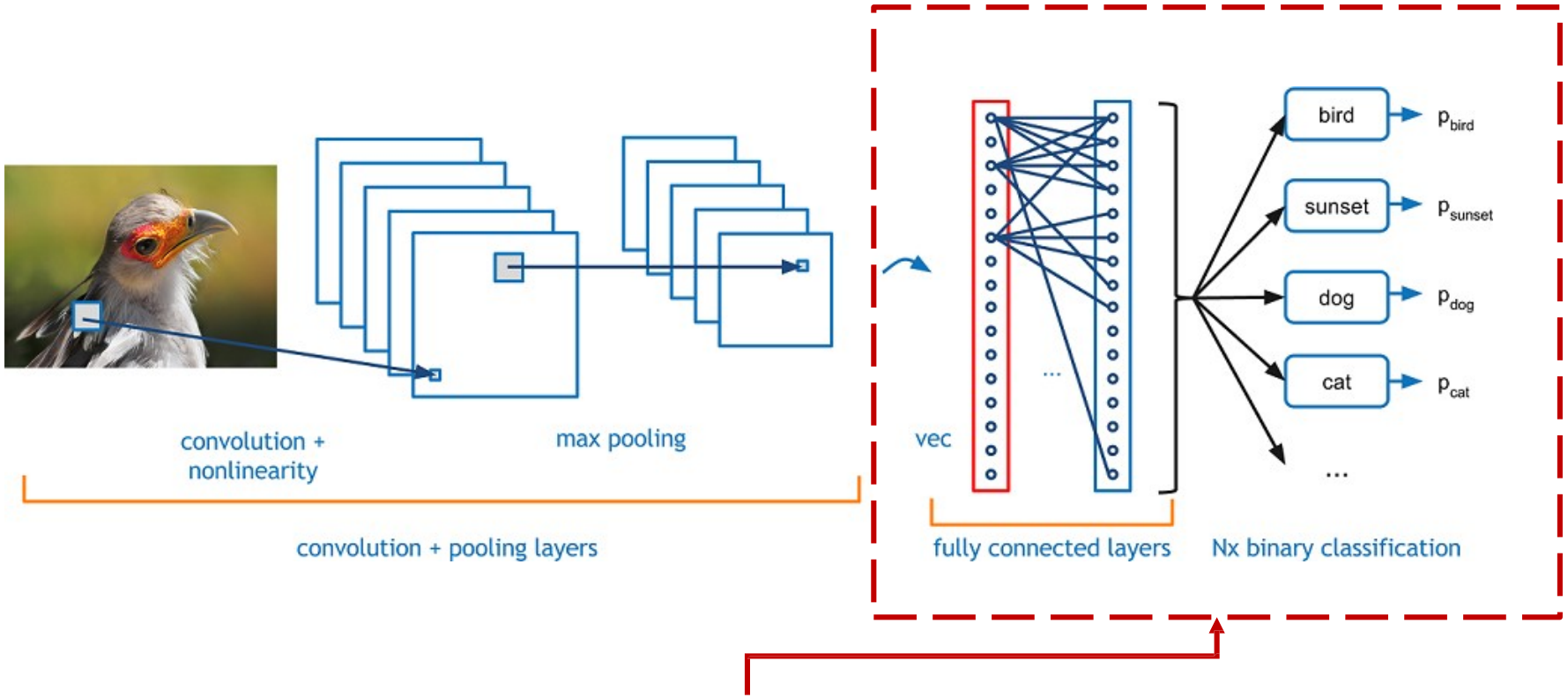




He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

Same Architecture, Many Applications

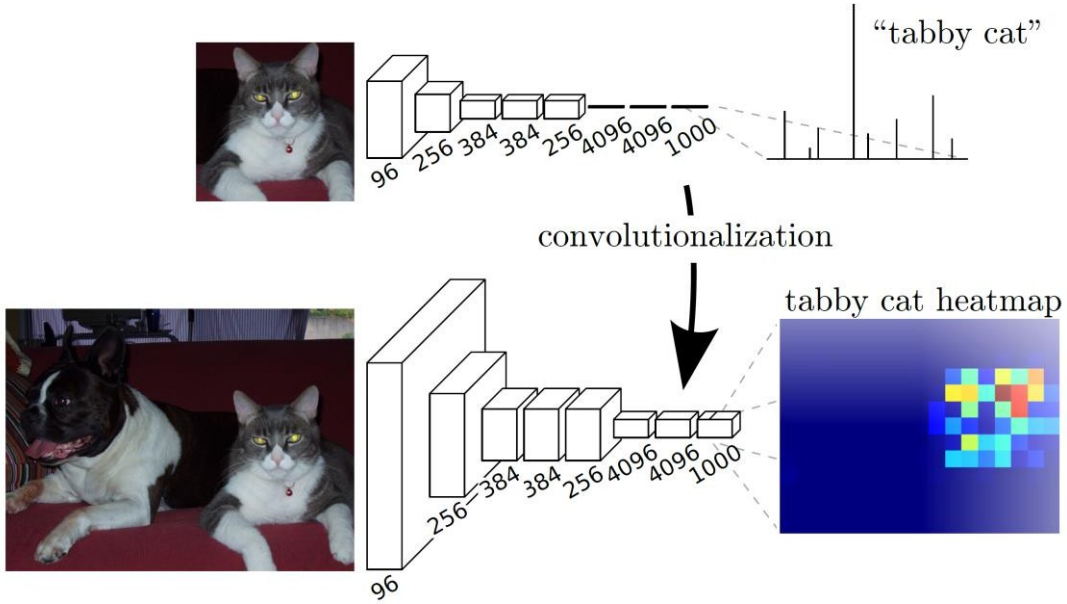


This part might look different for:

- Different image classification domains
- Image captioning with recurrent neural networks
- Image object localization with bounding box
- Image segmentation with fully convolutional networks
- Image segmentation with deconvolution layers

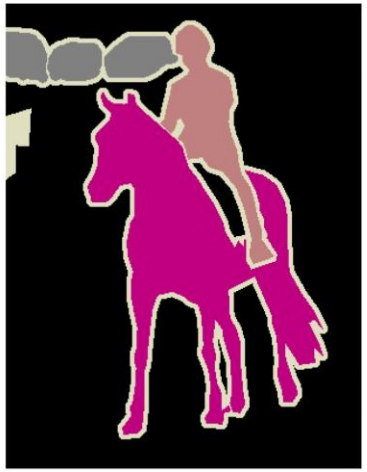
Slides Stolen L. Fridman:

Segmentation

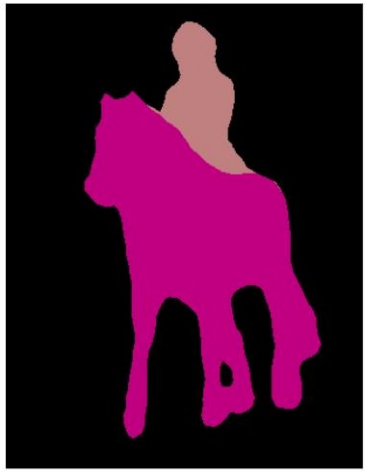


Original

Ground Truth



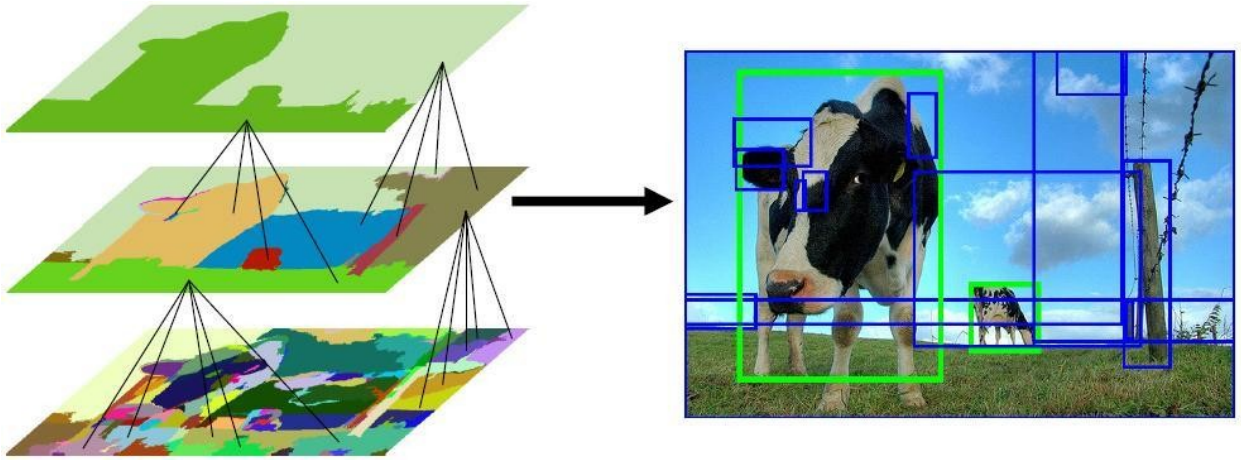
FCN-8



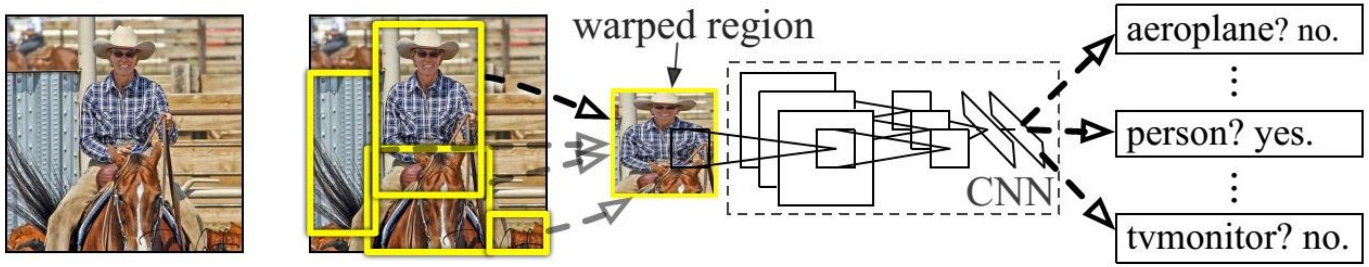
Slides Stolen L. Fridman:


Sky
Building
Pole
Road Marking
Road
Pavement
Tree
Sign Symbol
Fence
Vehicle
Pedestrian
Bike

Object Detection



R-CNN: *Regions with CNN features*

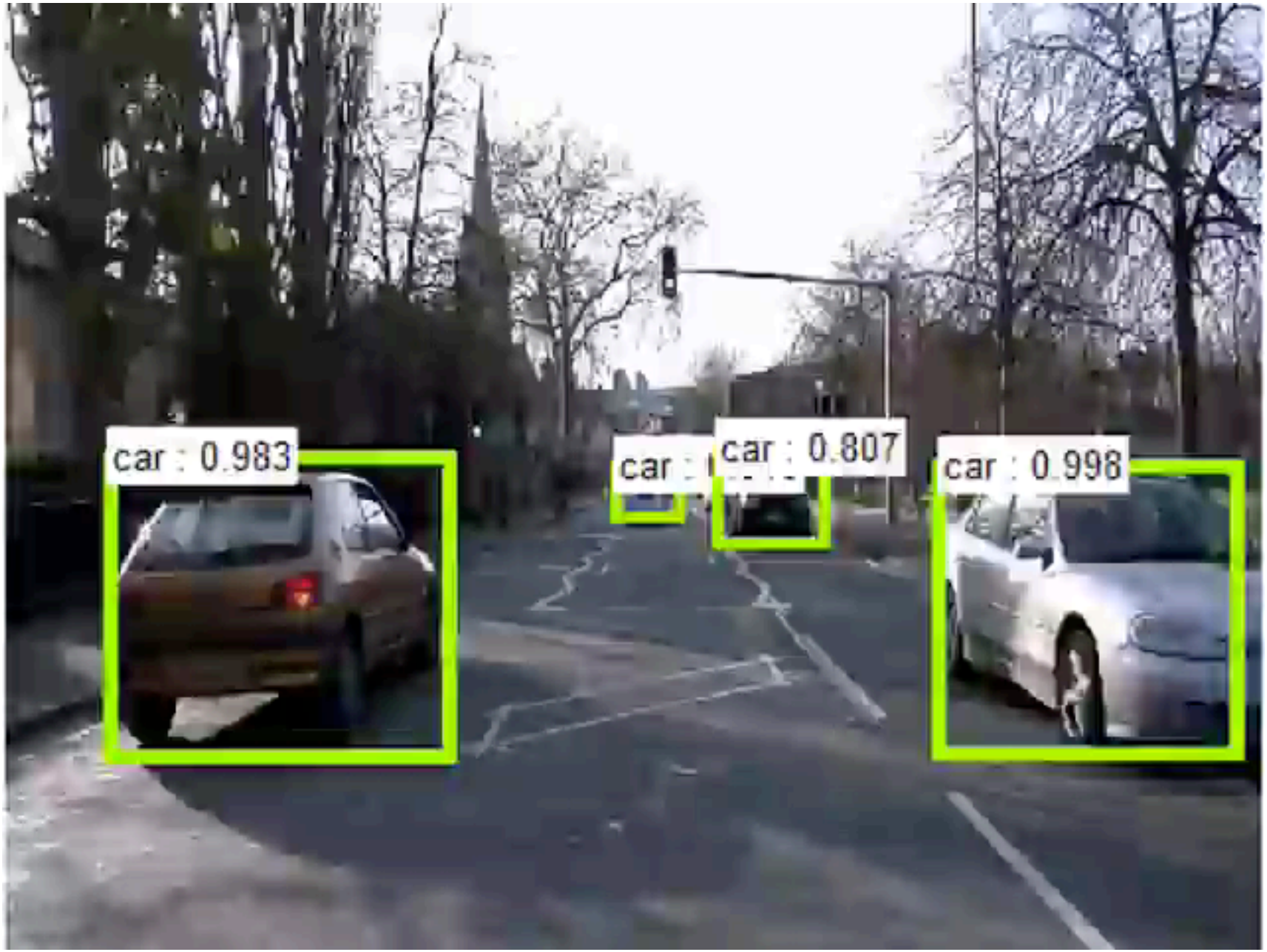


1. Input image

2. Extract region proposals (~2k)

3. Compute CNN features

4. Classify regions


car : 0.983
car : 0.807
car : 0.998

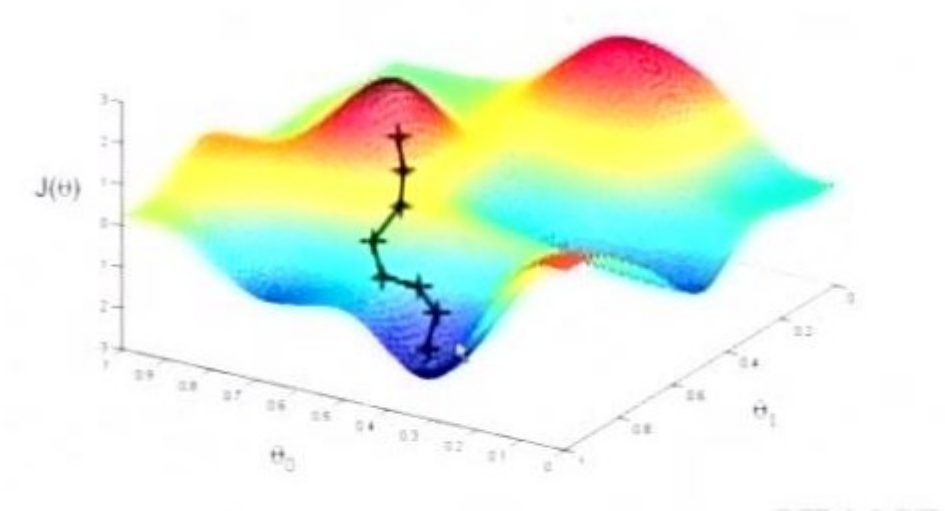
Training Data

Testing Data

Minibatches Reduce Gradient Variance

- Initialize θ randomly
- For N Epochs
  - For each training **batch {(x0, y0), … , (x_B, y_B)}:**
    - Compute Loss Gradient: $$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B}\sum_{i}^{B}\frac{\partial J_i(\theta)}{\partial \theta}$$
    - Update θ with update rule: $$\theta := \theta - \eta\frac{\partial J(\theta)}{\partial \theta}$$

More accurate estimate!



(Stolen from N. Locascio)

Advantages of Minibatches

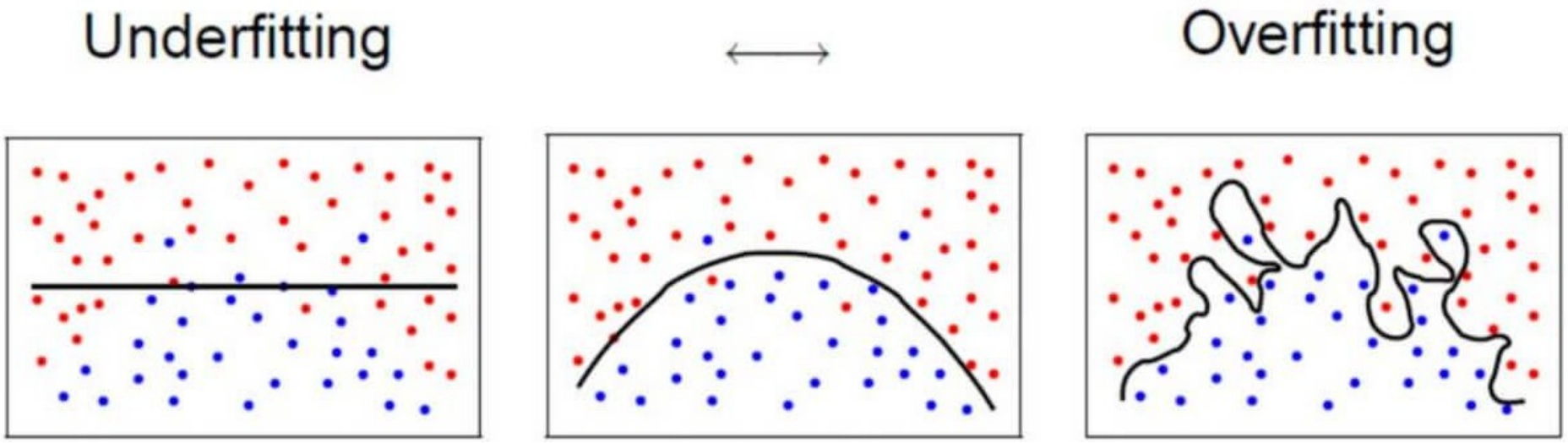
- More accurate estimation of gradient
  - Smoother convergence
  - Allows for larger learningrates
- Minibatches lead to fasttraining!
  - Can parallelize computation + achieve significant speed increases on GPU's

(Stolen from N. Locascio)

Training Neural Networks In Practice: Fighting Overfitting

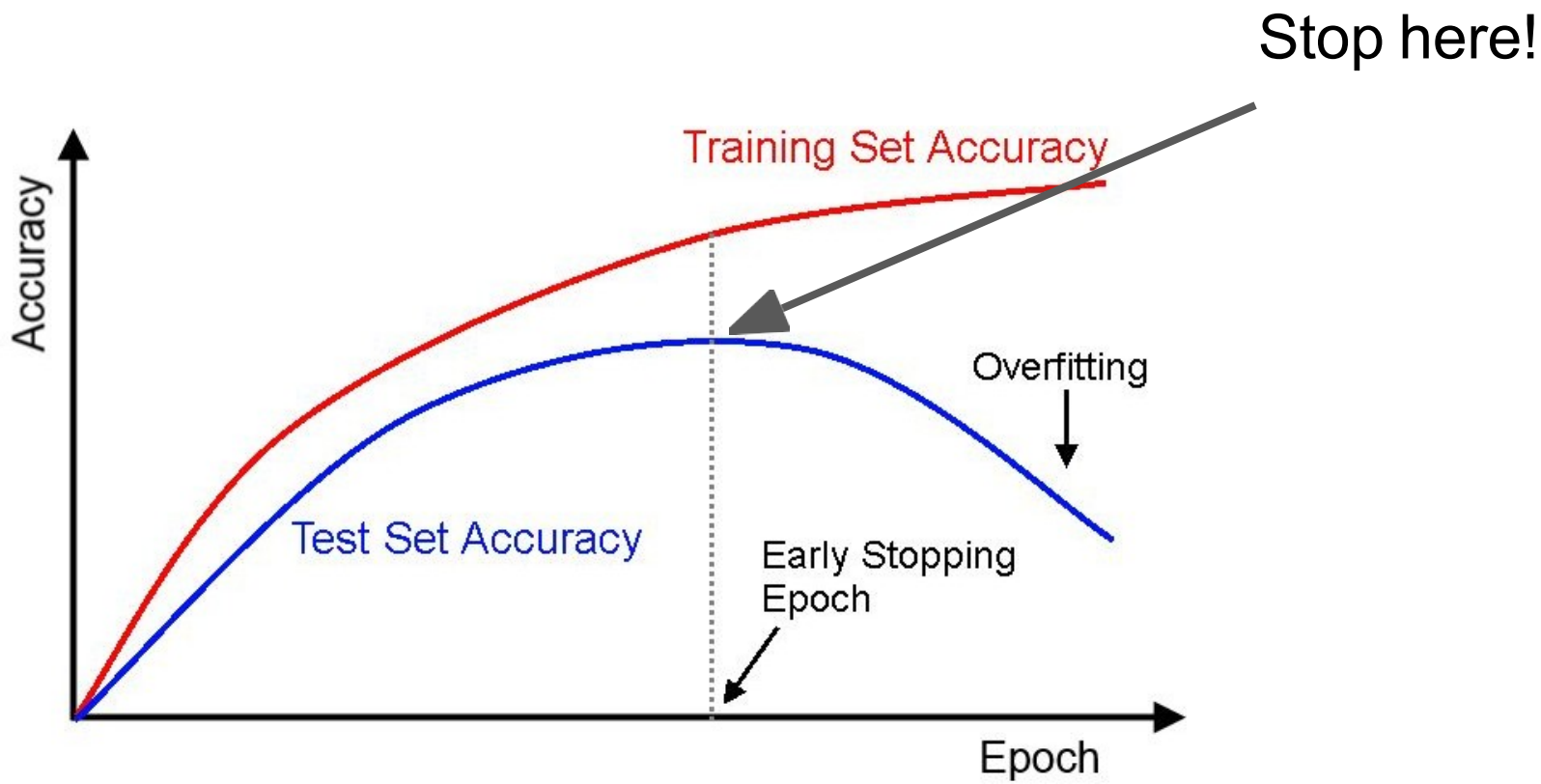
(Stolen from N. Locascio)

The Problem of Overfitting



(Stolen from N. Locascio)

Regularization II: Early Stopping



(Stolen from N. Locascio)

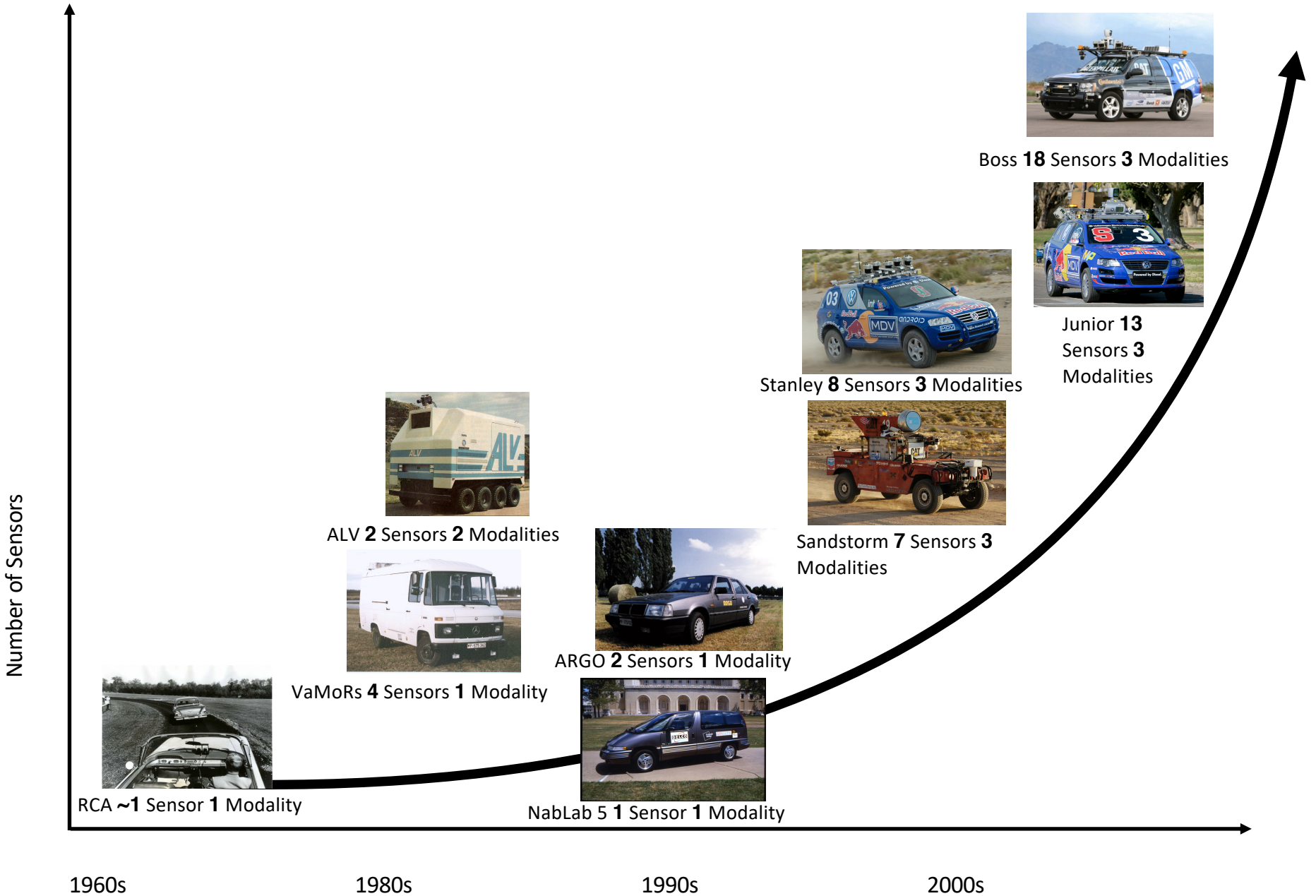
End-To-End

DAVE 2 Driving a Lincoln

- A convolutional neural network
- Trained by human drivers
- Learns perception, path planning, and control
  "pixel in, action out"
- Front-facing camera is the only sensor

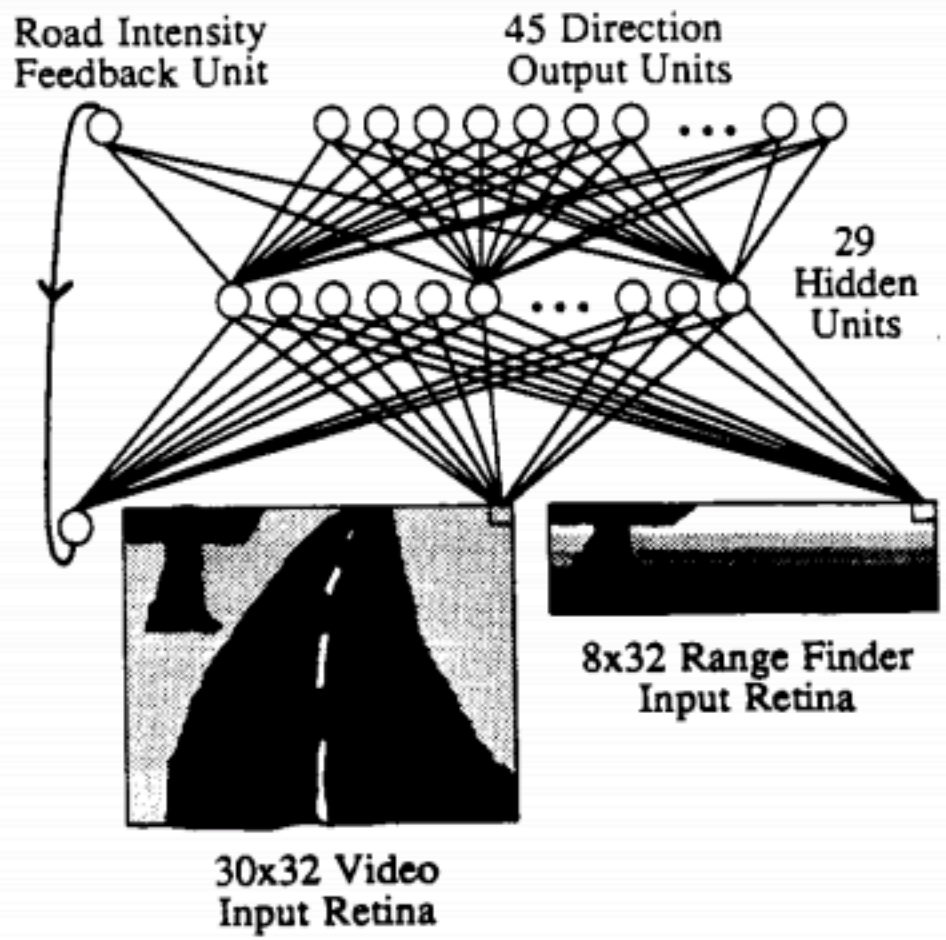
DAVE-1


Tyvek
Tyvek


Number of Sensors
1960s
1980s
1990s
2000s
RCA ~1 Sensor 1 Modality
ALV 2 Sensors 2 Modalities
VaMoRs 4 Sensors 1 Modality
ARGO 2 Sensors 1 Modality
NabLab 5 1 Sensor 1 Modality
Sandstorm 7 Sensors 3 Modalities
Stanley 8 Sensors 3 Modalities
Junior 13 Sensors 3 Modalities
Boss 18 Sensors 3 Modalities

1989

ALVINN Architecture

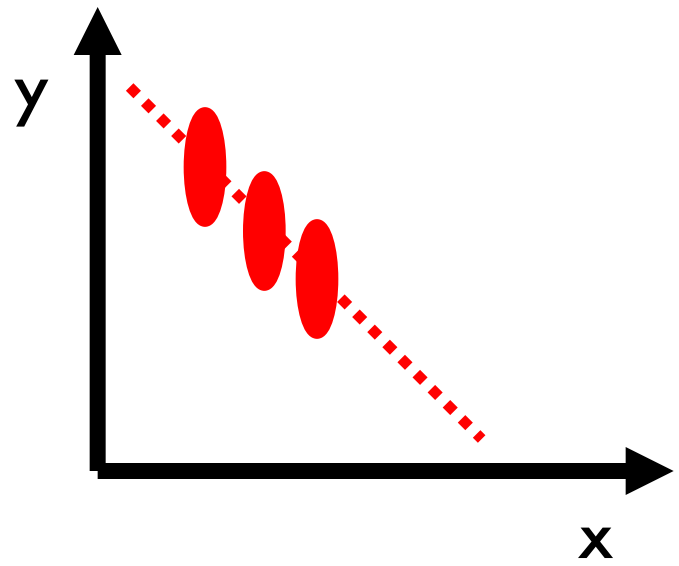


[Courtesy of Dean Pomerleau]

Hough transform

P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

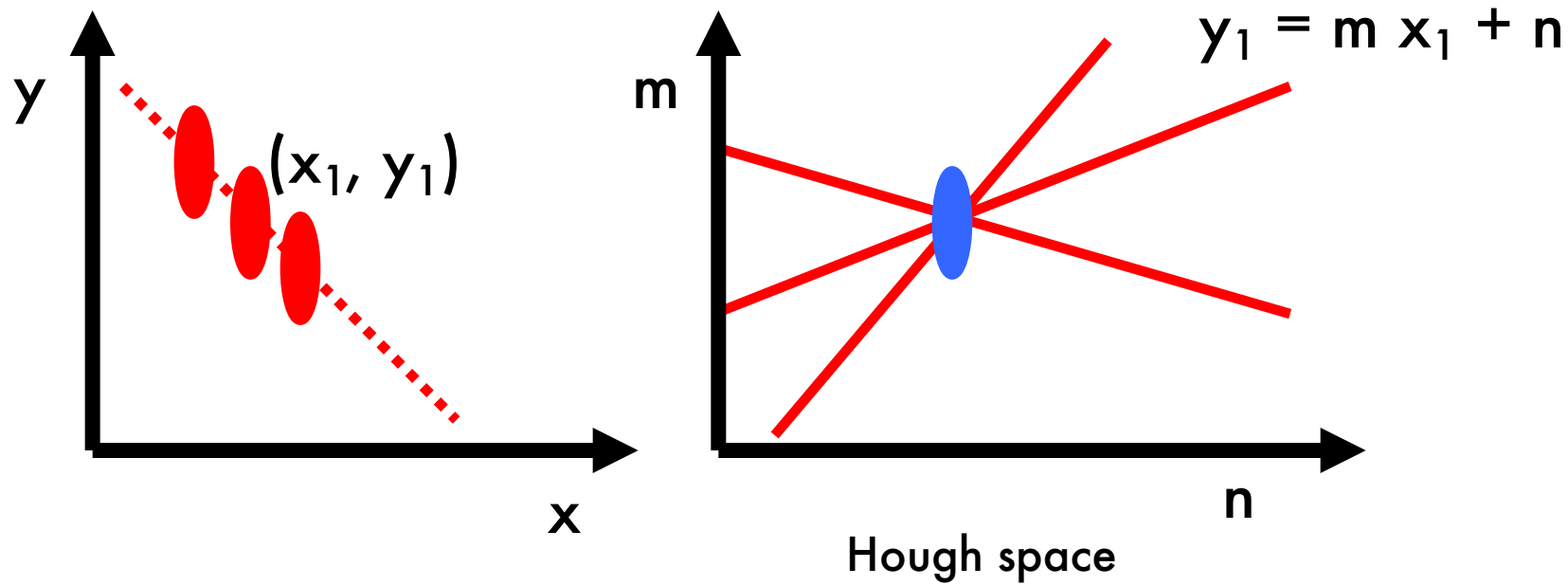
Given a set of points, find the curve or line that explains the data points best



Hough transform

P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

Given a set of points, find the curve or line that explains the data points best



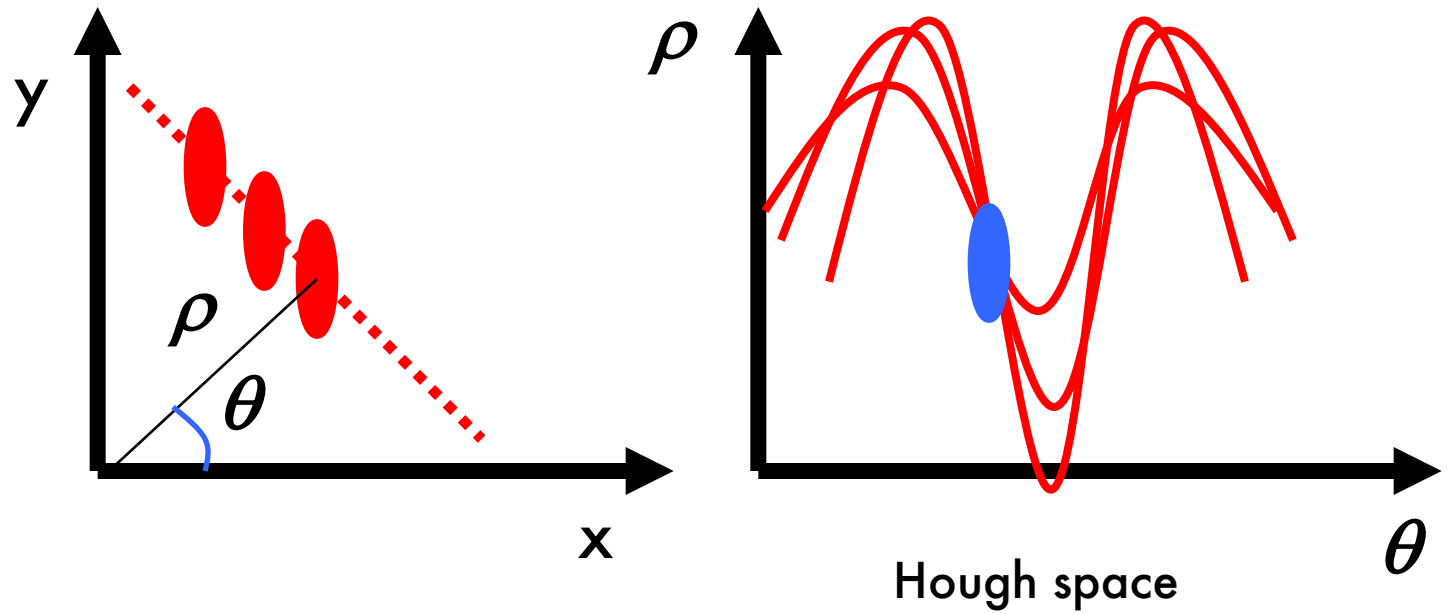
$$y = m\ x + n$$

Hough transform

P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

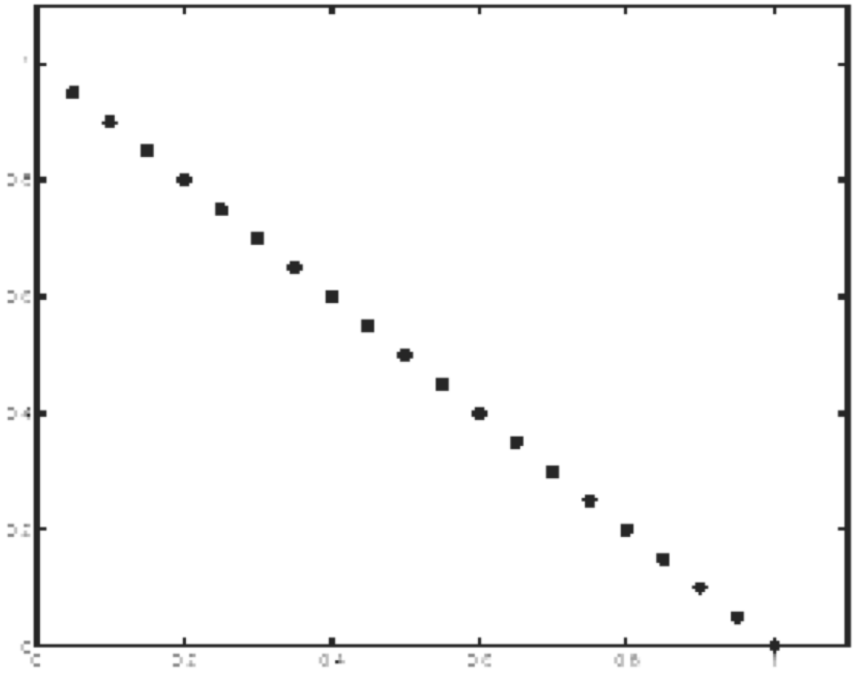
Issue : parameter space [m,n] is unbounded...

- Use a polar representation for the parameter space

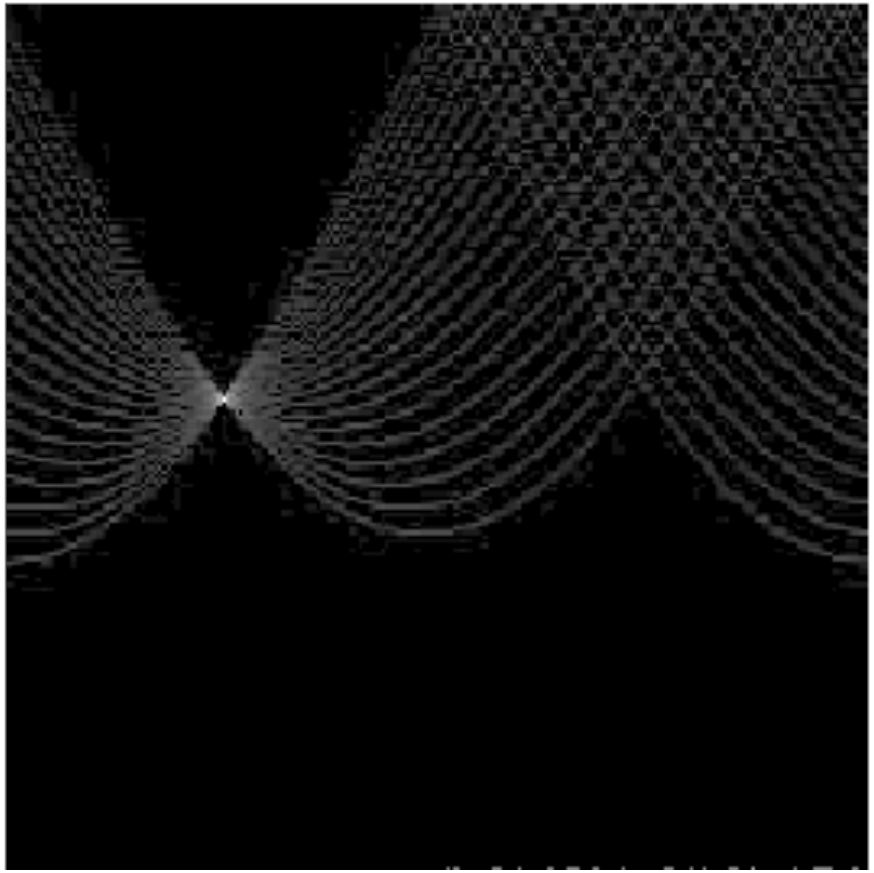


$$x\cos\theta + y\sin\theta = \rho$$

Hough transform - experiments



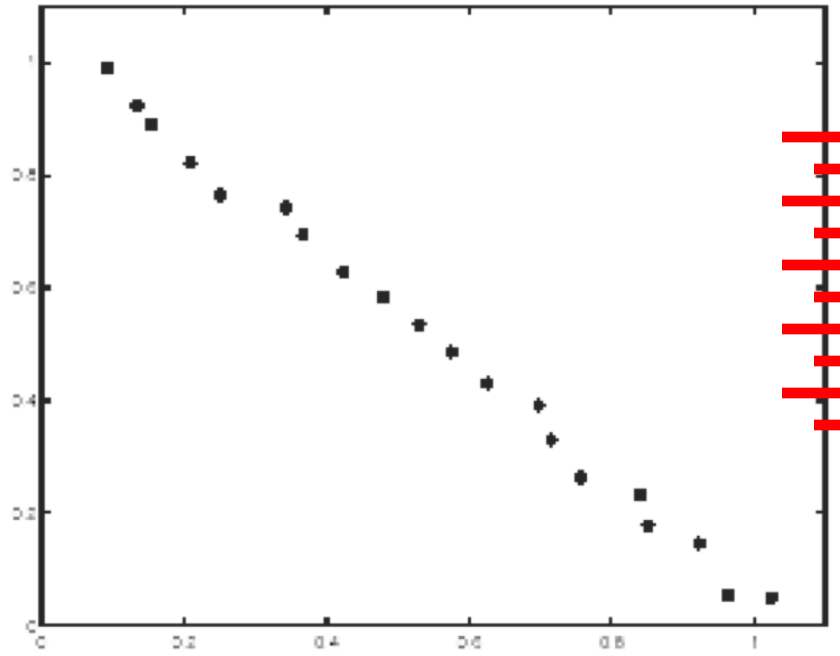
features



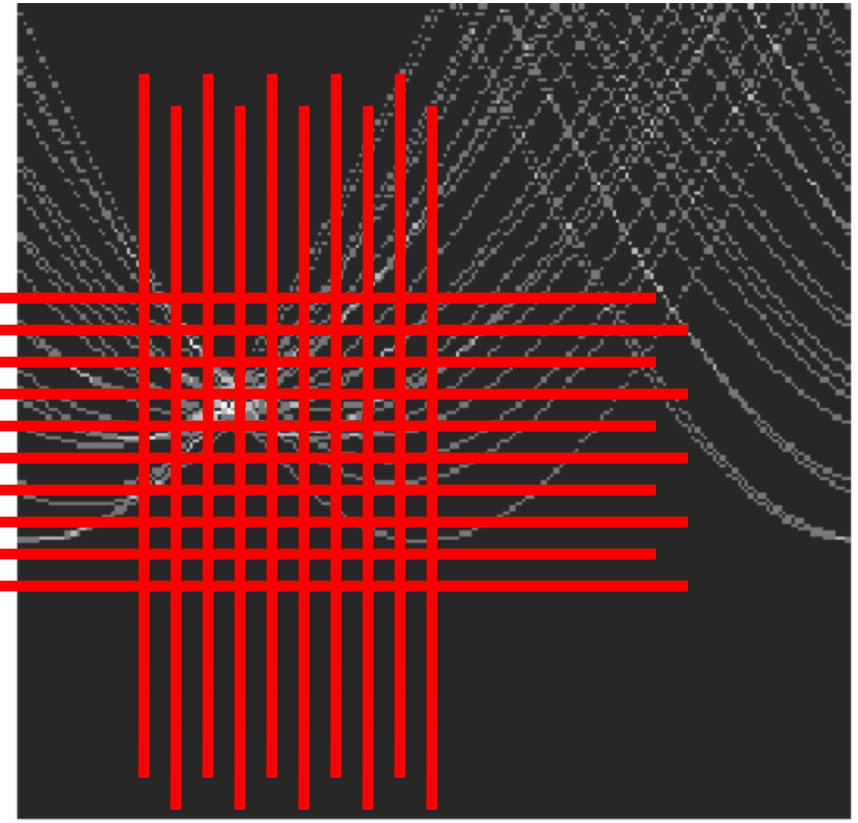
votes

Hough transform - experiments

Noisy data



features



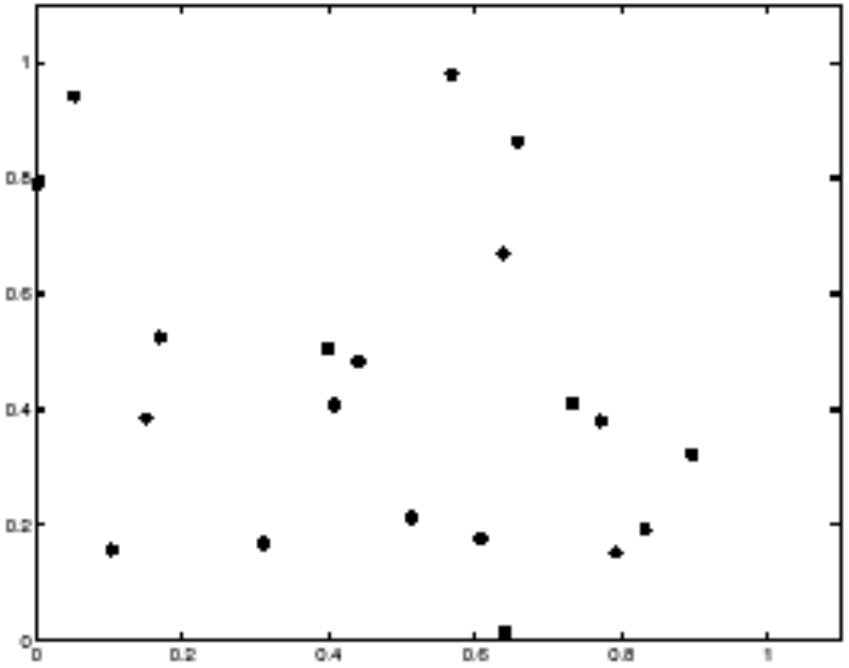
votes

How to compute the intersection point?

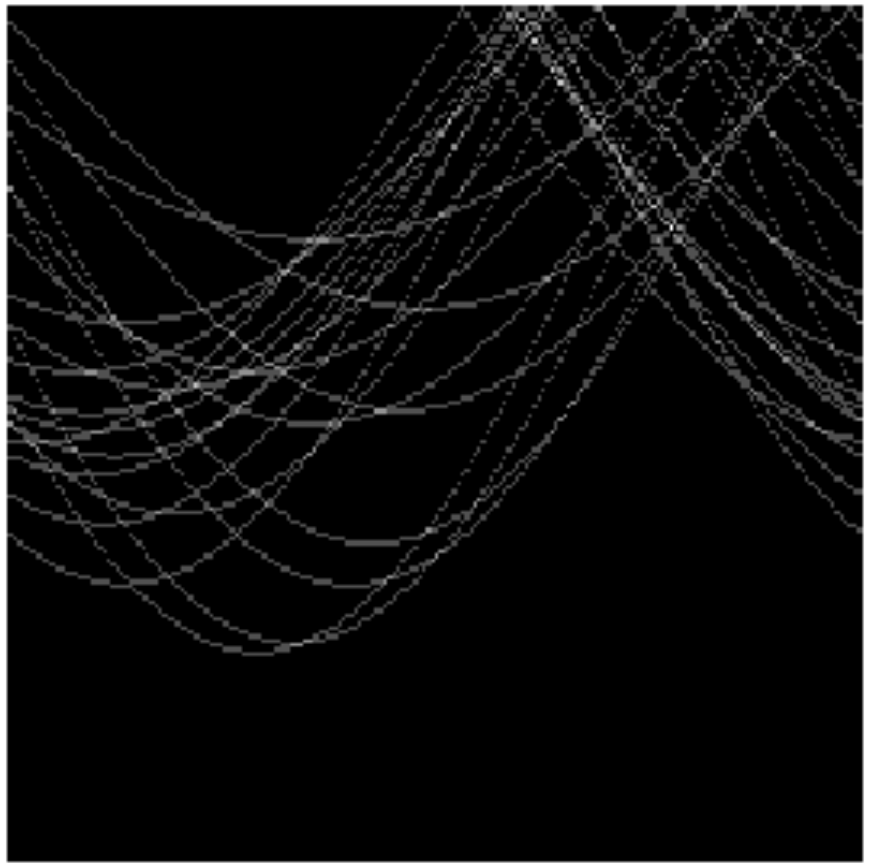
IDEA: introduce a grid a count intersection points in each cell

Issue: Grid size needs to be adjusted...

Hough transform - experiments



features



votes

Issue: spurious peaks due to uniform noise

Hough transform - conclusions

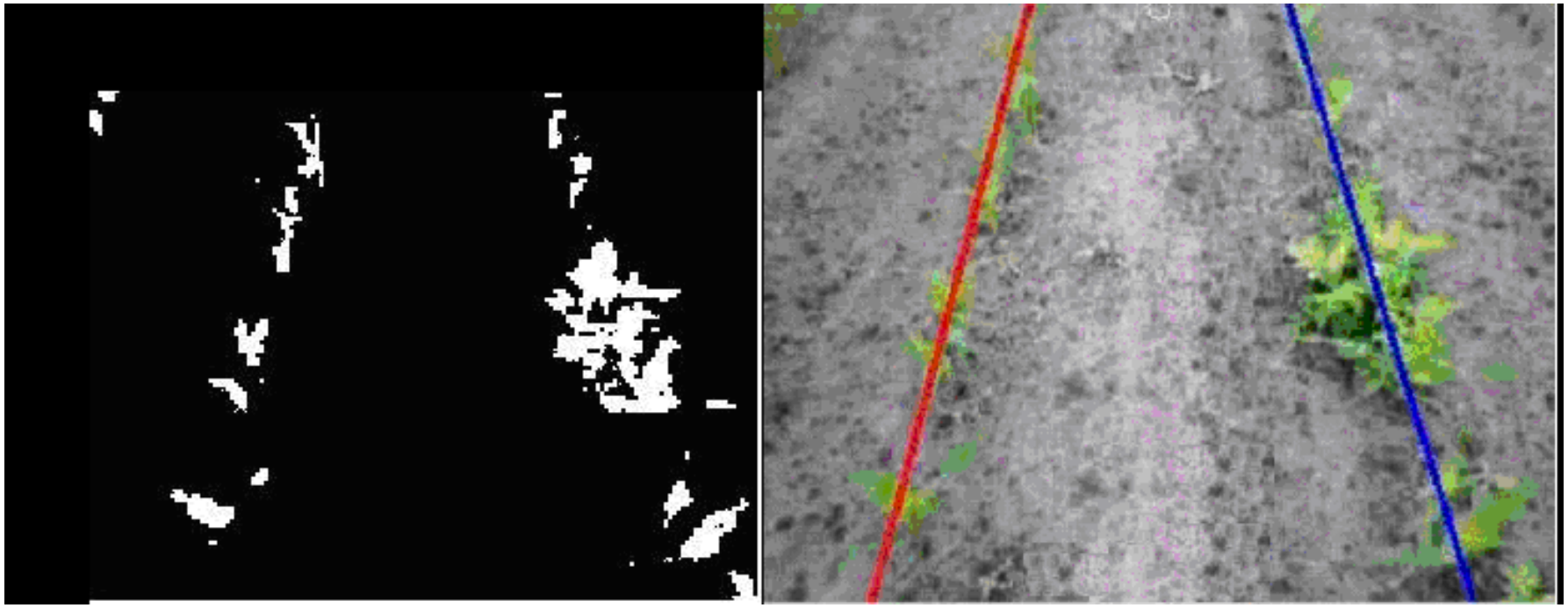
Good:

- All points are processed independently, so can cope with occlusion/outliers
- Some robustness to noise: noise points unlikely to contribute consistently to any single bin

Bad:

- Spurious peaks due to uniform noise
- Trade-off noise-grid size (hard to find sweet point)

Hough transform - experiments



Courtesy of TKK Automation Technology Laboratory

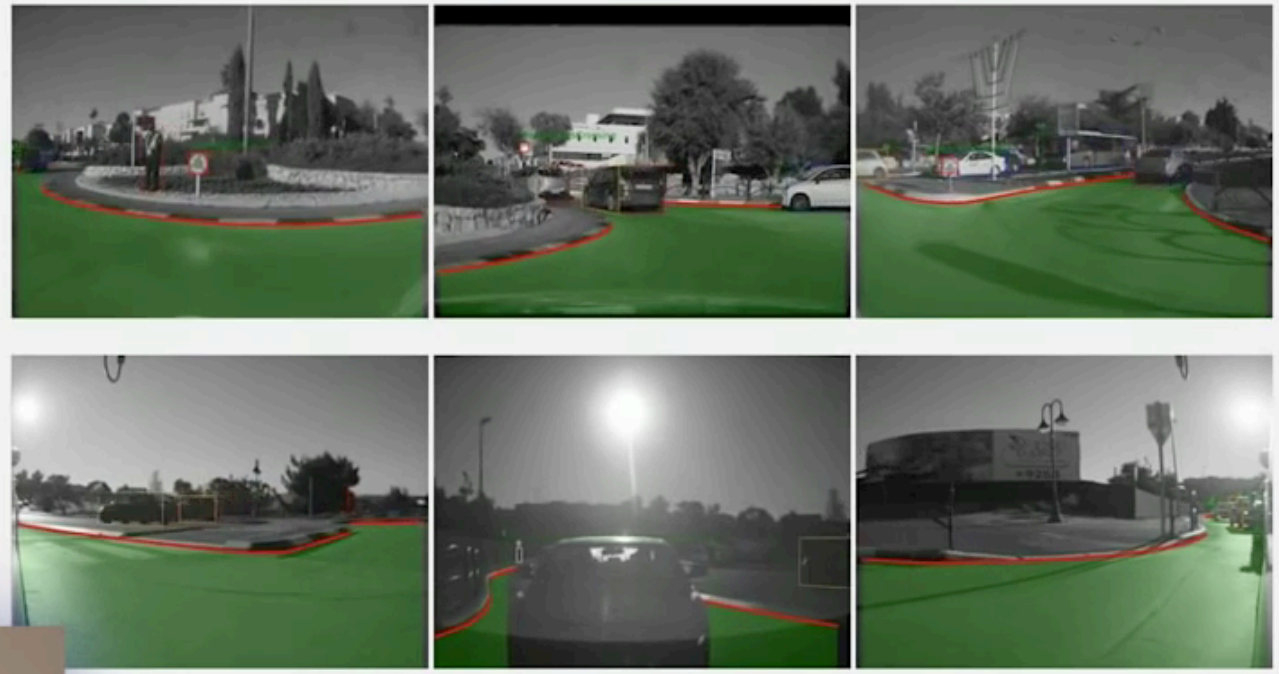
NavLab 97

Componet


Sky
Building
Pole
Road Marking
Road
Pavement
Tree
Sign Symbol
Fence
Vehicle
Pedestrian
Bike

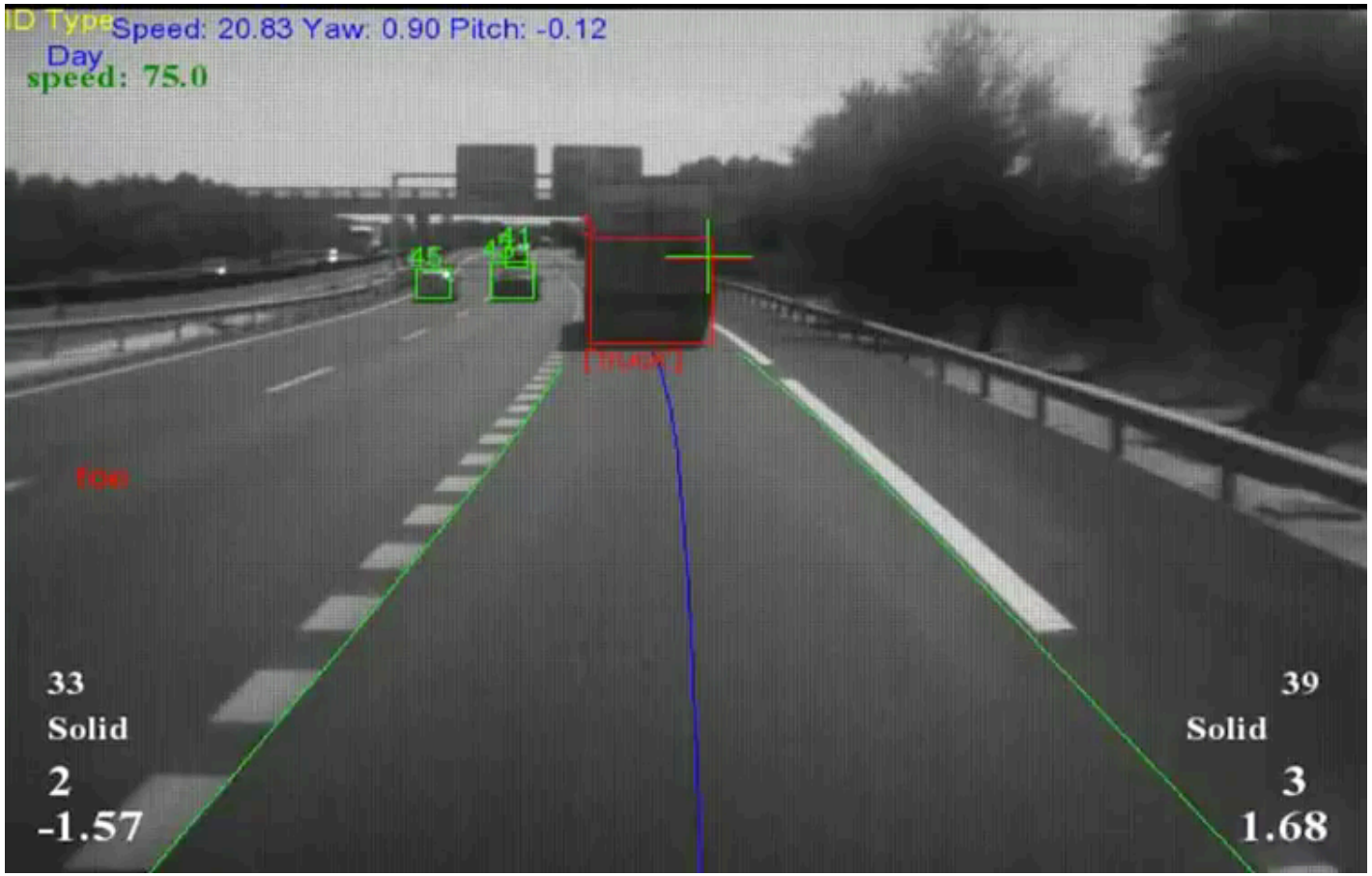

car 0.333
car 0.895
car 0.937
car 0.324
car 0.312
car 0.303
car 0.893
car 0.336

Free Space through Pixel Labeling



Our Vision. Your Safety ™

MOBILEYE®
Our Vision. Your Safety.™


ID Type
Speed: 20.83 Yaw: 0.90 Pitch: -0.12
Day
speed: 75.0
45
441
foe
33
Solid
2
-1.57
39
Solid
3
1.68

Hybrid?

Learning Affordance for Direct Perception in Autonomous Driving

Chenyi Chen Ari Seff Alain Kornhauser Jianxiong Xiao

Princeton University

Controversy

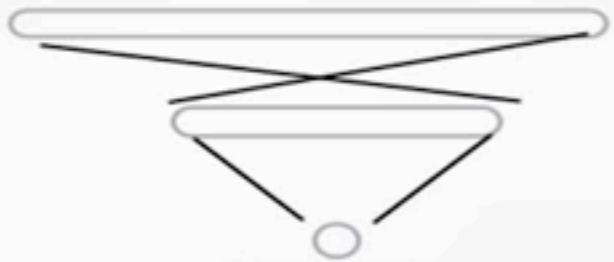
DNN Failure

Example: the n-bit parity problem

$x \in \{0,1\}^n$ input nodes

$\frac{n}{2} + 1$ hidden nodes

$y \in \{0,1\}$ output node



} weights are hard-wired

} weights found by system of linear equations

Setiono, 1997

- Even though there exists weights that solve the n-bit parity problems, "learning" them using the available training techniques does not work for n> 30.
- This failure to train a DNN holds true also for over-subscribed architectures.

DNN Failure

Example: Learning Arithmetic Operations

Hoshen & Peleg, 2015

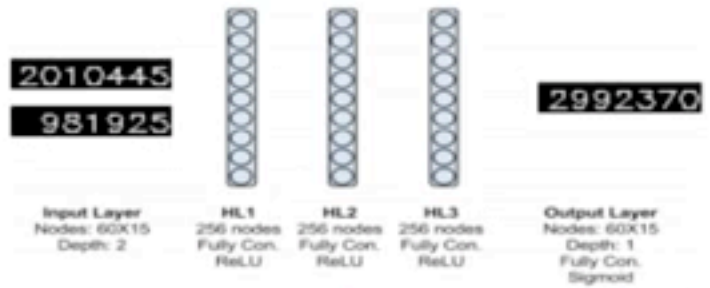


Figure 2. A diagram showing the construction of a neural network with 3 hidden layers able to preform addition using visual data. Two pictures are used as input and one picture as output. The network is fully connected and uses ReLU units in the hidden layers and sigmoid in the output layer. The hidden layers have 256 units each.

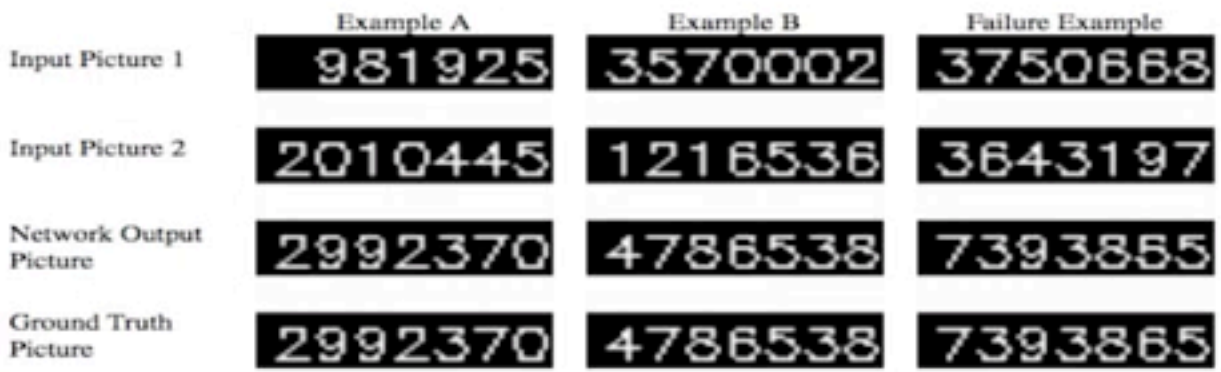


Figure 1. Input and output examples from our neural network trained for addition. The first two examples show a typical correc The last example shows a rare failure case.

- DNN failed on the task of multiplication - whatever architecture they used they were unsuccessful in training the DNN.

DNN Failure

Example: Pentomino Dataset

Gulcehre & Bengio, 2016

Figure 1: Different classes of Pentomino shapes used in our dataset.

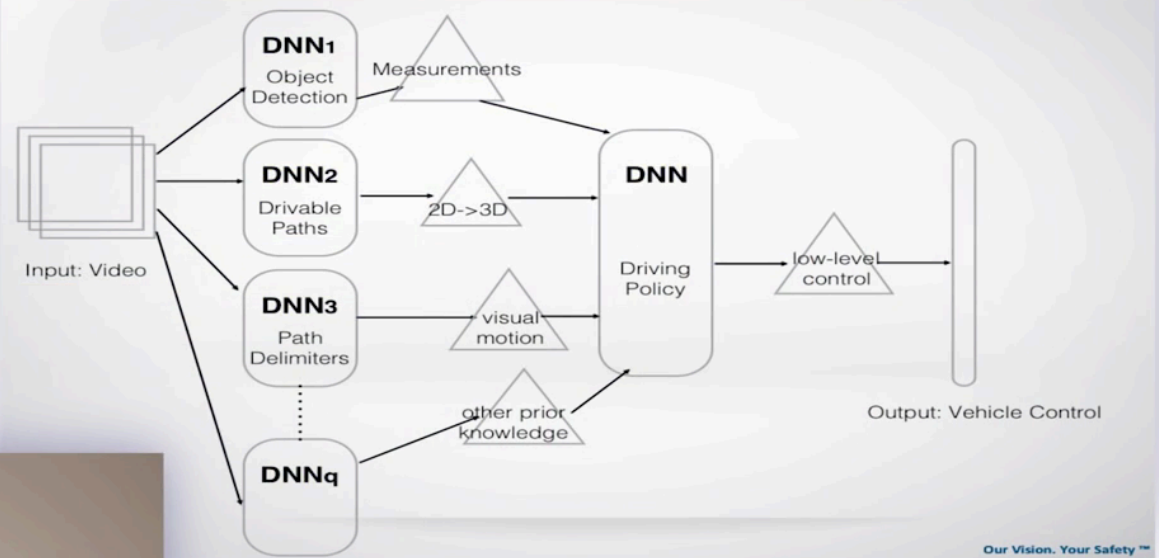
(a) sprites, not all same type (b) sprites, all same type

Figure 2: Left: (a): An example image from the dataset which has *a different sprite type* in it. Right (b): An example image from the dataset that has only one type of Pentomino object in it, but with different orientations and scales.

- Different part types and can appear following some 2D geometric transformations
- Task: find out whether all parts are of the same class or not.

- **DNN failed on the task when end-to-end was concerned.**
- **DNN succeeded when the task was broken down into first finding the category of each part and then making a decision whether all categories are the same.**

Semantic Abstraction (including non-learned blocks)



MOBILEYE®
Our Vision. Your Safety.™

What is Wrong with “End-to-End”?

- It is all about “corner cases” - rare events.
  - The probability of rare events decreases exponentially with end-to-end architecture.
- The importance of “Prior Knowledge”
  - DNNs could fail miserably on seemingly simple problems.

MOBILEYE®
Our Vision. Your Safety.™

DNNs for Control?

SE3-Nets: Learning Rigid Body Motion using Deep Neural Networks

Arunkumar Byravan and Dieter Fox
University of Washington, Seattle

Supplementary video
This video has five sections: “Approach” & four sections of “Results”

Deep Mind Human Models