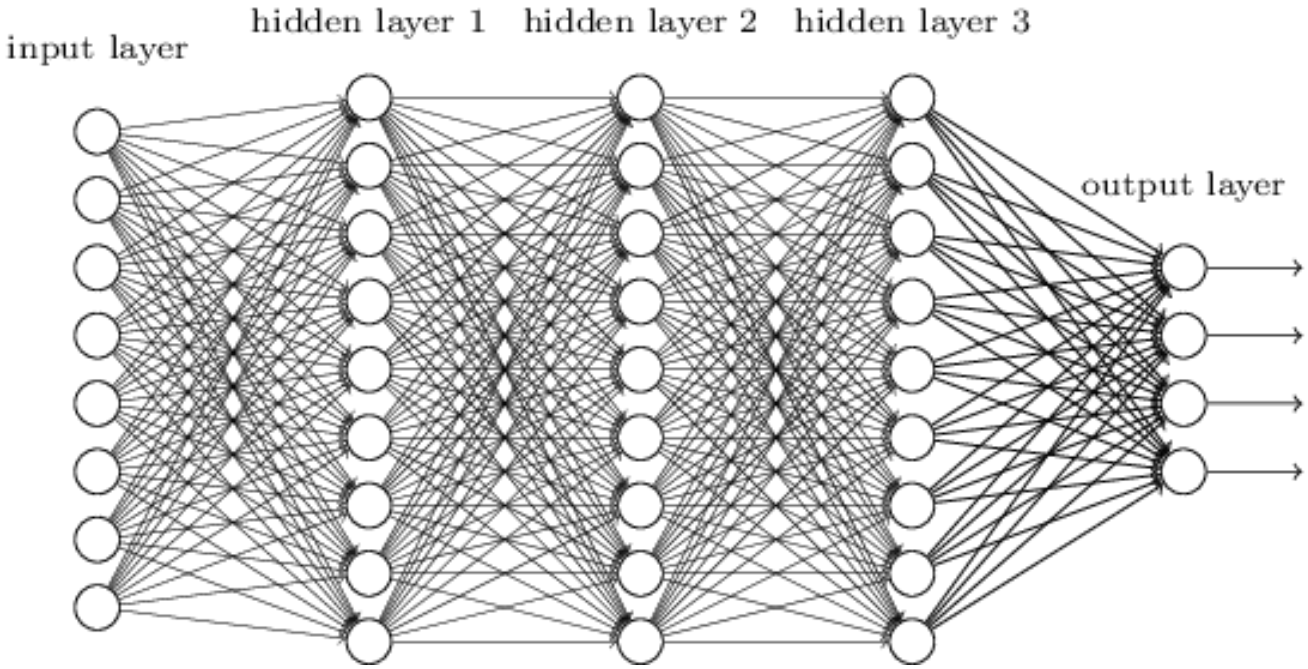
Convolutional Neural Nets

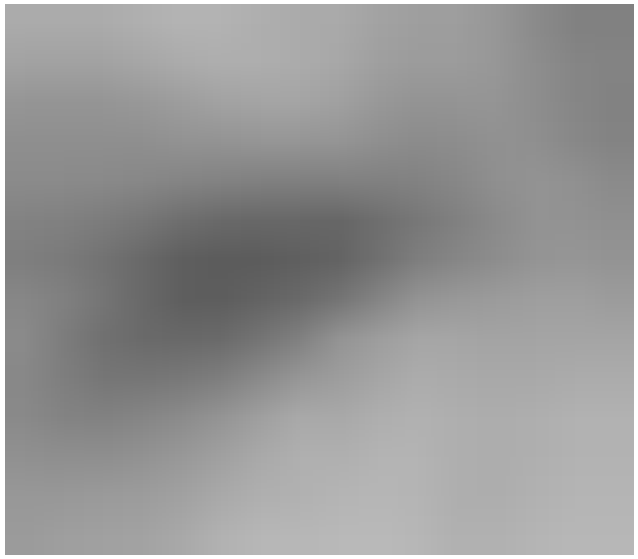
Pascal Fua
IC-CVLab

Reminder: Fully Connected Layers

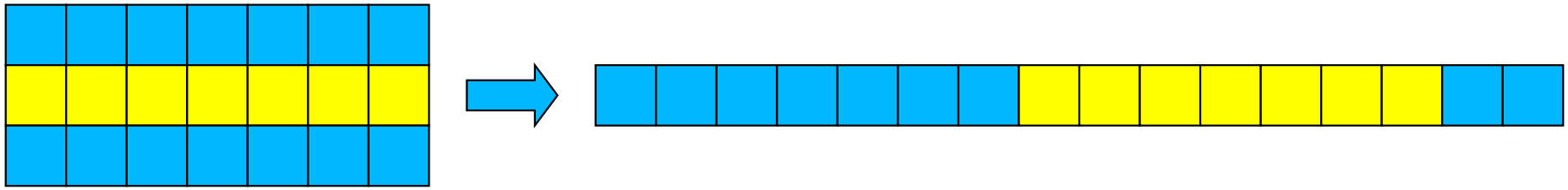


- The descriptive power of the net increases with the number of layers.
- In the case of a 1D signal, it is roughly proportional to $\prod_n W_n$, where W_n represents the width of a layer.
- The number of parameters grows like $\sum_n W_n W_{n+1}$

Processing Digital Images



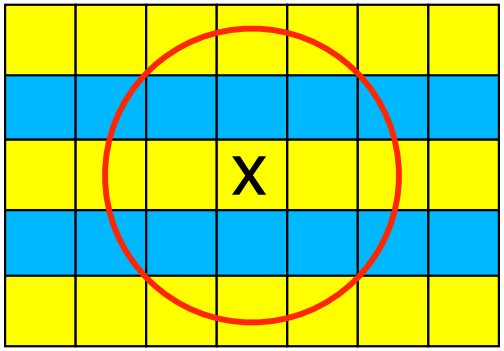
136 134 161 159 163 168 171 173 173 171 166 159 157 155
152 145 136 130 151 149 151 154 158 161 163 163 159 151
145 149 149 145 140 133 145 143 145 145 145 146 148 148
148 143 141 145 145 145 141 136 136 135 135 136 135 133
131 131 129 129 133 136 140 142 142 138 130 128 126 120
115 111 108 106 106 110 120 130 137 142 144 141 129 123
117 109 098 094 094 094 100 110 125 136 141 147 147 145
136 124 116 105 096 096 100 107 116 131 141 147 150 152
152 152 137 124 113 108 105 108 117 129 139 150 157 159
159 157 157 159 135 121 120 120 121 127 136 147 158 163
165 165 163 163 163 166 136 131 135 138 140 145 154 163
166 168 170 168 166 168 170 173 145 143 147 148 152 159
168 173 173 175 173 171 170 173 177 178 151 151 153 156
161 170 176 177 177 179 176 174 174 176 177 179 155 157
161 162 168 176 180 180 180 182 180 175 175 178 180 180



- A MxN image can be represented as an MN vector.
- It can therefore be used an input to an MLP.
- However
  - the neighborhood relationships are then lost,
  - the number of parameters grows very fast.

—> This is not the best approach.

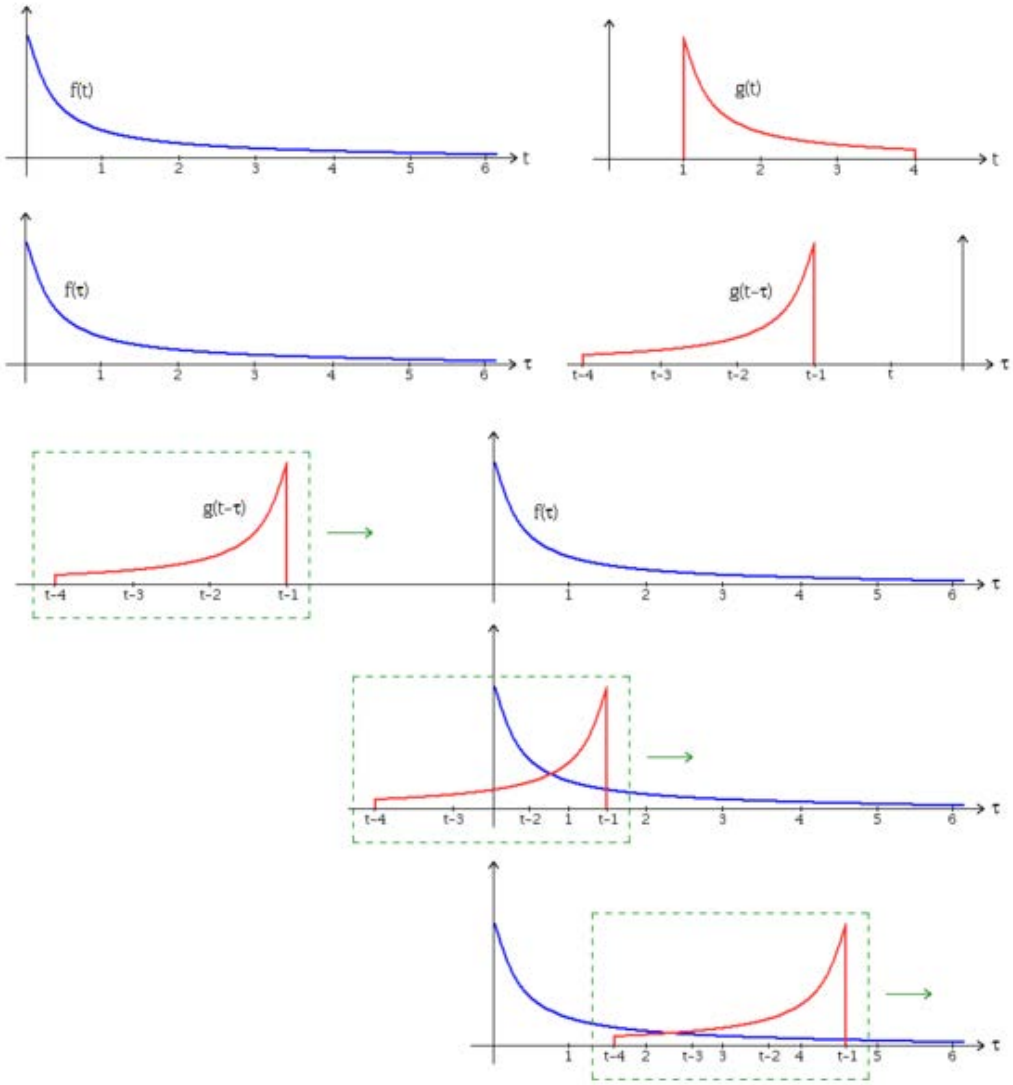
Image Specificities



- In a typical image, the values of neighboring pixels tend to be more highly correlated than those of distant ones.
- An image filter should be translation equivariant.

—> These two properties can be exploited to drastically reduce the number of weights required by CNNs using so-called convolutional layers.

1D Convolution in the Continuous Domain



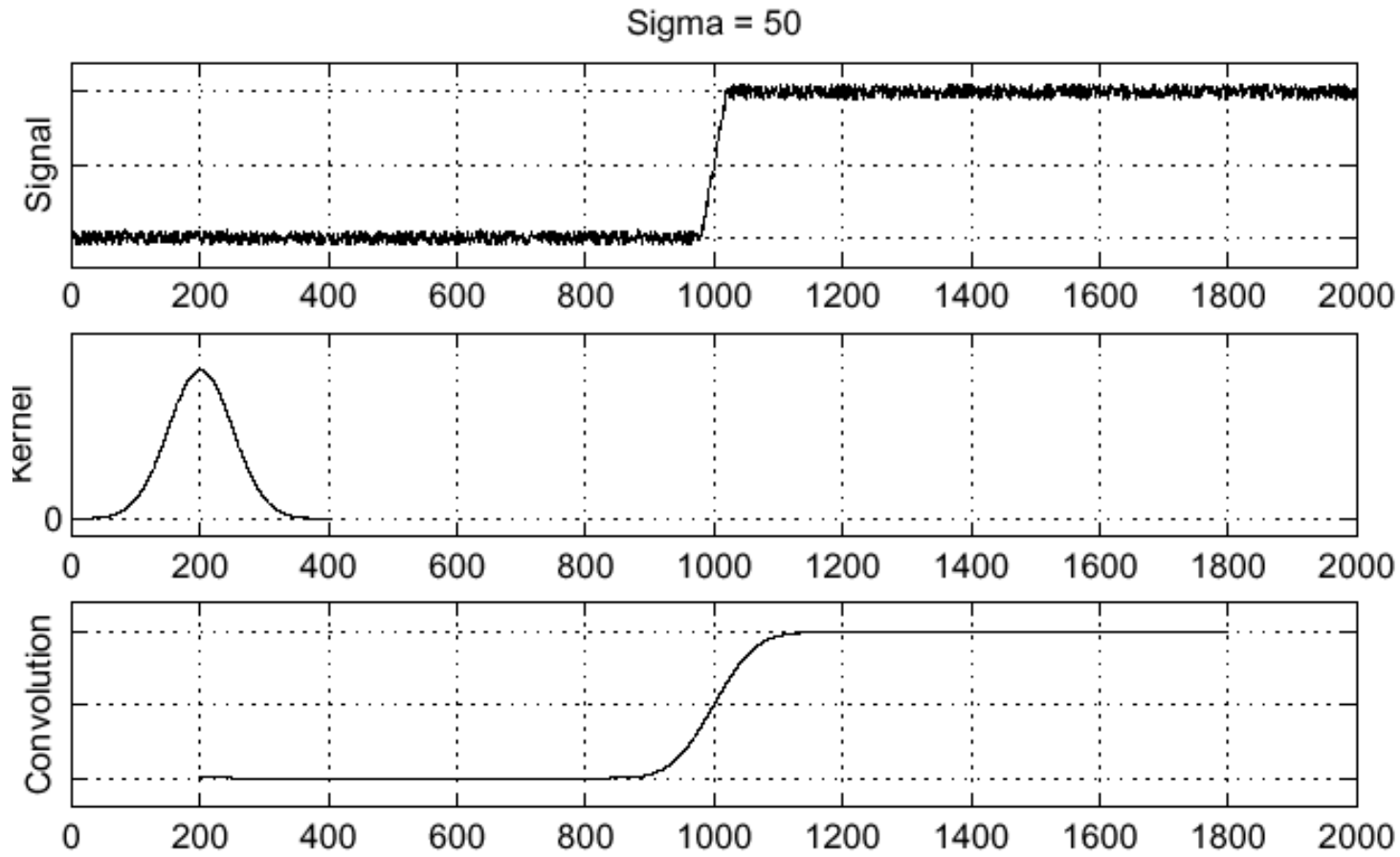
$$g * f(t) = \int_{\tau} g(t-\tau) f(\tau) d\tau$$

Example 1: Convolution with a Gaussian

f

g

$g * f$



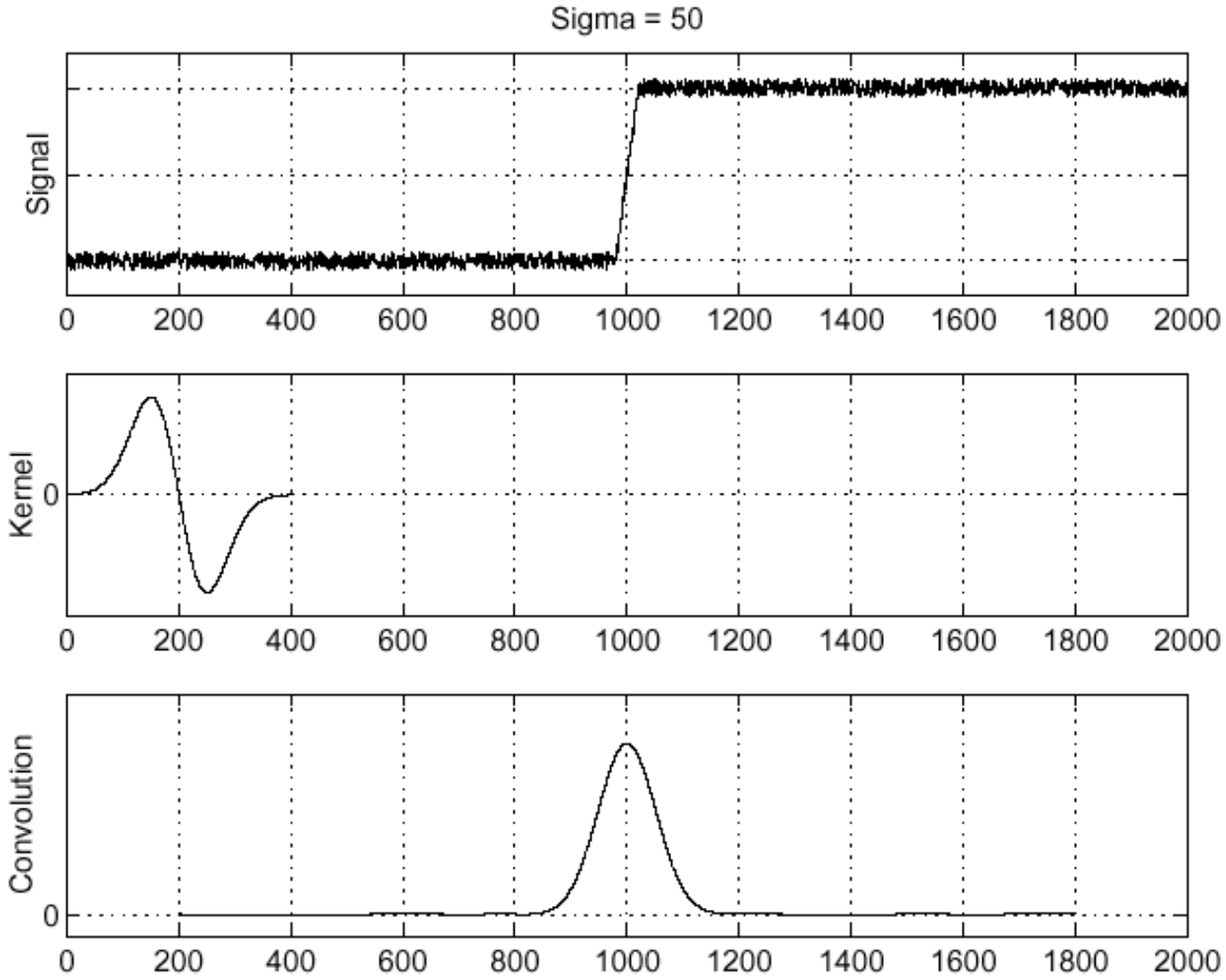
- Each sample is replaces by a weighted average of its neighbors.
- This yields a smoothed version of the original signal.

Example 2: Convolution with the Derivative of a Gaussian

$$f$$

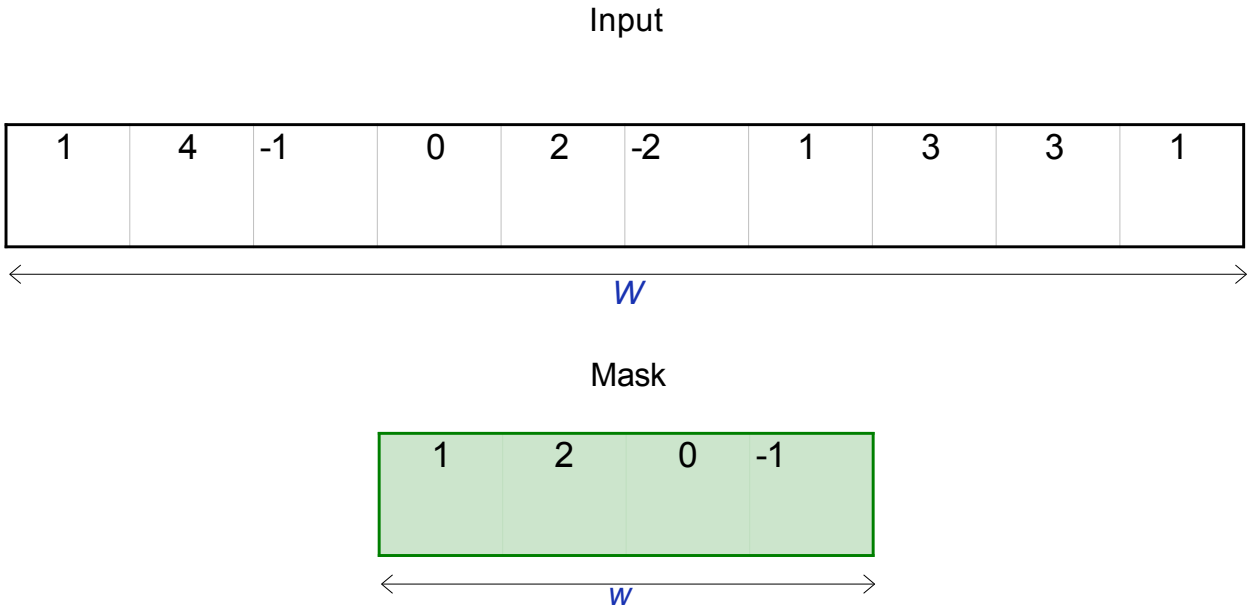
$$\frac{\partial g}{\partial x}$$

$$\frac{\partial}{\partial x}(g * f) = \frac{\partial g}{\partial x} * f$$

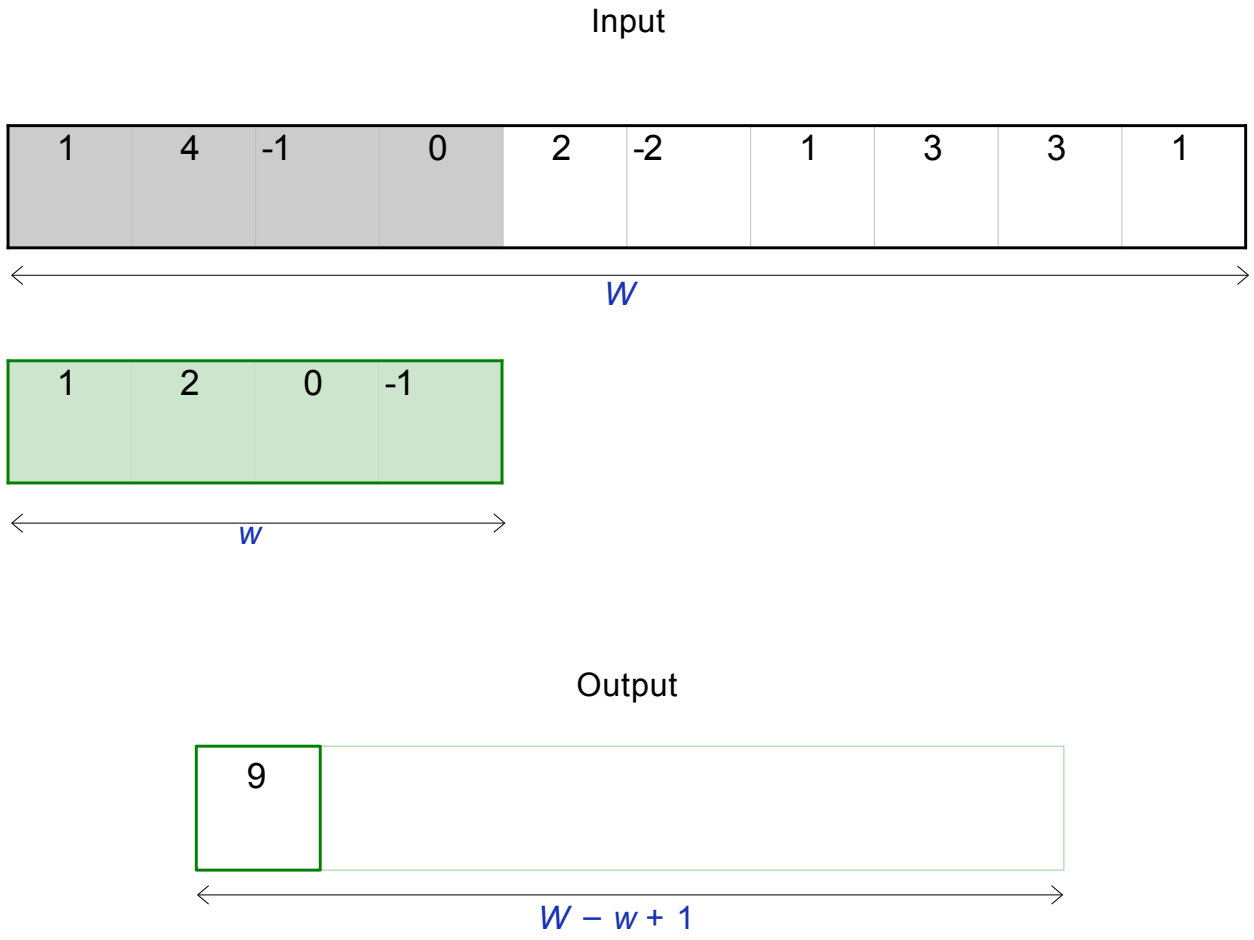


- Convolving with the derivative of a gaussian is the same as smoothing first and then differentiating.

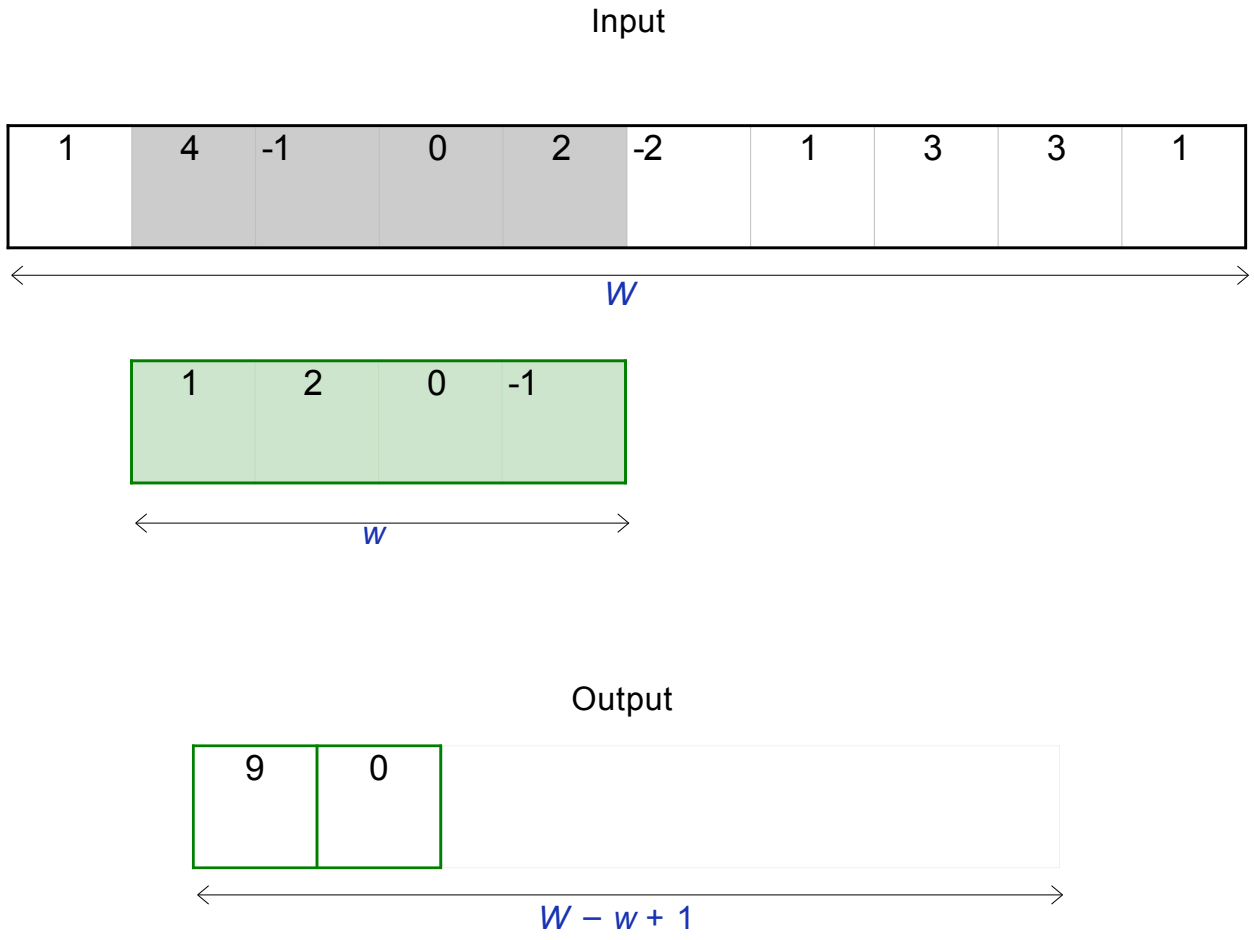
Discrete 1D Convolution



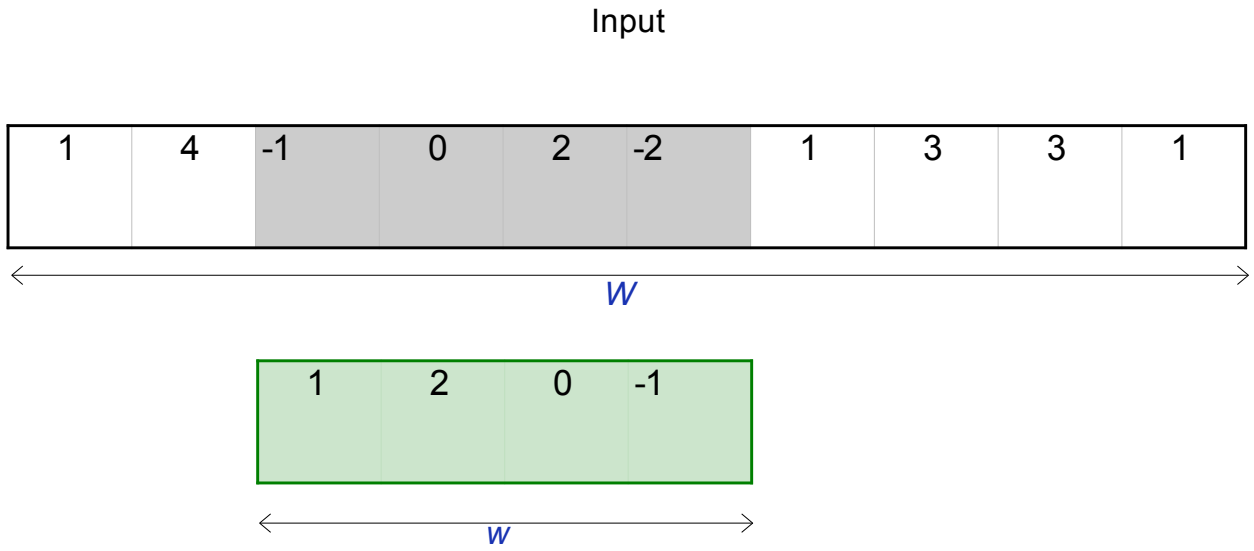
Discrete 1D Convolution



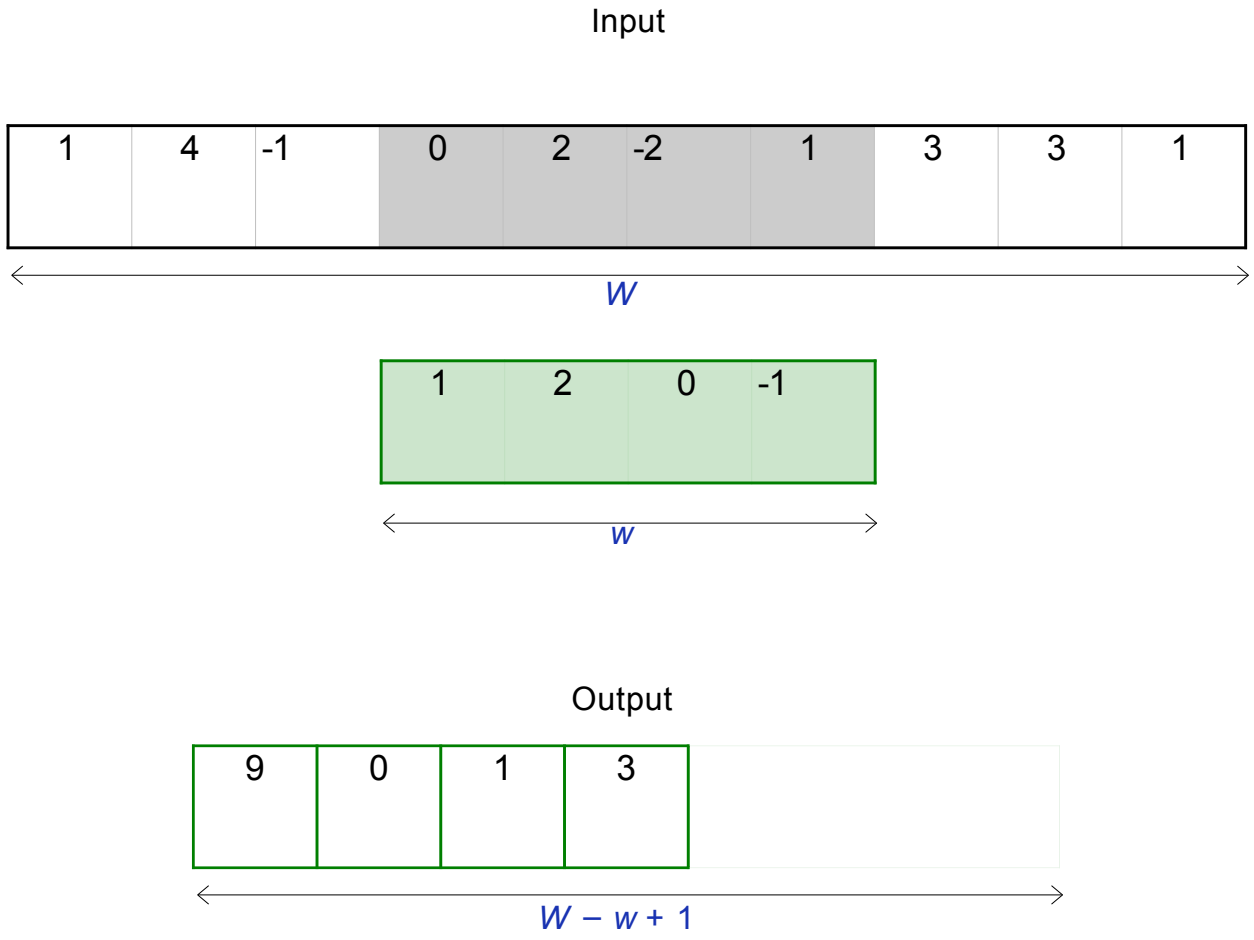
Discrete 1D Convolution



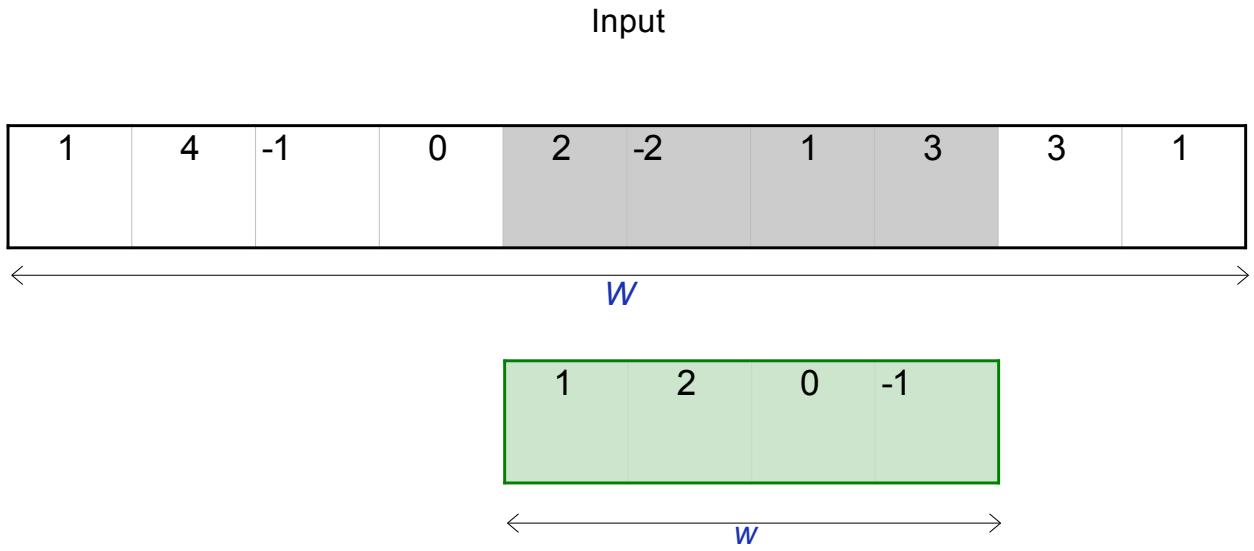
Discrete 1D Convolution



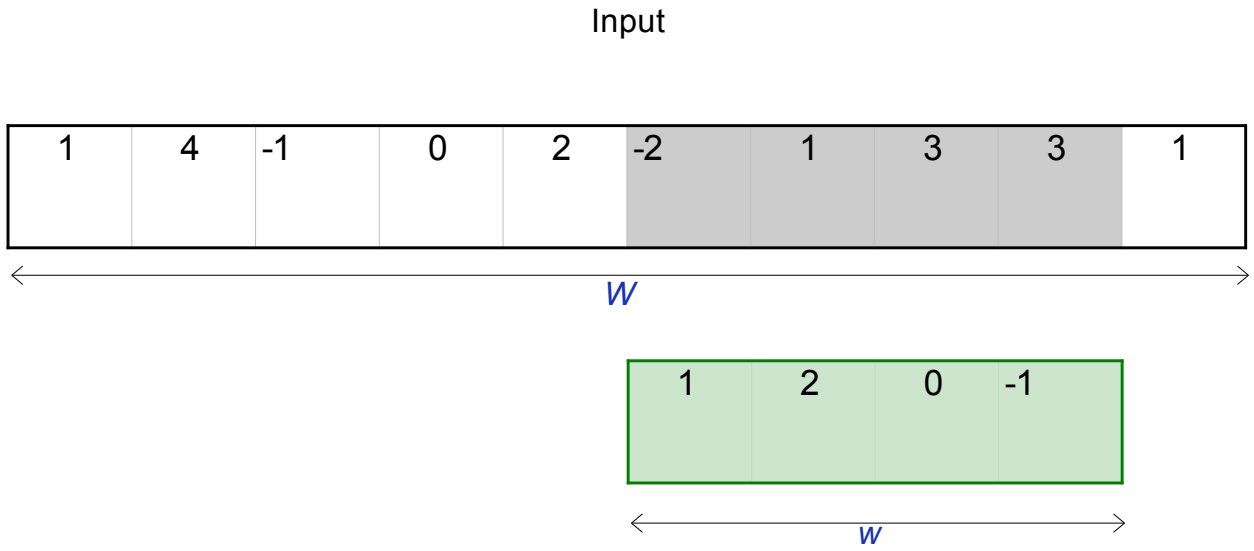
Discrete 1D Convolution



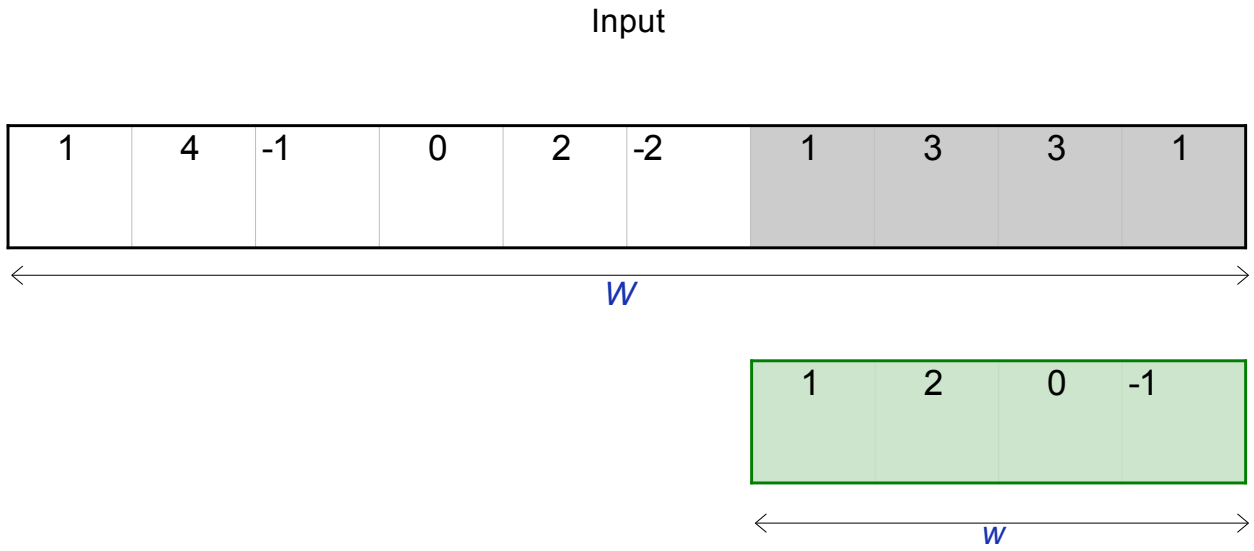
1D Convolution



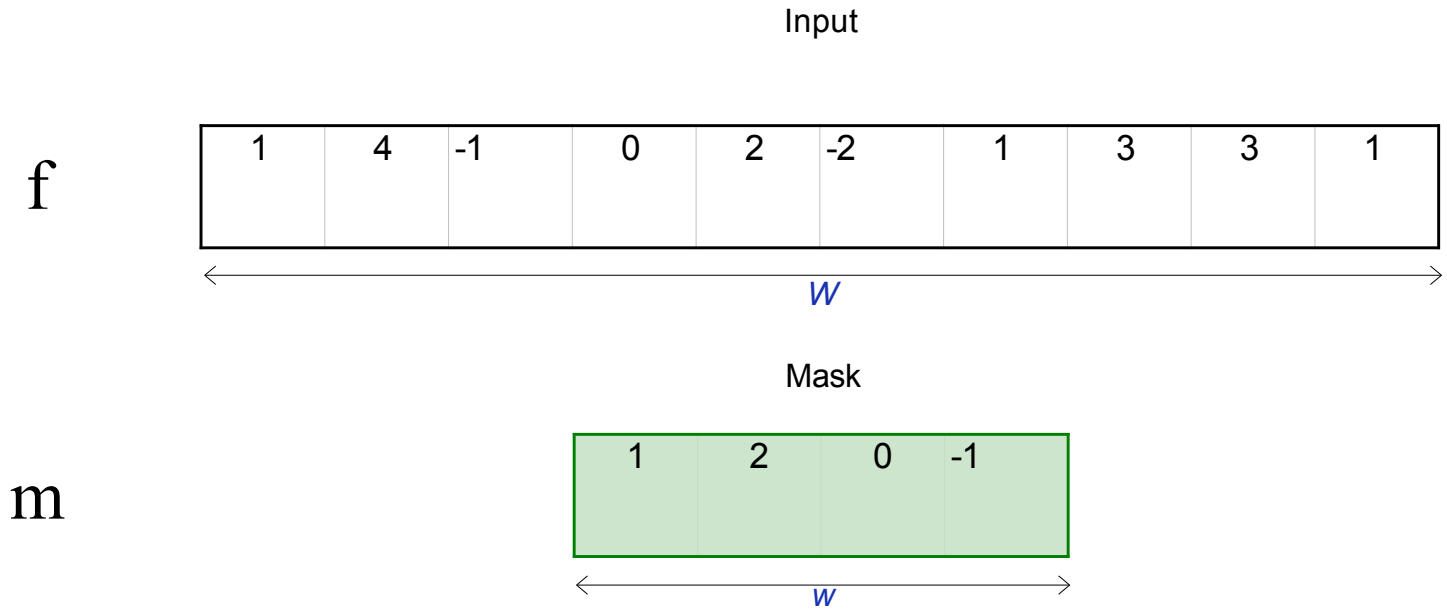
Discrete 1D Convolution



Discrete 1D Convolution



Discrete 1D Convolution

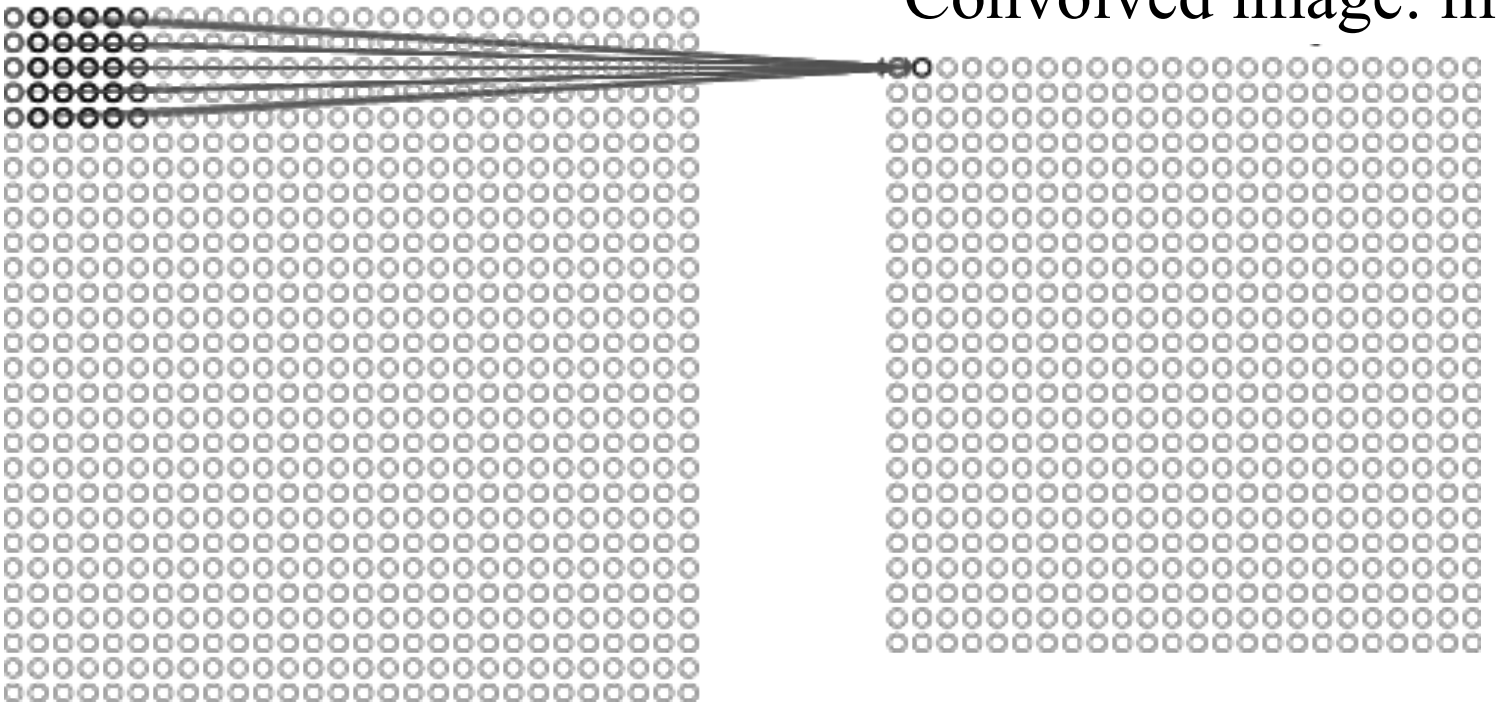


$$m * f(x) = \sum_{i=0}^{w} m(i) f(x - i)$$

Discrete 2D Convolution

Input image: f

Convolved image: m**f

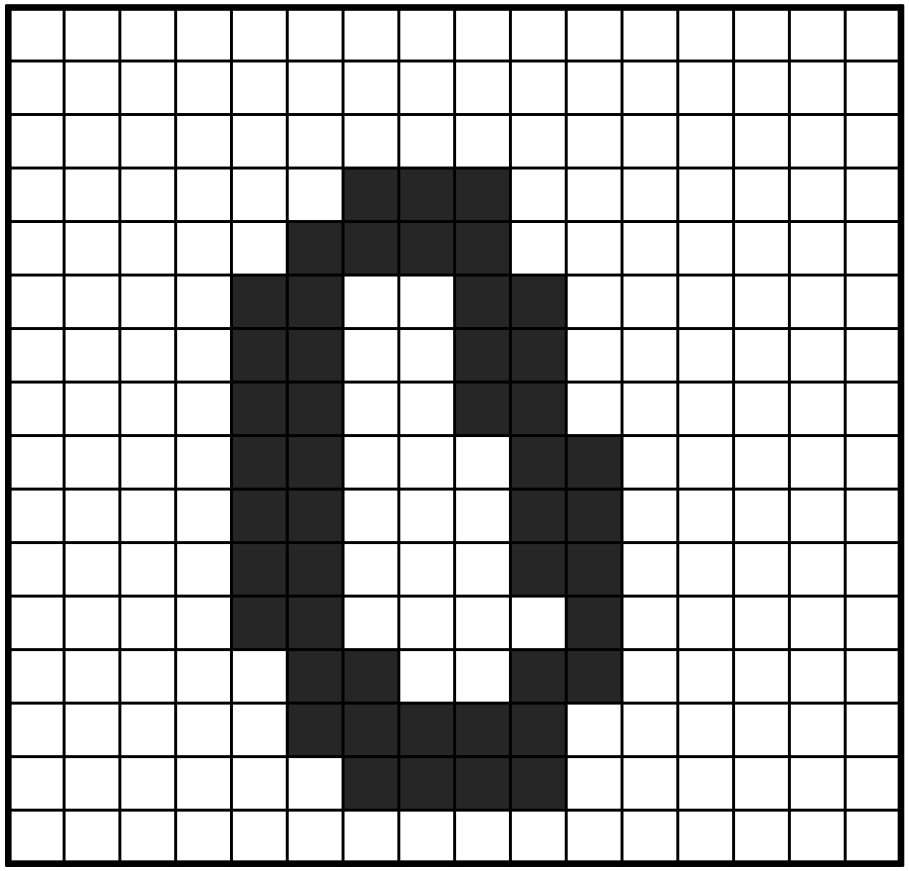


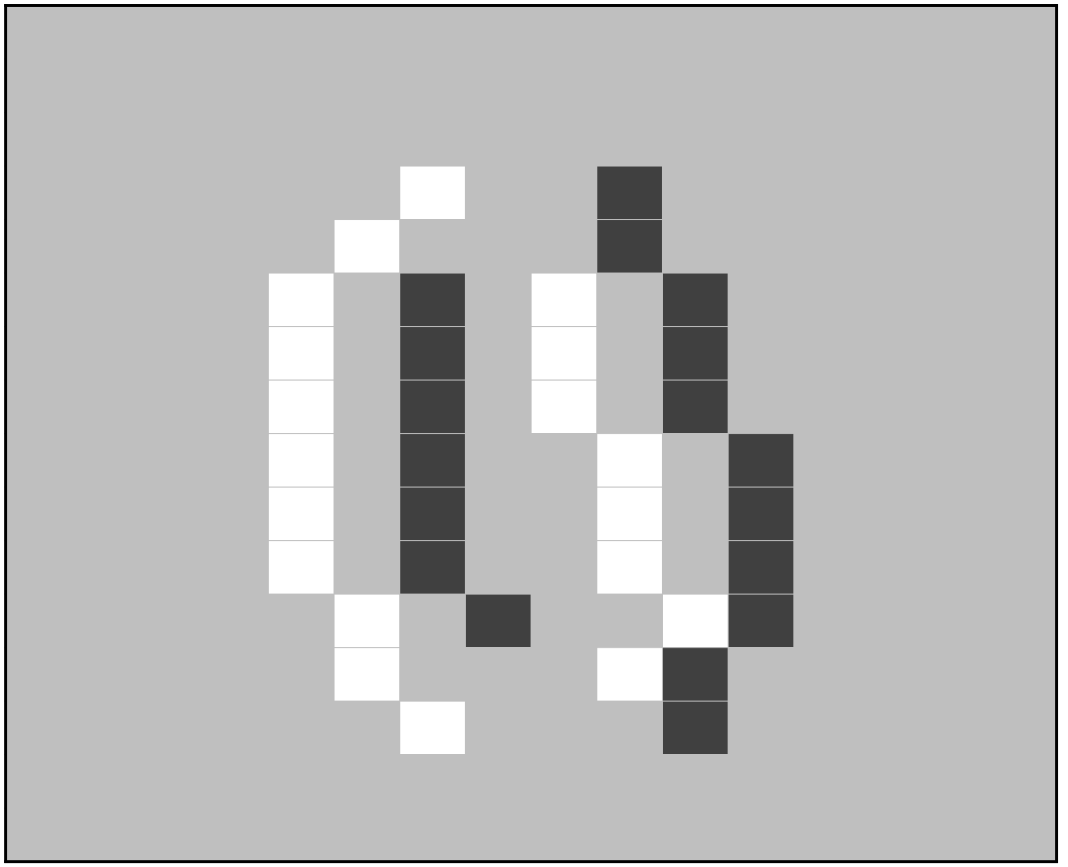
Convolution mask m, also known as a *kernel*.

$$\begin{bmatrix} m_{11} & \cdots & m_{1w} \\ \cdots & \cdots & \cdots \\ m_{w1} & \cdots & m_{ww} \end{bmatrix}$$

$$m**f(x,y) = \sum_{i=0}^{w}\sum_{j=0}^{w} m(i,j)f(x-i,y-j)$$

Convolution Example



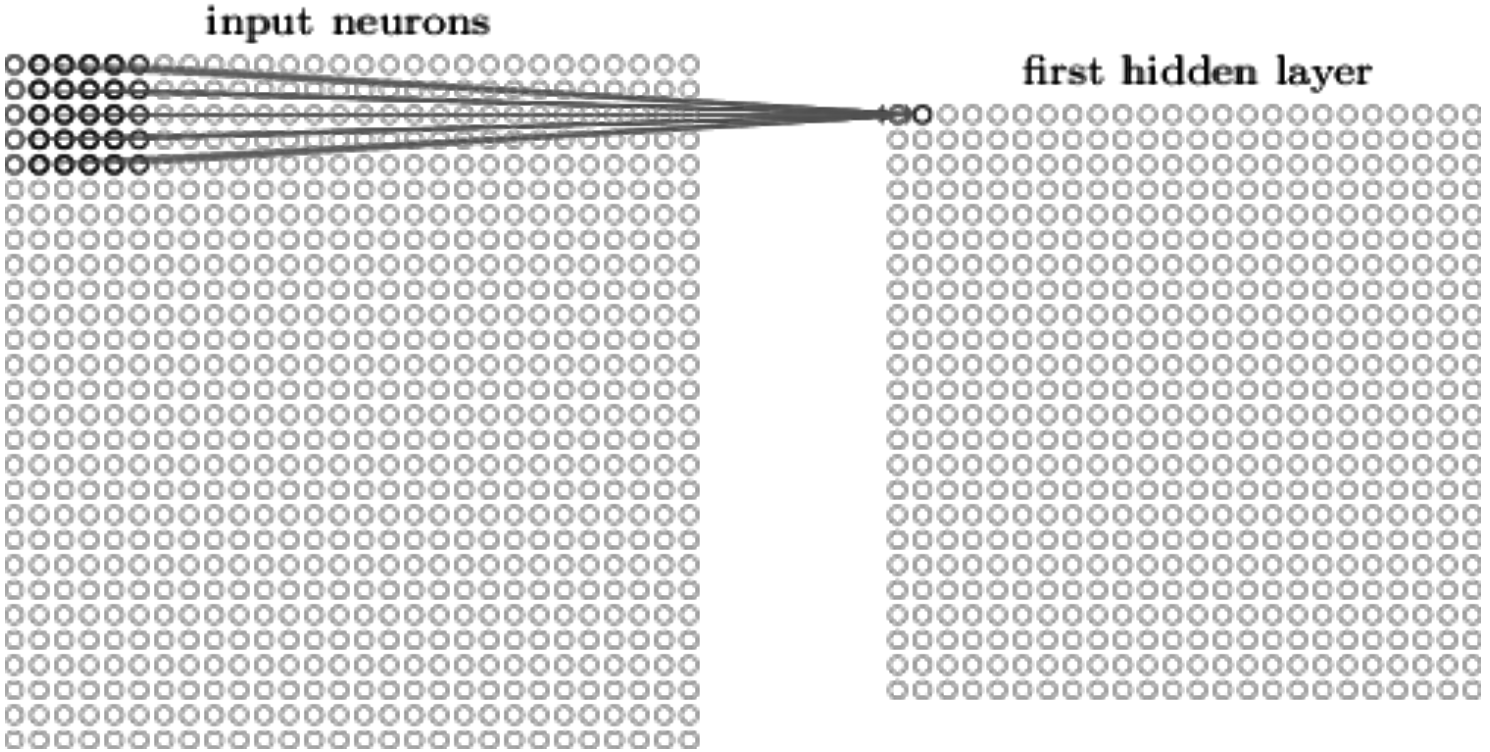


-1	**0**	**+1**
-1	**0**	**+1**
-1	**0**	**+1**

$$\mathbf{h}_{1,1} = \mathrm{s}'\left(\mathbf{f}_{1,1} * \mathbf{x} + \mathbf{b}_{1,1}\right)$$

This approximates an x derivative.

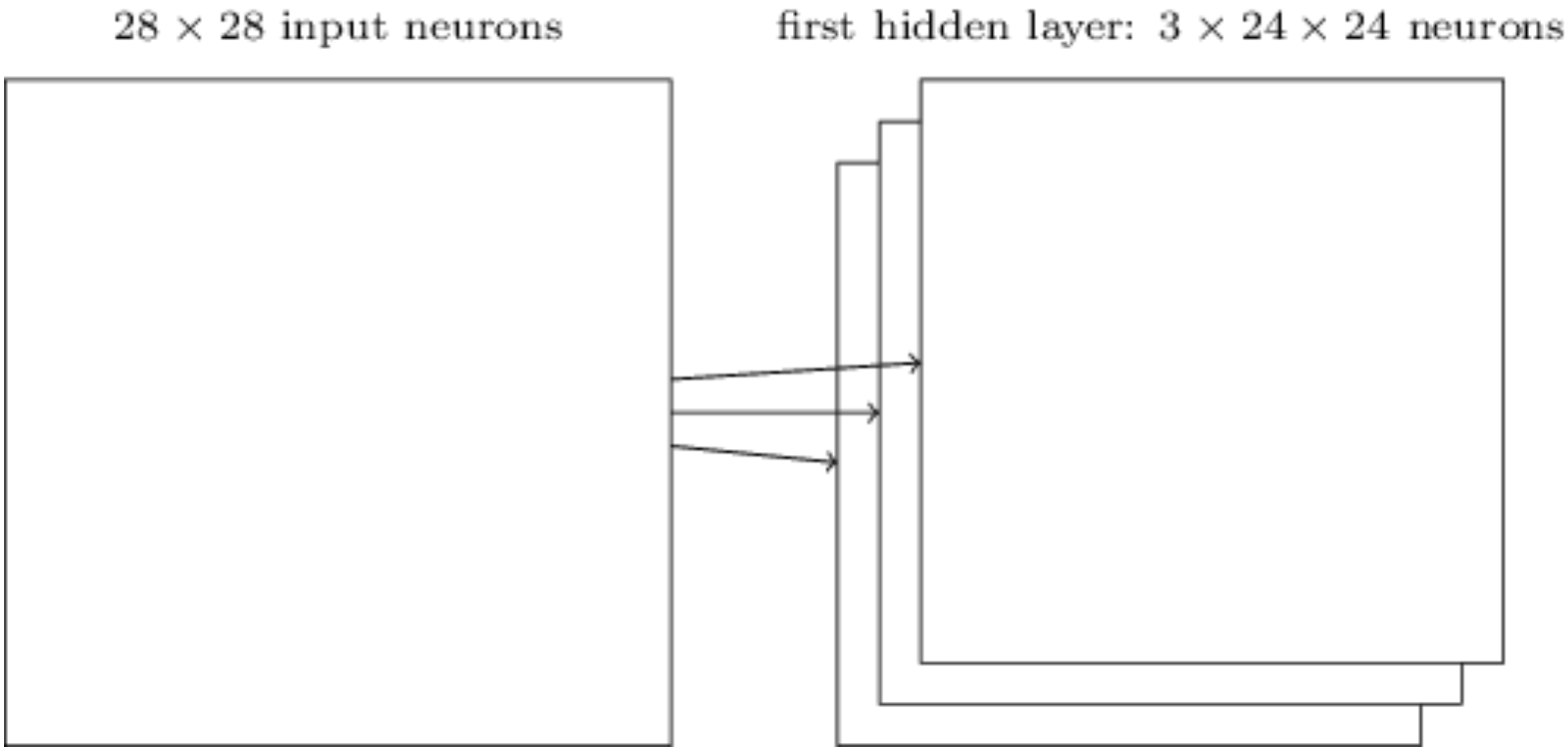
2D Convolutional Layer



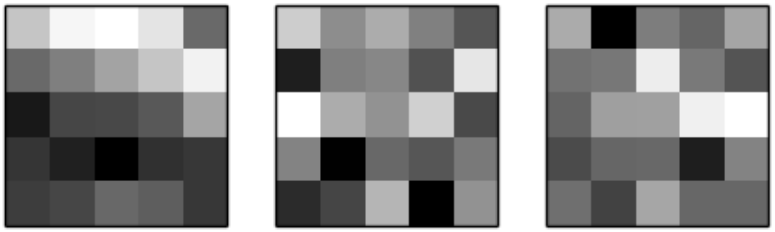
$$a^1_{i,j} = \sigma\left(b + \sum_{x=0}^{n_x}\sum_{y=0}^{n_y} w_{x,y} a^0_{i+x,j+y}\right)$$

- The same weights $w_{x,y}$ are used to compute all the activations.
- There are far fewer weights than in a fully connected layer.
- The neighborhood relationships are explicitly used.

Feature Maps

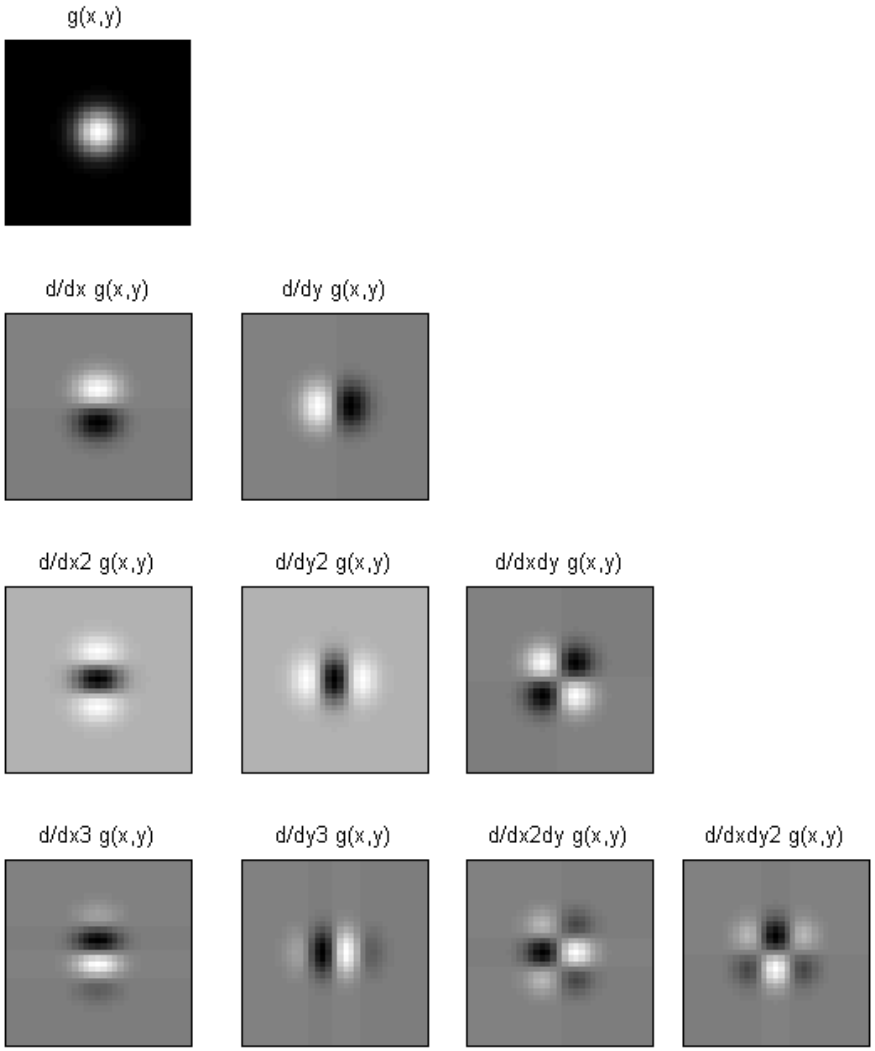


Filters:

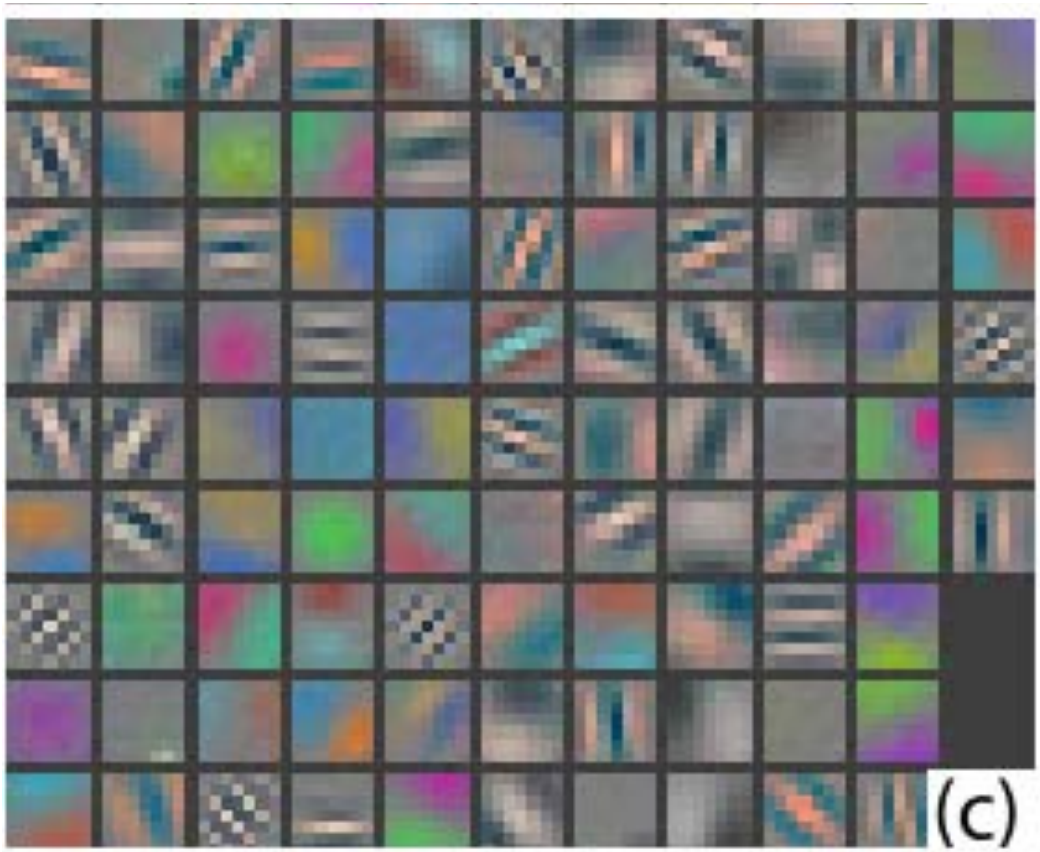


- In practice, one uses several **filters**, that is, sets of weights $w_{x,y}$, to compute several convolved versions of the input.
- These are known as **feature maps**.

Derivative Filters

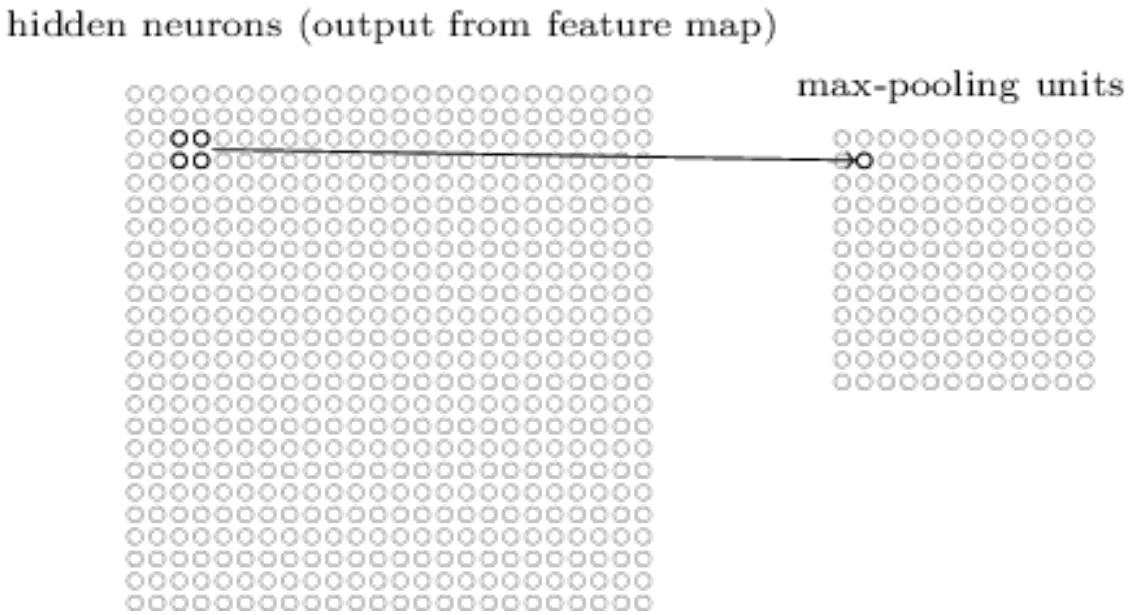


Derivatives



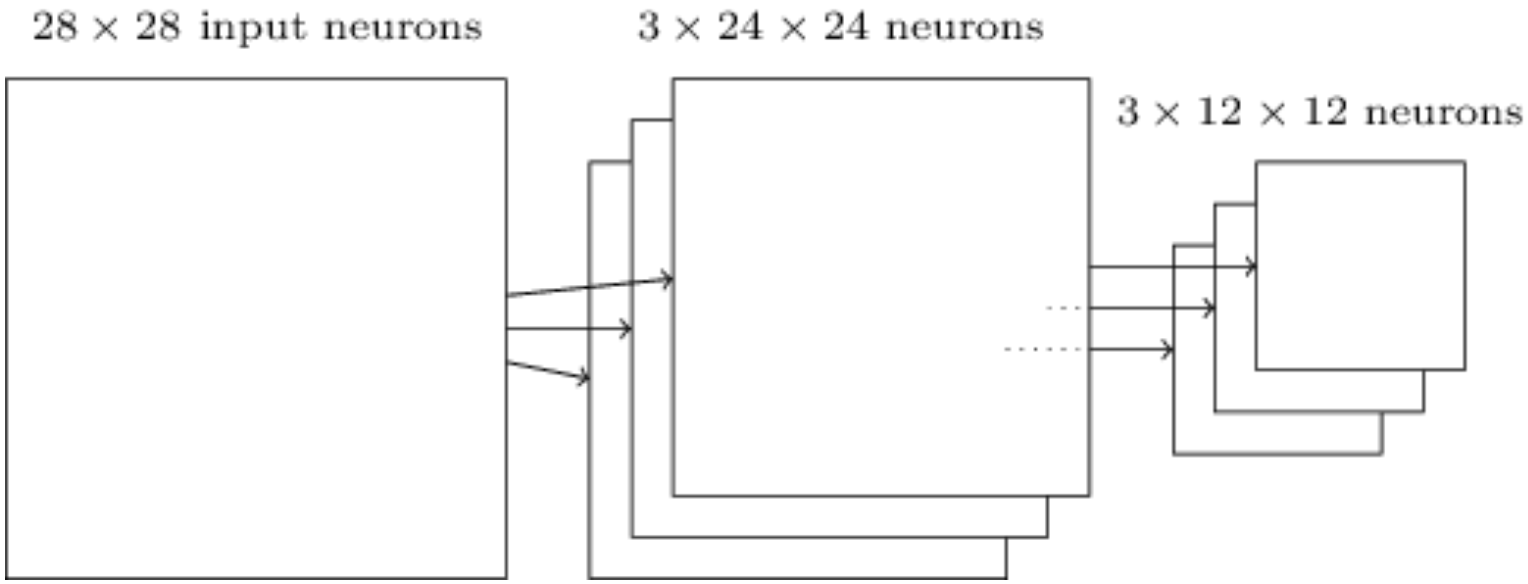
Learned filters

Pooling Layer



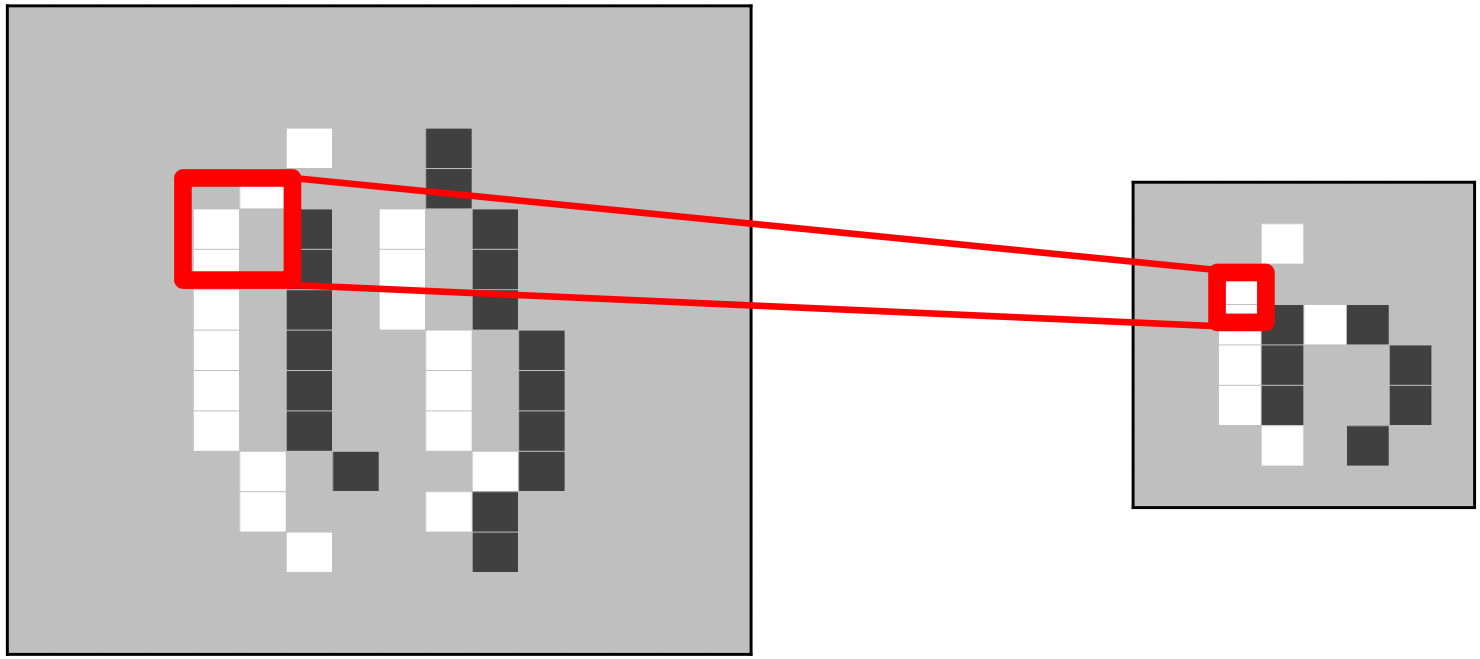
- Reduces the number of inputs by replacing all activations in a neighborhood by a single one.
- Can be thought as asking if a particular feature is present in that neighborhood while ignoring the exact location.

Adding the Pooling Layers



The output size is reduced by the pooling layers.

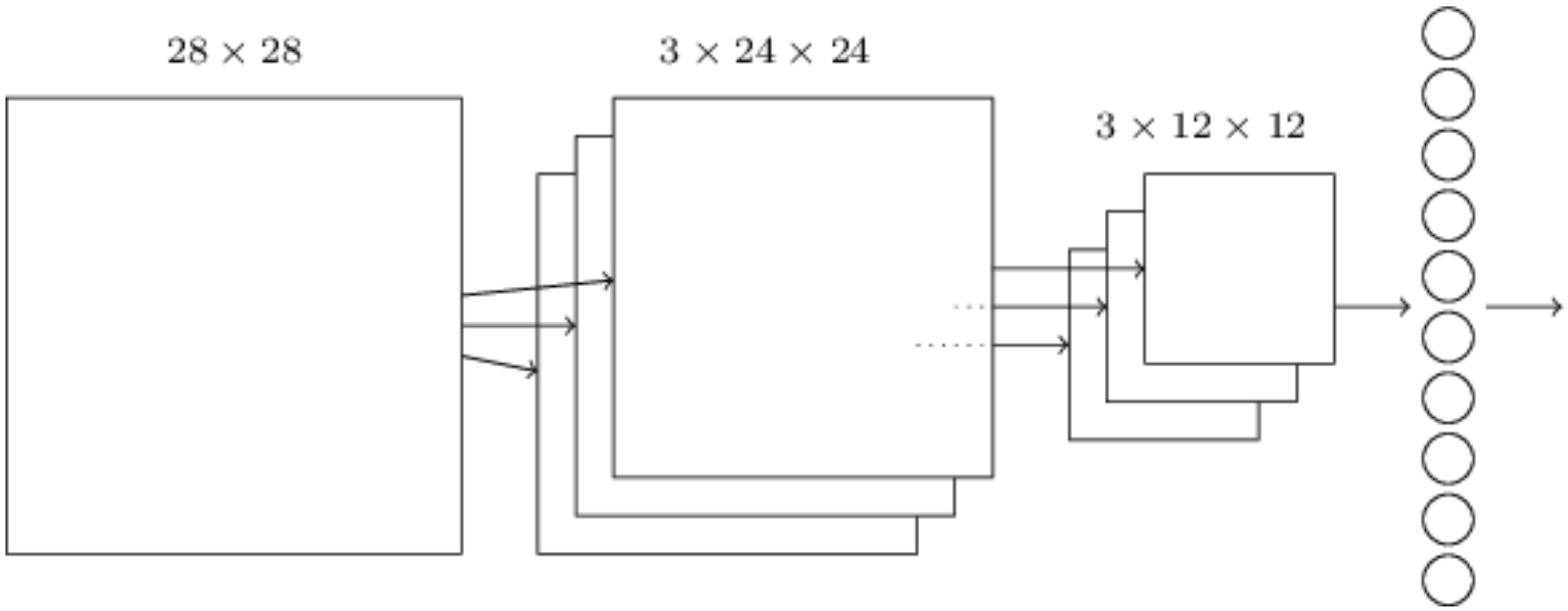
Pooling Example



Max-pooling:

$$\mathbf{h}_i[u,v] = \max\{ \quad \mathbf{h}_{i-1}[2u, \quad 2v], \quad \mathbf{h}_{i-1}[2u, \quad 2v+1], \quad \mathbf{h}_{i-1}[2u+1, \quad 2v], \quad \mathbf{h}_{i-1}[2u+1, \quad 2v+1] \quad \}$$

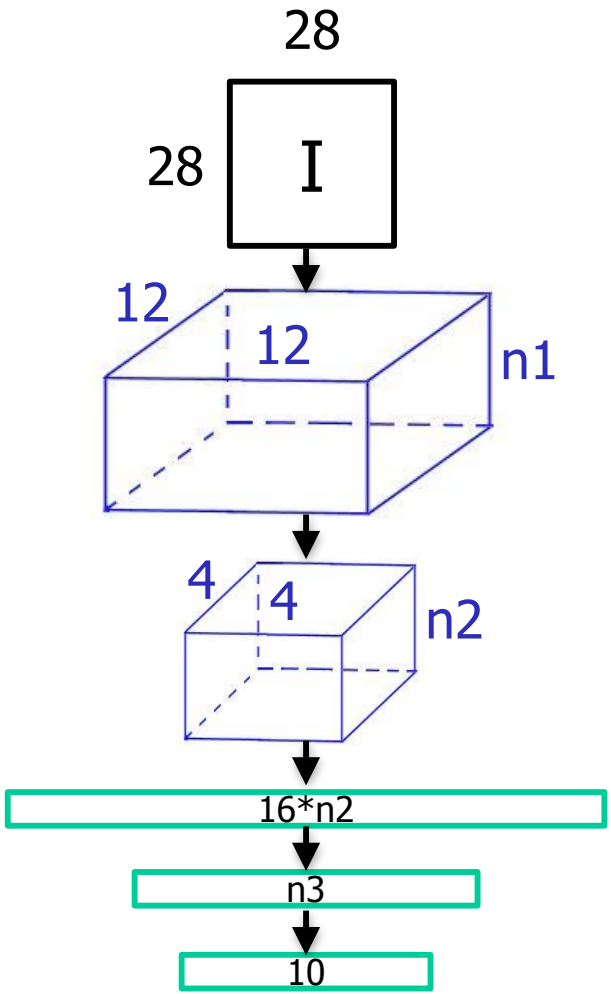
Adding a Fully Connected Layer



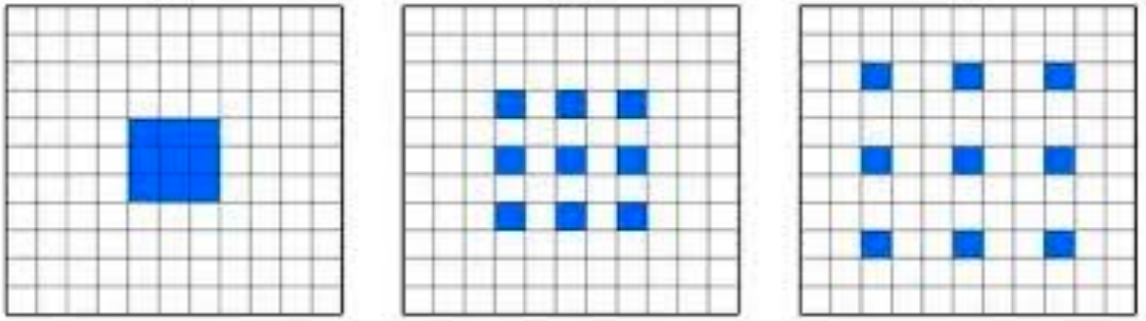
- Each neutron in the final fully connected layer is connected to all neurons in the preceding one.
- Deep architecture with many parameters to learn but still far fewer than an equivalent multilayer perceptron.

PyTorch Translation (1)

```
class ConvNet(nn.Module):

    def __init__(n1=10,n2=20,n3=50):
        self.n1 = n1
        self.n2 = n2
        self.cv1 = nn.Conv2d(1, n1,  kernel_size=5)
        self.cv2 = nn.Conv2d(n1,n2, kernel_size=5)
        self.fc1 = nn.Linear(n2*16, n3)
        self.fc2 = nn.Linear(n3,10)

    def forward(self,x):
        x = F.relu(F.max_pool2d(self.cv1(x), 2))
        x = F.relu(F.max_pool2d(self.cv2(x), 2))
        x = x.view(-1,16*self.n2)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x,dim=1)
```



Introducing a Stride Parameter

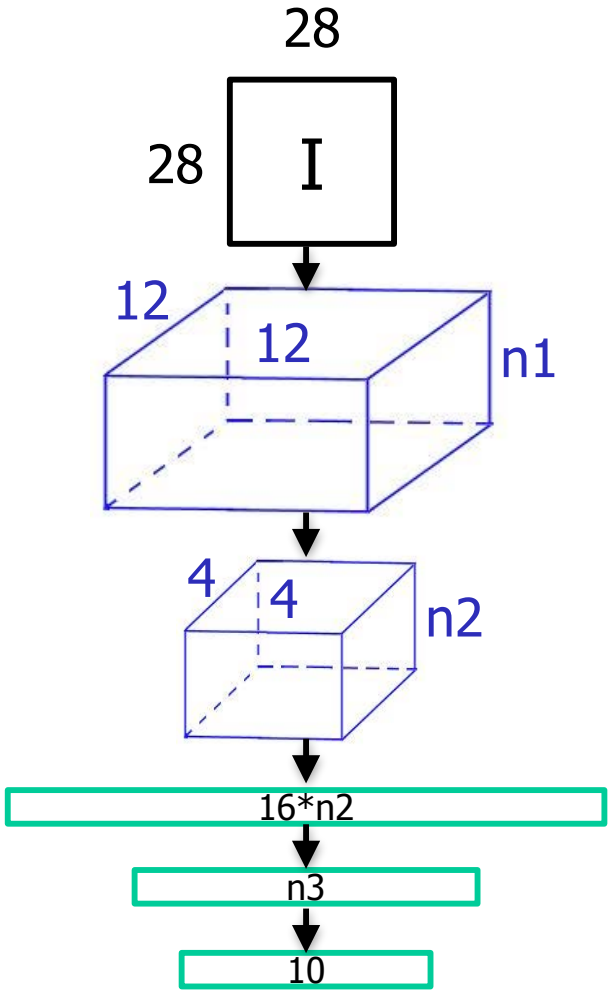


stride=1 stride=2 stride=3

- Using a stride larger than one reduces and convolves at the same time.
- This ignores parts of the signal and is often preceded by one of stride one to mitigate this.

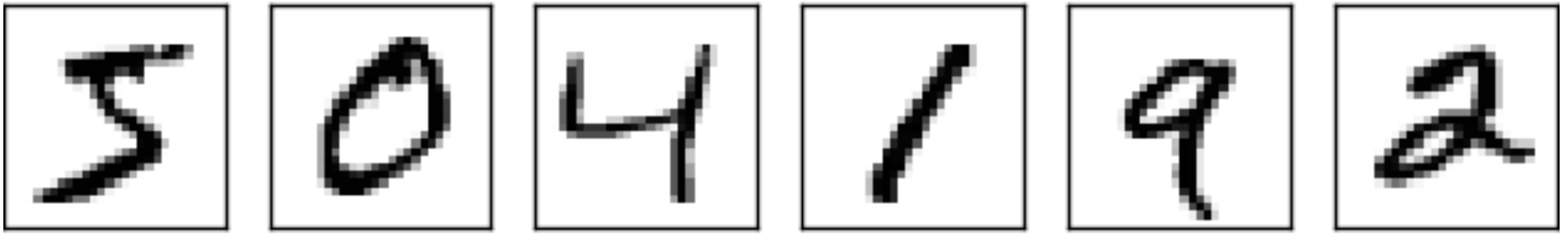
PyTorch Translation (2)

```
class ConvNet(nn.Module):

    def __init__(n1=10,n2=20,n3=50):
        self.n1 = n1
        self.n2 = n2
        self.cv1 = nn.Conv2d(1, n1,kernel_size=5,stride=2)
        self.cv2 = nn.Conv2d(n1,n2,kernel_size=5,stride=2)
        self.fc1 = nn.Linear(n2*16,n3)
        self.fc2 = nn.Linear(n3,10)

    def forward(self,x):
        x = F.relu(self.cv1(x))
        x = F.relu(self.cv2(x))
        x = x.view(-1, 16*self.n2)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x,dim=1)
```



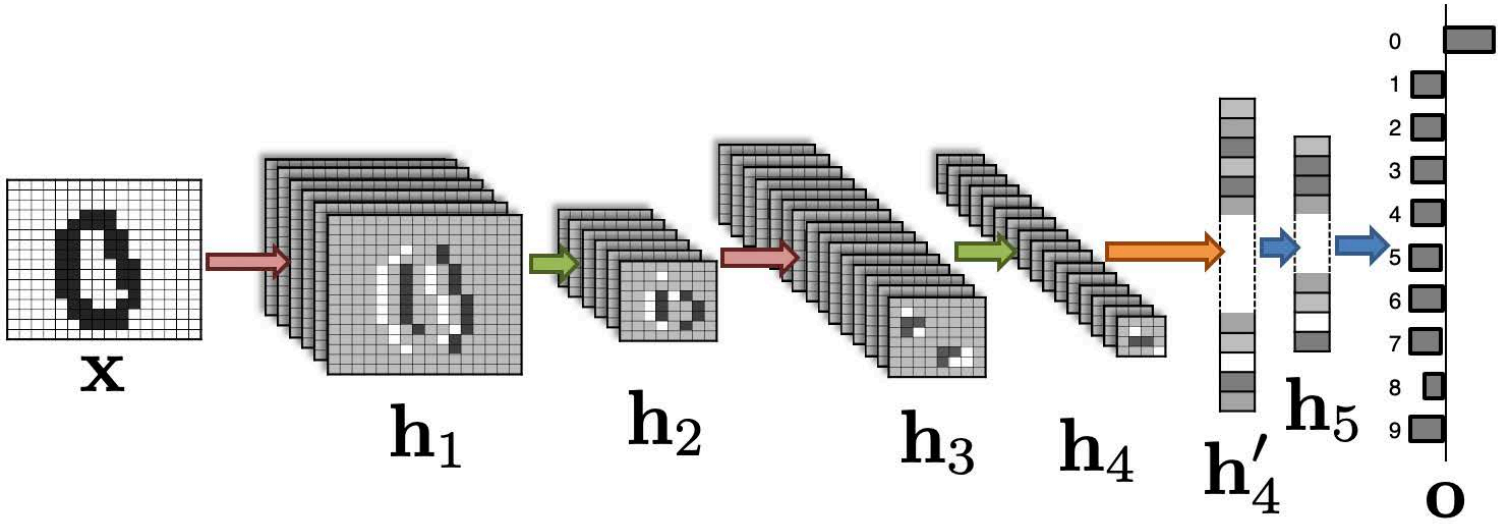
- No explicit pooling.
- Similar accuracy if additional convolution with stride one added.

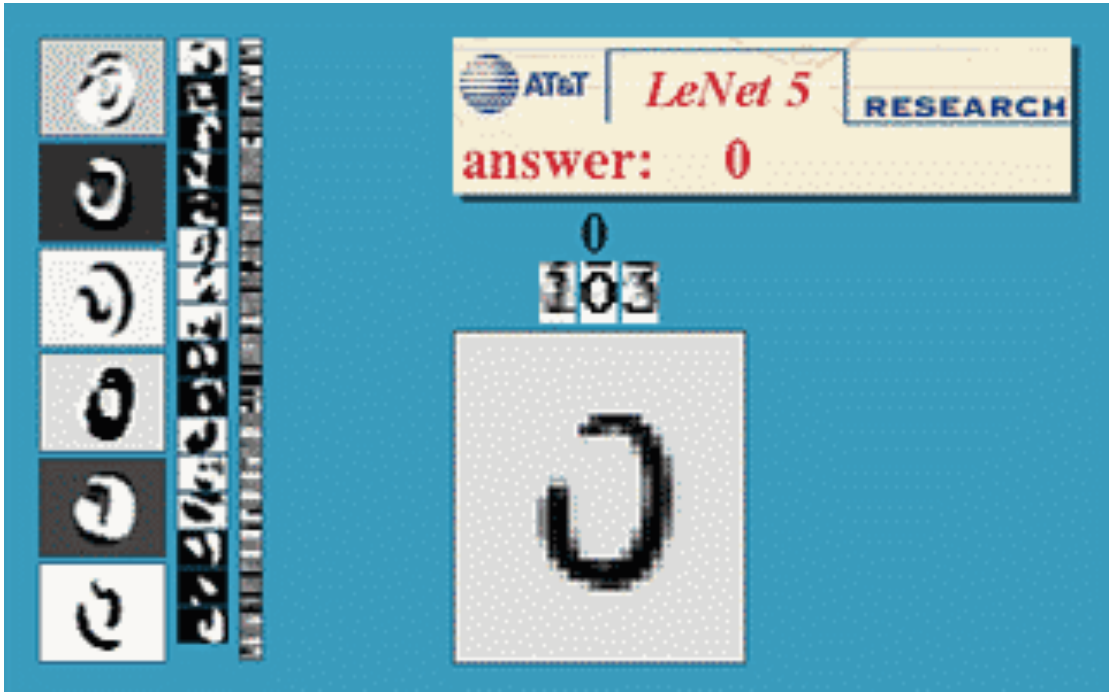
MNIST

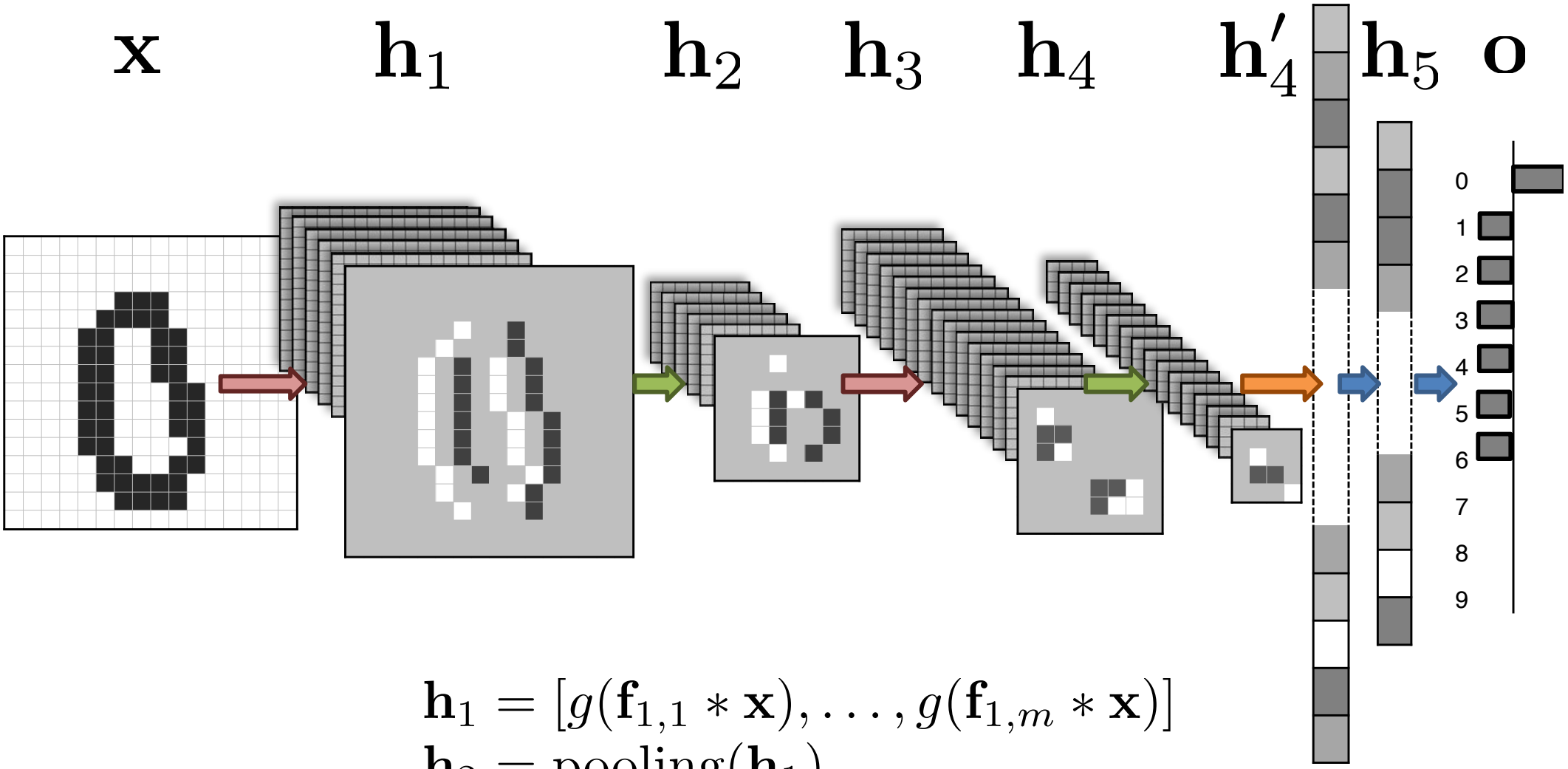


- The network takes as input 28x28 images represented as 784D vectors.
- The output is a 10D vector giving the probability of the image representing any of the 10 digits.
- There are 50’000 training pairs of images and the corresponding label, 10’000 validation pairs, and 5’000 testing pairs.

Lenet (1989-1999)

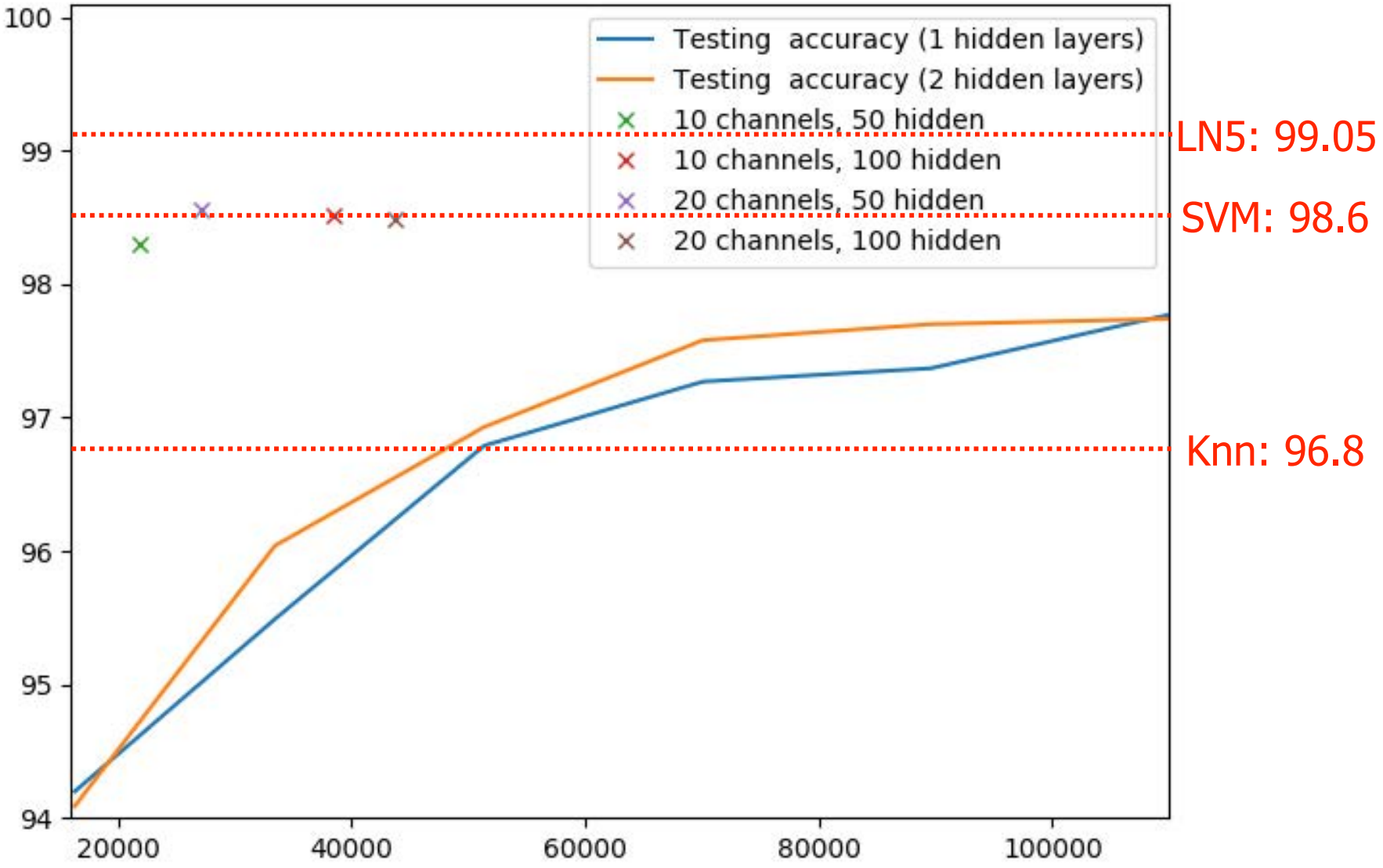






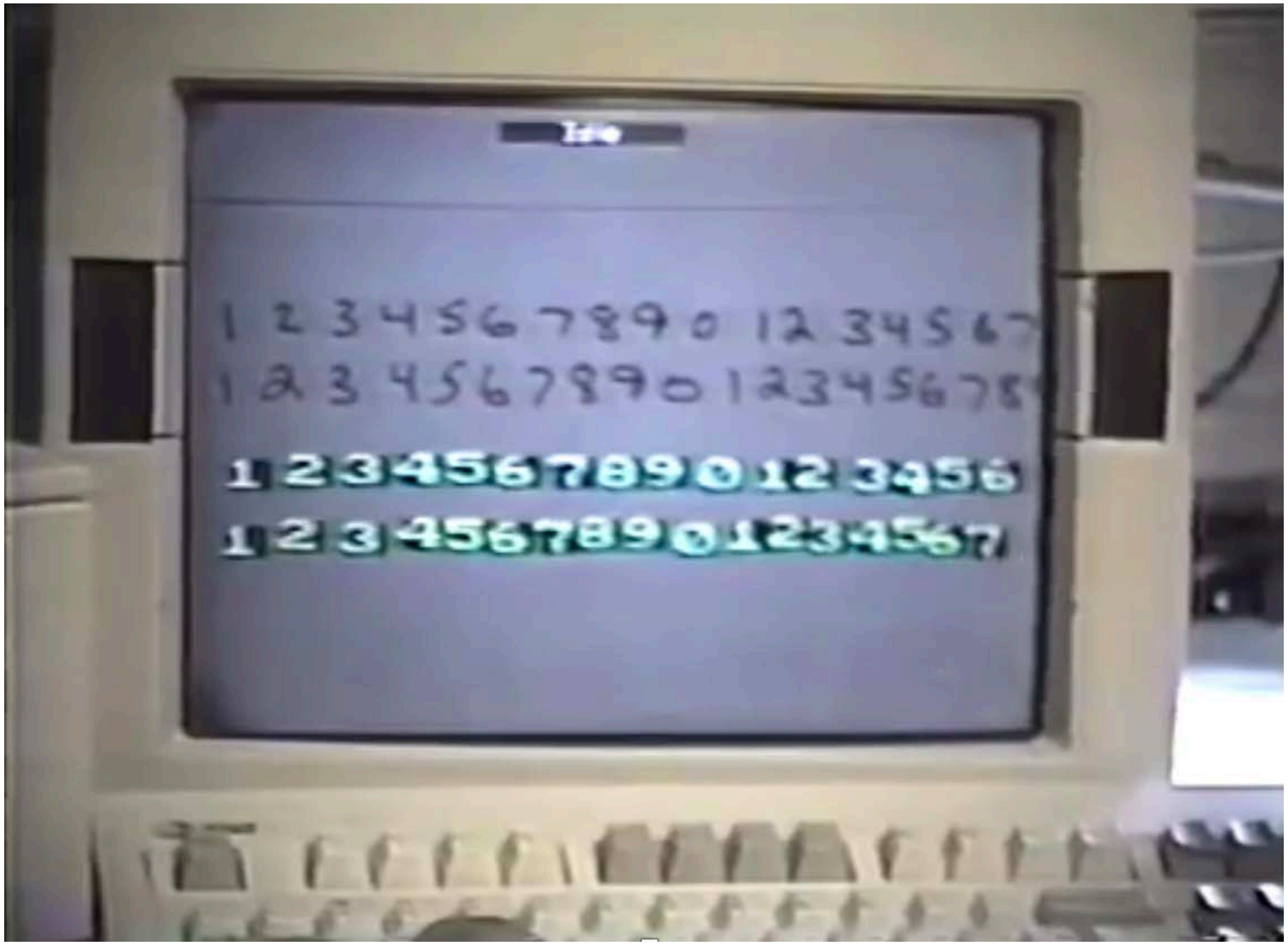
$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$
$$\mathbf{h}_2 = \text{pooling}(\mathbf{h}_1)$$
$$\mathbf{h}_3 = [g(\mathbf{f}_{3,1} * \mathbf{h}_2), \ldots, g(\mathbf{f}_{3,n} * \mathbf{h}_2)]$$
$$\mathbf{h}_4 = \text{pooling}(\mathbf{h}_3)$$
$$\mathbf{h}'_4 = \text{Vec}(\mathbf{h}_4)$$
$$\mathbf{h}_5 = g(\mathbf{W}_5\mathbf{h}'_4 + \mathbf{b}_5)$$
$$\mathbf{o} = \mathbf{W}_6\mathbf{h}_5 + \mathbf{b}_6$$

Lenet Results



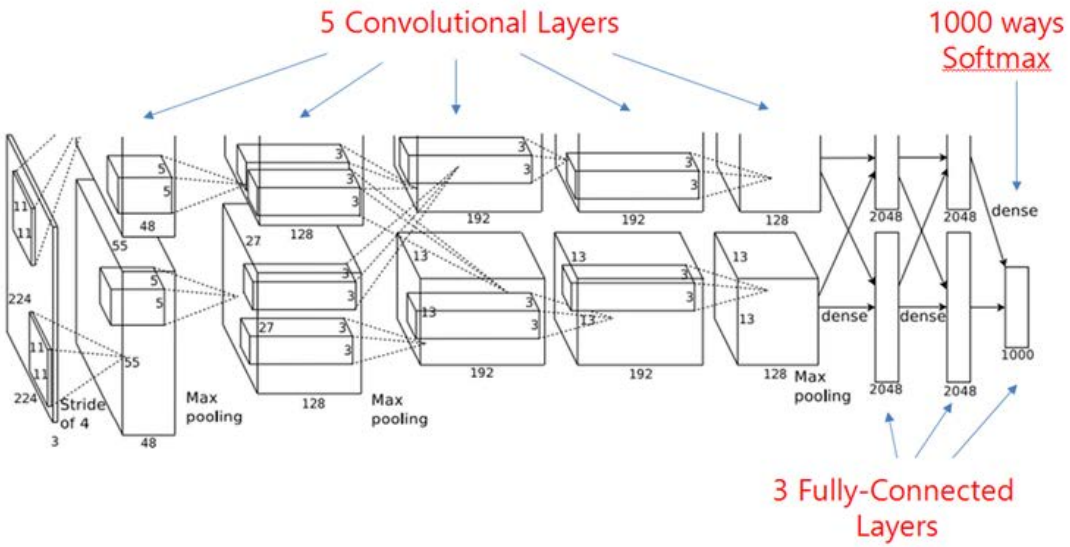
Given the appropriate architecture, the CNN outperforms the other approaches, whereas the MLP did not.

Lenet5 (1992)



- Worked beautifully on MNIST.
- Very few people believed it would scale up.

AlexNet (2012)



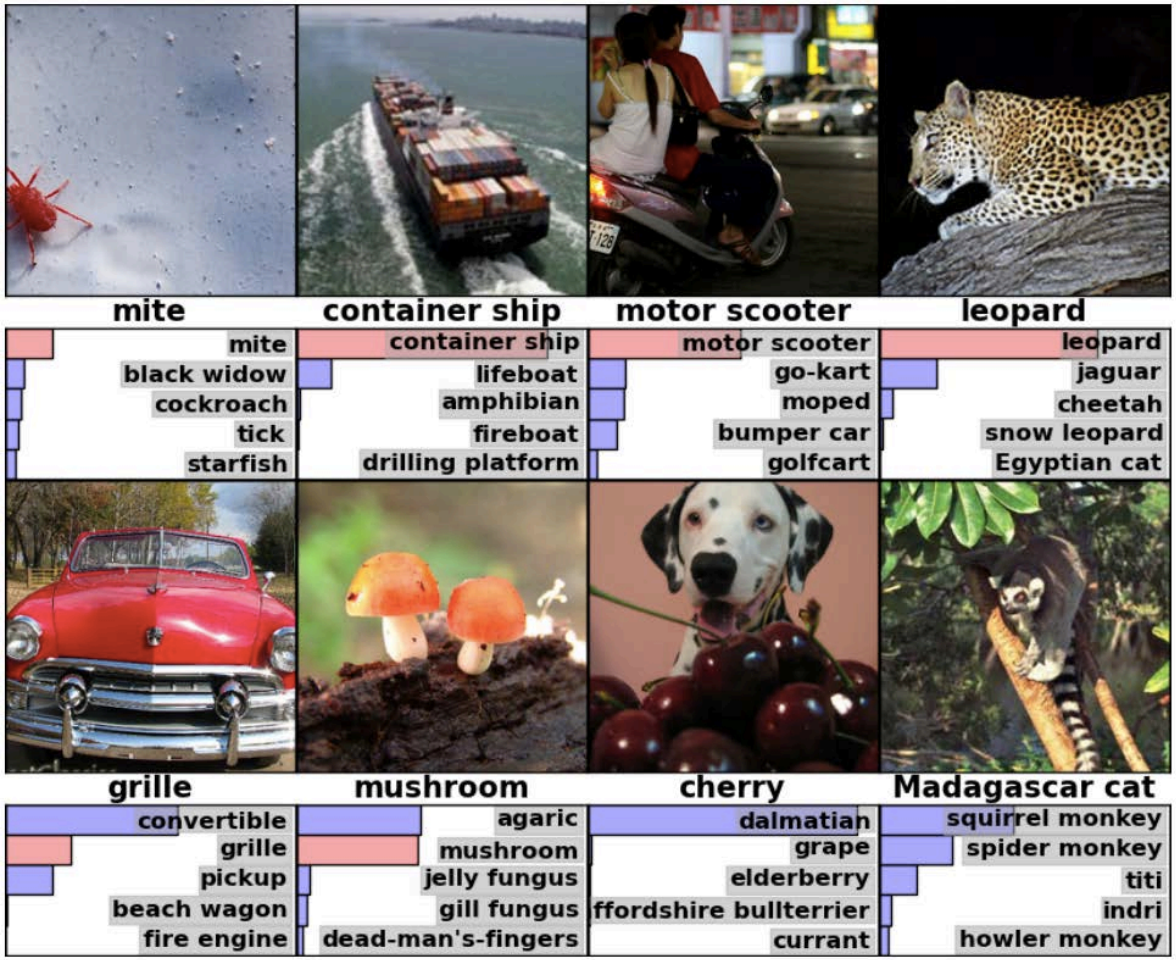
Task: Image classification

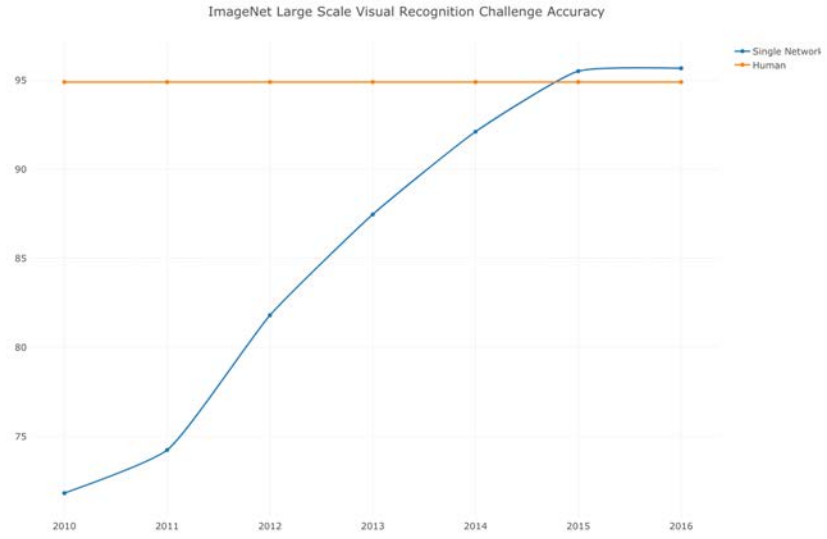
Training images: Large Scale Visual Recognition Challenge 2010

Training time: 2 weeks on 2 GPUs

Major Breakthrough: Training large networks has now been shown to be practical!!

AlexNet Results

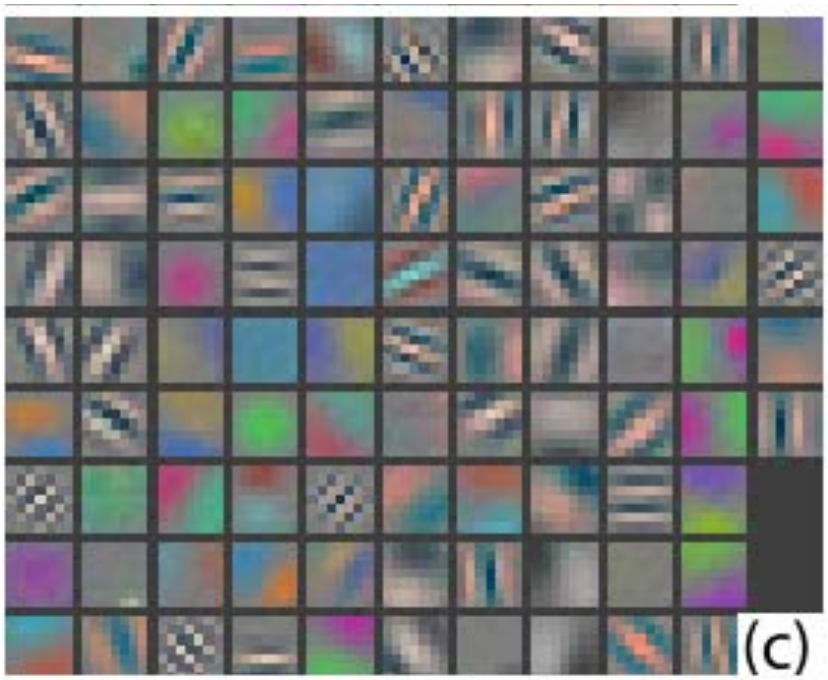




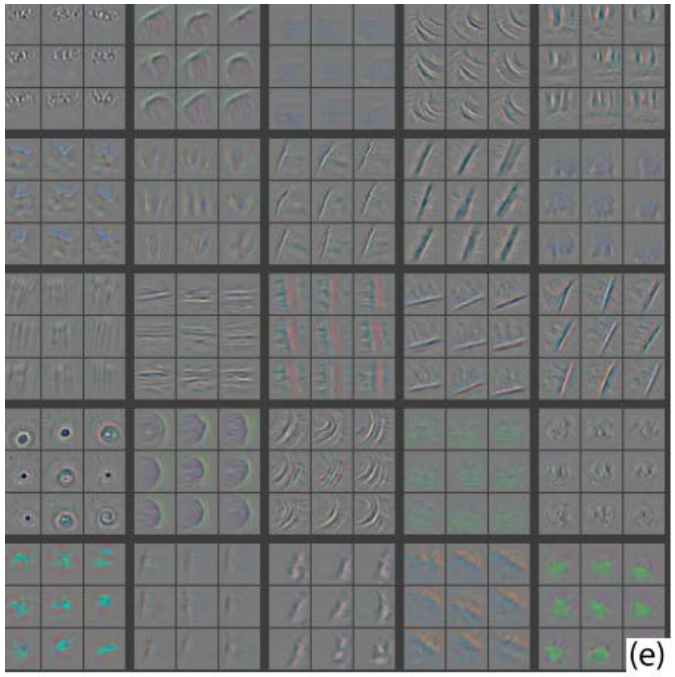
- At the 2012 ImageNet Large Scale Visual Recognition Challenge, AlexNet achieved a top-5 error of 15.3%, more than 10.8% lower than the runner up.
- Since 2015, networks outperform humans on this task.

Krizhevsky, NIPS'12

Feature Maps



First convolutional layer

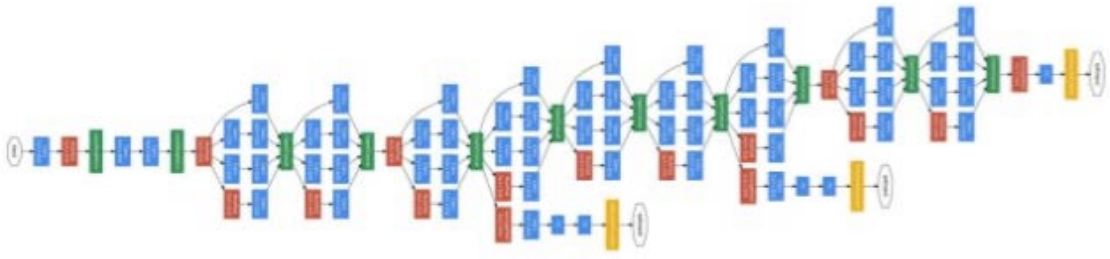


Second convolutional layer

- Some of the convolutional masks are very similar to oriented Gaussian or Gabor filters.
- The trained neural nets compute oriented derivatives, which the brain is also **believed** to do.

Size and Depth Matter

VGG19, 3 weeks of training.



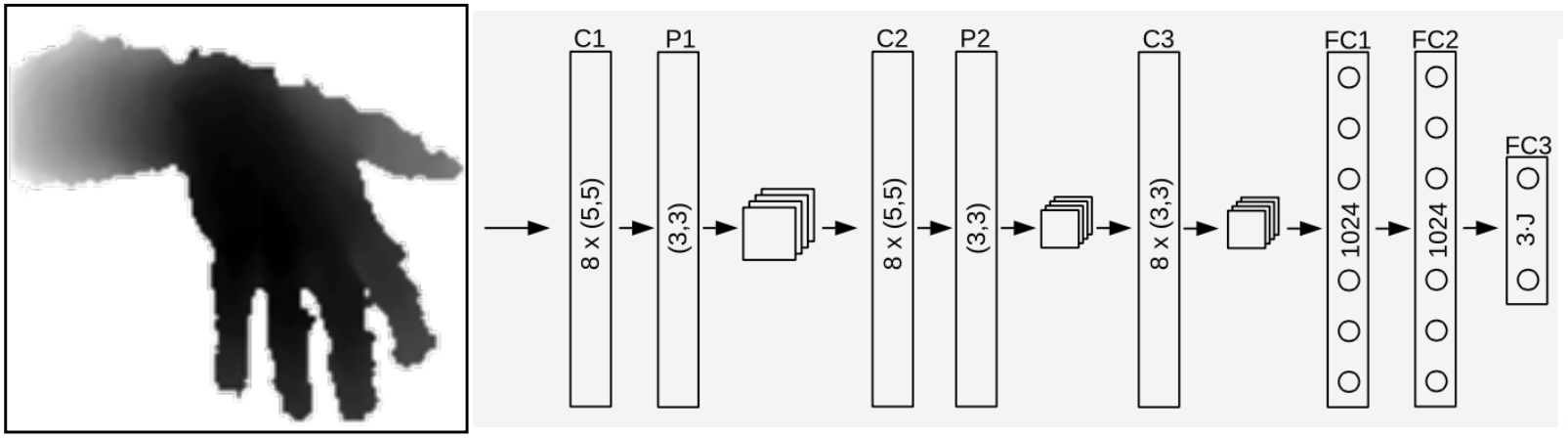
"hibiscus"

"dahlia"

GoogleLeNet.

"It was demonstrated that the representation depth is beneficial for the classification accuracy, and that state-of-the-art performance on the ImageNet challenge dataset can be achieved using a conventional ConvNet architecture."

Hand Pose Estimation (2015)

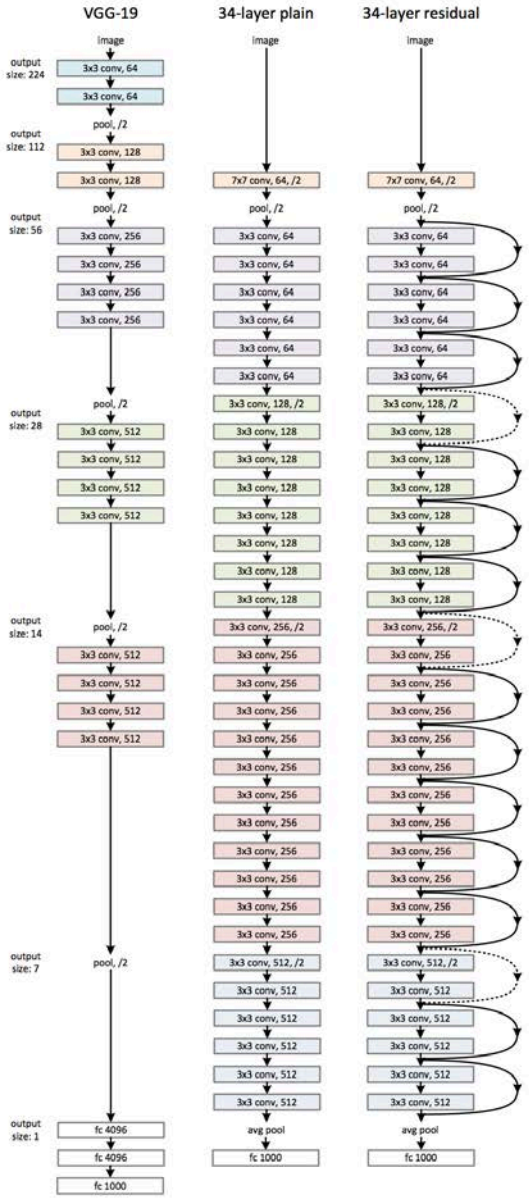


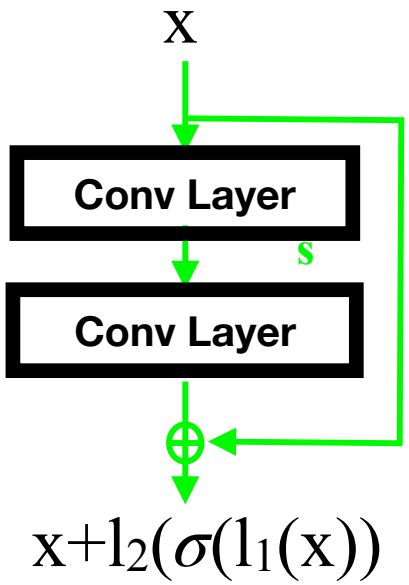
Input: Depth image.

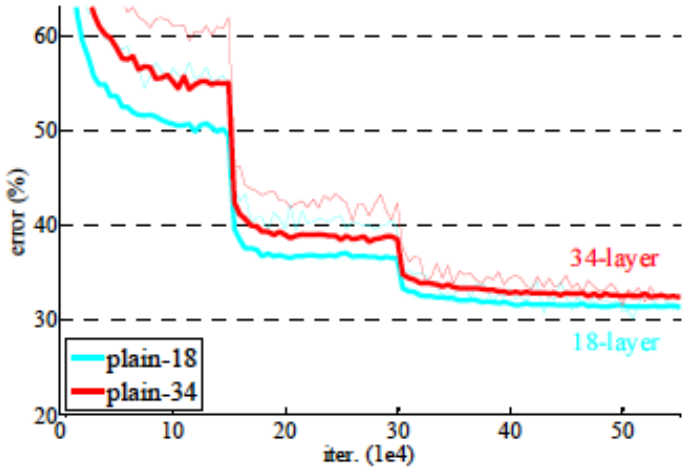
Output: 3D pose vector.

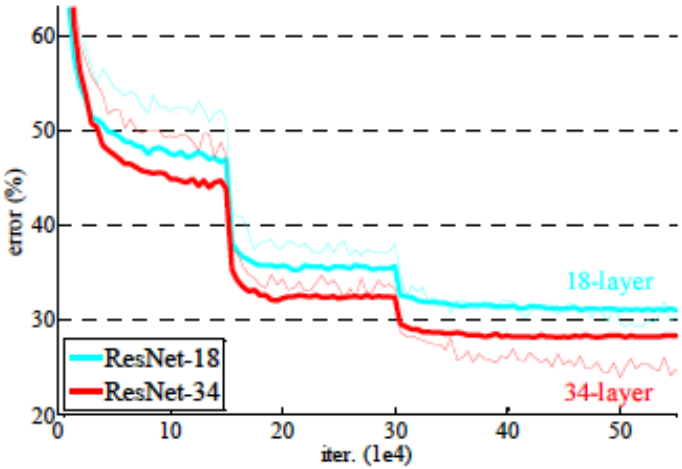
Oberweger et al. , ICCV'15

Deeper is Better





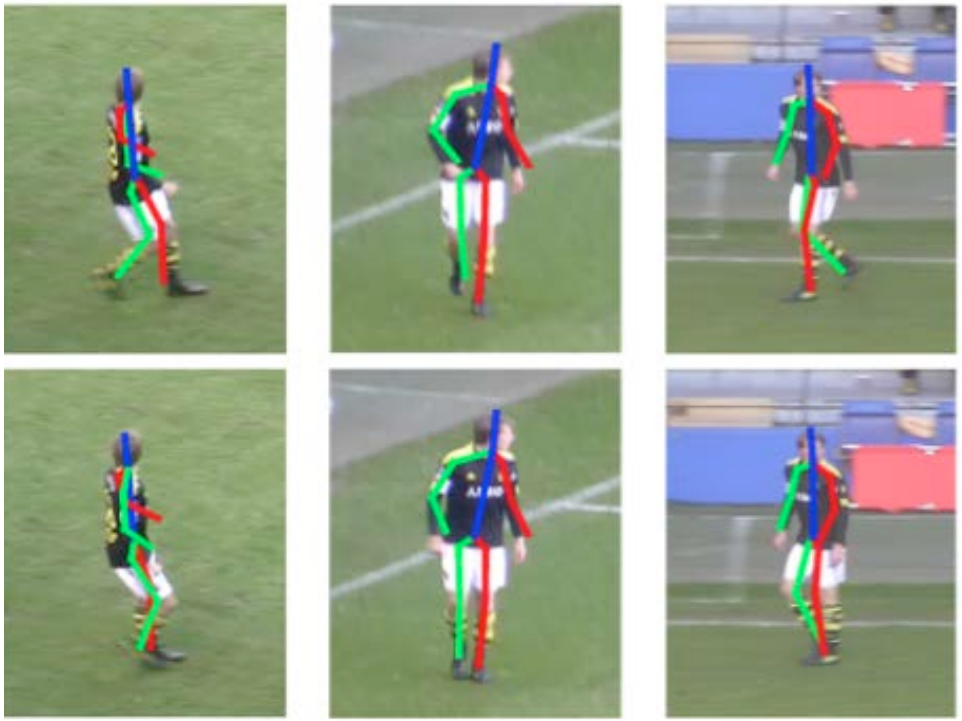




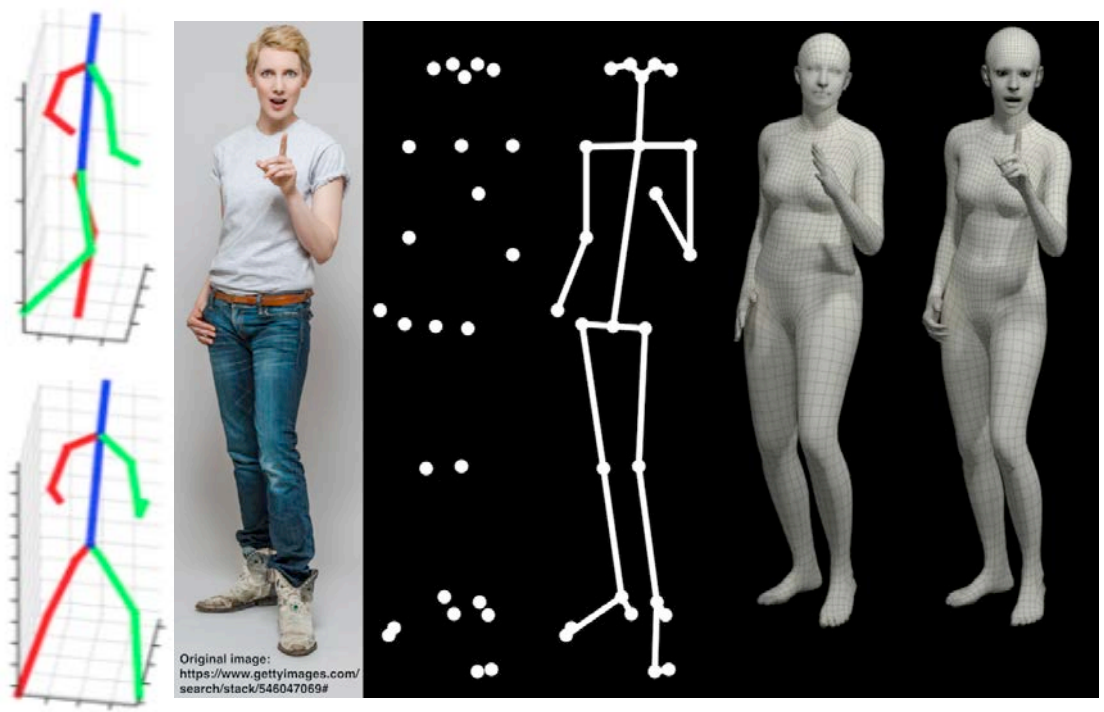
In general, the more ResNet layers, the better the results.

He et al. , CVPR'16

Human Body Pose Estimation



Tekin et al. , BMVC'16

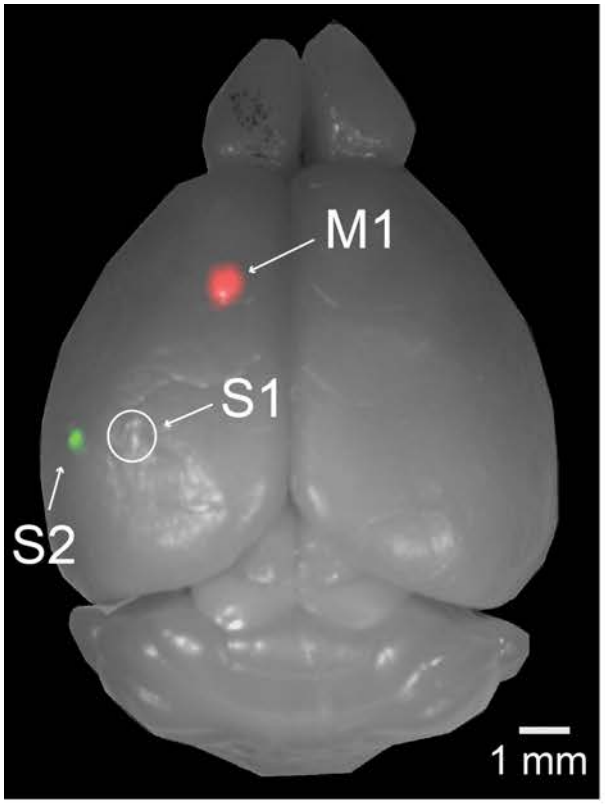


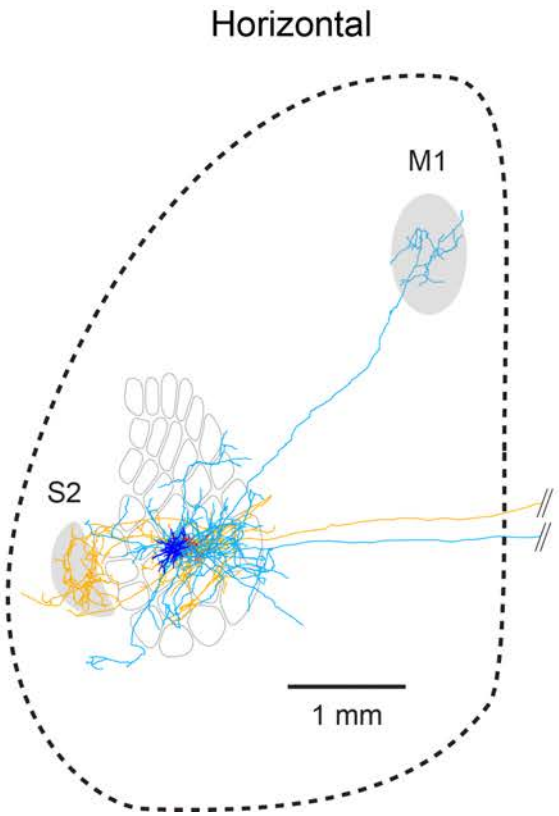
Pavlakos et al., CVPR'19

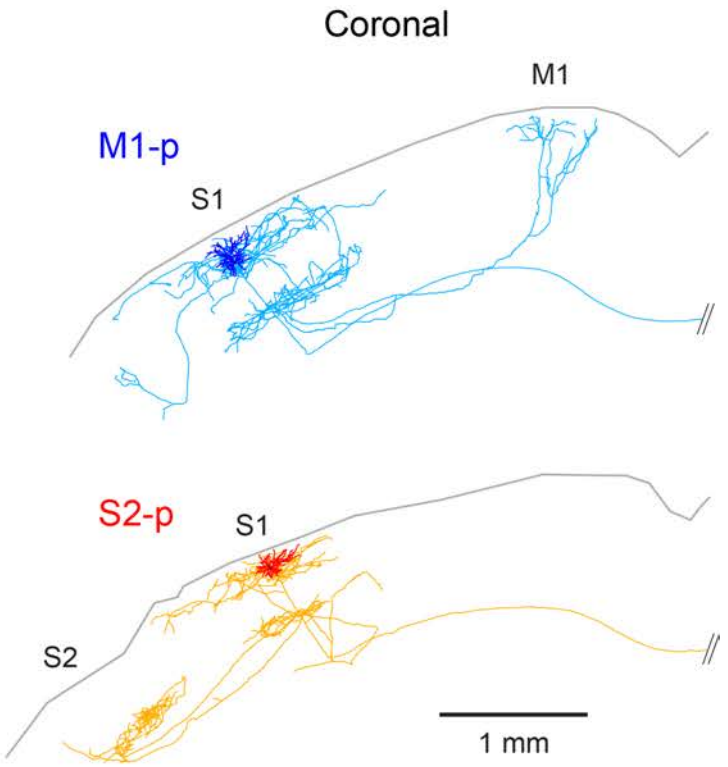
Now working on:

- Multiple people in crowded scenes.
- People wearing loose clothing.

Connectomics



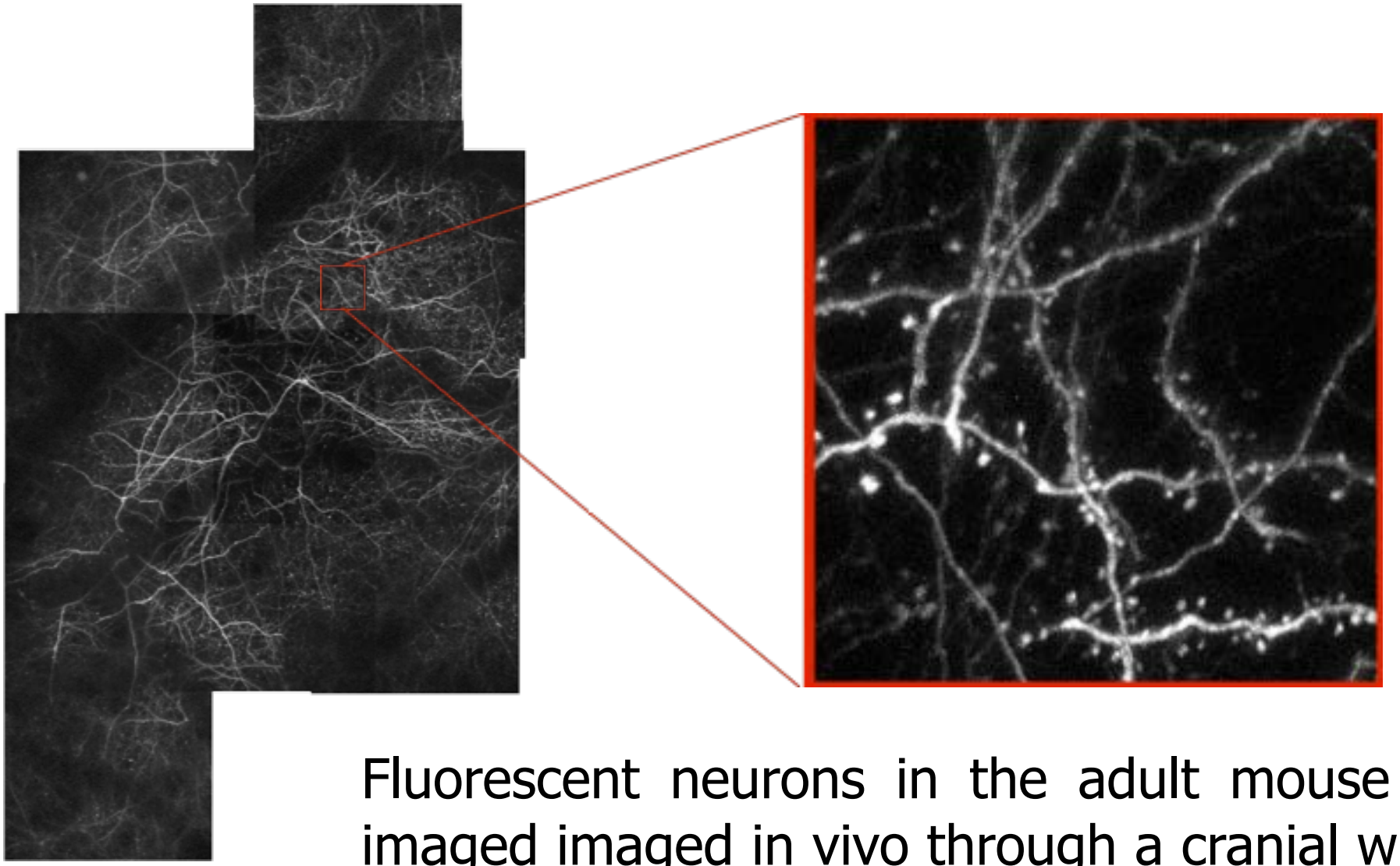




- Building the wiring diagram of the brain.
- Finding long range connections.

—> One step towards understanding how it works.

Dendrites and Axons



Fluorescent neurons in the adult mouse brain imaged imaged in vivo through a cranial window using a 2-photon microscope.

Cartography

The road centerlines are used to plot routes.

Before Machine Learning

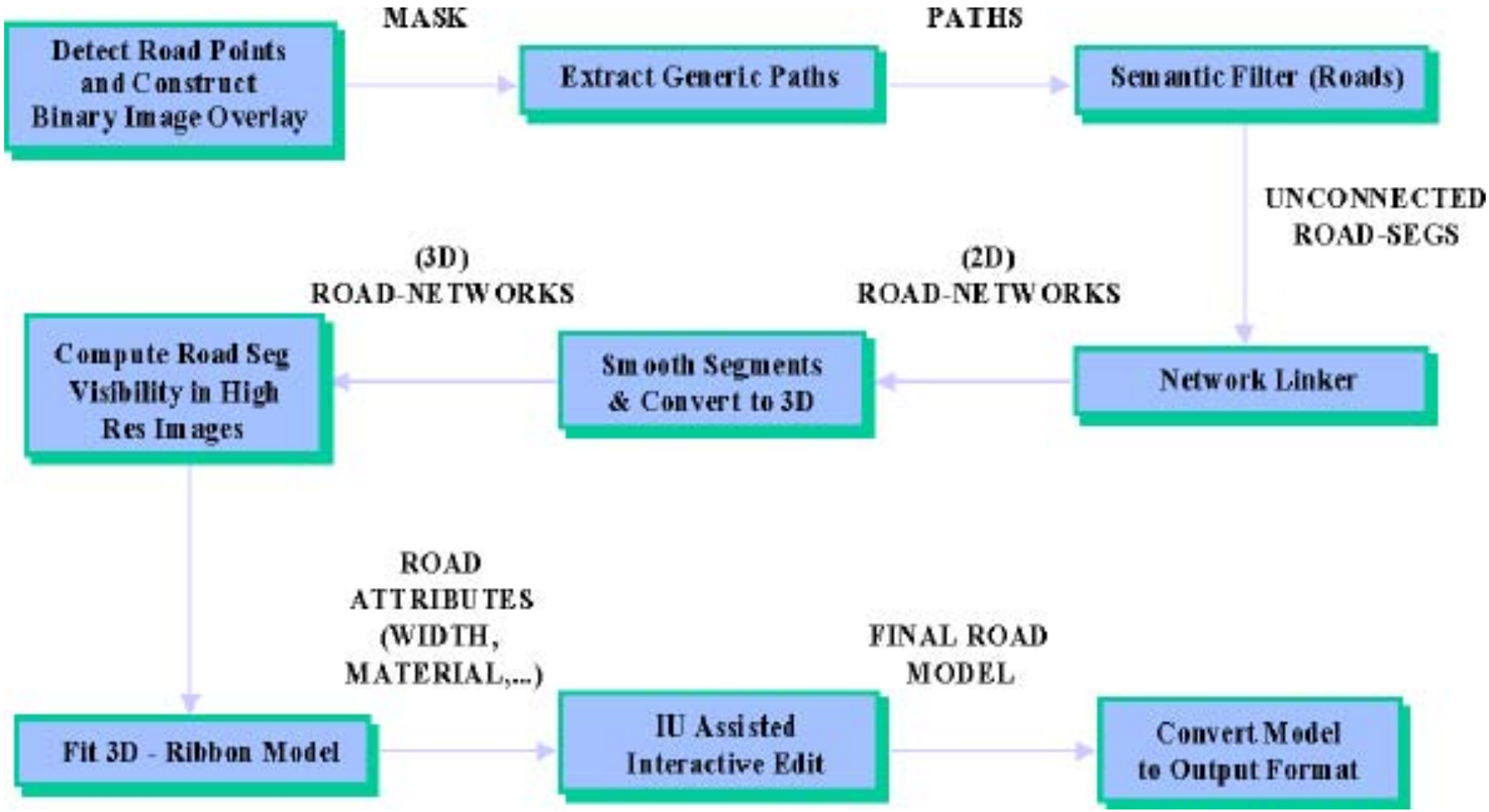
Detect road centerlines

Find generic paths

Apply semantic filter

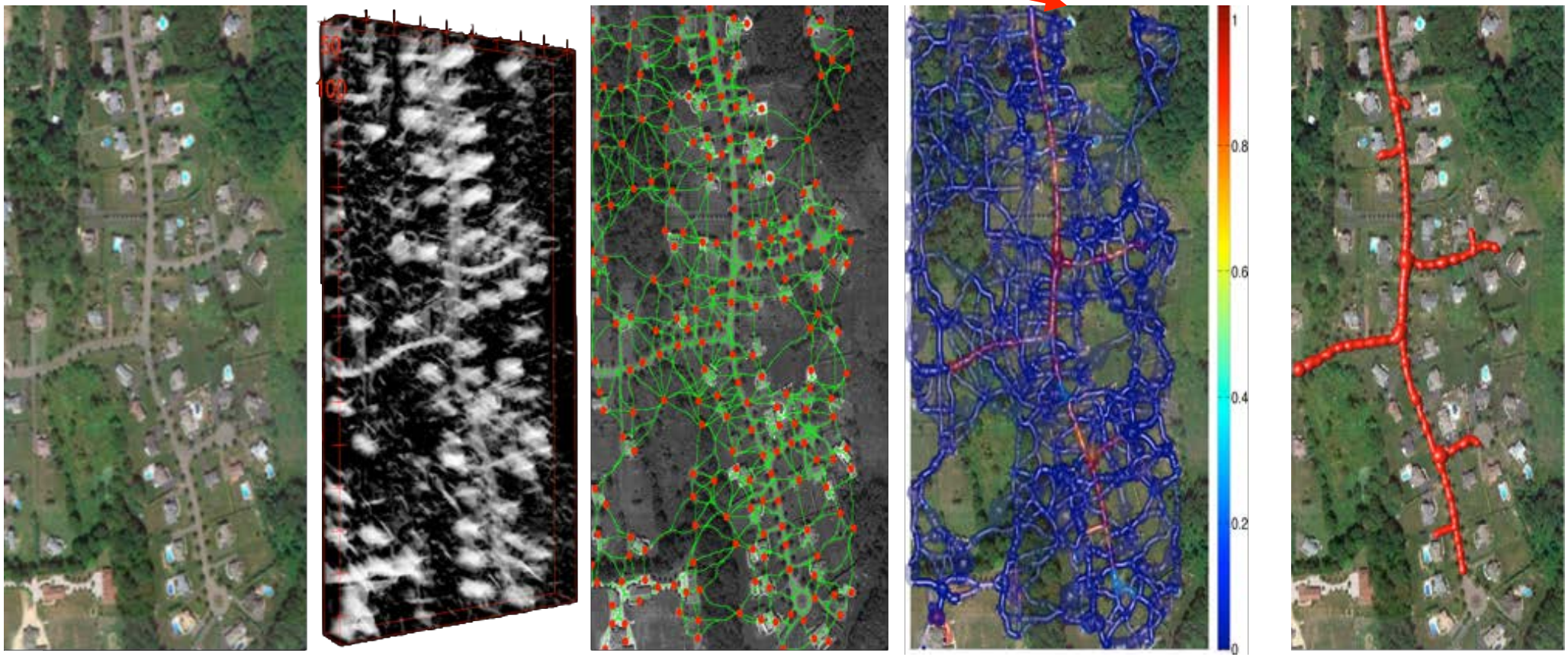
Fischler & Heller, 1998.

Boxology


Detect Road Points and Construct Binary Image Overlay
MASK
Extract Generic Paths
PATHS
Semantic Filter (Roads)
UNCONNECTED ROAD-SEGS
Network Linker
(2D) ROAD-NETWORKS
Smooth Segments & Convert to 3D
(3D) ROAD-NETWORKS
Compute Road Seg Visibility in High Res Images
ROAD ATTRIBUTES (WIDTH, MATERIAL,...)
Fit 3D - Ribbon Model
IU Assisted Interactive Edit
FINAL ROAD MODEL
Convert Model to Output Format

After Machine Learning

Train a classifier to do this.

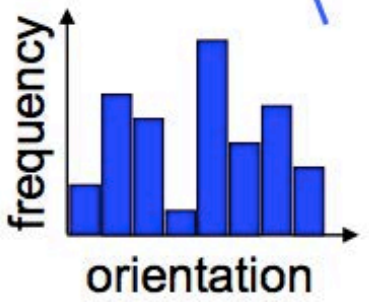


To train the classifier, we must associate a feature vector to each path and they all must be of the same dimension.

Turetken et al., PAMI'16.

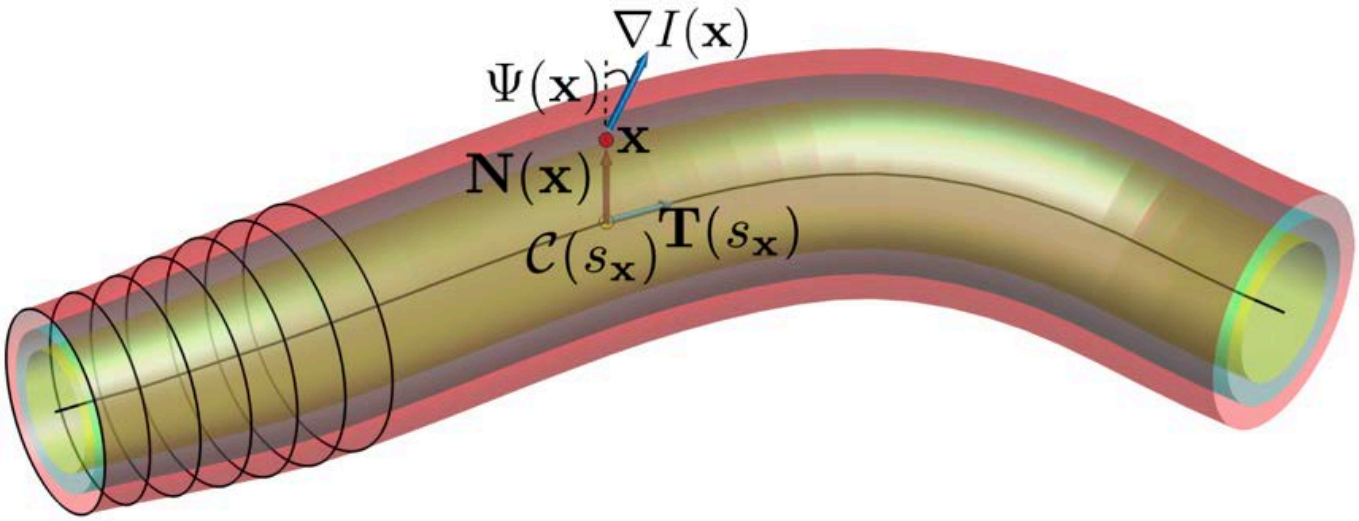
Histogram of Oriented Gradients

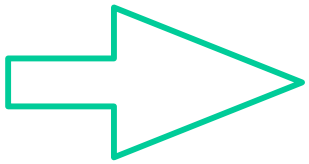
- tile window into 8 x 8 pixel cells
- each cell represented by HOG



Feature vector dimension = 16 x 8 (for tiling) x 8 (orientations) = 1024

Histogram of Gradient Deviations

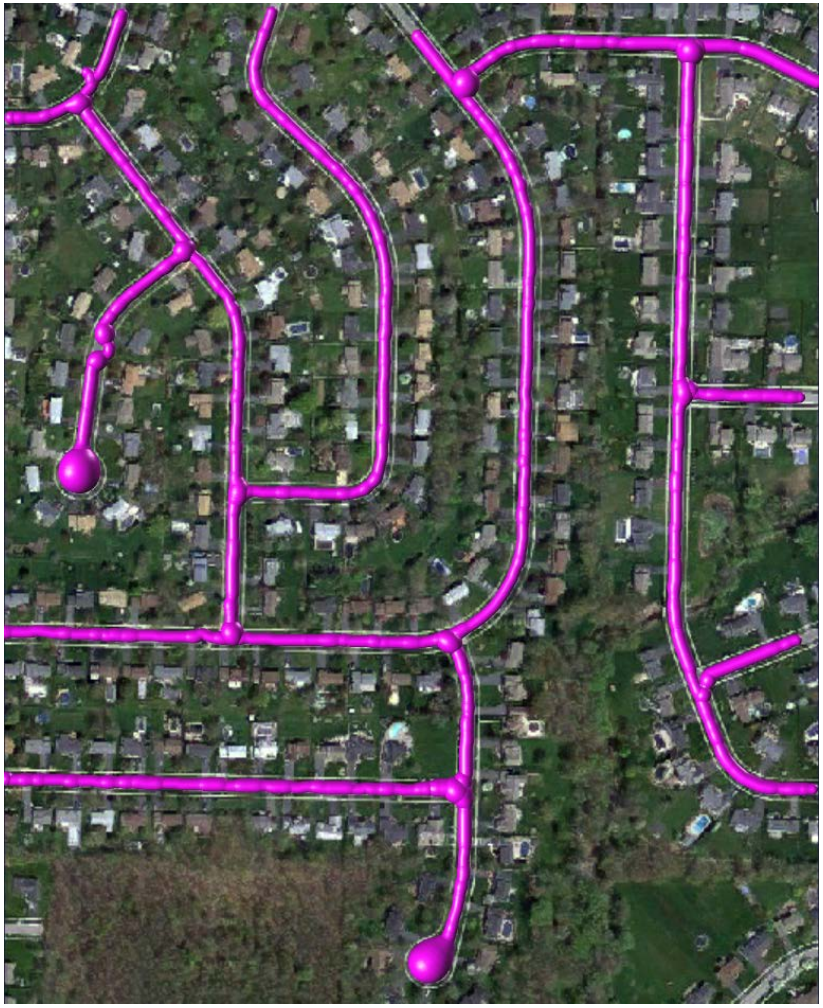




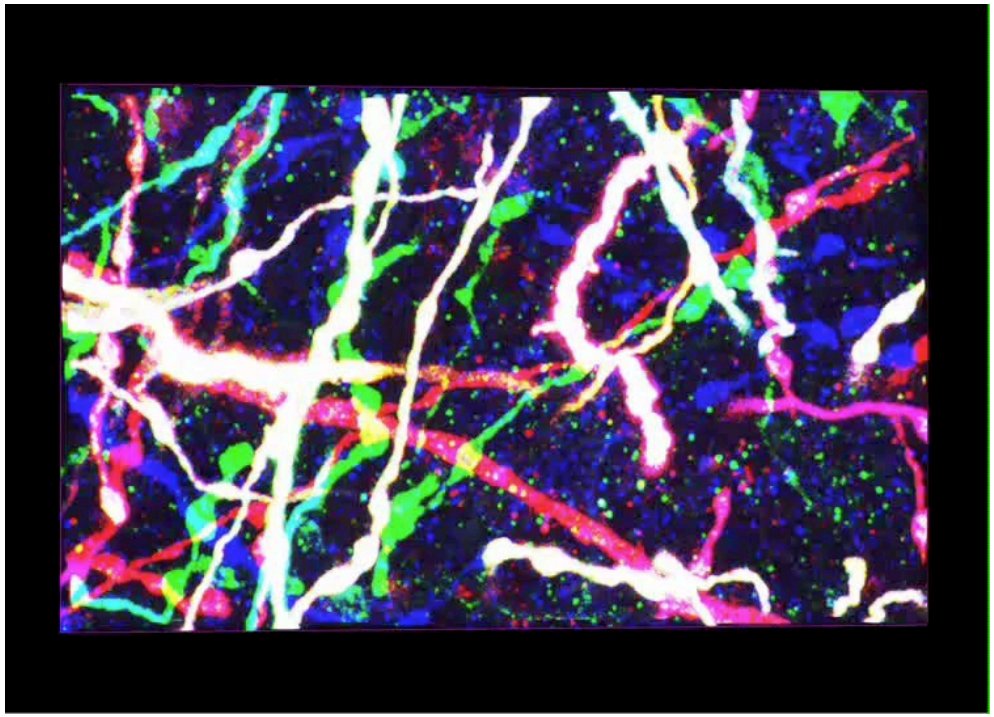
$$\Psi(\mathbf{x}) = \begin{cases} \text{angle}(\nabla I(\mathbf{x}), \mathbf{N}(\mathbf{x})) \text{ , if } \|\mathbf{x} - \mathcal{C}(s_\mathbf{x})\| > \varepsilon \\ \text{angle}(\nabla I(\mathbf{x}), \mathbf{\Pi}(\mathbf{x})) \text{ , otherwise,} \end{cases}$$

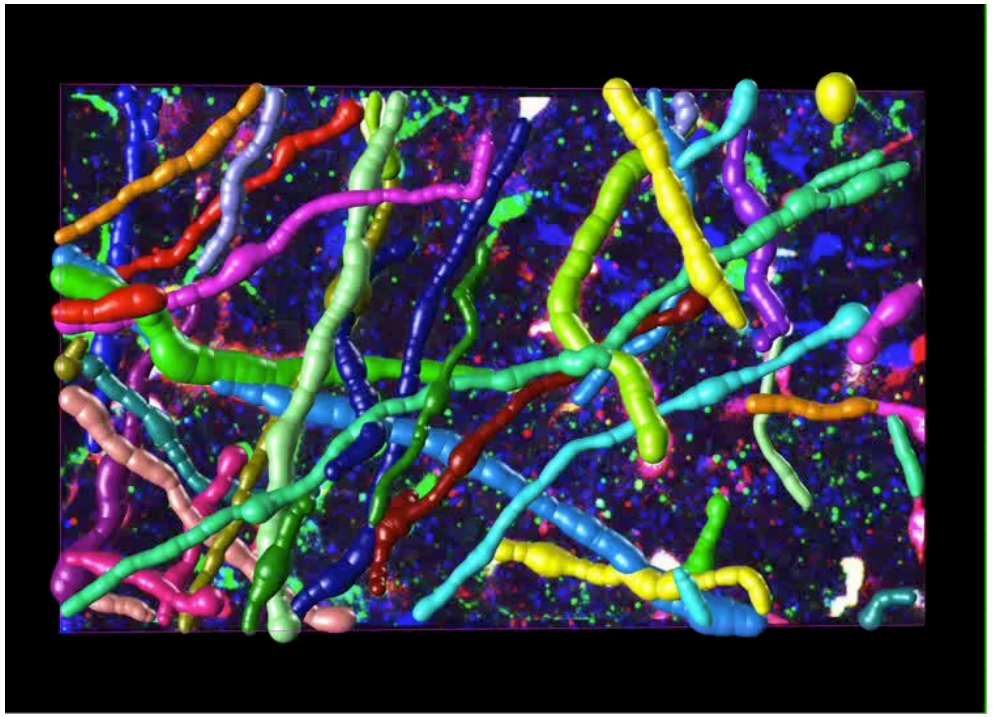
—> One histogram per radius interval plus four geometric features (curvature, tortuosity, ….).

Roads

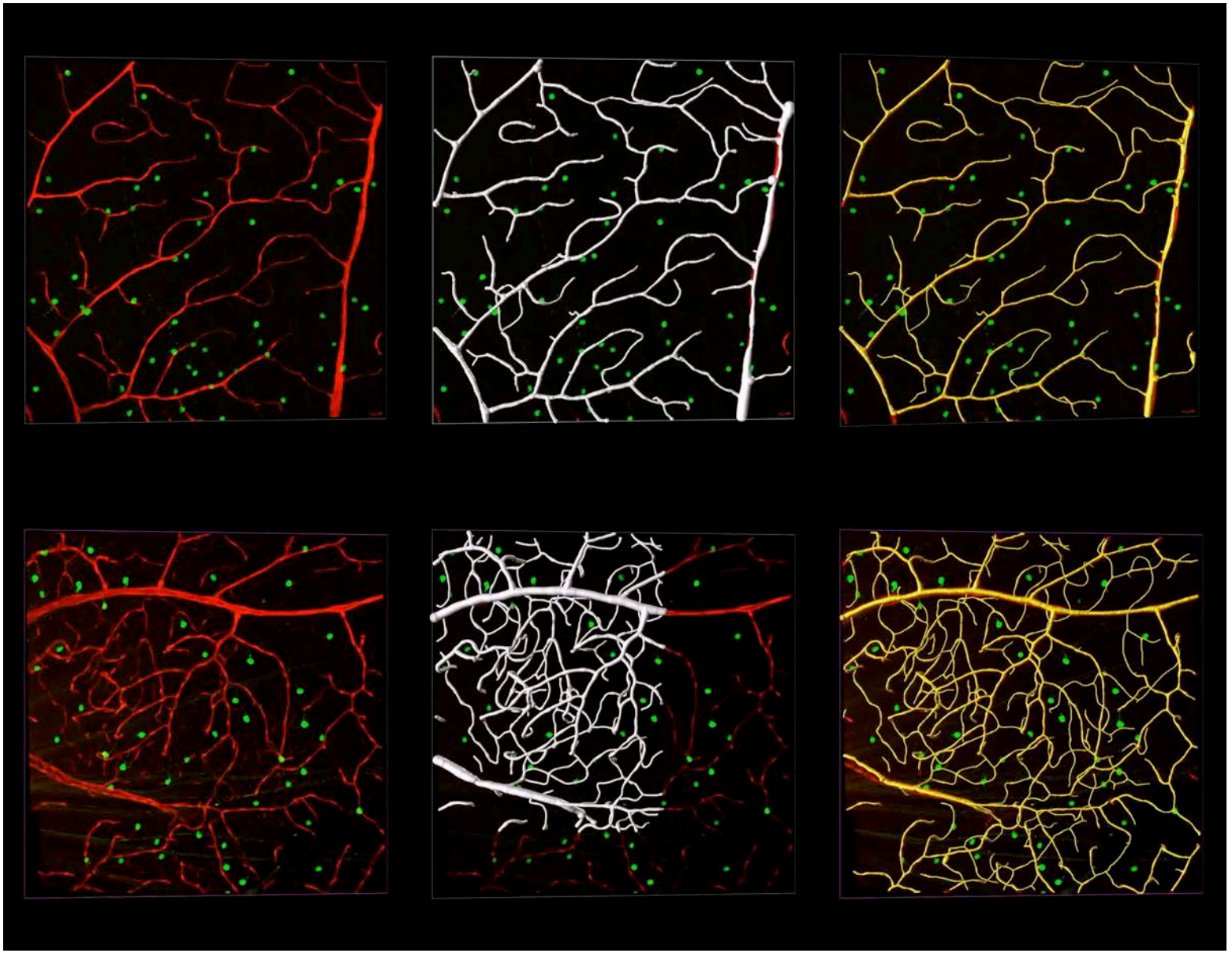


Brainbow Images





Blood Vessels



Deep Learning Tsunami

AlexNet 2012

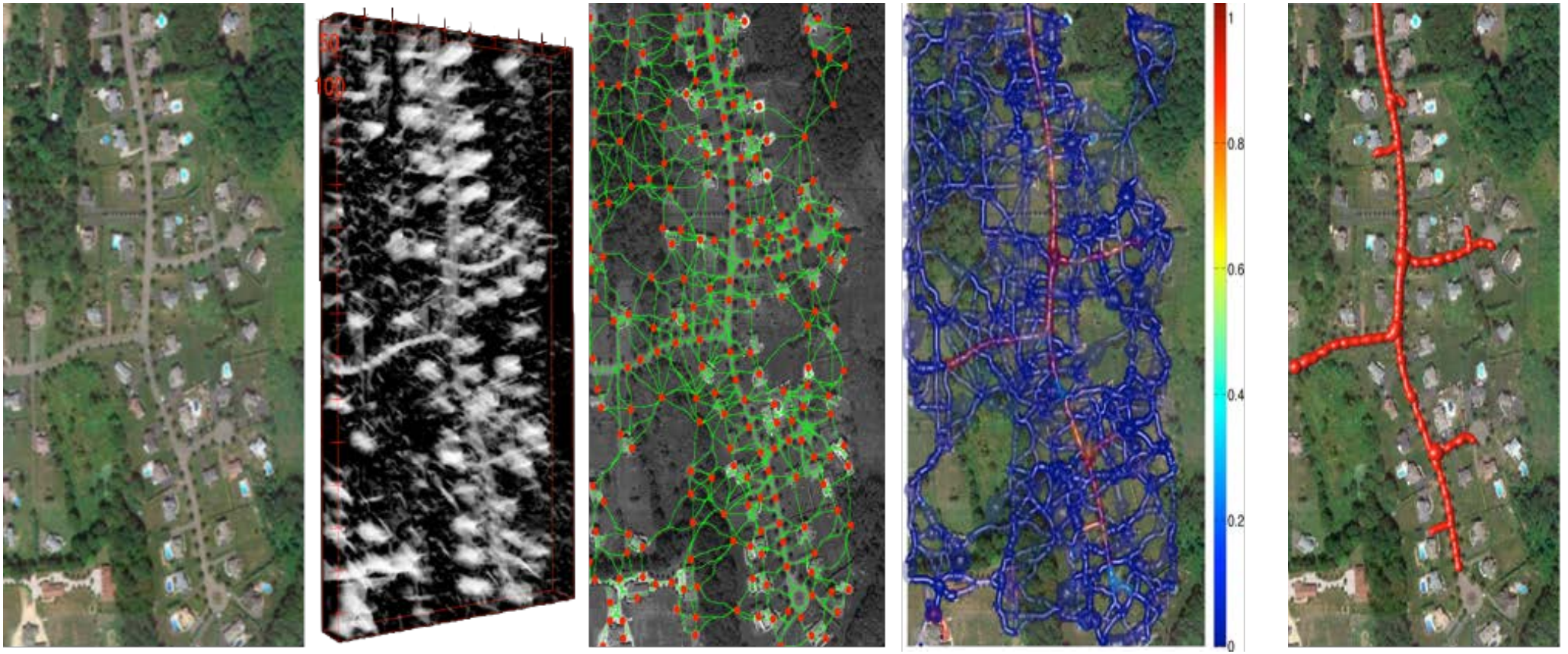
The end of computer science as we know it

or .....

An opportunity to revisit and improve the pipeline:

- Reformulate individual components in terms of CNNs.
- Make them consistent with each other.

Before Deep Learning

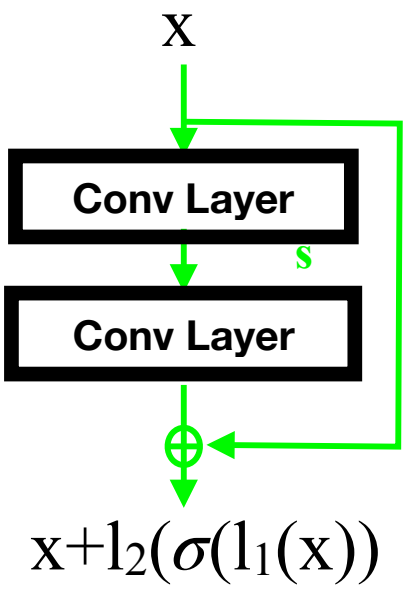


- Machine learning enables the **same** algorithm to work in many different contexts but requires hand-designed features.
- However, computing the tubularity and classifying the paths are closely related tasks. They should not be treated separately.

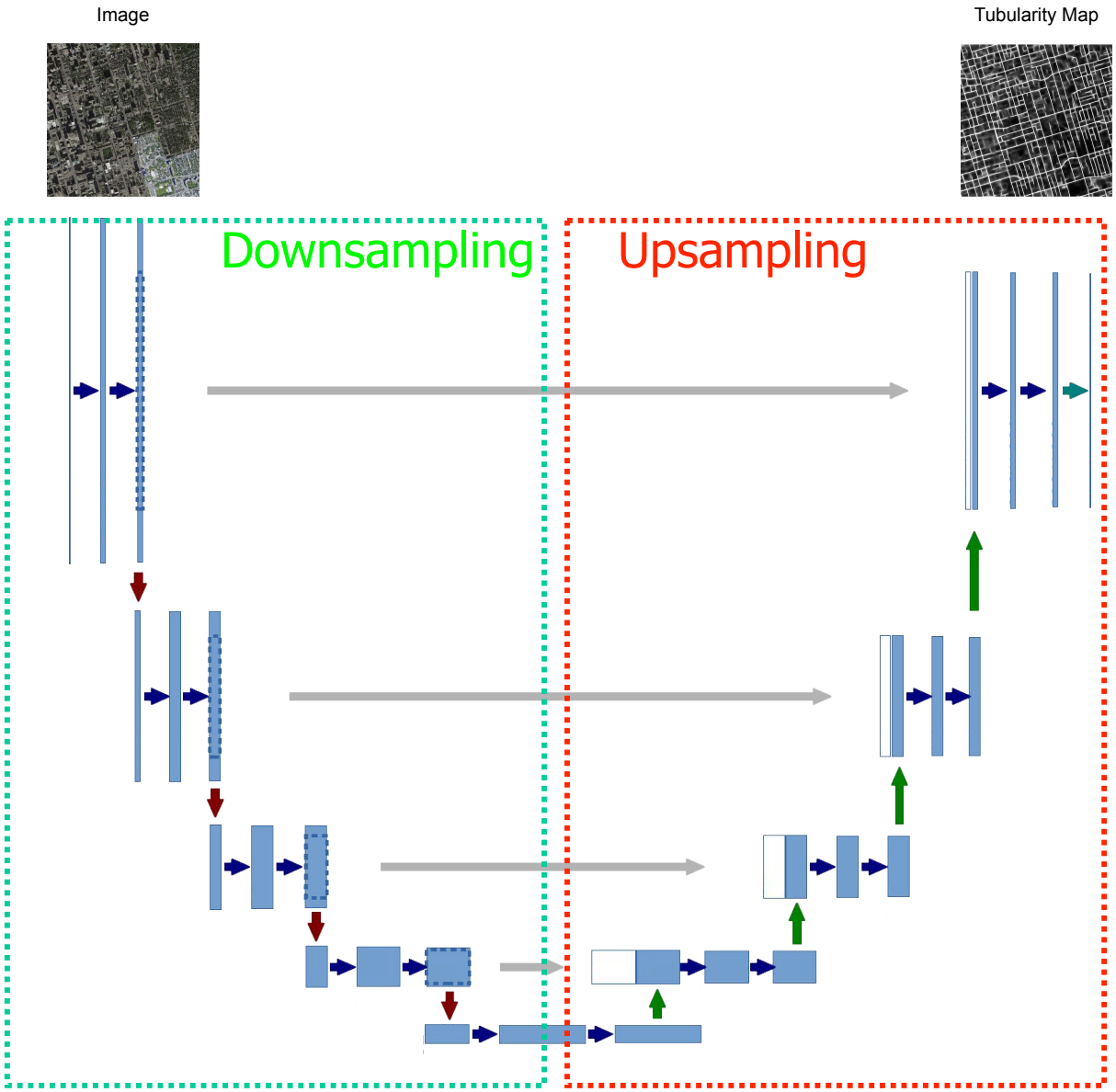
—> Can we use Deep Learning to account for this?

Turetken et al., PAMI'16

From ResNet to U-Net

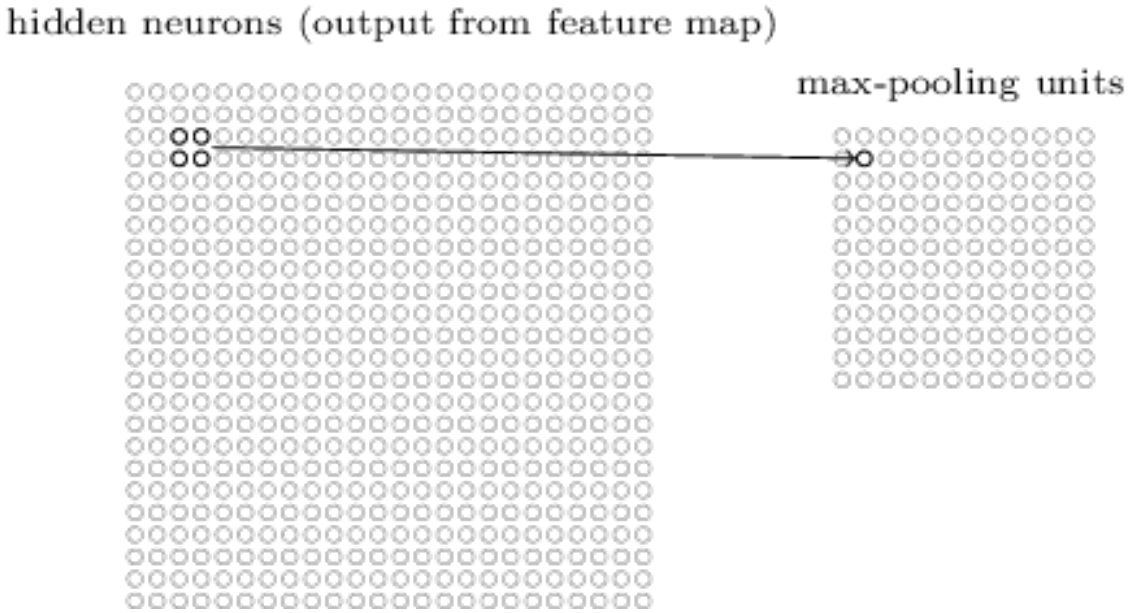


ResNet block



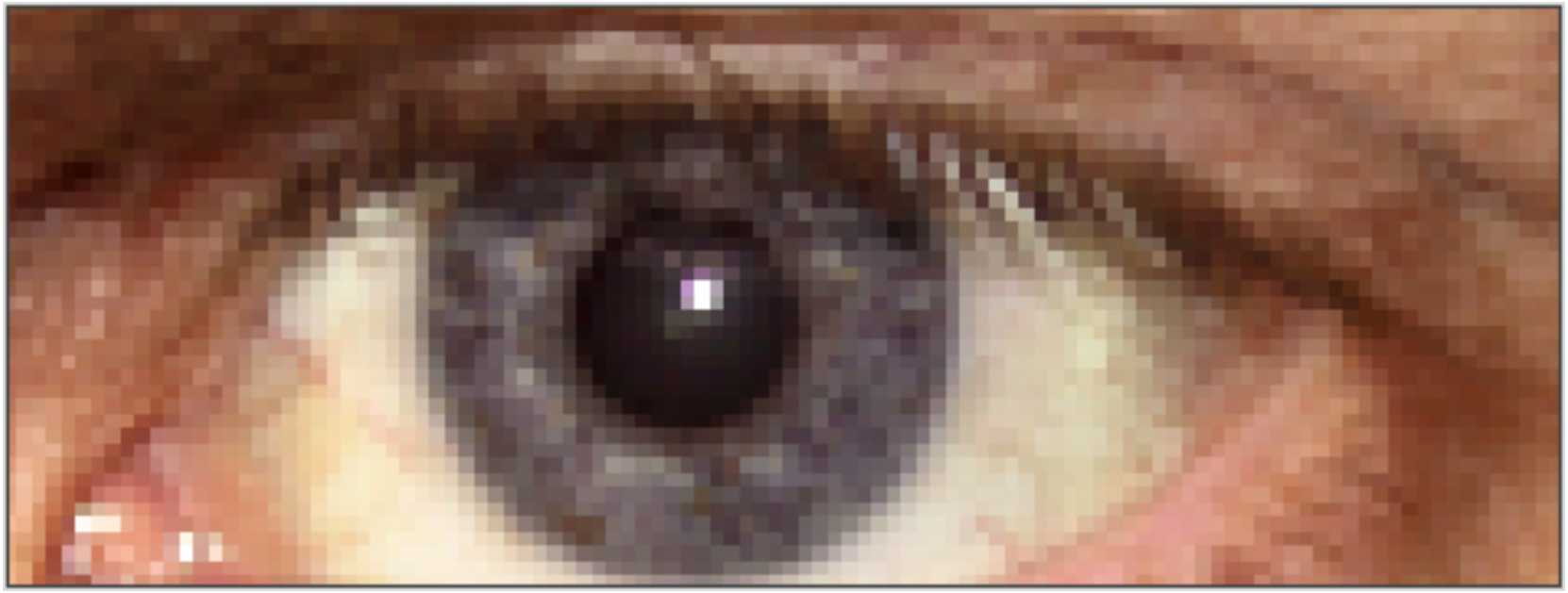
U-Net

Reminder: Downsampling by Pooling



- Reduces the number of inputs by replacing all activations in a neighborhood by a single one.
- Can it be reversed?

Upsampling by Duplication



i1	i2
i3	i4

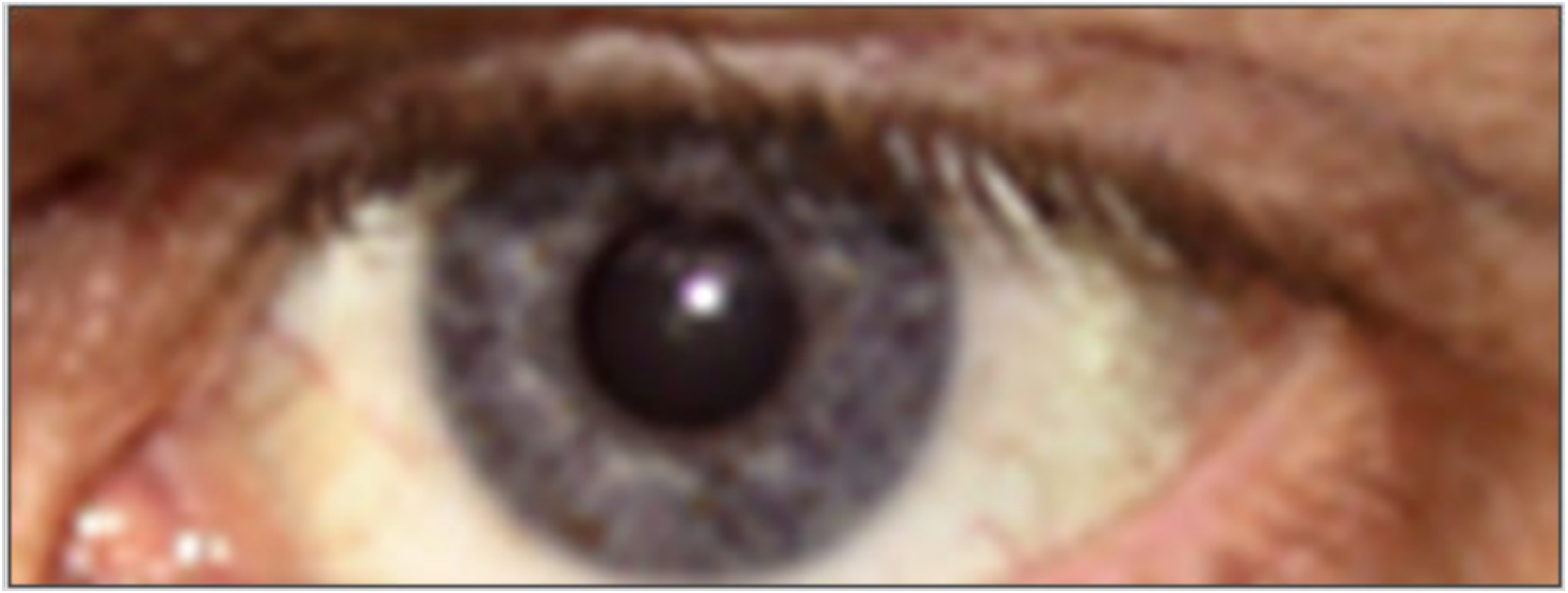
i1	i1	i2
i1	i1	i2
i3	i3	i4

Upsampling by Interpolation

i1	i2
i3	i4

i1	i5=(i1+i2)/2	i2
i6=(i1+i3)/2	i9=(i1+i2+i3+i4)/4	i7=(i2+i4)/2
i3	i8=(i3+i4)/2	i4

Upsampling by Bilinear Interpolation

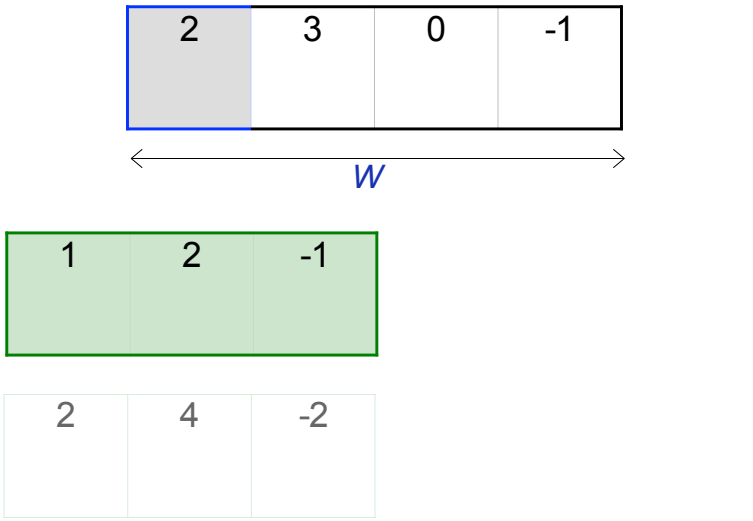


i1	i2
i3	i4

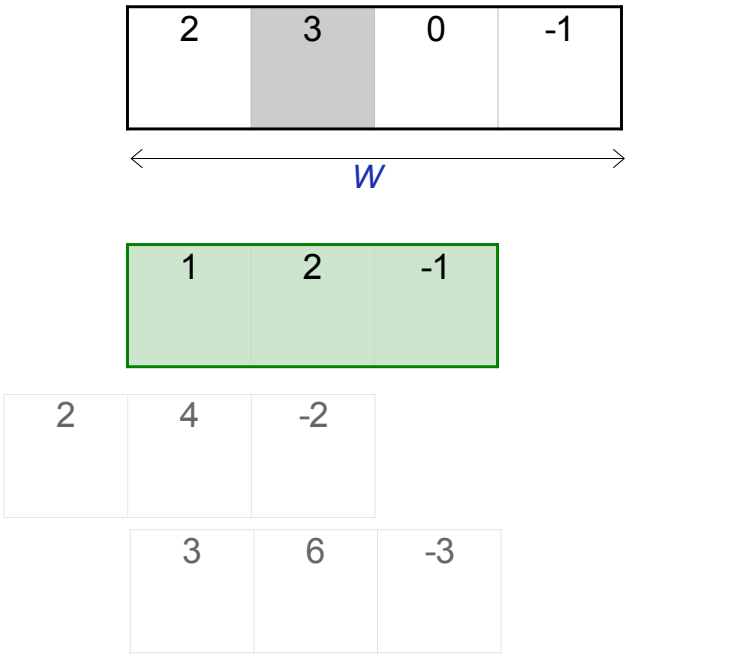
	I10=(i1 + i2 + . + .) / 4	
i1	i11=(i10+i1+i2+i9)/4	i2
	i9=(i1+i2+i3+i4)/4	
i3		i4

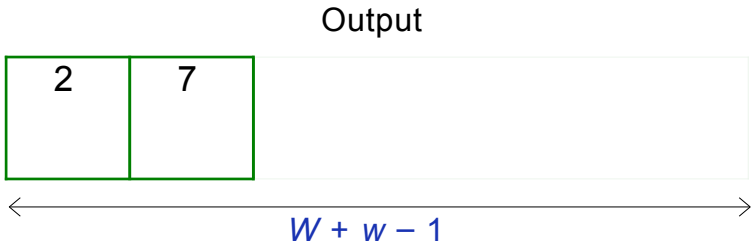
—> Each pixel is the average of 4 neighbors.

Upsampling by Transposed 1D Convolution

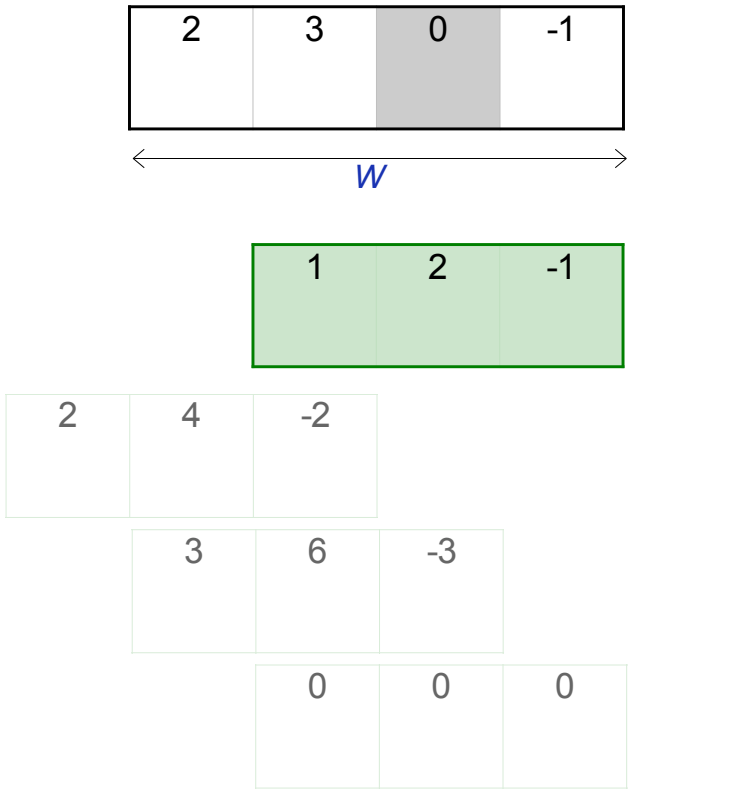


Upsampling by Transposed 1D Convolution

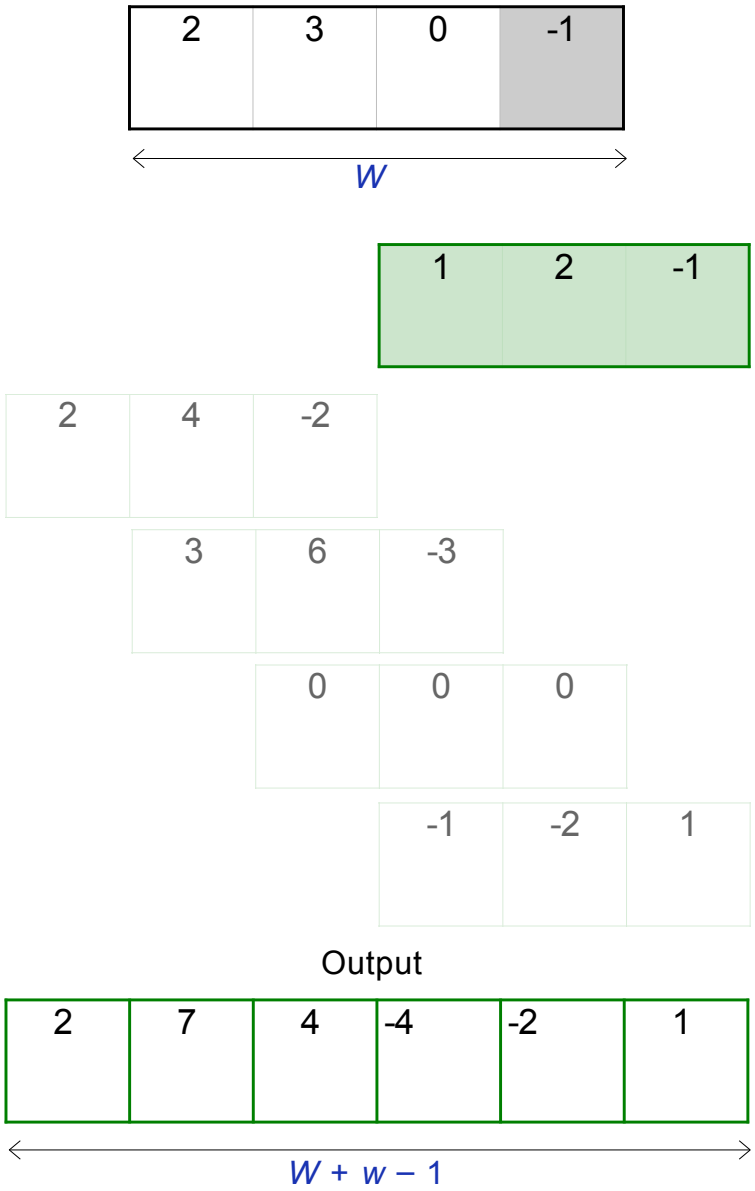




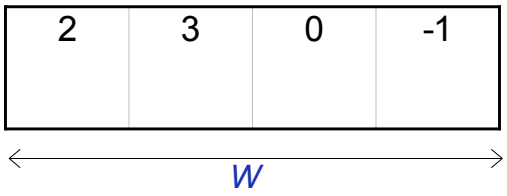
Transposed 1D Convolution

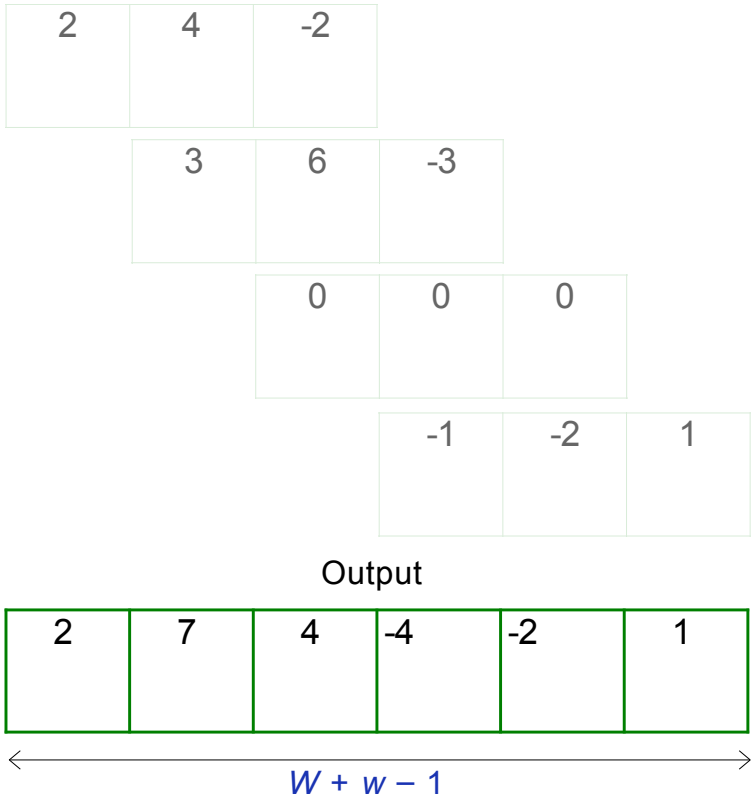


Transposed 1D Convolution

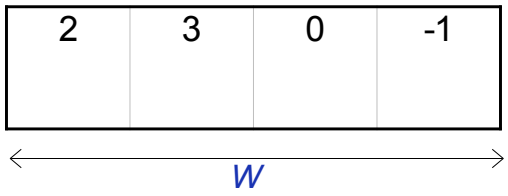


Transposed 1D Convolution

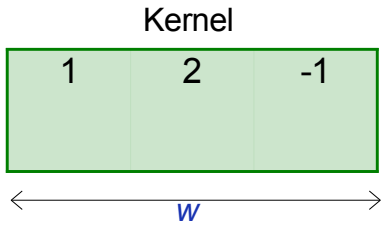


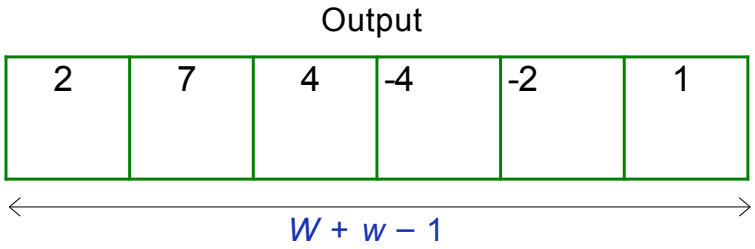


Transposed 1D Convolution

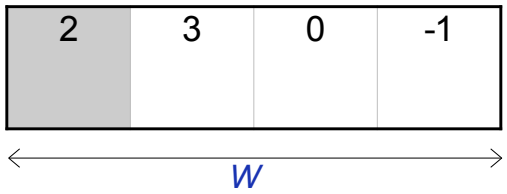


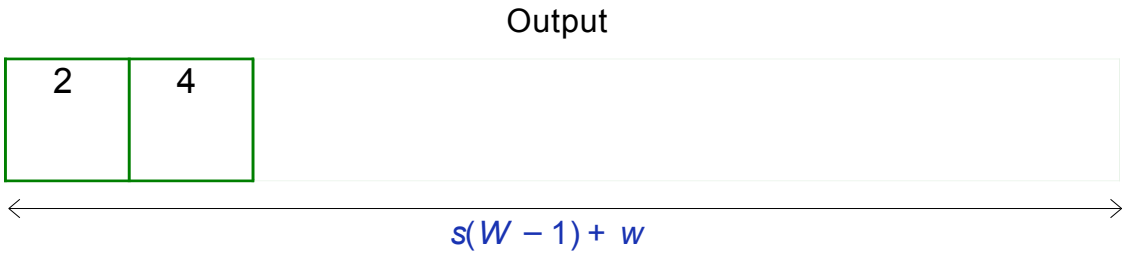
- The summations are performed in the vertical direction instead of the horizontal one.
- If we wrote this in terms of a fully connected layer, this would amount to transposing the weight matrix.
- Can be extended to 2D layers.



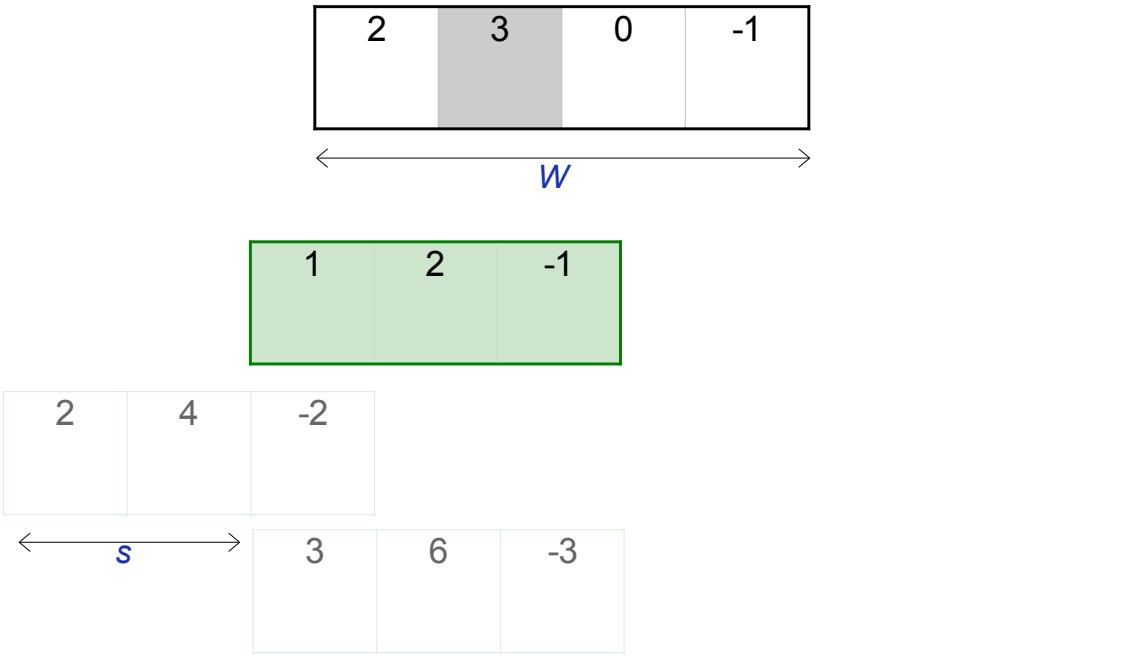


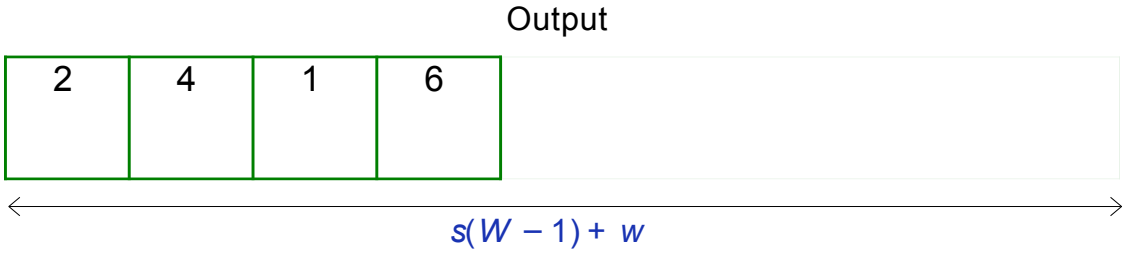
Introducing a Stride Parameter



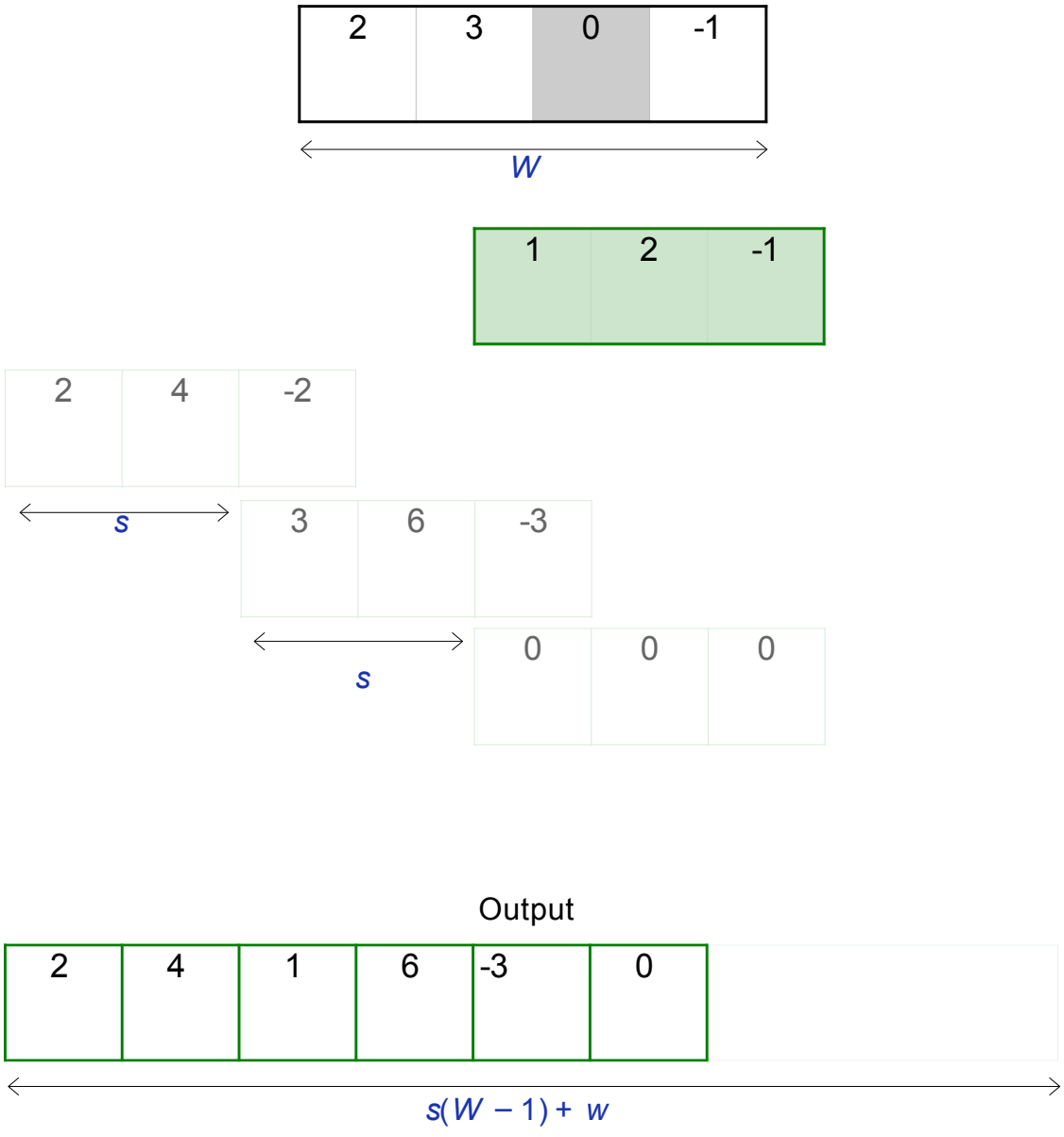


Introducing a Stride Parameter

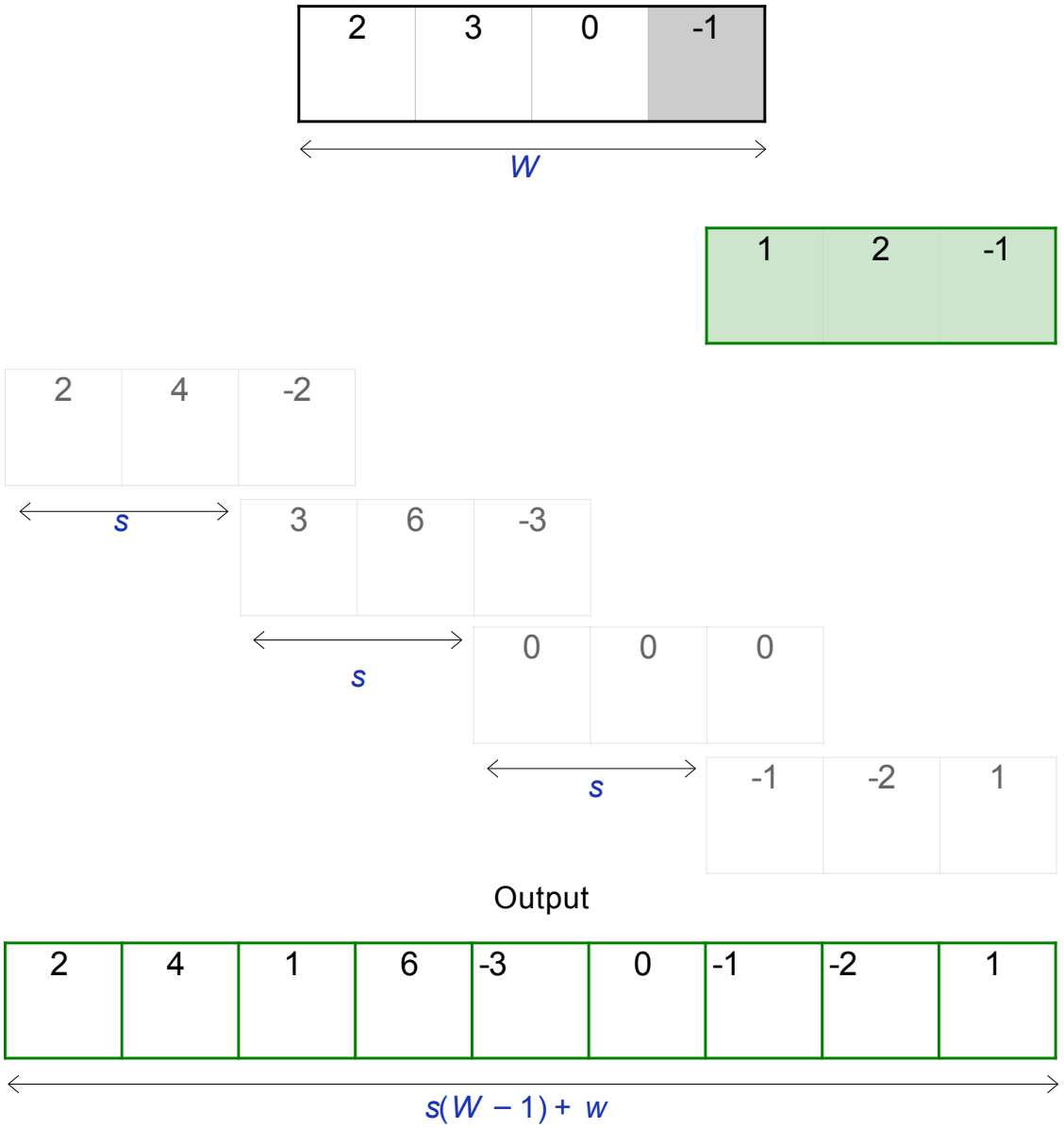




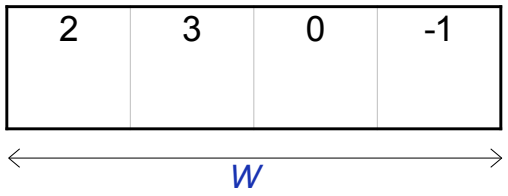
Introducing a Stride Parameter

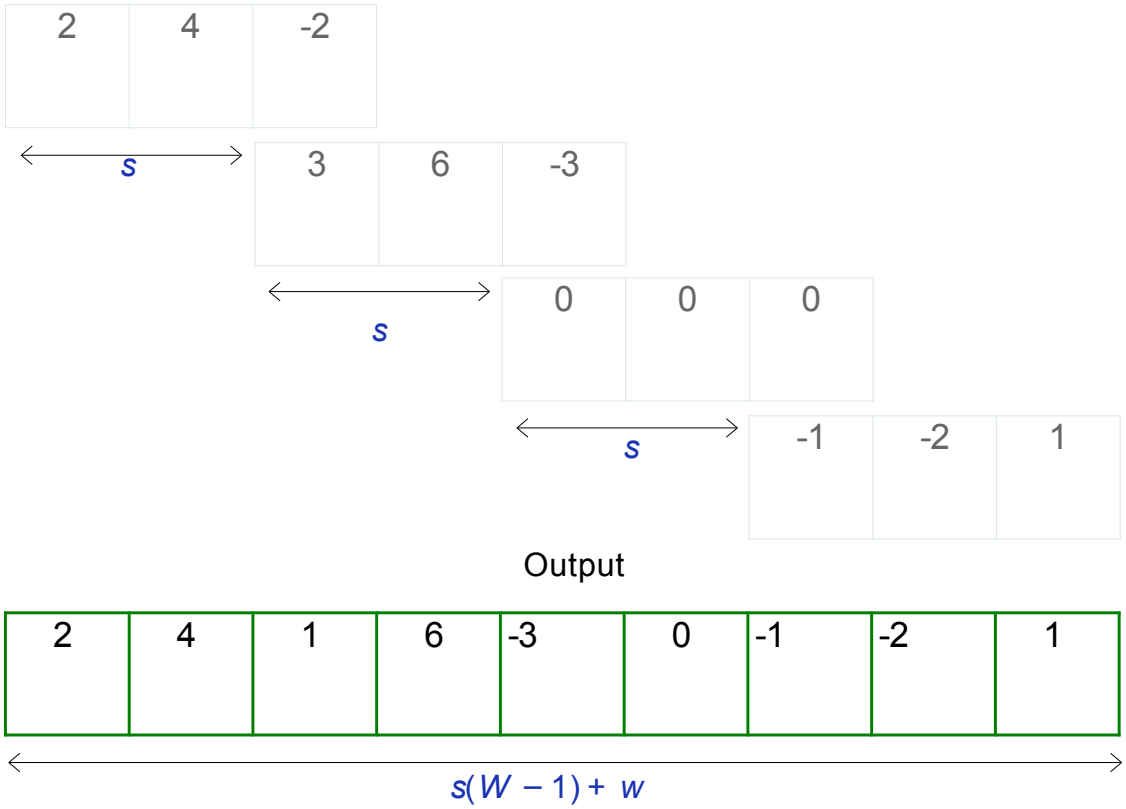


Introducing a Stride Parameter

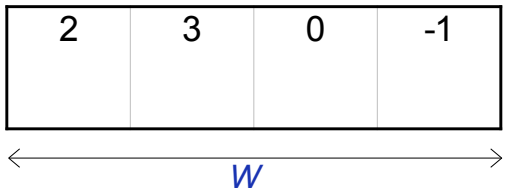


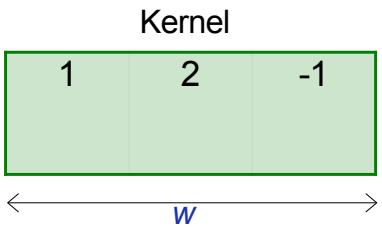
Introducing a Stride Parameter

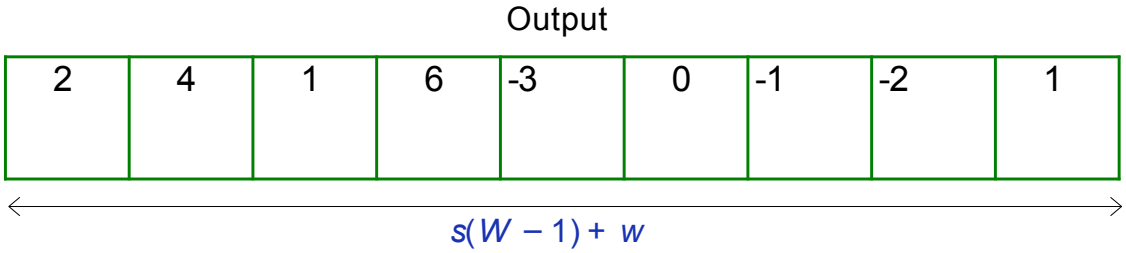




Introducing a Stride Parameter



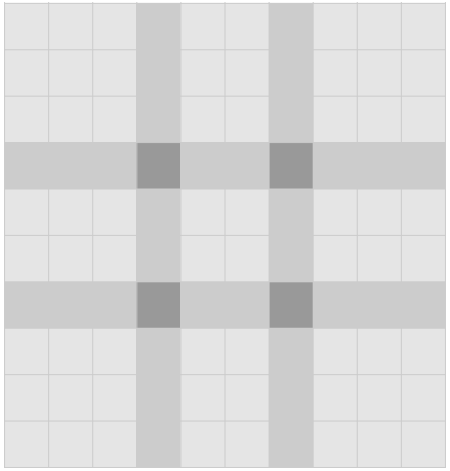




Shrinking and Reexpanding

- The composition of a convolution and a transposed convolution with the same parameters keep the signal size roughly unchanged.

- Transposed convolutions may create grid-structure artifacts because generated pixels are not all covered similarly.

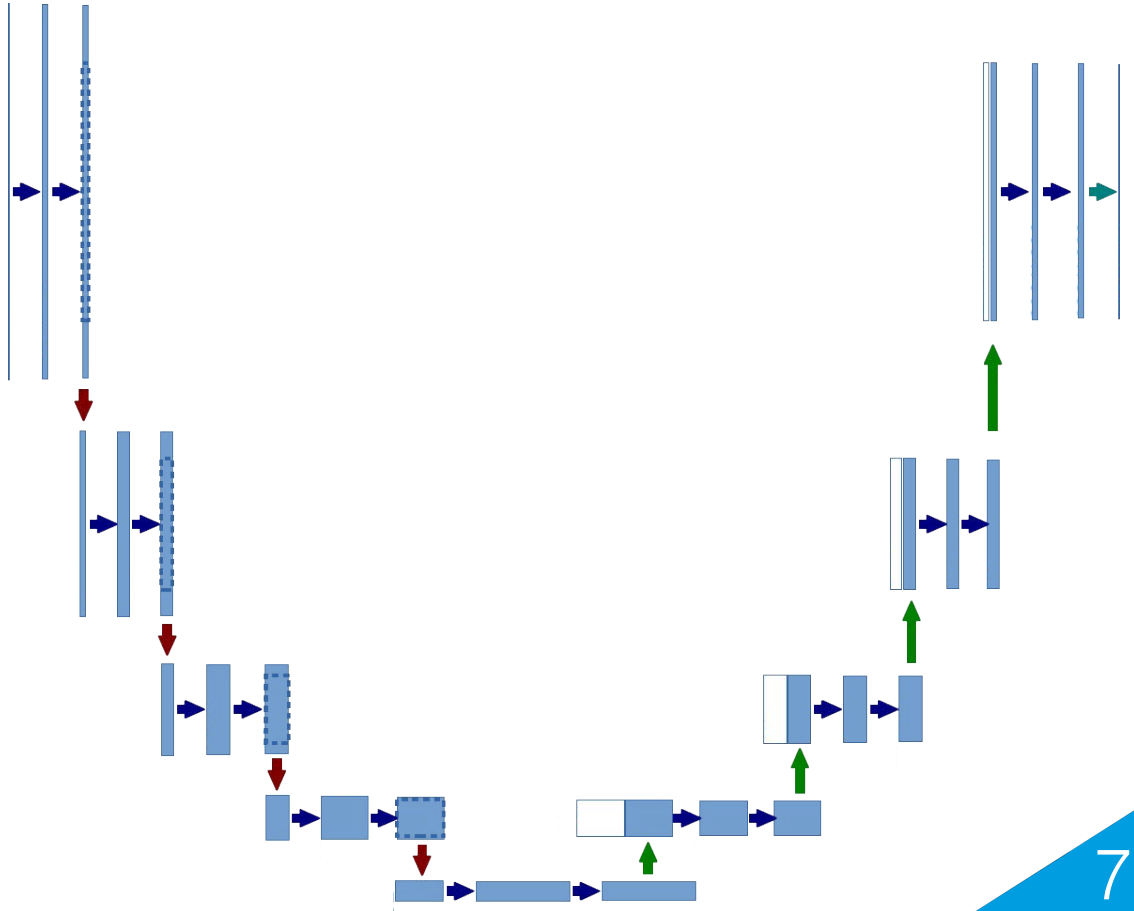


4×4 kernel and stride 3

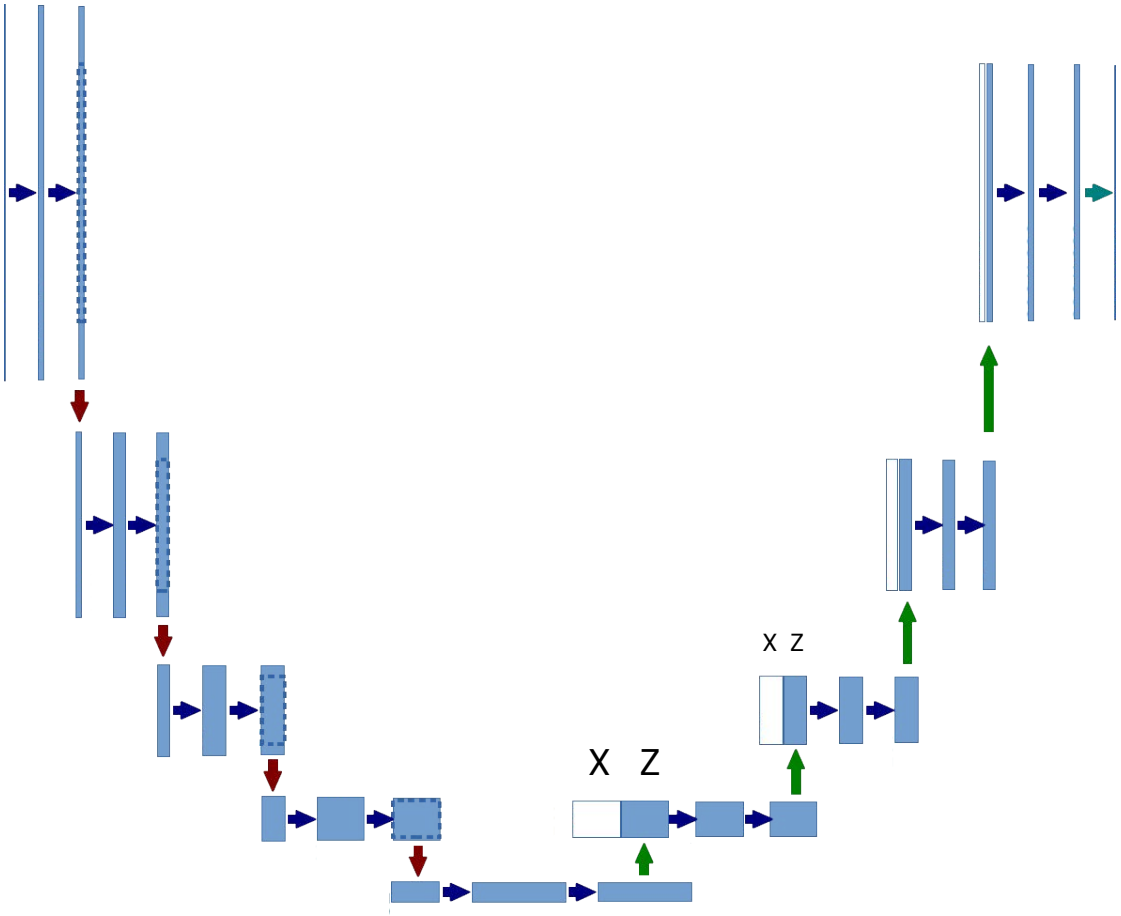
- Smoothing layers are required to prevent this.

Autoencoder in Python

```
class Autoencoder(nn.Module):

  def __init__(nChannel=10,nHidden=50):
      self.convE1 = nn.Conv2d(1, nChannel, kernel_size=5,padding=2)
      self.convE2 = nn.Conv2d(nc, nChannel, kernel_size=5,padding=2)
      self.convD1 = nn.ConvTranspose2d(nChannel,nChannel,kernel_size=4,stride=2,padding=1)
      self.convD2 = nn.ConvTranspose2d(nChannel,1 ,kernel_size=4,stride=2,padding=1)

  def encode(self,x):
      x = self.convE1(x)
      x = sigma(F.max_pool2d(x,2))
      x = self.convE2(x)
      x = sigma(F.max_pool2d(x,2))
      return x

 def decode(self, z):
      z= sigma(self.convD1(z)
      z= self.convD2(z)
      return z

  def forward(self,x):
      z = self.encode(x)
      return self.decode(z)
```



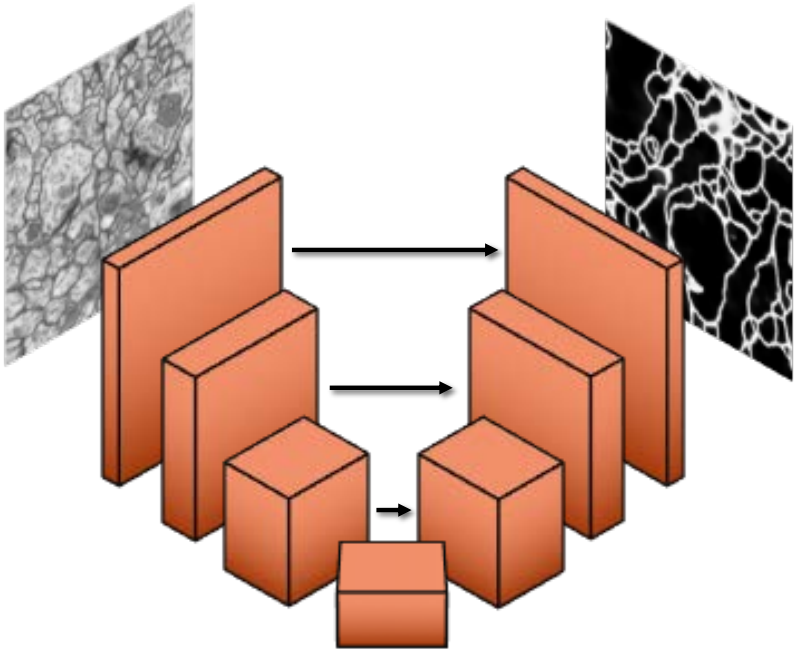
From Autoencoder to UNet




```
def encode(self,x):
        x = self.convE1(x)
        x = sigma(F.max_pool2d(x,2))
        z = self.convE2(x)
        z = sigma(F.max_pool2d(z,2))
        return z

def decode(self, z):
        z = sigma(self.convD1(z)
        z = self.convD2(z)
        return z

def forward(self,x):
        z = self.encode(x)
        return self.decode(z)
```

Concatenate z on the way up with the equivalent x on the way down, before running self.conD1 and self.convD2

Training UNet



Minimize

$$L_{BCE} = \frac{1}{N}\sum_{i=1}^{N} y_n log(\hat{y}_n) + (1 - y_n) log(\hat{y}_n)$$

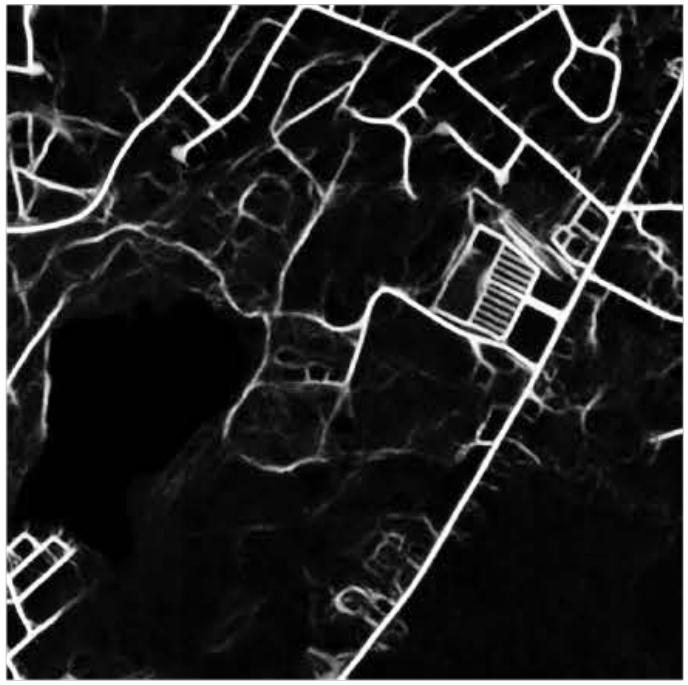
where

- $\hat{\mathbf{y}} = f_{\mathbf{w}}(\mathbf{x})$,
- $\mathbf{x}$ in an input image,
- $\mathbf{y}$ the corresponding ground truth.

Mosinska et al, CVPR'18.

Tubularity Map

Image



BCE Loss

Ground truth

Iterative Refinement

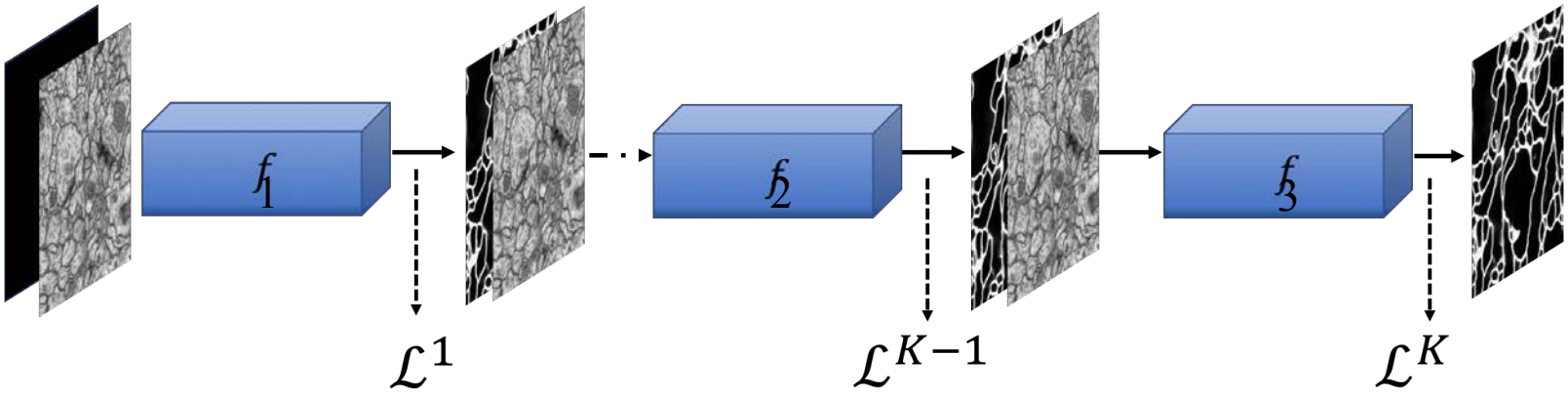
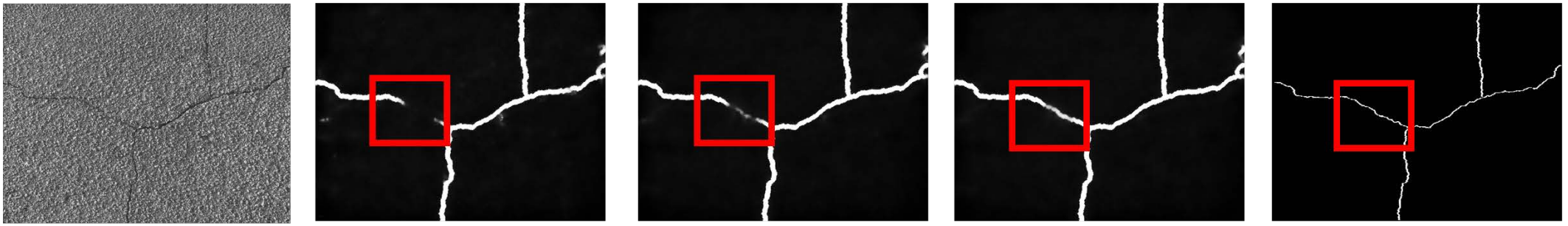
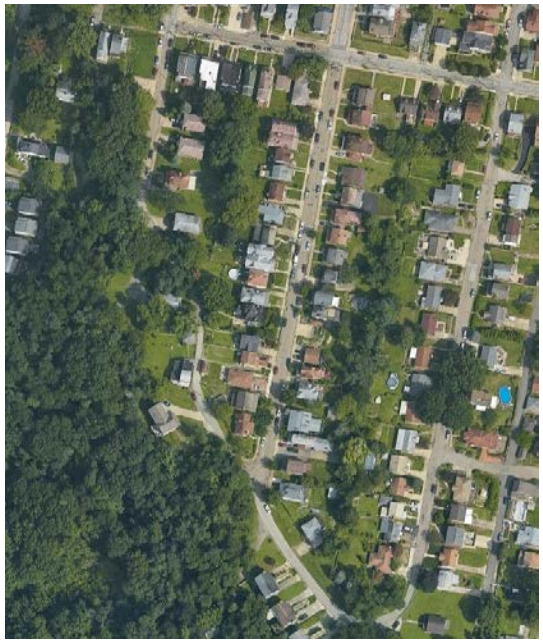


Image | Iter 1 | Iter 2 | Iter 3 | Ground truth

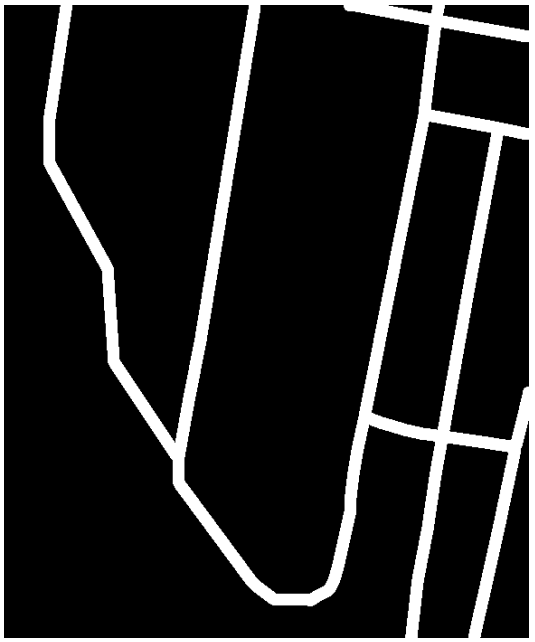


Use the same network to progressively refine the results keeping the number of parameters constant

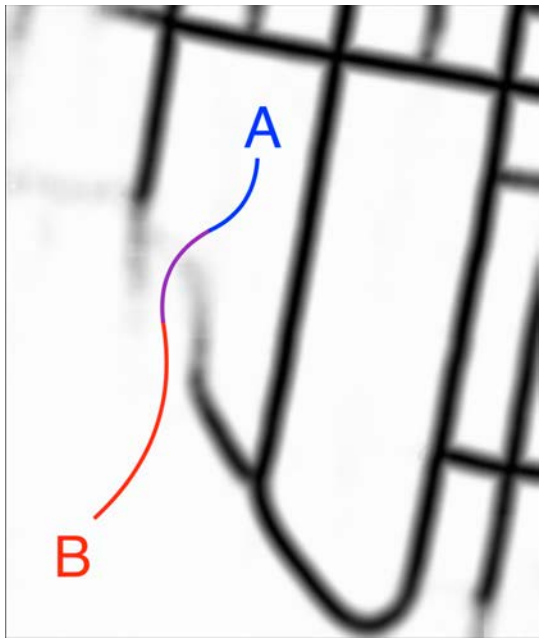
Accounting for Topology



Image

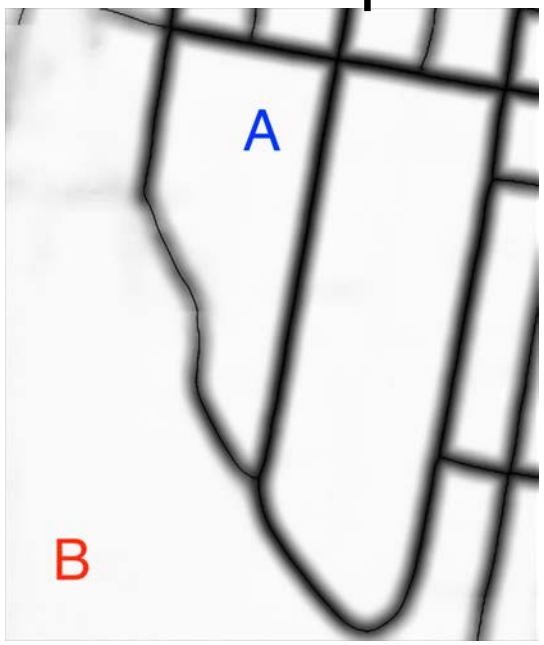


Ground truth



UNet output

—> Add a term in the loss function that penalizes the existence of a path between A and B.

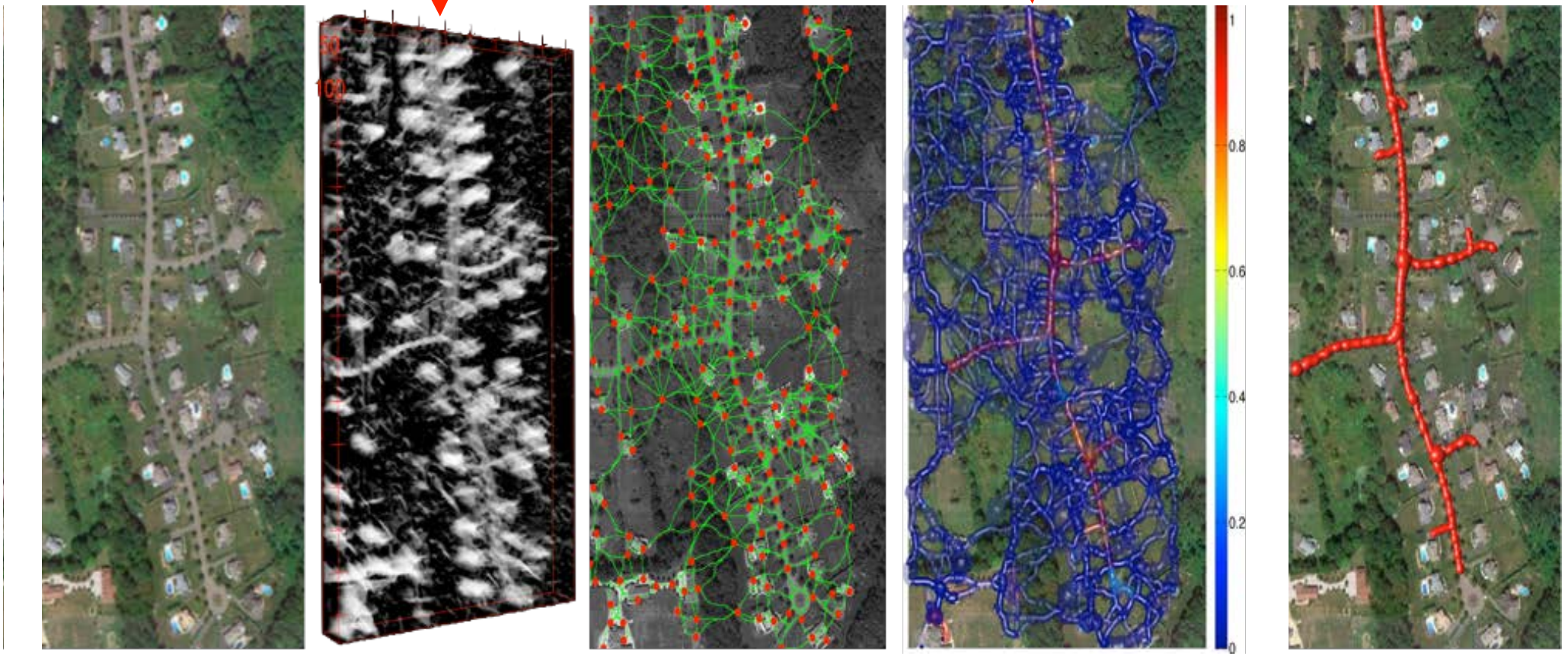


Improved output

Before Deep Learning

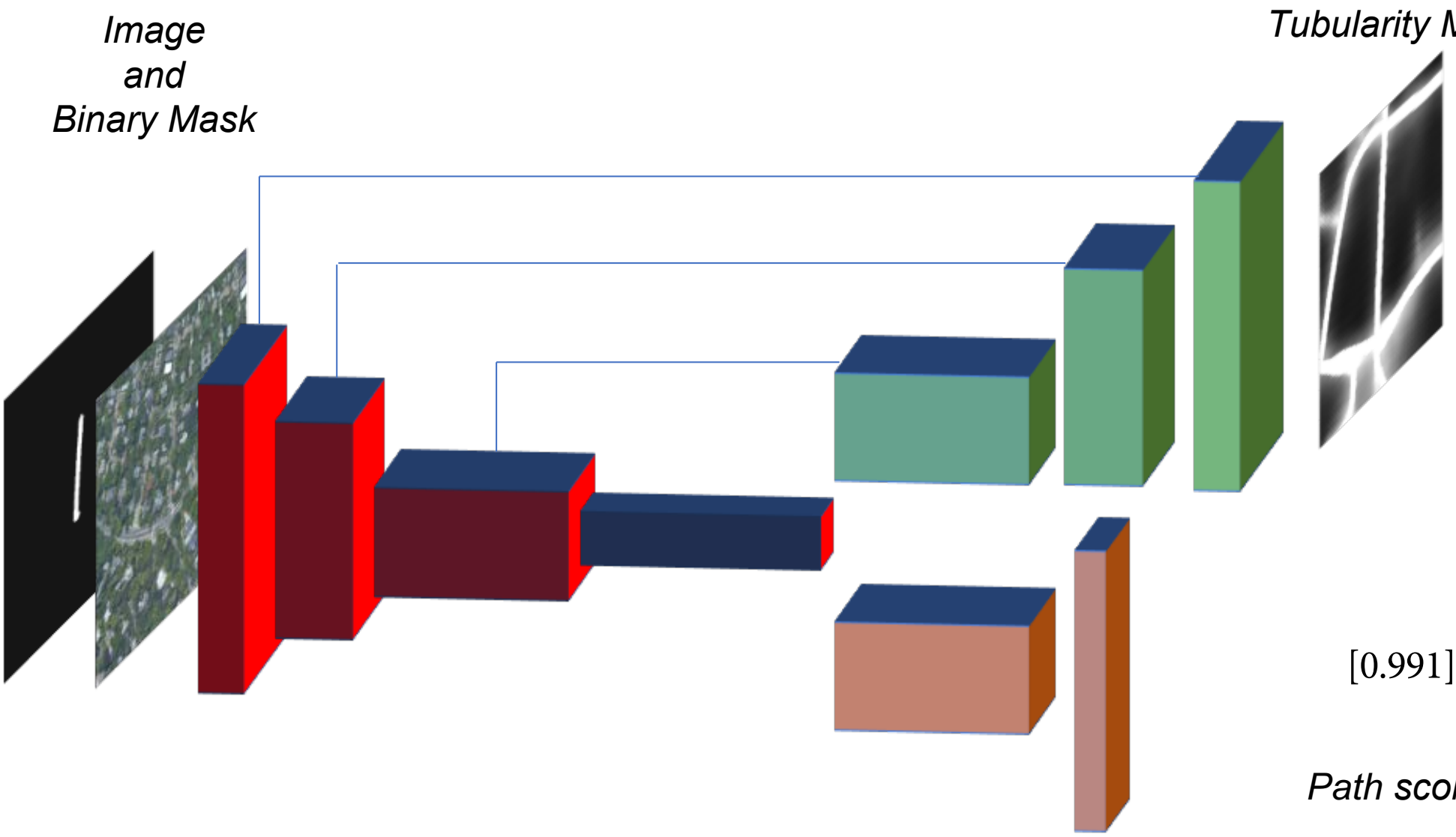
U-Net does this better.

Can it also do this?



Turetken et al., PAMI'16.

Dual Use UNet

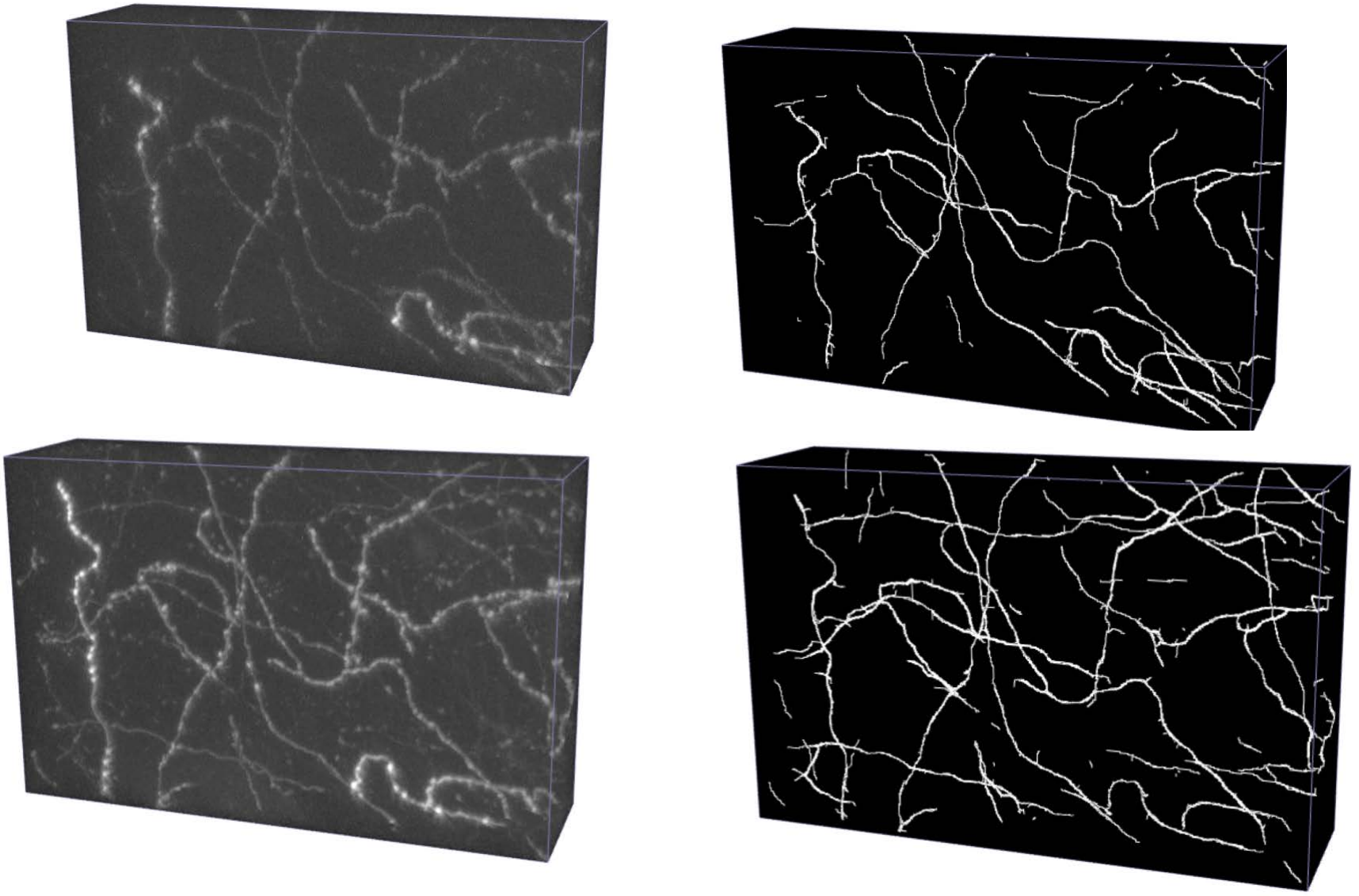


After Deep Learning

1. Compute a probability map.
2. Sample and connect the samples.
3. Assign a weight to the paths.
4. Retain the best paths.

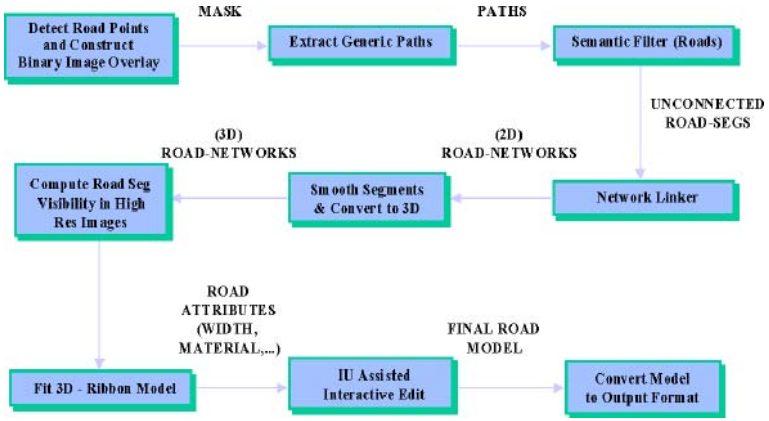
Streets of Toronto

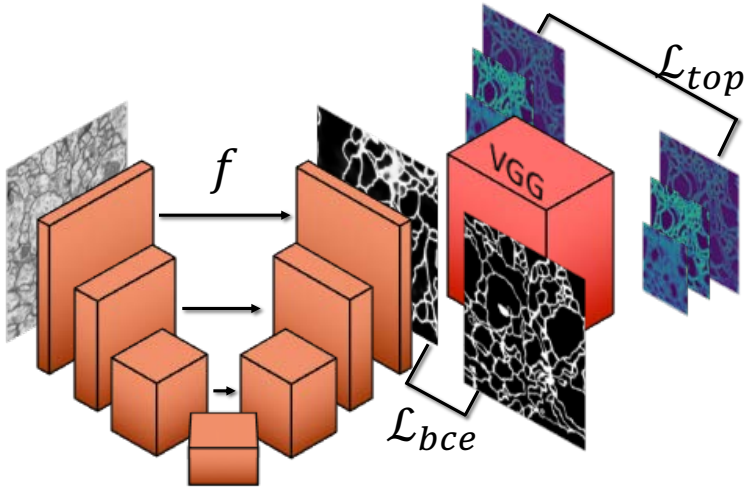
Dendrites and Axons



- Deep learning allows the **same** algorithm to work in different contexts.
- The implementation is informed by earlier approaches.

1998 - 2038





1998 2018 2038

It is difficult to make predictions, especially about the future.
Sometimes attributed to Niels Bohr.