Introduction to Machine Learning

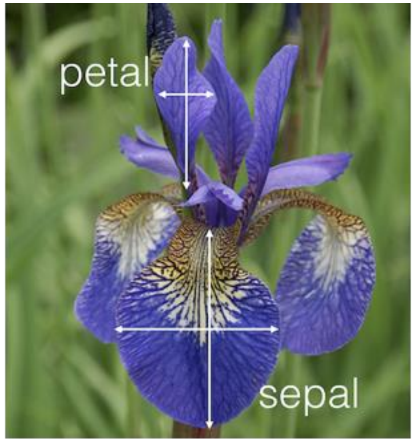
Part III: network architectures

Judith Katzy
Hamburg, September 2024

HELMHOLTZ

Multi-variate classification based on features

identify Iris plants as belonging into 3 different categories based on their petal and sepal length and width



“engineered features”

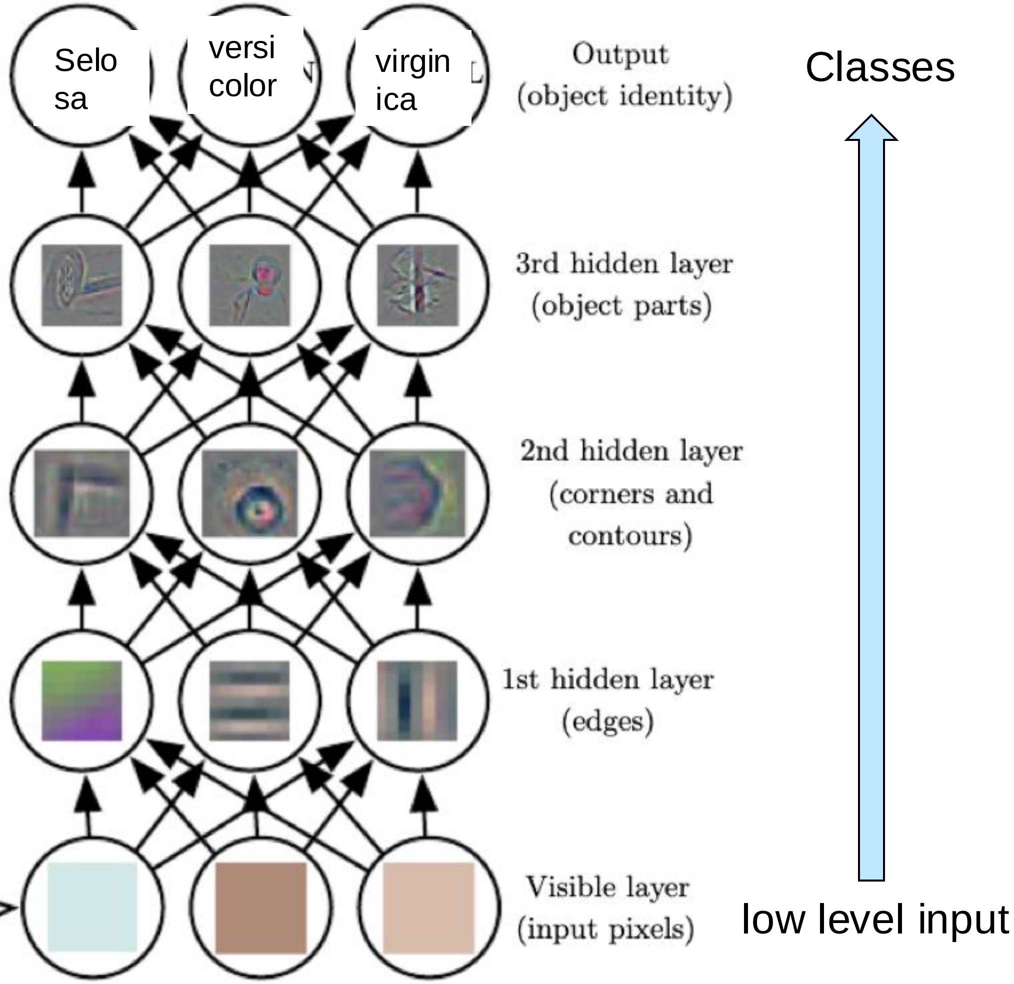
Iris setosa				*Iris versicolor*				*Iris virginica*			
Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width
5·1	3·5	1·4	0·2	7·0	3·2	4·7	1·4	6·3	3·3	6·0	2·5
4·9	3·0	1·4	0·2	6·4	3·2	4·5	1·5	5·8	2·7	5·1	1·9
4·7	3·2	1·3	0·2	6·9	3·1	4·9	1·5	7·1	3·0	5·9	2·1
4·6	3·1	1·5	0·2	5·5	2·3	4·0	1·3	6·3	2·9	5·6	1·8
5·0	3·6	1·4	0·2	6·5	2·8	4·6	1·5	6·5	3·0	5·8	2·2
5·4	3·9	1·7	0·4	5·7	2·8	4·5	1·3	7·6	3·0	6·6	2·1
4·6	3·4	1·4	0·3	6·3	3·3	4·7	1·6	4·9	2·5	4·5	1·7
5·0	3·4	1·5	0·2	4·9	2·4	3·3	1·0	7·3	2·9	6·3	1·8
4·4	2·9	1·4	0·2	6·6	2·9	4·6	1·3	6·7	2·5	5·8	1·8

Morphological Measures of Iris Flowers (Part of the Iris Dataset, Source & License)

Fisher, R.A.(1936), the use of multiple measurements in taxonomic problems, Annals of Eugenics, 7:179-188

Deep learning

Picture taken as
Matrix of pixels



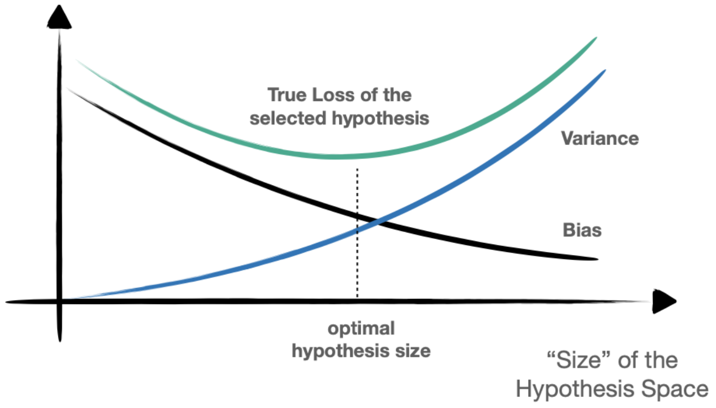
The assumption is that effective machine-learned tasks should start from low level in puts and go through layers of abstraction to learn the classification

Beyond depth….

- Can we push this further, should we move away from universal function approximators?

➢ bias variance tradeoff: reduce as much as you can

General Idea: should match **data modality & task**

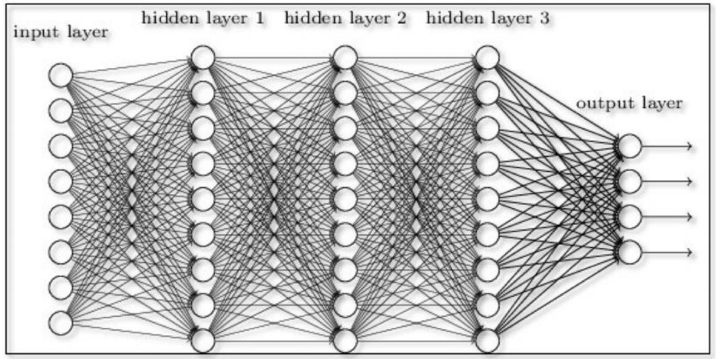


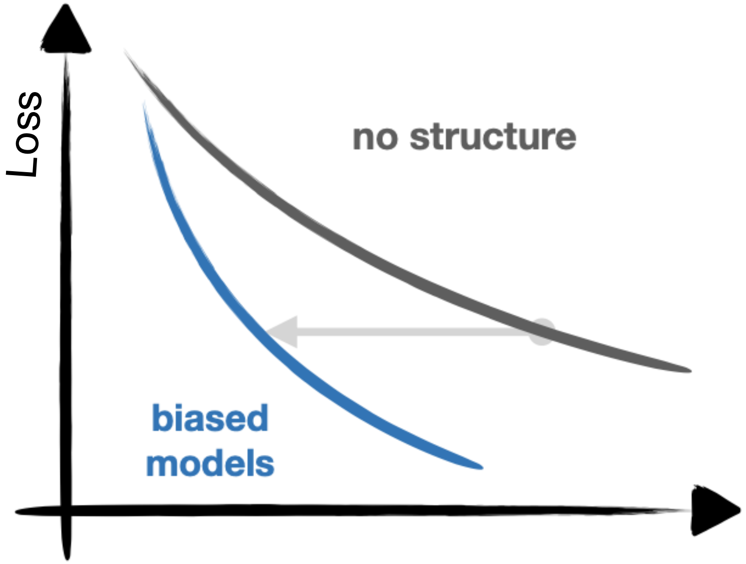
Inductive Bias

If we can throw out irrelevant functions, which we know can’t be the solution, we **bias** our inductive process towards good solution

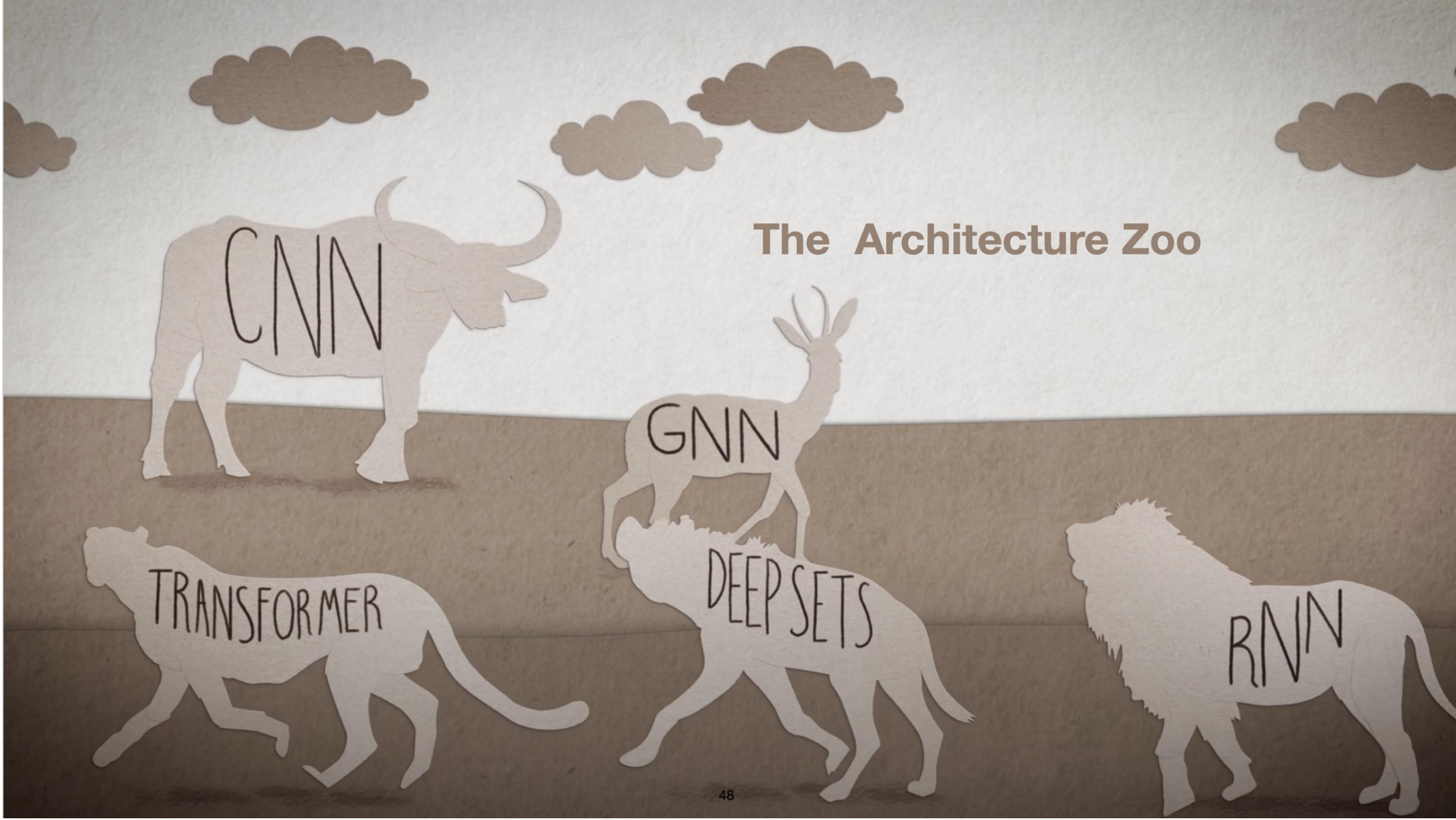
- ➢ here: bias is good

Unstructured models





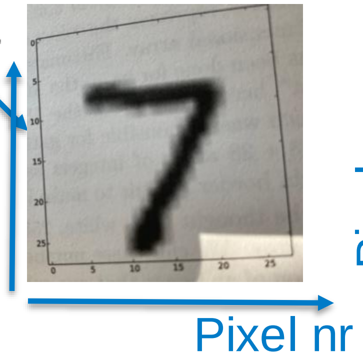
The Architecture Zoo

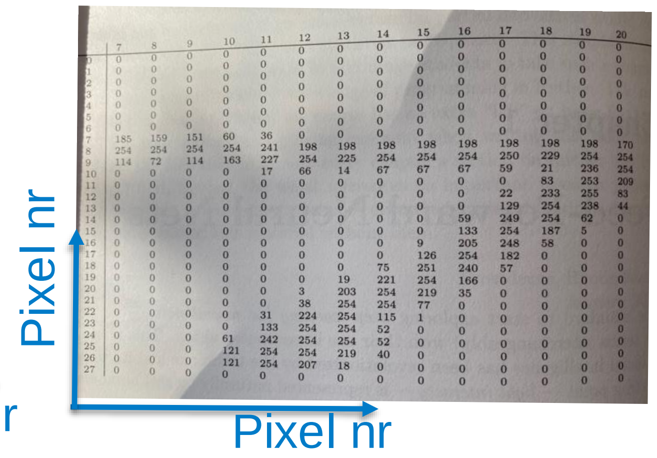


MNIST

2 dim regular grid

MNIST (Modified National Institute of Standards and Technology database) set of 70000 handwritten digits classified into 10 classes (0-9)





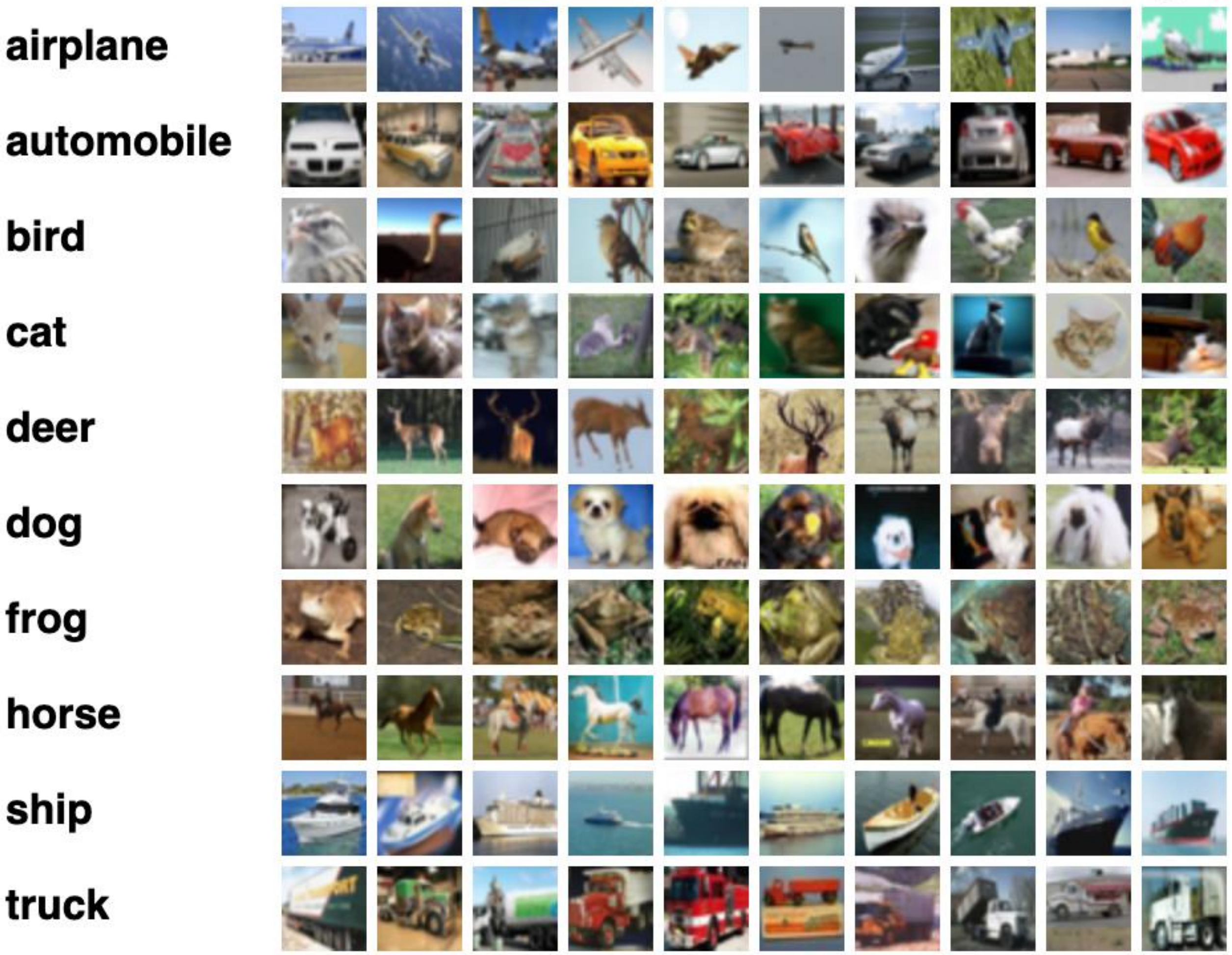
Grid of 28x28 values (pixels)
Each pixel value [0,255] indicating black/grey shade

CIFAR

2 dim regular grid

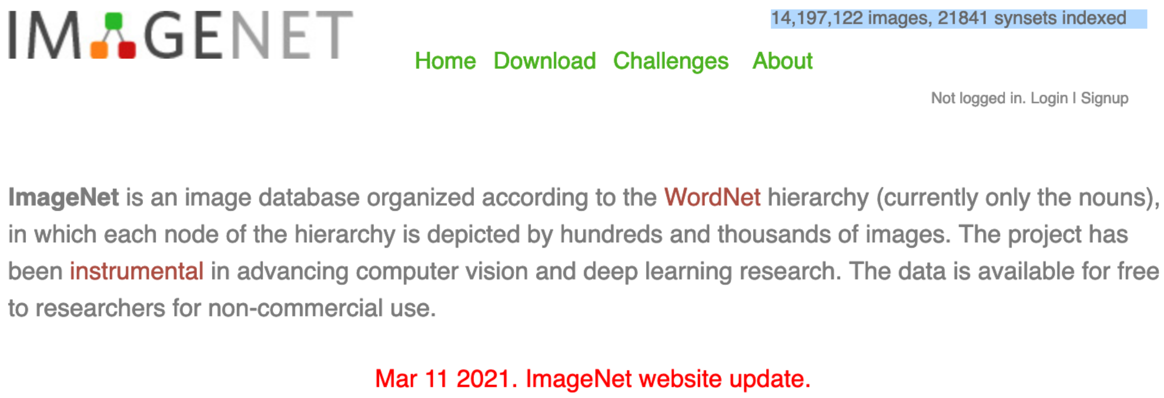
60000 32x32 color images (32x32 values each for Red Green Blue (RGB)) in 10 classes

10 randomly selected pictures out of 6000 per class



IMAGENET

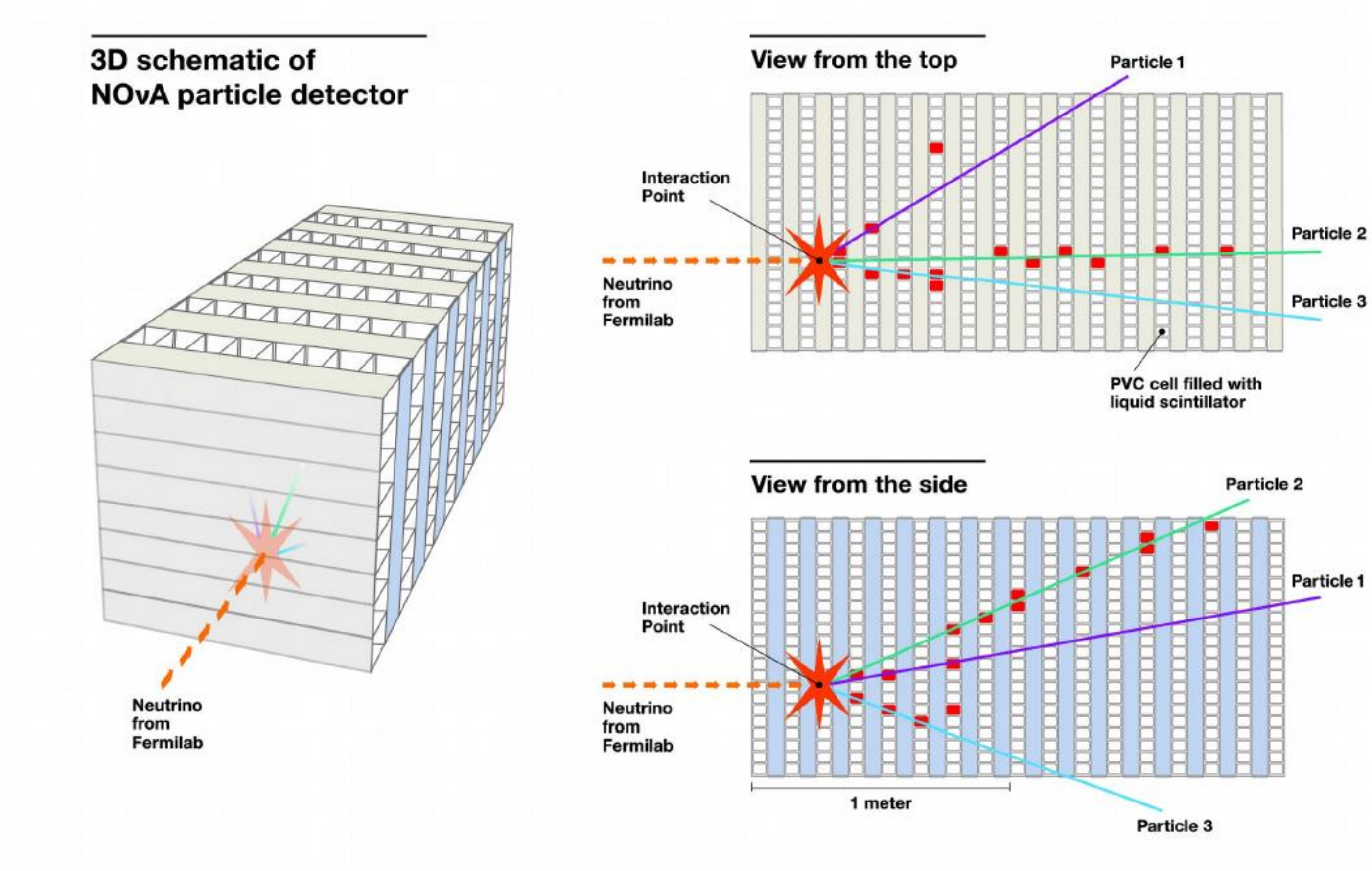
2 dim regular grid



- **14,197,122 images, 21841 categories; ~650 annotated images per category**
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - > Annually contest (solutions on kaggle)
  - > Challenges are on object detection, object location etc.
  - > Winners usually provide significant steps forward in CNN architectures or methods (reference networks)

Particle detector as image

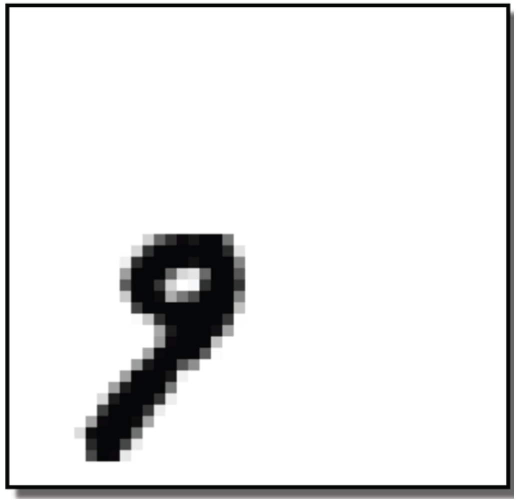
Identify particles in a sampling calorimeter of the Nova detector



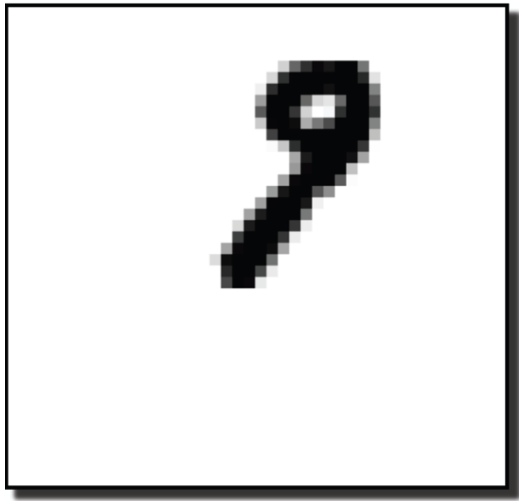
Convolutional Neural Networks

Translational invariance

Is there a 9 in the picture?



And now?



Implement algorithm supporting translational invariance of local structures

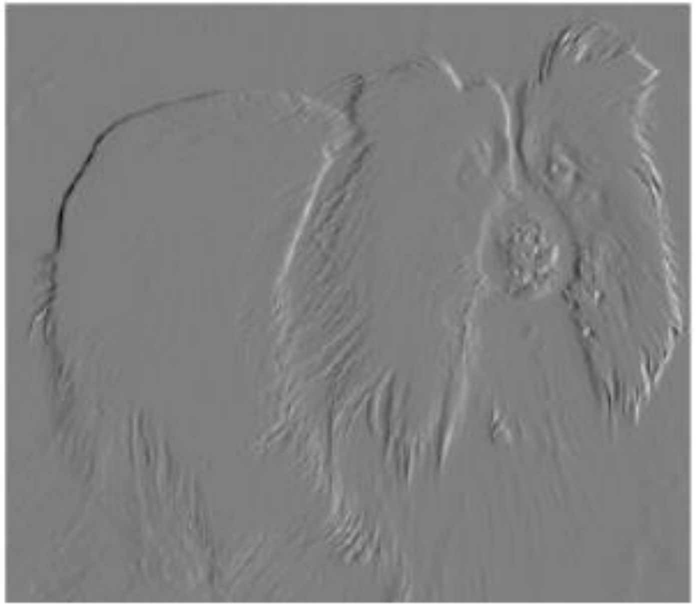
- Of the first successes of deep learning in the early 80’s

For each pixel: subtract the value of its neighboring pixel on the left

?

Edge detection

For each pixel: subtract the value of its neighboring pixel on the left



Convolutions

Two key ideas lead to the use of convolutions as building blocks of networks

➢ Local connectivity and weight sharing

Convolution

$$F(\tau) = f * g \ = \int f(\tau - t)\, g(t)\, dt$$

input distribution

"kernel" or "filter"

Convolution in 2 dimension – discrete case

$$I(i, j) = \text{Image}$$

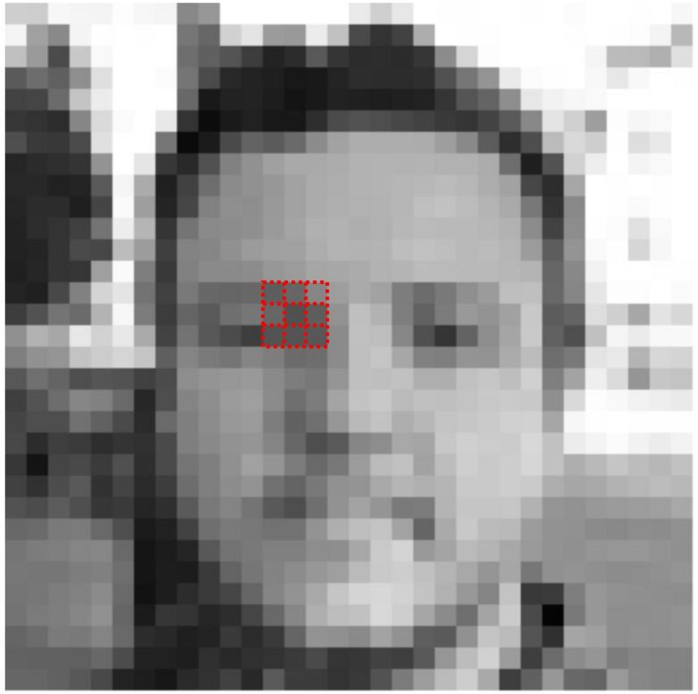
$$K(m, n) = \text{Kernel}$$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n)$$

Filtered image

Analysing images by convolution: blurr

$$S(i,j) = (I * K)(i,j) = \sum \sum I(i-m, j-n) K(m,n)$$



input image

I(i-m,j-n)

$$\begin{pmatrix} 94 \times 0.0625 + 103 \times 0.125 + 134 \times 0.0625 \\ + 103 \times 0.125 + 81 \times 0.25 + 94 \times 0.125 \\ + 83 \times 0.0625 + 85 \times 0.125 + 88 \times 0.0625 \end{pmatrix}$$

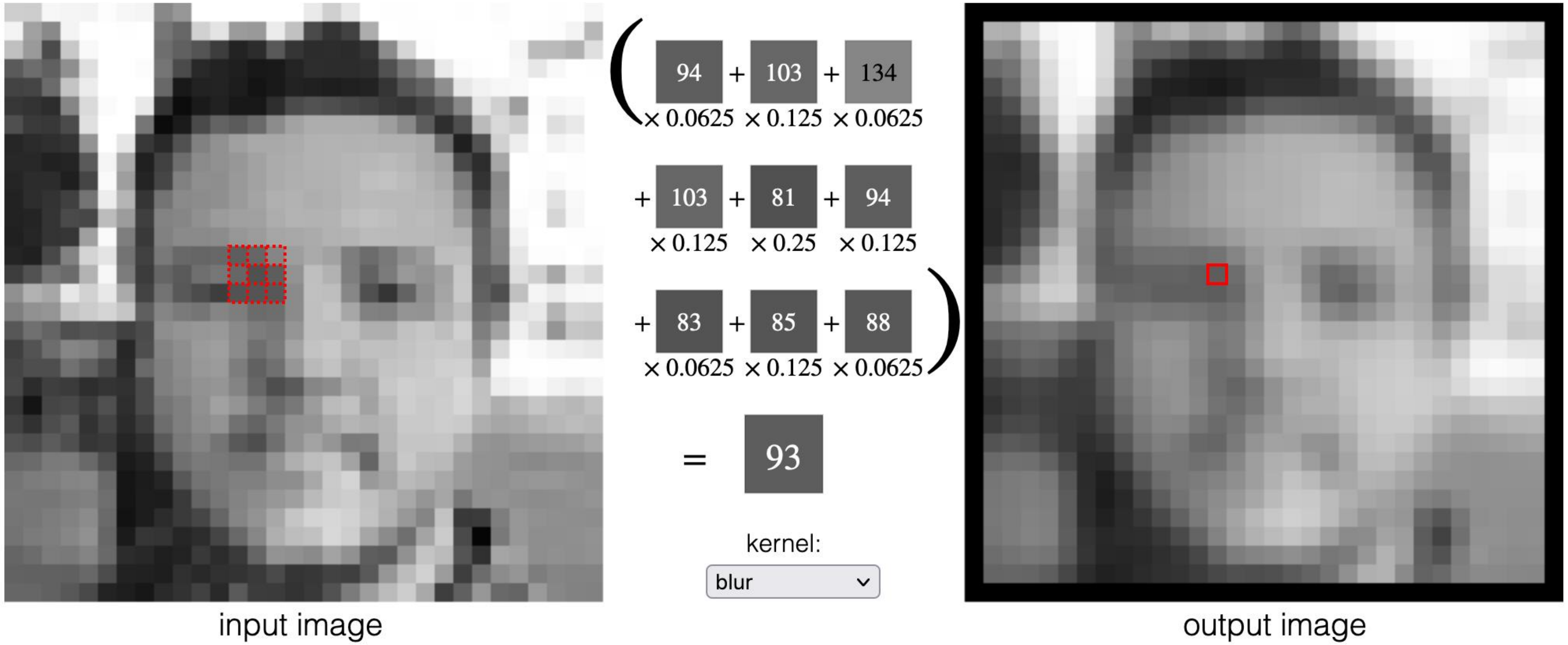
K(m,n)

$$\begin{pmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{pmatrix}$$

S(i,j)

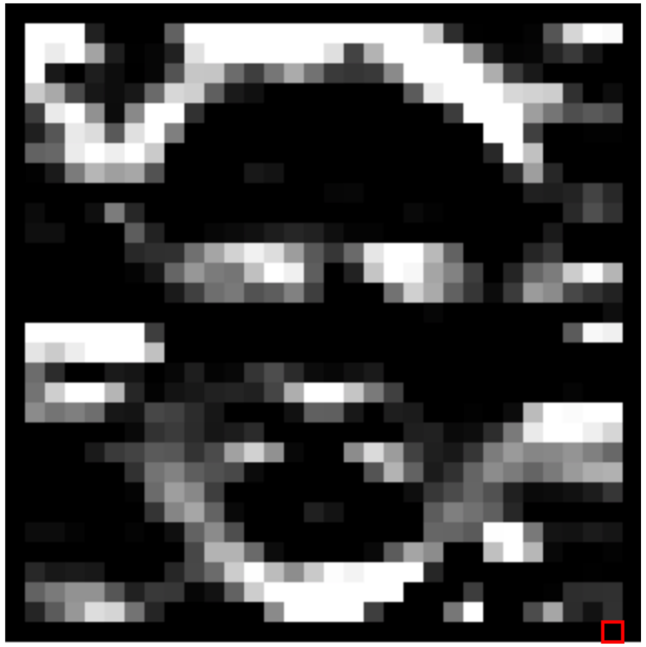
= 93

Sliding through the full picture



Analysing images by convolution: top sobel

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

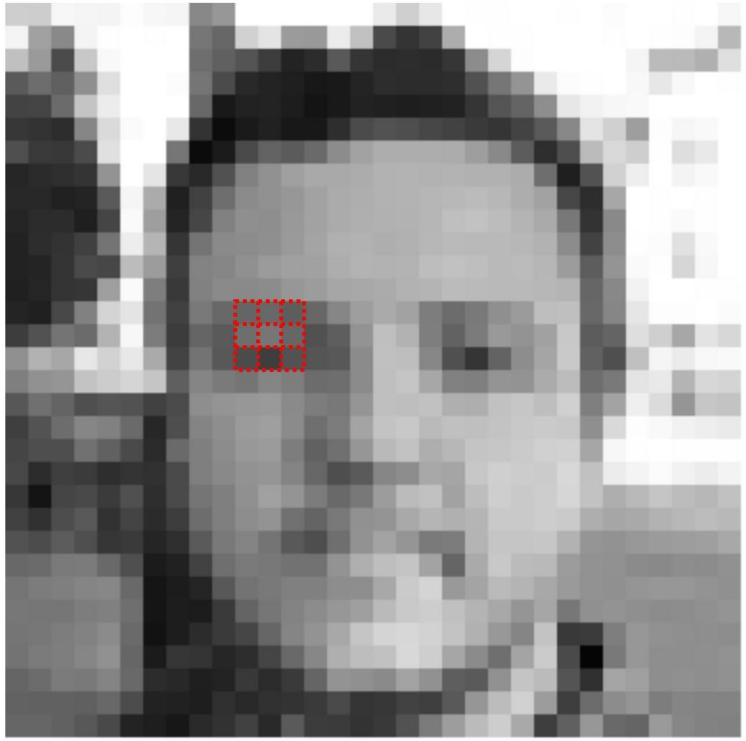


Analysing images by convolution: emboss

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

Analysing images by convolution: sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



input image

$$\left(\underset{\times 0}{102} + \underset{\times -1}{104} + \underset{\times 0}{94} + \underset{\times -1}{111} + \underset{\times 5}{119} + \underset{\times -1}{103} + \underset{\times 0}{76} + \underset{\times -1}{62} + \underset{\times 0}{83} \right)$$

$$= 215$$

kernel:

sharpen

output image

Have fun with kernels

https://setosa.io/ev/image-kernels/

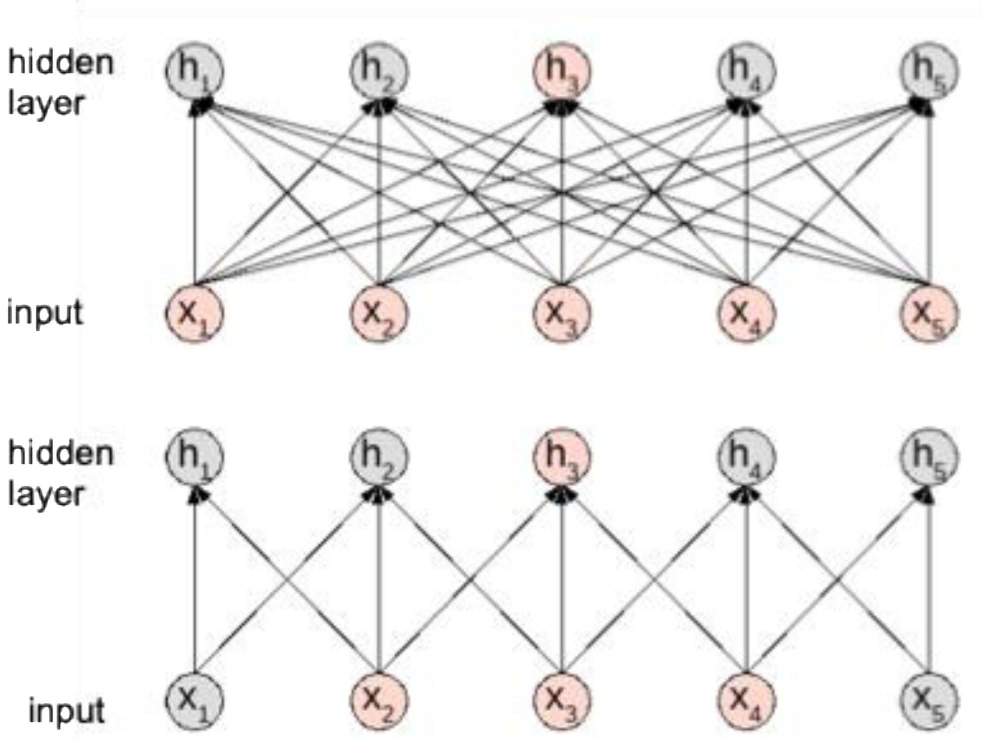
In CNNs we train filters of various sizes

$$\begin{bmatrix} w & w_2 \\ w_3 & w_4 \end{bmatrix} \quad \begin{bmatrix} w & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} \quad \begin{bmatrix} w & w_2 & w_3 & w_4 \\ w_5 & w_6 & w_7 & w_8 \\ w_9 & w_{10} & w_{11} & w_{12} \\ w_{13} & w_{14} & w_{15} & w_{16} \end{bmatrix} \quad \ldots.$$

How can we implement the “filter” process in a neural network architecture?

Sparse local connectivity

1 dim case

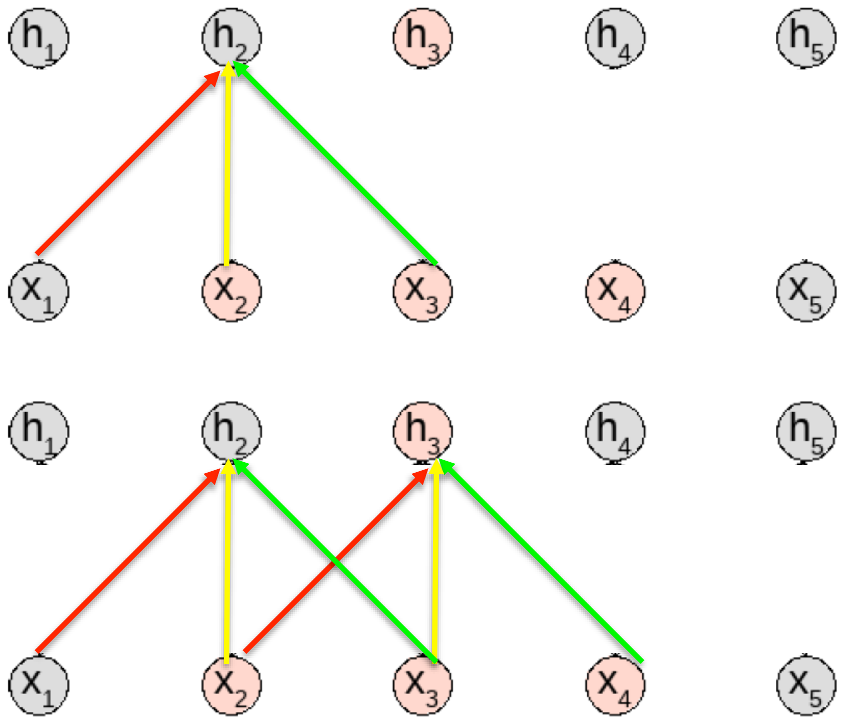


Fully connected neural network:
h_3 receives input from all input nodes

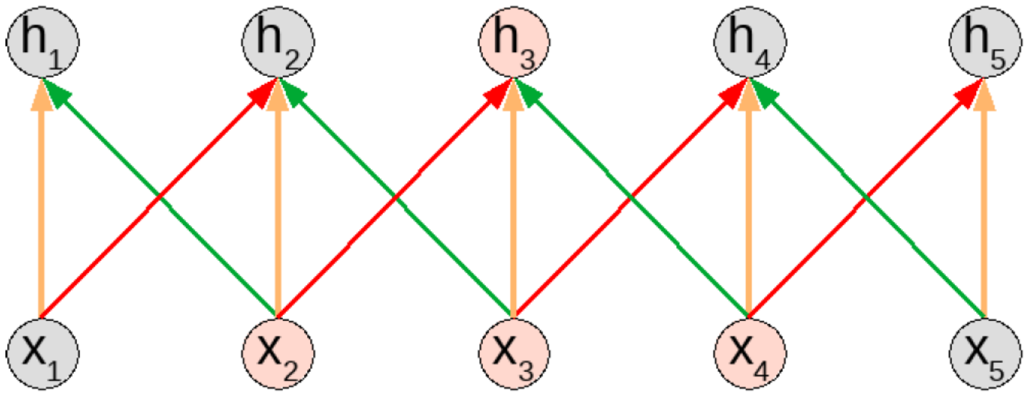
CNN:
h_3 receives input from few input nodes

Weight sharing

Simulate filter process: Same weights when moving over the input

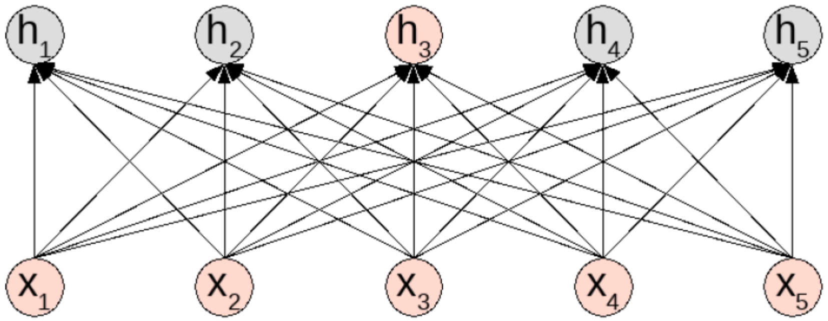


Weight sharing complete layer

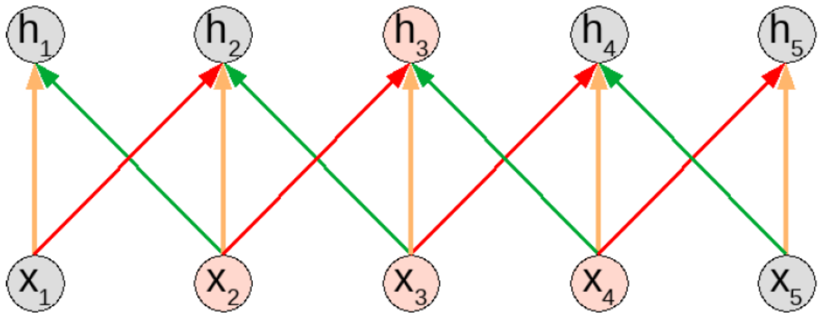


Same color = shared weight

Significant reduction of training weights



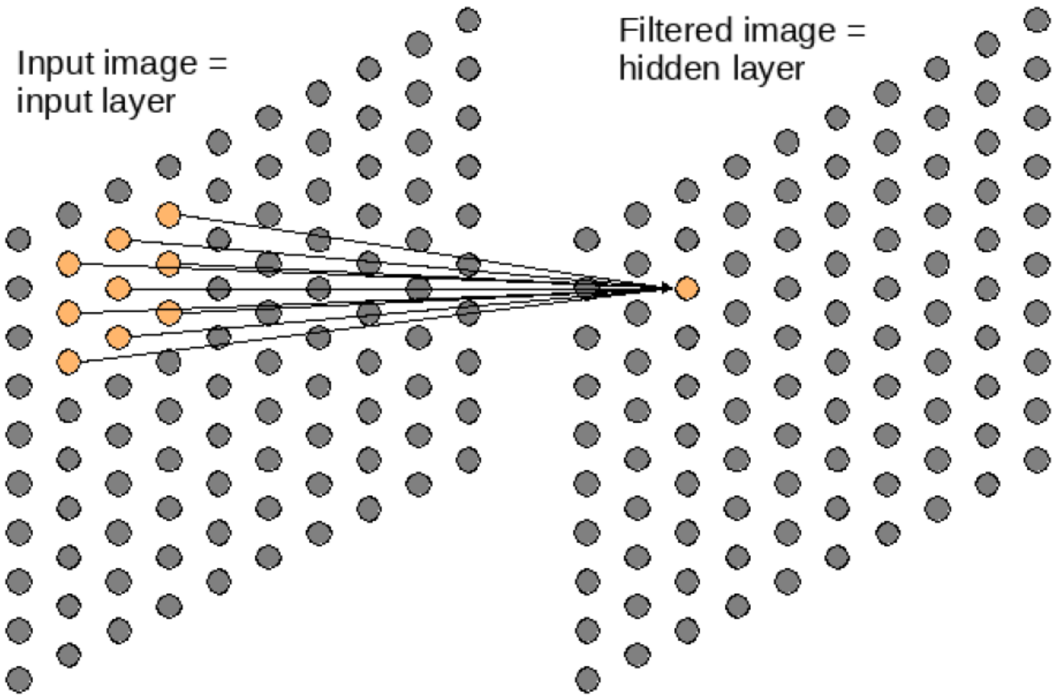
Fully connected neural network:
25 (unique) weights (+5 biases)



CNN:
3 (unique) weights (+3 biases)

For both cases: at hidden node bias is added and activation function $\sigma\,(\Sigma\, W_j\, x_j + b)$ is applied

2d images with 2d filters



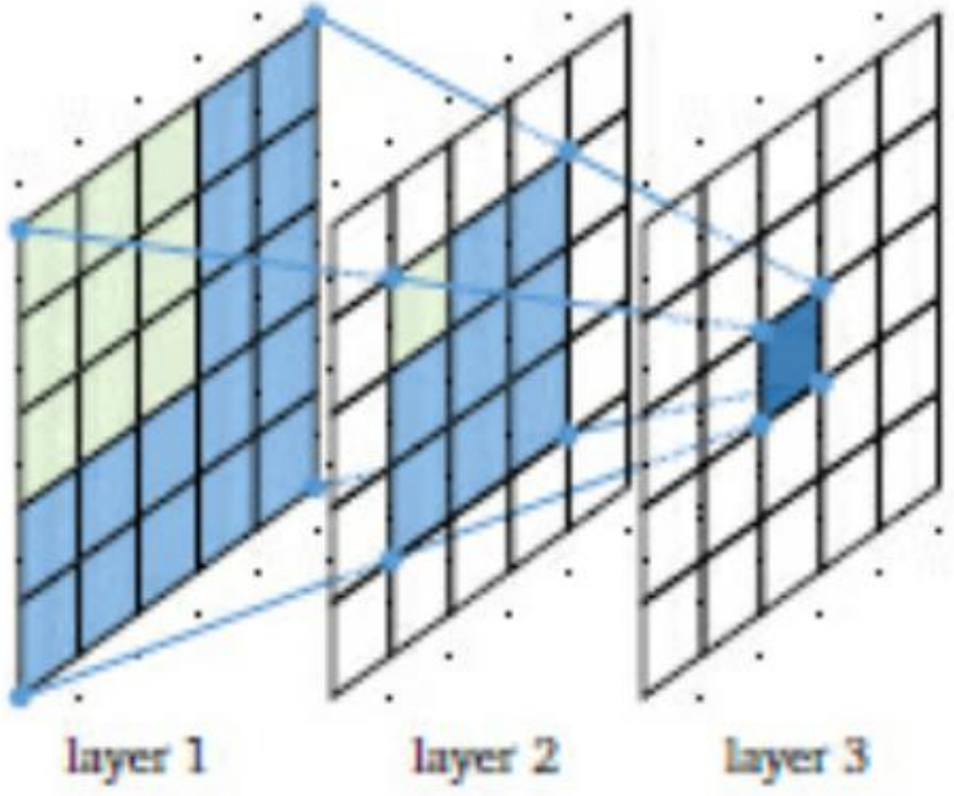
“feature map”

Each hidden node shares the weight with all other hidden nodes of the layer

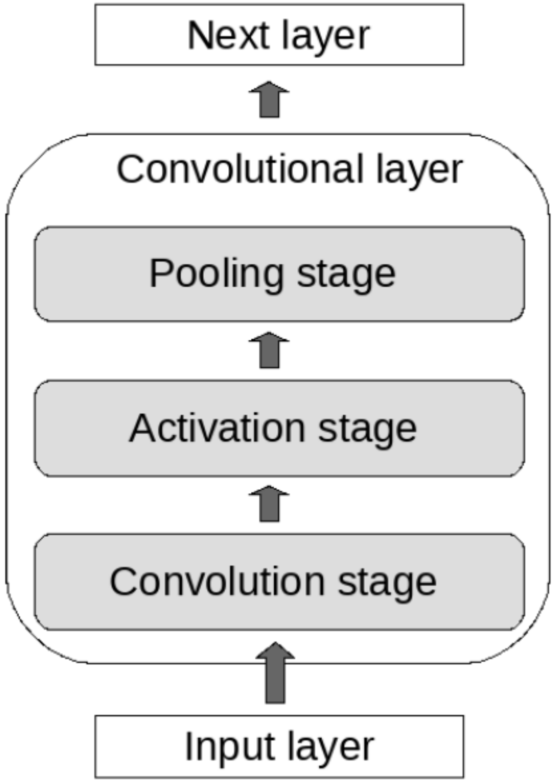
At hidden node: add bias and apply activation function $\sigma\ (\Sigma\ W_j\ x_j + b)$

Going deeper….

- Receptive field of multi-layer CNN

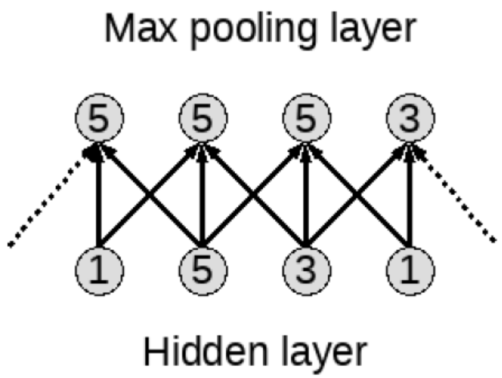


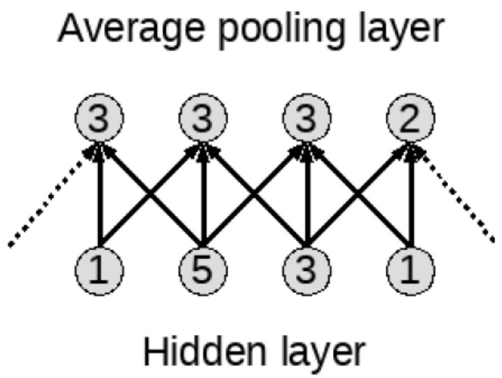
Convolutional layer



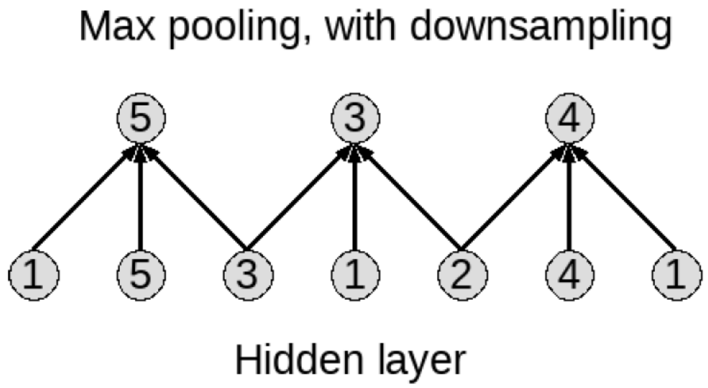
Typically ReLu

Pooling in 1dim CNN



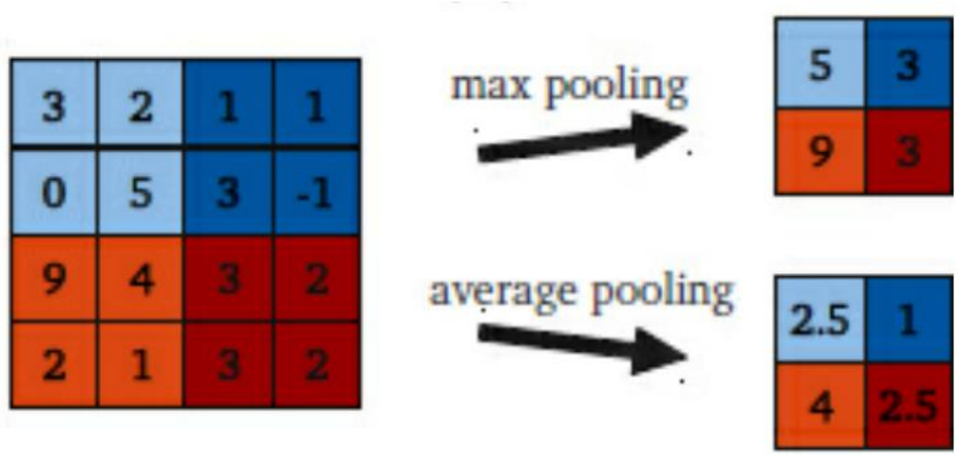


no downsampling



with downsampling

2d CNNs: Pooling

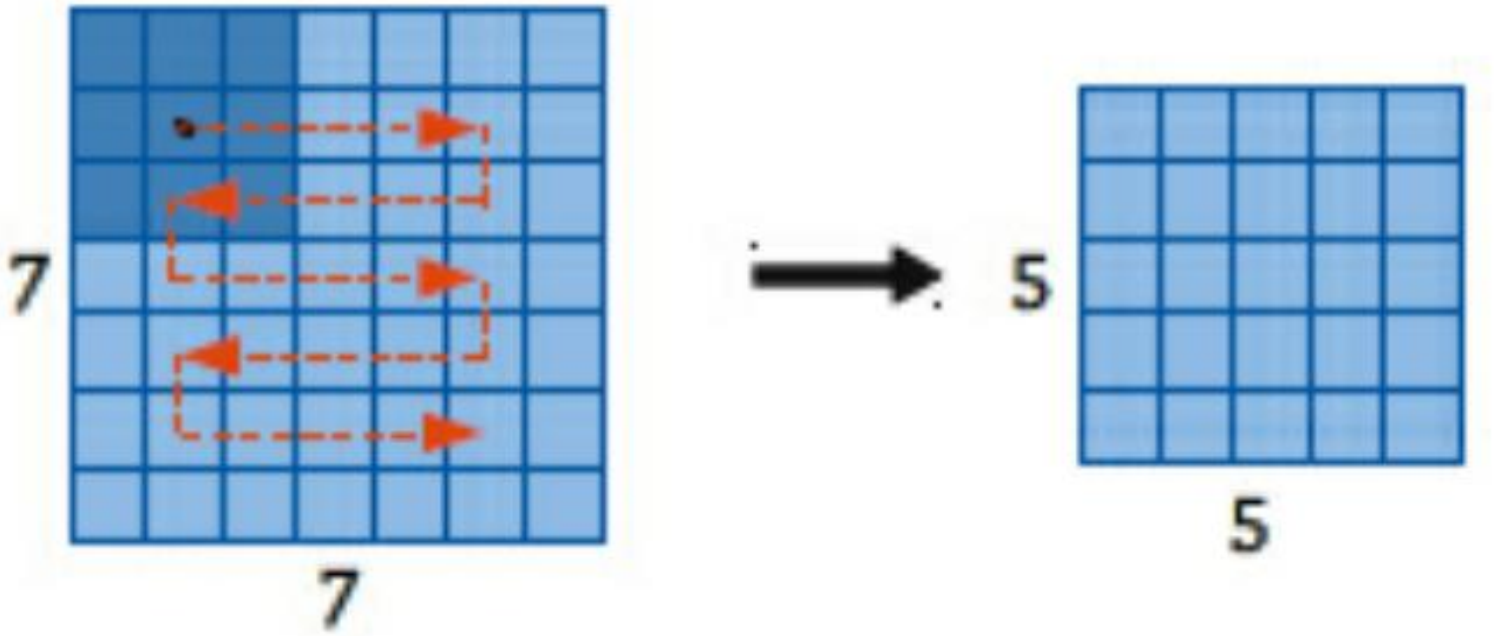


- Max pooling over spatial positions is naturally invariant to translation
- Downsampling of image size

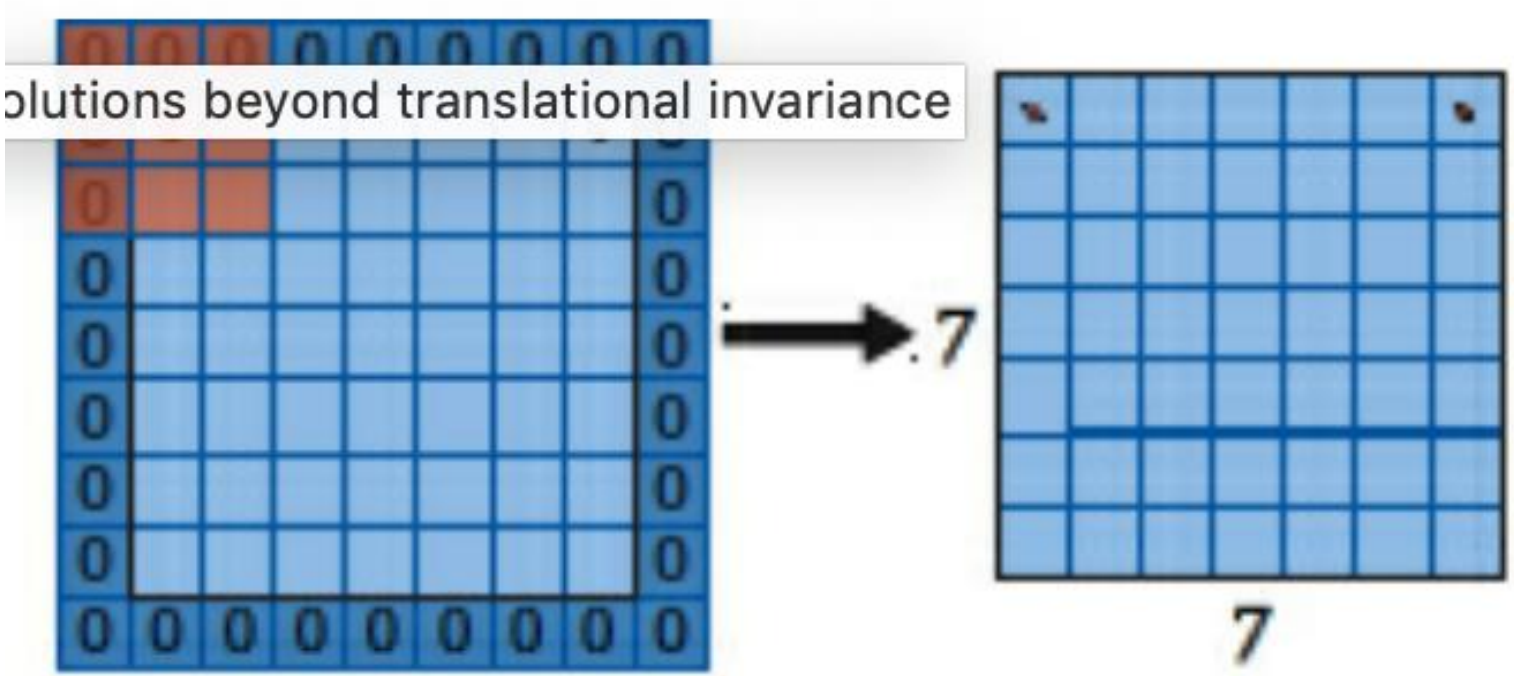
- Also possible: global pooling over complete feature map

> Drastically reduced image size

Details of convolution: Padding

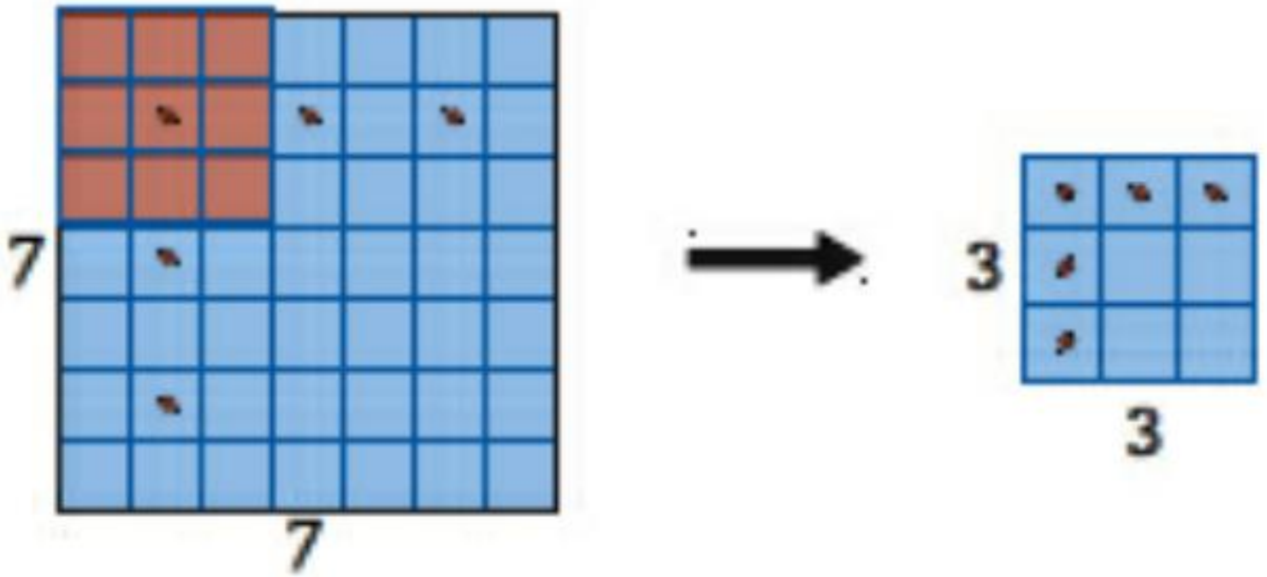


Details of convolution: Padding



- Zero padding to treat edges when keeping image size

Details of convolution: Stride

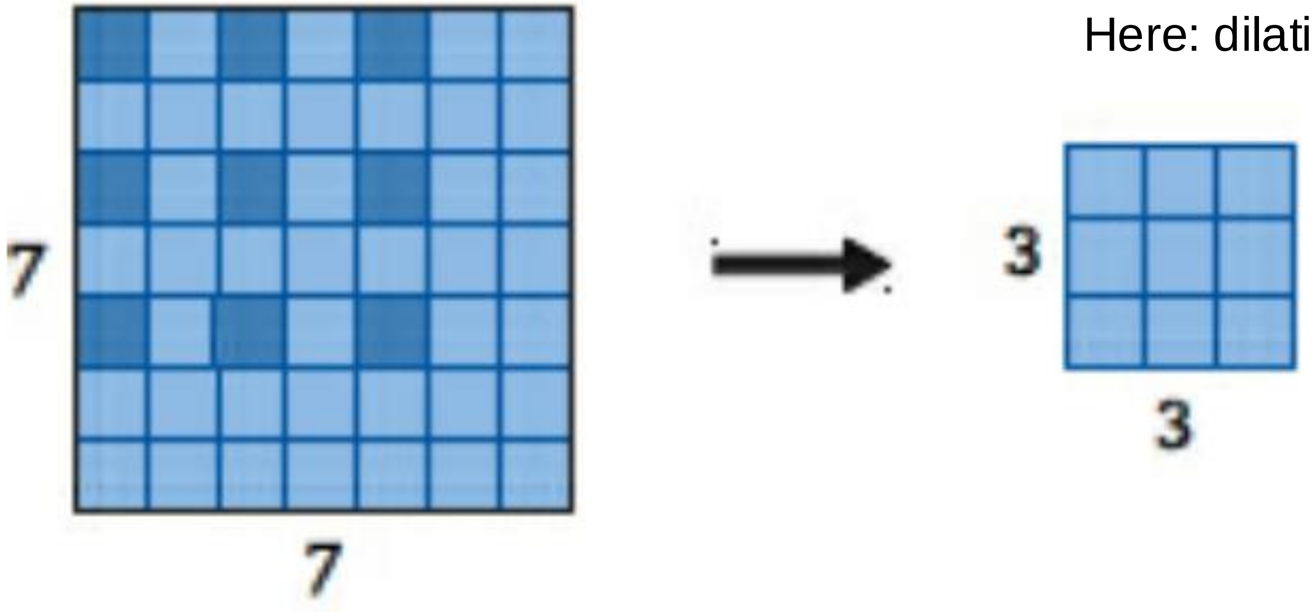


$$\text{output width} = \frac{W - F_w + 2P}{S_w} + 1$$

$$\text{output height} = \frac{H - F_h + 2P}{S_h} + 1$$

P = padding
S = stride
F = filter size

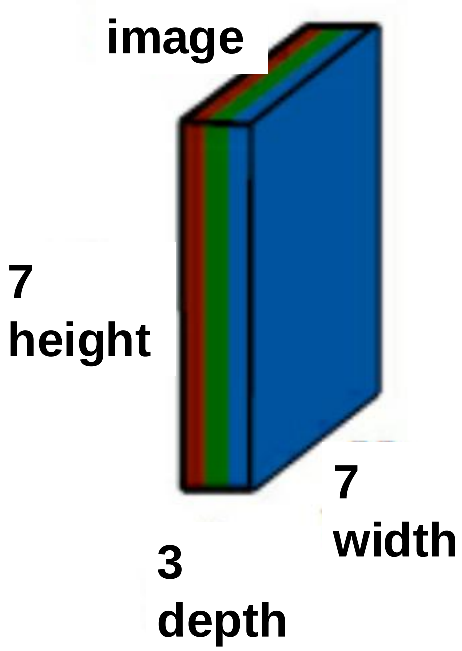
Details of convolution: Dilation

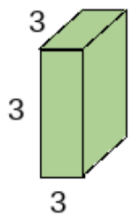


Here: dilation rate = 1 on a 3x3 filter

- Filter with Gaps to capture larger area without increasing the number of weights
  > Useful to create large receptive field of view within a few layers

Coloured images (RGB)



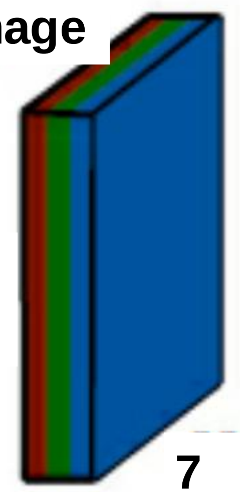


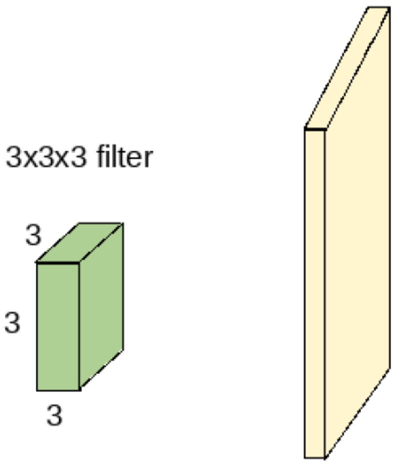
Remember: Filters always extend to the full dept of the input image

Coloured images (RGB)

Regardless of the input depth the output has depth 1

5x5x1 feature map

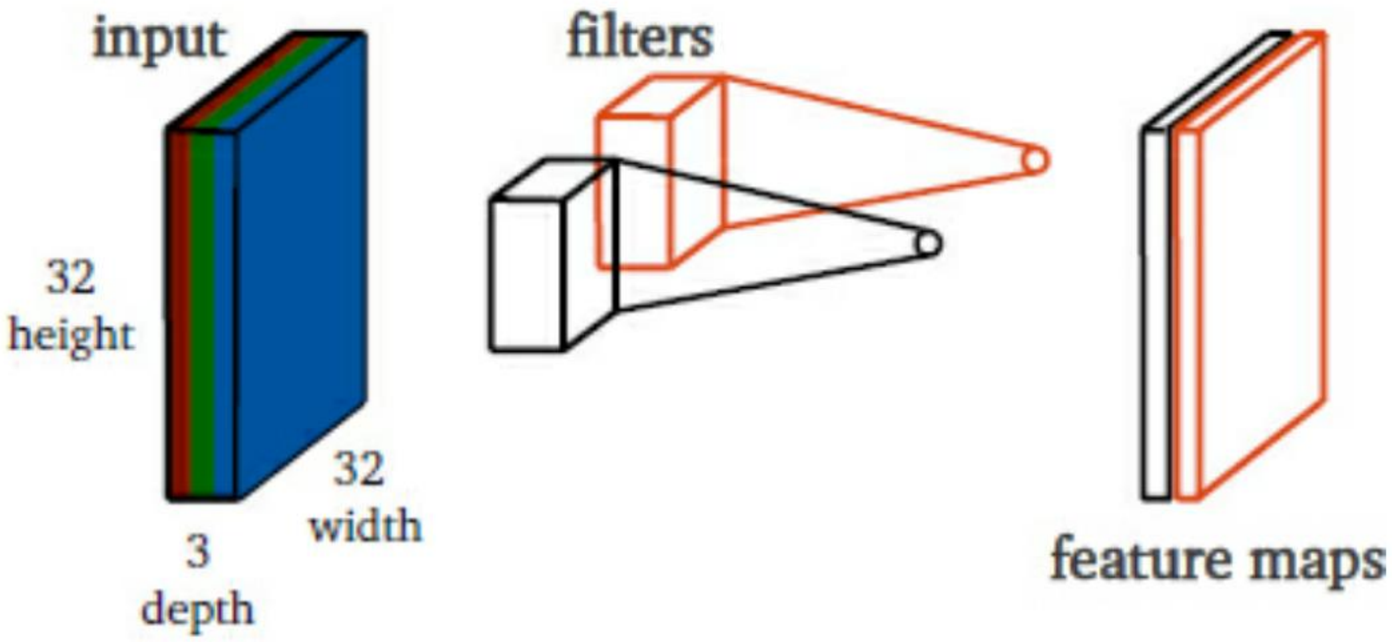




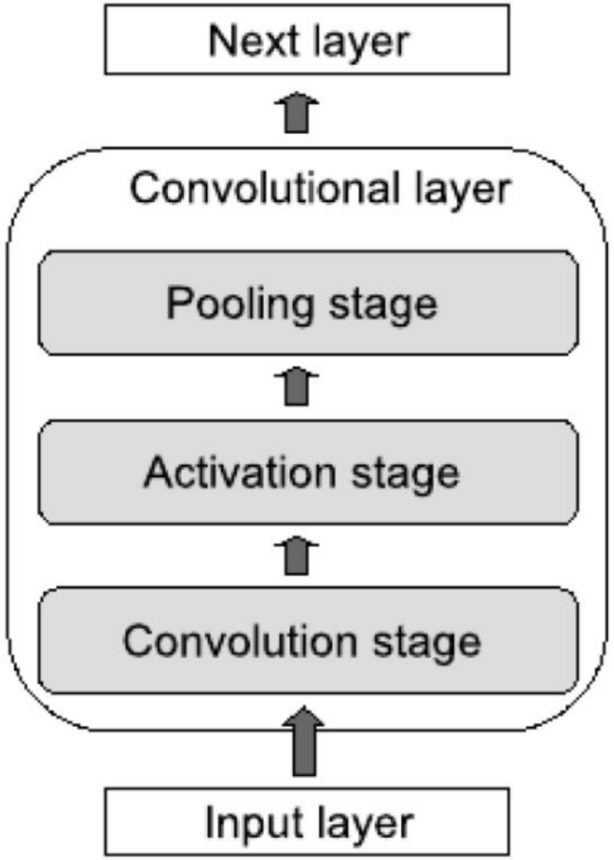
27 filter weights + 1 bias

$$Y'_{i,j,f} = \sigma\left(\sum_{k=-1}^{1}\sum_{l=-1}^{1}\sum_{c=1}^{3} W_{k,l,c,f}\, X_{i+k,\, j+l,\, c} + b_f\right)$$

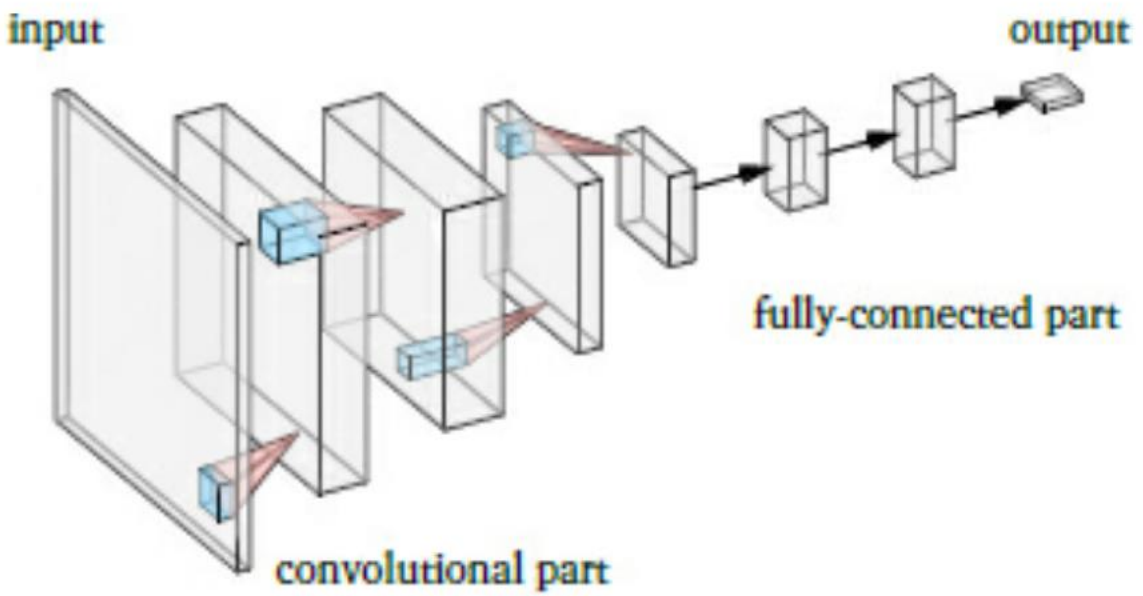
Coloured images, multiple filters



Building CNN out of building blocks

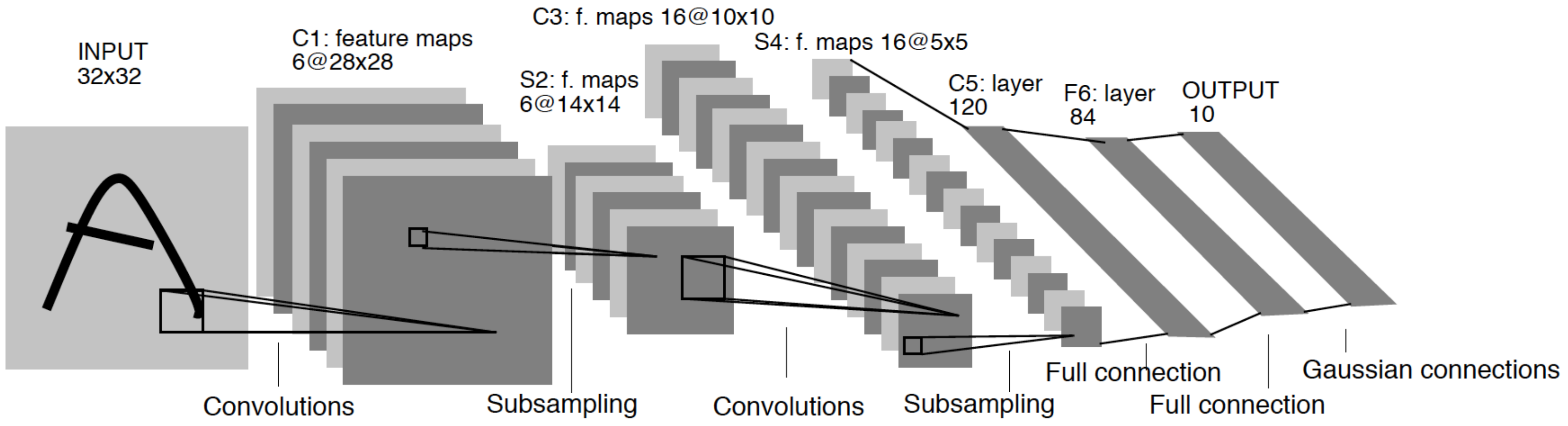


Complete CNN



LeNet-5

- One of the very first CNNs, LeCun (1998)



CNN in PyTorch

```
self.layers.append(torch.nn.Conv1d(in_channels=1,out_channels=20,kernel_size=11))
self.layers.append(torch.nn.ReLU())
```

-> More in Peters exercis

Image data

Standard data sets to compare ML algorithms

Examples of famous CNNs

Alex Net

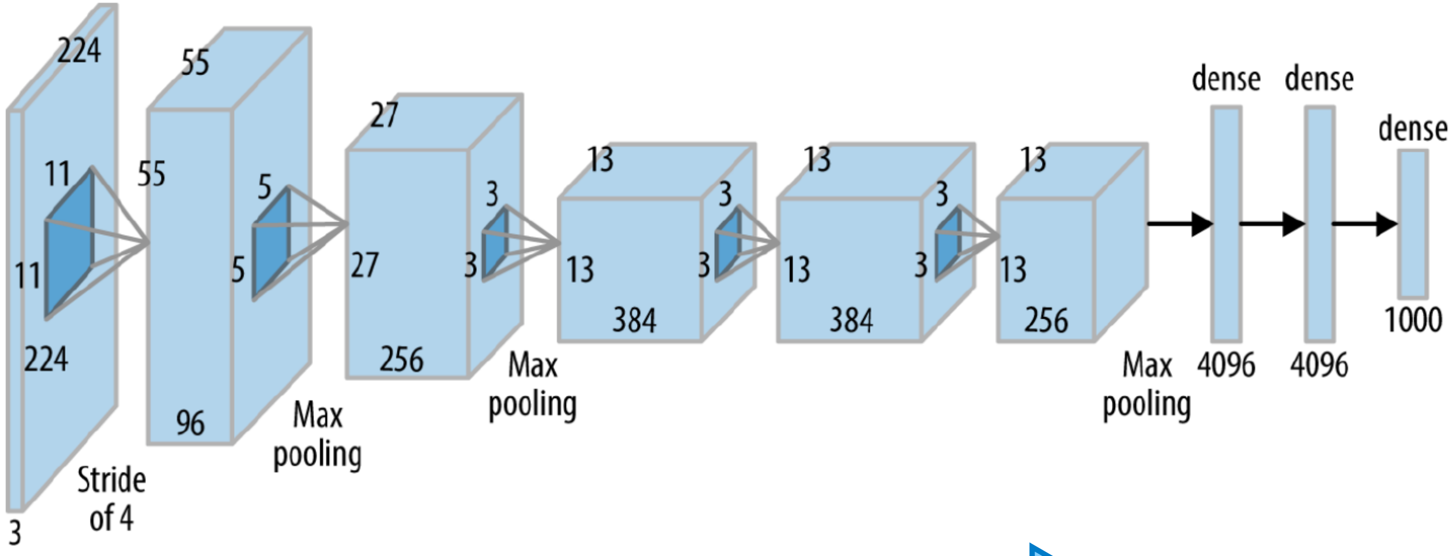
https://dl.acm.org/doi/10.1145/3065386

- Classifies 1000 objects of Imagenet
- 1.2 million training images
- 100 000 test images

> Winner of ILSVRC2012
  > 10% better than 2nd best network
> One of the most influential papers on CNN (8000 citations)
  > Establish **large deep convolutional networks** for imaging

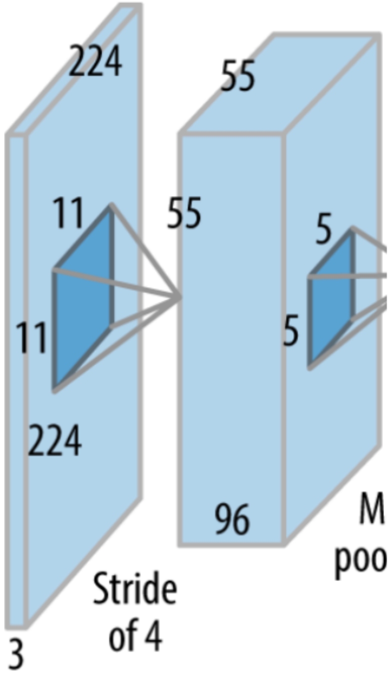
Alex Net structure

5 convolutions **3 dense layers**



Reducing spatial dimension = increasing number of filters

> First convolutional layer:



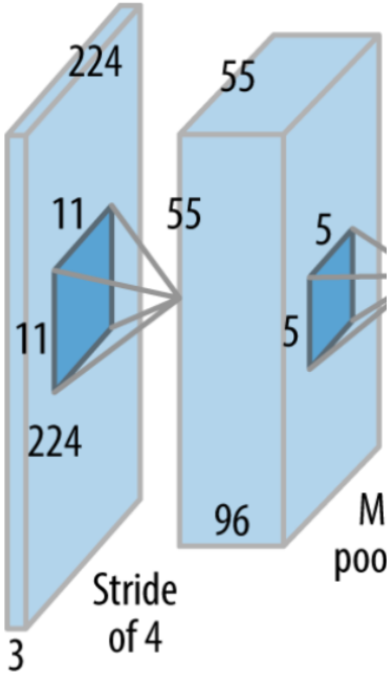
- Images: 227x227x3
- Filter size: 11x11
- Stride: 4
- Conv layer output: 55x55x**96**

W
F
S

Output size: $\frac{W - F + 2P}{S} + 1 = (227 - 11) / 4 + 1 = 55$

P: padding, here P=0

AlexNet - number of parameters



Number of weights: 11 * 11* 3 + 1 per filter
96 filters
(11*11*3+1)*96 = 34944 weights

Number of parameters

Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters
3* 227 * 227					
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944
96 * 55 * 55					
Max Pooling	3 * 3		2		
96 * 27 * 27					
Norm					
Conv2 + Relu	5 * 5	256	1	2	(5 * 5 * 96 + 1) * 256=614656
256 * 27 * 27					
Max Pooling	3 * 3		2		
256 * 13 * 13					
Norm					
Conv3 + Relu	3 * 3	384	1	1	(3 * 3 * 256 + 1) * 384=885120
384 * 13 * 13					
Conv4 + Relu	3 * 3	384	1	1	(3 * 3 * 384 + 1) * 384=1327488
384 * 13 * 13					
Conv5 + Relu	3 * 3	256	1	1	(3 * 3 * 384 + 1) * 256=884992
256 * 13 * 13					
Max Pooling	3 * 3		2		
256 * 6 * 6					
Dropout (rate 0.5)					
FC6 + Relu					256 * 6 * 6 * 4096=37748736
4096					
Dropout (rate 0.5)					
FC7 + Relu					4096 * 4096=16777216
4096					
FC8 + Relu					4096 * 1000=4096000
1000 classes					
Overall					62369152=62.3 million
Conv VS FC					Conv:3.7million

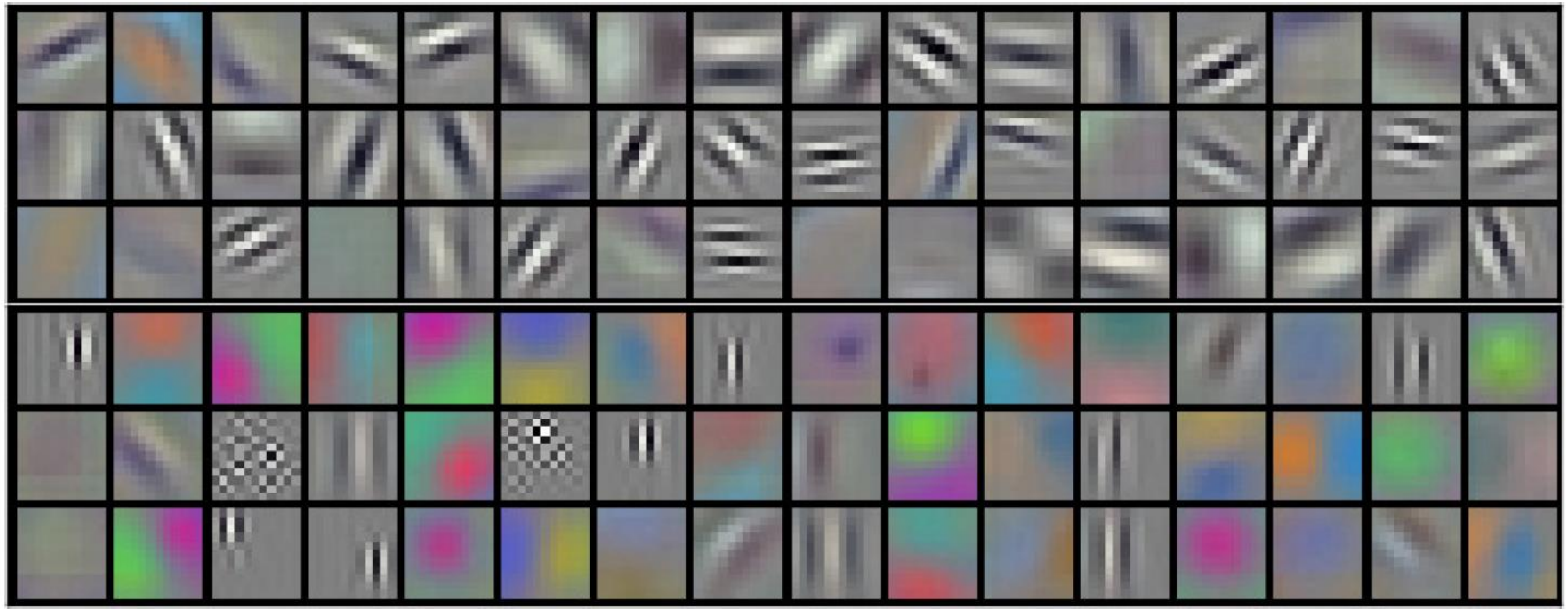
convolutions:
3 million

fully connected part:
59 million

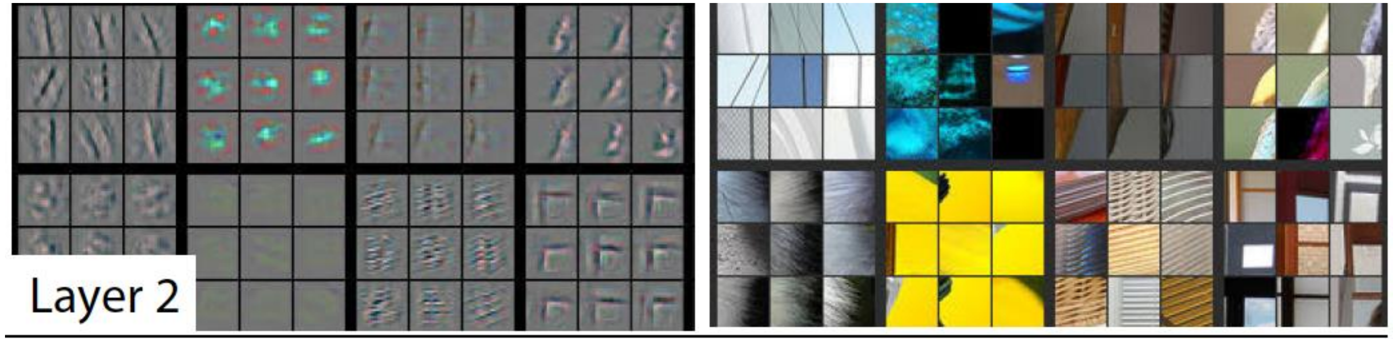
total: 62 million

FC 2 layer dense NN:
~$6*10^9$ weights

What type of filters are being learned?



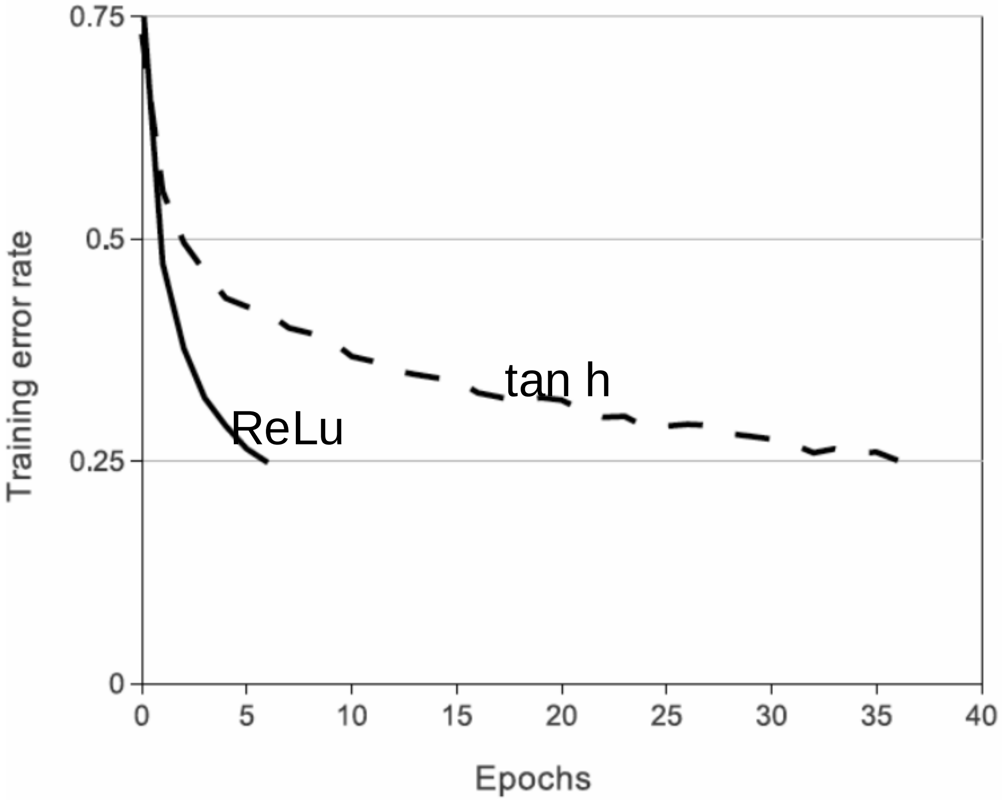
96 filters of the first layer

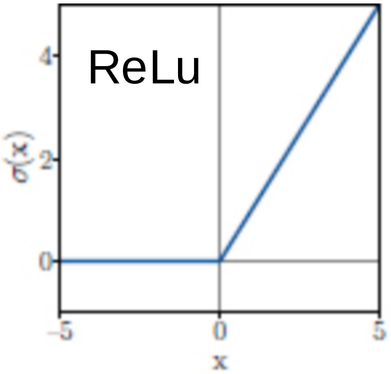


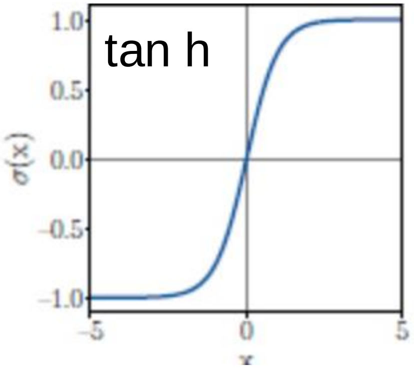
Lower layers of network learn simple shapes / rough structure

Higher layers of network learn specific features/detailed modification

AlexNet: activation function



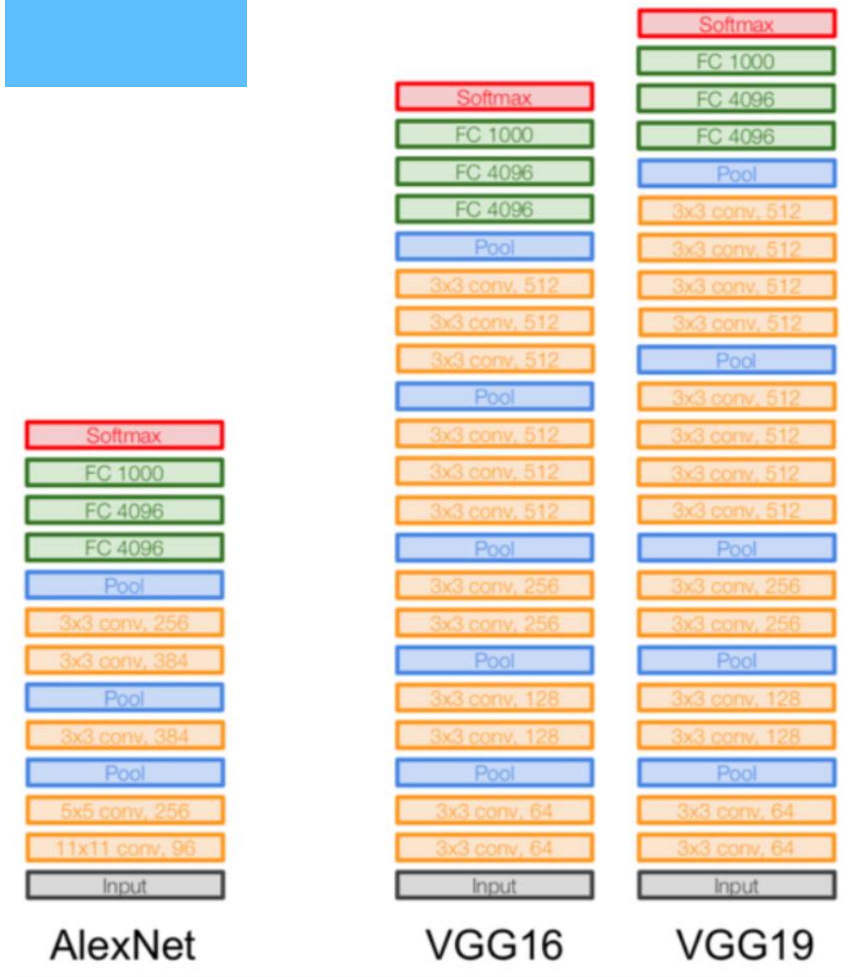




VGG net

Going deeper

- Much smaller filters
- Much deeper network
- Winner of ILSVRC2014 in localization, 2^{nd} in classification



Stack filters of 3x3 with stride 1 in 3 layers

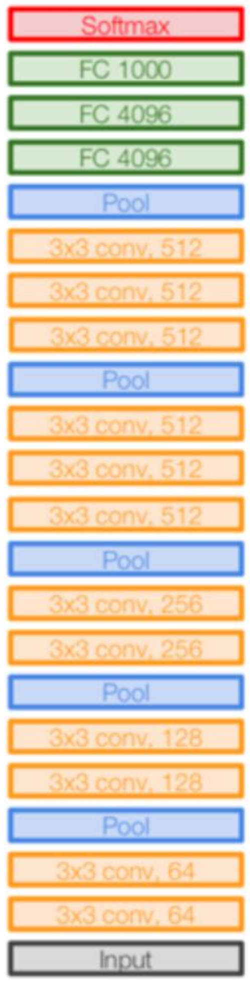
- deeper network possible due to smaller filters

Initialisation with paratemeters of pre-trained swallow network

Memory usage of VGG

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K
CONV3-512: [14x14x512] memory: 14*14*512=100K
CONV3-512: [14x14x512] memory: 14*14*512=100K
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image
TOTAL params: 138M parameters



VGG16

Memory usage of VGG

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K
CONV3-512: [14x14x512] memory: 14*14*512=100K
CONV3-512: [14x14x512] memory: 14*14*512=100K
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image
TOTAL params: 138M parameters

Largest **memory consumption** in intial layers (by feature maps)

Largest **number of parameters** in final dense layers

Summary

- Method outperforming fully connected feed-forward NN for image like data

- Performs hierarchical learning, explores local structures and translational invariance

- Low number of parameters (compared to fully connected deep neural networks) and significantly shorter training time

ML playgrounds

Machine Learning Playground

http://ml-playground.com/

ConvNetJS

Deep Learning in your browser

https://cs.stanford.edu/people/karpathy/convnetjs/

https://cs.stanford.edu/people/karpathy/convnetjs/

Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

https://playground.tensorflow.org/

Experiments with Google

COLLECTION

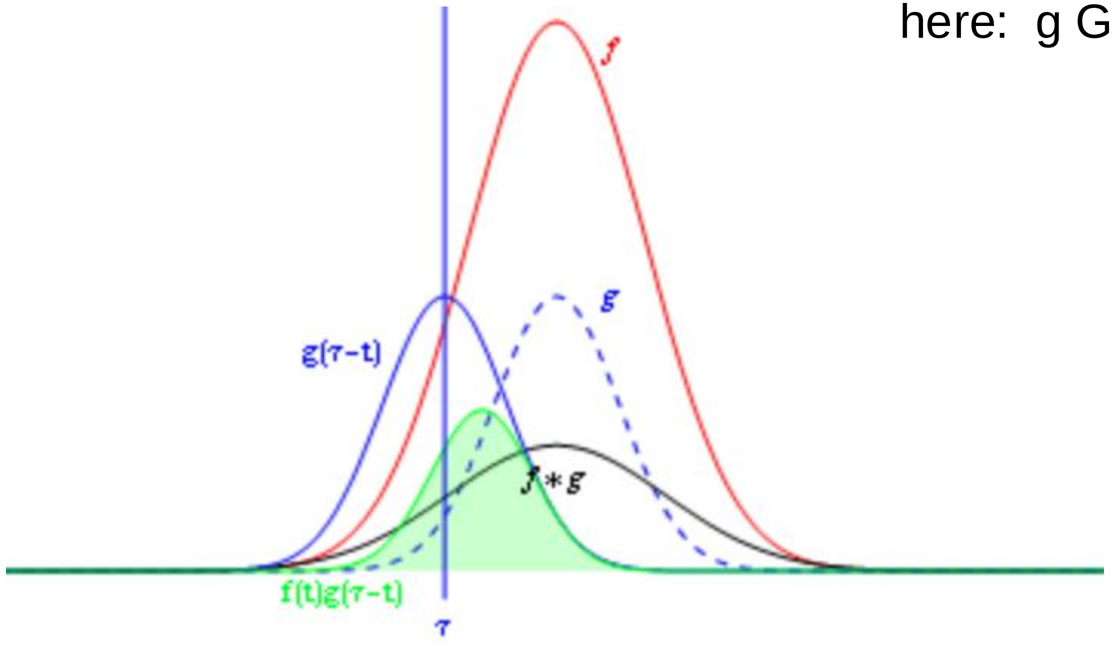
AI Experiments

https://experiments.withgoogle.com/collection/ai

Back-up

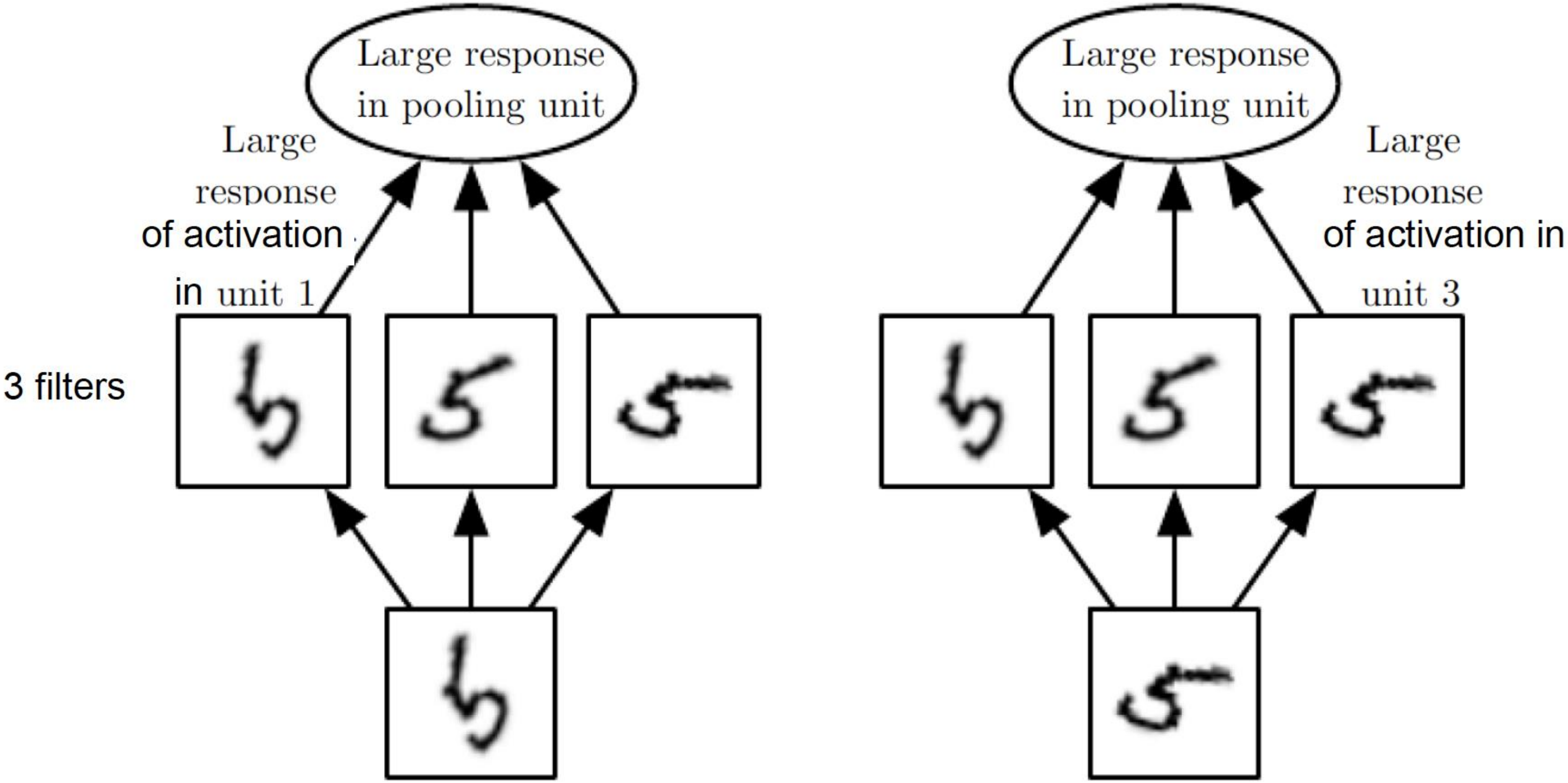
Convolution

here: g Gaussian



$$F(\tau) = f{*}g = \int f(t)\, g(\tau-t)\, dt$$

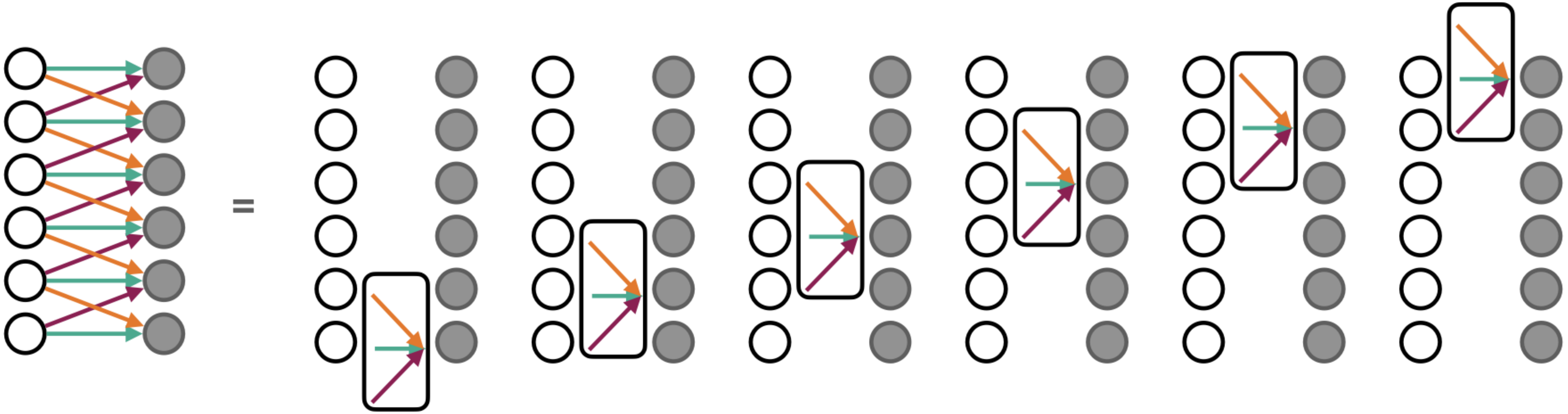
Invariance to local transformation



deeplearningbook.org

Convolutions

- Equivalent to a filter that slides across the inputs



Convolution: Gaussian smearing