Day 4 Lecture 1

Convolutional Neural Networks

Verónica Vilaplana
veronica.vilaplana@upc.edu

Associate Professor
Universitat Politecnica de Catalunya

Me

- Verónica Vilaplana Besler
  - Email: veronica.vilaplana@upc.edu
  - Office: UPC, Campus Nord, D5 118

- Teaching Experience
  - Signal Processing
  - Image Processing and Computer Vision
  - Machine Learning

- Research Experience
  - MSc Mathematics by Universidad de Buenos Aires (Argentina)
  - MSc Computer Sciences by Universidad de Buenos Aires (Argentina)
  - PhD on Image analysis by UPC (Spain)
  - Interest in machine and deep learning for medical imaging applications, segmentation and super-resolution in remote sensing

Index

- Motivation:
  - Local connectivity
  - Parameter sharing
- Layers
  - Convolutional
  - Pooling
  - Fully connected
  - Activation functions
  - Normalization

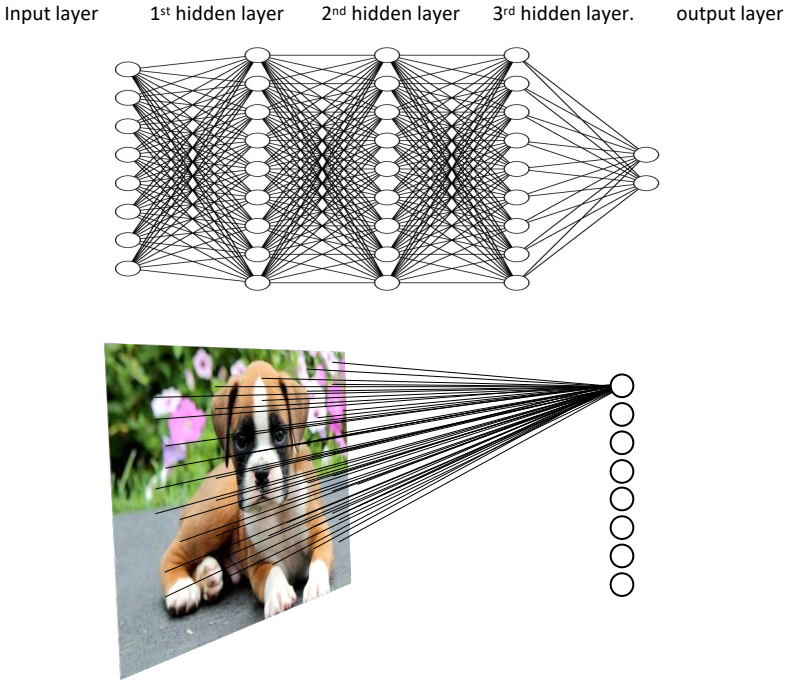
Motivation

Neural networks for visual data

- Example: Image classification
  - Given some input image, does the image contain a dog? (1/0)

image size 400x290

1 (yes)



each neuron connected to 116000 inputs

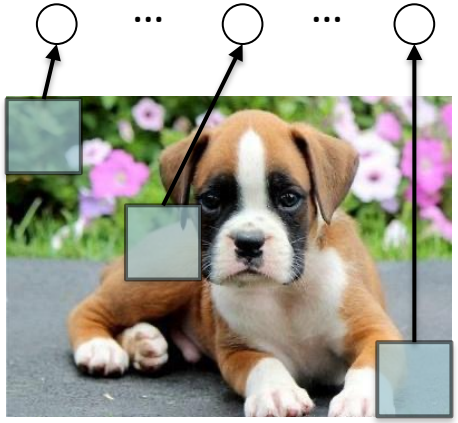
Neural networks for visual data

- We can design neural networks that are specifically adapted for such problems
  - must deal with very high-dimensional inputs
    - 400 x 290 pixels = 116000 inputs, or 3 x 116000 if RGB pixels
  - can exploit the 2D topology of pixels (or 3D for video / some medical data)
  - can build in invariance to certain variations we can expect (translations, illumination…)

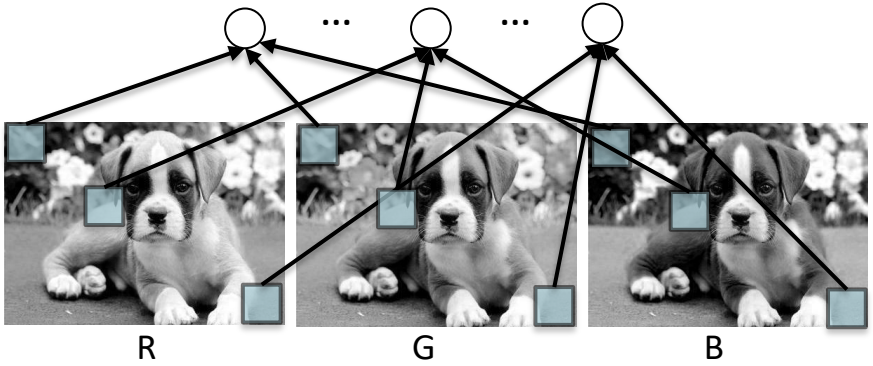
- CNNs are a specialized kind of neural network for processing data that has a known, grid-like topology. They leverage these ideas:
  - Sparsity of connections (local connectivity)
  - Parameter sharing

Convolutional neural networks

- First idea: local connectivity
  - each hidden unit is connected only to a subregion (patch) of the input image: receptive field
  - it is connected to all channels
    - 1 if greyscale image
    - 3 (R, G, B) for color image
    - …



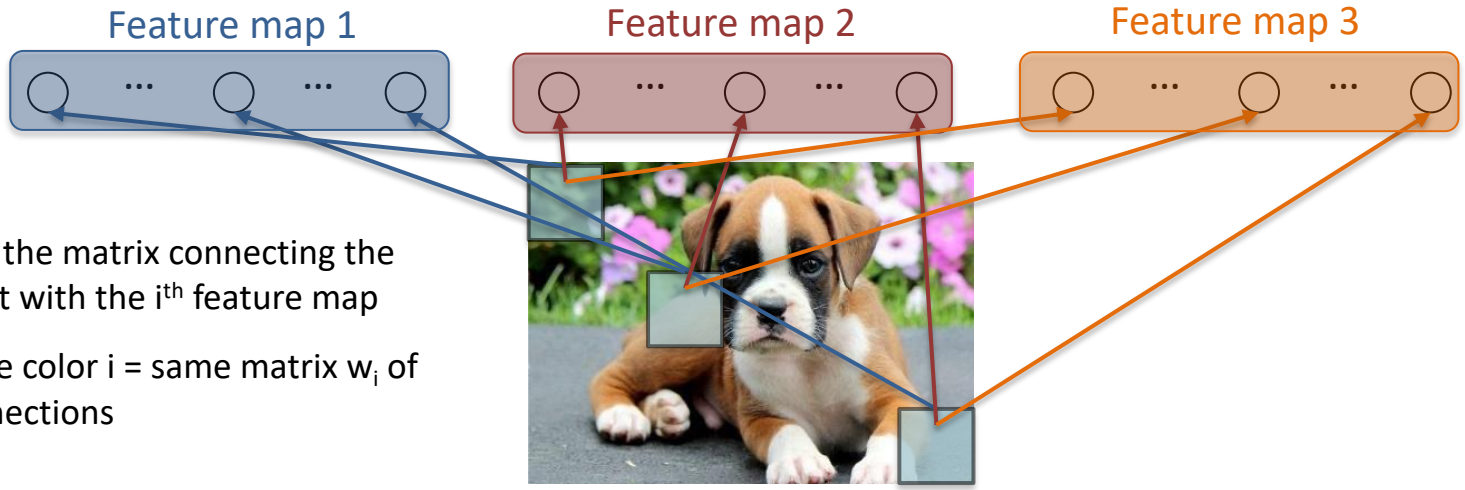
- Solves the following problems:
  - fully connected hidden layer would have an unmanageable number of parameters
  - computing the non-linear activations of the hidden units would be very expensive



Each unit connected to the same patch in the three (RGB) channels

Convolutional neural networks

- Second idea: parameter sharing across certain units
  - units organized into the same ‘‘feature map’’ share parameters
  - hidden units within a feature map cover different positions in the image
- Solves the following problems:
  - reduces even more the number of parameters
  - will extract the same features at every position (features are ‘‘equivariant’’)



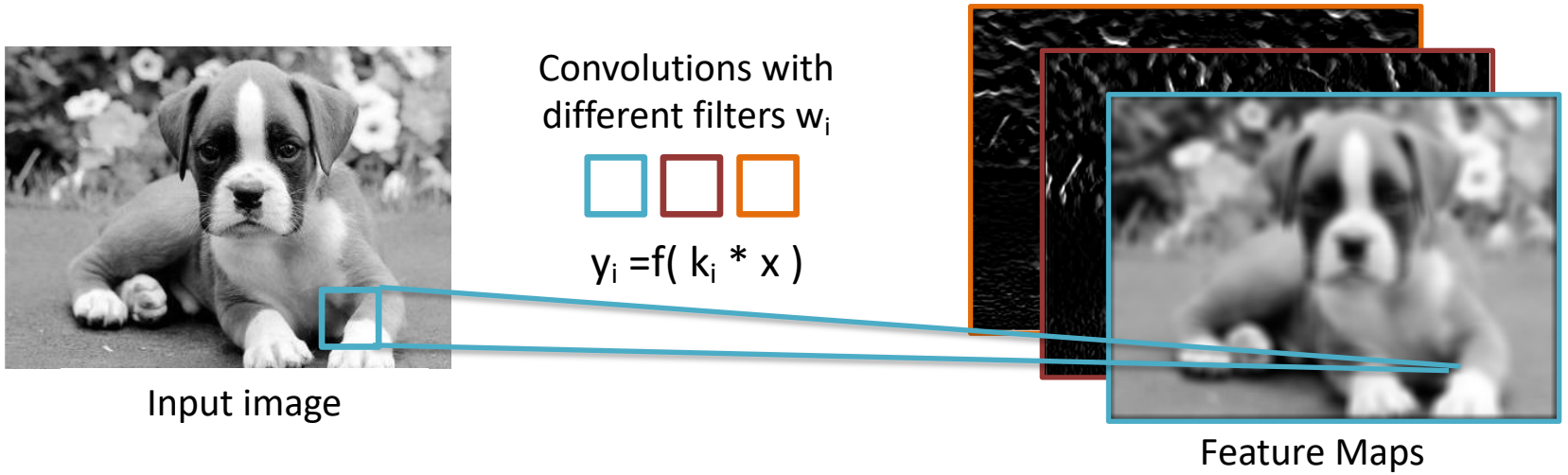
w_i is the matrix connecting the input with the i^{th} feature map

Same color i = same matrix w_i of connections

Convolutional neural networks

- Second idea: parameter sharing across certain units

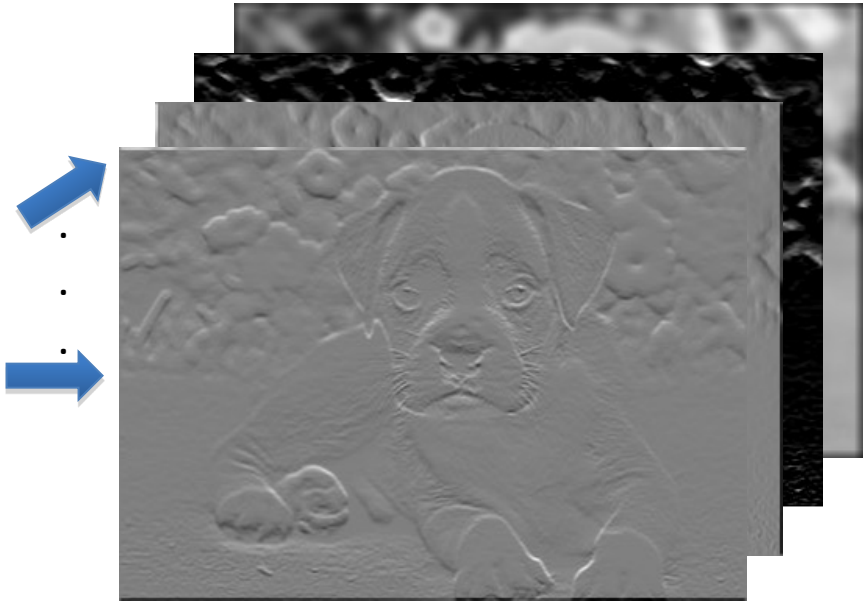
- Each feature map forms a 2D grid of features
  - can be computed with a discrete convolution of a kernel matrix k_i
    which is the hidden weights matrix w_i with its rows and columns flipped



Convolutional neural networks

- Convolution as feature extraction:
  applying a filterbank

Input

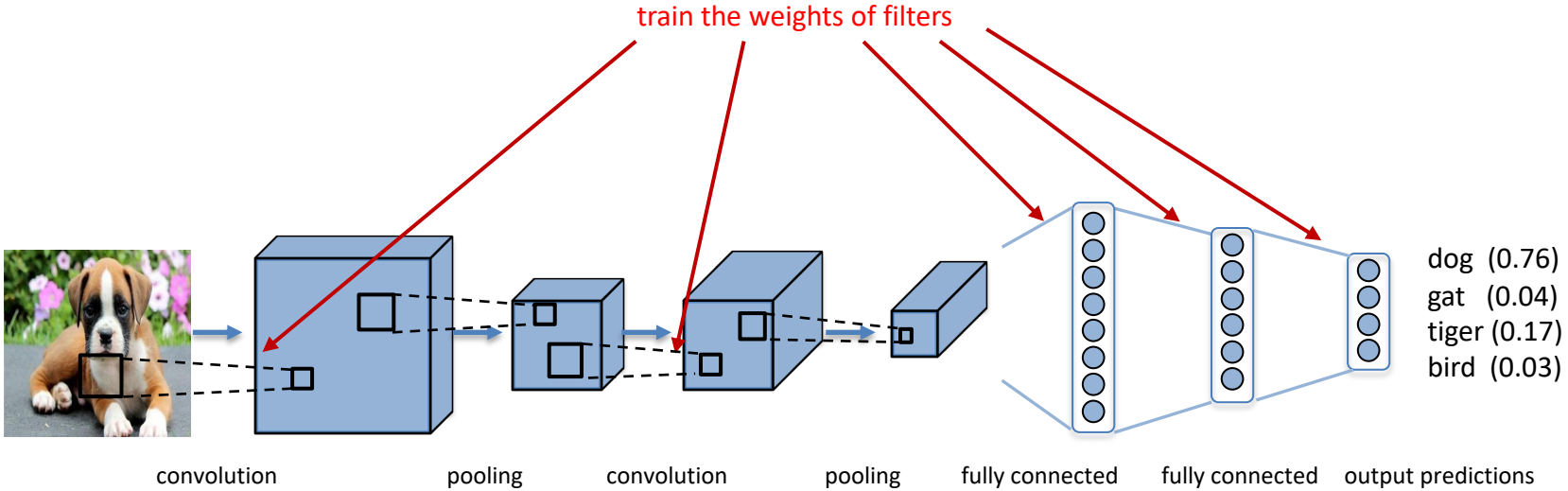


Feature Map

But filters are learned!

Convolutional Neural Networks

- Basic architecture: convolutional neural networks alternate between convolutional layers (followed by a nonlinearity) and pooling layers, with some final fully connected layers

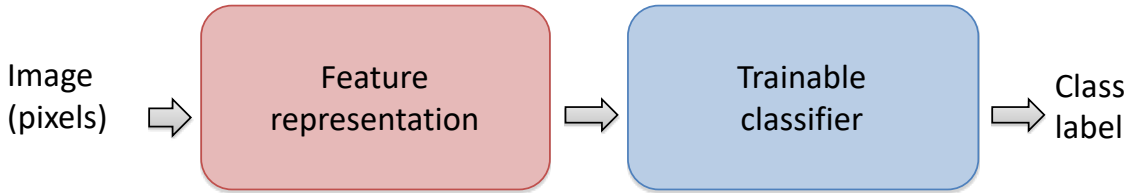


- For recognition: output layer is a regular, fully connected layer with softmax non-linearity
  - output provides an estimate of the conditional probability of each class
- The network is trained by stochastic gradient descent (& variants)

End to end learning

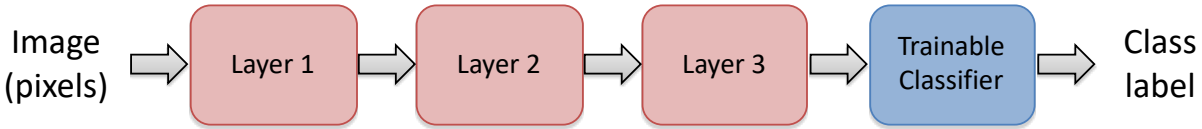
"Classic" recognition pipeline

- Hand-crafted feature representation (difficult / requires expert knowledge): ex. Histograms, BoW,…
- Off-the-shelf trainable classifier



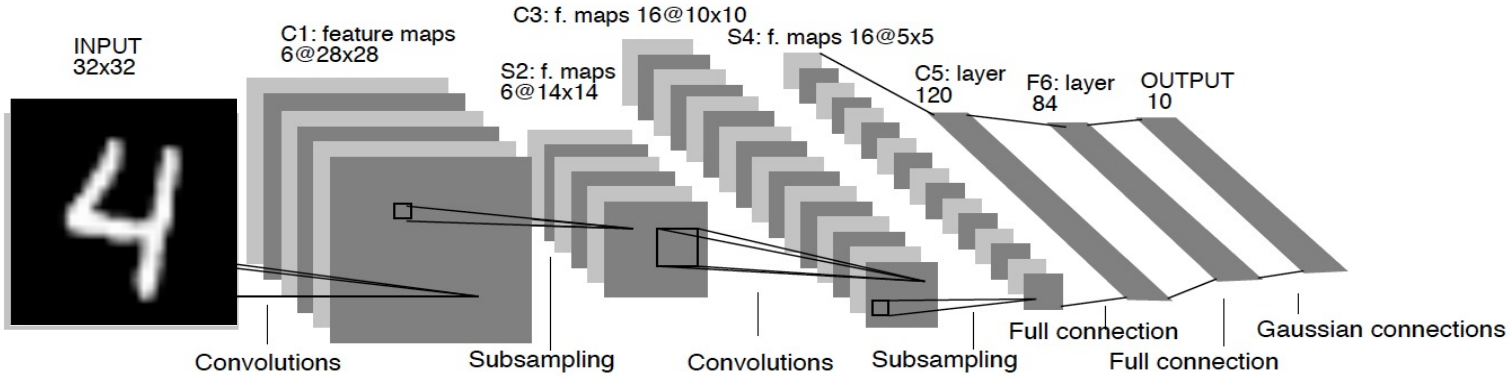
"CNN deep" recognition pipeline

- Learn a feature hierarchy from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly ("end to end training", joint optimization of features and classifier)



Example: LeNet-5

- LeCun et al., 1998



MNIST digit classification problem
handwritten digits

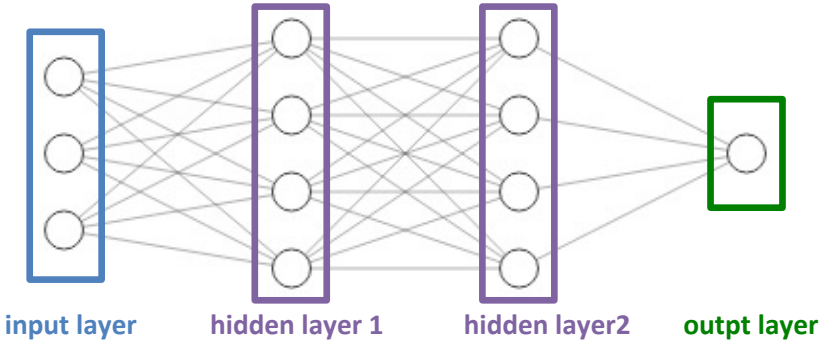
60,000 training examples
10,000 test samples
10 classes
28x28 grayscale images

Conv filters were 5x5, applied at stride 1
Sigmoid or tanh nonlinearity
Subsampling (average pooling) layers were 2x2 applied at stride 2
Fully connected layers at the end
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

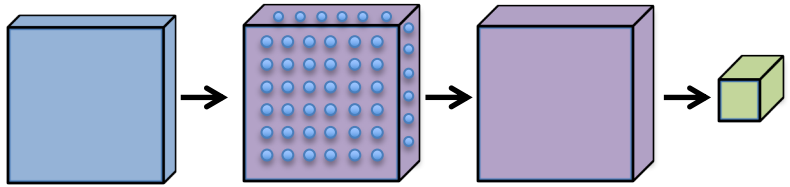
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, 1998.

Layers

Convolutional Neural Networks



A regular 3-layer Neural Network (MLP)



A CNN with 3 layers

- In a CNN inputs are images
- Neurons are arranged in 3 dimensions: width, height and depth
- CNN layers transform volumes into volumes

Convolutional layer

- 2D “convolution”

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

7x7

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

3x3 filter

Convolutional layer

- 2D “convolution”

1x1+3x2+0x1+2x0+1x0+1x0+5x(-1)+3x(-2)+2x(-1)= **-6**

1 (1)	3 (2)	0 (1)	1	4	3	2
2 (0)	1 (0)	1 (0)	2	5	0	1
5 (-1)	3 (-2)	2 (-1)	3	5	3	2
4	5	6	1	0	2	3
2	1	3	4	1	5	1
2	0	3	2	6	5	8
4	6	2	1	2	3	2

7x7

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

3x3 filter

=

-6				

5x5

Convolutional layer

- 2D “convolution”

3x1+0x2+1x1+1x0+1x0+2x0+3x(-1)+2x(-2)+2x(-1)= **-5**

1	3 (1)	0 (2)	1 (1)	4	3	2
2	1 (0)	1 (0)	2 (0)	5	0	1
5	3 (-1)	2 (-2)	2 (-1)	5	3	2
4	5	6	1	0	2	3
2	1	3	4	1	5	1
2	0	3	2	6	5	8
4	6	2	1	2	3	2

7x7

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

3x3 filter

=

-6	-5			

5x5

Convolutional layer

- 2D “convolution”

4x1+3x2+2x1+5x0+0x0+1x0+5x(-1)+3x(-2)+2x(-1)= -1

1	3	0	1	4 (1)	3 (2)	2 (1)
2	1	1	2	5 (0)	0 (0)	1 (0)
5	3	2	2	5 (-1)	3 (-2)	2 (-1)
4	5	6	1	0	2	3
2	1	3	4	1	5	1
2	0	3	2	6	5	8
4	6	2	1	2	3	2

7x7

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

3x3 filter

=

-6	-5	-5	-3	-1

5x5

Convolutional layer

- 2D “convolution”

2x1+1x2+1x1+5x0+3x0+2x0+4x(-1)+5x(-2)+6x(-1)= **-15**

1	3	0	1	4	3	2
2 (1)	1 (2)	1 (1)	2	5	0	1
5 (0)	3 (0)	2 (0)	2	5	3	2
4 (-1)	5 (-2)	6 (-1)	1	0	2	3
2	1	3	4	1	5	1
2	0	3	2	6	5	8
4	6	2	1	2	3	2

7x7

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

3x3 filter

=

-6	-5	-5	-3	-1
-15				

5x5

Convolutional layer

- 2D “convolution”

1x1+5x2+1x1+6x0+5x0+8x0+2x(-1)+3x(-2)+2x(-1)= **2**

1	3	0	1	4	3	2
2	1	1	2	5	0	1
5	3	2	2	5	3	2
4	5	6	1	0	2	3
2	1	3	4	1 (1)	5 (2)	1 (1)
2	0	3	2	6 (0)	5 (0)	8 (0)
4	6	2	1	2 (-1)	3 (-2)	2 (-1)

7x7

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

3x3 filter

=

-6	-5	-5	-3	-1
-15	-13	2	9	-1
6	-2	-1	4	1
15	10	-5	-16	-17
-11	0	6	3	2

5x5

Convolutional layer

- 2D “convolution”

In a CNN filter weights w_i are learned during training

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

7x7

Input NxN

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

3x3 filter

FxF filter

=

?	?			

5x5

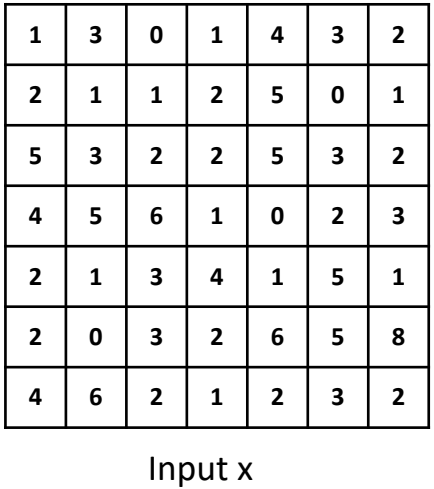
output is smaller than input:

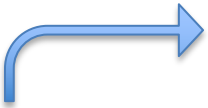
output size = 7-3+1 = 5

Output size = N – F + 1

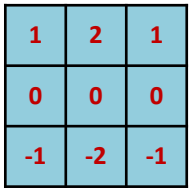
Convolutional layer

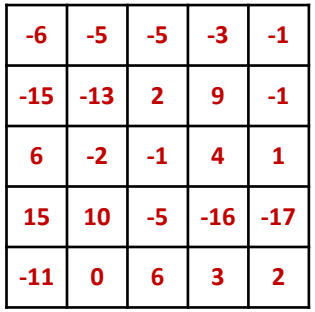
- Convolution vs cross-correlation



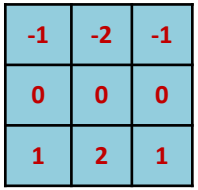


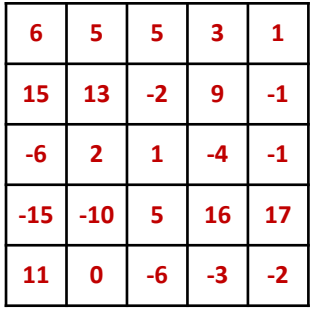
cross-correlation
x[m,n]*h[-m,-n]





convolution
x[m,n]*h[m,n]
(h horizontal and vertical flip)



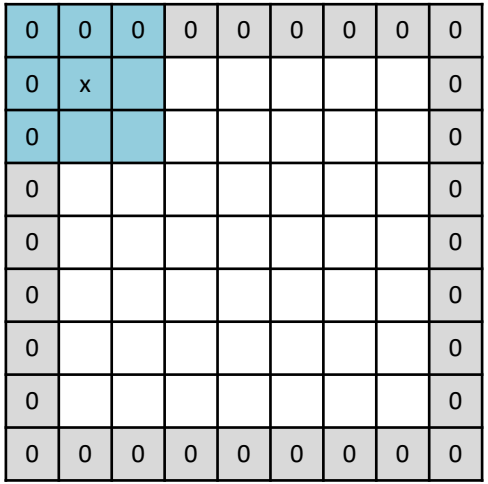


Convolutional layer

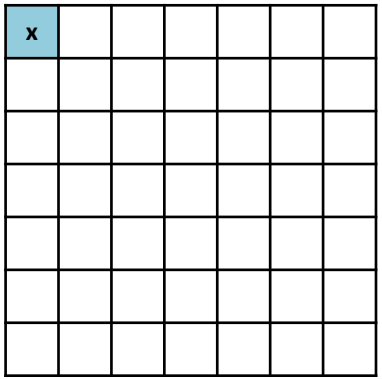
- Padding

Sometimes it is convenient to add zeros around the input border so that the input and output width and height are the same

In general, filters FxF, zero-padding with P= (F-1)/2 to **preserve size spatially**



zero-padding in the border
7x7



padding P=1, N=7, F=3
output size 7-3+2+1=7

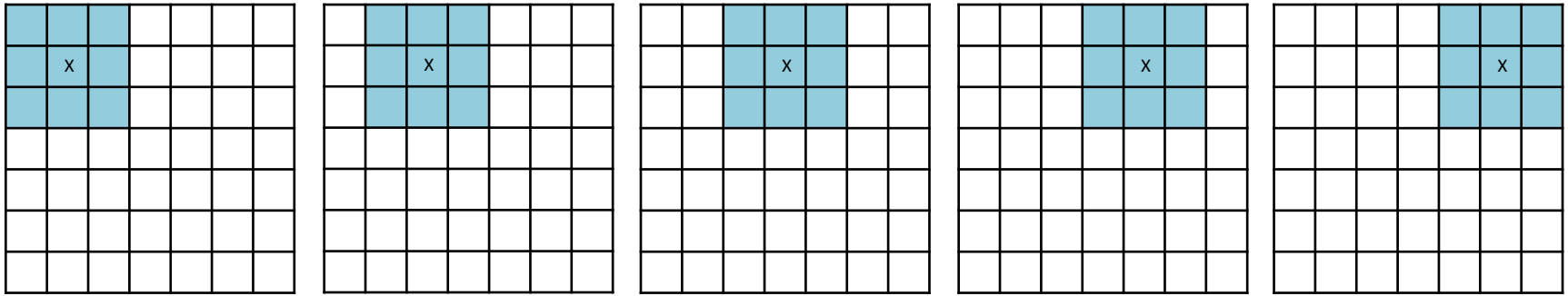
With padding P
Output size: **N – F + 2P +1**

Convolutional layer

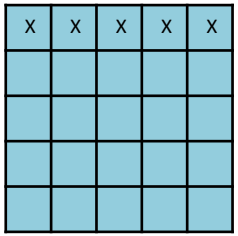
- Stride

  Is the number of pixels by which we slide the kernel over the input matrix.

  Larger stride produces smaller feature maps.



stride 1:
7x7 input (spatially)
a 3x3 filter: 5x5 output

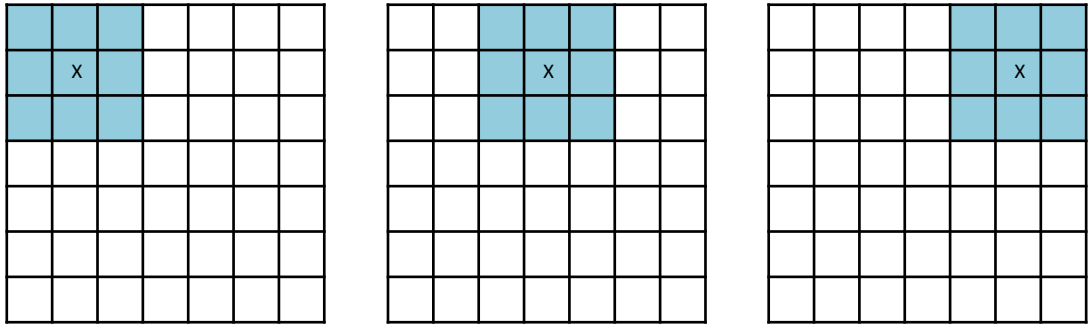


Convolutional layer

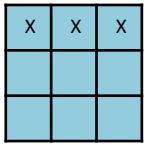
- Stride

  Is the number of pixels by which we slide the kernel over the input matrix.

  Larger stride produces smaller feature maps.



stride 2:
7x7 input (spatially)
a 3x3 filter: 3x3 output

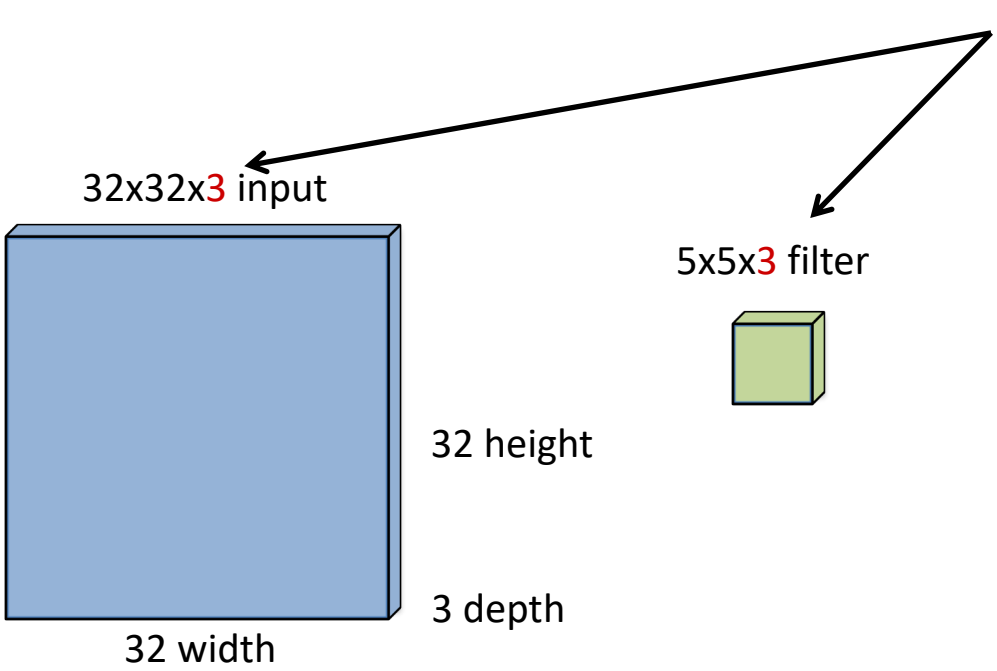


No padding (P=0), stride S
Output size: **(N-F)/ S + 1**

With padding P, stride S
Output size: **(N-F+2P)/S + 1**

Convolutional layer

- Convolution on a volume

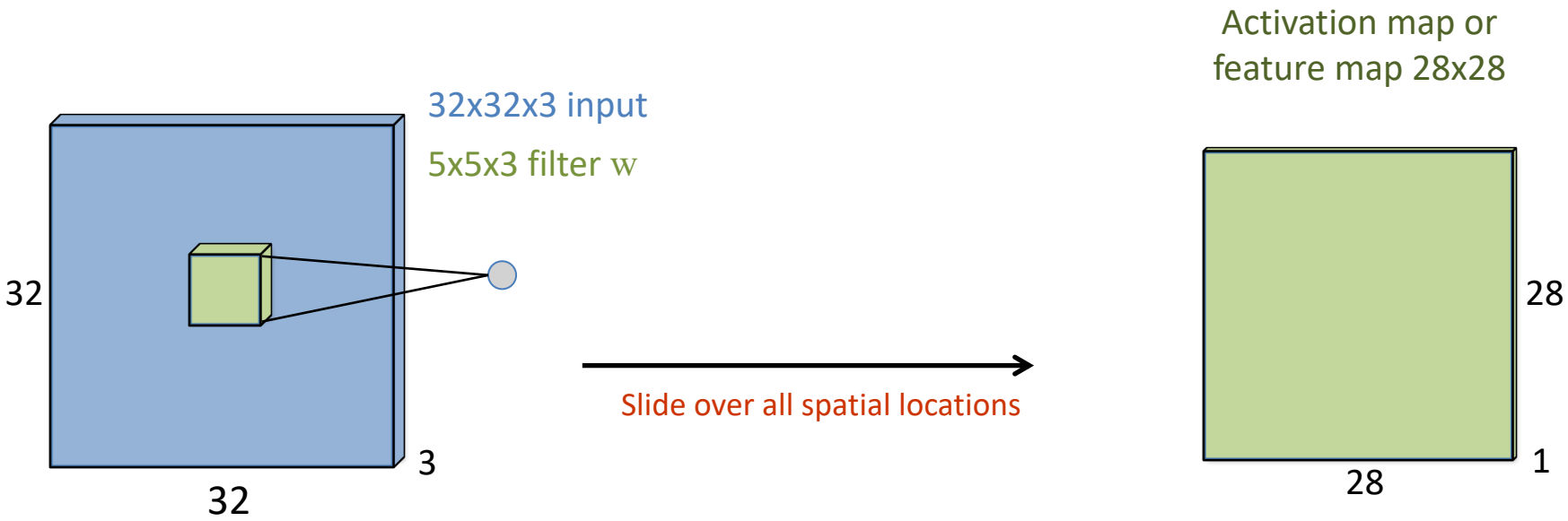


Convolve the filter with the input
i.e. slide over the input spatially, computing dot products

Example: color image (3 channels)

Convolutional layer

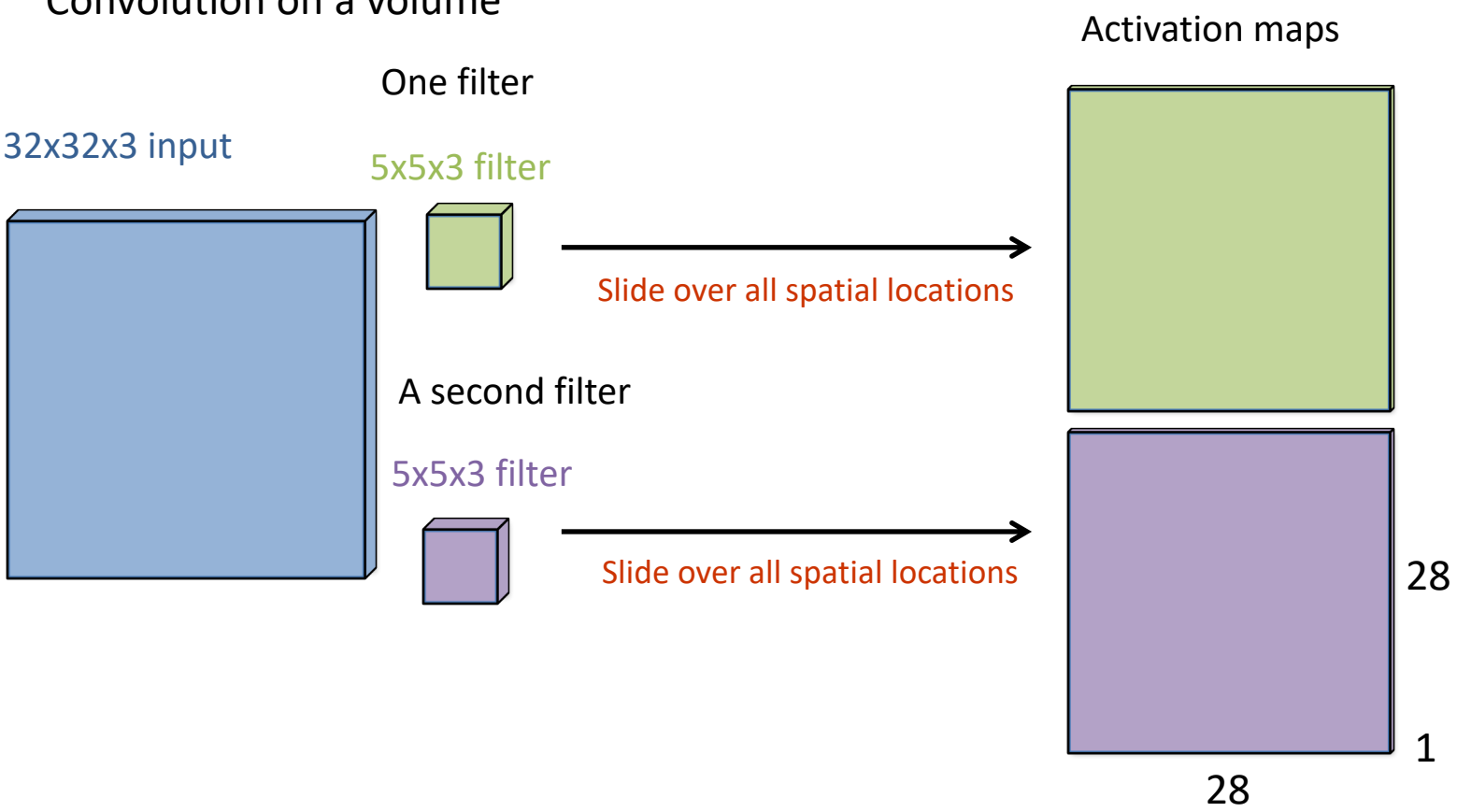
- Convolution on a volume



Each number is the result of the dot product between the filter and a small 5x5x3 patch of the input: 5x5x3=75-dim dot product + bias $w^t x+b$

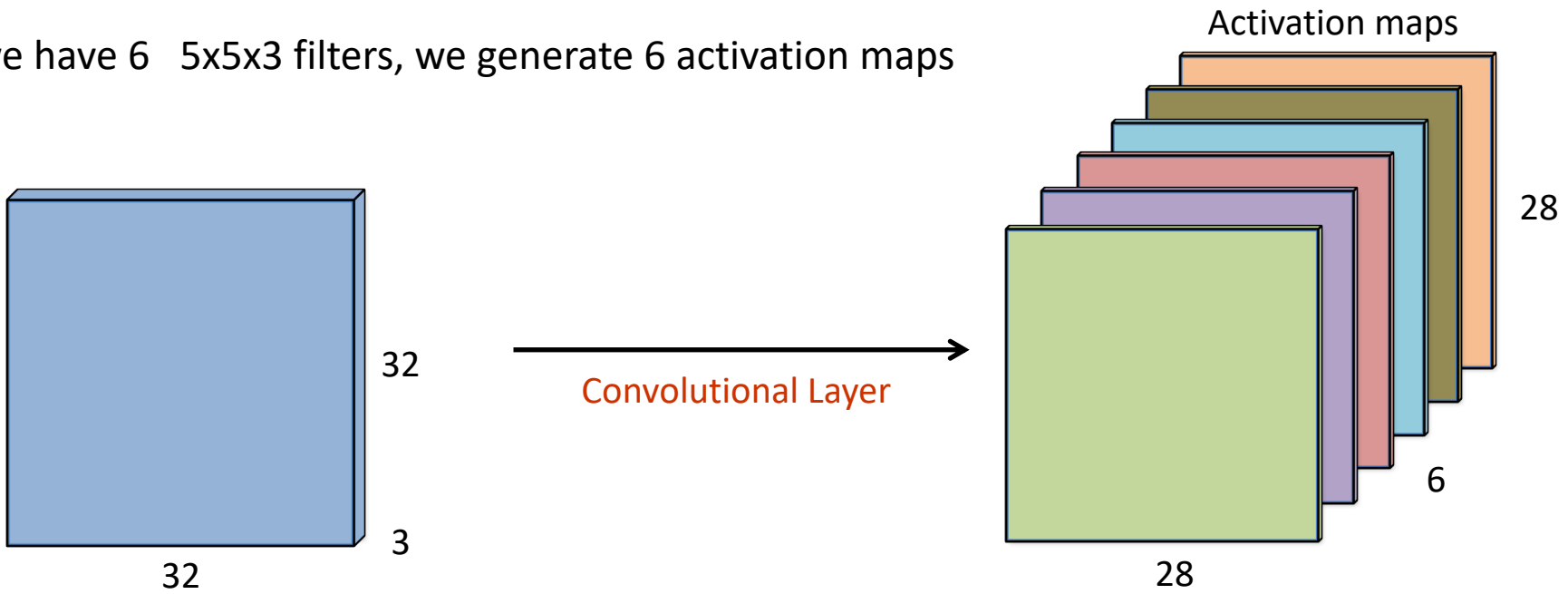
Convolutional layer

- Convolution on a volume



Convolutional layer

If we have 6 5x5x3 filters, we generate 6 activation maps

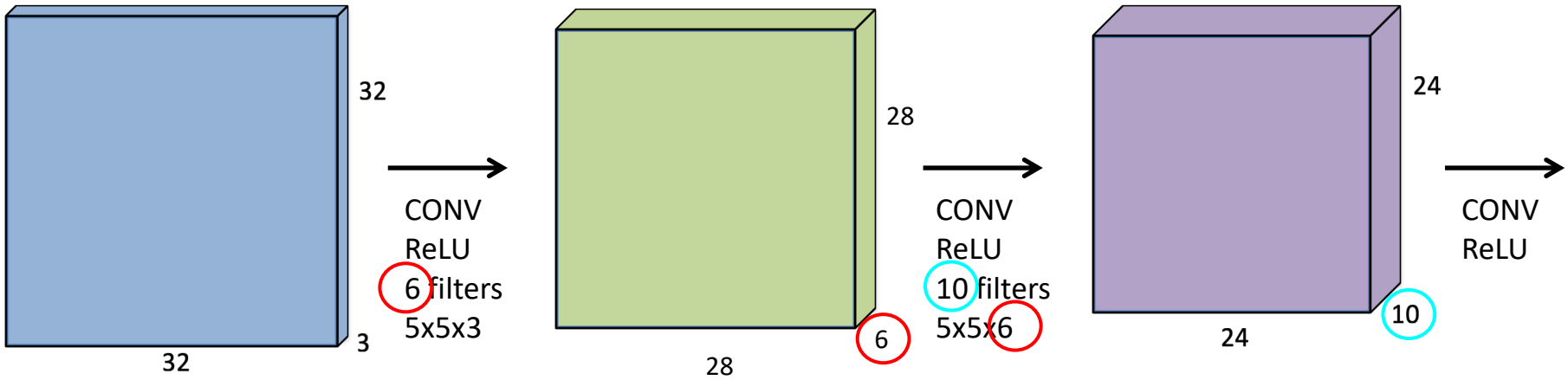


Maps are stacked to generate a new volume of size 28x28x6

Applying a bank of filters to an input (3D matrix) produces a volume in which each slice is an output of convolution with one filter.

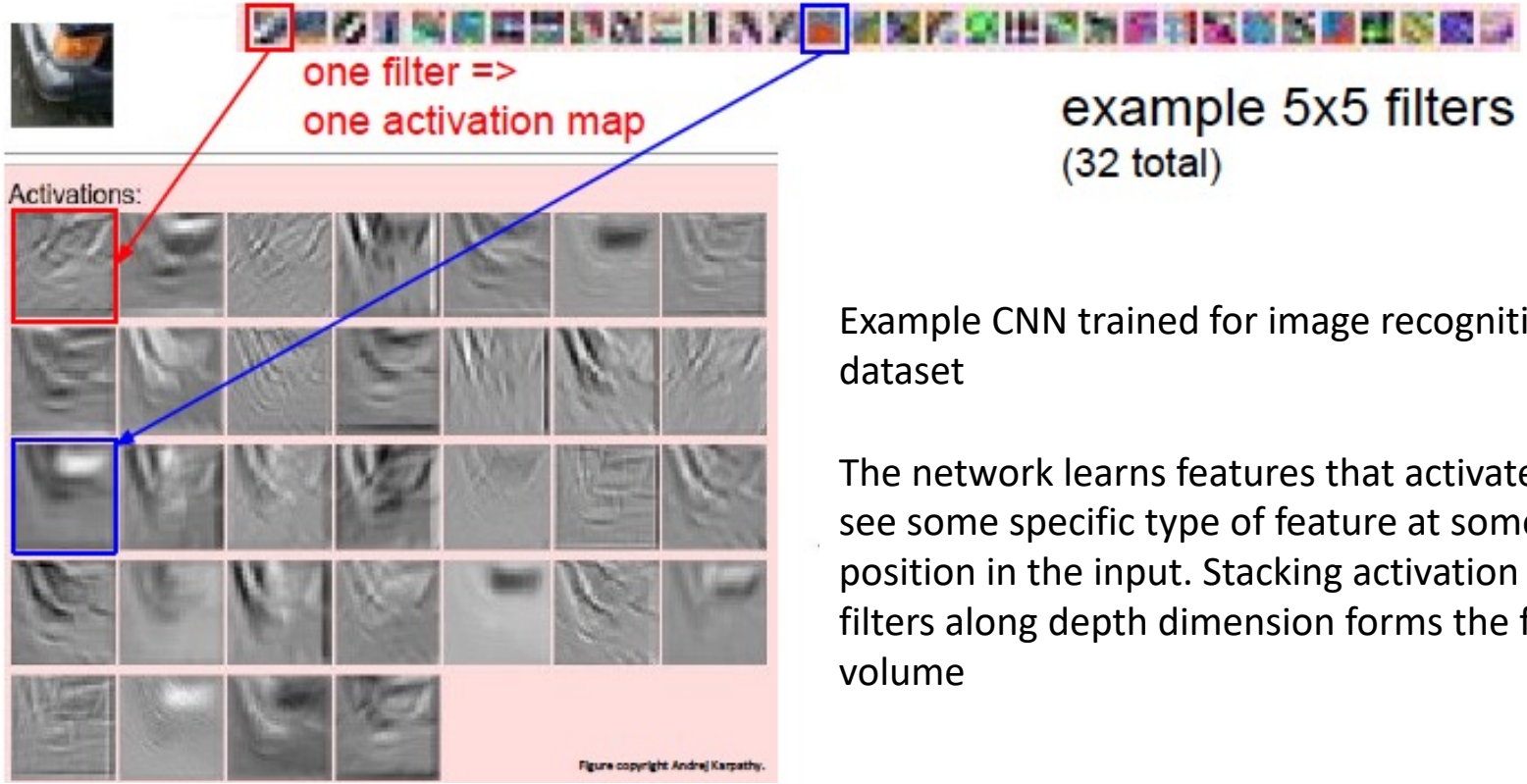
Convolutional layer

ConvNet is a sequence of Convolutional Layers, interspersed with activation functions and pooling layers (and a small number of fully connected layers)



Each convolutional layer contains some filters; they are applied to the output volume of the previous layer. Each convolution produces a slice in the new volume.

Example: filters and activation maps



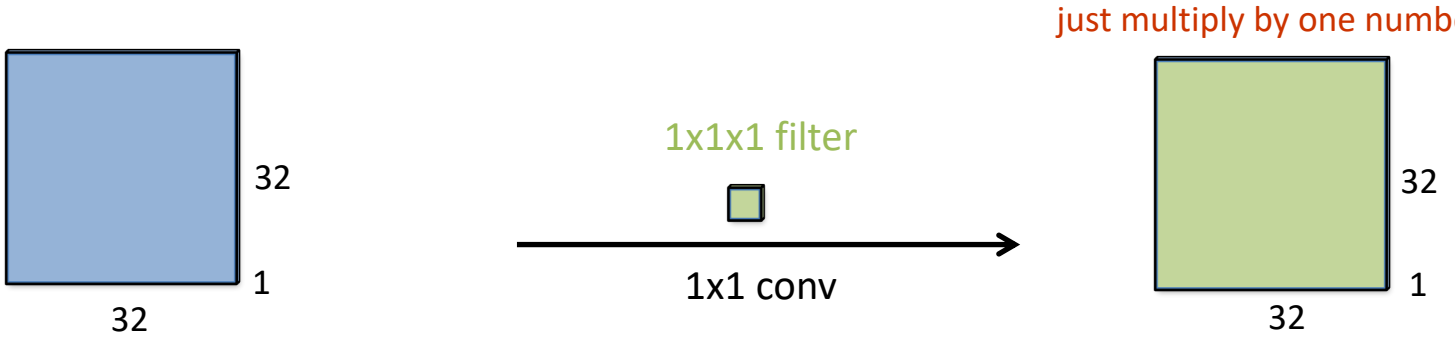
Example CNN trained for image recognition on CIFAR dataset

The network learns features that activate when they see some specific type of feature at some spatial position in the input. Stacking activation maps for all filters along depth dimension forms the full output volume

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

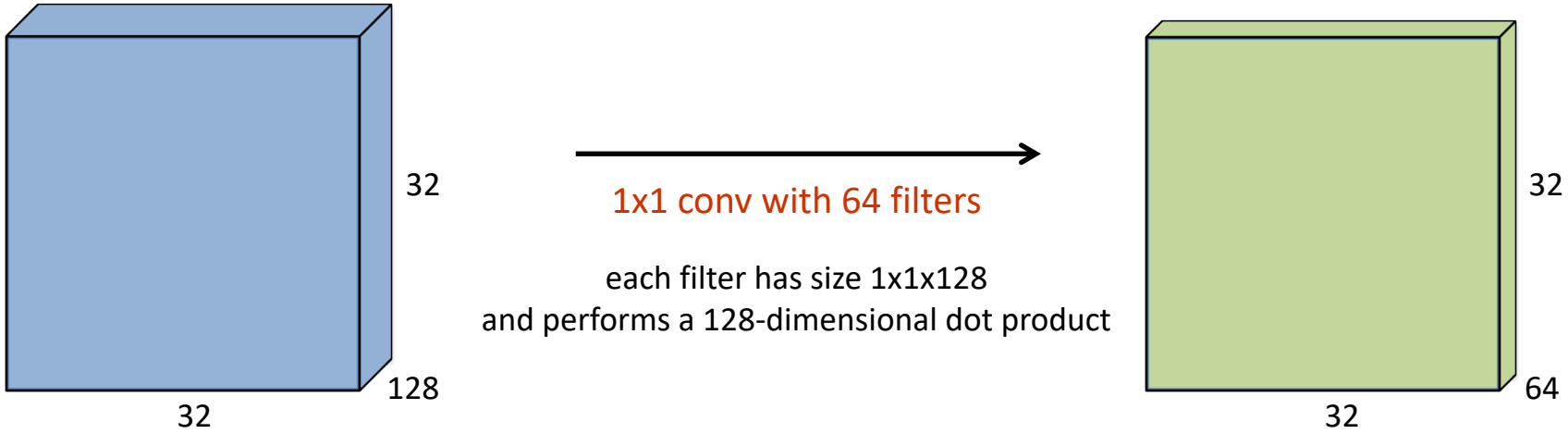
1x1 convolutions

What does a 1x1 convolution do?

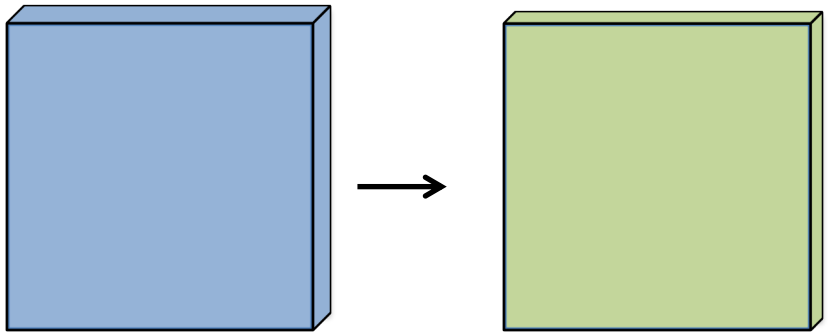


1x1 convolutions

1x1 convolution layers are used to reduce dimensionality (number of feature maps)



Exercise: size, parameters

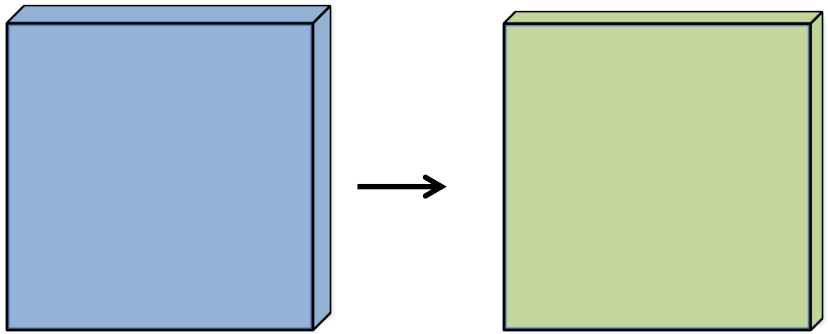


What is the output volume size?

How many parameters does this layer have?

Input volume: 32x32x3
10 5x5 filters, stride 1, padding 2

Exercise: size, parameters



Input volume: 32x32x3
10 5x5 filters, stride 1, padding 2

What is the output volume size?
(N + 2P – F) / S + 1
(32+2*2-5)/1+1= 32 spatially so 32x32x10

How many parameters does this layer have?
each filter has 5x5x3 + 1 =76 params (+1 for bias)
-> 76x10 = 760 parameters

Summary: convolutional layer

- Input volume size W1 x H1 x D1
- Four hyperparameters:
  - number of filters K
  - kernel size F
  - stride S
  - amount of zero padding P
- Output volume size W2 x H2 x D2
  - W2 = (W1 – F + 2P) / S + 1
  - H2 = (H1 – F + 2P) / S + 1
  - D2 = K
- Parameters: (F x F x D1) x K weights and K biases
- In the output volume, the d^{th} depth slice is the result of performing a valid convolution of the d^{th} filter over the input volume with a stride of S, and then offset by d^{th} bias

Exercises

1. Suppose your input is a 300x300 color (RGB) image and you are not using a convolutional network. If the first hidden layer has 100 neurons, each one fully connected to the input, how many parameters does this hidden layer have (including the bias parameter)?
   a) 9,000,001
   b) 9,000,100
   c) 27,000,001
   d) 27,000,100

2. Suppose your input is a 300x300 color (RGB) image and you use a convolutional layer with 100 filters, each 5x5. How many parameters does this layer have, including bias parameters?
   a) 2501
   b) 2600
   c) 7500
   d) 7600

Exercises

3. You have an input volume that is 63x63x16 and convolve it with 32 filters that are each 7x7, with stride 2 and no padding. What is the size of the output volume?
   a) 16x16x32
   b) 29x29x16
   c) 29x29x32
   d) 16x16x16

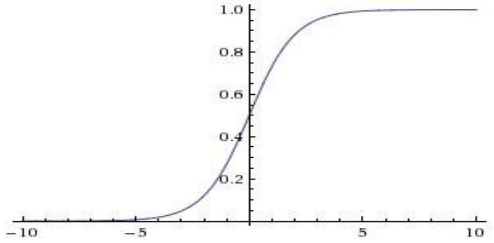
4. You have an input volume that is 63x63x16 and convolve it with 32 filters that are each 7x7, with stride 1. You want the output with the same size. What is the padding?
   a) 1
   b) 2
   c) 3
   d) 7

Activation functions

- Desirable properties: mostly smooth, continuous, differentiable, fairly linear

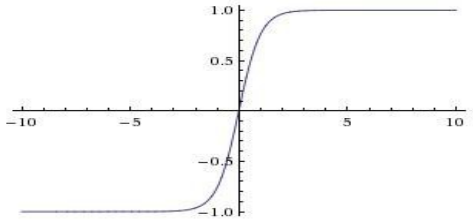
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



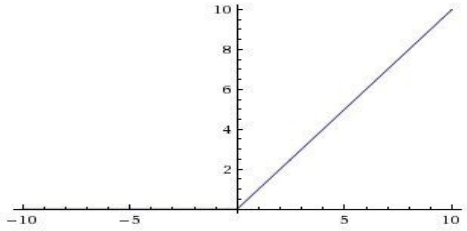
tanh

$$\sigma(x) = \tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



ReLu

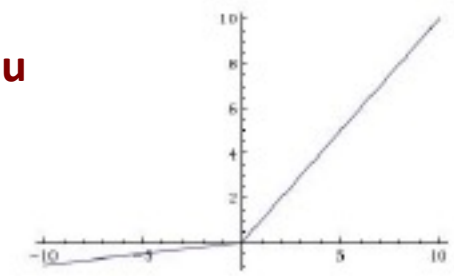
$$\sigma(x) = \max(0, x)$$



Leaky ReLu / PRelu

$$\sigma(x) = \max(0.01x, x)$$

$$\sigma(x) = \max(\alpha x, x)$$

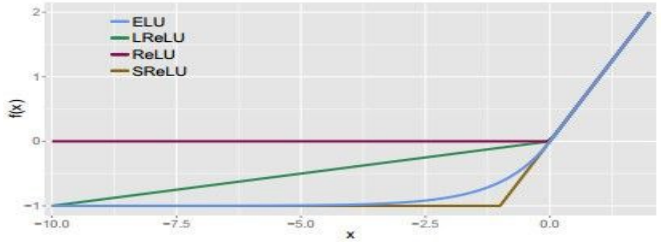


Maxout

$$\sigma(x) = \max(w_1^T x + b_1, w_2^T x + b_2)$$

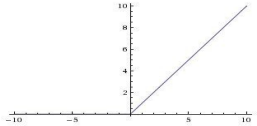
ELU

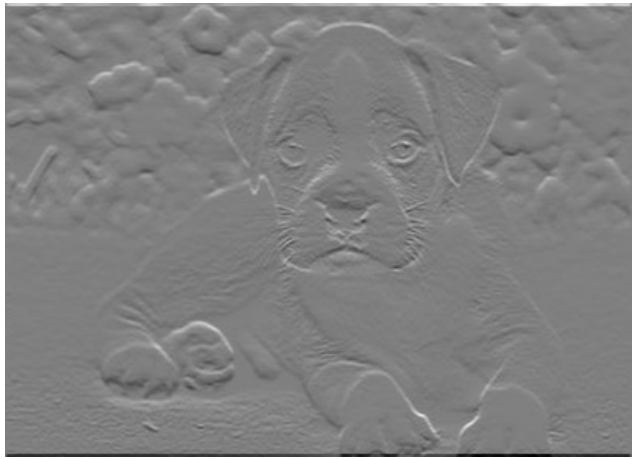
$$\sigma(x) = \begin{cases} x & x > 0 \\ \alpha(e^{x} - 1) & x \le 0 \end{cases}$$



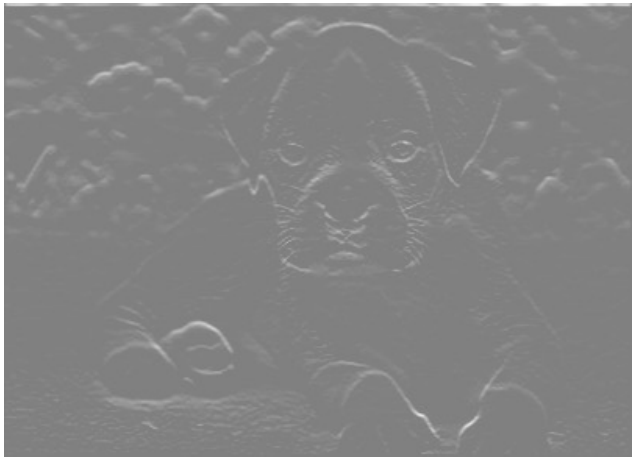
Activation functions

- Example ReLu





ReLu

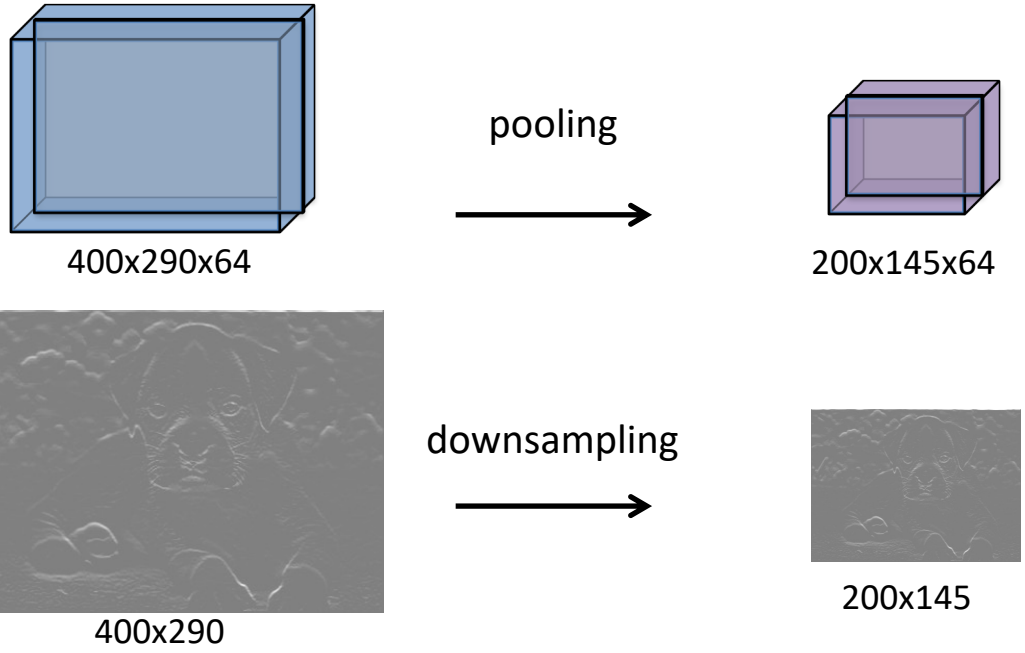


negative values (<128), positive values (≥128)

Only non-negative values

Pooling layer

- Makes the representations smaller and more manageable for later layers
- Useful to get invariance to small local changes
- Operates over each activation map independently



Pooling layer

- Max pooling

single depth slice

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

max pool
with 2x2 filter and stride 2

4	9	6
3	6	3

Pooling layer

- Average pooling

single depth slice

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

average pool
with 2x2 filter and stride 2

2	4	3.5
2	3.25	1.25

Summary: pooling layer

- Input volume size W1xH1xD1
- Two hyperparameters
  - spatial extent F
  - stride S
- Output volume size W2xH2xD2
  - W2 = (W1-F) / S + 1
  - H2 = (H1-F) / S + 1
  - D2 = D1
- Introduces **zero parameters** since it computes a fixed function of the input

Pros:
- reduces the number of inputs to the next layer, allowing us to have more feature maps
- invariant to small translations of the input

Cons:
- after several layers of pooling we have lost information about precise position of things

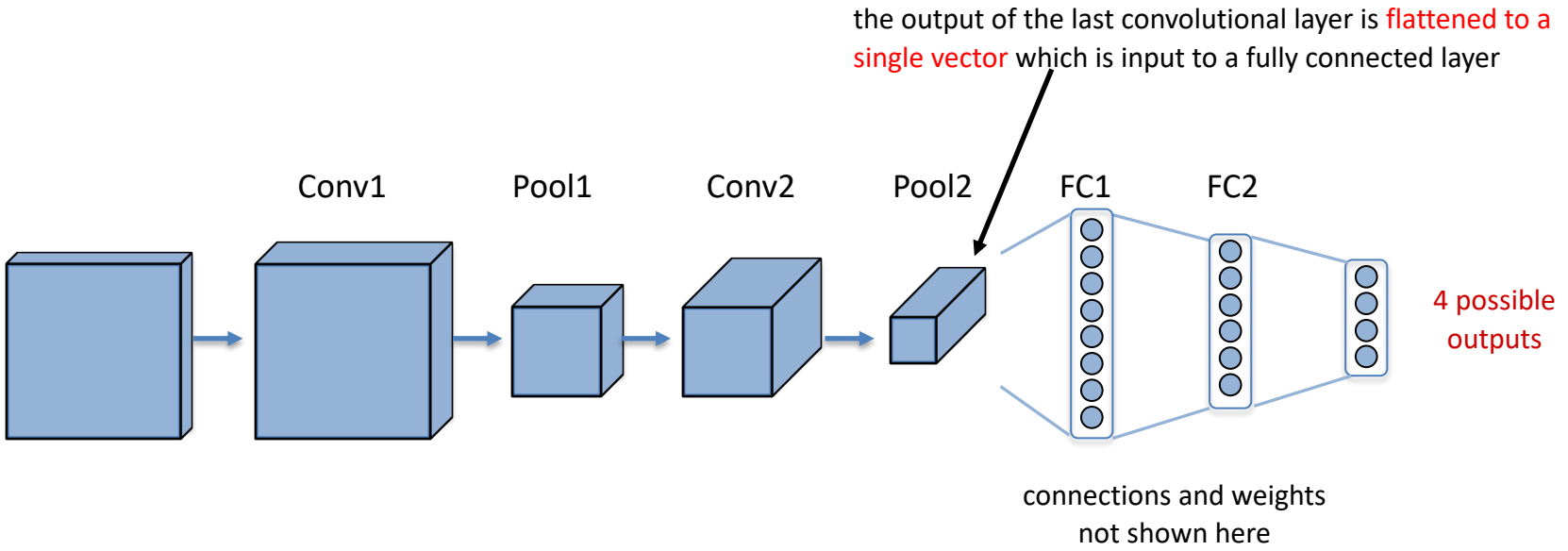
Exercises

5. You have an input volume that is 32x32x16 and apply max pooling with a stride of 2 and a filter size of 2. What is the output size?
   a) 32x32x8
   b) 15x15x16
   c) 16x16x16
   d) 116x16x8

6. You have an input volume that is 15x15x8 and pad it with ‘P=2’. What is the dimensión of the resulting volume after padding?
   a) 19x19x12
   b) 17x17x10
   c) 19x19x8
   d) 17x17x8

Fully connected layer

- In the end it is common to add one or more fully (or densely) connected layers.
- Every neuron in the previous layer is connected to every neuron in the next layer (as in regular neural networks). Activation is computed as **matrix multiplication** plus bias
- At the output, softmax activation for classification



Normalization layer

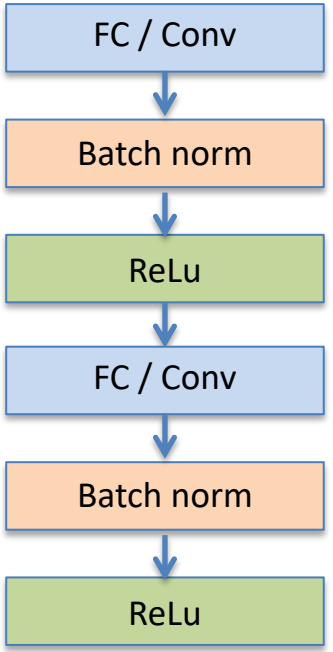
- As learning progresses, the distribution of the layer inputs changes due to parameter updates
- This can result in most inputs being in the non-linear regime of the activation function, slowing down learning
- One technique used to reduce this effect is **batch normalization**
  - Explicitly force the layer activations to have zero mean and unit variance (mean and variance per channel, for all samples in the minibatch)

$$\hat{x}^{(k)} = \frac{x^{(k)} - E(x^{(k)})}{\sqrt{\text{var}(x^{(k)})}}$$

  - Adds a learnable scale and bias term to allow the network to still use the nonlinearity

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

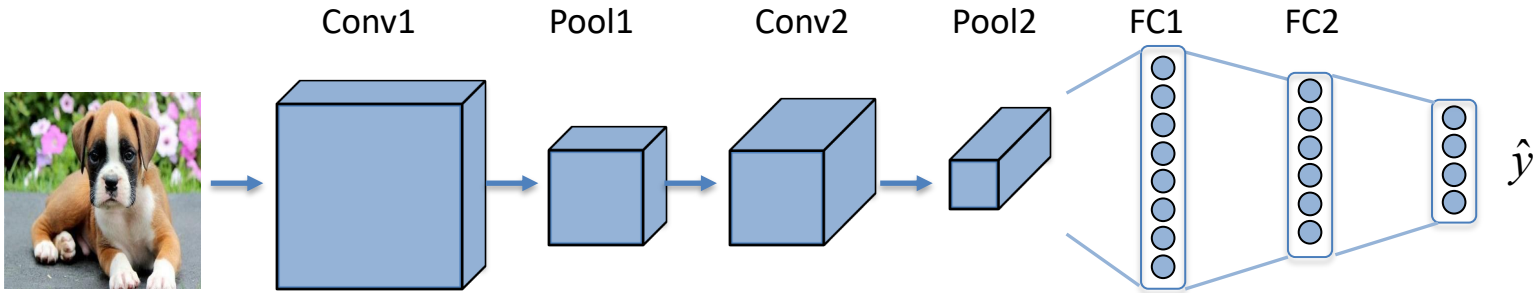
- There are other normalization strategies



Ioffe and Szegedy, 2015. "Batch normalization: accelerating deep network training by reducing internal covariate shift"

Training a convolutional network

- Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})$



- Cost $J = \frac{1}{N} \sum_{i=1}^{N} L(y^{(i)}, \hat{y}^{(i)})$

- Use gradient descent to optimize parameters to reduce J
  - backpropagation is used similarly as in a fully connected network

Exercise

- Consider the following CNN. Complete the table with the volume sizes and number of parameters
  input - Conv1 - Pool1 - Conv2 - Pool2 - FC1 - FC2 - Out(softmax)

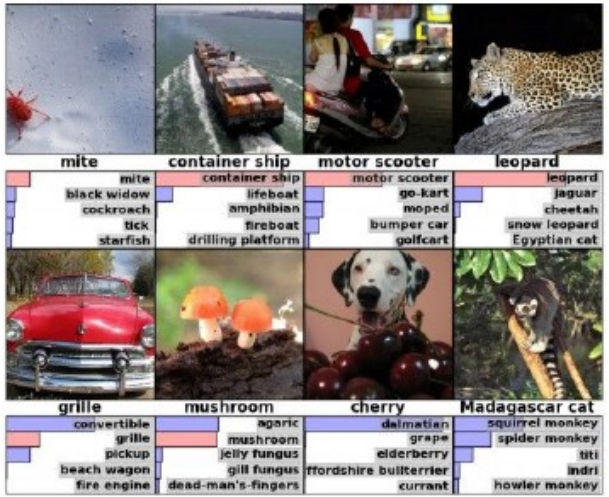
Layers	Activation shape	# Parameters
Input RGB 32x32	(32x32x3)	
Conv1 F=5, S=1, P=0, 8 filters		
Pool1 F=2, S=2		
Conv2 F=5,S=1,P=0, 16 filters		
Pool2 F=2, S=2		
FC1 120 units		
FC2 84 units		
Output 10 units		

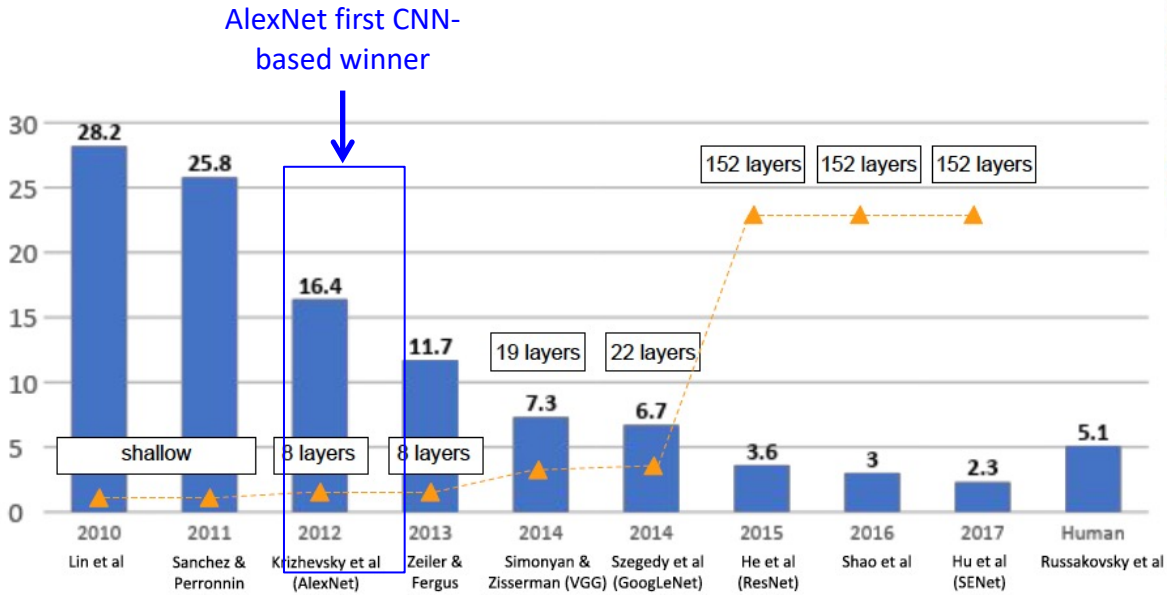
Some architectures

for Visual Recognition

Example ImageNet

Large Scale Visual Recognition Challenge
Image Classification
1000 object clases
Images: 1.2 M train. 100k test.



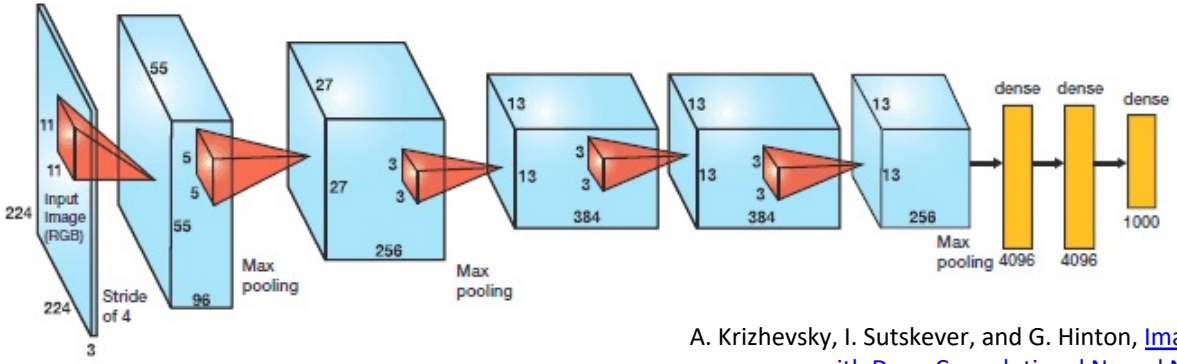


Metric
Top 5 error rate
- Algorithm predicts at most 5 classes in descending order of confidence
-If the correct class is among the first 5 predictions, error is 0, otherwise is 1

www.image-net.org/challenges/LSVRC/

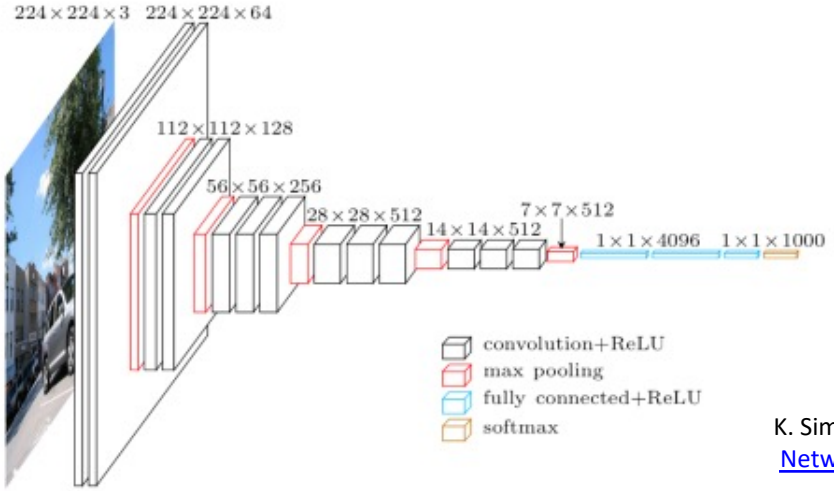
Some CNN architectures used in ImageNet

- AlexNet (2012)



A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

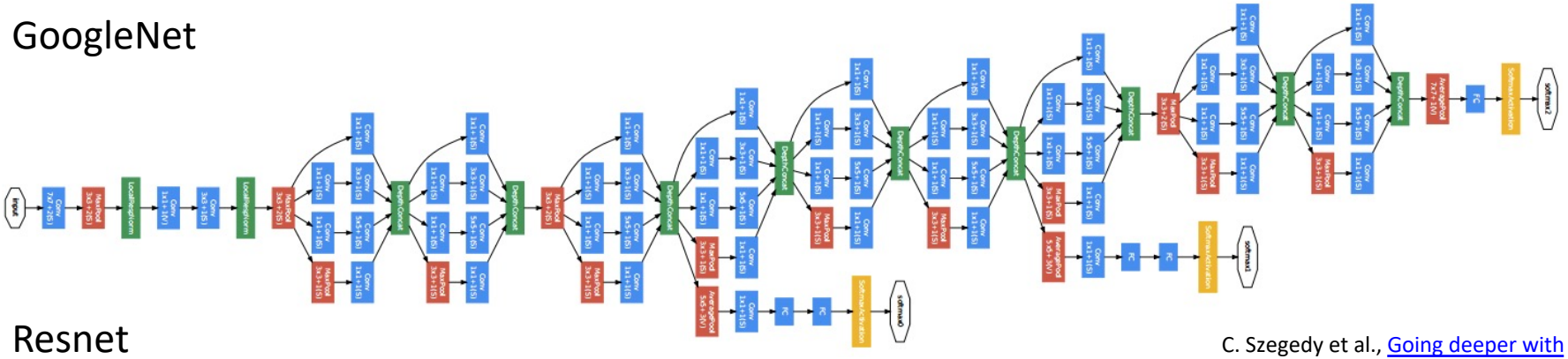
- VGG Net (2014)



K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

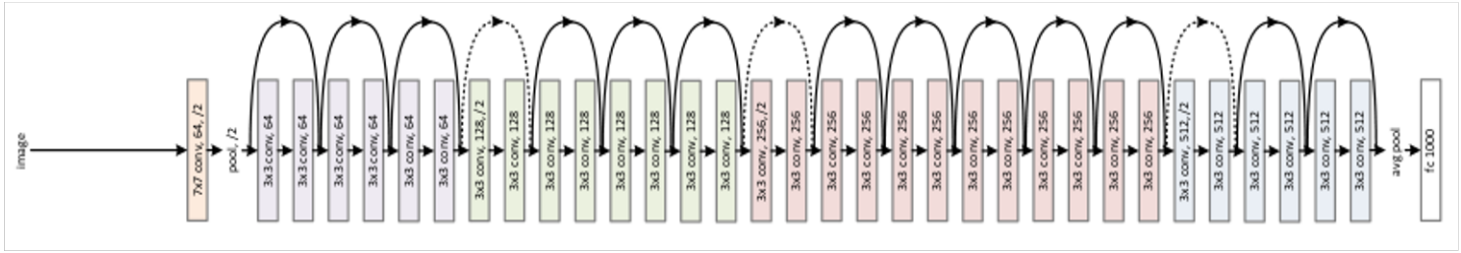
Some CNN architectures used in ImageNet

- GoogleNet



- Resnet

C. Szegedy et al., Going deeper with convolutions, CVPR 2015



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

Summary

- Convolutional neural networks are a specialized kind of neural network for processing data that has a grid-like topology

- CNNs leverage these ideas:
  - local connectivity
  - parameter sharing

- Layers: convolutional, non-linear activation, pooling

- Architectures for object recognition in images
  - LeNet:, AlexNet, VGG, GoogLeNet, ResNet

Soluctions

- Previous exercises: 1 – d) ; 2 – d) ; 3 – c) ; 4 – c) ; 5 – c) ; 6 c)å

- Consider the following CNN. Complete the table with the volume sizes and number of parameters
  input - Conv1 - Pool1 - Conv2 - Pool2 - FC1 - FC2 - Out(softmax)

Layers	Activation shape	# Parameters
Input RGB 32x32	(32,32,3)	0
Conv1 F=5, S=1, P=0, 8 filters	(28,28,8)	(5x5x3+1)x8 = 608
Pool1 F=2, S=2	(14,14,8)	0
Conv2 F=5,S=1,P=0, 16 filters	(10,10,16)	(5x5x8+1)x16 = 3216
Pool2 F=2, S=2	(5,5,16)	0
FC1 120 units	(120,1)	(5x5x16)x120+120 = 48120
FC2 84 units	(84,1)	120x84+84 = 10164
Output 10 units	(10,1)	84x10+84 = 84