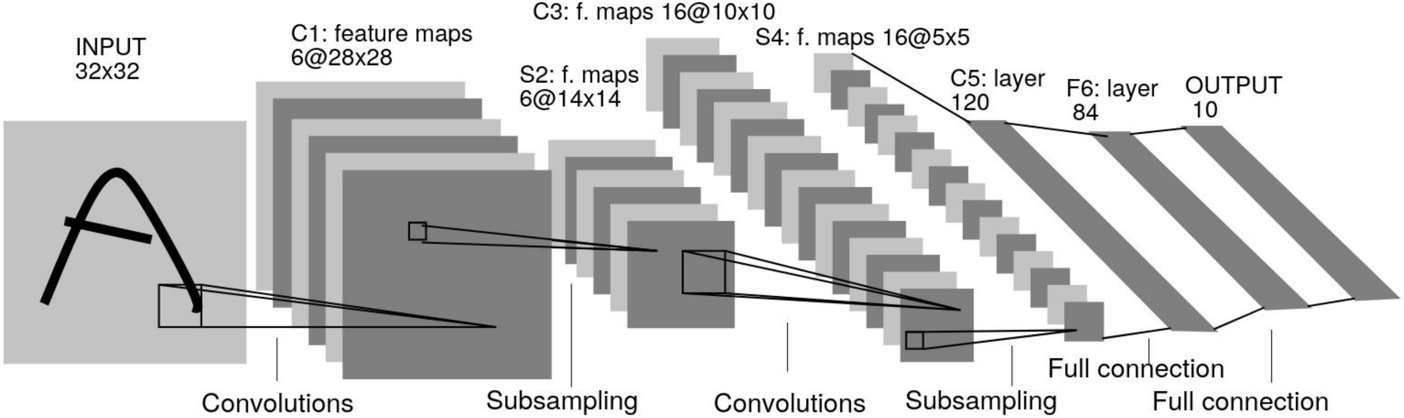
Convolutional Neural Networks



Stephan Alaniz
stephan.alaniz@telecom-paris.fr

Recap

Data: $(\mathbf{x}_1, y_1) \dots (\mathbf{x}_n, y_n)$

Network: $f(x, \mathbf{w})$

Loss: $\mathcal{L}(f(x_i, \mathbf{w}), y_i) \;= \mathcal{L}_i(\mathbf{w})$

Total objective: $E(\mathbf{w}) = \sum_{i=1}^{N} \mathcal{L}_i(\mathbf{w})$

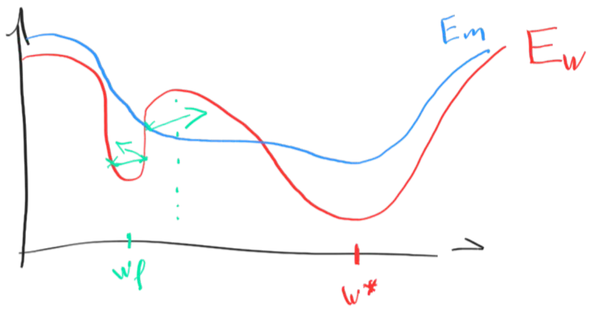
Gradient Descent: $\mathbf{w}' = \mathbf{w} - \eta \frac{\partial E}{\partial \mathbf{w}}$

Stochastic Gradient Descent

Data: $B_m = \{(\mathbf{x}_1, y_1) \dots (\mathbf{x}_m, y_m)\}$ with $m < n$

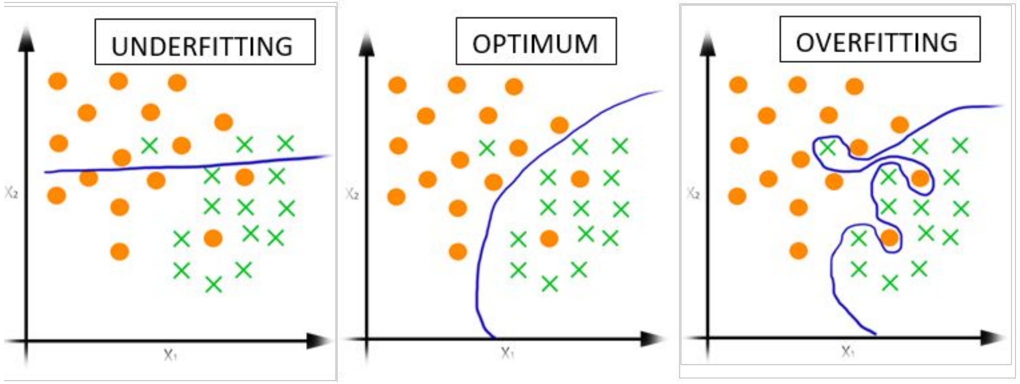
Objective: $E_m(\mathbf{w}) = \sum_{i \in B_m} \mathcal{L}_i(\mathbf{w})$

SGD:
- select random samples B_m
- $\mathbf{w}' = \mathbf{w} - \eta \frac{\partial E_m}{\partial \mathbf{w}}$

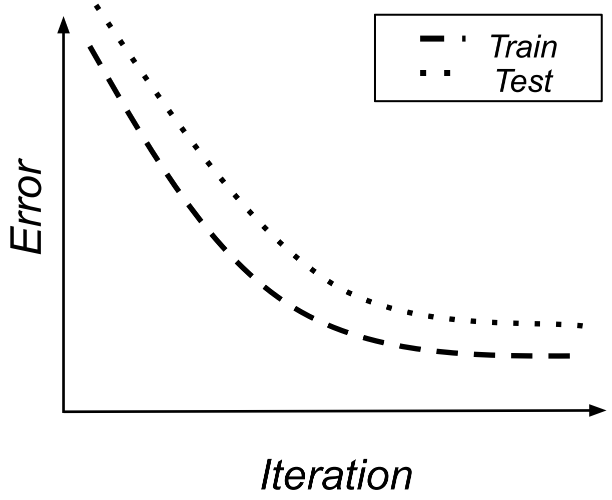


Overfitting

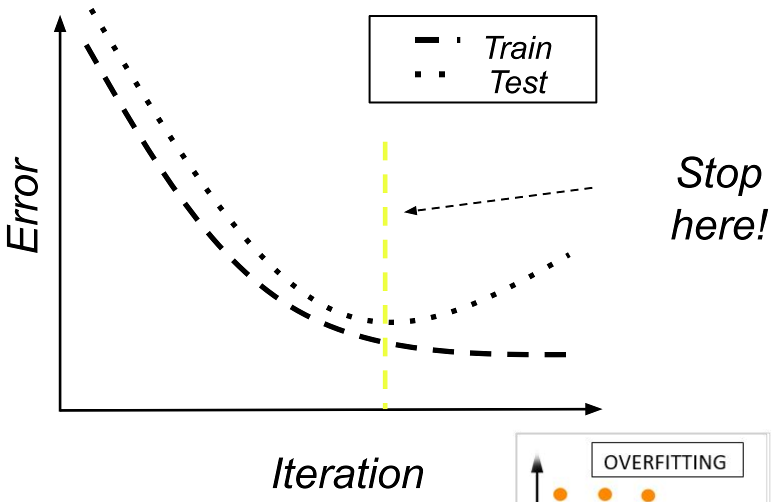
- Example problem: binary classification
  - Non linearly-separable
- As we add complexity we better fit samples
  - We learn to classify the train samples right
  - Poor *generalization* ability



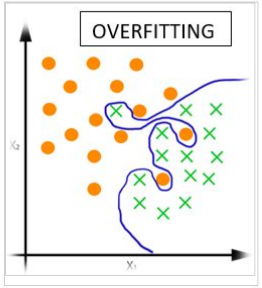
Detecting Overfit



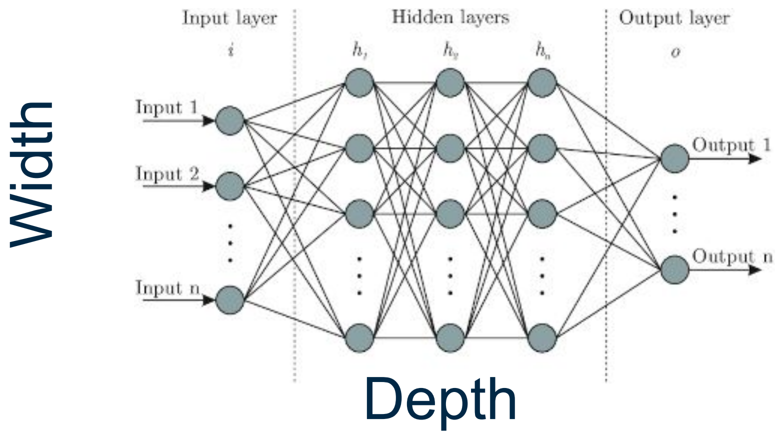
underfit



Overfit



Last time: Multi-layer perceptron

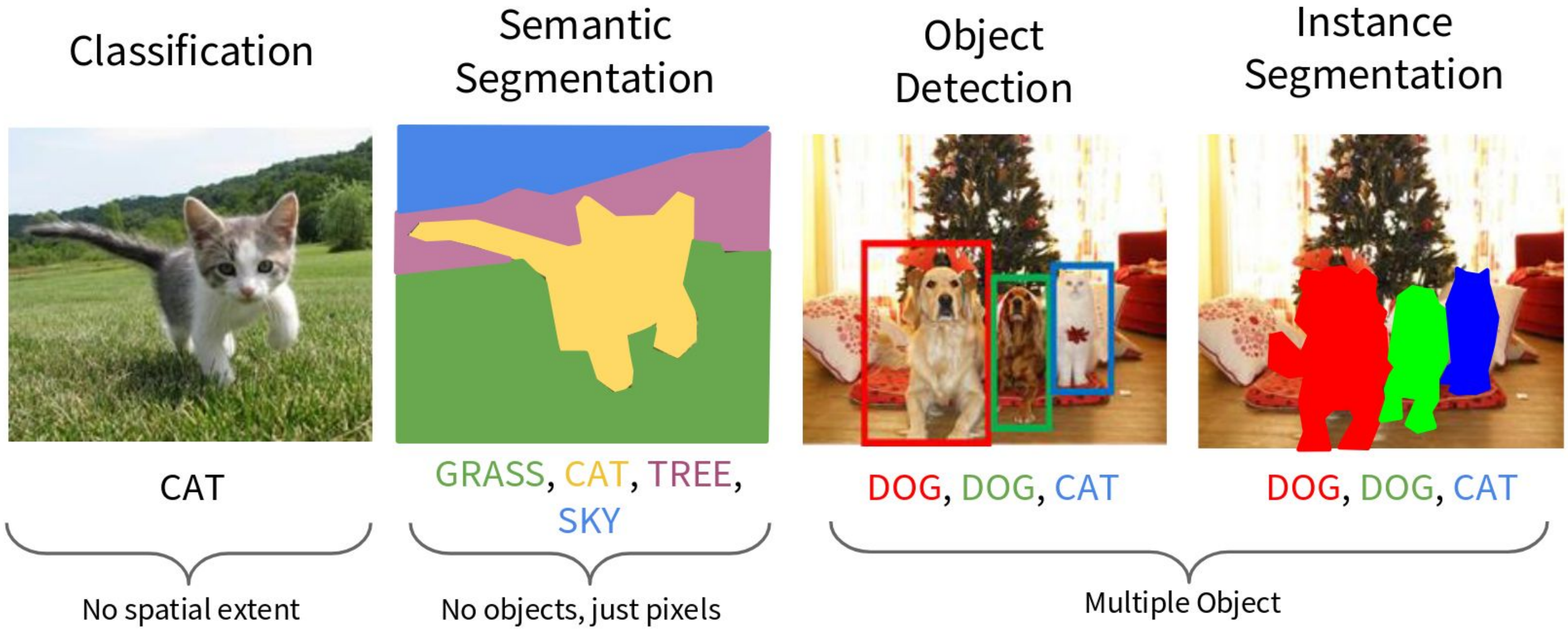


Why not just use MLPs for everything?

→ Does not exploit regularities in data, e.g., images.

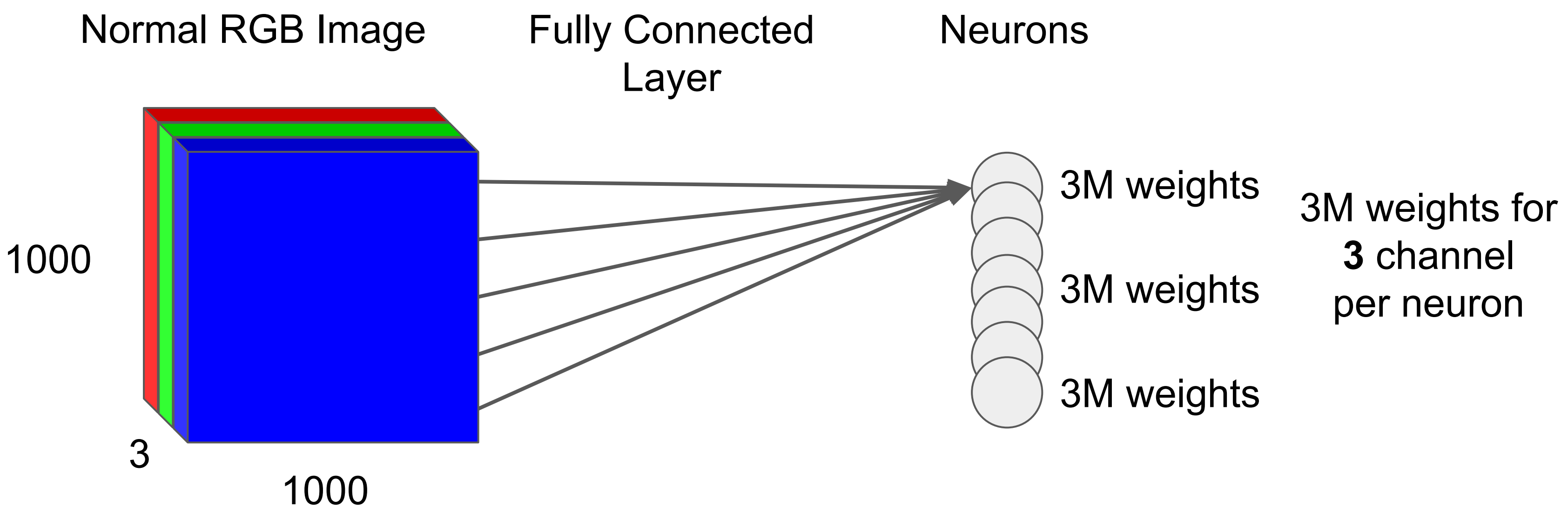
→ Inefficient in parameter count.

Computer Vision Tasks



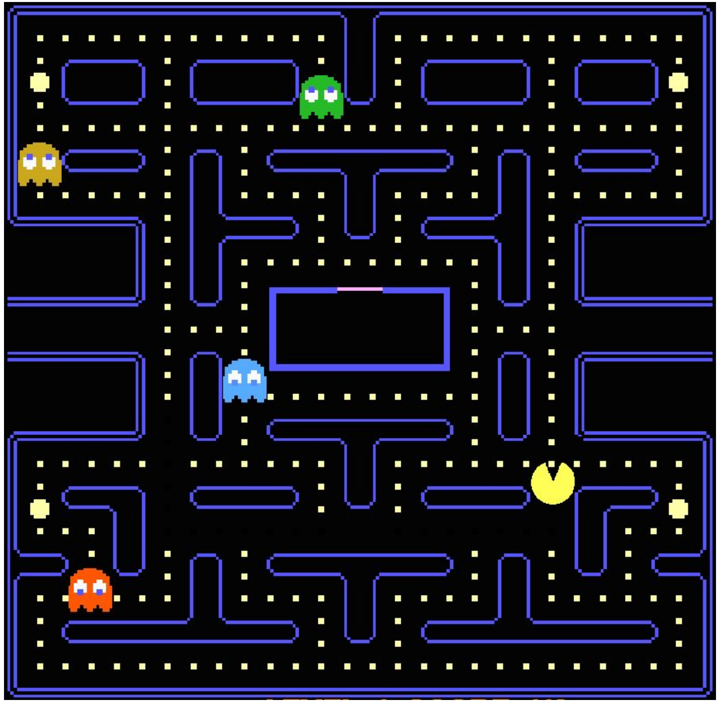
[Li et al., CS231n, Course Slide from Lecture 9]

Problems with Fully-Connected Layers



1000 neuron layer = **3B** weights

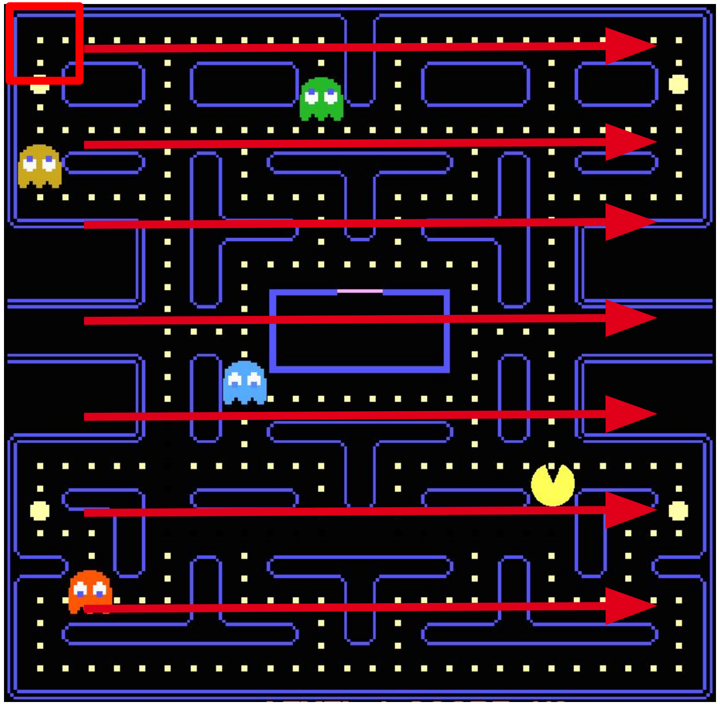
Pacman detector



Pacman detector

Filters

Convolve over image

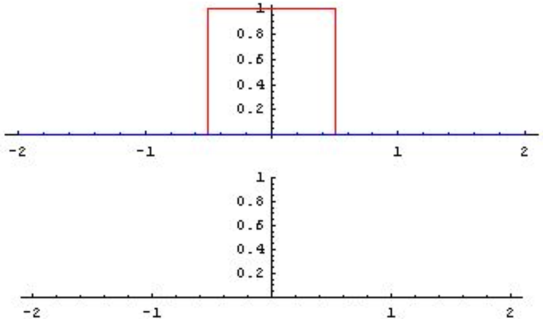


The Convolution Operator

- Usually defined as $k * f$ (k and f continuous functions over x)

$$(k * f)(x) = \int_{t=-+\infty}^{+\infty} k(t) f(x - t) dt$$

- E.g.: sliding filter (or *kernel*) *k* applied to signal *f*



(https://fr.wikipedia.org/wiki/Produit_de_convolution)

From continuous to discrete

- Continuous: sliding filter (or *kernel*) *k* applied to signal *f*

$$(k * f)(x) = \int_{t=-+\infty}^{+\infty} k(t) f(x-t) dt$$

- Discrete:

$$(k * f)(x) = \sum_{t=-a}^{a} k(t) f(x-t)$$

- 2D:

$$(k * f)(x, y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} k(dx, dy) f(x - dx, y - dy)$$

Simplify

$$(k * f)(x, y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} k'(dx, dy) f(x + dx, y + dy)$$

Discrete 1D Convolution

f	6	-2	2	3	1	7	7	-2	8	6

g	1/3	1/3	1/3

Discrete 1D Convolution

f	**6**	**-2**	**2**	**3**	**1**	**7**	**7**	**-2**	**8**	**6**
g	**1/3**	**1/3**	**1/3**							
*f * g*		**2**								

$6 * \frac{1}{3} - 2 * \frac{1}{3} + 2 * \frac{1}{3} = 2$

Discrete 1D Convolution

f	6	-2	2	3	1	7	7	-2	8	6
g		1/3	1/3	1/3						
$f * g$		2	1							

$$-2 * \tfrac{1}{3} + 2 * \tfrac{1}{3} + 3 * \tfrac{1}{3} = 1$$

Discrete 1D Convolution

f	6	-2	2	3	1	7	7	-2	8	6
g			1/3	1/3	1/3					
f * *g*		2	1	2						

$$2 * \tfrac{1}{3} + 3 * \tfrac{1}{3} + 1 * \tfrac{1}{3} = 2$$

Discrete 1D Convolution

f	6	-2	2	3	1	7	7	-2	8	6
g				1/3	1/3	1/3				
$f * g$		2	1	2	11/3					

$$3 * \tfrac{1}{3} + 1 * \tfrac{1}{3} + 7 * \tfrac{1}{3} = 11/3$$

Discrete 1D Convolution

f	6	-2	2	3	1	7	7	-2	8	6
g					1/3	1/3	1/3	→		
$f * g$		2	1	2	11/3	5				

$$1 * \tfrac{1}{3} + 7 * \tfrac{1}{3} + 7 * \tfrac{1}{3} = 5$$

Discrete 1D Convolution

f	6	-2	2	3	1	7	7	-2	8	6
g						1/3	1/3	1/3		
*f * g*		2	1	2	11/3	5	4			

$$7 * \tfrac{1}{3} + 7 * \tfrac{1}{3} - 2 * \tfrac{1}{3} = 4$$

Discrete 1D Convolution

f	6	-2	2	3	1	7	7	-2	8	6
g							1/3	1/3	1/3	
$f * g$		2	1	2	11/3	5	4	13/3		

$$7 * \tfrac{1}{3} - 2 * \tfrac{1}{3} + 8 * \tfrac{1}{3} = 13/3$$

Discrete 1D Convolution

f	6	-2	2	3	1	7	7	-2	8	6
g								1/3	1/3	1/3
f * *g*		2	1	2	11/3	5	4	13/3	4	

$-2 * \frac{1}{3} + 8 * \frac{1}{3} + 6 * \frac{1}{3} = 4$

Discrete 1D Convolution

6	-2	2	3	1	7	7	-2	8	6

1/3	1/3	1/3

???	2	1	2	11/3	5	4	13/3	4	???

What to do at the boundary?

Discrete 1D Convolution

6	-2	2	3	1	7	7	-2	8	6

1/3	1/3	1/3

2	1	2	11/3	5	4	13/3	4

What to do at the boundary?

Option 1: Shrink
"Valid" Convolution

Discrete 1D Convolution

0	6	-2	2	3	1	7	7	-2	8	6	0

1/3	1/3	1/3

4/3	2	1	2	11/3	5	4	13/3	4	14/3

What to do at the boundary?

Option 2: Pad Signal (e.g. with 0’s)
“Same” Convolution

Discrete 2D Convolution

Input 5x5

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

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

22		

$$5 * 5 - 1 * (-3) - 1 * 3 - 1 * 0 - 1 * 3 = 22$$

Discrete 2D Convolution

Input 5x5

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

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

22	-17	

5 * 0 - 1 * 4 - 1 * 5 - 1 * 0 - 1 * 8
= -17

Discrete 2D Convolution

Input 5x5

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

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

22	-17	-19

5 * 0 - 1 * 8 - 1 * 0 - 1 * 5 - 1 * 6
= 19

Discrete 2D Convolution

Input 5x5

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

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

22	-17	-19
-5		

5 * 3 - 1 * 5 - 1 * 1 - 1 * 8 - 1 * 6
= -5

Discrete 2D Convolution

Input 5x5

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

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

22	-17	-19
-5	26	

5 * 8 - 1 * 0 - 1 * 3 - 1 * 6 - 1 * 5
= 26

Discrete 2D Convolution

Input 5x5

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

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

22	-17	-19
-5	26	18

5 * 6 - 1 * 0 - 1 * 8 - 1 * 1 - 1 * 3
= 18

Discrete 2D Convolution

Input 5x5

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

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

22	-17	-19
-5	26	18
17		

5 * 6 - 1 * 3 - 1 * 5 - 1 * 5 - 1 * 0
= 17

Discrete 2D Convolution

Input 5x5

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

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

22	-17	-19
-5	26	18
17	3	

5 * 5 - 1 * 8 - 1 * 6 - 1 * 3 - 1 * 5
= 3

Discrete 2D Convolution

Input 5x5

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

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

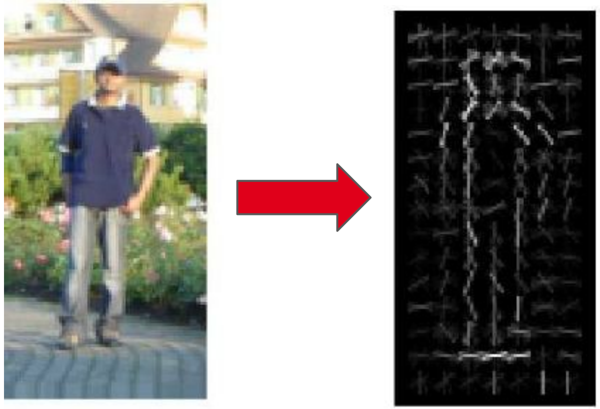
22	-17	-19
-5	26	18
17	3	10

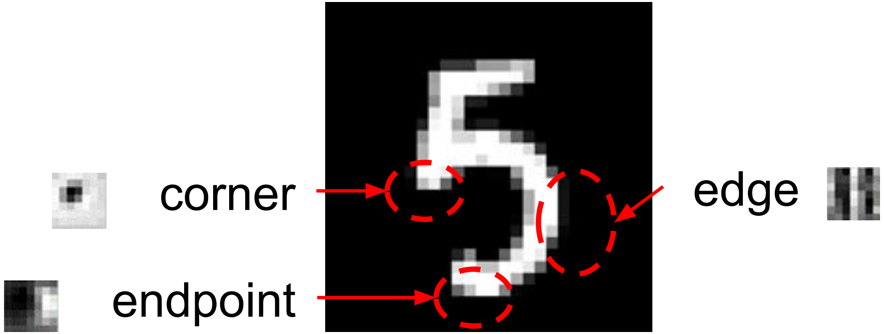
$$5 * 3 - 1 * 6 - 1 * 5 - 1 * (-2) - 1 * (-4)$$
$$= 10$$

Feature Learning

- Images characterized by *features* such as *edges*, *corners*, etc.

Previously: hand-crafted





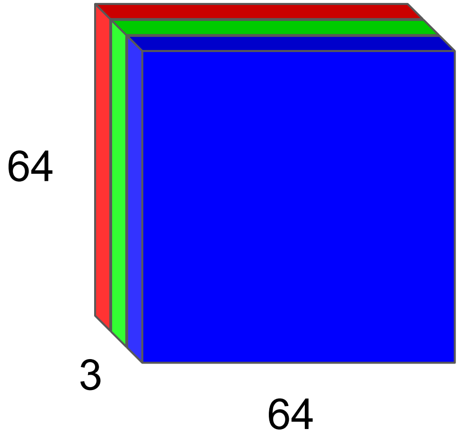
- E.g.: histograms of oriented gradients

- Requires ad-hoc feature detector design

- Let the network learn local feature detector(s)

Convolutions on RGB Images

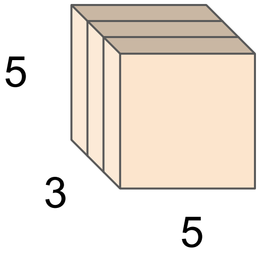
RGB Image



3 x 64 x 64

depth x width x height

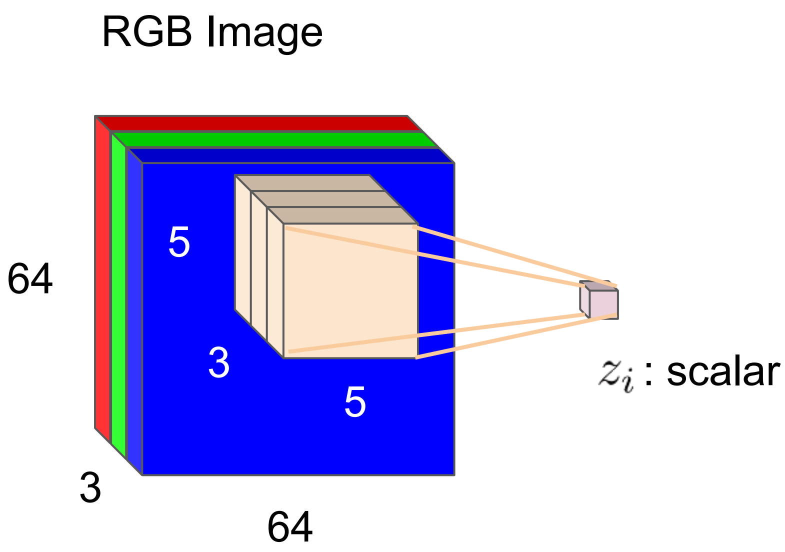
Filter/Kernel



3 x 5 x 5

Apply convolution:
- slide filter over all image locations
- apply dot product

Convolutions on RGB Images



pixels **x**: 3 x 64 x 64

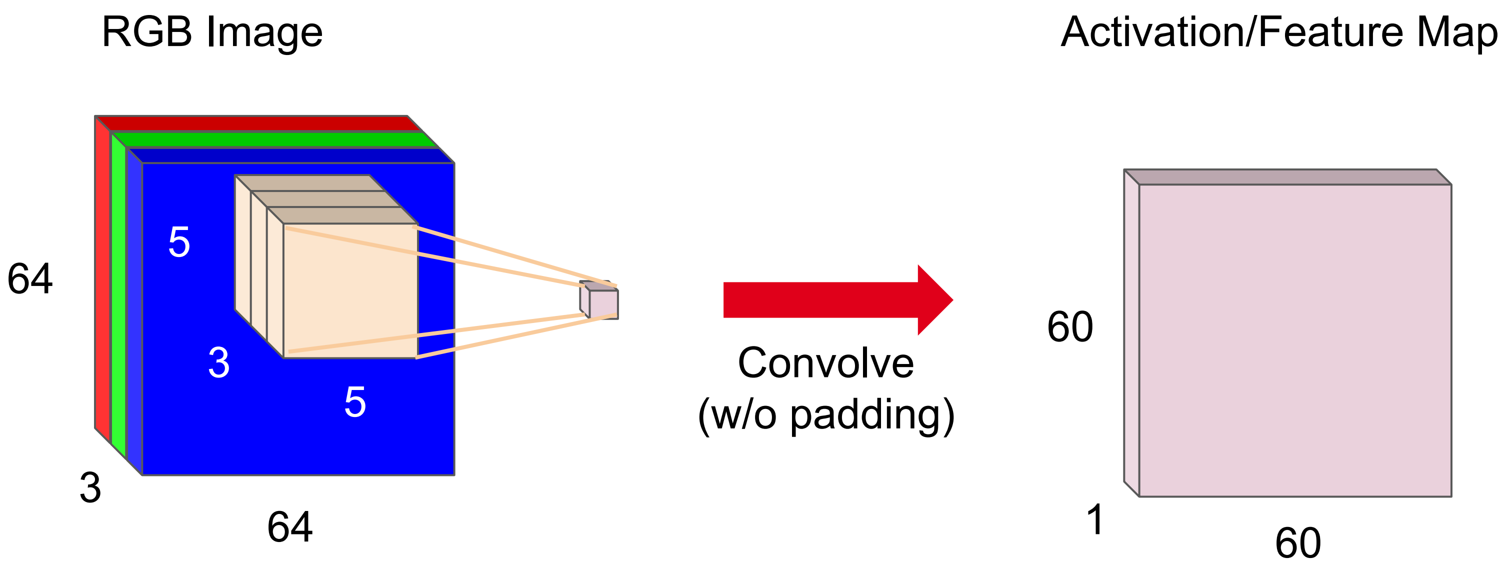
weights **w**: 3 x 5 x 5

1 number per image location:
- dot product between filter weights w and x_i-th chunk of image.

$$z_i = \mathbf{w}^\top \mathbf{x}_i + b$$

(3 x 5 x 5) x 1 (3 x 5 x 5) x 1 1

Convolutions on RGB Images

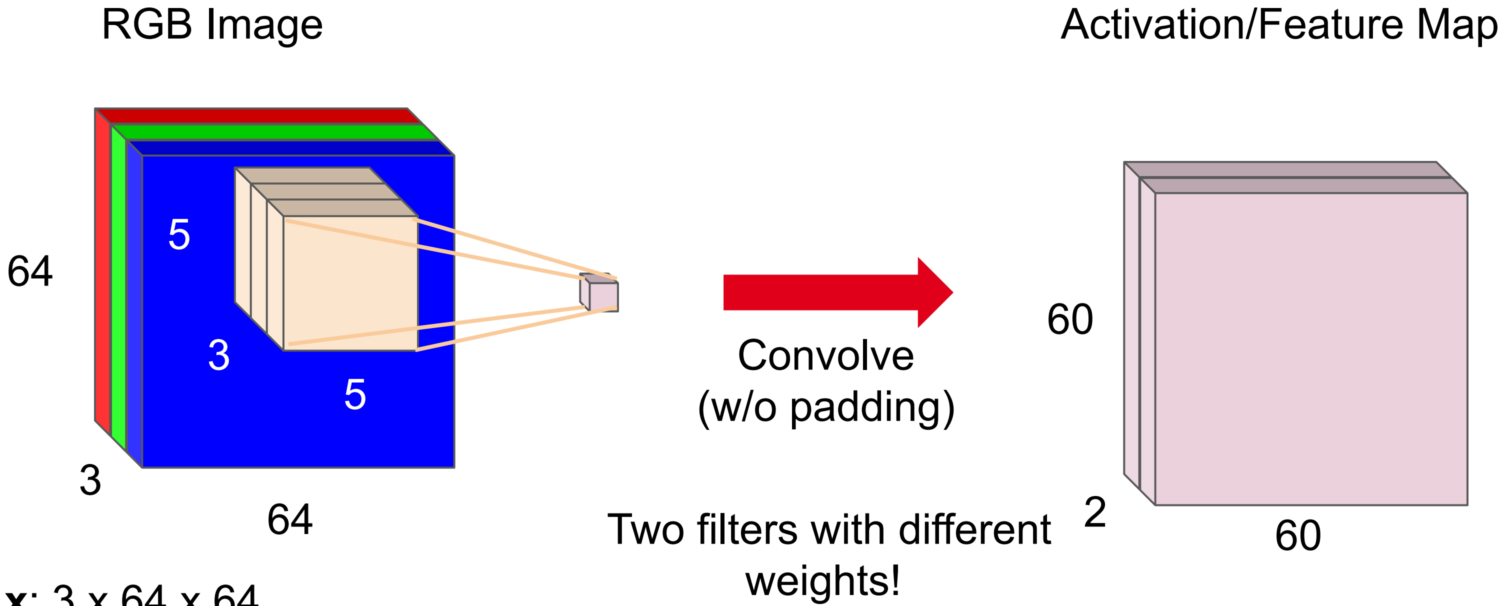


pixels **x**: 3 x 64 x 64

weights **w**: 3 x 5 x 5

output **z**: 1 x 60 x 60

Convolution Layer

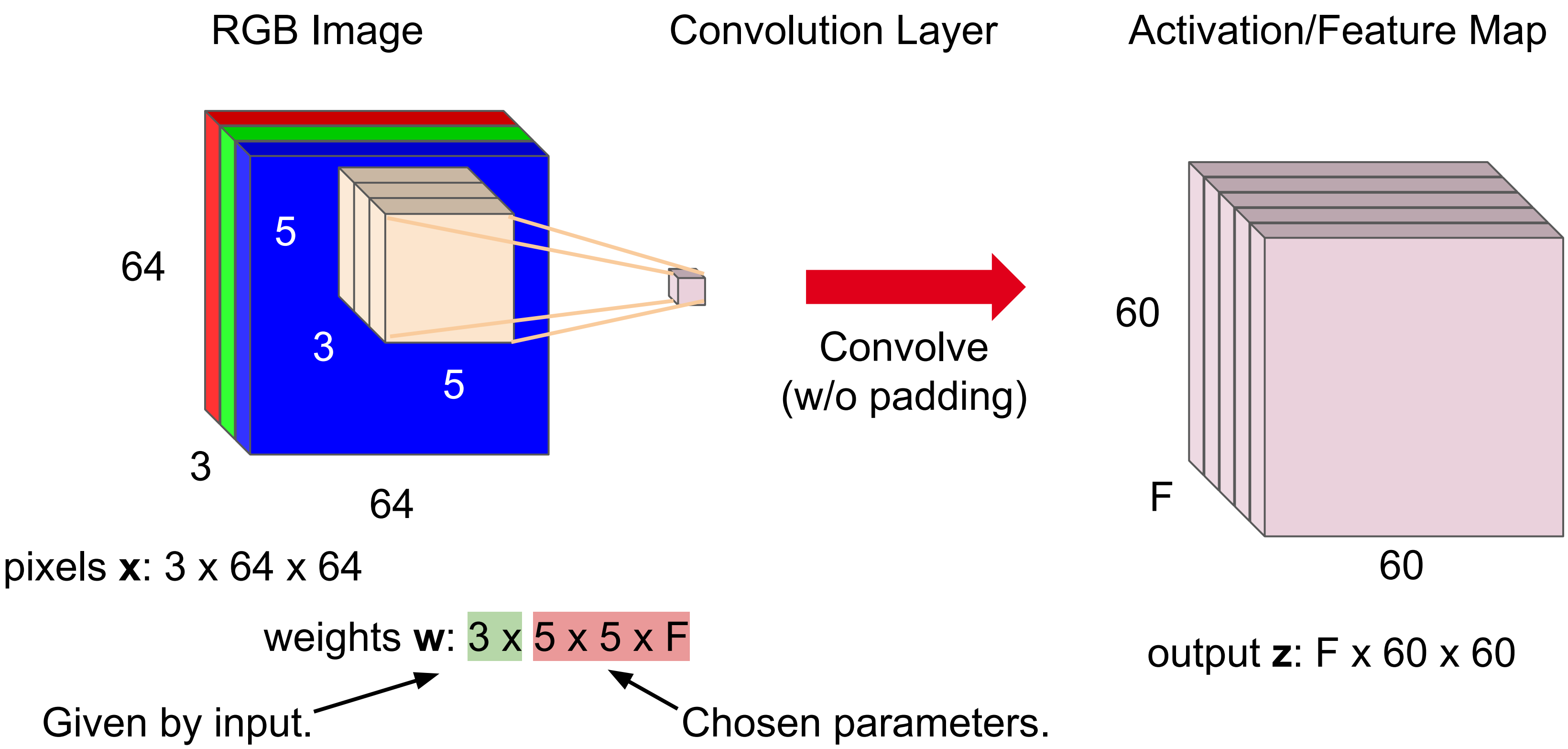


pixels **x**: 3 x 64 x 64

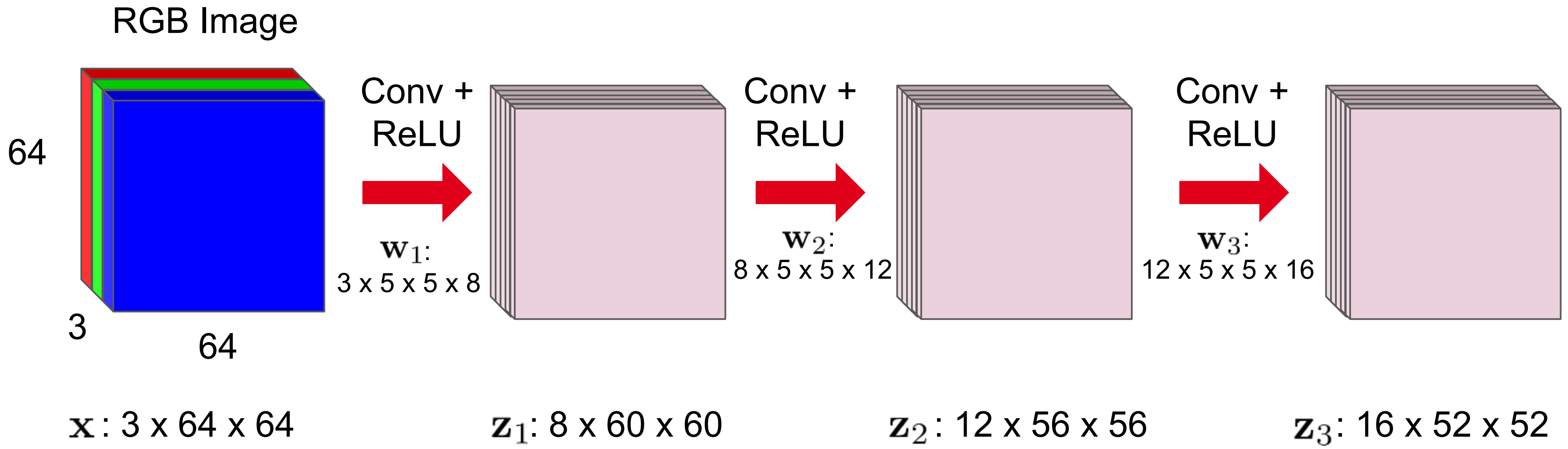
weights **w**: 3 x 5 x 5 x 2

output **z**: 2 x 60 x 60

Convolution Layer



Convolutional Neural Network (CNN)

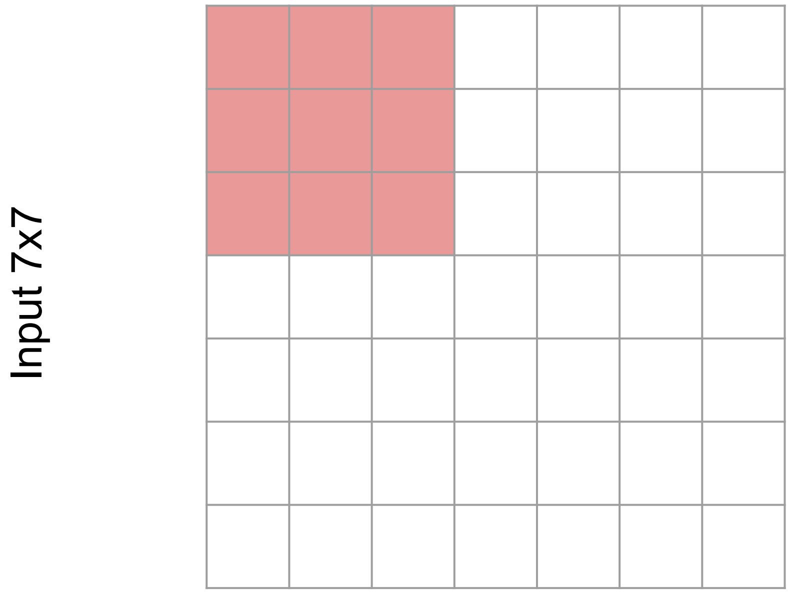


Learned Filters of a Convolutional Network

Low-Level Features

[Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ECCV’14]

Convolution Layer: Hyperparameters



Valid Convolution (no padding)

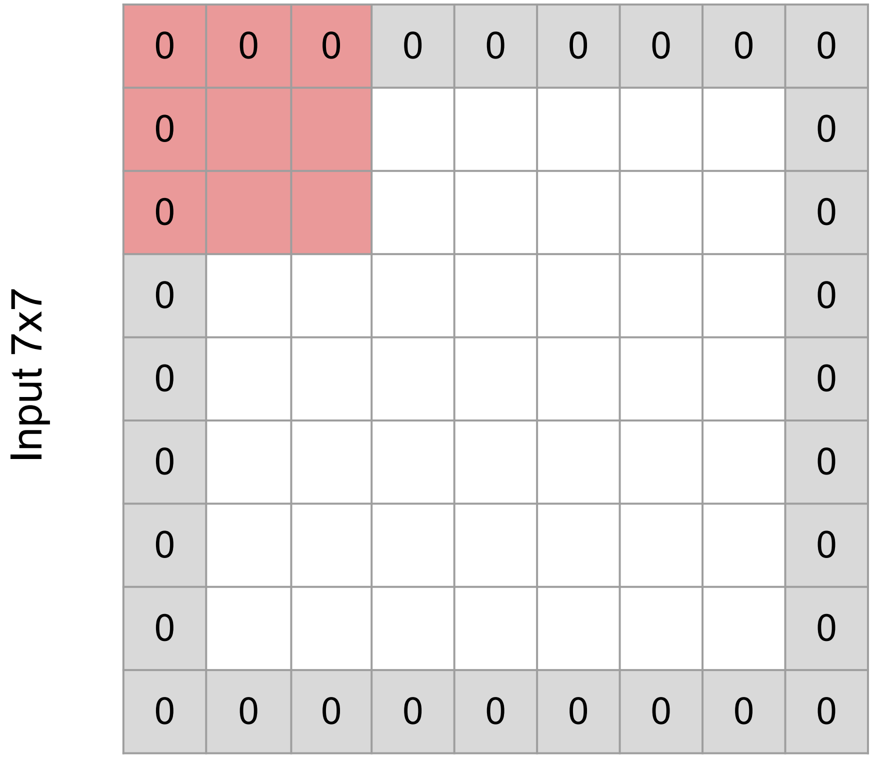
Input (N x N): 7 x 7
Filter (K x K): 3 x 3
Padding (P): 0

Output: 5 x 5

Output size:

$$(N + 2P - K + 1) \times (N + 2P - K + 1)$$

Convolution Layer: Hyperparameters



Same Convolution (input size = output size)

Input (N x N): 7 x 7
Filter (K x K): 3 x 3
Padding (P): 1

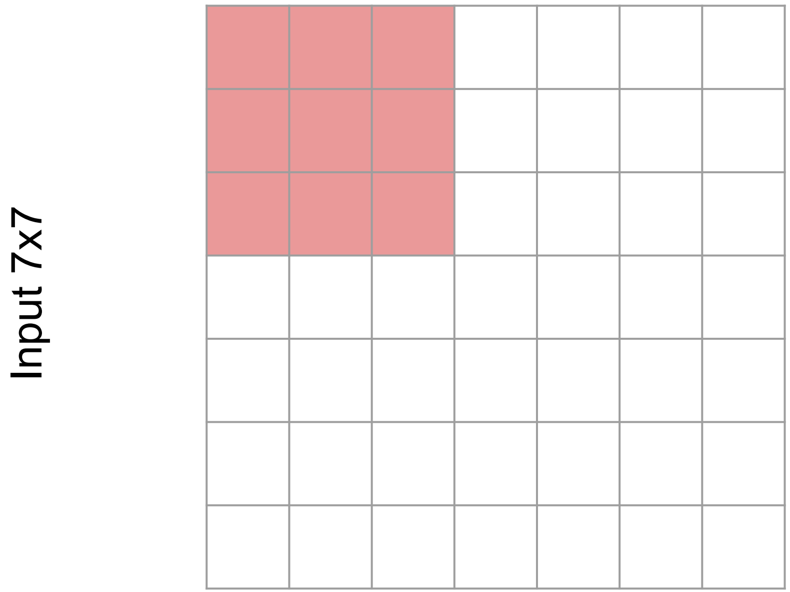
$$P = \frac{K-1}{2}$$

Output: 7 x 7

Output size:

$$(N + 2P - K + 1) \times (N + 2P - K + 1)$$

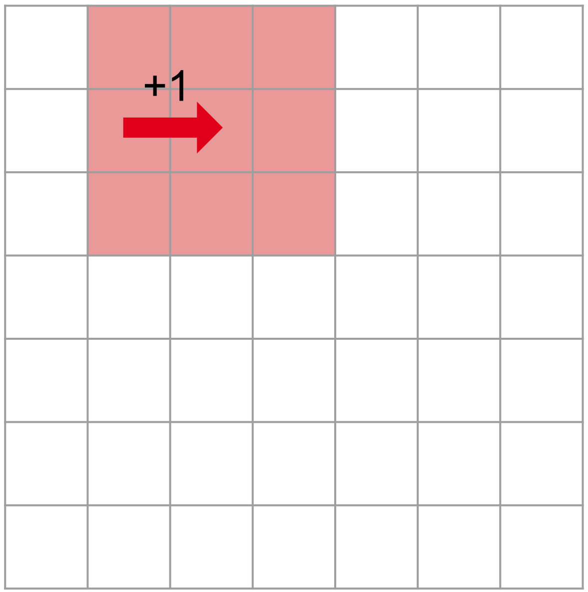
Convolution Layer: Hyperparameters



Input (N x N):	7 x 7
Filter (K x K):	3 x 3
Padding (P):	0
Stride (S):	**1**
Output:	7 x 7

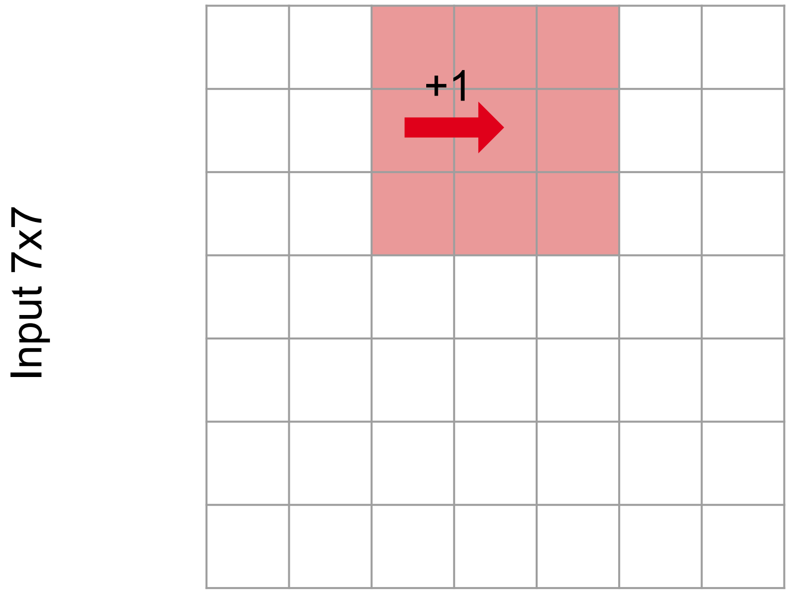
Convolution Layer: Hyperparameters

Input 7x7



Input (N x N): 7 x 7
Filter (K x K): 3 x 3
Padding (P): 0
Stride (S): 1
Output: 7 x 7

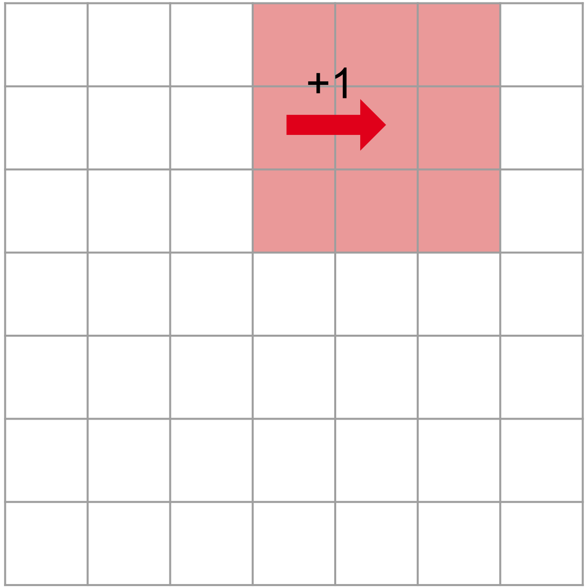
Convolution Layer: Hyperparameters



Input (N x N):	7 x 7
Filter (K x K):	3 x 3
Padding (P):	0
Stride (S):	**1**
Output:	7 x 7

Convolution Layer: Hyperparameters

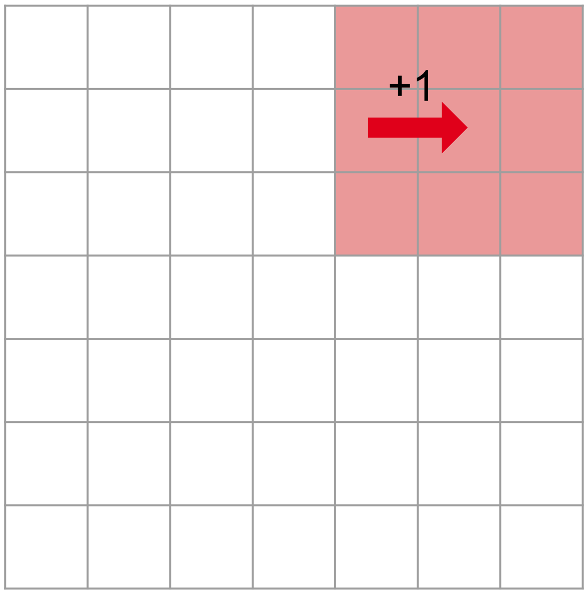
Input 7x7



Input (N x N):	7 x 7
Filter (K x K):	3 x 3
Padding (P):	0
Stride (S):	**1**
Output:	7 x 7

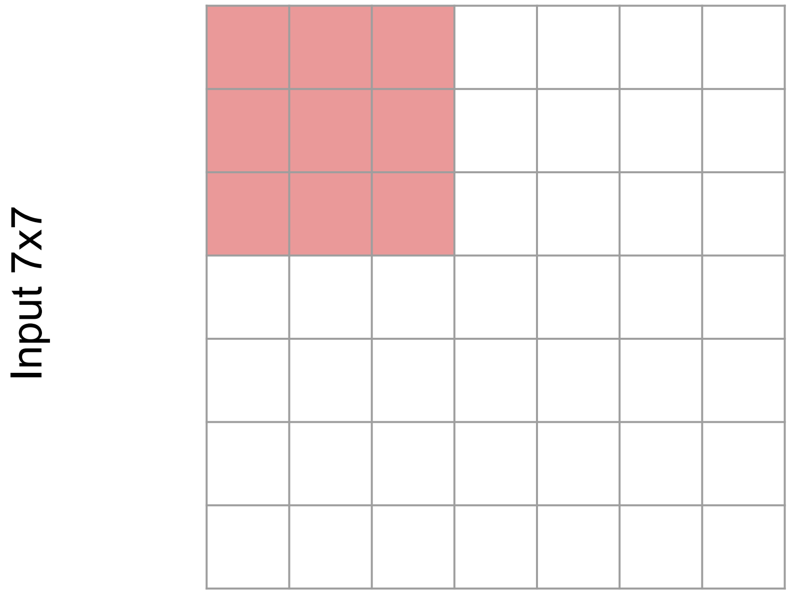
Convolution Layer: Hyperparameters

Input 7x7



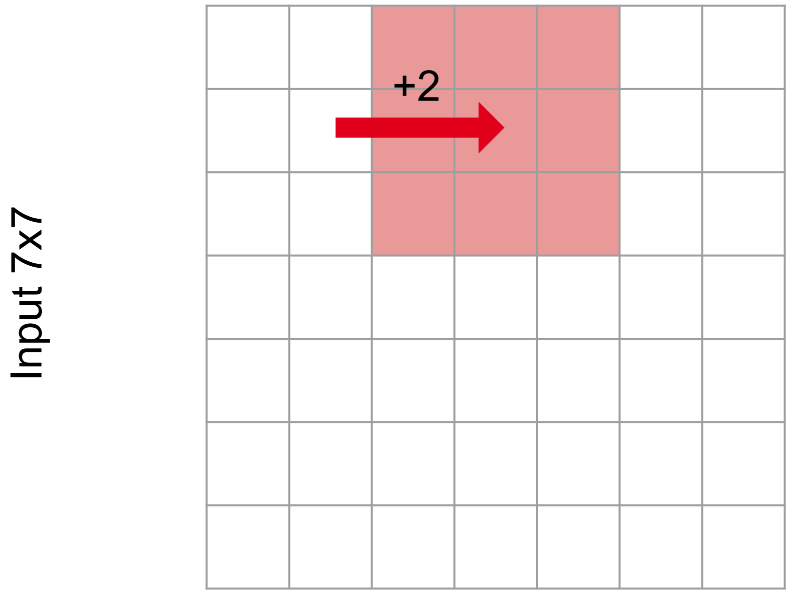
Input (N x N): 7 x 7
Filter (K x K): 3 x 3
Padding (P): 0
Stride (S): 1
Output: 7 x 7

Convolution Layer: Hyperparameters



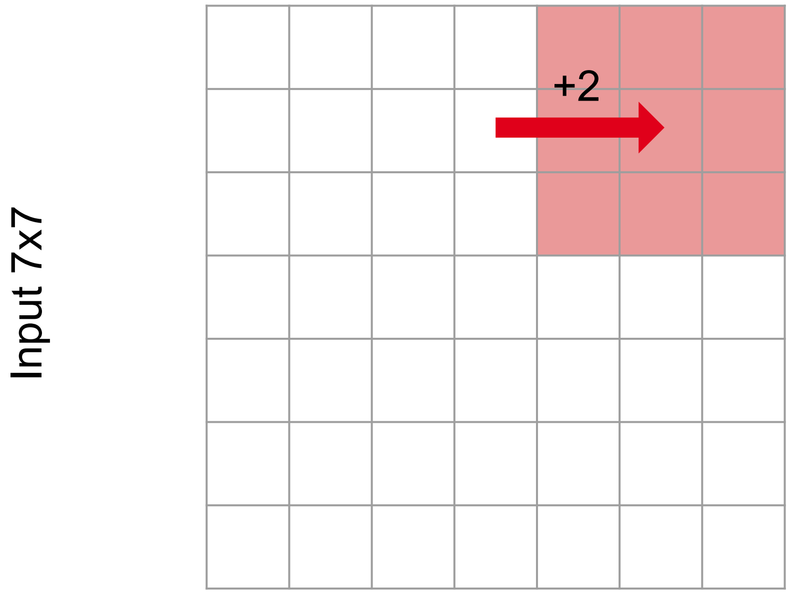
Input (N x N): 7 x 7
Filter (K x K): 3 x 3
Padding (P): 0
Stride (S): 2

Convolution Layer: Hyperparameters



Input (N x N): 7 x 7
Filter (K x K): 3 x 3
Padding (P): 0
Stride (S): 2

Convolution Layer: Hyperparameters



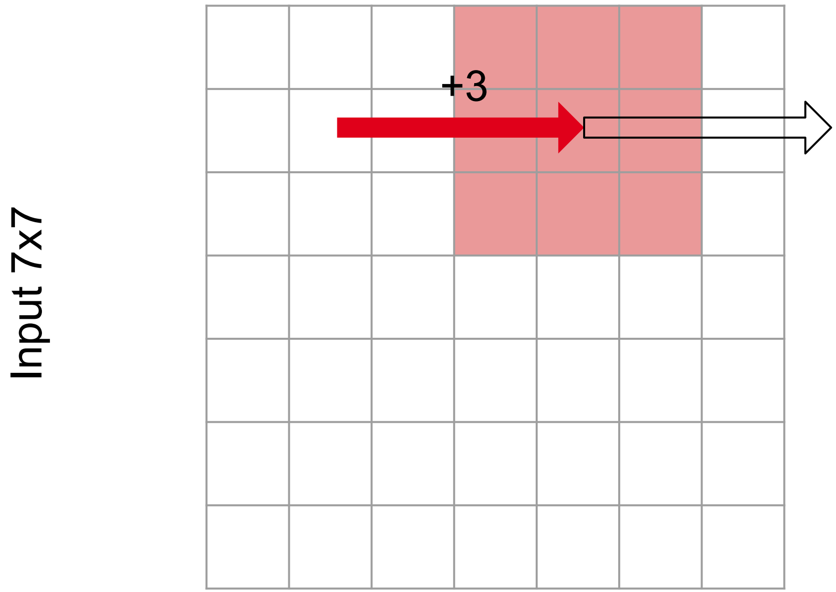
Input (N x N): 7 x 7
Filter (K x K): 3 x 3
Padding (P): 0
Stride (S): 2
Output: 3 x 3

Output size:

$$\left(\left\lfloor\frac{N+2P-K}{S}\right\rfloor+1\right)\times\left(\left\lfloor\frac{N+2P-K}{S}\right\rfloor+1\right)$$

$\lfloor\,\rfloor$ denotes floor operation

Convolution Layer: Hyperparameters



Input (N x N): 7 x 7
Filter (K x K): 3 x 3
Padding (P): 0
Stride (S): 3
Output: 2 x 2

Output size:

$$\left(\left\lfloor\frac{N+2P-K}{S}\right\rfloor+1\right)\times\left(\left\lfloor\frac{N+2P-K}{S}\right\rfloor+1\right)$$

$\lfloor\ \rfloor$ denotes floor operation

Quiz: Output size and #Parameters

Suppose you have an **RGB** input image of size **128 x 128**.
In your layer, you apply **16** convolutional filters of size **7 x 7**
with **stride 2** and **padding 3**.

Q1: What is the output size after applying the convolutional layer?

A1: $\left(\left\lfloor \frac{128 + 2 * 3 - 7}{2} \right\rfloor + 1\right) = 64$ $\rightarrow$ 16x64x64 (FxHxW)

Hint: $\left(\left\lfloor \frac{N + 2P - K}{S} \right\rfloor + 1\right)$

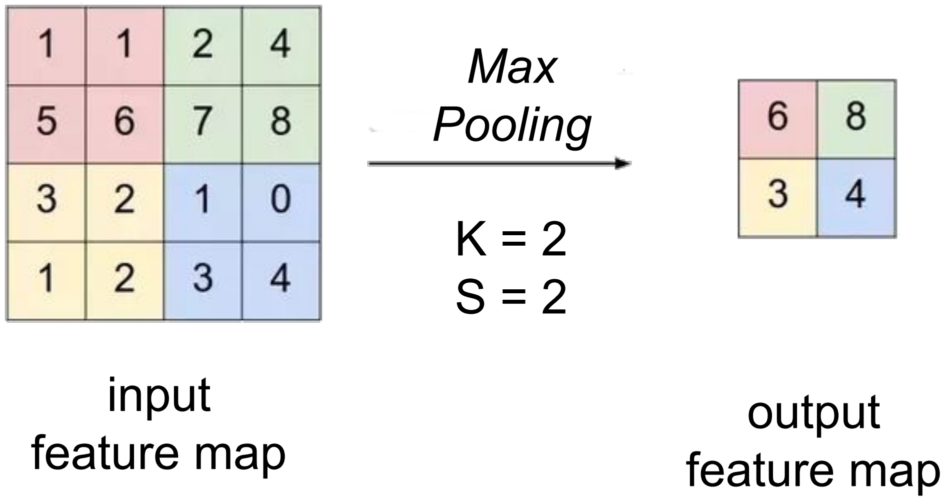
Q2: How many parameters does the layer have?

A2: (3 x 7 x 7 + 1) x 16 = 2368
Image size, stride, and padding do not affect parameter count.

Hint: Don’t forget the bias.

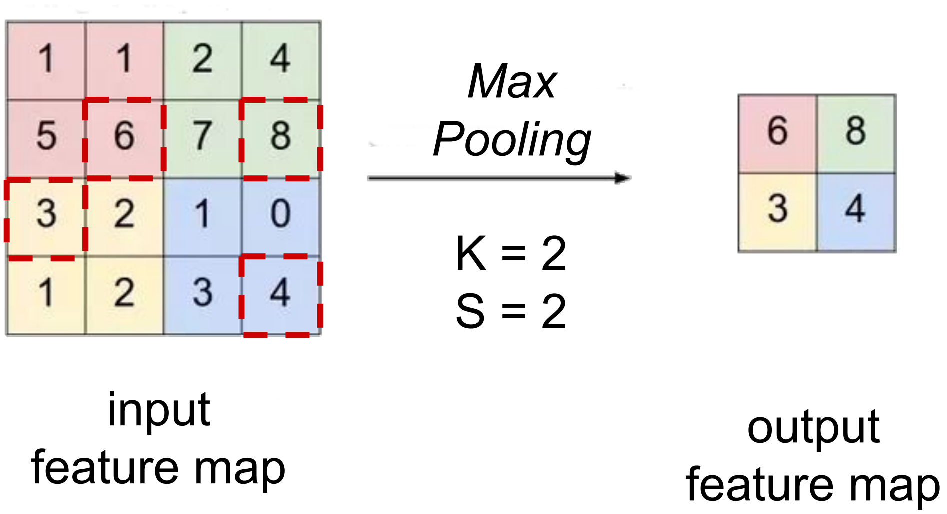
Pooling: Max Pooling Layer

- Pick maximum value for each, e.g, 2x2 non-overlapping area
  - Feature map spatial subsampling



Pooling: Max Pooling Layer

- Pick maximum value for each, e.g, 2x2 non-overlapping area
  - Feature map spatial subsampling



Convolution:
“Feature Extraction”

Pooling:
“Feature Selection”

Pooling: Average Pooling Layer

- Pick **average** value for each, e.g, 2x2 non-overlapping area
  - Feature map spatial subsampling

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

input
feature map

Average Pooling →

K = 2
S = 2

3.25	5.25
2	2

output
feature map

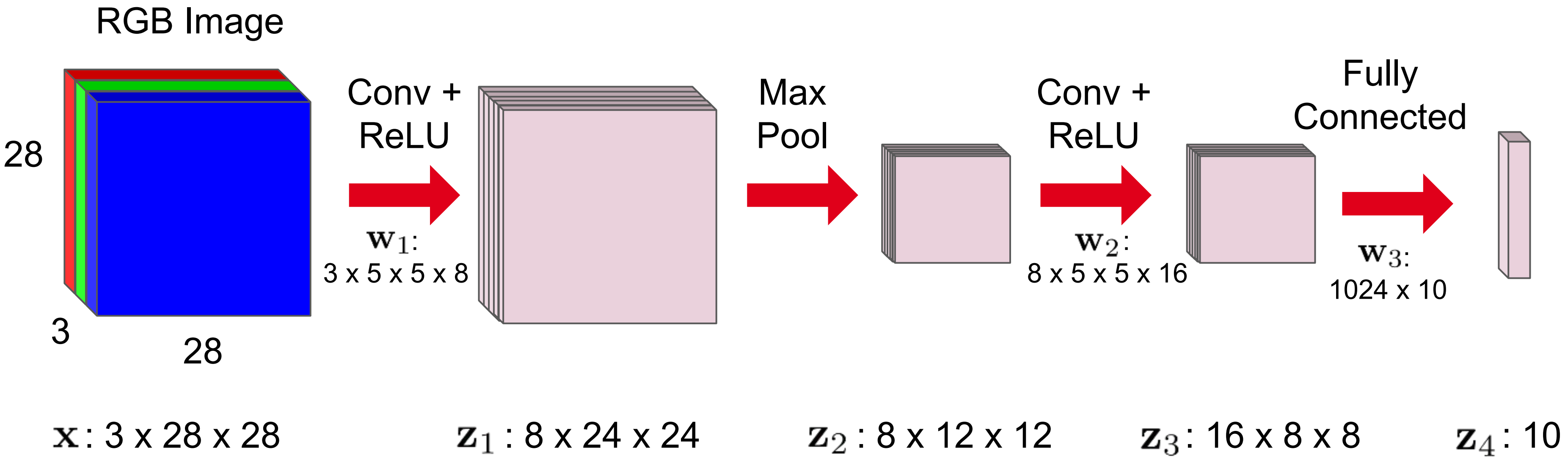
Output size:

$$\frac{N-K}{S}+1$$

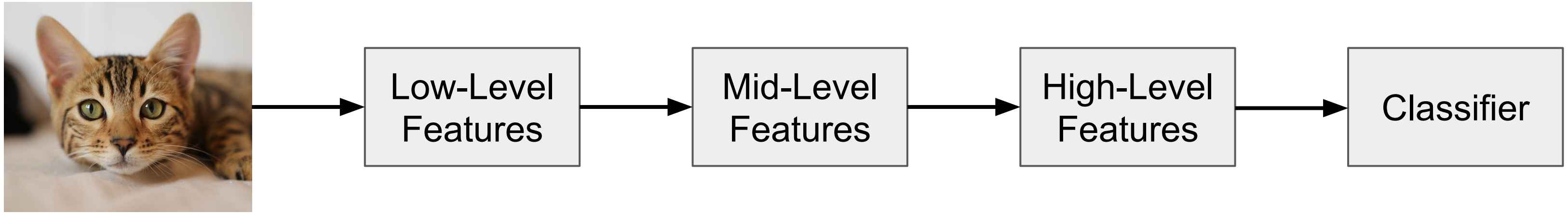
Channel size unchanged (applied to each channel independently)

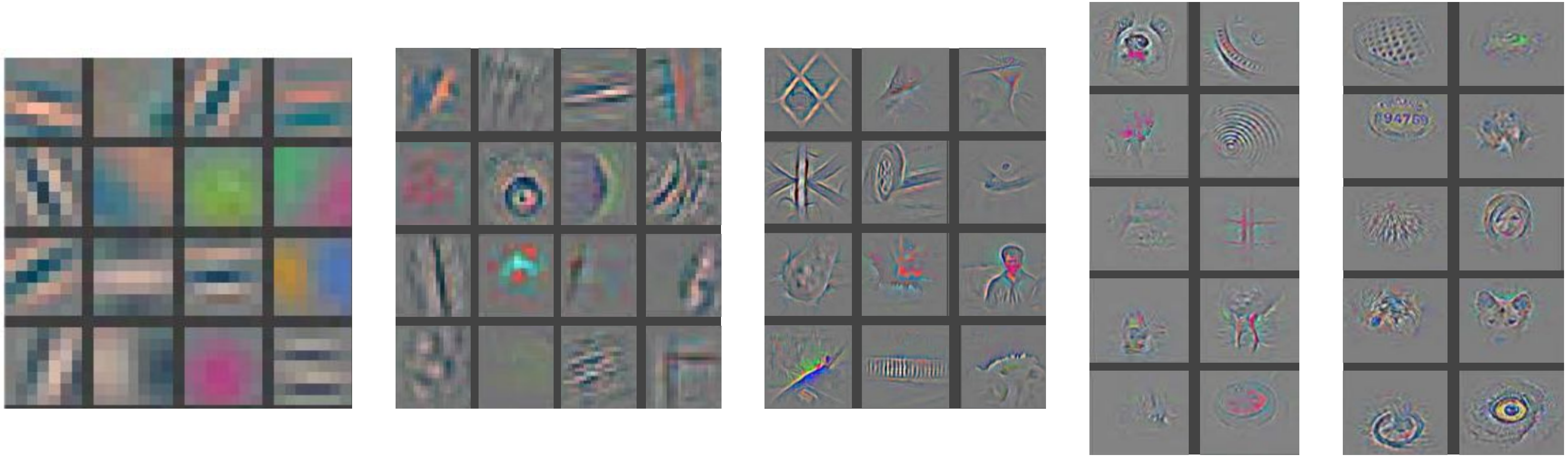
Pooling has no parameters

Convolutional Neural Network (CNN)



Learned Filters of a Convolutional Network

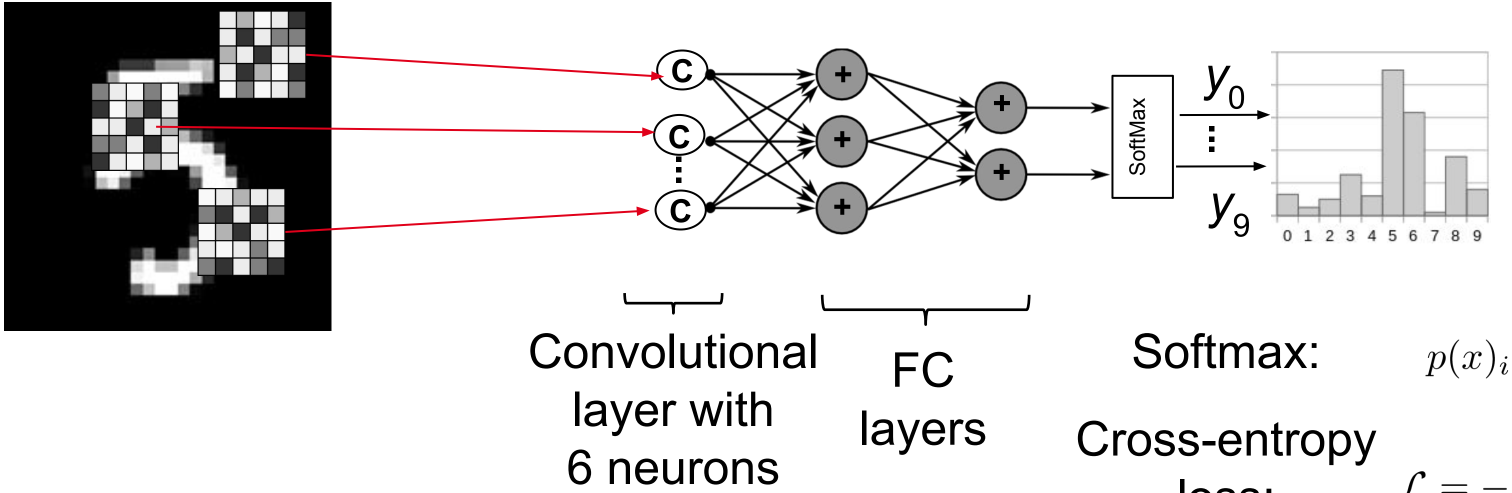




[Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ECCV'14]

Example: Convolutional *LeNet300*

- Task: MNIST digit recognition, image classification
- Data: 28x28 gray scale images, 10 digits
- Network: CNN followed by fully connected layers
- Output: class probability distribution (C=10)



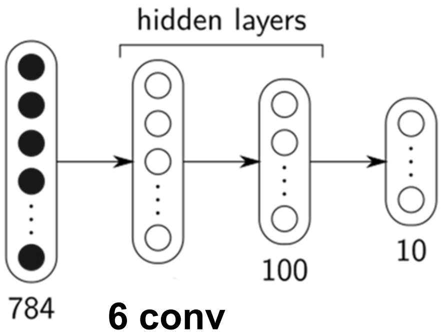
Softmax: $$p(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Cross-entropy loss: $$\mathcal{L} = -\sum_{c=1}^{C} y_c \log p(x)_c$$

Convolutional *LeNet300* - Complexity

- 1 st layer complexity drops 230k -> 156 params!

	Fully Connected		Convolutional	
Layer	Type	Complexity [prms]	Type	Complexity [prms]
1	FC-300	300 * (28*28) = 230k	Conv-6	
2	FC-100	100 * 300 = 30k	FC-100	
3	FC-10	10 * 100 = 1k	FC-10	

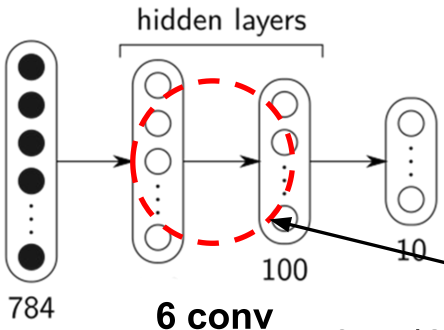


6 conv neurons (6 filters)

Convolutional *LeNet300* - Complexity

- Total complexity soars 260k -> 400k params!

	Fully Connected		Convolutional	
Layer	Type	Complexity [prms]	Type	Complexity [prms]
1	FC-300	300 * (28*28) = 230k	Conv-6	6 * (5x5 +1) * 1 = 156
2	FC-100	100 * 300 = 30k	FC-100	100 * (6 * (28x28)) = 400k
3	FC-10	10 * 100 = 1k	FC-10	10 * 100 = 1k
	~260k		~400k	



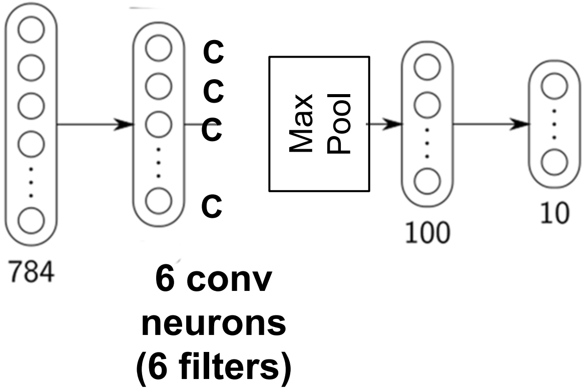
6 conv neurons (6 filters)

6 x (28x28) = 4700 features

Convolutional *LeNet300* - Complexity

- Complexity from ~260k to ~118k params thanks to *Maxpooling*

	Fully Connected		Convolutional	
Layer	Type	Complexity [prms]	Type	Complexity [prms]
1	FC-300	300 * (28*28) = 230k	Conv-6	6 * (5x5 +1) * 1 = 156
2	FC-100	100 * 300 = 30k	FC-100	100 * (6 * (14x14)) = ~~400k~~ 117k
3	FC-10	10 * 100 = 1k	FC-10	10 * 100 = 1k
Tot	~260k		~118k	



Convolutional *LeNet300* – Performance

- Experiments on MNIST 28x28 dataset

Network	Num. Layers	Error [%]
Fully connected LeNet300	1 FC output layer (10 U)	12.0
	1 hidden FC (300 U), 1 output FC (10 U)	4.7
	2 hidden FCs (300 + 100 U), 1 output FC (10 U)	3.05
Convol. LeNet300	1 Conv (3 F), 1 output FC (*LeNet1*)	1.7
	2 conv (6+16 F), 3 FC layer	0.95

Better performance for lower complexity

Convolutional *LeNet300* – Reflexions

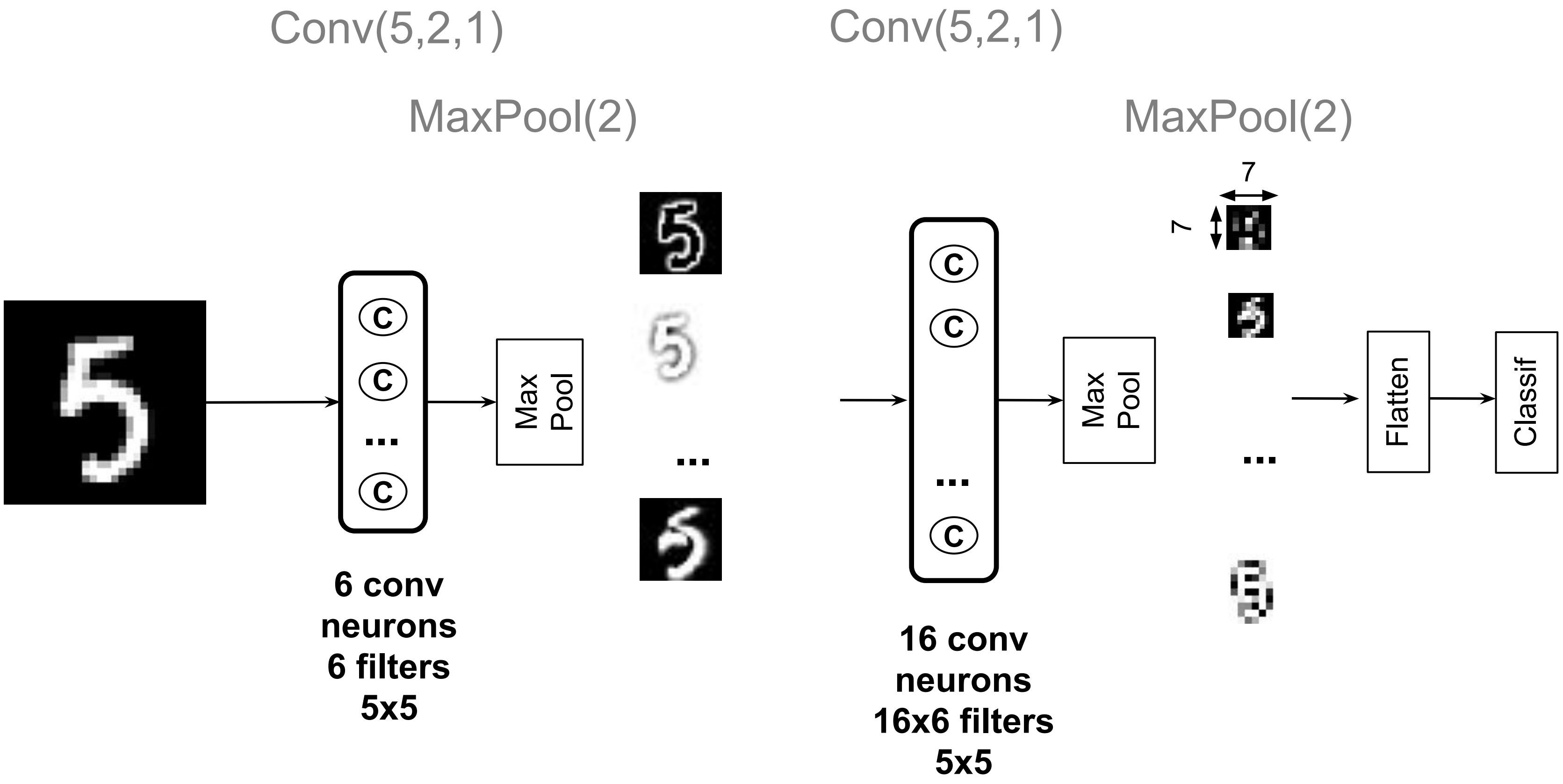
The convolutional LeNet300 performs better than its fully connected counterpart despite:

- it has fewer parameters due to the convolutional layers
- the filters are not big enough (5x5) to capture an entire digit (at least 20x20 pixels in a 28x28 image)

Let us define at the ***receptive field***

- The *receptive field of a feature* is its *back-projection* through the pooling and convolutional layers within the input image

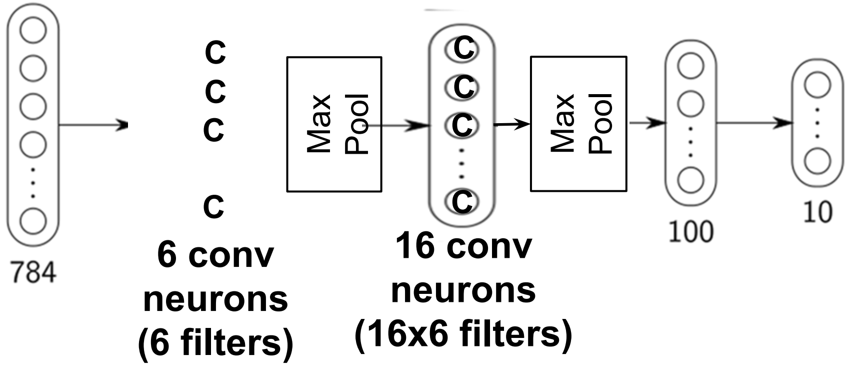
Convolutional *LeNet300* – Receptive Field



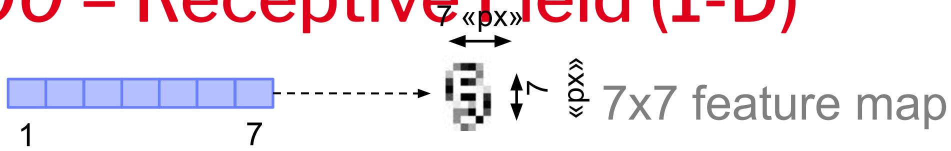
Convolutional *LeNet300* - Complexity

- Total complexity drops from ~260k to ~82k params

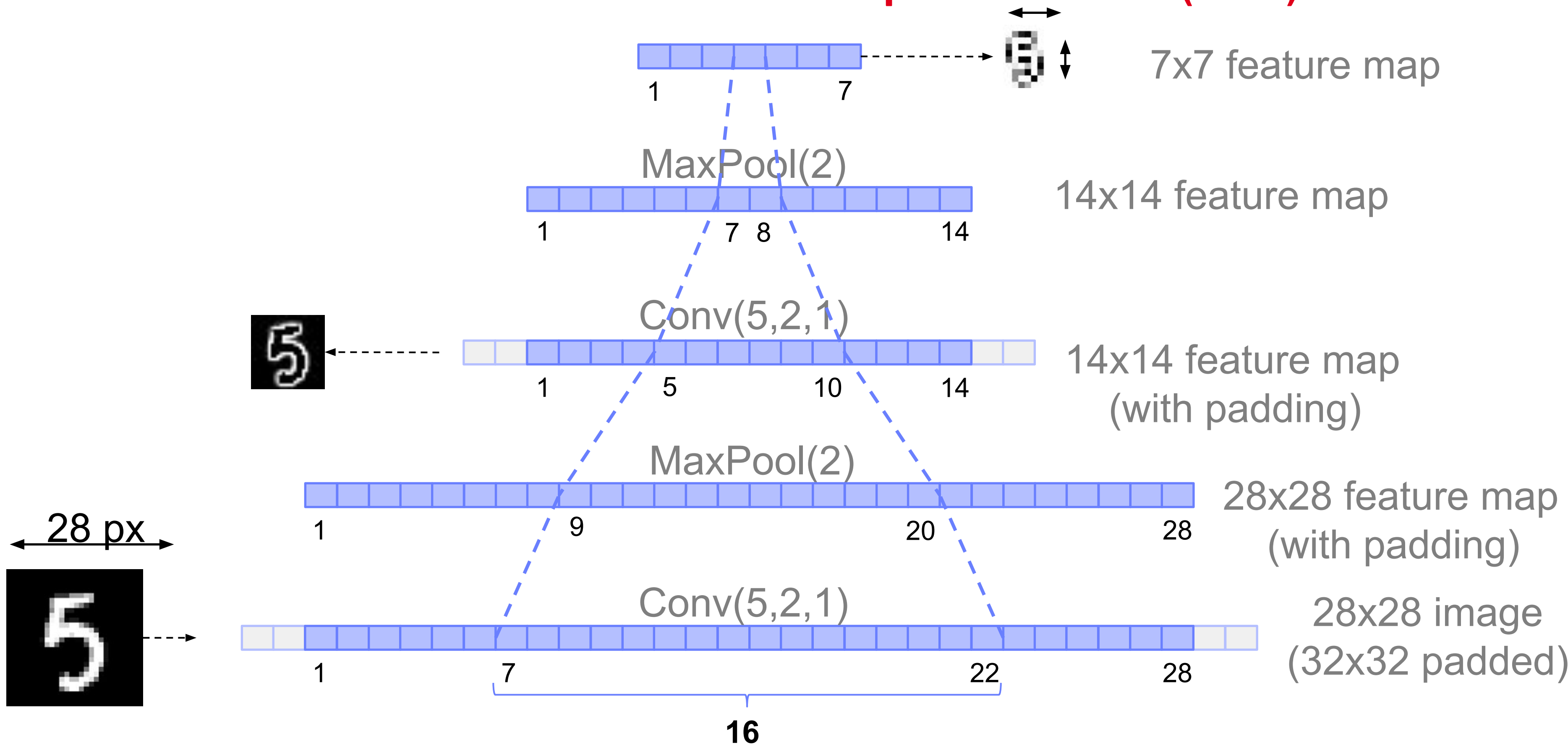
	Fully Connected		Convolutional	
Layer	Type	Complexity [prms]	Type	Complexity [prms]
1	FC-300	300 * (28*28) = 230k	Conv-6	6 * (5x5 +1) * 1 = 156
2	FC-100	100 * 300 = 30k	Conv-16	16 * (5x5 +1) * 6 = 2496
3			FC-100	100 * ((16 * 7x7) +1) = 78k
4	FC-10	10 * 100 = 1k	FC-10	10 * 100 = 1k
Tot	~260k		~82k	



Convolutional *LeNet300* – Receptive Field (1-D)

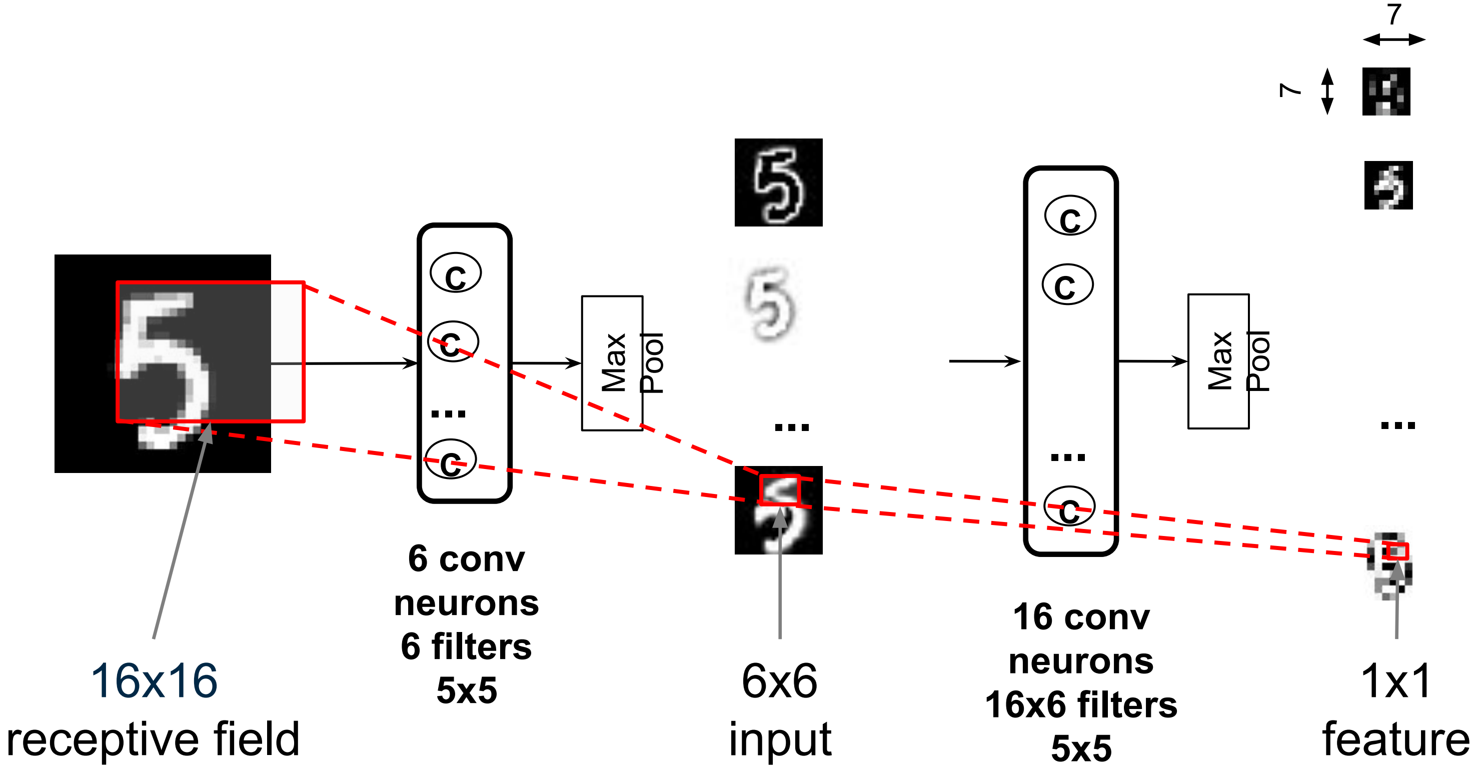


Convolutional *LeNet300* – Receptive Field (1-D)



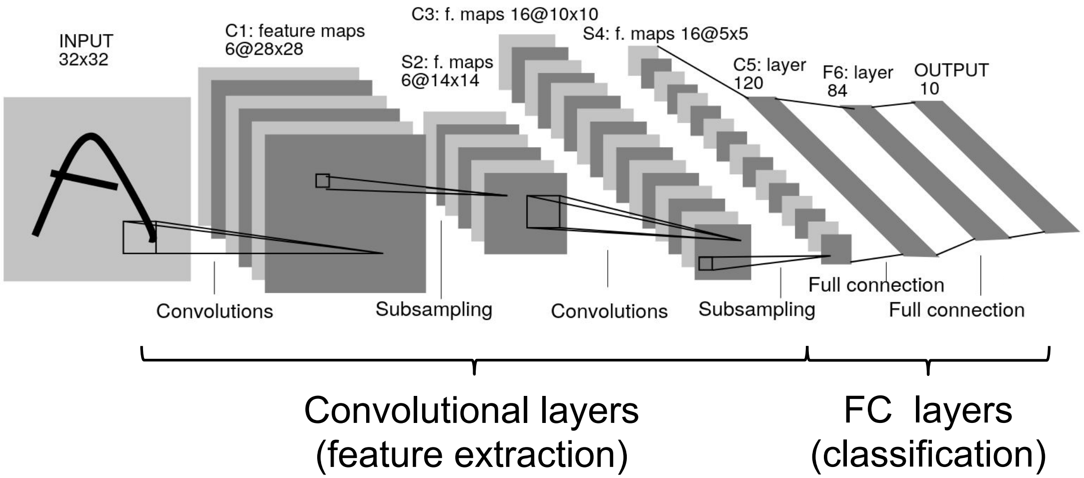
Convolutional *LeNet300* – Receptive Field

- This holds for *central* features in the last feature map



Convolutional Networks – LeNet5

- Stacked sigmoid convolutional layers for feature extraction
- Repeated *convolve-and-pool* pattern
- Multiple FC layer for classification



Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, November 1998 (PDF available online)

Gradient-Based Learning Applied to Document Recognition

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—

Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

Real-life document recognition systems are composed of multiple modules including field extraction, segmentation, recognition, and language modeling. A new learning paradigm, called Graph Transformer Networks (GTN), allows such multi-module systems to be trained globally using Gradient-Based methods so as to minimize an overall performance measure.

Two systems for on-line handwriting recognition are described. Experiments demonstrate the advantage of global training, and the flexibility of Graph Transformer Networks.

A Graph Transformer Network for reading bank check is also described. It uses Convolutional Neural Network character recognizers combined with global training techniques to provides record accuracy on business and personal checks. It is deployed commercially and reads several million checks per day.

I. Introduction

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

The main message of this paper is that better pattern recognition systems can be built by relying more on automatic learning, and less on hand-designed heuristics. This is made possible by recent progress in machine learning and computer technology. Using character recognition as a case study, we show that hand-crafted feature extraction can be advantageously replaced by carefully designed learning machines that operate directly on pixel images. Using document understanding as a case study, we show that the traditional way of building recognition systems by manually integrating individually designed modules can be replaced by a unified and well-principled design paradigm, called *Graph Transformer Networks*, that allows training all the modules to optimize a global performance criterion.

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, November 1998 (PDF available online)

Convolutional Network Demo from 1993 – LeNet1

This is a demo of LeNet 1, the first convolutional network that could recognize handwritten digits with good speed and accuracy […] developed between 1988 and 1993 […] at Bell Labs in Holmdel, NJ. This "real time" demo shows ran on a DSP card sitting in a 486 PC with a video camera and frame grabber card. The DSP card had a […] 32-bit floating-point DSP and could reach an amazing 12.5 million multiply-accumulate operations per second. Shortly after […], we started working with a development group and a product group at NCR (then a subsidiary of AT&T). NCR soon deployed ATM machines that could read the numerical amounts on checks, initially in Europe and then in the US. At some point in the late 90's these machines were processing 10 to 20% of all the checks in the US.

Y. LeCun "Convolutional Network Demo from 1993" (https://www.youtube.com/watch?v=FwFduRA_L6Q)

References – Most Relevant

- Y.LeCun, Y.Bengio, G.Hinton, *Deep Learning,* Nature, 2015
  see shared material folder
- Andrej Karpathy's CNN online course
  http://cs231n.github.io/
- Yann LeCun's 1998 paper on CNNs
  Gradient-Based Learning Applied to Document Recognition
  see shared material folder
- I. Goodfellow, Y. Bengio, A. Courville Deep Learnig
  https://www.deeplearningbook.org/
- Fei-Fei Li et al., CS231n: Deep Learning for Computer Vision, Stanford
  https://cs231n.github.io/convolutional-networks/
- Daniel Cremers, Introduction to Deep Learning Course, TUM
  https://cvg.cit.tum.de/teaching/ws2024/i2dl

Questions ?