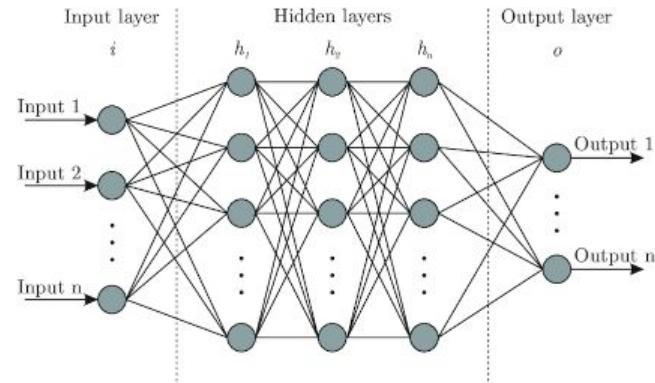


# Convolutional Neural Networks



Stéphane Lathuilière

# Last time: Multi-layer perceptron



# Recap

Data :  $(x_1, y_1), \dots, (x_N, y_N)$

Network :  $\varphi(x, w)$

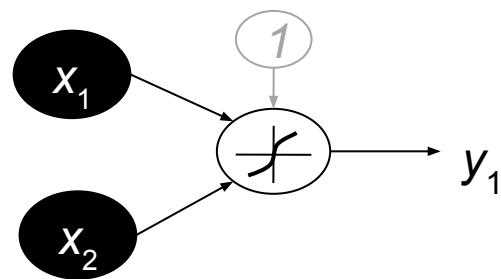
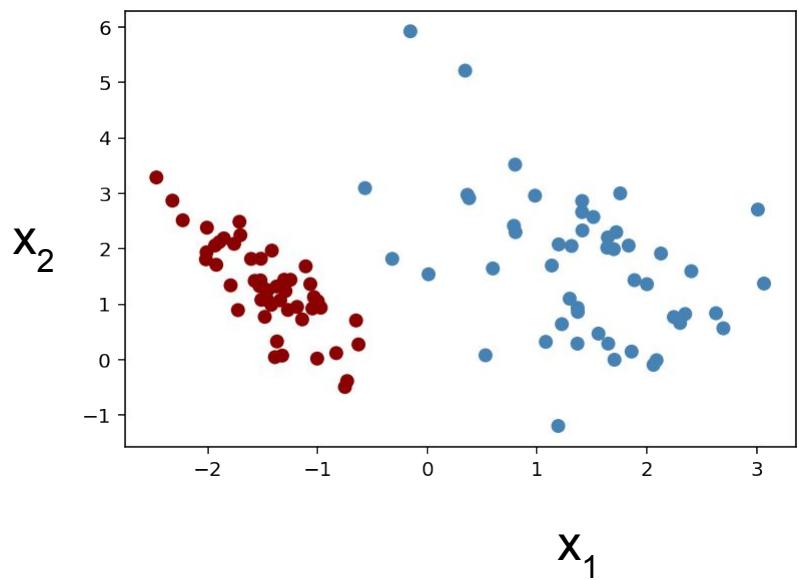
Loss :  $L(\varphi(x_n, w), y_n) = L_n(w)$

Total objective :  $E(w) = \sum_{n=1}^N L_n(w)$

GD :  $w' = w - \eta \frac{\partial E}{\partial w}$

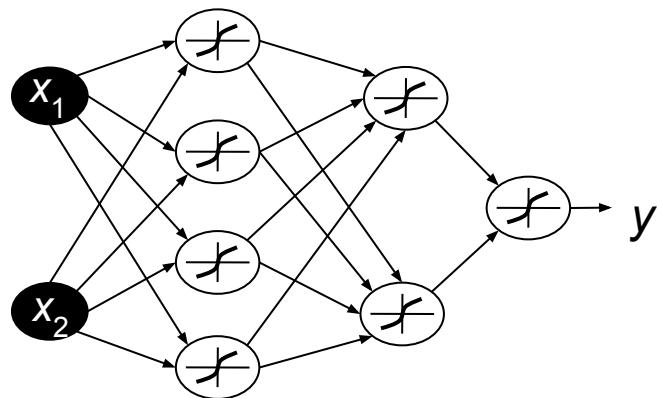
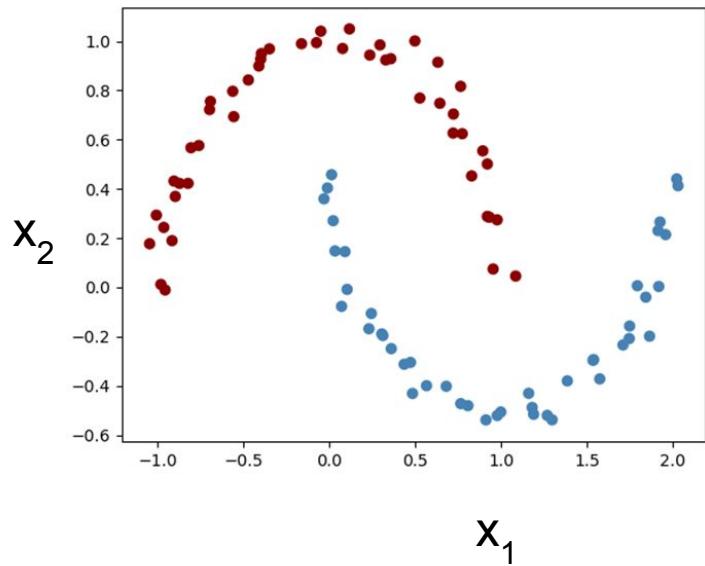
# Batch Training - Single Layer Linear Classifier Example

- Linearly separable point clouds



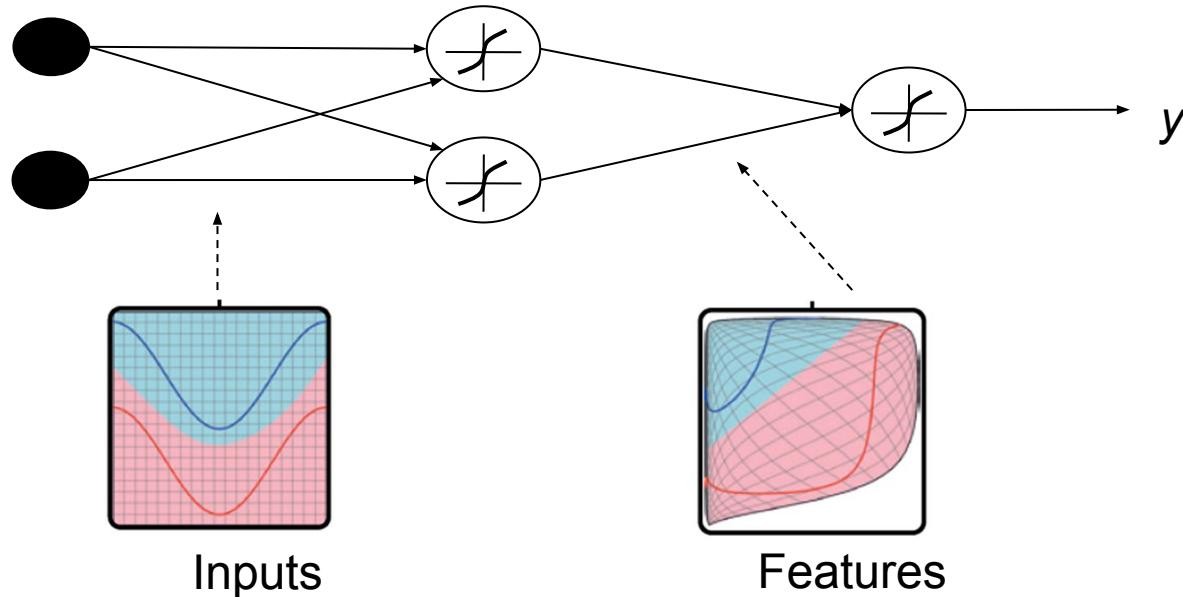
# Multilayer Classifier – Example

- Non-linearly separable classes



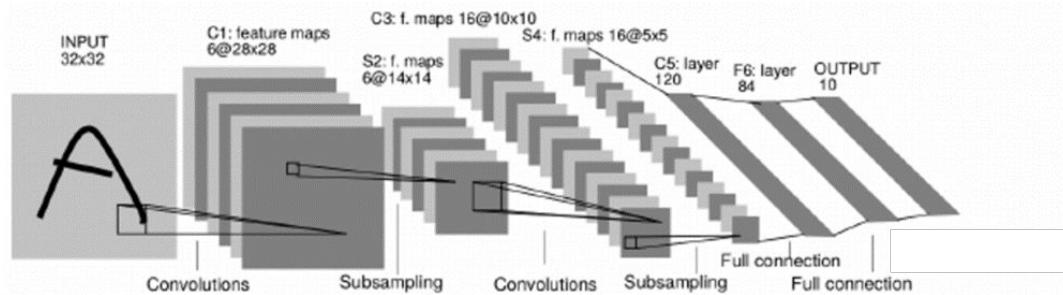
# Representation Learning

- The hidden layer(s) produces linearly separable features
- The output layer is a linear classifier

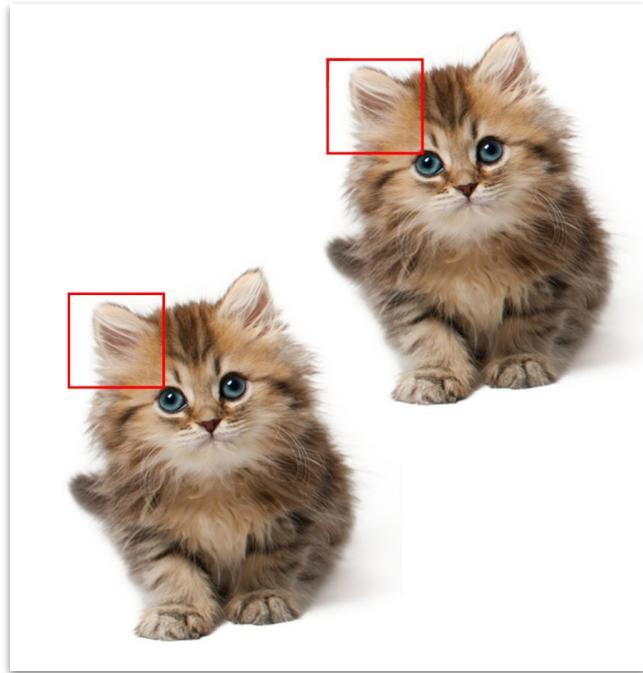


Yann LeCun, Yoshua Bengio, Geoffrey Hinton, "Deep learning", Nature, Vol 521, 2015

# ConvNets

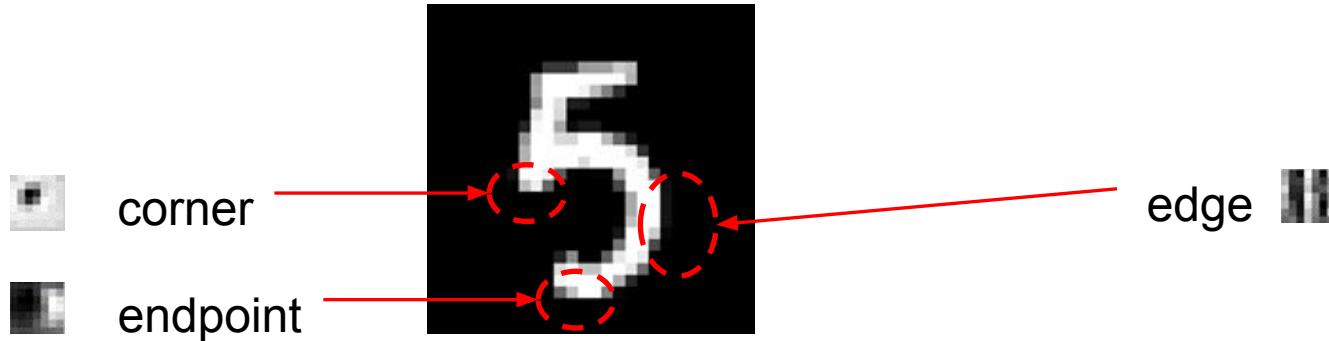


# Equivariance to shift



# Feature Learning

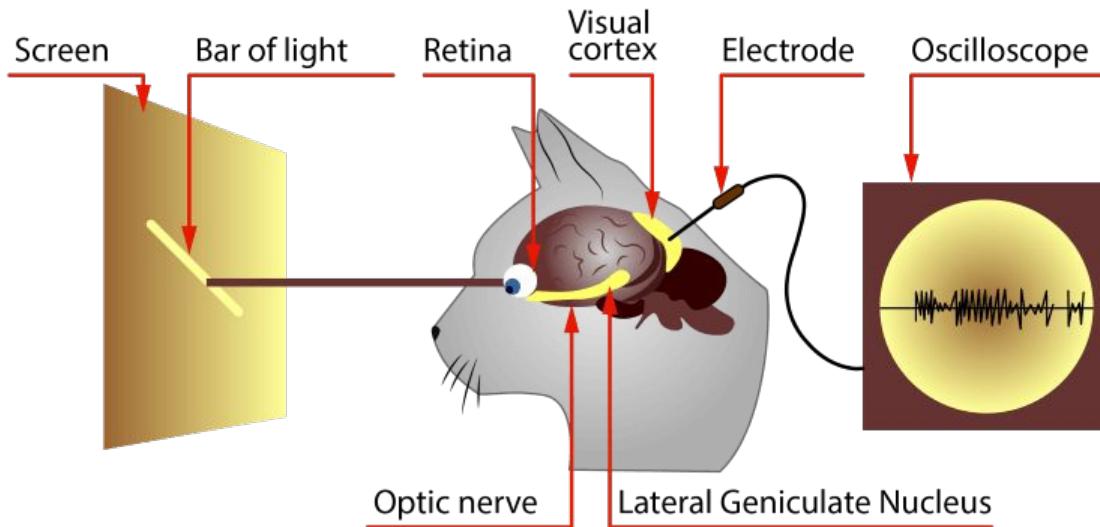
- Images characterized by *features* such as *edges*, *corners*, etc.



We know how to locate keypoints and encode features.

- Requires ad-hoc feature detector design
- Let the network learn local feature detector(s)
  - Requires new spatial topology-aware neuron structure.

# The Hubel and Wiesel experiments ('60s)



Visual cortex neurons acting as specialized **feature** detectors

- Some fire upon seeing horizontal lines, others vertical lines
- Some fire upon seeing circles, other squares

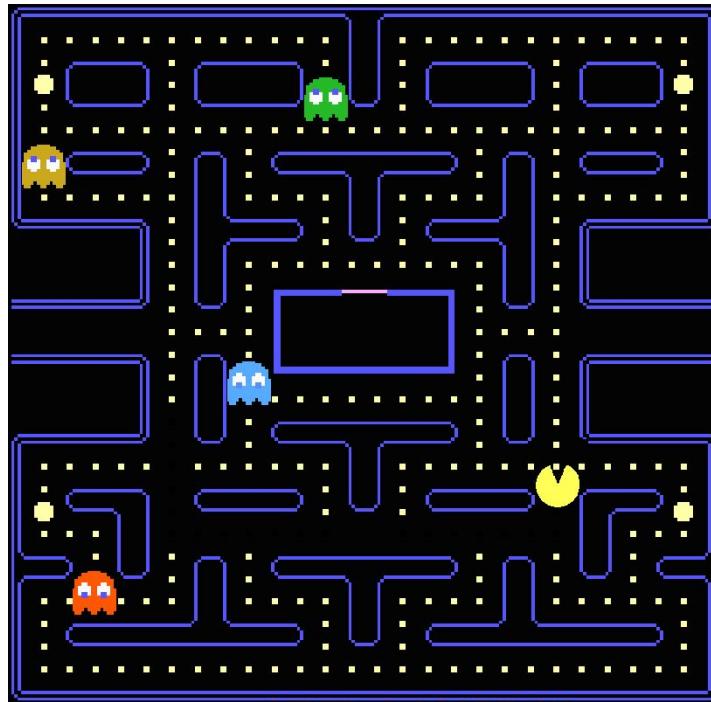
# The Hubel and Wiesel experiments ('60s)



Different detectors for different **features** (circles, crosses)

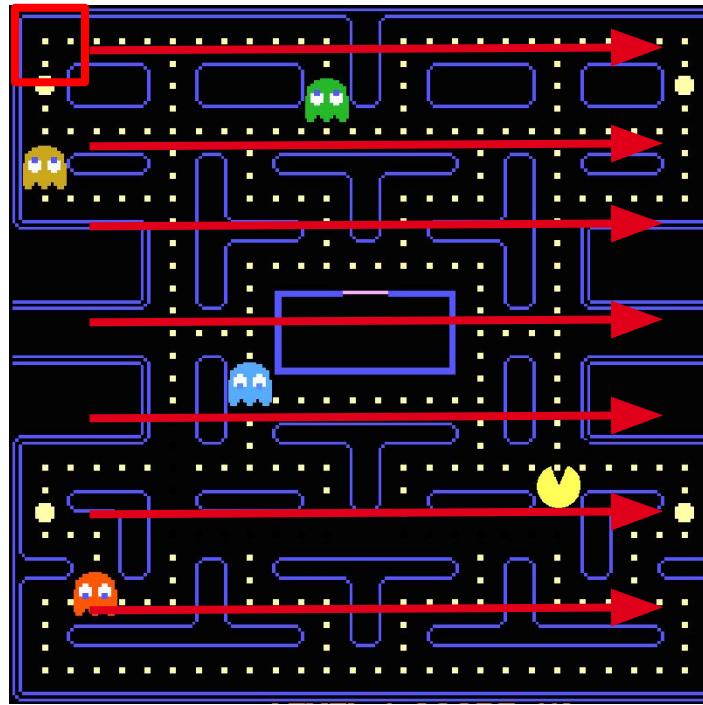
- Watch [video](#) with audio (buzz = excited neurons)

# Pacman detector



# Pacman detector

patches

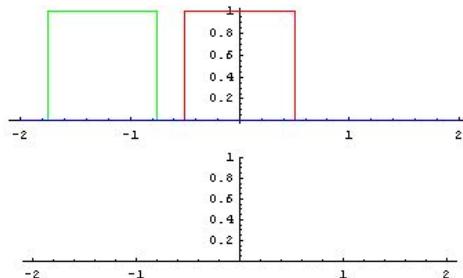


# The Convolution Operator

- 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  $f$

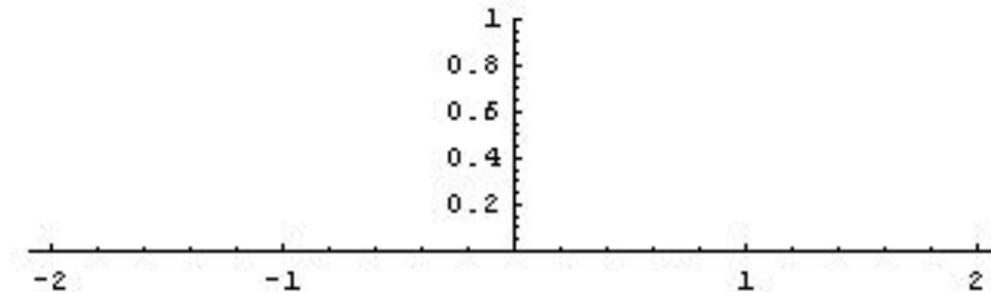
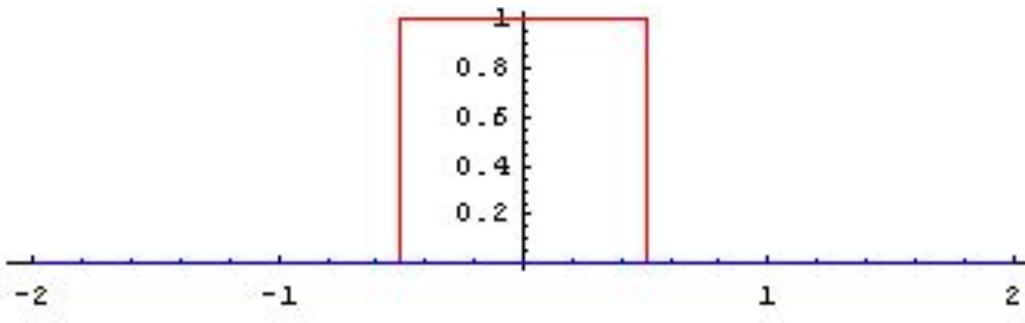


([https://fr.wikipedia.org/wiki/Produit\\_de\\_convolution](https://fr.wikipedia.org/wiki/Produit_de_convolution))

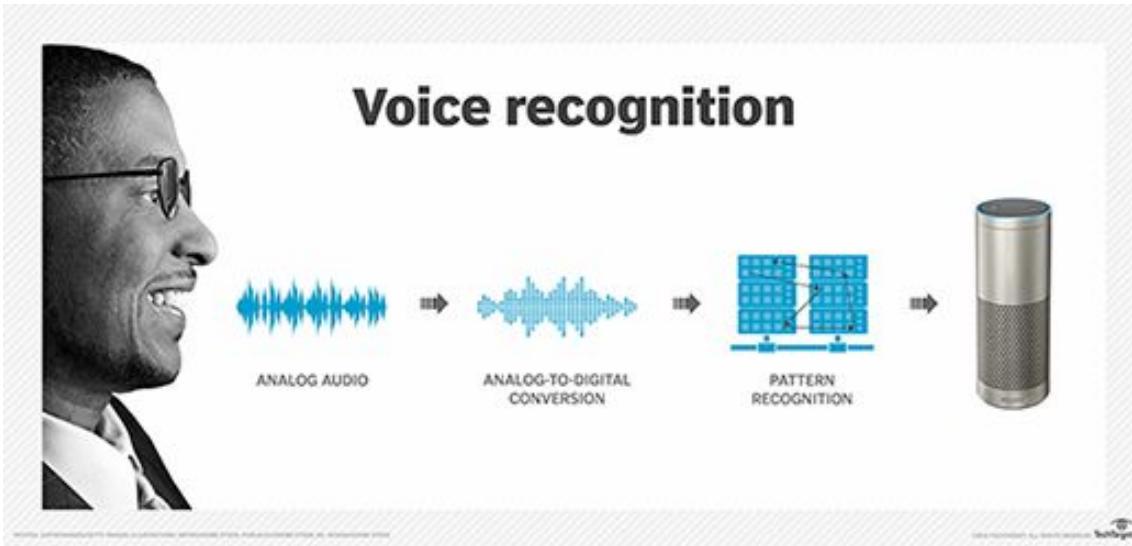
# The Convolution Operator

- 

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



# Example: Speech Recognition



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

# The 2D Convolution – An Example

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

$$\left( \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [2, 2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

# The 2D Convolution – An Example

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

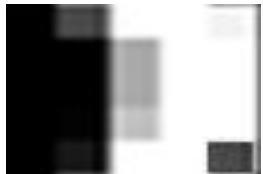
$$\left( \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [2, 2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

With bias:

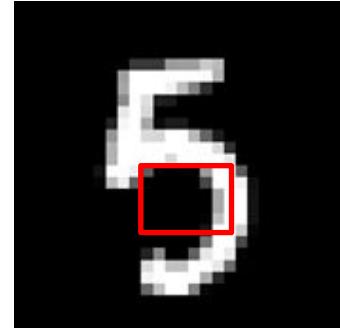
$$(k * f)(x, y) = \sum_{dx=-a}^a \sum_{dy=-b}^b k'(dx, dy) f(x + dx, y + dy) + b$$

# The 2D Convolution – An Example

- 5x5 input image crop



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



- 3x3 filter

1	0	1
0	1	0
1	0	1

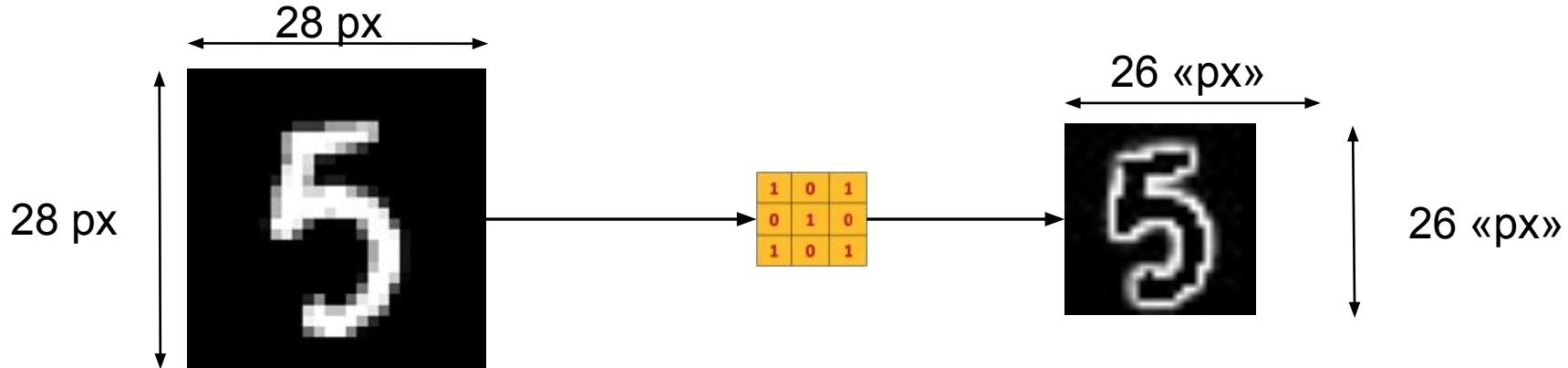
$$\sum_{dx=-a}^a \sum_{dy=-b}^b k'(dx, dy) f(x + dx, y + dy)$$

- 3x3 *feature map*

4	3	4
2	4	3
2	3	4

# Narrow Convolution

- Feature map smaller than input image



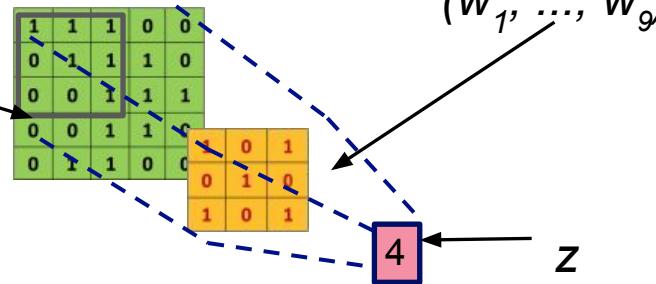
# Convolutional Layers – An Example

**It's your turn to play!**

# The Convolutional Neuron

## □ 2D convolution operation

$(x_1, \dots, x_{25})$

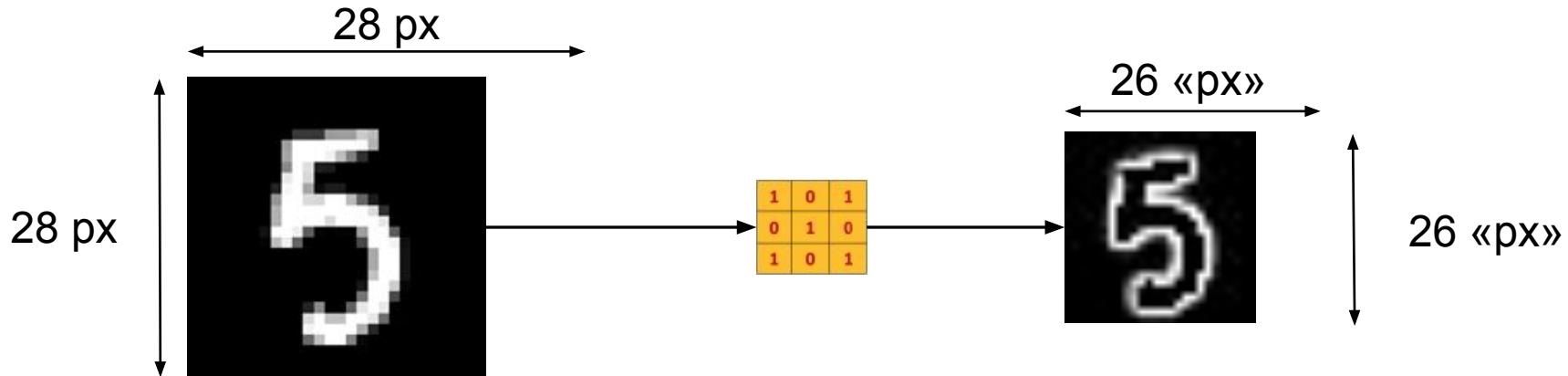


# The Convolutional Neuron

- 
- 
- A convolution layer can be seen as a fully connected layer where
  - we impose 0 outside the kernel window
  - the weights are shared over different locations
-

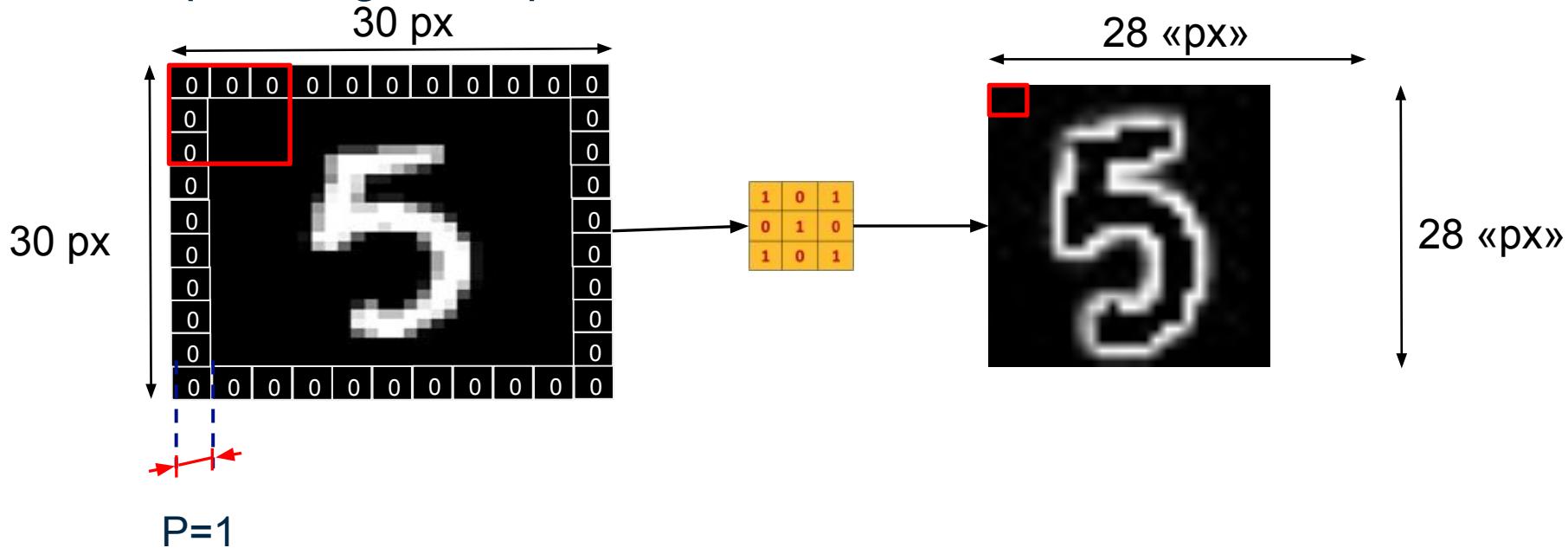
# Narrow Convolution

- Feature map smaller than input image



# Wide Convolution

- Zero-padding all around image ( $F=3$ ,  $S=1$ ,  $P=1$ )
  - Input image size preserved



# The Convolutional Layer – Multiple Input Channels

- Most images RGB, not grayscale

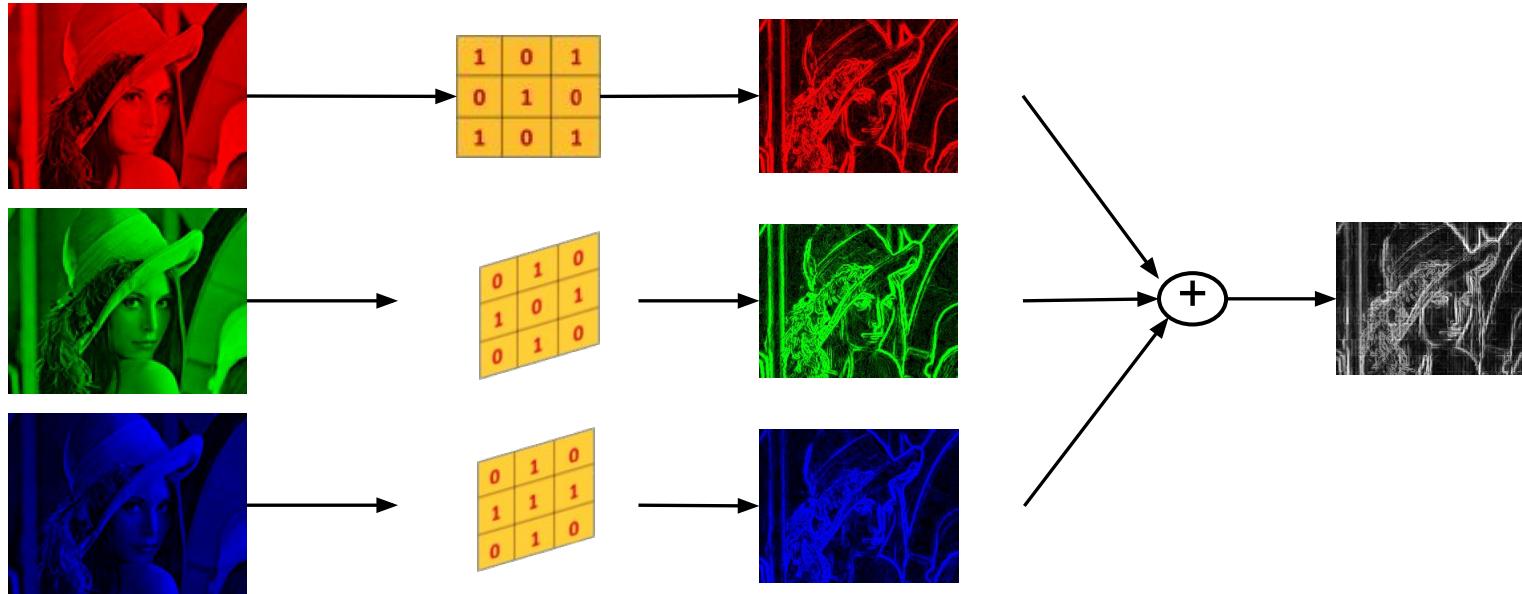


=



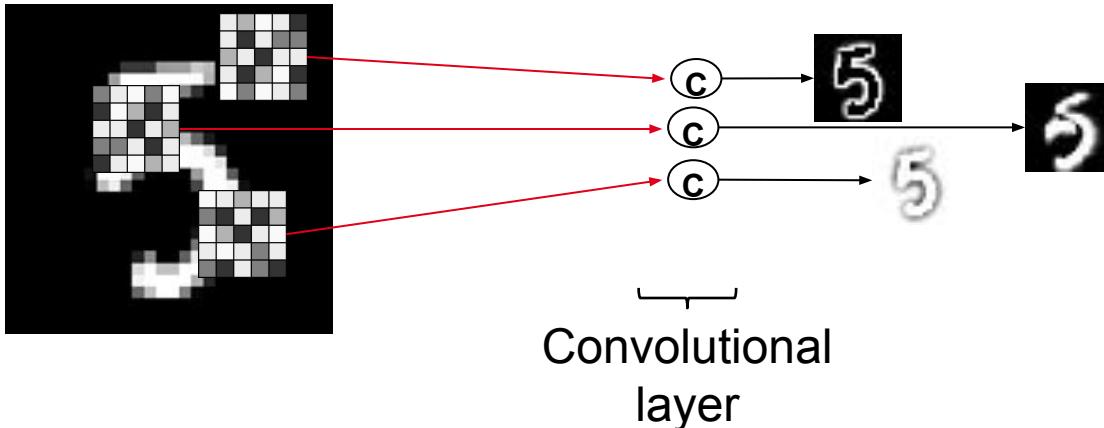
# The Convolutional Layer – Multiple Input Channels

- A **different** filter is independently applied to every channel
- Co-located features are summed



# The Convolutional Layer - Complexity

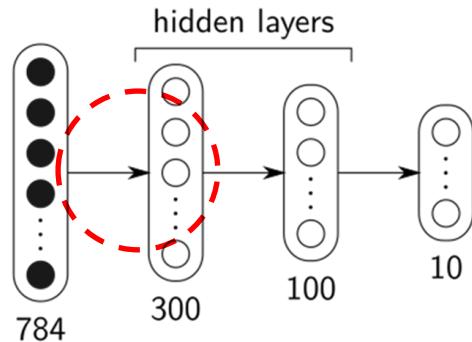
- Number of parameters per FC layer was (e.g.,  $n_{in} = 28^2$ )  
 $m * (1+n)$
- Number of parameters per convolutional layer (e.g., F=3)  
 $m * (1+F^2) * \#ch$



# Fully Connected LeNet300

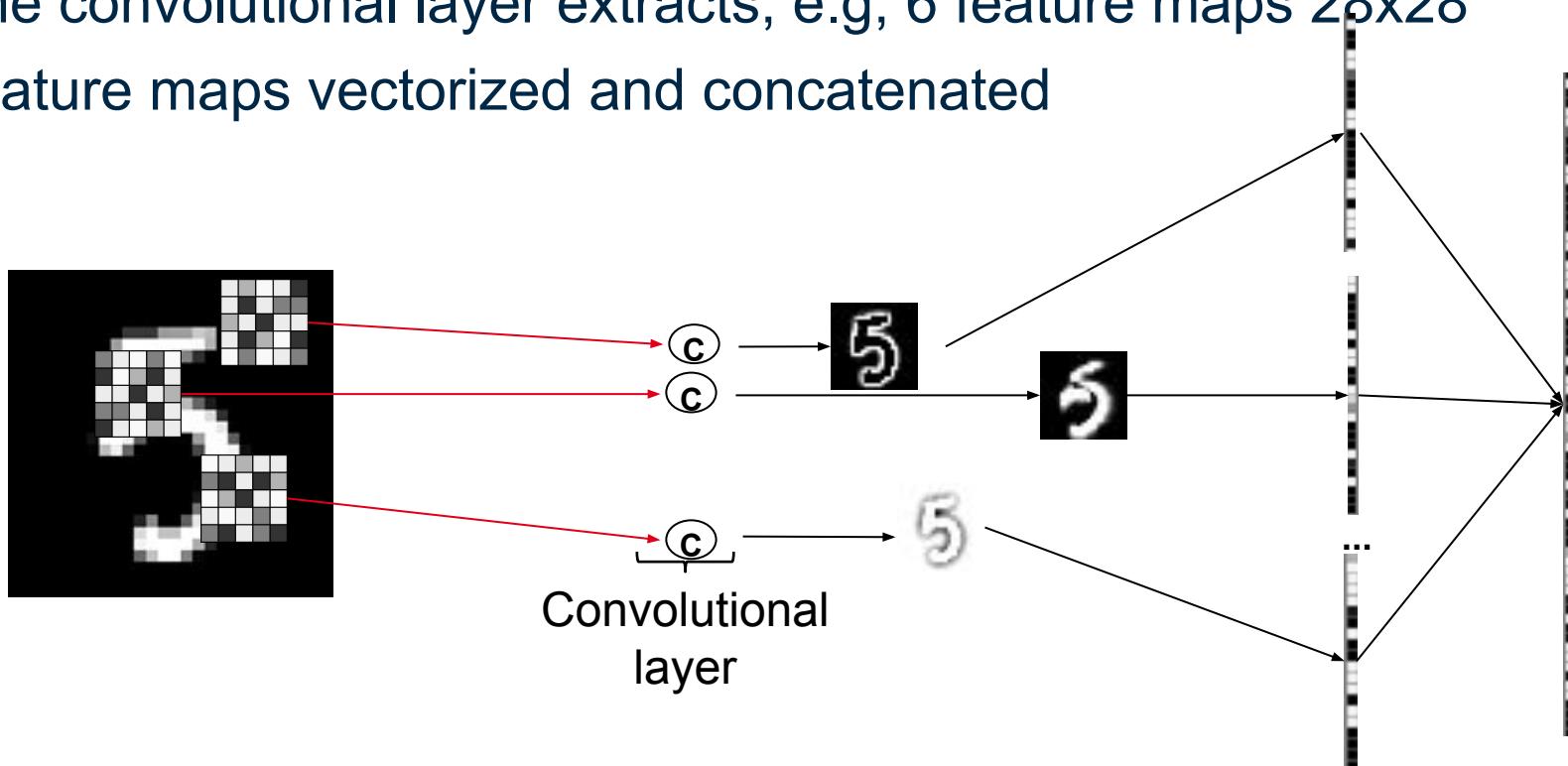
- One of LeNet300 problems was 1st FC layer complexity

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



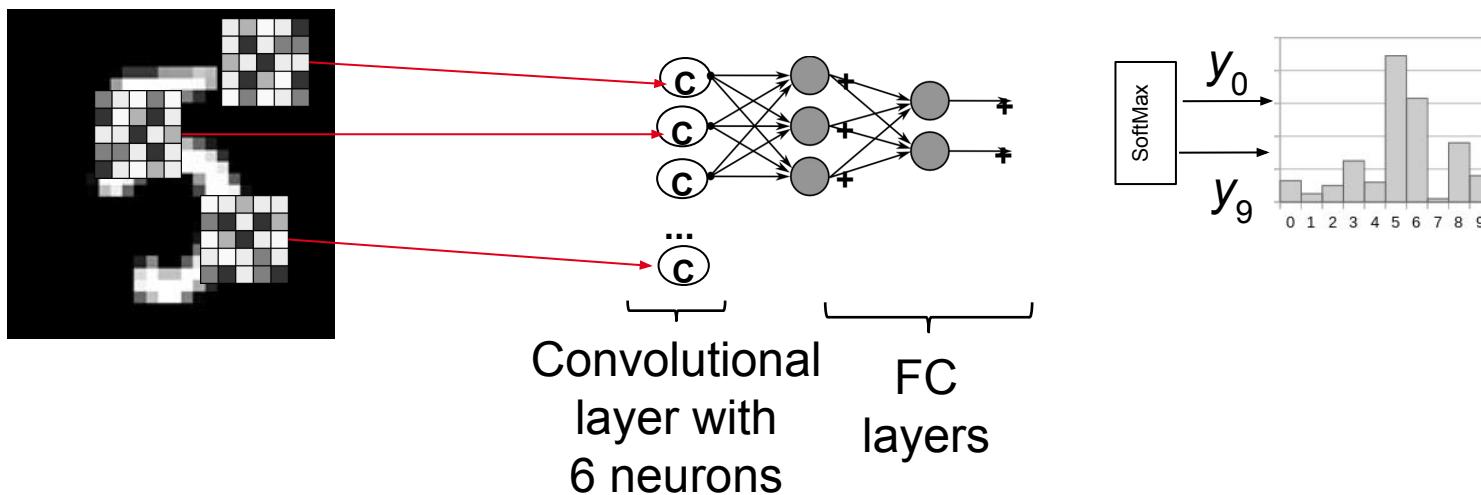
# Convolutional LeNet300

- One convolutional layer extracts, e.g, 6 feature maps 28x28
- Feature maps vectorized and concatenated



# Convolutional LeNet300

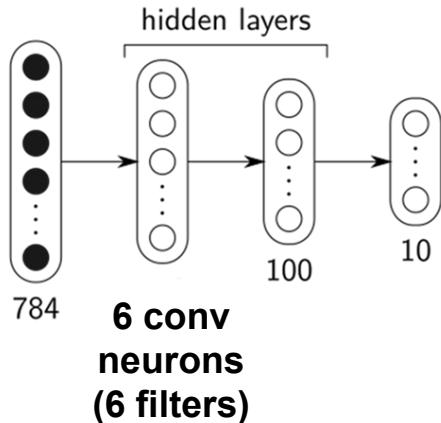
- One convolutional layer extracts, e.g, 6 feature maps 28x28
- Feature maps vectorized and concatenated
- One or more FC layers classify feature maps
- Network output is class probability distribution ( $C=10$ )



# Convolutional LeNet300 - Complexity

- 1 st layer complexity drops 230k -> 60 params!

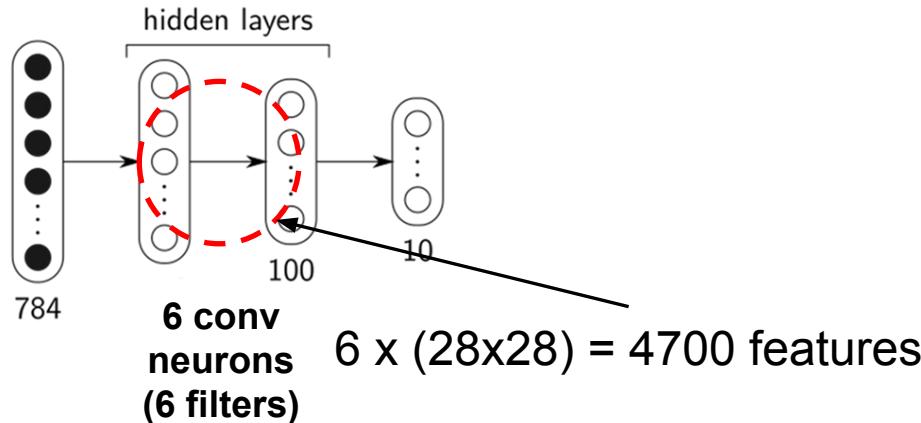
	Fully Connected		Convolutional	
Layer	Type	Complexity [prms]	Type	Complexity [prms]
1	FC-300	$300 * (28*28) = 230k$	Conv-6	
2	FC-100	$100 * 300 = 30k$	FC-100	
3	FC-10	$10 * 100 = 1k$	FC-10	



# Convolutional LeNet300 - Complexity

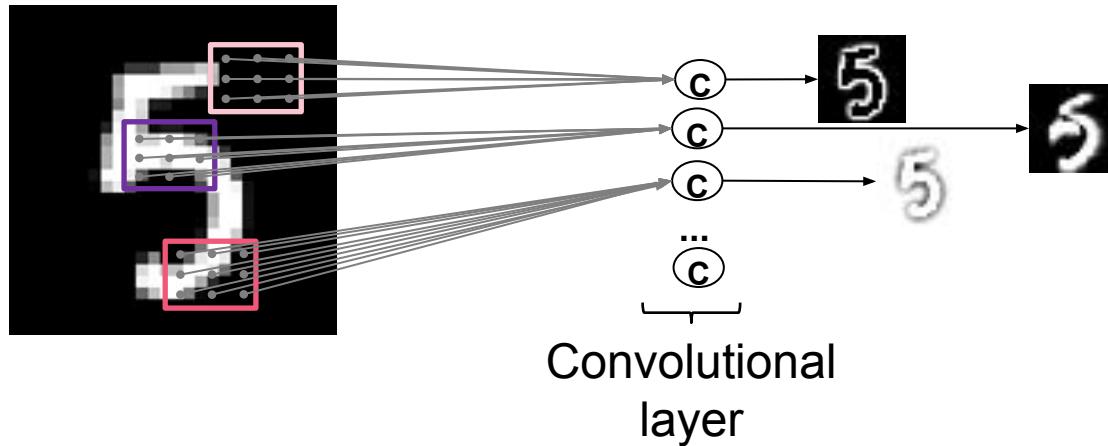
- Total complexity soars 260k → 400k params!

	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	$10 * 100 = 1k$
	$\sim 260k$		$\sim 400k$	



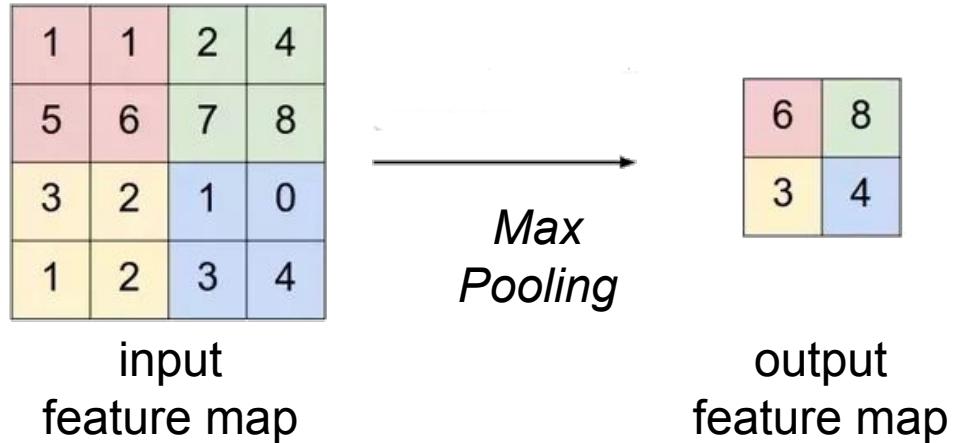
# Convolutional LeNet300

- Feature maps can be seen as *filtered images*
  - Feature maps can be subsampled as images are



# The MaxPooling Layer

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



Why just not averaging ? Maxima suppression

# The Convolve-and-Pool Pattern

- Each Conv layer is followed by MaxPool layer

28x28  
image



28x28  
feature map

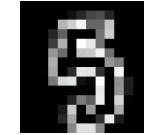


5x5  
filters

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1
0	1	0	0	1
0	1	1	0	0

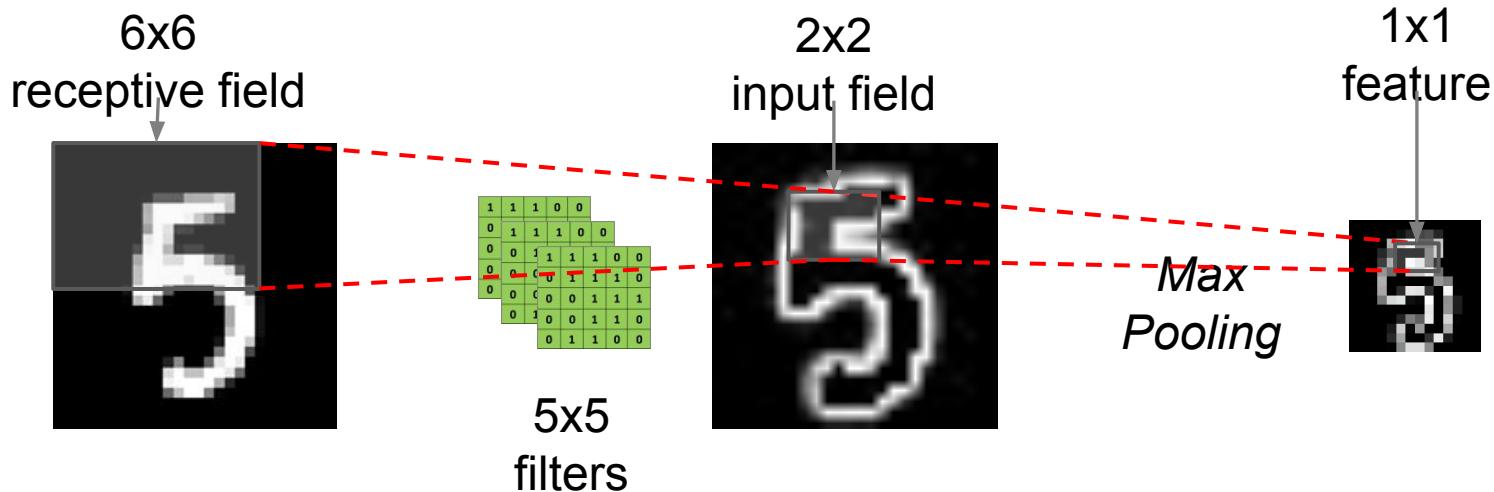
14x14  
feature map

*Max  
Pooling*



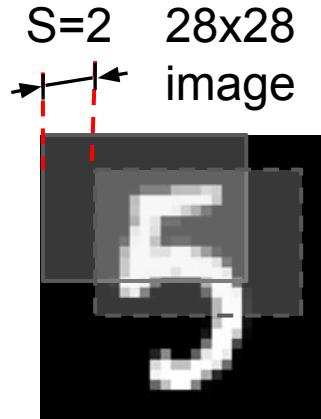
# The Convolve-and-Pool Pattern

- Each Conv layer is followed by MaxPool layer
- Increases each feature *receptive field*
  - In case  $F=5$ ,  $S=1$ ,  $P=2$ : from  $5 \times 5$  to  $6 \times 6$



# The Convolve-with-Stride Pattern

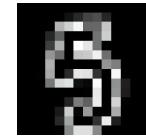
- Each Conv has stride S=2, no MaxPooling
  - No non-maxima suppression
  - Standard in recent architectures



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1
0	1	0	0	1
0	1	1	0	0

5x5  
filters

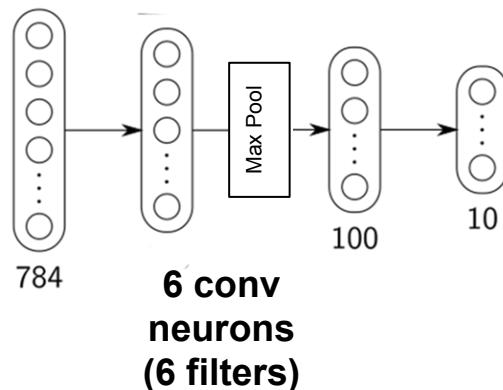
14x14  
feature map



# Convolutional LeNet300 - Complexity

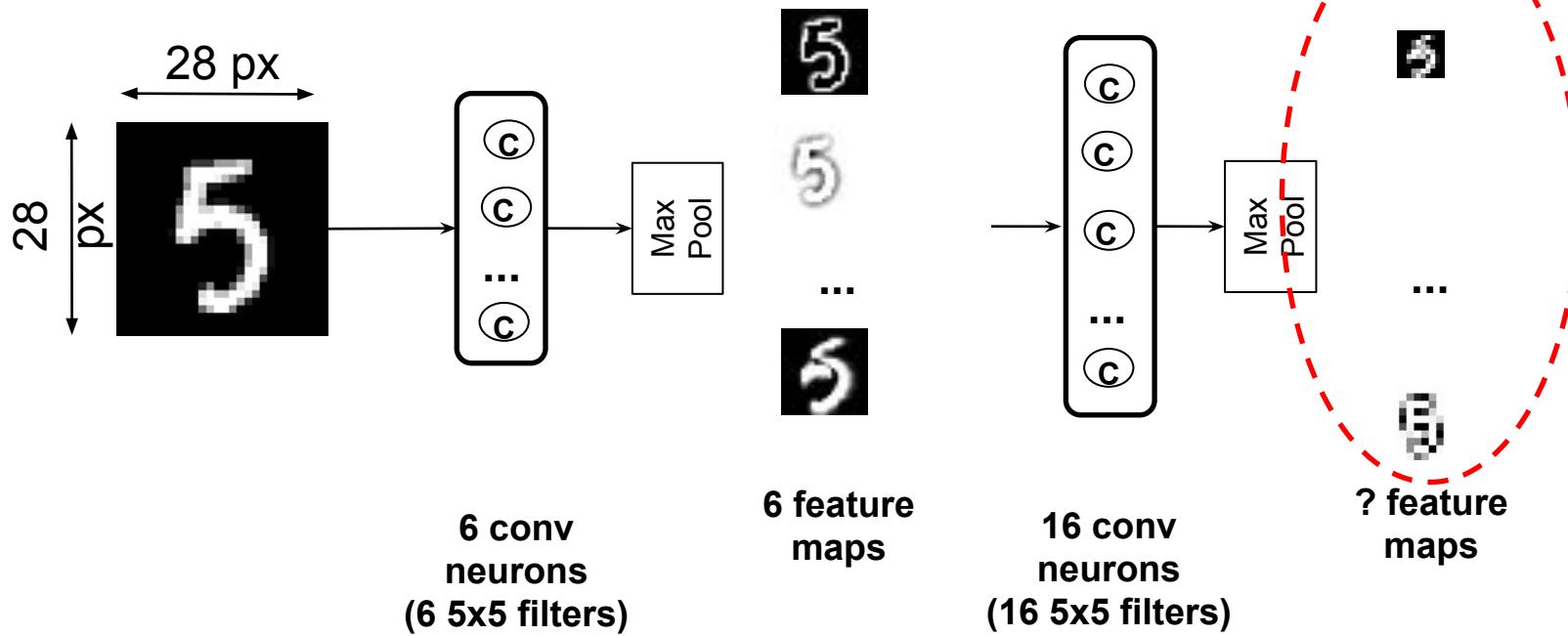
- Complexity from ~260k to ~160k params thanks to *Maxpooling*

Layer	Fully Connected		Convolutional	
	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	$100 * (6 * (14x14)) = 400k$
3	FC-10	$10 * 100 = 1k$	FC-10	$10 * 100 = 1k$
Tot		~260k		~118k



# Stacking Convolutional Layers

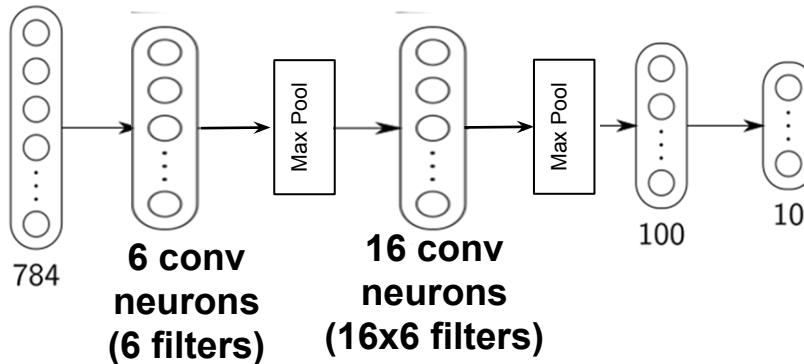
- Output of the first *conv-and-pool* layer
  - Input to second *conv-and-pool* layer



# Convolutional LeNet300 - Complexity

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

Layer	Fully Connected		Convolutional	
	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



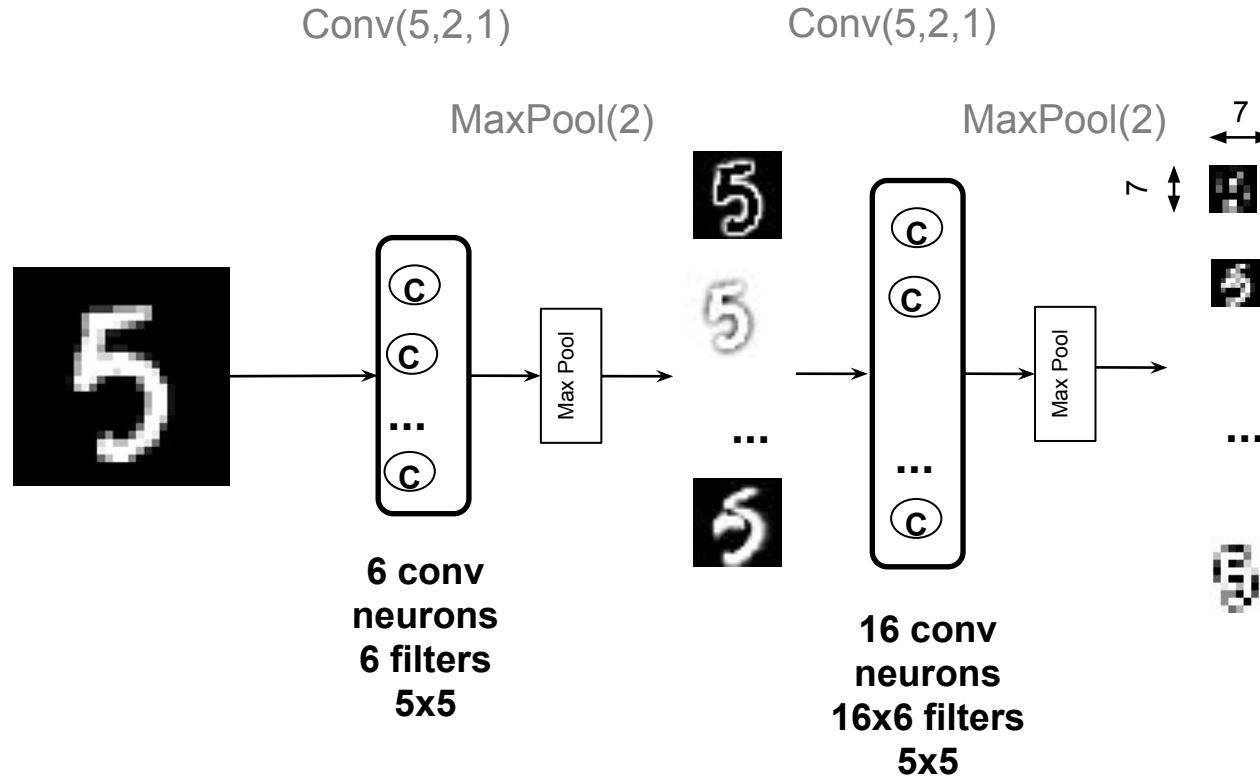
# Convolutional LeNet300 – Performance

- Experiments on MNIST 32x32 dataset

Network	Num. Layers	Error [%]
<i>Fully connected LeNet300</i>	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
<i>Convol. LeNet300</i>	2 Conv (3 F), 1 output FC ( <i>LeNet1</i> )	1.7
	2 conv (6+16 F), 3 FC layer	0.95

Better performance for lower complexity

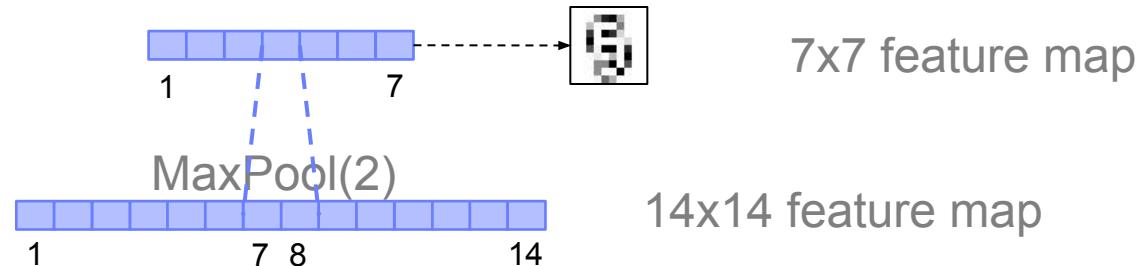
# Convolutional LeNet300 – Receptive Field



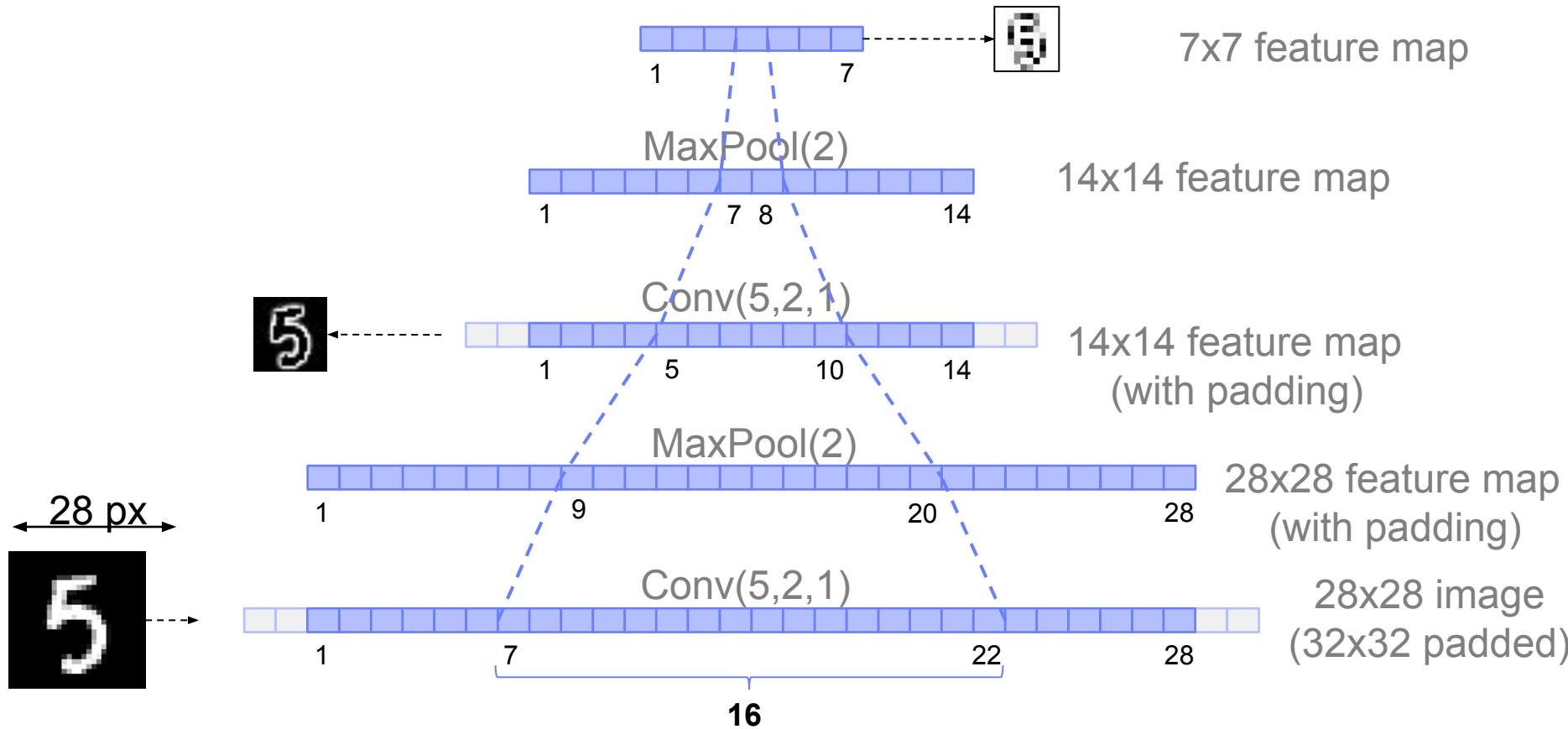
# Convolutional LeNet300 – Receptive Field (1-D)



# Convolutional LeNet300 – Receptive Field (1-D)

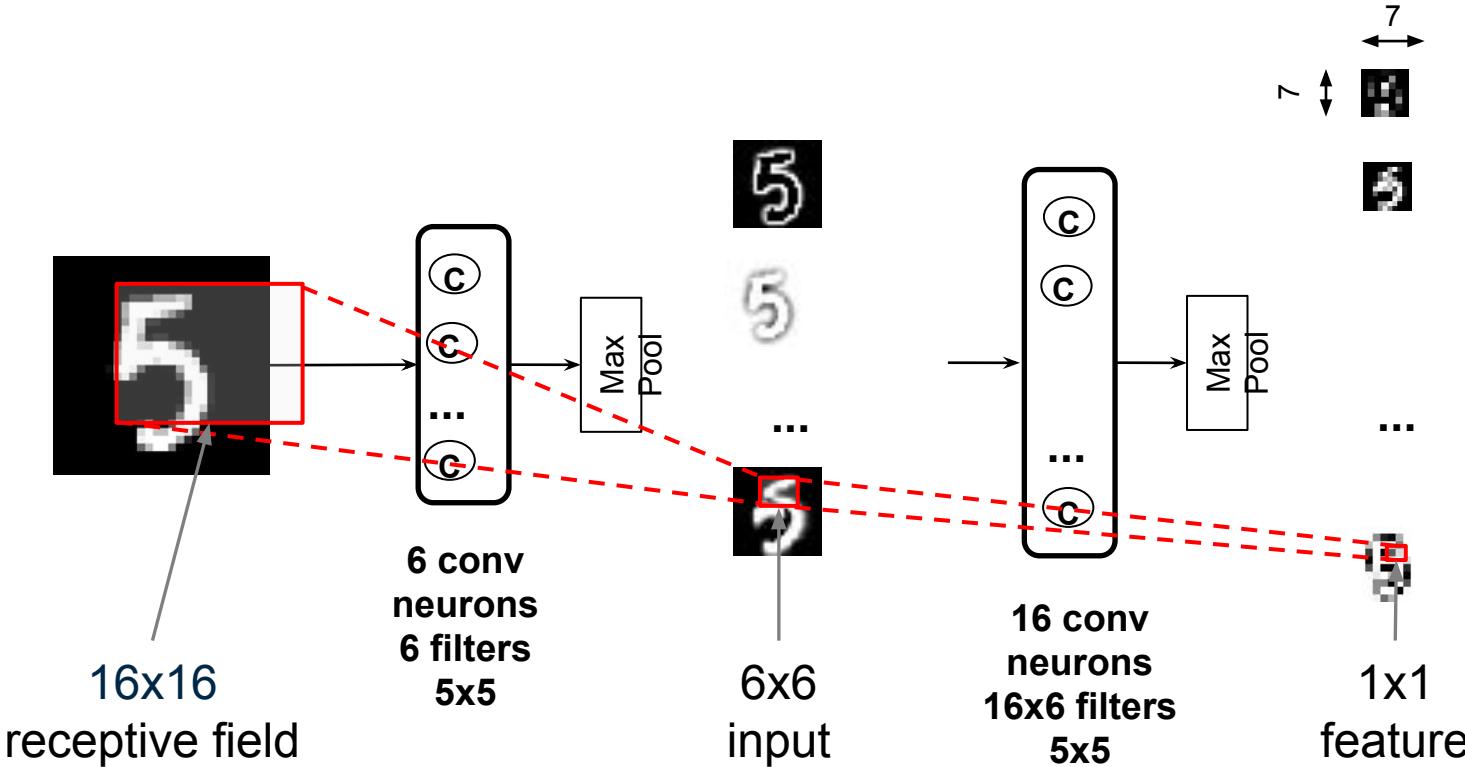


# Convolutional LeNet300 – Receptive Field (1-D)

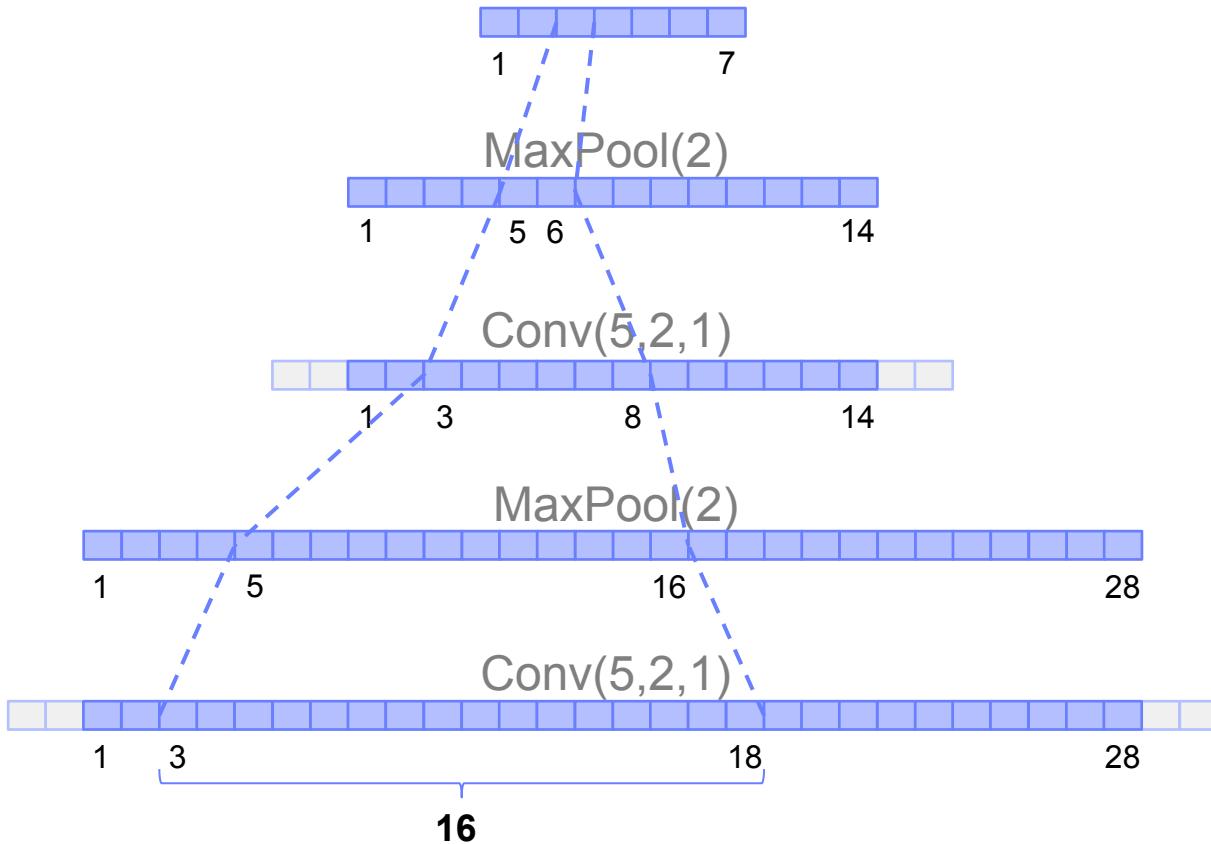


# Convolutional LeNet300 – Receptive Field

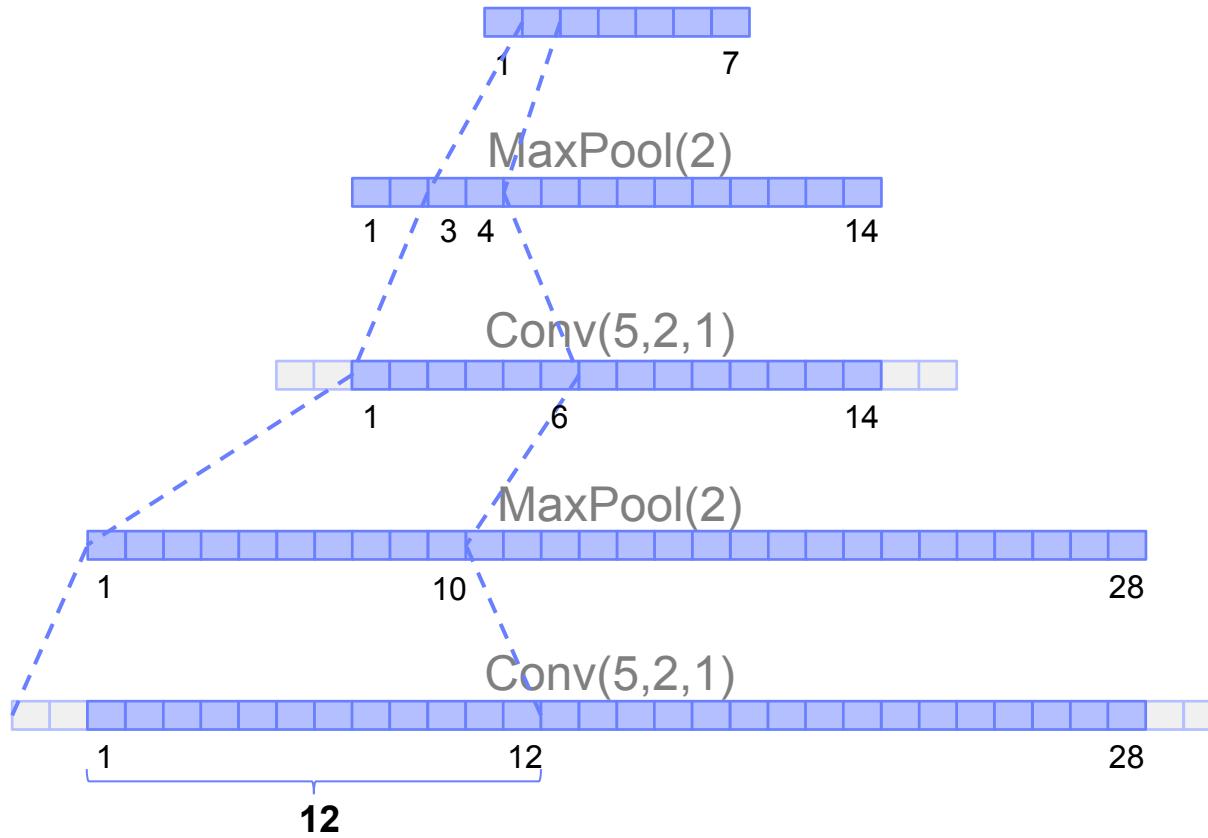
- This holds for *central* features in the last feature map



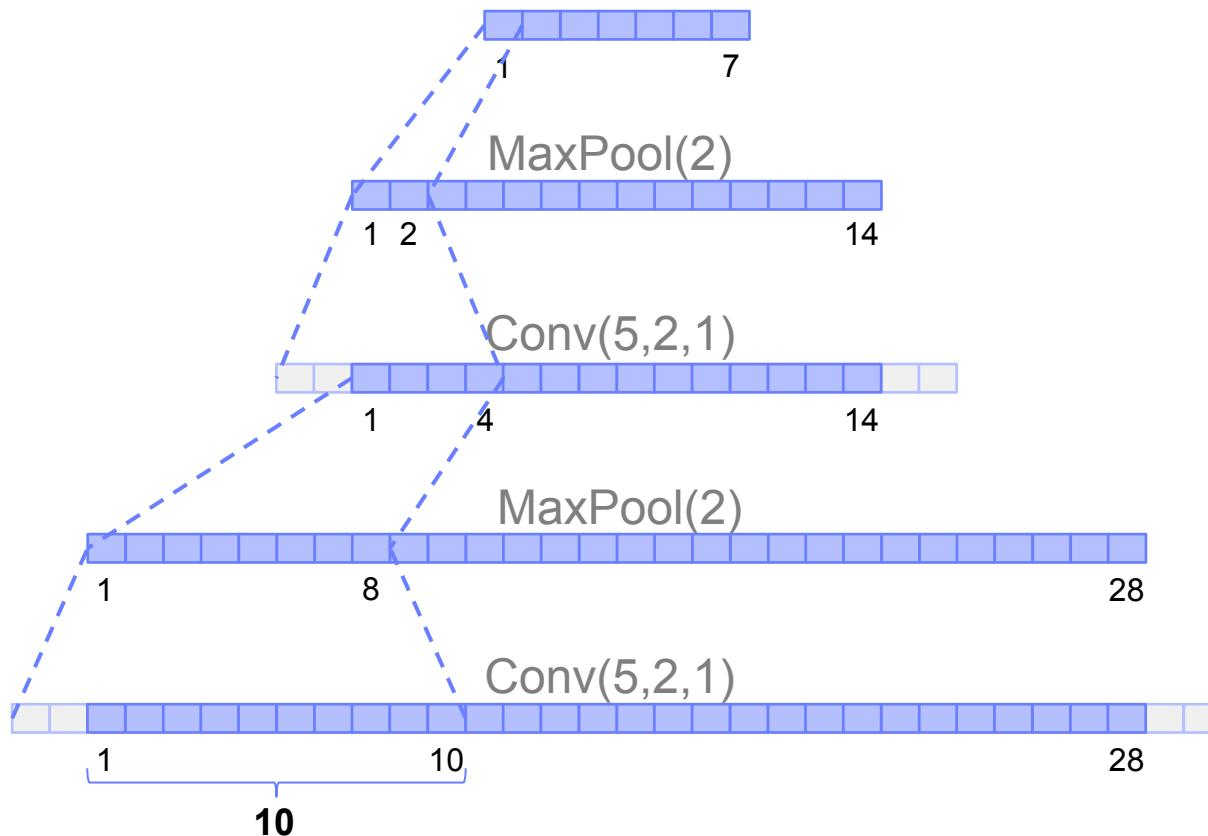
# Convolutional LeNet300 – Receptive Field (1-D)



# Convolutional LeNet300 – Receptive Field (1-D)



# Convolutional LeNet300 – Receptive Field (1-D)



# Convolutional LeNet300 – Receptive Field (1-D)

- Not all pixels contribute to the final FMs equally
  - Pixels on the border contribute less

	Pixel																												
FM#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	
1	1	1	1	1	1	1	1	1	1	1																			
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
3			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
4					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
5							1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6										1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7																													
Tot	2	2	3	3	3	3	4	4	4	4	4	3	3	3	3	4	4	4	4	4	4	4	3	3	3	2	2		

Receptive field computation for convnets <https://distill.pub/2019/computing-receptive-fields/>

# Convolutional LeNet300 – Receptive Field (1-D)

- Not all pixels contribute to the final FMs equally
  - Pixels on the border contribute less
- Narrow convolution input stride  $2^{\wedge} \# \text{maxpools}$ 
  - Features only partially overlap

	Pixel																											
FM#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	1	1	1	1	1	1	1	1	1																			
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6						4																						
7								4																				
Tot	2	2	3	3	3	3	4	4	4	4	4	3	3	3	3	3	4	4	4	4	4	4	3	3	3	3	2	2

Receptive field computation for convnets <https://distill.pub/2019/computing-receptive-fields/>

# Convolutional LeNet300 – Receptive Field (1-D)

- Not all pixels contribute to the final FMs equally
  - Pixels on the border contribute less
- Narrow convolution input stride  $2^{\wedge} \# \text{maxpools}$ 
  - Features only partially overlap
- Receptive field up to  $16 \times 16$

Pixel

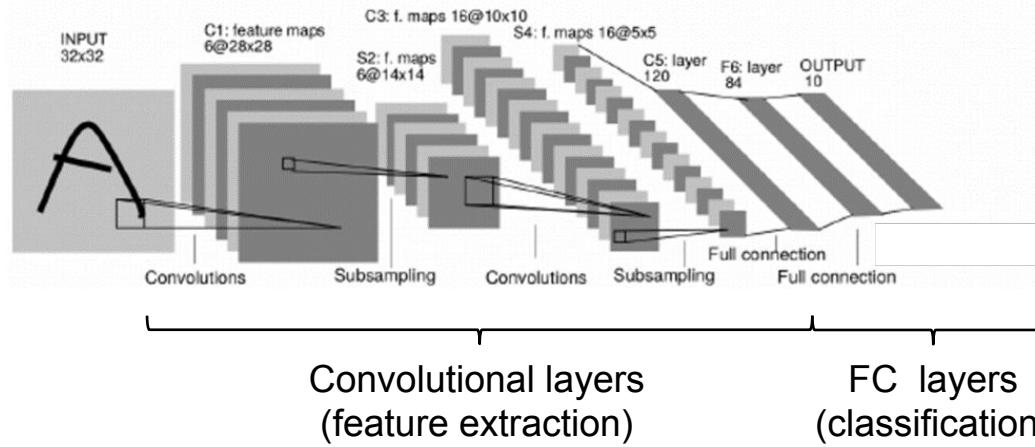
Possible to detect features larger than filters

FM#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Tot	2	2	3	3	3	3	4	4	4	4	4	3	3	3	3	4	4	4	4	4	4	4	3	3	3	3	2	2	

Receptive field computation for convnets <https://distill.pub/2019/computing-receptive-fields/>

# 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

PROC. OF THE IEEE, NOVEMBER 1998

1

## 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 to 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 global 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.

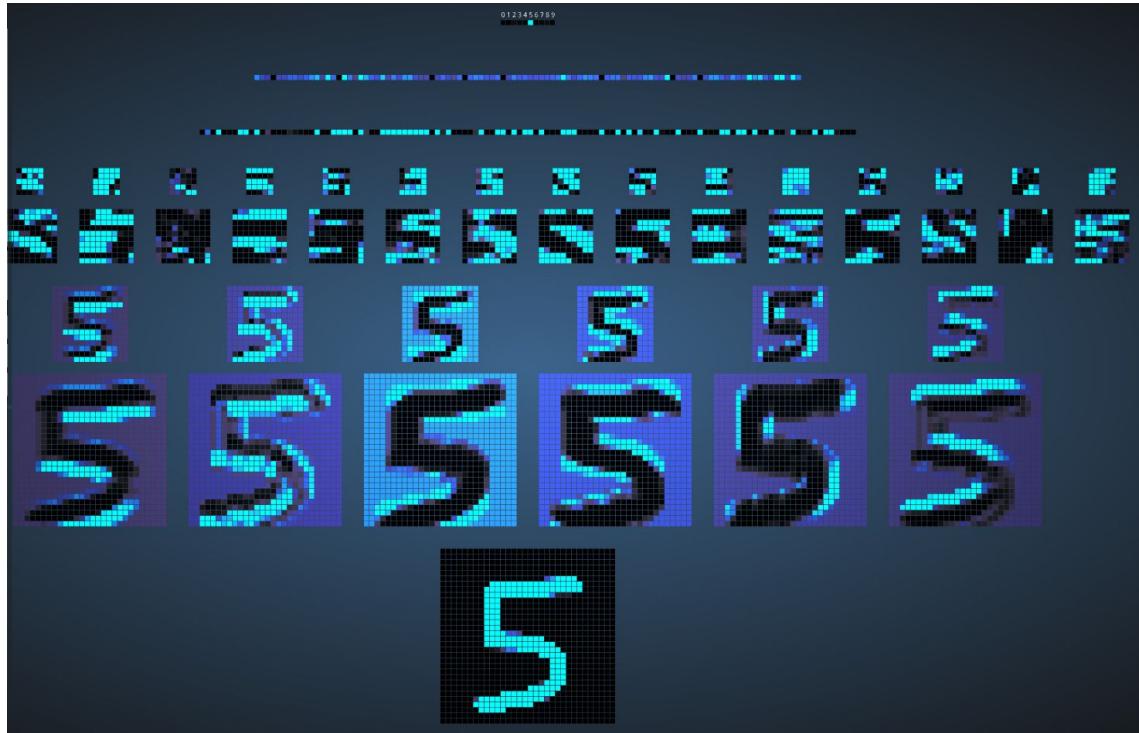
### I. INTRODUCTION

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.

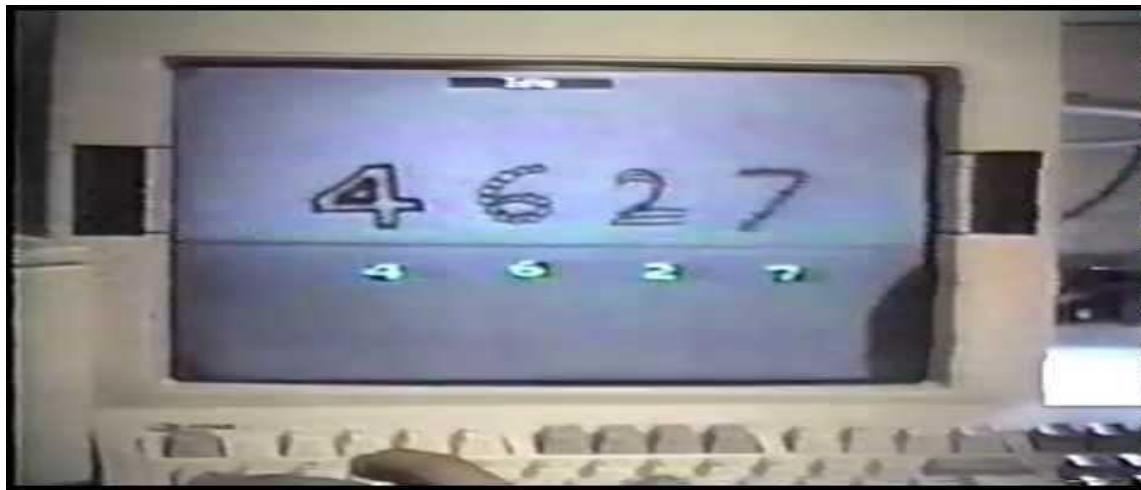
Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition,  
Proceedings of the IEEE, November 1998 (PDF available online)

# Convolutional Networks – Visualized



<http://scs.ryerson.ca/~aharley/vis/conv/flat.html>

# 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.*

# 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/>

# Questions ?

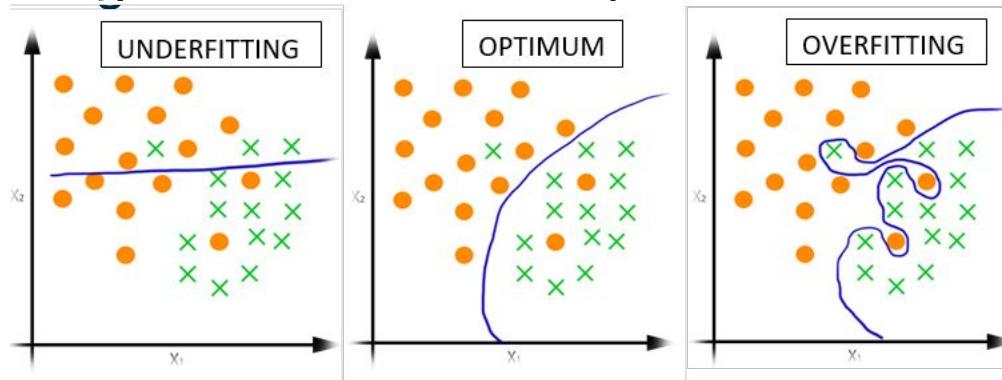


## Quiz:

- Mini-bach vs batch
- Kernel size
- Number of channels
- Padding
- Stride

# 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

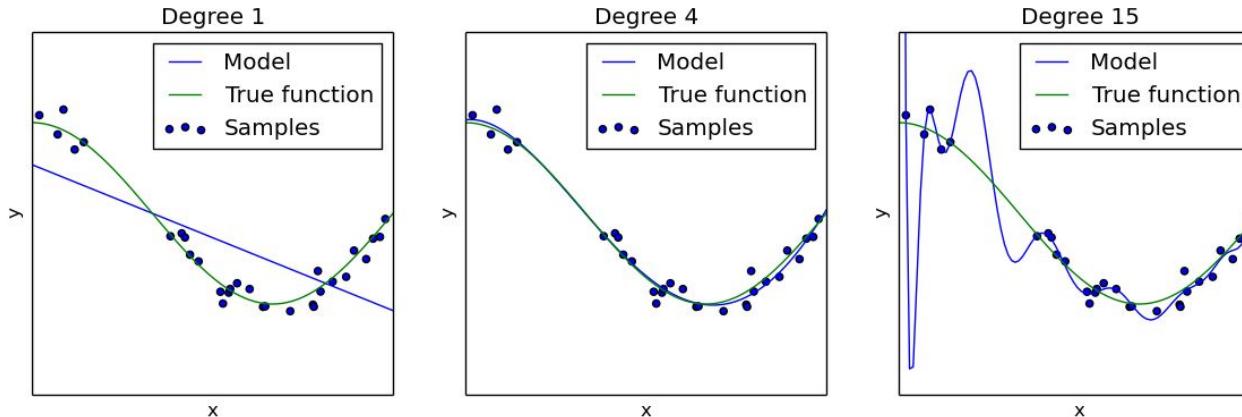


# Overfitting

- Example problem: polynomial curve fitting

$$y = G(x) + N(x)$$

- We want to fit the generator function
- As we add parameters (poly degree) we better fit samples
- Overfitting -  $\rightarrow$  fitting the noise  $N$  rather than generator  $G$



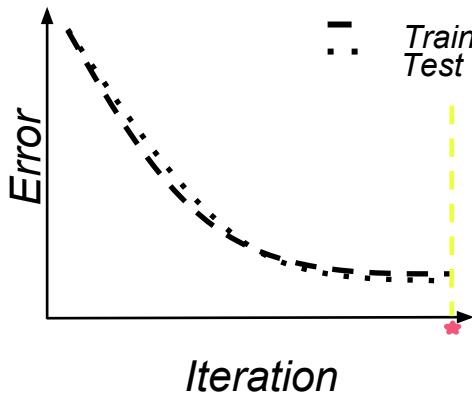
# Overfitting

- Example problem: polynomial curve fitting

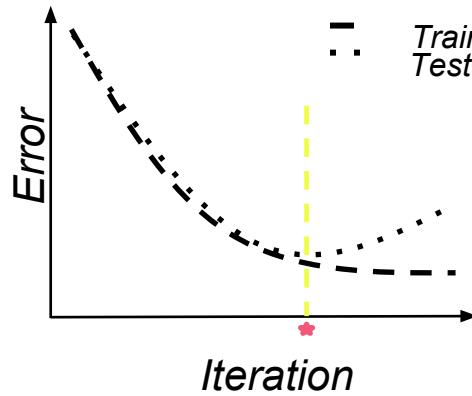
$$y = G(x) + N(x)$$

- We want to fit the generator function
- As we add parameters (poly degree) we better fit samples
- Overfitting - > fitting the noise  $N$  rather than generator  $G$

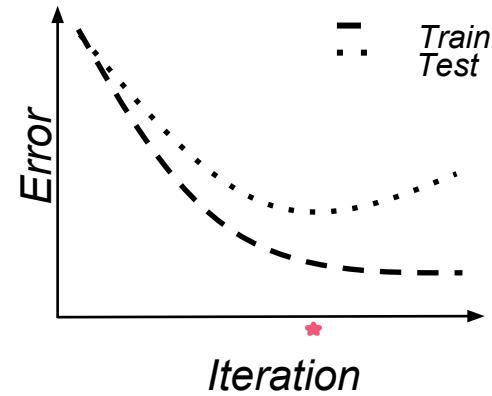
# Verifying Overfit



Underfitting?



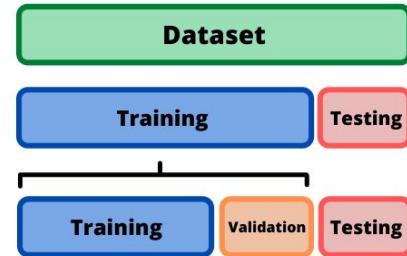
Good fit  
until ☆



Overfit

# Avoiding Overfit: 5 techniques

- How to avoid overfit ?
  - **Early stop:** require a validation set :  
->Split annotated dataset into *train*, *validation* and *test*
    - E.g: 60% train, 20% validation and 20% test
  - **Lower capacity/complexity** model (fewer parameters)
    - Still suitable for problem ?
  - Introduce **regularization** factor  $\lambda$ 
$$J(\mathbf{w}) = E(\mathbf{w}) + \lambda R(\mathbf{w})$$



# Regularizer

- Introduce a regularization term  $R(w)$

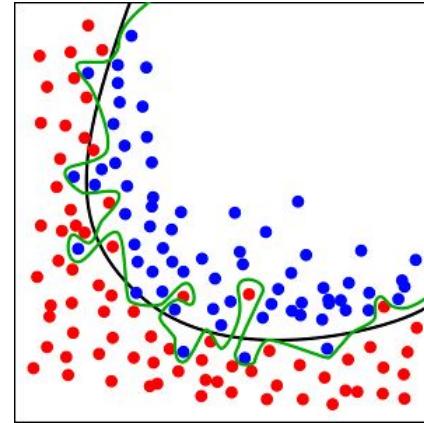
$$R(w) = \|w\|_2^2$$

- Minimize  $J = E + \text{regularization term } R(w)$

$$J = E(w, x) + \lambda R(w)$$

- Effect

- *Penalizes* solutions with large weights
  - *Promotes* solutions with smaller weights

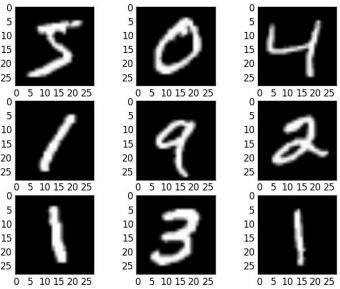


# Data Augmentation - MNIST

- Different people may write same digit in different ways
  - More intra-class variance



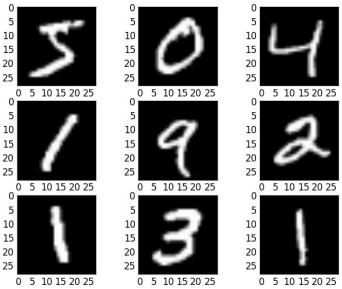
# Data Augmentation - MNIST



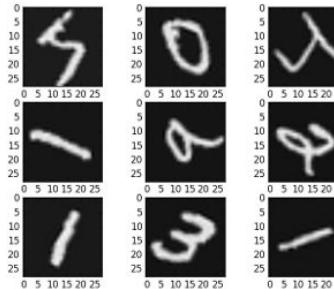
Original samples

<https://www.codesofinterest.com/2018/02/using-data-augmentations-in-keras.html>

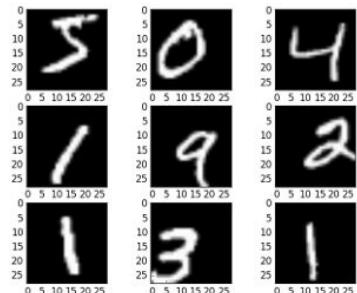
# Data Augmentation - MNIST



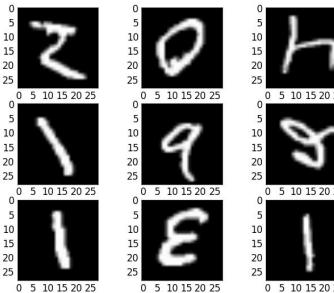
Original samples



Random rotations



Random shift



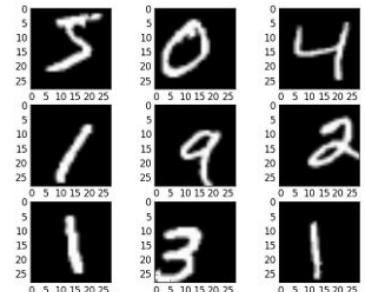
Random flips

<https://www.codesofinterest.com/2018/02/using-data-augmentations-in-keras.html>

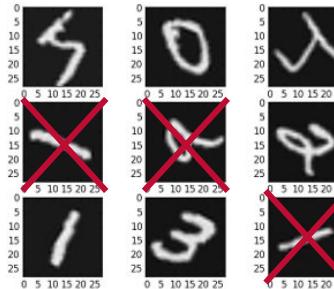
# Data Augmentation - MNIST



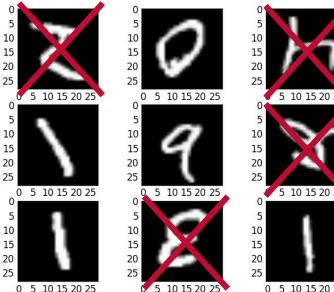
Original samples



Random shift



Random rotations



Random flips

<https://www.codesofinterest.com/2018/02/using-data-augmentations-in-keras.html>

# Data Augmentation – LeNet300 over MNIST

- Train set size does not change (50k images)
- Random augmentation at each epoch

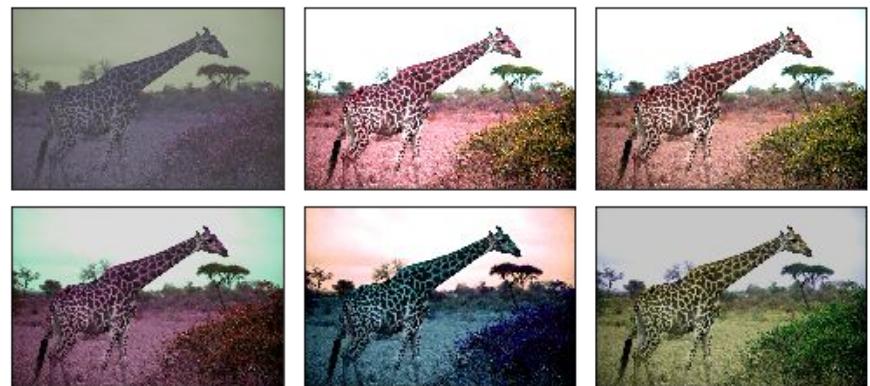
Network Topology	Error [%]
1 hidden (300 U), 1 output (10 U)	4.7
<b>1 hidden (300 U), 1 output (10 U), distorted train set</b>	<b>3.6 (-0.9)</b>
2 hidden (300 + 100 U), 1 output (10 U)	3.05
<b>2 hidden (300 + 100 U), distorted train set</b>	<b>2.45 (-0.6)</b>

# Data Augmentation – Another Example

Geometric transformations



Color jittering

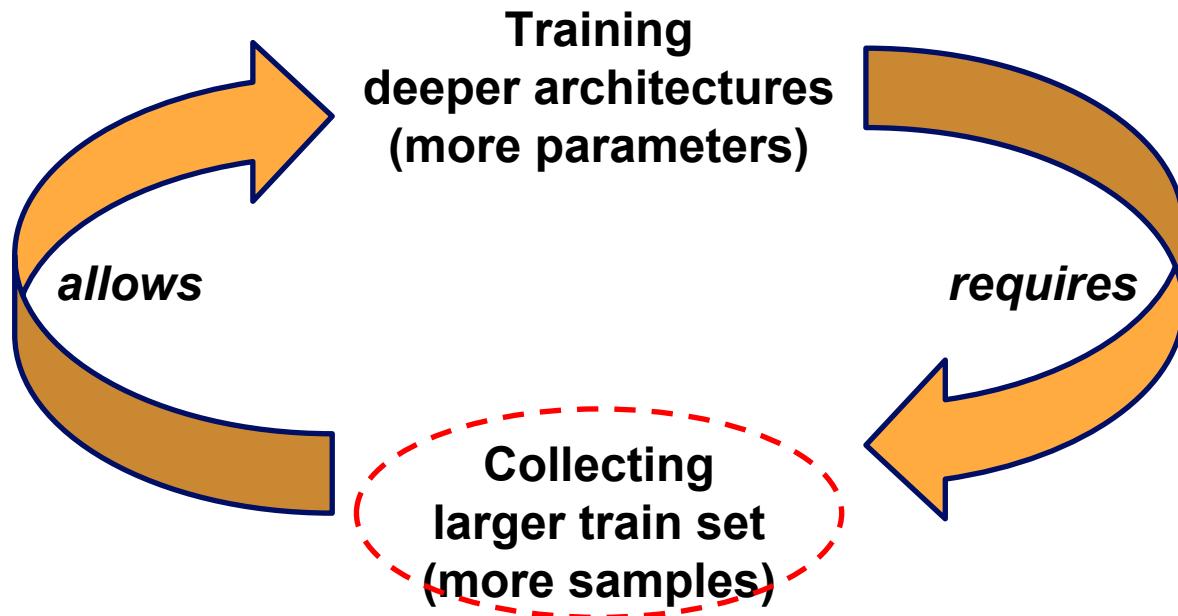


<https://www.codesofinterest.com/2018/02/using-data-augmentations-in-keras.html>

[https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data\\_augmentation.html](https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html)

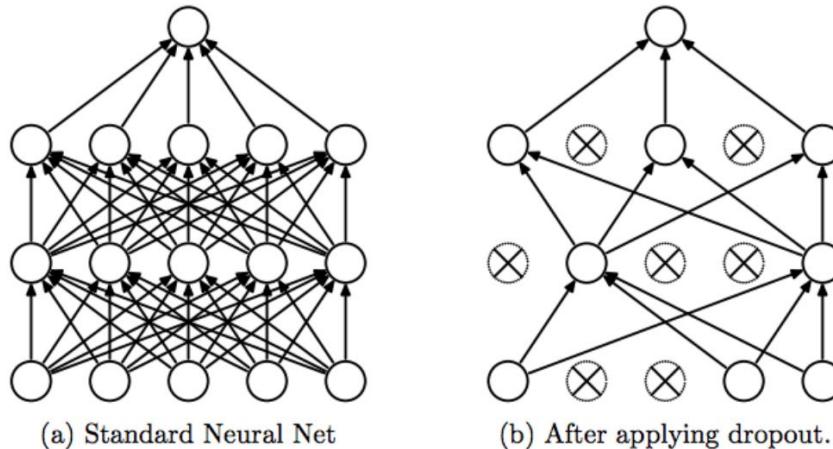
# Dataset Augmentation

- Deeper supervised architectures improve performance, but..



# Dropout

- Training: For each hidden layer, for each sample, for each iteration, ignore each neuron with  $p_{drop} = 0.5$



N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov,  
"Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

# Dropout



N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov,  
"Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

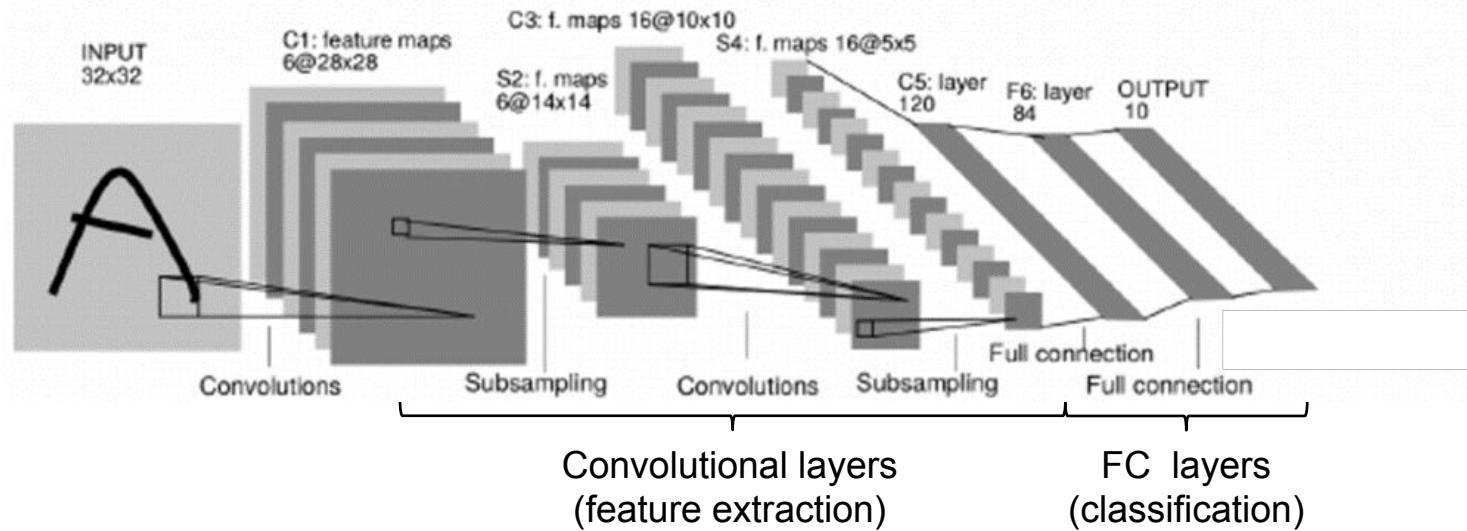
# Avoiding Overfit

- How to avoid overfit ?
  - Early-stop
  - Lower capacity/complexity model (fewer parameters)
    - Still suitable for problem ?
  - Introduce regularization factor  $\lambda$ 
$$J(\mathbf{w}, \mathbf{x}) = E(\mathbf{w}, \mathbf{x}) + \lambda R(\mathbf{w})$$
  - Improve train set
    - Data augmentation
  - Committees of networks
    - *Dropout*

# From Shallow to Deep Architectures

# Recap - LeNet5

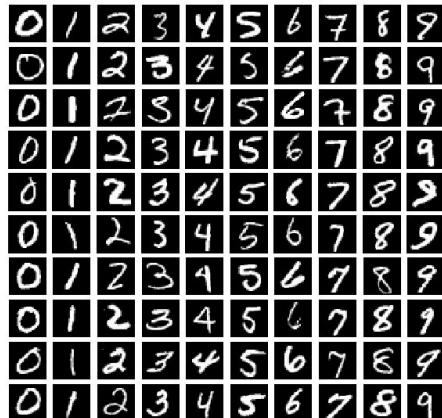
- Repeated *convolve-and-pool* pattern
  - feature extraction
- Multiple FC layers



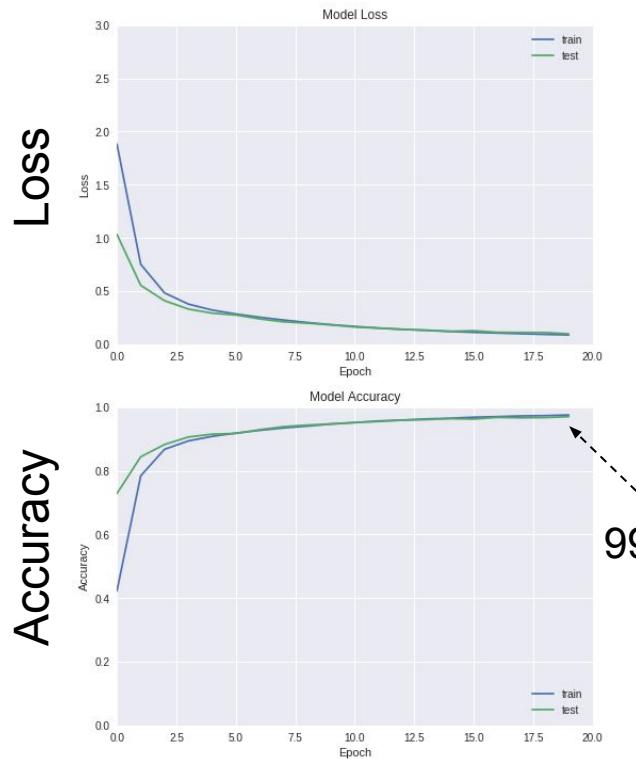
Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition,  
Proceedings of the IEEE, November 1998 (PDF available online)

# Recap - Convolutional Networks

Network	Architecture	Error [%]
LeNet300	1 FC output layer (10 U)	12.0
	1 hidden FC (300 U), 1 out FC (10 U)	4.7
	2 hidden FCs (300 + 100 U), 1 out FC (10 U)	3.05
LeNet5	2 Conv (3 F), 1 out FC (LeNet1)	1.7
	2 conv (6+16 F), 3 FC layer	0.95



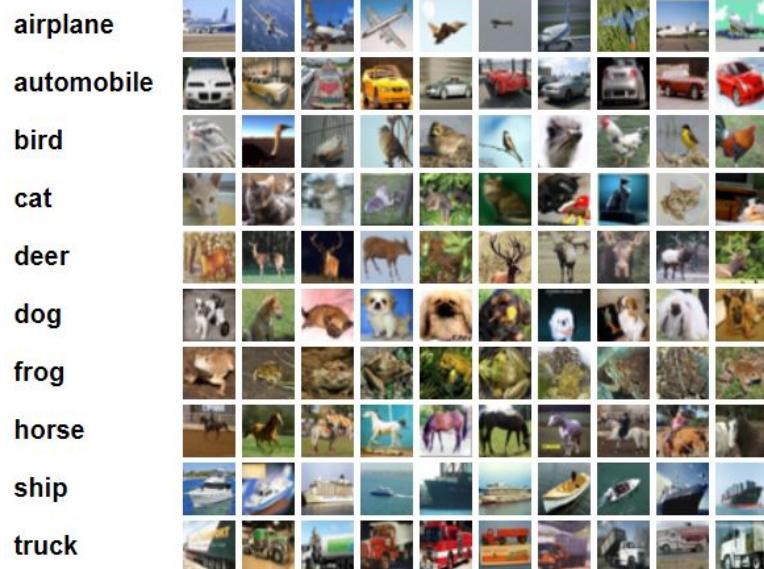
# LeNet5: MNIST vs CIFAR10



MNIST

# The CIFAR10 Datasets

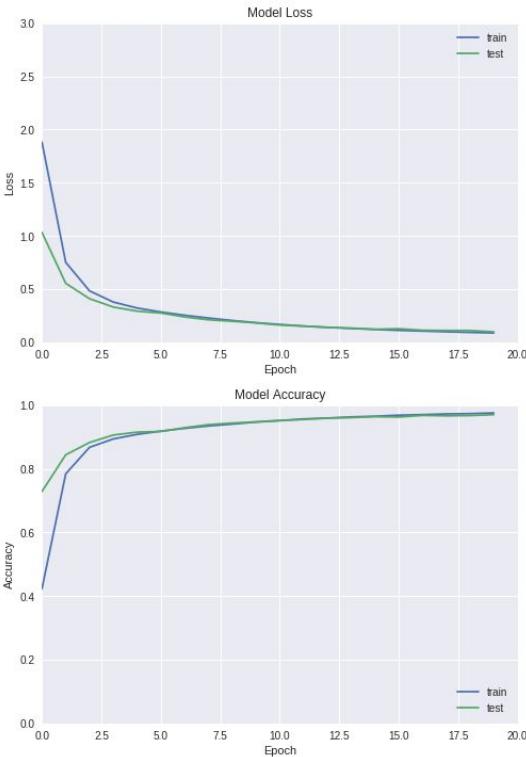
- *CIFAR10 dataset (2009)*
  - 50k train images. 10k test images. 10 classes, 32x32



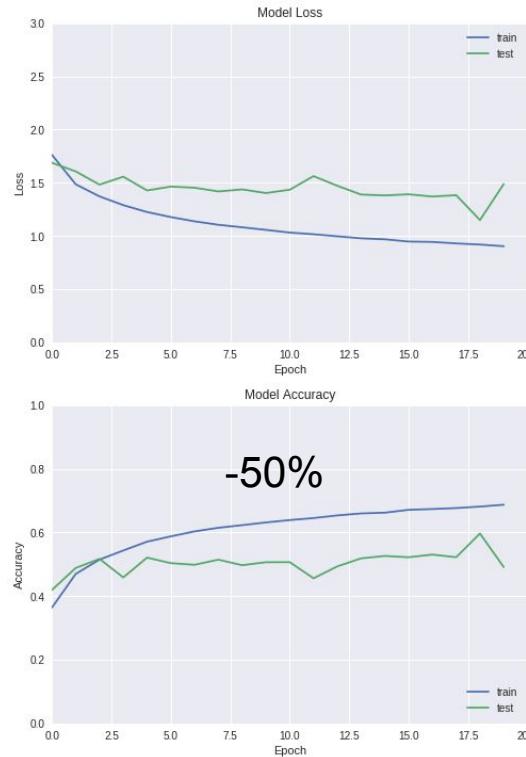
<https://www.cs.toronto.edu/~kriz/cifar.html>

# LeNet5: MNIST vs CIFAR10

Loss



MNIST

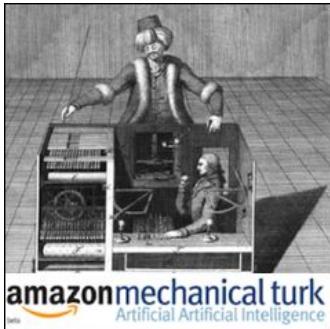


CIFAR-10

-50%

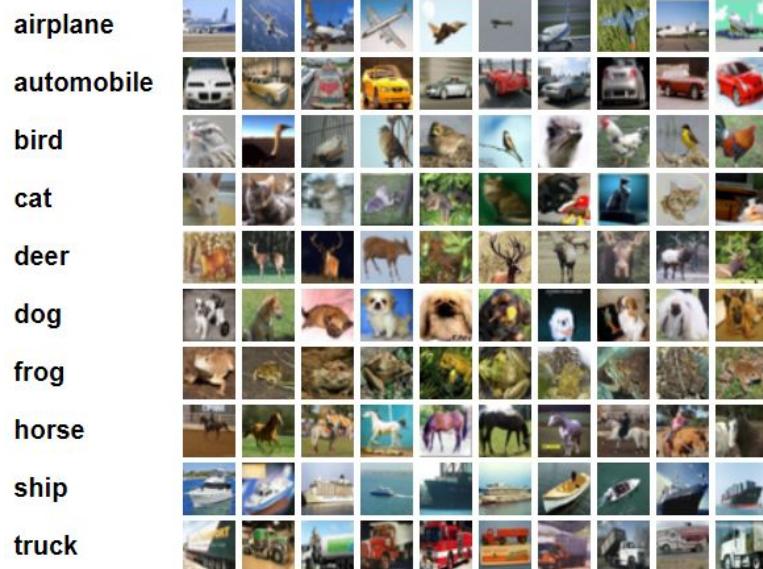
# From shallow to deep networks

- More parameteres (layers) -> more complex tasks
  - More annotated training samples to avoid *overfitting*



# The CIFAR10 Datasets

- *CIFAR10 dataset (2009)*
  - 50k train images. 10k test images. 10 classes, 32x32



<https://www.cs.toronto.edu/~kriz/cifar.html>

# «Early» Datasets for Image Classification

- CalTech101 dataset (2003)
  - ~10k images, 101 classes, variable size



Deng, Jia; Dong, Wei; Socher, Richard; Li, Li-Jia; Li, Kai; Fei-Fei, Li (2009), "ImageNet: A Large-Scale Hierarchical Image Database" (PDF), 2009 conference on Computer Vision and Pattern Recognition

# «Early» Datasets for Image Classification

- CalTech256 dataset (2007)
  - ~30k images, 256 classes, variable size



Fei-Fei, Li, Rob Fergus, and Pietro Perona. "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories." Computer vision and Image understanding 106, no. 1 (2007): 59-70.

# ImageNet Large Scale Visual Recognition Contest (ILSVRC)

- Originally presented in 2009 (3 M images, 5k classes)
- Since 2010 dataset for ILSVRC (1M images, 1k classes)



Deng, Jia; Dong, Wei; Socher, Richard; Li, Li-Jia; Li, Kai; Fei-Fei, Li (2009), "ImageNet: A Large-Scale Hierarchical Image Database" (PDF), 2009 conference on Computer Vision and Pattern Recognition

# From shallow to deep networks

- More parameters (layers) -> more complex tasks
  - More annotated training samples to avoid *overfitting*
  - Increased (training) complexity



# From shallow to deep networks

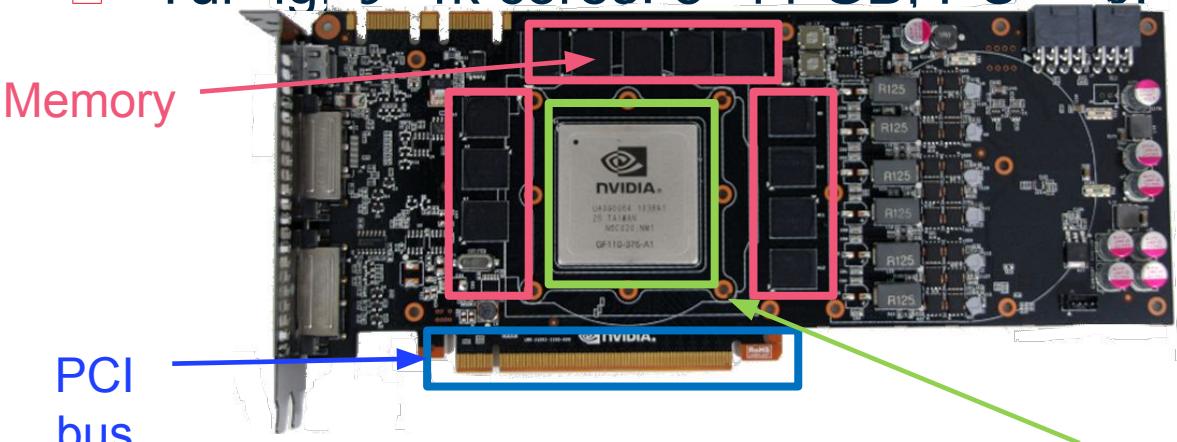
- CRAY Trinity (End 2010's)
  - X64 architecture, ~1000 CPUs, 174M USD



# GPUs as Massively Parallel Computers



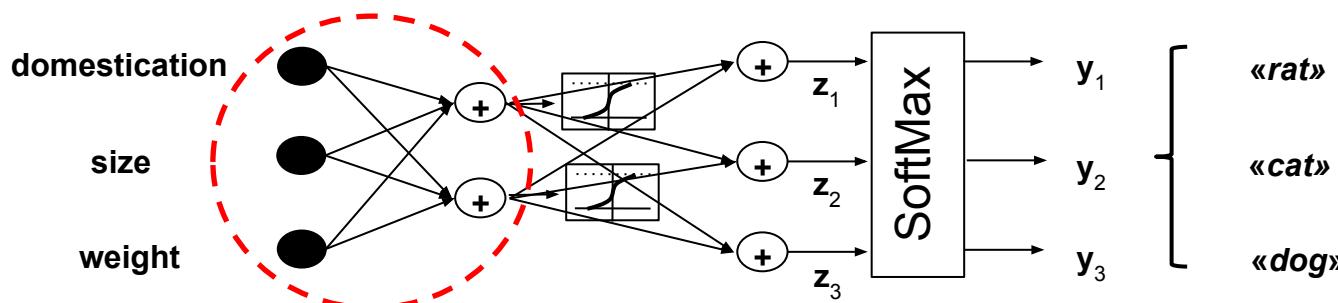
- GTX 580 (2010)
  - Fermi, 512 cores, ~2 GB RAM, PCI 2.0, 500 USD
- RTX 2080 (2018)
  - Turing, 3~4k cores, 8~11 GB, PCI 3.0, 800~1200 USD



Massive  
multicore  
processor

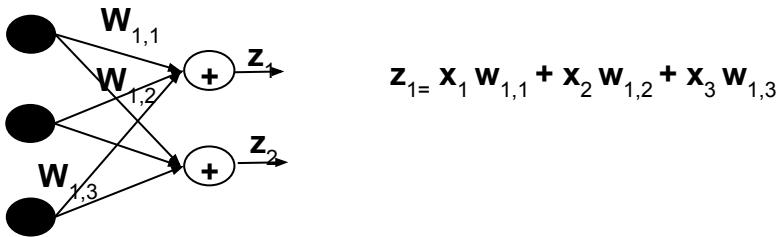
# Matrix-Vector Multiplications in FCNs

- Simple rat/cat/dog classifier
  - Focus on hidden layer



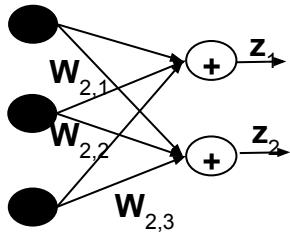
# Matrix-Vector Multiplications in FCNs

- Simple rat/cat/dog classifier
  - Focus on hidden layer



# Matrix-Vector Multiplications in FCNs

- Simple rat/cat/dog classifier
- Focus on hidden layer

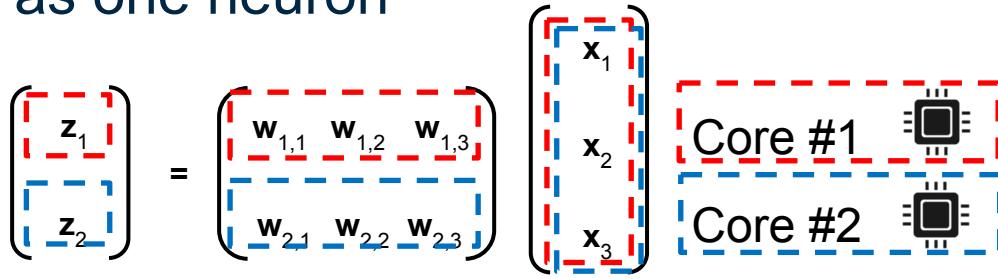
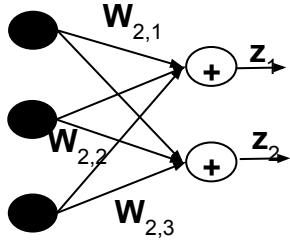


$$z_1 = x_1 w_{1,1} + x_2 w_{1,2} + x_3 w_{1,3}$$

$$z_2 = x_1 w_{2,1} + x_2 w_{2,2} + x_3 w_{2,3}$$

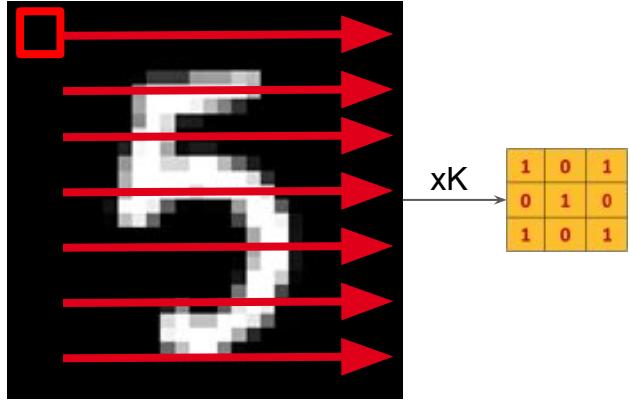
# Matrix-Vector Multiplications in FCNs

- Simple rat/cat/dog classifier
  - Focus on hidden layer
  - Each core performs as one neuron



The same multiplications ( $xW$ ) are applied to every input in the mini-batch. GPUs are efficient to perform many times the same matrix multiplications.

# Convolution on GPUs

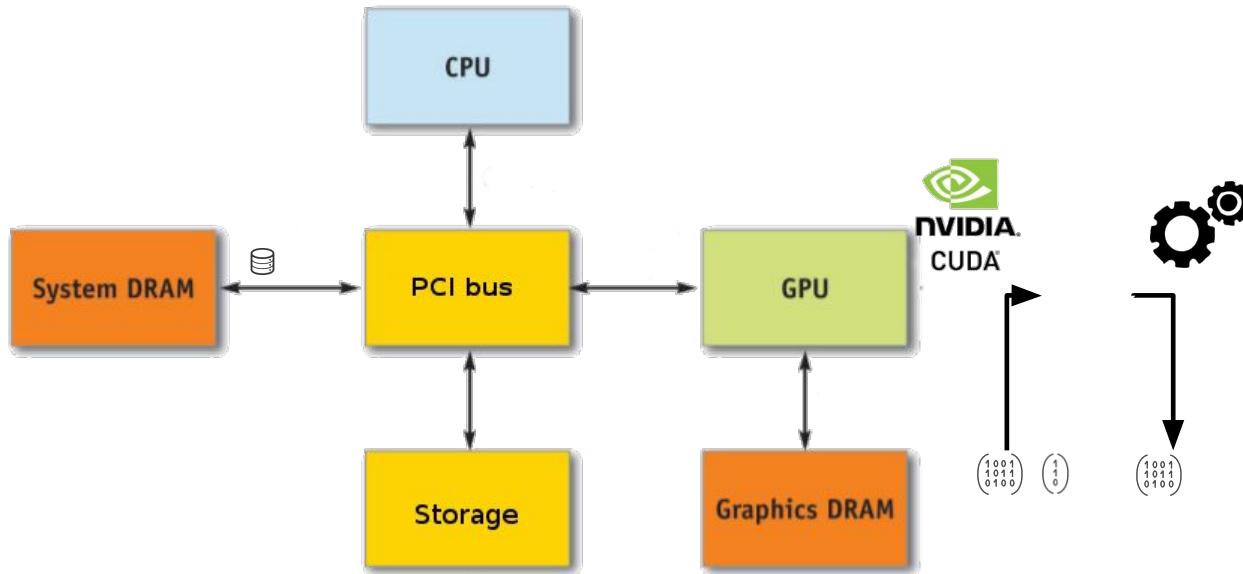


- Multiple threads are used to calculate the convolution operator of multiple pixels simultaneously.
- Each thread having access to the coefficients of the convolution kernel calculates the double sum of the convolution operator.
- The kernel coefficients being constant during the whole execution can be stored into the shared memory

Example: a filter of the first layer

- input:  $28 \times 28 = 784$
- If the GPUs has more than 784 cores, it can compute all the filter outputs in parallel.

# GP-GPUs Typical Workflow



# Check your GPUs

```
sh-4.2$ nvidia-smi
Wed Dec 11 09:52:10 2019
+-----+
| NVIDIA-SMI 418.87.01      Driver Version: 418.87.01      CUDA Version: 10.1      |
+-----+
| GPU  Name      Persistence-M | Bus-Id      Disp.A  | Volatile Uncorr. ECC  |
| Fan  Temp     Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M.  |
|=====+=====+=====+=====+=====+=====+=====+=====|
| 0  Tesla K80        On          00000000.00.1E.0 Off |                0 |
| N/A  44C     P0    60W / 149W | 11292MiB / 11441MiB | 13%       Default |
+-----+
+-----+
| Processes:
| GPU  PID  Type  Process name                  GPU Memory  |
|               Usage
|=====+=====+=====+=====+=====+=====+
| 0    9389   C    ...naconda3/envs/tensorflow_p36/bin/python 11012MiB  |
| 0    21268   C    ...naconda3/envs/tensorflow_p36/bin/python  267MiB  |
+-----+
```

# Check your GPUs

```
sh-4.2$ nvidia-smi
Wed Dec 11 09:52:10 2019
+-----+
| NVIDIA-SMI 418.87.01      Driver Version: 418.87.01      CUDA Version: 10.1 |
+-----+
| GPU  Name      Persistence-M | Bus-Id      Disp.A  | Volatile Uncorr. ECC |
| Fan  Temp     Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|=====+=====+=====+=====+=====+=====+=====+=====|
| 0  Tesla K80        On          00000000.00.1E.0 Off |                0 |
| N/A  44C     P0    60W / 149W | 11292MiB / 11441MiB | 13%       Default |
+-----+
                                          Memory full  GPU underused
+-----+
| Processes: List of processes                               GPU Memory |
| GPU  PID  Type  Process name                           Usage      |
|=====+=====+=====+=====+=====+=====|
| 0    9389   C    ...naconda3/envs/tensorflow_p36/bin/python 11012MiB |
| 0    21268   C    ...naconda3/envs/tensorflow_p36/bin/python  267MiB |
+-----+
```

# Check your GPUs: exemple

```
xiongyu@ubuntu:~$ watch nvidia-smi
|    0      47741      C ./mtcnn_c                               461MiB |
Every 2.0s: nvidia-smi

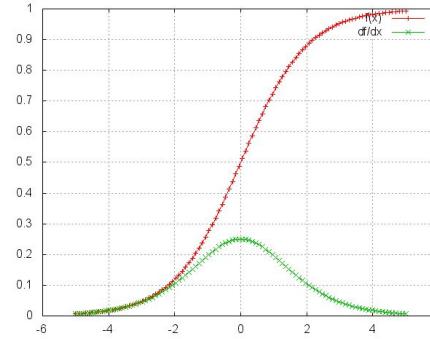
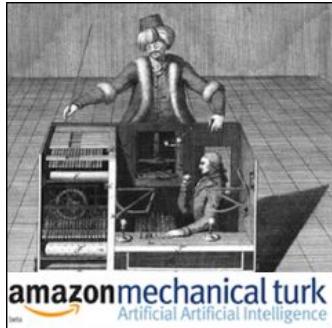
Fri Jun 29 11:25:43 2018
+-----+
| NVIDIA-SMI 390.48                 Driver Version: 390.48 |
+-----+
| GPU  Name      Persistence-M | Bus-Id      Disp.A  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+=====+=====|
|    0  Tesla M40        On   | 00000000:03:00.0 Off |          0 |
|  0%   57C   P0  217W / 250W | 10806MiB / 11448MiB |    99%     Default |
+-----+
|    1  Tesla M40        On   | 00000000:04:00.0 Off |          0 |
|  0%   58C   P0  200W / 250W | 9992MiB / 11448MiB |    99%     Default |
+-----+
|    2  Tesla M40        On   | 00000000:84:00.0 Off |          0 |
|  0%   56C   P0  222W / 250W | 9955MiB / 11448MiB |    99%     Default |
+-----+
|    3  Tesla M40        On   | 00000000:85:00.0 Off |          0 |
|  0%   57C   P0  203W / 250W | 9960MiB / 11448MiB |    98%     Default |
+-----+
+-----+
| Processes:                               GPU Memory |
| GPU     PID  Type  Process name           Usage   |
|=====+=====+=====+=====+=====+=====|
|    0     35346  C   python2                110MiB |
|    0     55323  C   python                10673MiB |
|    1     55323  C   python                9970MiB |
```

# Check your GPUs: exemple

NVIDIA-SMI 418.67			Driver Version: 418.67		CUDA Version: 10.1		
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
					Memory-Usage	GPU-Util	Compute M.
0	Tesla V100-SXM2...	off	00000000:15:00.0	Off		0	
N/A	39C	P0	55W / 300W		0MiB / 32480MiB	0%	Default
1	Tesla V100-SXM2...	off	00000000:16:00.0	Off		0	
N/A	54C	P0	234W / 300W		31116MiB / 32480MiB	100%	Default
2	Tesla V100-SXM2...	off	00000000:3A:00.0	Off		0	
N/A	42C	P0	140W / 300W		31048MiB / 32480MiB	45%	Default
3	Tesla V100-SXM2...	off	00000000:3B:00.0	Off		0	
N/A	42C	P0	56W / 300W		0MiB / 32480MiB	0%	Default
4	Tesla V100-SXM2...	off	00000000:89:00.0	Off		0	
N/A	36C	P0	56W / 300W		0MiB / 32480MiB	0%	Default
5	Tesla V100-SXM2...	off	00000000:8A:00.0	Off		0	
N/A	38C	P0	56W / 300W		0MiB / 32480MiB	0%	Default
6	Tesla V100-SXM2...	off	00000000:B2:00.0	Off		0	
N/A	38C	P0	55W / 300W		0MiB / 32480MiB	0%	Default
7	Tesla V100-SXM2...	off	00000000:B3:00.0	Off		0	
N/A	41C	P0	56W / 300W		0MiB / 32480MiB	0%	Default
Processes:							
GPU	PID	Type	Process name	GPU Memory Usage			
1	22063	C	python	31105MiB			
2	38649	C	python3	31037MiB			

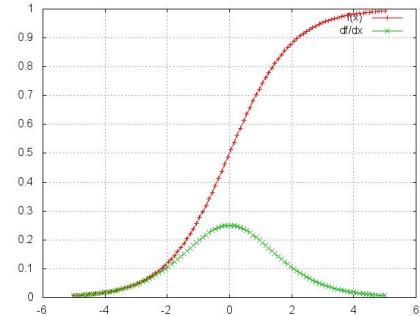
# From shallow to deep networks

- More parameteres (layers) -> more complex tasks
  - More annotated training samples to avoid *overfitting*
  - Increased (training) complexity
  - *Vanishing gradient* (with sigmoid activations)



# Vanishing Gradient Problem

$$g'(z) = \frac{e^{-z}}{(1 + e^{-z})^2}$$

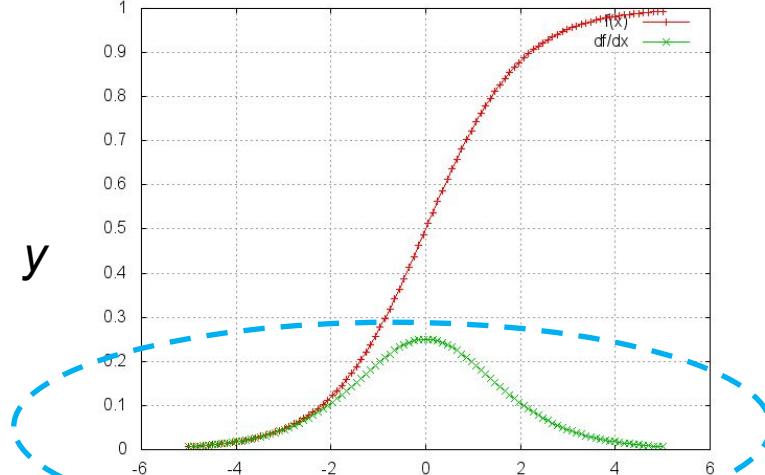
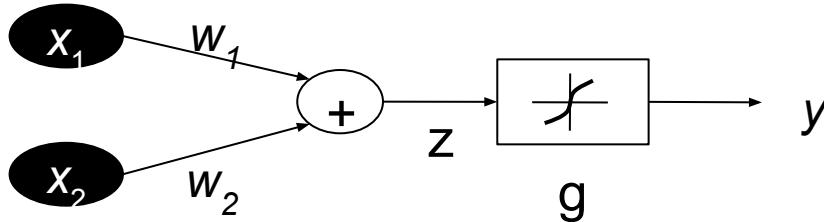


# The Sigmoid Activation

- Compute gradient of  $E$  w.r.t. to each  $w_n$  via *chain rule*, e.g.:

$$\frac{dE}{dw_1} = \frac{dE}{dy} \frac{dy}{dz} \frac{dz}{dw_1}$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

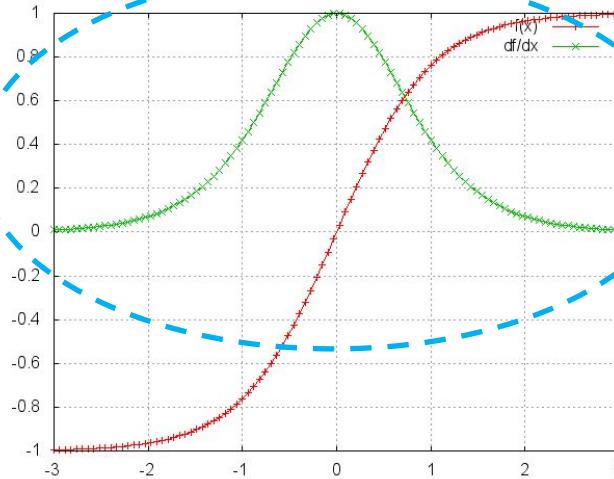
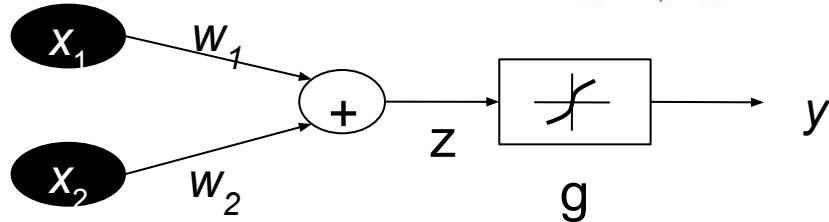


# The Hyperbolic Tangent Activation

- Compute gradient of  $E$  w.r.t. to each  $w_n$  via *chain rule*, e.g.:

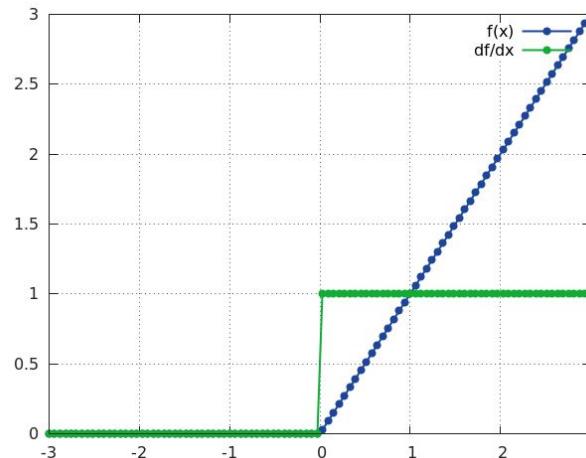
$$\frac{dE}{dw_1} = \frac{dE}{dy} \frac{dy}{dz} \frac{dz}{dw_1}$$

$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



# The Rectified Liner Unit - ReLU

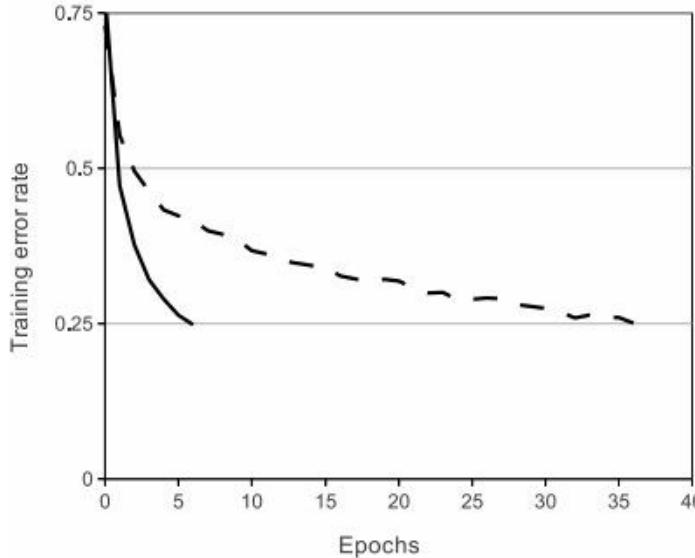
- Rectified Linear Units (*halfwave rectifier*)  $y = \max(0, x)$ 
  - Mitigates gradient vanishing problem
  - Easy to compute
  - Sparse activations
  - Biological plausibility  
(one-sided)



X. Glorot, A. Bordes, Y. Bengio. "Deep Sparse Rectifier Neural Networks." In Aistats, vol. 15, no. 106, p. 275. 2011.

# The Rectified Liner Unit - ReLU

- CIFAR-10 training error a 4-layer ConvNet
- 6~7 times faster convergence

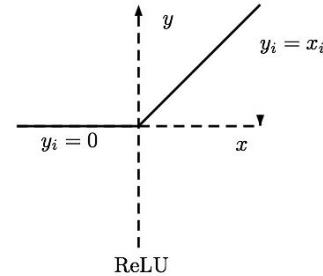


A. Krizhevsky, I. Sutskever, G. E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

# The Leaky/Parametric ReLU

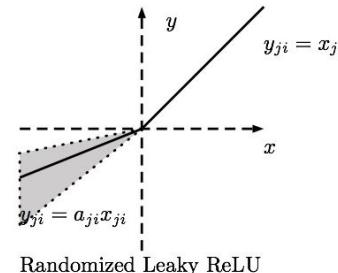
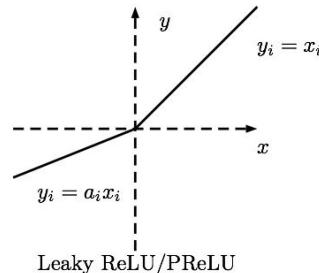
- ReLU may yield *dead neurons*
  - Gradient propagation problem
- Leaky ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$



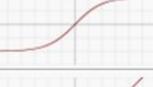
- Parametric ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$



X. Glorot, A. Bordes, Y. Bengio. "Deep Sparse Rectifier Neural Networks." In Aistats, vol. 15, no. 106, p. 275. 2011.

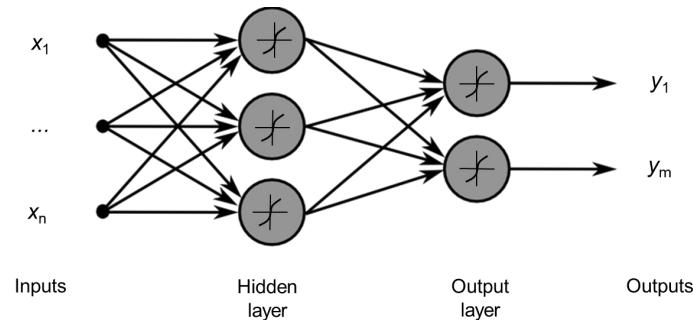
# Activation Functions Summary

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Logistic (a. k. a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

S.Sharma "Activation Functions Explained" <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

# Which Activation Function ?

- Output layer
  - Classification
    - Sigmoid (binary) or softmax (multiclass)
  - Bounded zero-mean regression
    - Hyperbolic tangent
  - Unbounded regression
    - Linear
- Hidden layer
  - ReLU-like



# The Rectified Liner Unit - ReLU

Biol. Cybernetics 36, 193–202 (1980)



Biological  
Cybernetics

© by Springer-Verlag 1980

## Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position

Kunihiko Fukushima

NHK Broadcasting Science Research Laboratories, Kinuta, Setagaya, Tokyo, Japan

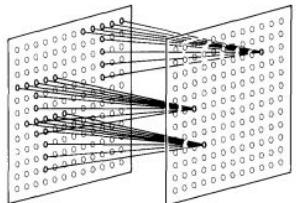


Fig. 3. Illustration showing the input interconnections to the cells within a single cell-plane

All the neural cells employed in this network is of analog type. That is, the inputs and the output of a cell take non-negative analog values proportional to the pulse density (or instantaneous mean frequency) of the firing of the actual biological neurons.

S-cells have shunting-type inhibitory inputs similarly to the cells employed in the conventional cognitron (Fukushima, 1975). The output of an S-cell in the  $k_l$ -th S-plane in the  $l$ -th module is described below.

$$u_{SI}(k_l, \mathbf{n}) = r_l \cdot \varphi \left[ \frac{1 + \sum_{k_{l-1}=1}^{K_{l-1}} \sum_{v \in S_l} a_l(k_{l-1}, v, k_l) \cdot u_{CI-1}(k_{l-1}, \mathbf{n} + \mathbf{v})}{1 + \frac{2r_l}{1+r_l} \cdot b_l(k_l) \cdot v_{CI-1}(\mathbf{n})} - 1 \right],$$

where

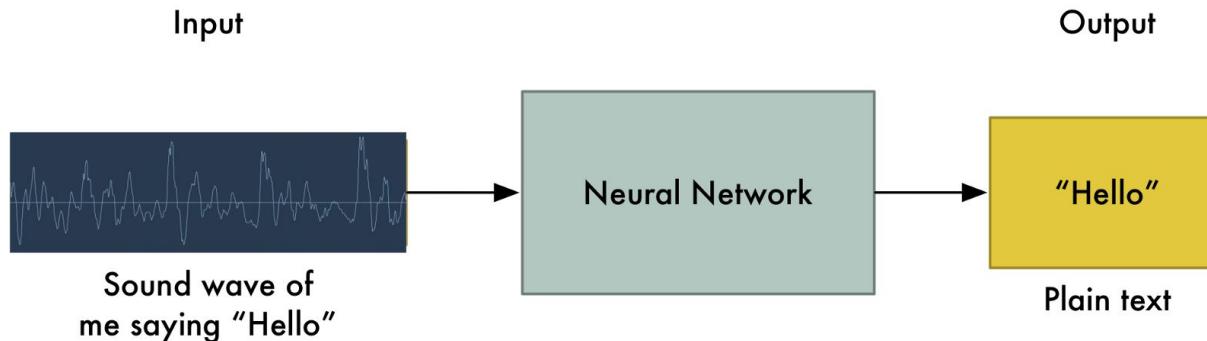
$$\varphi[x] = \begin{cases} x & x \geq 0 \\ 0 & x < 0. \end{cases} \quad (2)$$

In case of  $l=1$  in (1),  $u_{CI-1}(k_{l-1}, \mathbf{n})$  stands for  $u_0(\mathbf{n})$ , and we have  $K_{l-1} = 1$ .

Fukushima, Kunihiko, Sei Miyake, and Takayuki Ito. "Neocognitron: A neural network model for a mechanism of visual pattern recognition." IEEE transactions on systems, man, and cybernetics 5 (1983): 826-834.

# Deep Neural Networks for Speech Recognition

- First successes training *deep* NNs in speech recognition
  - More layers equal better performance
  - “Audio far simpler signals than image/video”

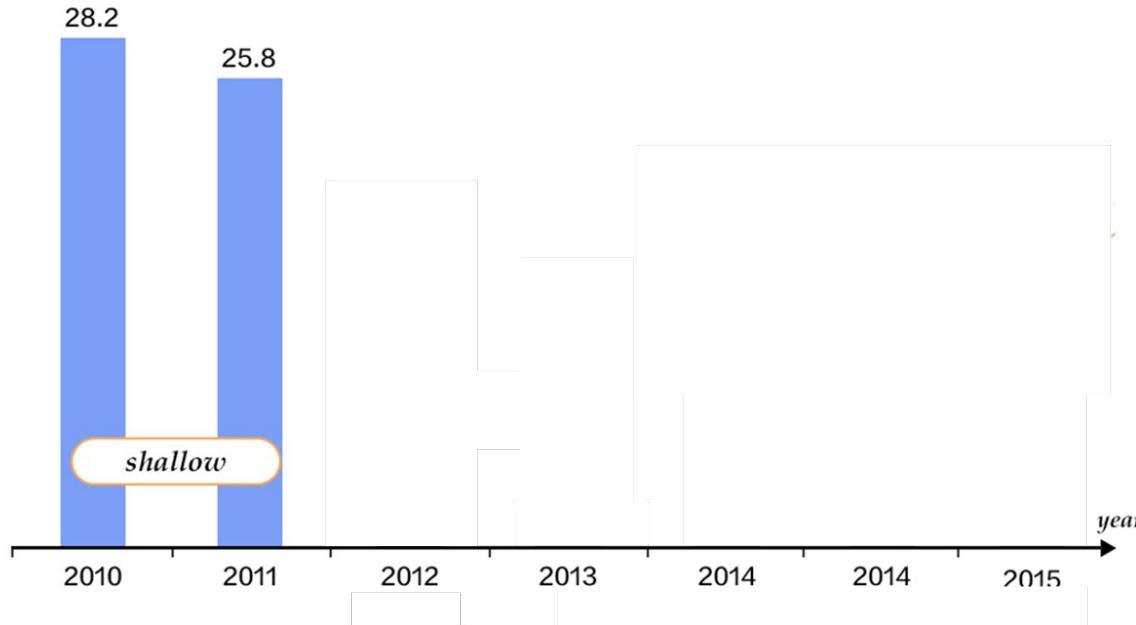


# Part 4

# Deep Neural Networks for vision

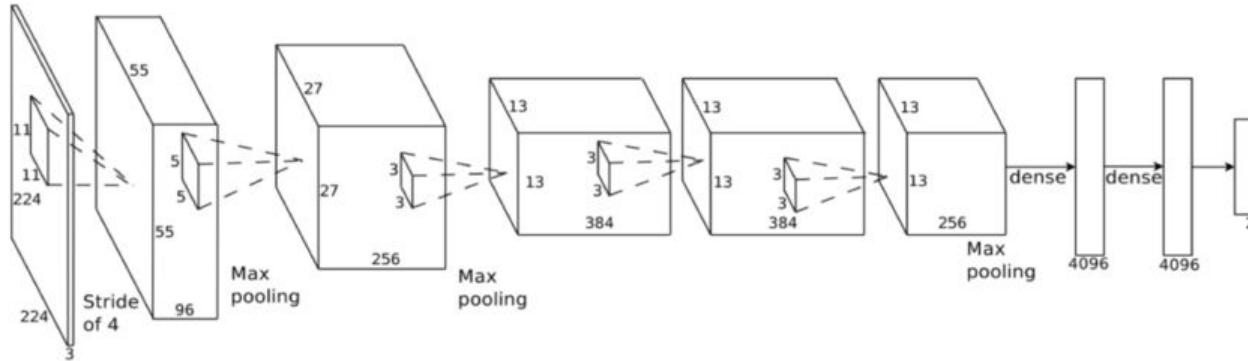
# ImageNet Challenge Before the Deep Era

- 2010: SIFT descriptors + SVN (NEC)
- 2011: SIFT descriptors, Fisher Vectors, SVM (XRCE)



# The AlexNet Architecture (2012)

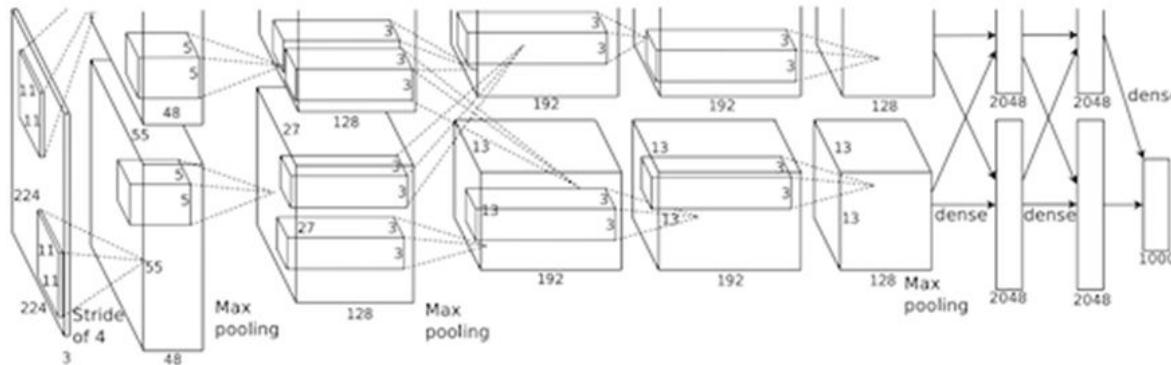
- First «deep» convolutional network
  - 5 convolutional layers, 3 fully connected layers
  - 62.3 Mprms (conv layers 6% but take 95% of time)



A. Krizhevsky, I. Sutskever, G. E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

# The AlexNet Architecture (2012)

- Trained over two GTX580 GPUs (2GB memory each)
  - Split convolutions to different GPUs
  - Distribute the fully connected layers to different GPUs
  - Trained on 2 x GTX 580 for 5~6 days (90 epochs)



A. Krizhevsky, I. Sutskever, G. E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

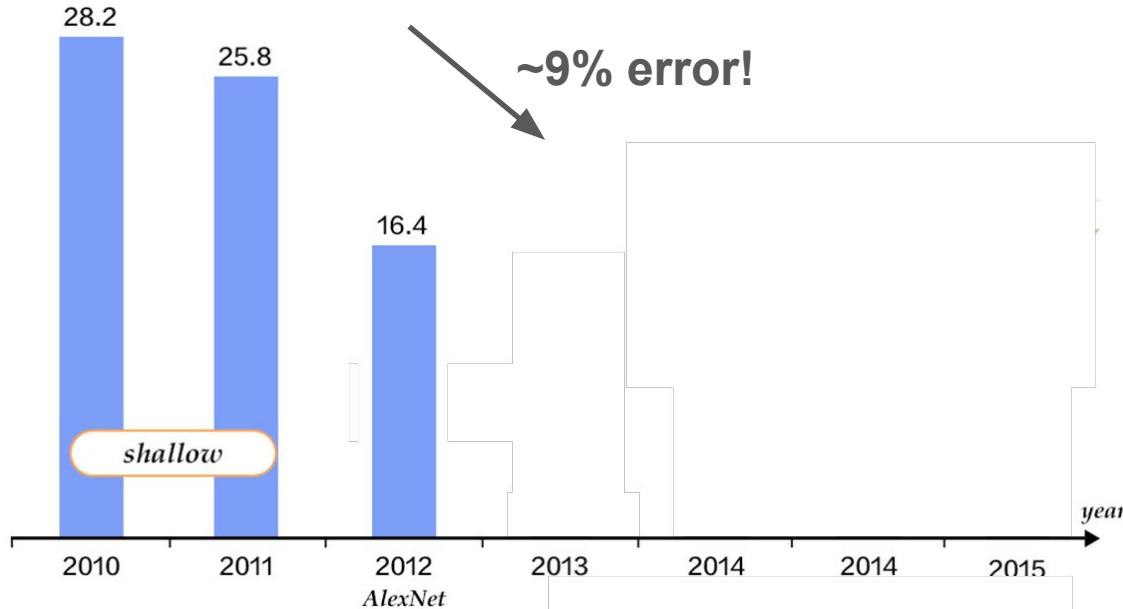
# The AlexNet Architecture (2012)

- (Main) differences w.r.t. *LeNet5*
  - Deeper than LeNet5 (5 Conv w.r.t. 3)
  - ReLU activations in place of sigmoids
  - Dropout before FC layers (+ L2 regularization)
  - Batch size 128 images
  - Data augmentation
  - LR divide by 10 when valid error settles
- However
  - **No batch normalization**

A. Krizhevsky, I. Sutskever, G. E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

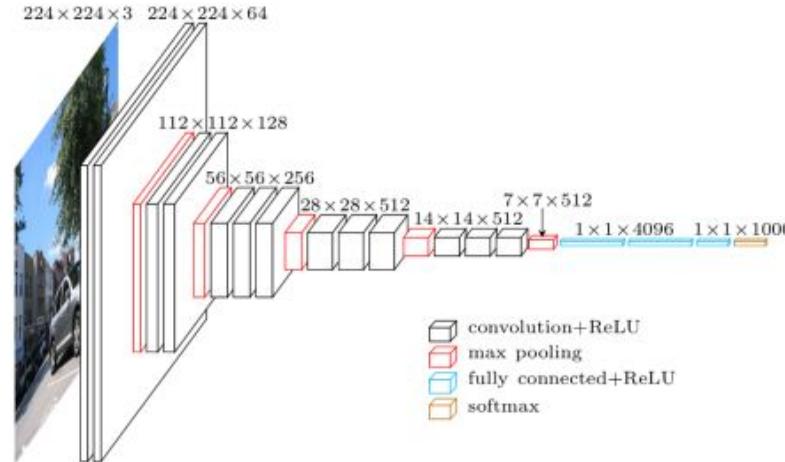
# The AlexNet Architecture (2012)

- 2012 ILSVRC winner with top-5 error rate 16.4% (vs. 26.2%)
- **Problem:** very large 11x11 filters in first conv layer



# VGGNet (2014)

- Up to 19 convolutional layers, 3 fully connected layers
- **Key idea:** 3x3 filters everywhere



K. Simonyan, A. Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

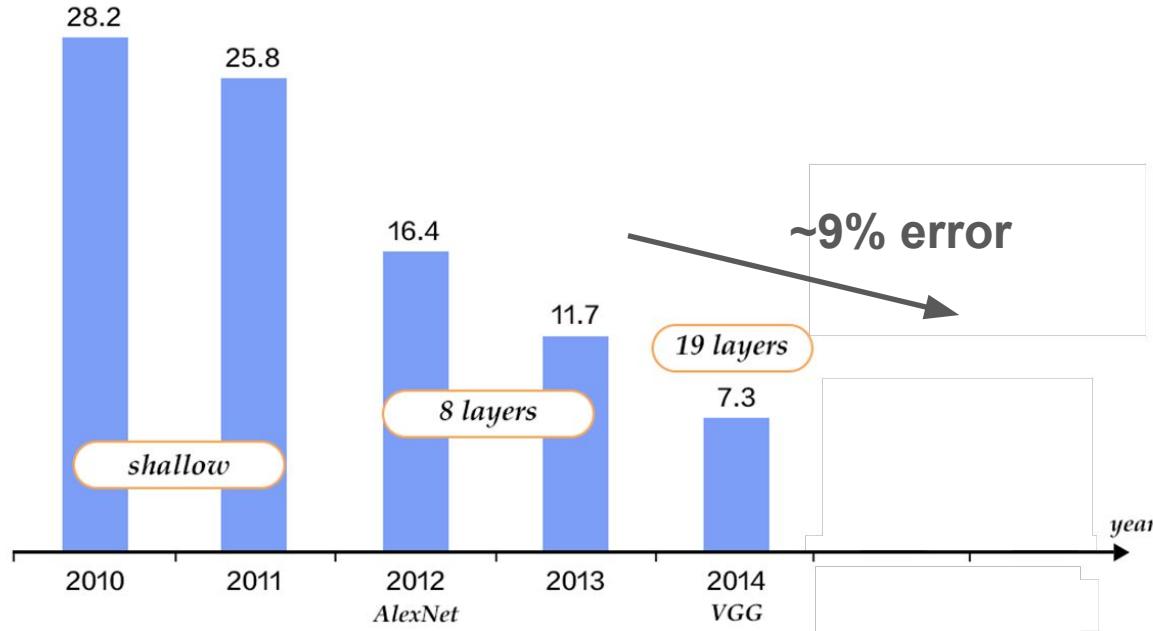
# VGGNet (2014)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

K. Simonyan, A. Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

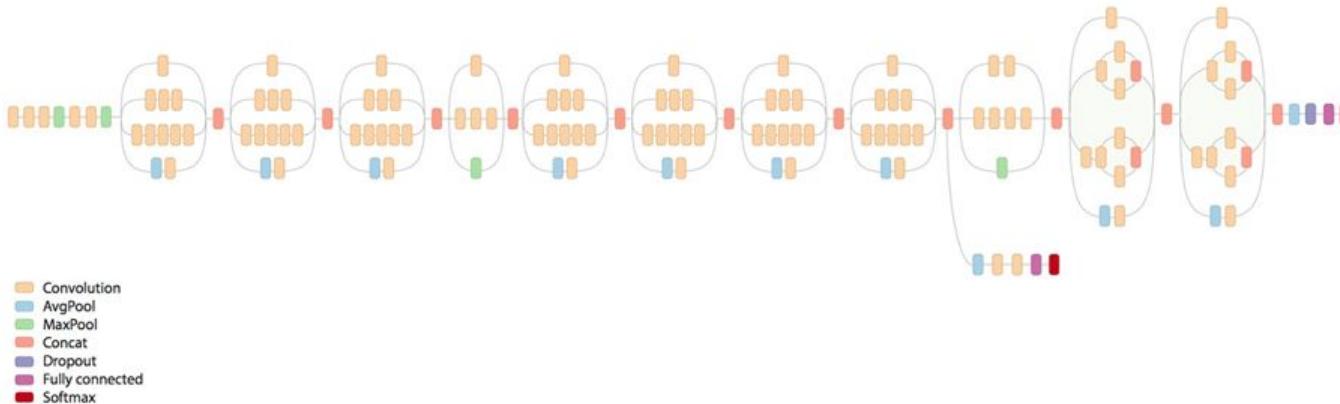
# VGGNet (2014)

- 2014 ILSVRC top-5 runner with error rate 7,3%



# GoogLeNet (2015)

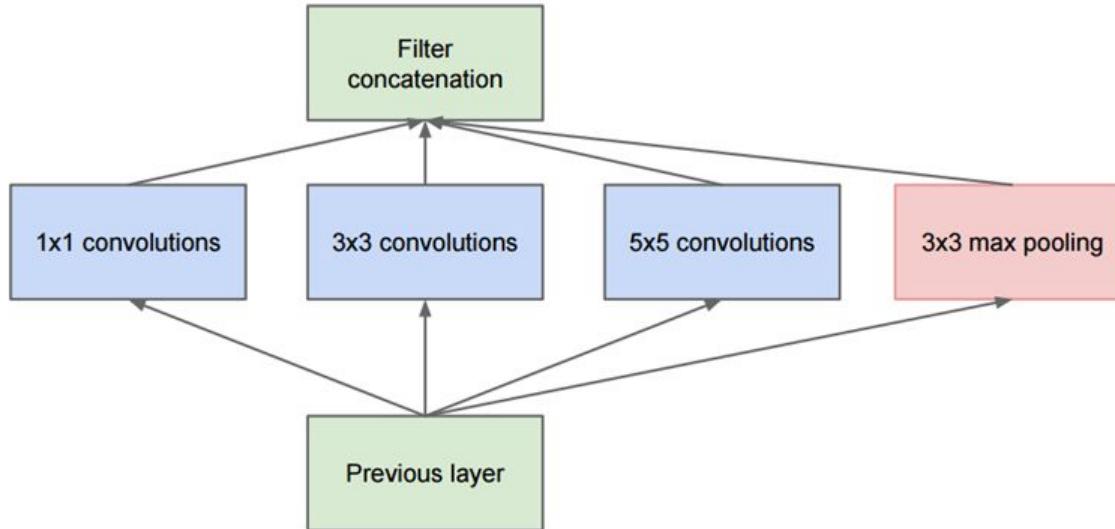
- Big IT firm (Google) wins ILSVRC
- Non-strictly sequential data processing (*Inception module*)



Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.

# GoogLeNet (2015)

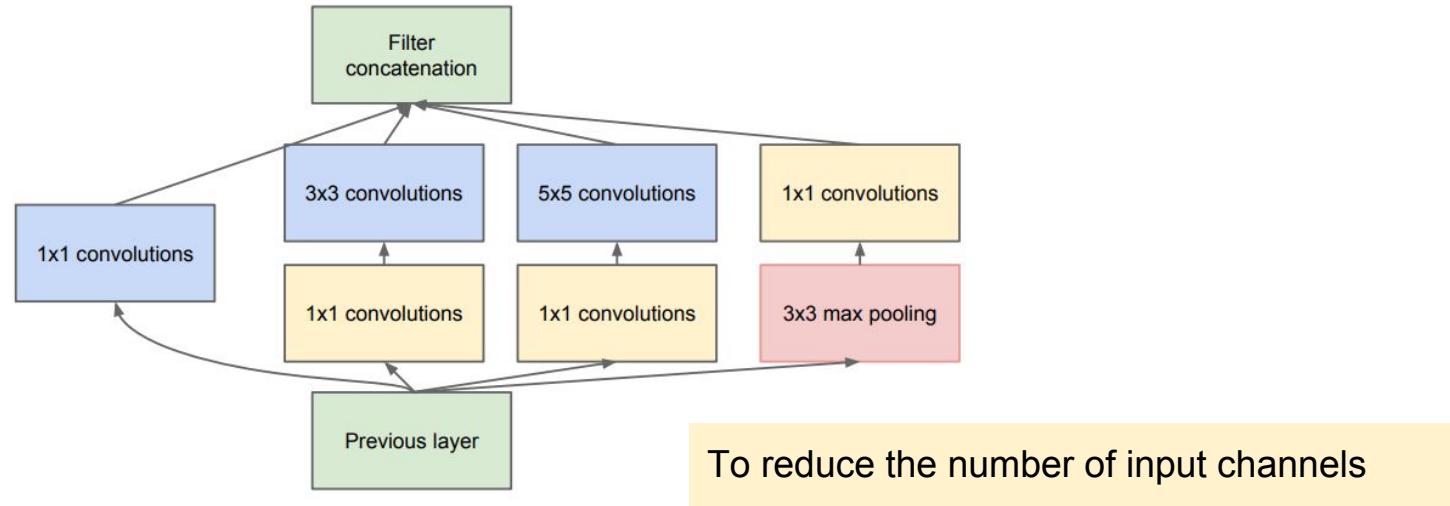
- Inception module simplified view
  - Key idea: do convolutions and pooling in parallel



Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.

# GoogLeNet (2015)

- Inception module simplified view
  - Key idea: do convolutions and pooling in parallel

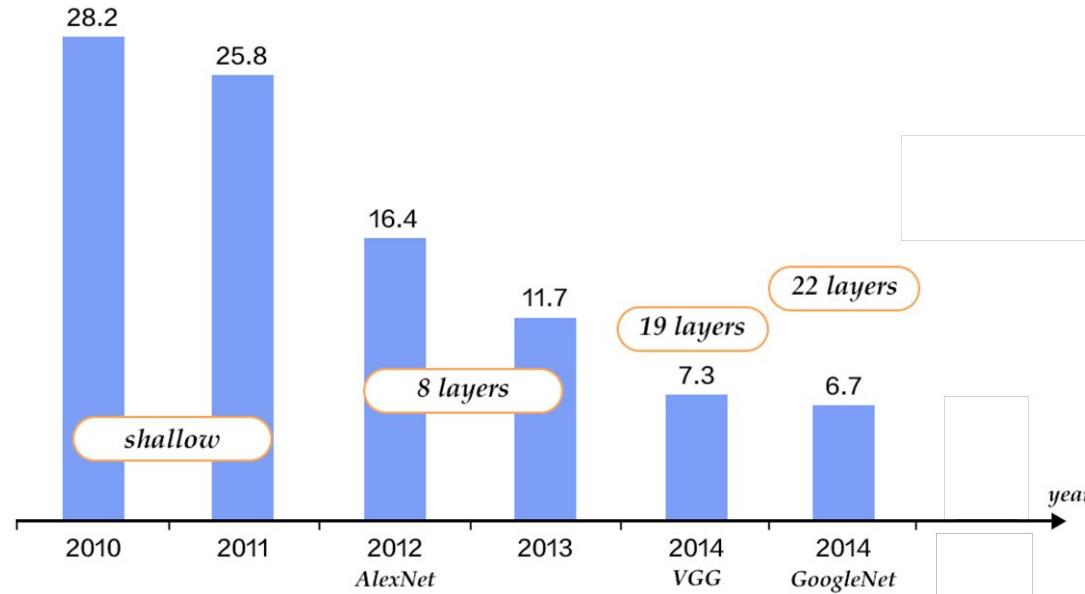


(b) Inception module with dimension reductions

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.

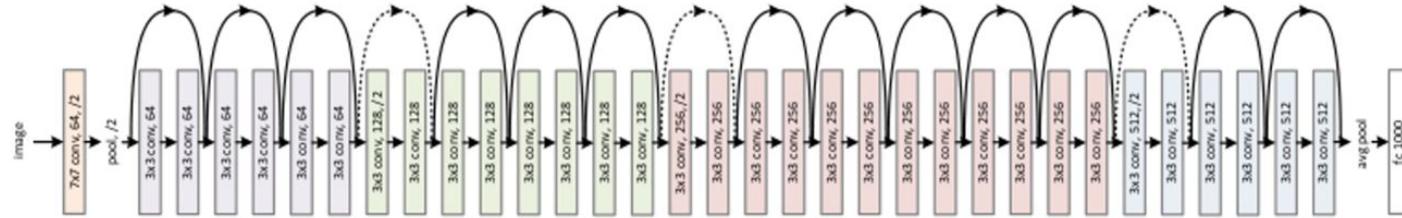
# GoogLeNet (2015)

- 2014 ILSVRC winner with top-5 error rate 6.7%



# ResNet (2015-present)

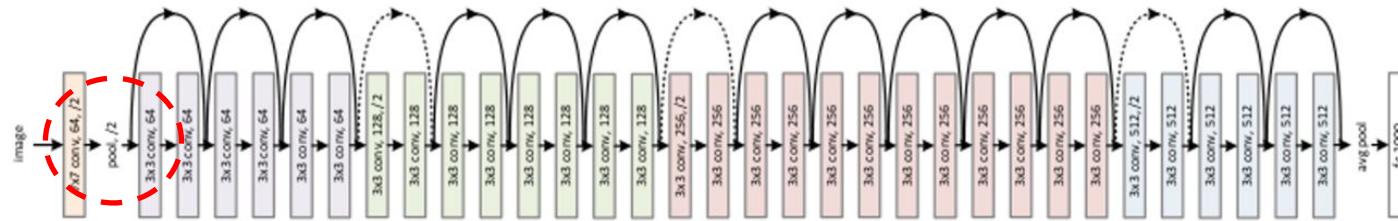
- 2015 ILSVRC winner with top-5 error rate 6.7%
- 18, **34**, 50, 101, 151 layers



He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

# ResNet (2015-present)

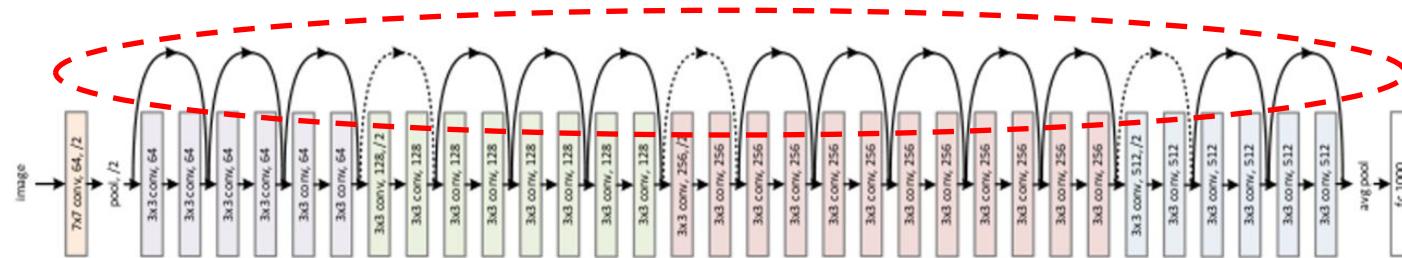
- 2015 ILSVRC winner with top-5 error rate 6.7%
- 18, **34**, 50, 101, 151 layers
- (Almost) *pool-less* (2px convolution stride)



He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

# ResNet (2015-present)

- 2015 ILSVRC winner with top-5 error rate 6.7%
- 18, **34**, 50, 101, 151 layers
- (Almost) pool-less (2px convolution stride)

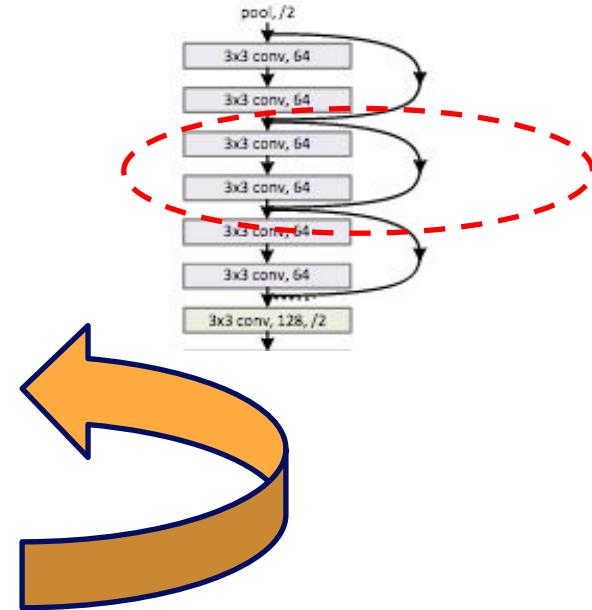
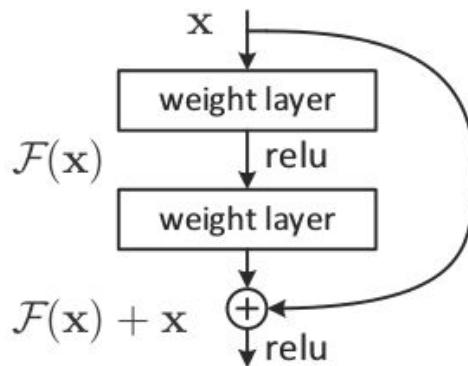


- Relies on skip connections

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

# ResNet (2015)

- Relies on skip/shortcut connections
  - Gradient backprop easier



Understanding and Implementing Architectures of ResNet

<https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624>

# ResNet (2015)

- Relies on skip/shortcut connections
  - Gradient backprop easier

Understanding and Implementing Architectures of ResNet

<https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624>

# ResNet (2015)

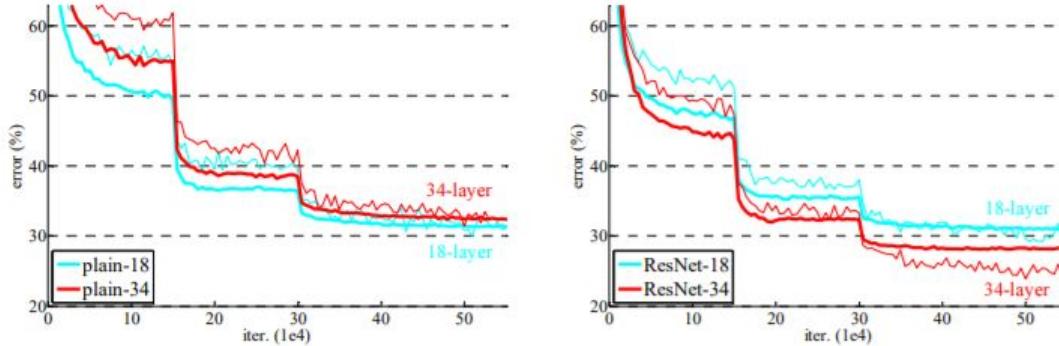
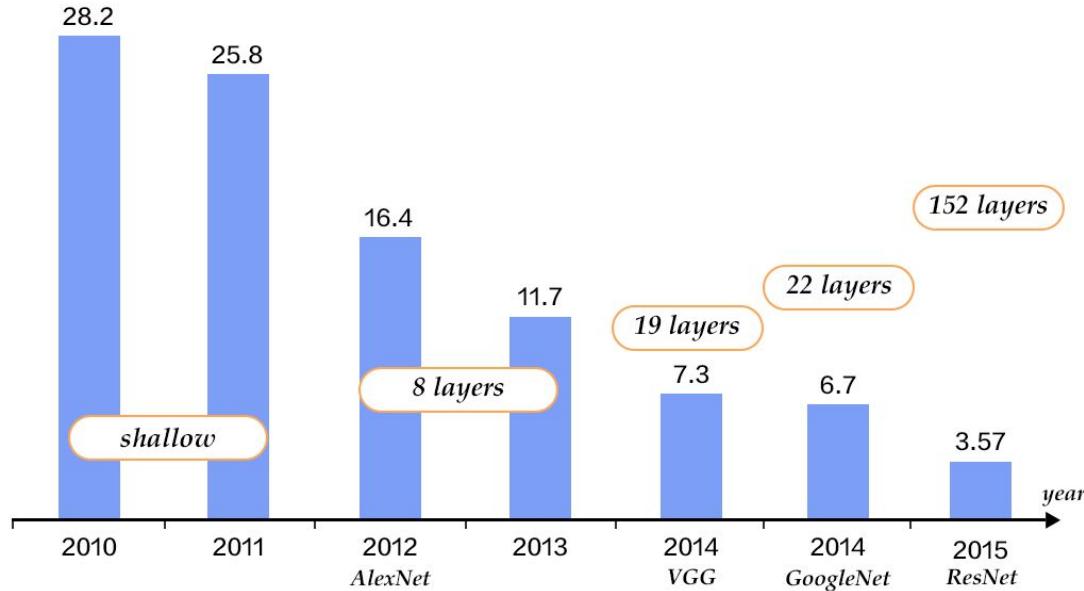


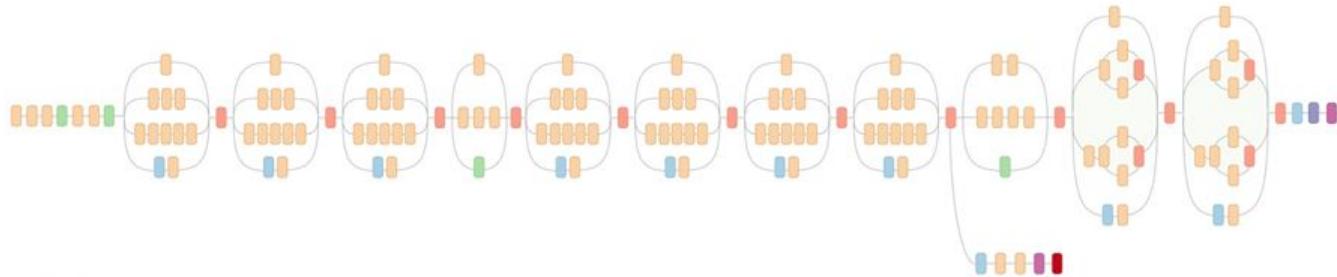
Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

# ILSVRC – Deeper and Better



# Why Deeper is Better ?



- Convolution
- AvgPool
- MaxPool
- Concat
- Dropout
- Fully connected
- Softmax

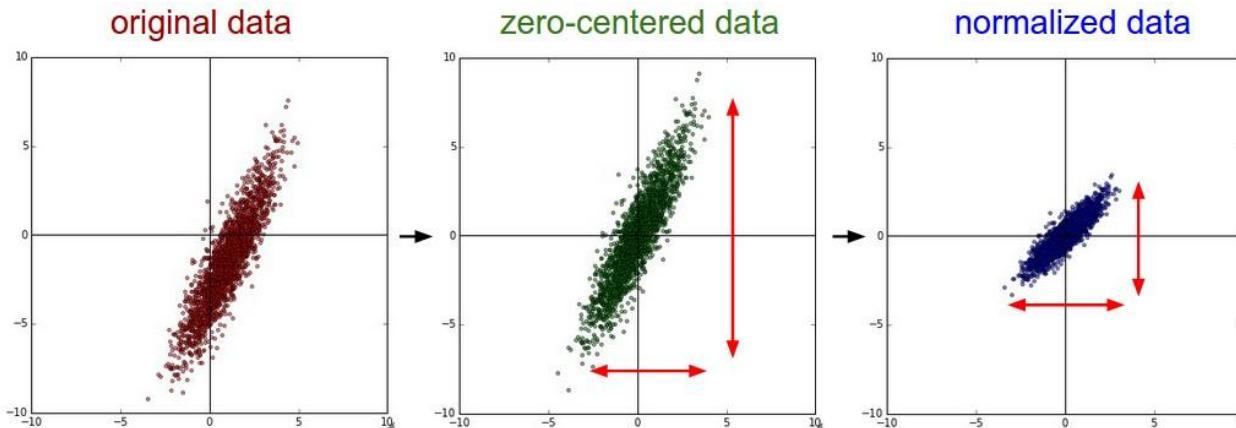


Visualizing and Understanding Deep Neural Networks by Matt Zeiler  
<https://www.youtube.com/watch?v=ghEmQSxT6tw>

# Recap: Input Normalization

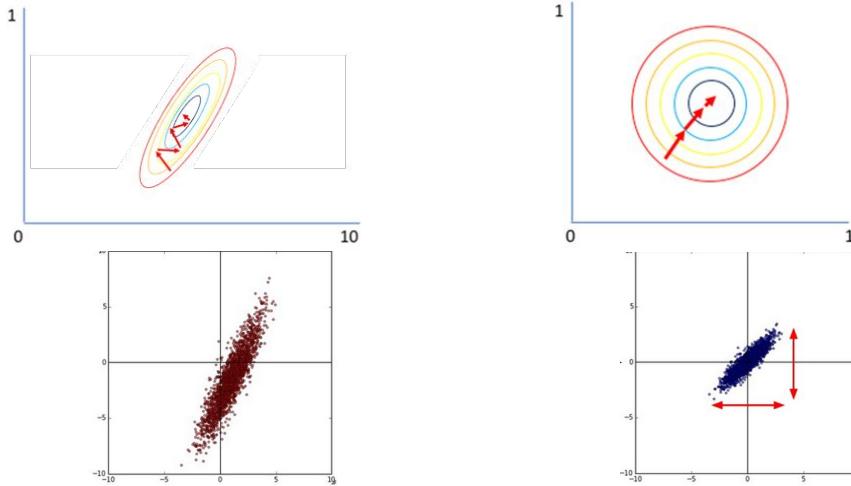
- Input mean and stdev normalized ->  $\mu=0, \sigma=1$

$$x_{i,j} = x_i - \mu(x_{i,j}) / \sigma(x_{i,j})$$

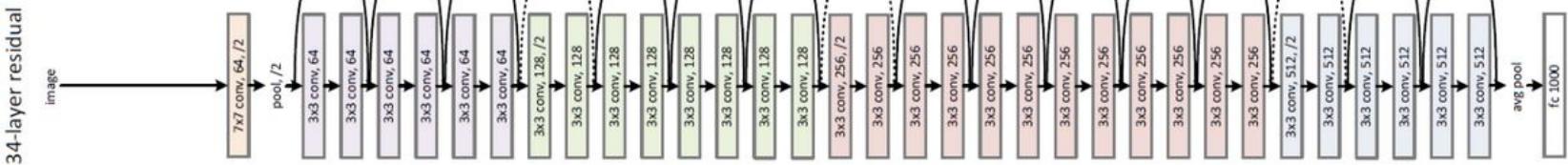


# Recap: Input Normalization

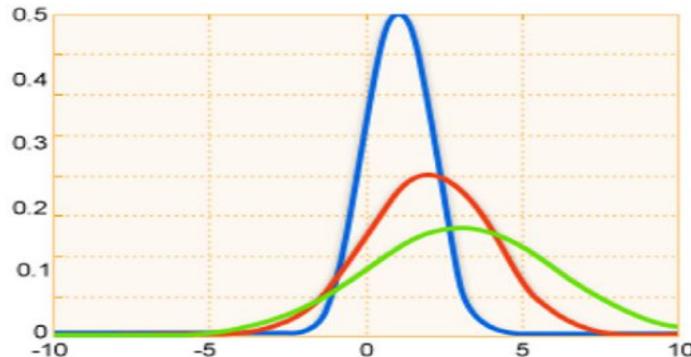
- All  $w_i$  updated according to same step-size/LR  $\eta$
- Assumption:  $dE/dw_i$  comparable for all  $w_i$ 
  - Otherwise, we would need separate  $\eta_i$  (complex problem)



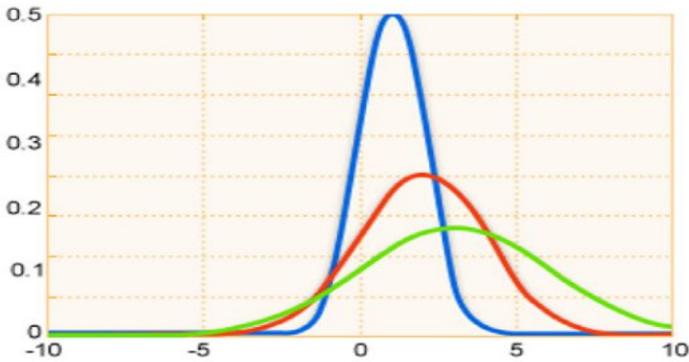
# Layers: Batch-normalization



- Problem: how to estimate  $\mu$ ,  $\sigma$  of hidden layers inputs?



# Layers: Batch-normalization



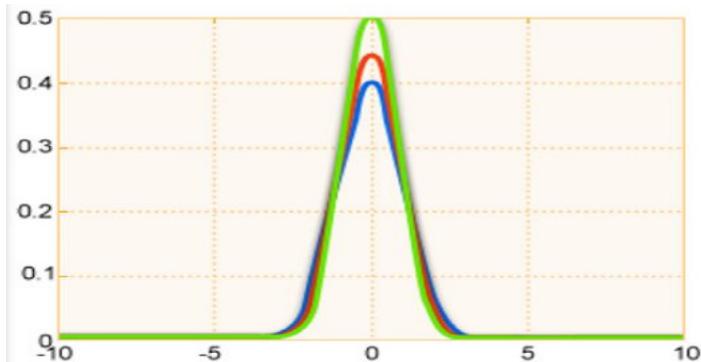
**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;  
Parameters to be learned:  $\gamma, \beta$   
**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

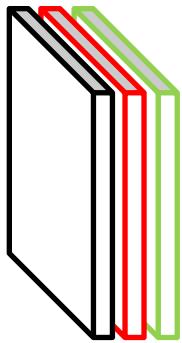
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

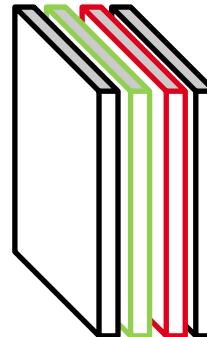


Batch Normalization: Accelerating Deep Network  
Training by Reducing Internal Covariate Shift  
Sergey Ioffe, Christian Szegedy

# Convolve-ReLU-Pool-BatchNorm



Conv + ReLU  
MaxPooling  
BatchNorm

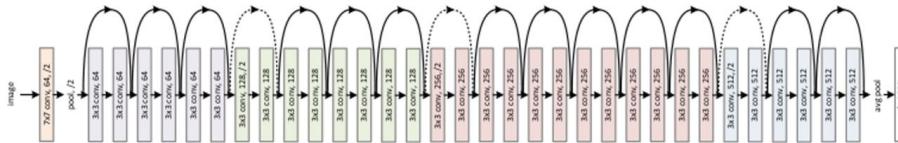


Conv  
MaxPooling  
BatchNorm  
ReLU

S. Ioffe, C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

# Training from Scratch

- Train *ResNet* to recognize K custom objects classes
  - Long training time



# The Cost of Training a Deep Network from Scratch

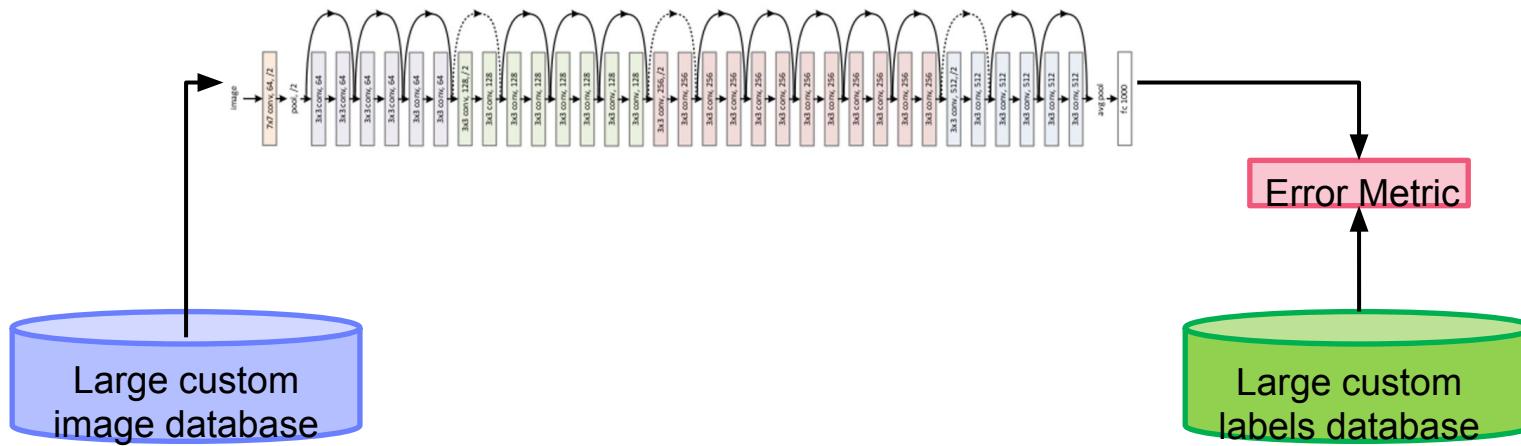
- AlexNet (2012) took 5~6 days over two GTX 580 GPUs
- Nowadays many startups *loan* training time

Select Open-Source Reference Models	Normal training cost (TF 1.8)	Preemptible training cost (TF 1.8)
<a href="#">ResNet-50</a> (with optimizations from fast.ai): Image classification	~\$25	~\$7.50
<a href="#">ResNet-50</a> (original implementation): Image classification	~\$59	~\$18
<a href="#">AmoebaNet</a> : Image classification (model architecture evolved from scratch on TPUs to maximize accuracy)	~\$49	~\$15
<a href="#">RetinaNet</a> : Object detection	~\$40	~\$12
<a href="#">Transformer</a> : Neural machine translation	~\$41	~\$13
<a href="#">ASR Transformer</a> : Speech recognition (transcribe speech to text)	~\$86	~\$27

Cost of training from scratch some deep convolutional networks over Google's TPU cloud (2017)  
[https://www.theregister.co.uk/2018/06/20/google\\_cloud\\_tpus/](https://www.theregister.co.uk/2018/06/20/google_cloud_tpus/)

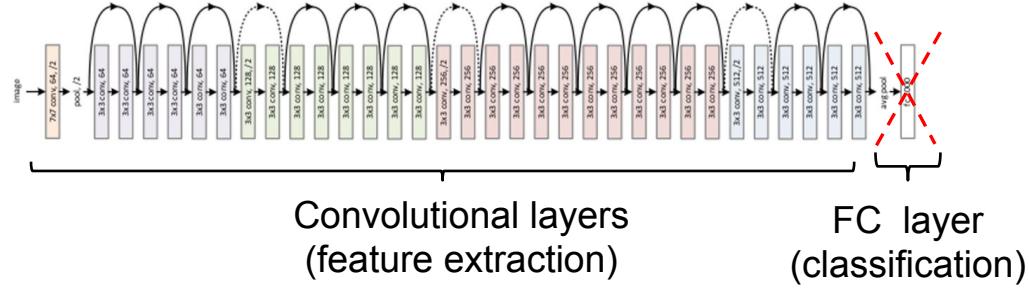
# Training from Scratch

- Train *ResNet* to recognize K custom objects classes
  - Long training time
  - Must collect and label **many** train samples



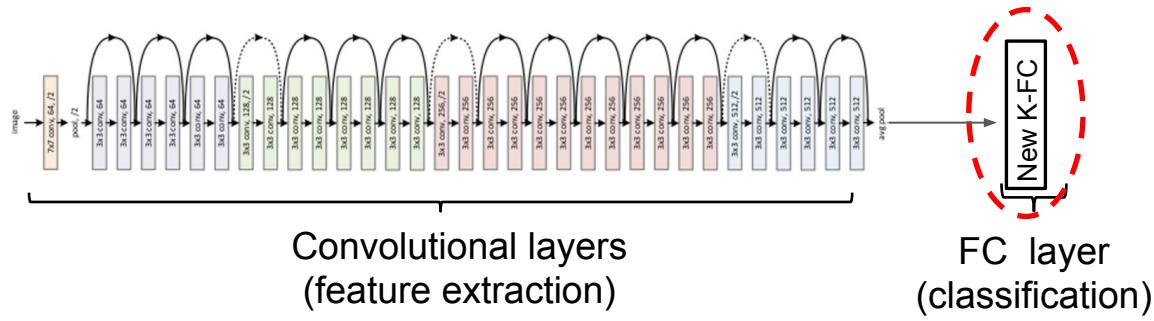
# Transfer Learning

Take ResNet pretrained on *ImageNet*



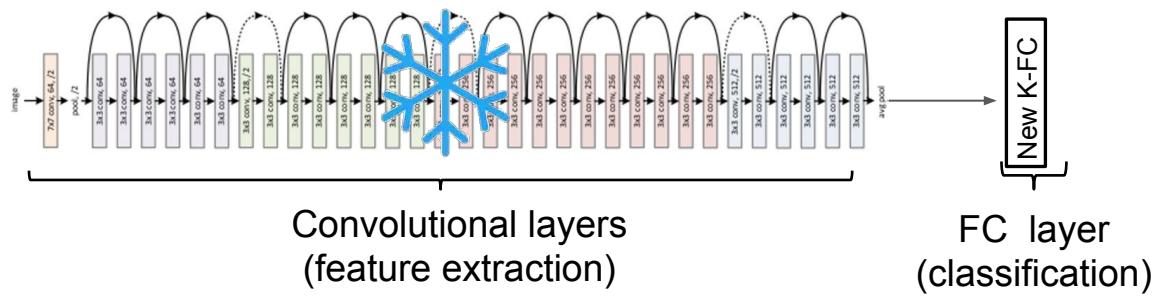
# Transfer Learning

- Take ResNet pretrained on *ImageNet*
- Replace FC layer(s) with ad-hoc K-units FC layer



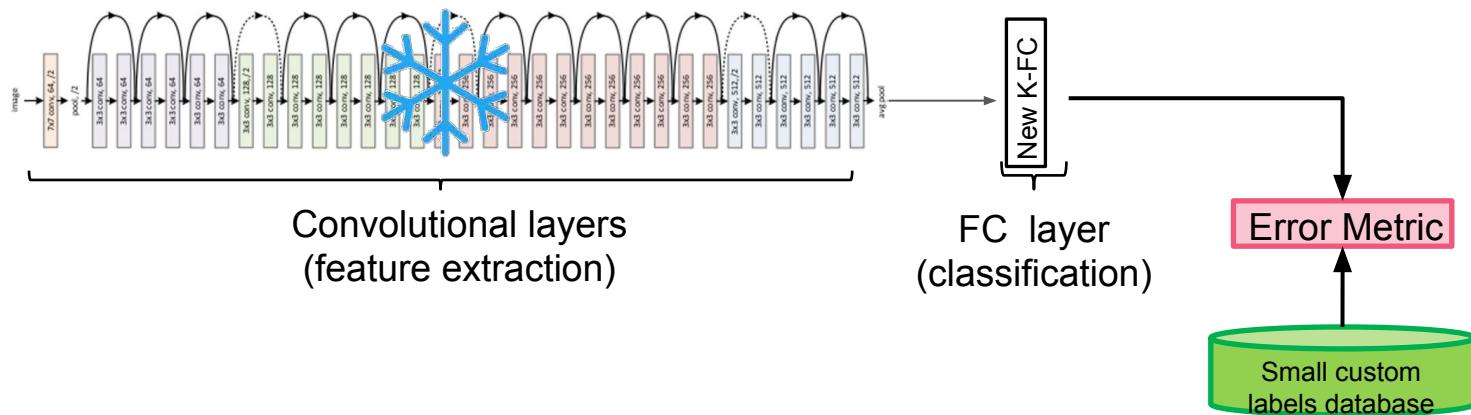
# Transfer Learning

- Take ResNet pretrained on *ImageNet*
- Replace FC layer(s) with ad-hoc K-units FC layer
- Freeze (early) convolutional layers ( $\eta=0$  or close to)



# Transfer Learning

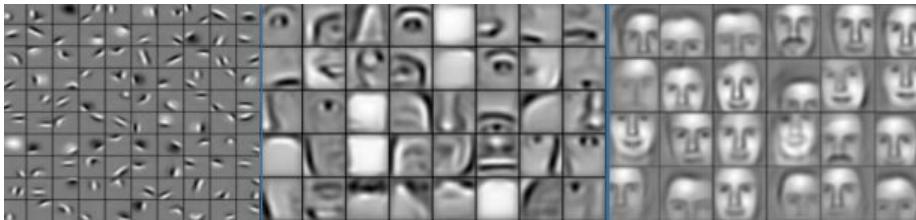
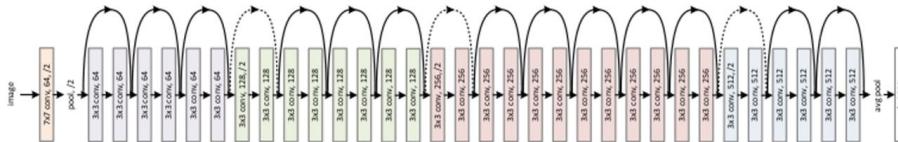
- Take ResNet pretrained on *ImageNet*
- Replace FC layer(s) with ad-hoc K-units FC layer
- Freeze (early) convolutional layers ( $\eta=0$  or close to)



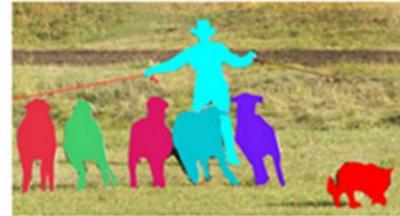
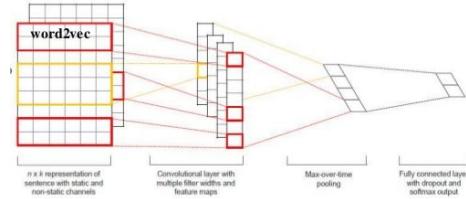
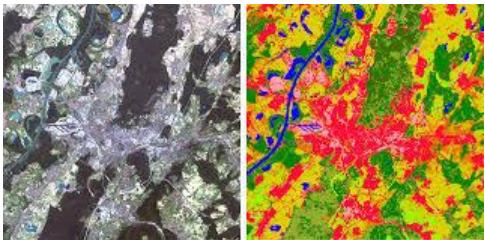
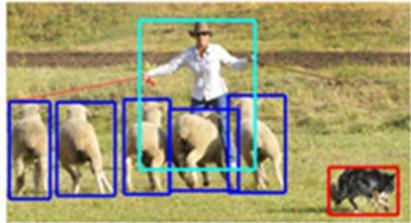
- Refine network over **small** custom dataset

# Transfer Learning – Why it Works ?

- Early conv. layers more difficult to trend (faint error gradients)
  - Very low level filters (edges, etc.)
  - «Reusing» pre-learned feature detectors



# CNNs – Beyond Image Classification



# Who's Who



Yoshua Bengio  
*Université de Montréal*



Geoffrey Hinton  
*Google  
University of Toronto*



Yann LeCun  
*Facebook AI Research  
New York University*

# References (1)

- Y. LeCun, L. Bottou, Y. Bengio and P. Haffner. "*Gradient-Based Learning Applied to Document Recognition*", Proceedings of the IEEE 86, no. 11 (1998): 2278-2324.
- D. Williams, G. Hinton. "*Learning representations by back-propagating errors.*" Nature 323, no. 6088 (1986): 533-538.
- A. Krizhevsky, I. Sutskever, G. E. Hinton. "*Imagenet classification with deep convolutional neural networks.*" In Advances in neural information processing systems, pp. 1097-1105. 2012.
- C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich. "*Going deeper with convolutions.*" In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1-9. 2015.
- K. Simonyan, A. Zisserman. "*Very deep convolutional networks for large-scale image recognition.*" arXiv preprint arXiv:1409.1556 (2014).
- P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun. "*Overfeat: Integrated recognition, localization and detection using convolutional networks.*" arXiv preprint arXiv:1312.6229 (2013).
- S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. Kruthiventi, R. V. Babu. "*A Taxonomy of Deep Convolutional Neural Nets for Computer Vision.*" arXiv preprint arXiv:1601.06615 (2016).
- G.S. Hsu, J.C. Chen, Y.Z. Chung. "*Application-oriented license plate recognition.*" IEEE Transactions on Vehicular technology 62, no. 2 (2013): 552-561.
- H. Li, C. Shen. "*Reading Car License Plates Using Deep Convolutional Neural Networks and LSTMs*" arXiv preprint

# References (2)

Jürgen Schmidhuber. "Deep learning in neural networks: An overview." *Neural networks* 61 (2015): 85-117.

C. M. Bishop. "Neural networks for pattern recognition" Oxford university press, 1995.

Jürgen Schmidhuber. "Deep learning in neural networks: An overview", Neural Networks Volume 61, pp. 85–117 (2015)

# Other Resources

Stanford Press “*Feature extraction using convolution*”

[http://deeplearning.stanford.edu/wiki/index.php/Feature\\_extraction\\_using\\_convolution](http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution)

Ujjwal Karn, “*An Intuitive Explanation of Convolutional Neural Networks*”

<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

Sebastian Raschka, “*Single-Layer Neural Networks and Gradient Descent*”

[https://sebastianraschka.com/Articles/2015\\_singlelayer\\_neurons.html](https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html)

Denny Britz, “*Understanding Convolutional Neural Networks for NLP*”

<http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>

Adit Deshpande, “*A Beginner’s Guide To Understanding Convolutional Neural Networks*”

<https://adeshpande3.github.io/adeshpande3.github.io/Networks/>

The University of British Columbia, “CPSC 522 - Artificial Intelligence 2”

<http://wiki.ubc.ca/Course:CPSC522>

Michael Nielsen, “*Neural Networks and Deep Learning*”

<http://neuralnetworksanddeeplearning.com/index.html>

I. Goodfellow, Y. Bengio, A. Courville, “*Deep Learning*”

<http://www.deeplearningbook.org/>

# Questions ?

