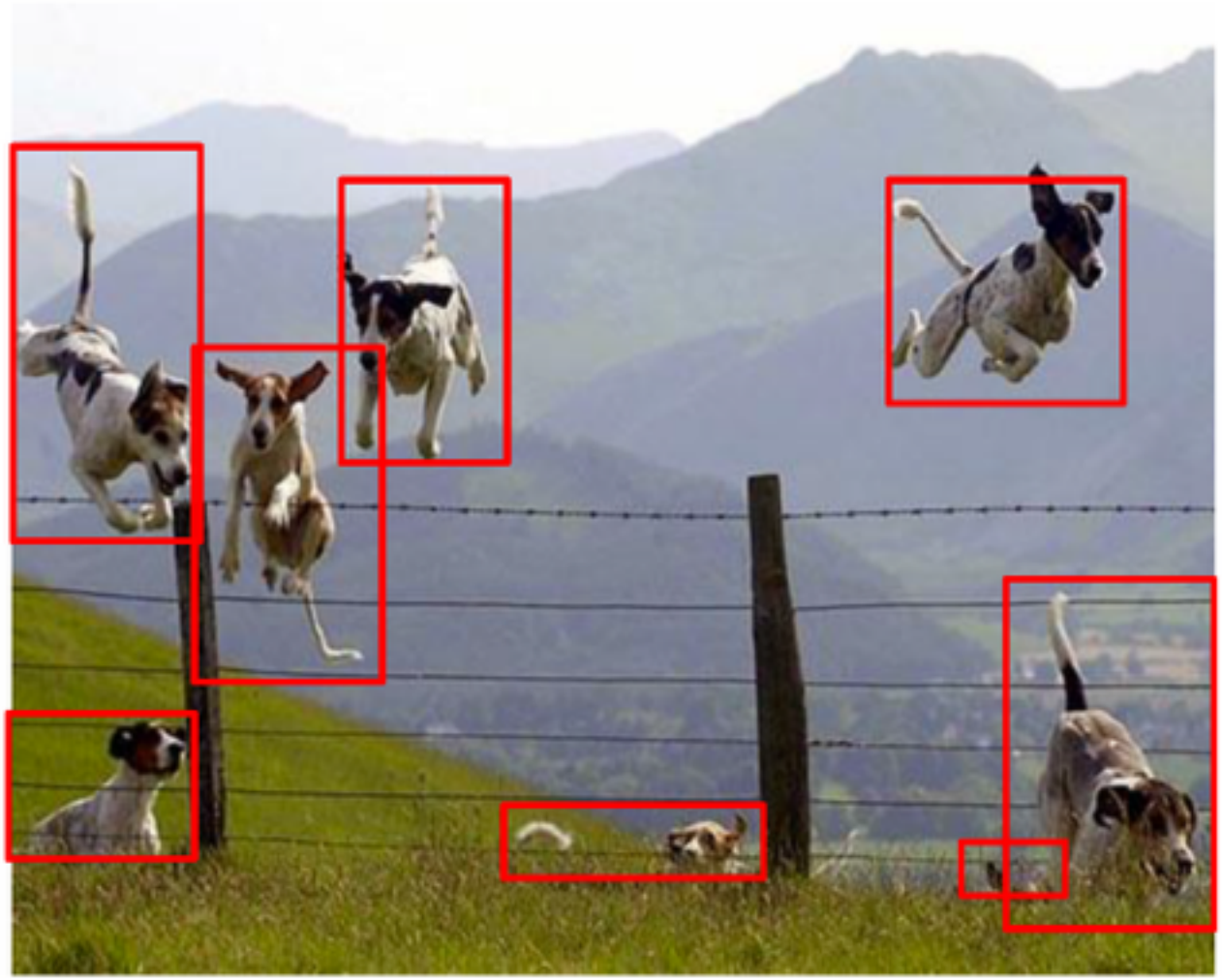
Intro to Deep Learning

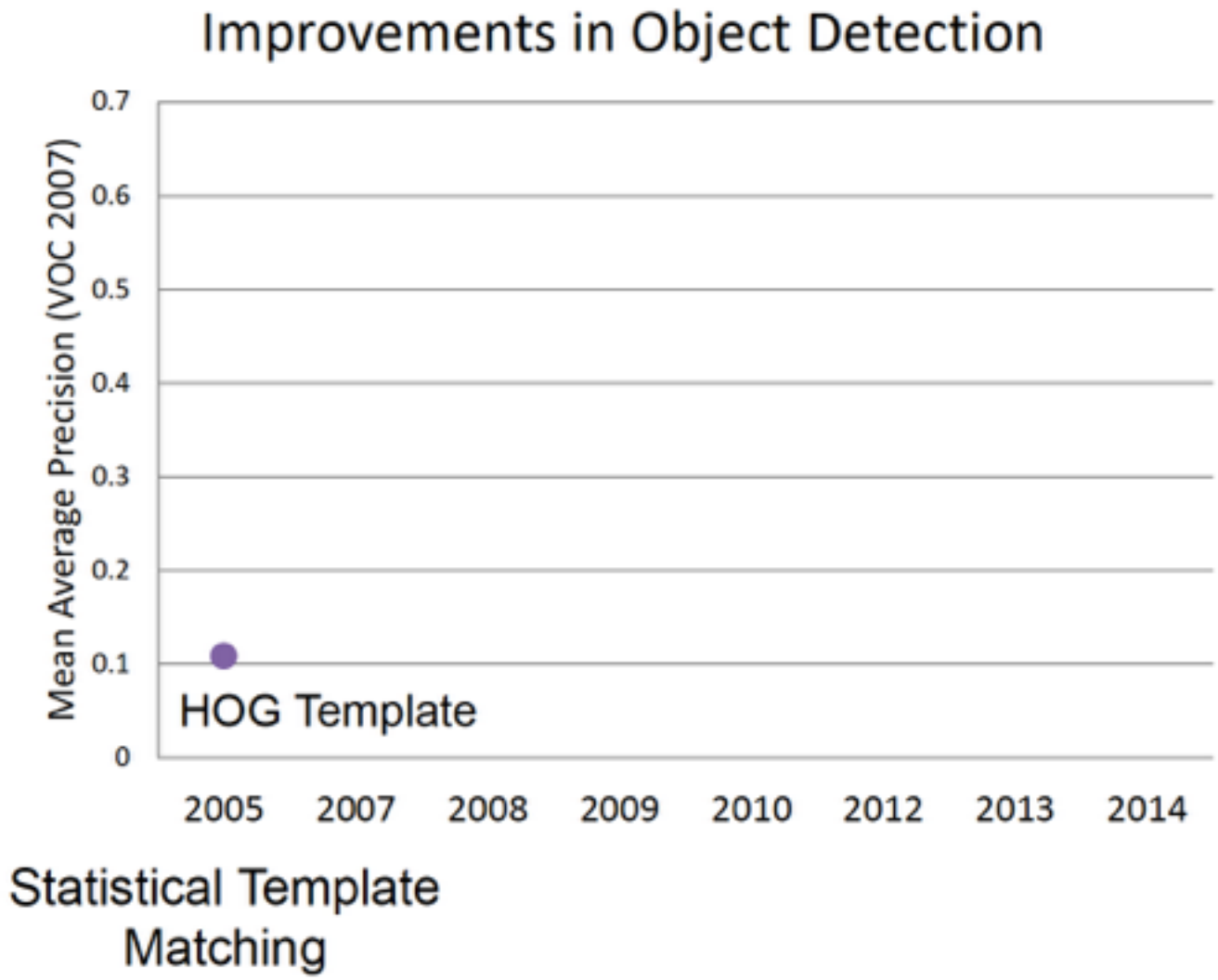
Slides Credit: Andrej Karapathy, Derek Hoiem, Marc Aurelio, Yann LeCunn

Why this class?

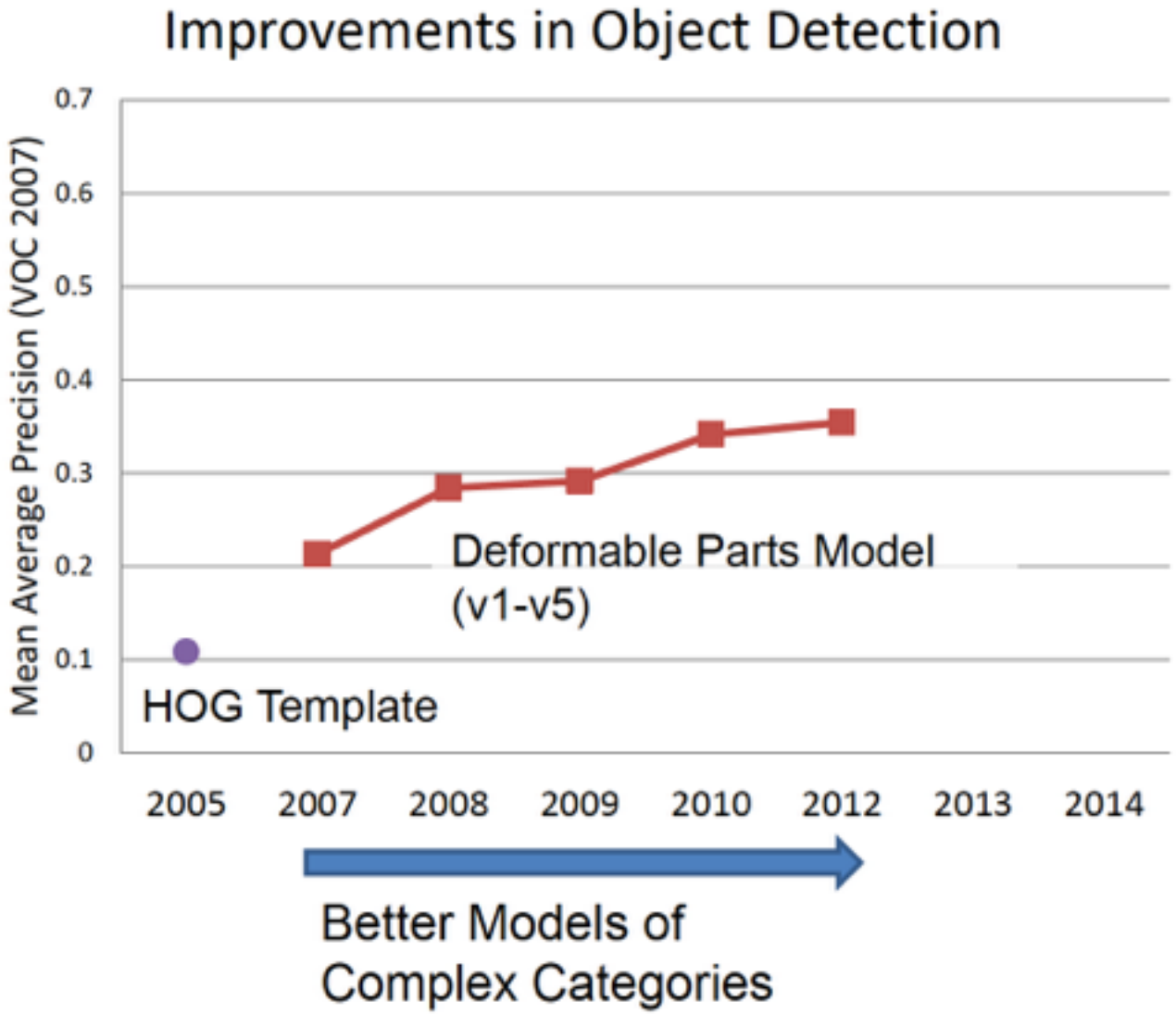
Object detection



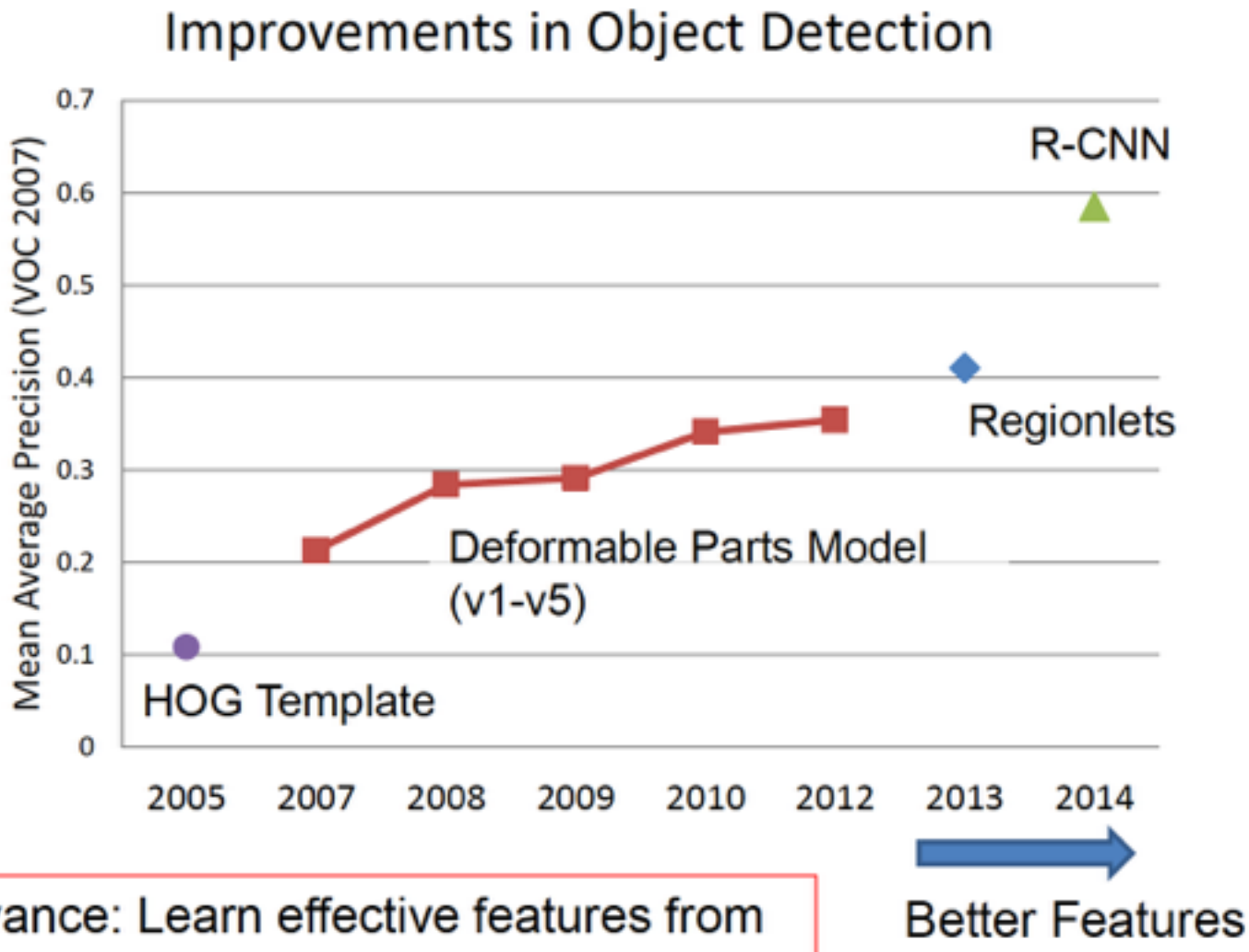
Where are the objects of interest?



HOG: Dalal-Triggs 2005



HOG: Dalal-Triggs 2005 DPM: Felzenszwalb et al. 2008-2012



Key Advance: Learn effective features from massive amounts of labeled data *and* adapt to new tasks with less data

HOG: Dalal-Triggs 2005 DPM: Felzenszwalb et al. 2008-2012 Regionlets: Wang et al. 2013 R-CNN: Girshick et al. 2014

Deep Features

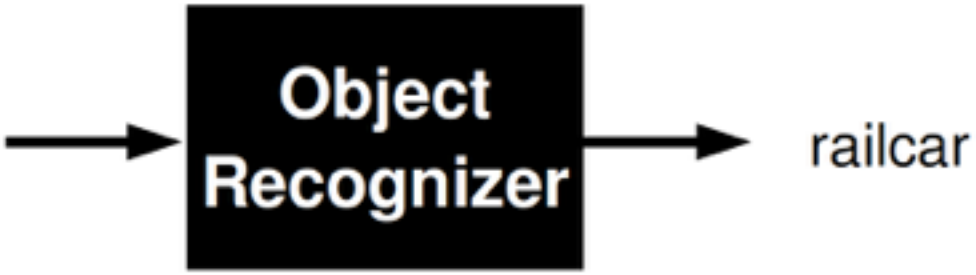
- Have been able to harness the big data in the most efficient and effective manner.

- Lead to several state of the art results!

CONV NETS: EXAMPLES

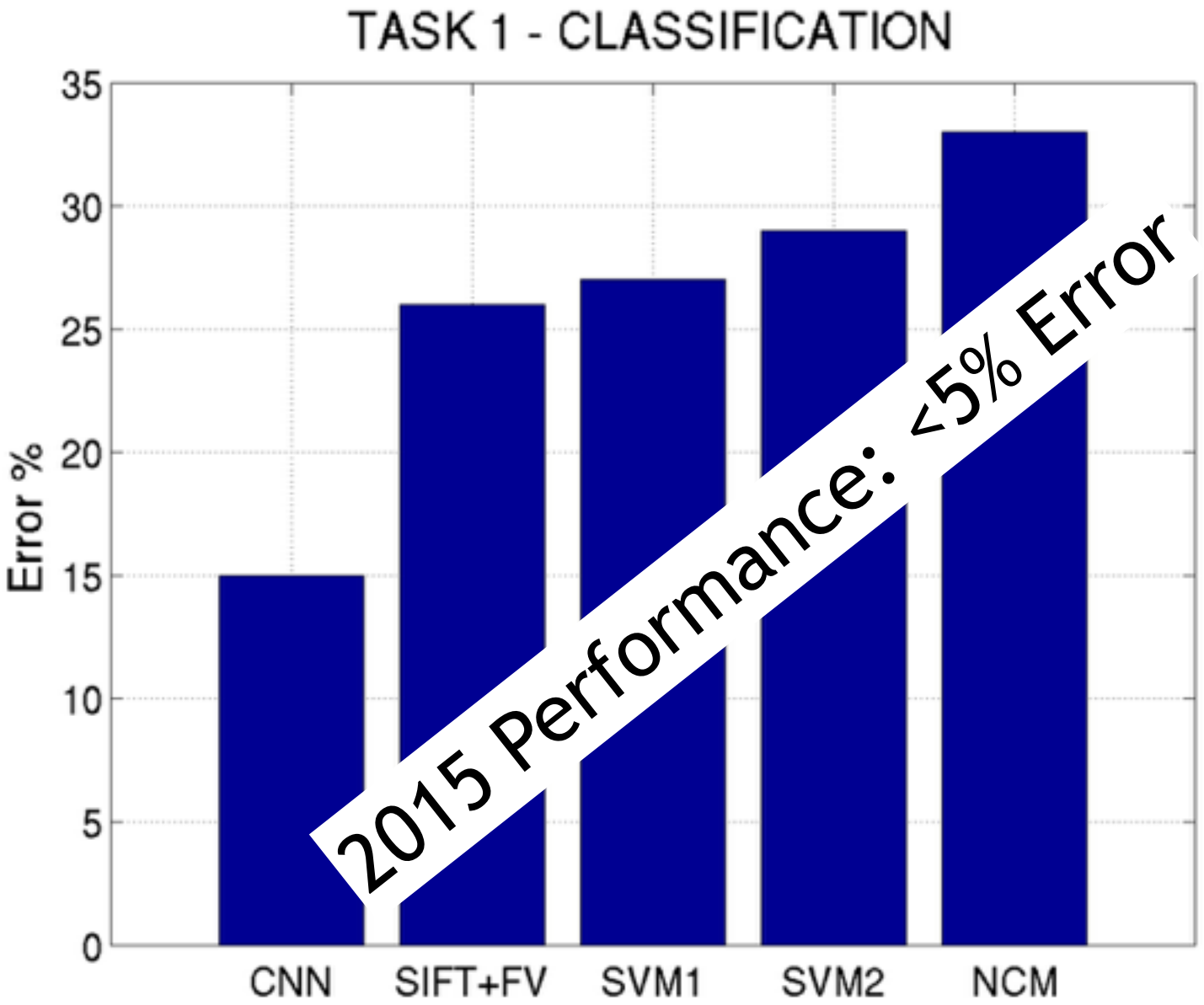
- **Image classification**

IMAGENET



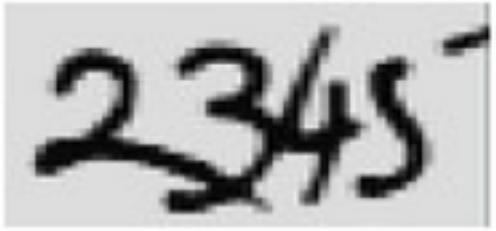
Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012

Results: ILSVRC 2012



CONV NETS: EXAMPLES

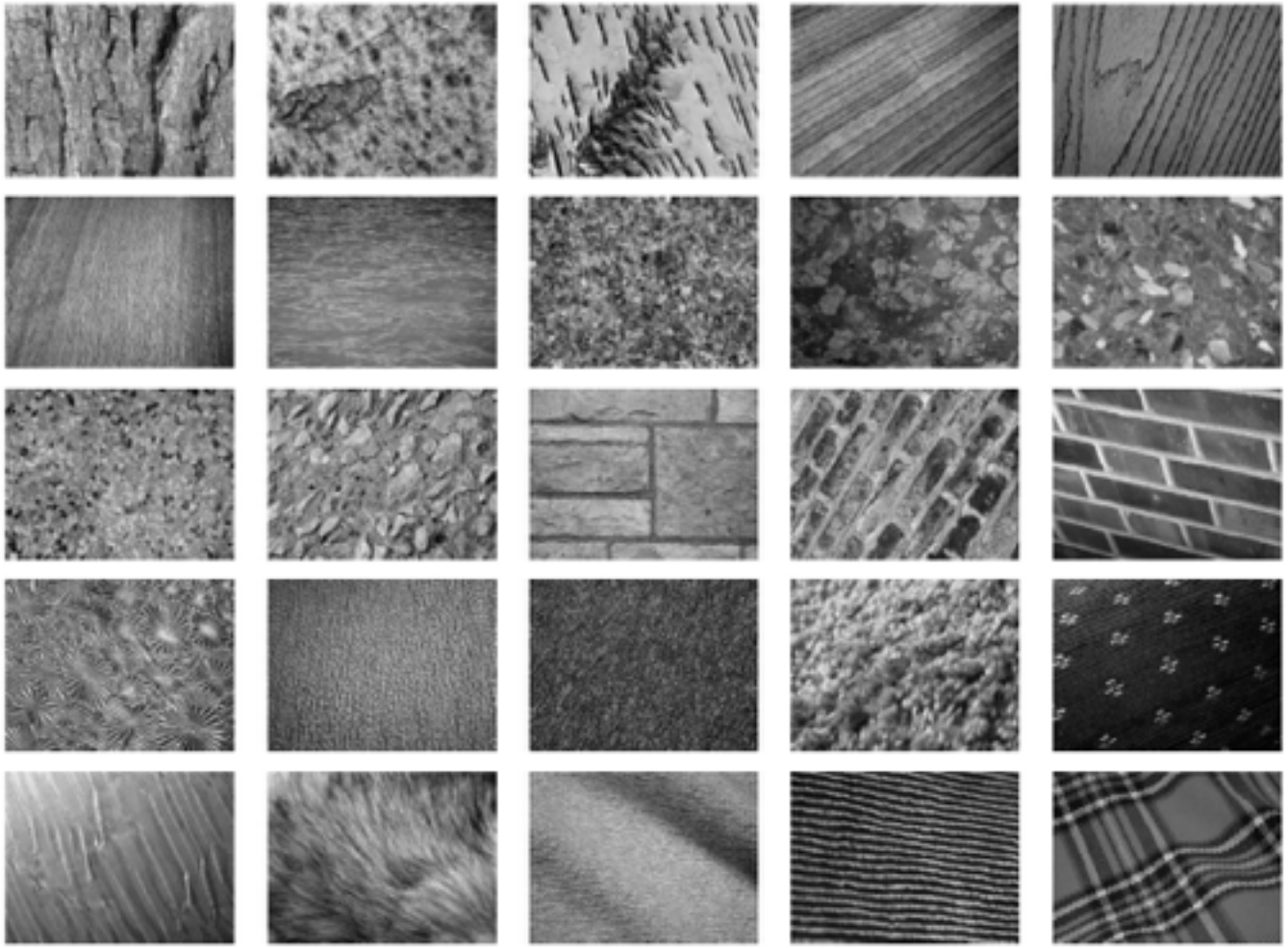
- OCR / House number & Traffic sign classification



Ciresan et al. “MCDNN for image classification” CVPR 2012
Wan et al. “Regularization of neural networks using dropconnect” ICML 2013

CONV NETS: EXAMPLES

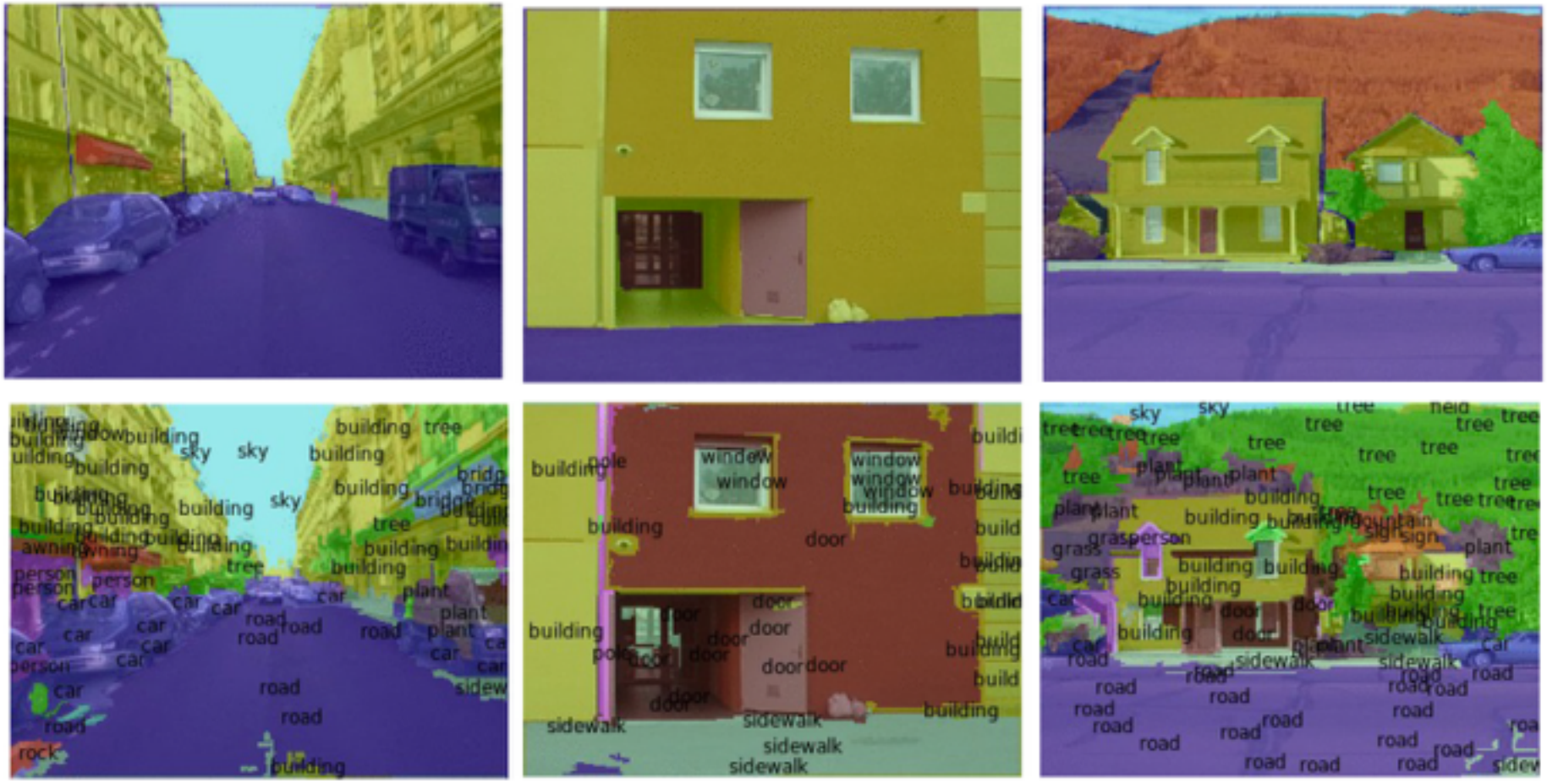
- **Texture classification**



Sifre et al. “Rotation, scaling and deformation invariant scattering...” CVPR 2013

CONV NETS: EXAMPLES

- **Scene Parsing**



Farabet et al. “Learning hierarchical features for scene labeling” PAMI 2013

CONV NETS: EXAMPLES

- Action recognition from videos

Taylor et al. “Convolutional learning of spatio-temporal features” ECCV 2010

What is deep learning?

Ideal Features

Ideal Feature Extractor

- window, top-left
- clock, top-middle
- shelf, left
- drawing,middle
- statue, bottom left
- …
- hat, bottom right

Ideal Features Are Non-Linear

I_1

→ **Ideal Feature Extractor** →

- club, **angle = 90**
- man, frontal pose

...

?

→ **Ideal Feature Extractor** →

- club, **angle = 270**
- man, frontal pose

...

I_2

→ **Ideal Feature Extractor** →

- club, **angle = 360**
- man, side pose

...

Ideal Features Are Non-Linear

I_1

Ideal Feature Extractor

- club, **angle = 90**
- man, frontal pose
- ...

INPUT IS NOT THE AVERAGE!

Ideal Feature Extractor

- club, **angle = 270**
- man, frontal pose
- ...

I_2

Ideal Feature Extractor

- club, **angle = 360**
- man, side pose
- ...

Ideal Features Are Non-Linear

I_1

→ Ideal Feature Extractor →

- club, **angle = 90**
- man, frontal pose

...

→ Ideal Feature Extractor →

- club, **angle = 270**
- man, frontal pose

...

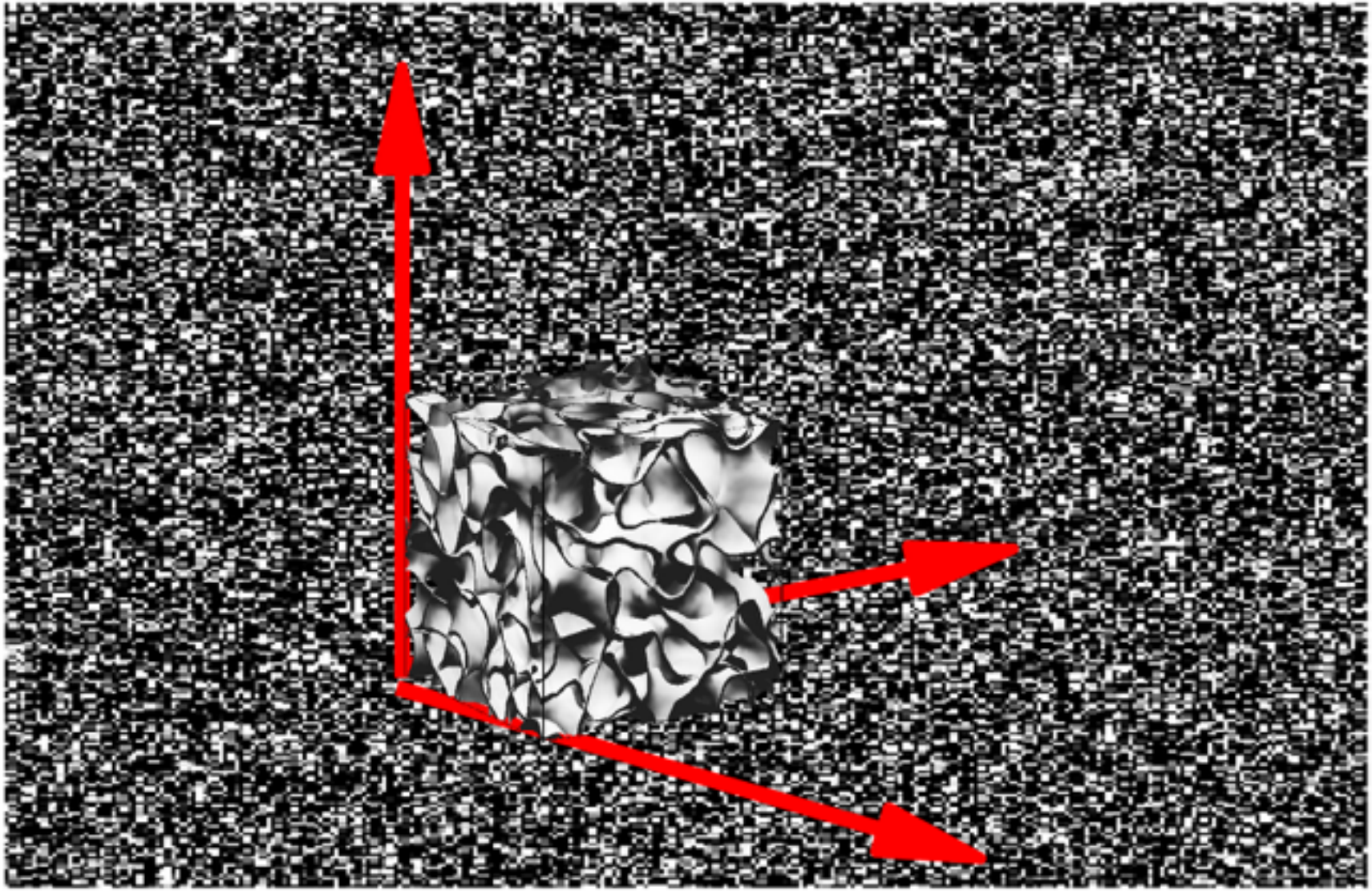
I_2

→ Ideal Feature Extractor →

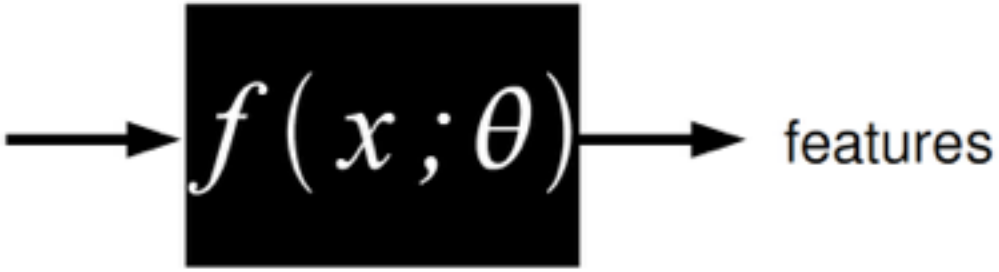
- club, **angle = 360**
- man, side pose

...

Space of Visual Features: Non Linear



Learning Non-Linear Features

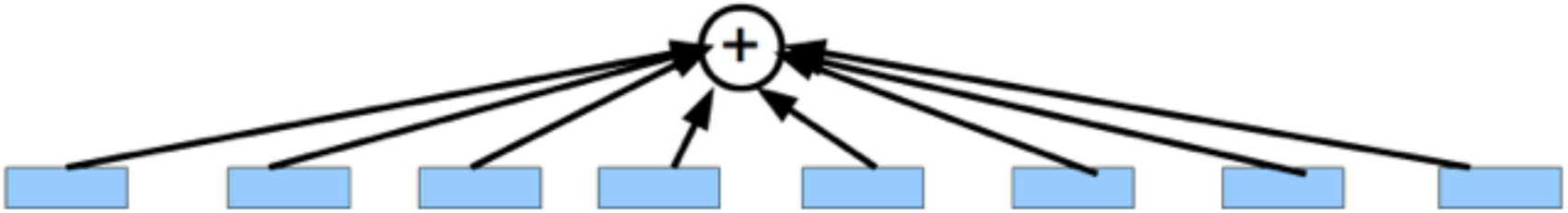


Q.: which class of non-linear functions shall we consider?

Learning Non-Linear Features

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

Proposal #1: linear combination $f(x) \approx \sum_j g_j$



Proposal #2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$

Learning Non-Linear Features

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

Proposal #1: linear combination $f(x) \approx \sum_j g_j$

- Kernel learning
- Boosting
- ...

Shallow

Proposal #2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$

- Deep learning
- Scattering networks (wavelet cascade)
- S.C. Zhou & D. Mumford "grammar"

Deep

- Theoretician's dilemma: "We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?"

$$y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \qquad y = F(W^1.F(W^0.X))$$

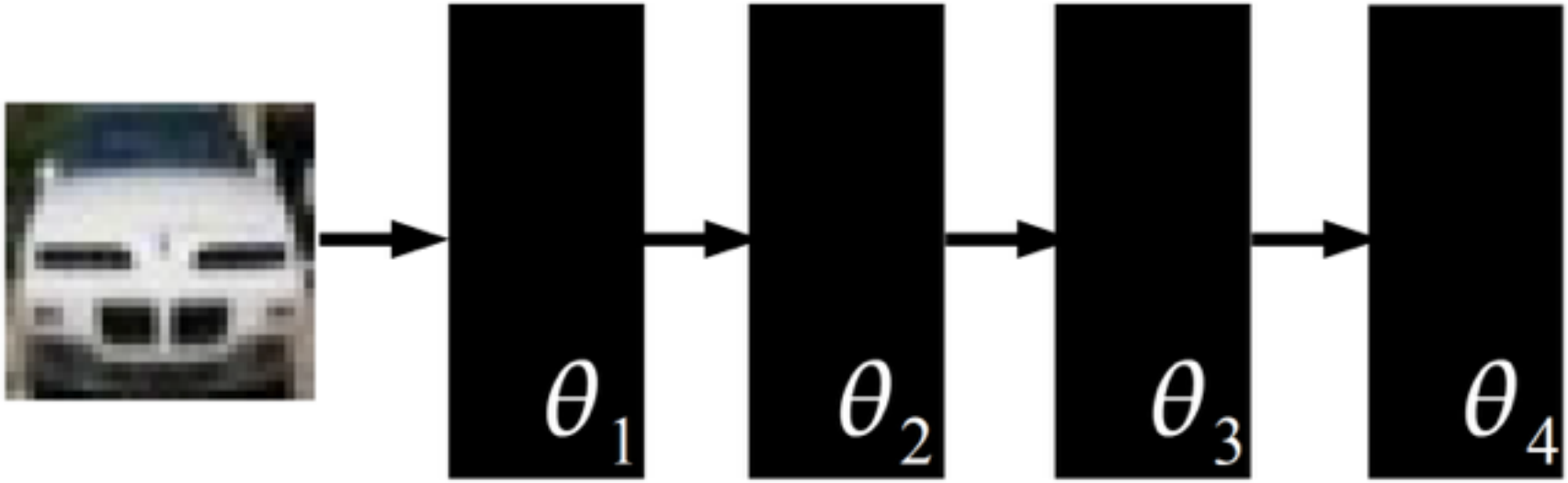
  - kernel machines (and 2-layer neural nets) are "universal".
- Deep learning machines

$$y = F(W^K.F(W^{K-1}.F(....F(W^0.X)...)))$$

- Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition
  - they can represent more complex functions with less "hardware"
- We need an efficient parameterization of the class of functions that are useful for "AI" tasks (vision, audition, NLP...)

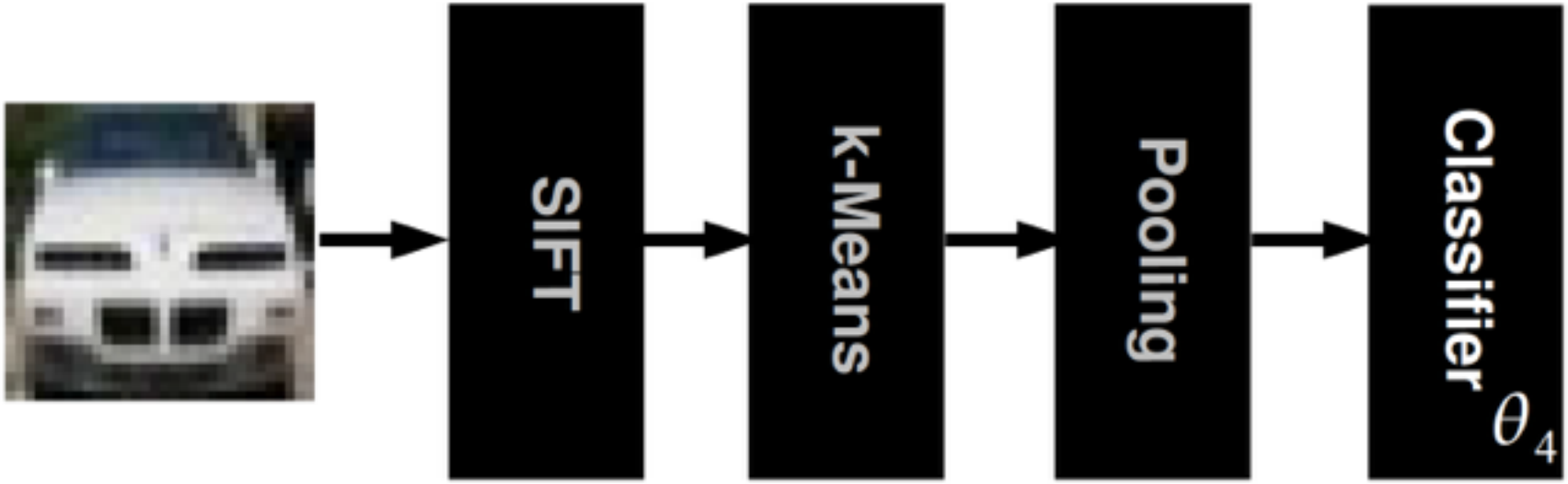
A Potential Problem with Deep Learning

Optimization is difficult: non-convex, non-linear system



A Potential Problem with Deep Learning

Optimization is difficult: non-convex, non-linear system

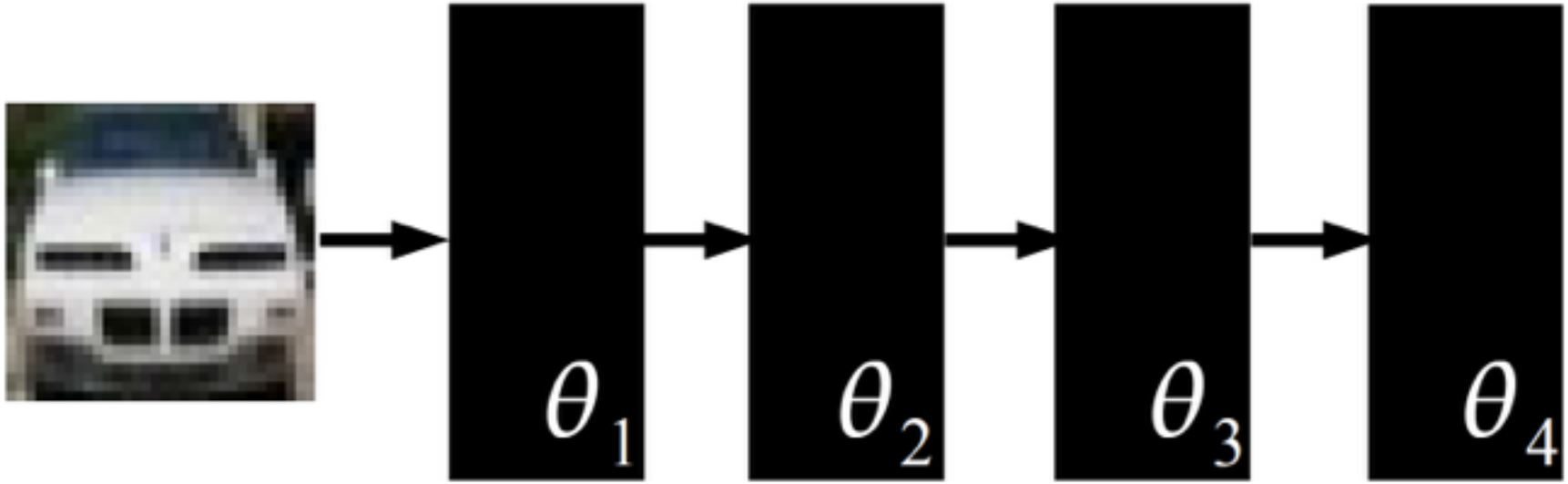


Solution #1: freeze first N-1 layer (engineer the features)

It makes it shallow!

A Potential Problem with Deep Learning

Optimization is difficult: non-convex, non-linear system



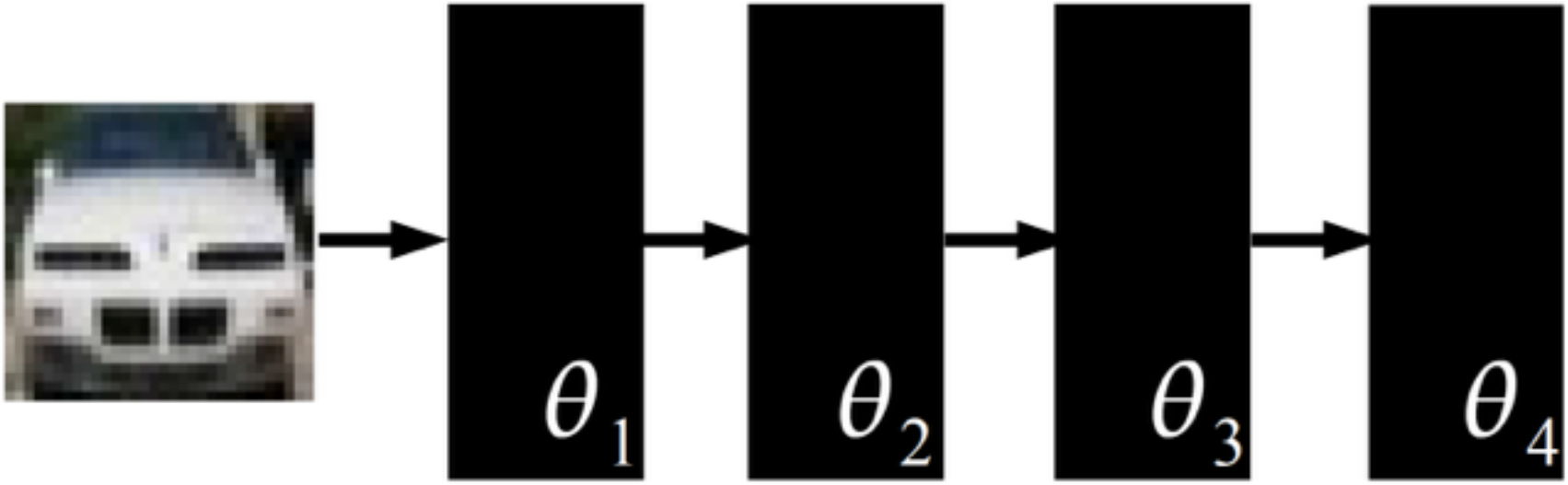
Solution #2: live with it!

It will converge to a local minimum.

It is much more powerful!!

Deep Learning in Practice

Optimization is easy, need to know a few tricks of the trade.



Q: What's the feature extractor? And what's the classifier?

A: No distinction, end-to-end learning!

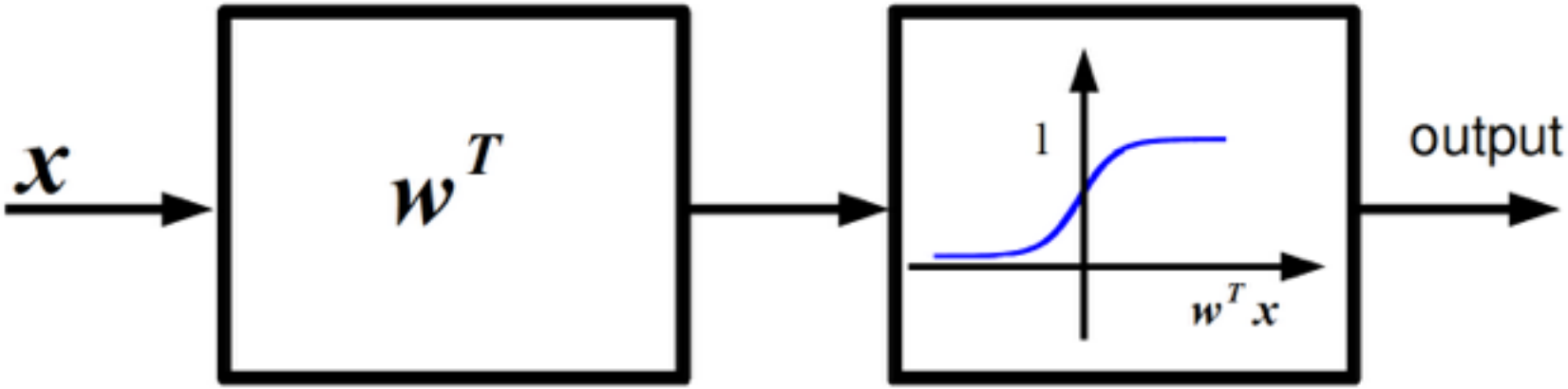
A Deep Learning method is: a method which makes predictions by using a sequence of non-linear processing stages. The resulting intermediate representations can be interpreted as feature hierarchies and the whole system is jointly learned from data.

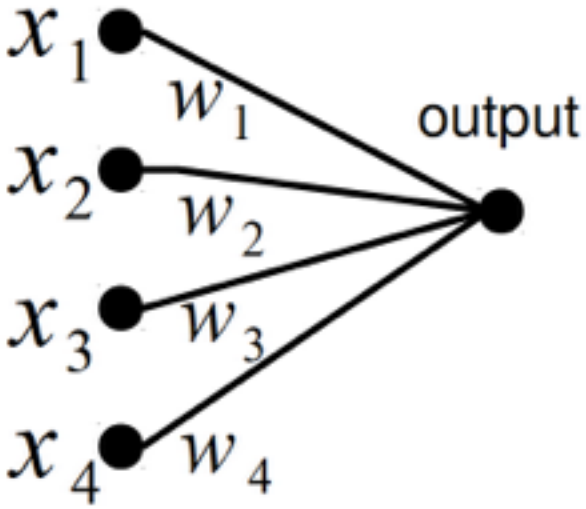
Some deep learning methods are probabilistic, others are loss-based, some are supervised, other unsupervised...

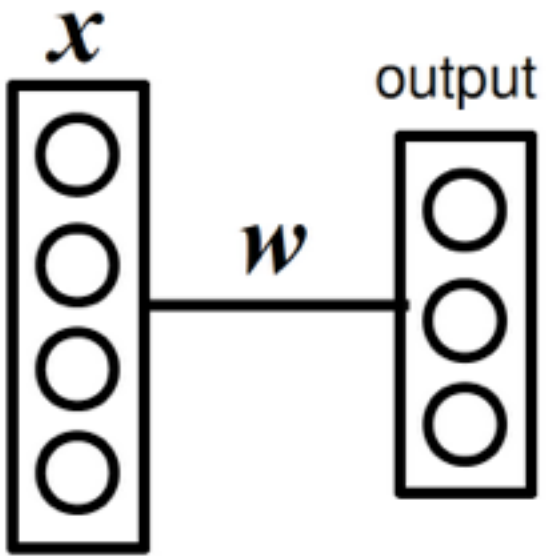
It's a large family!

Deep Neural Networks

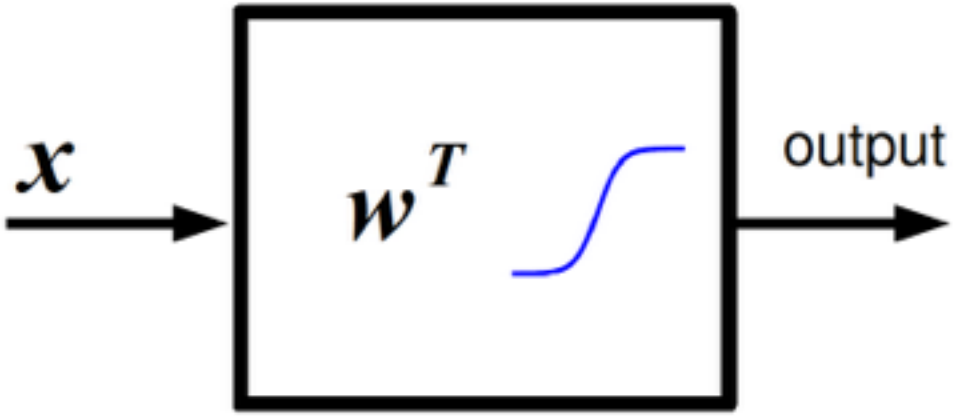
Graphical Representation

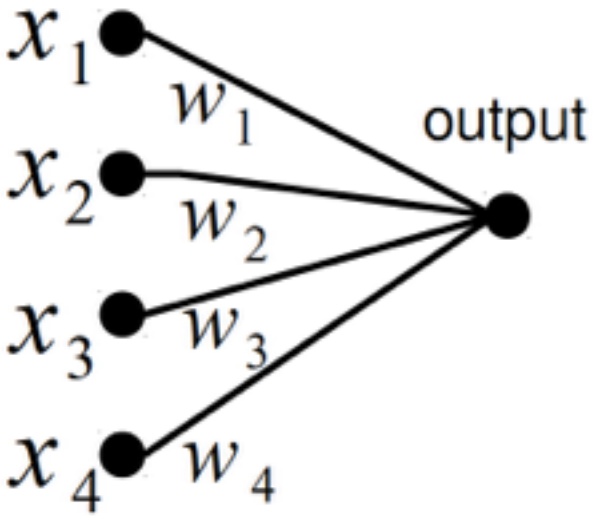


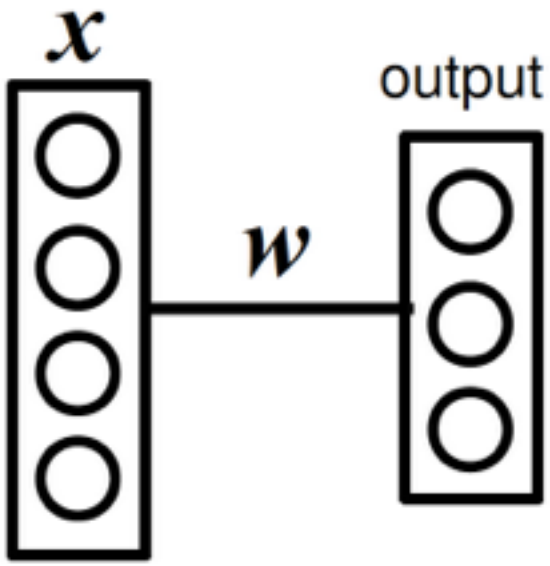




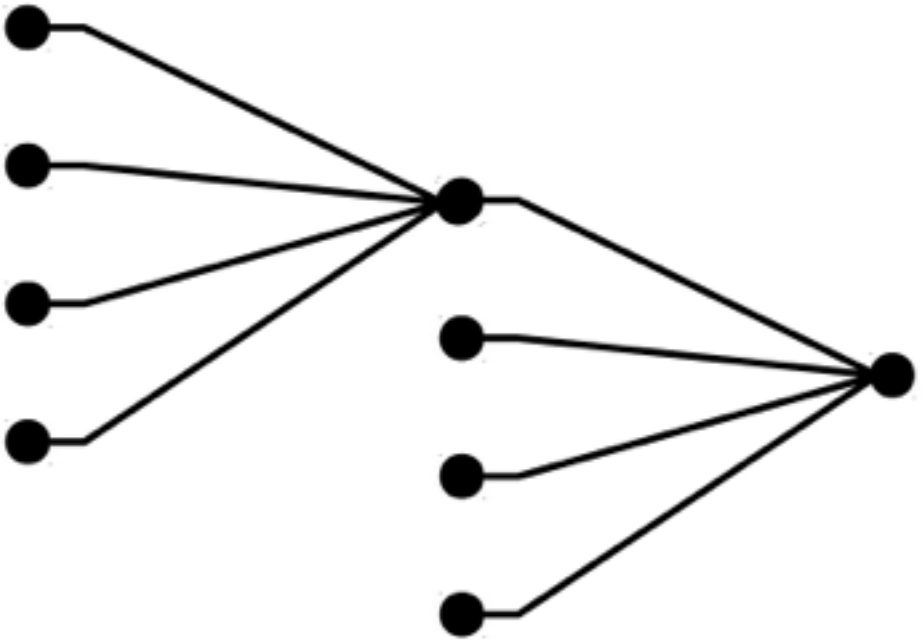
Graphical Representation



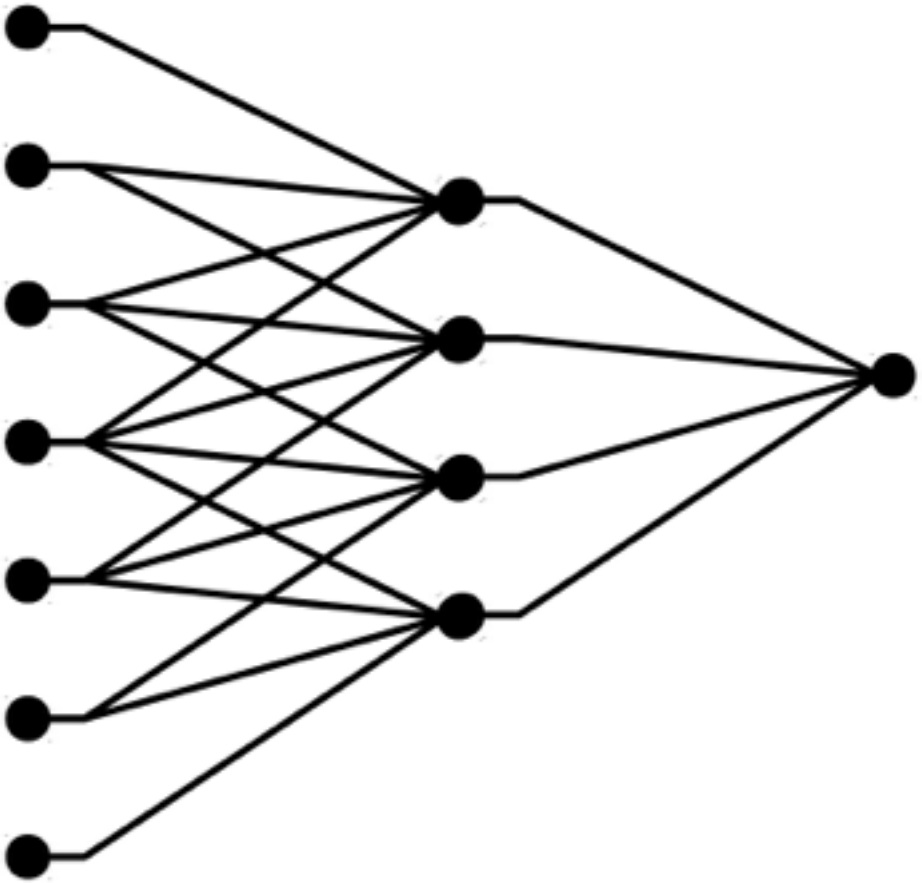




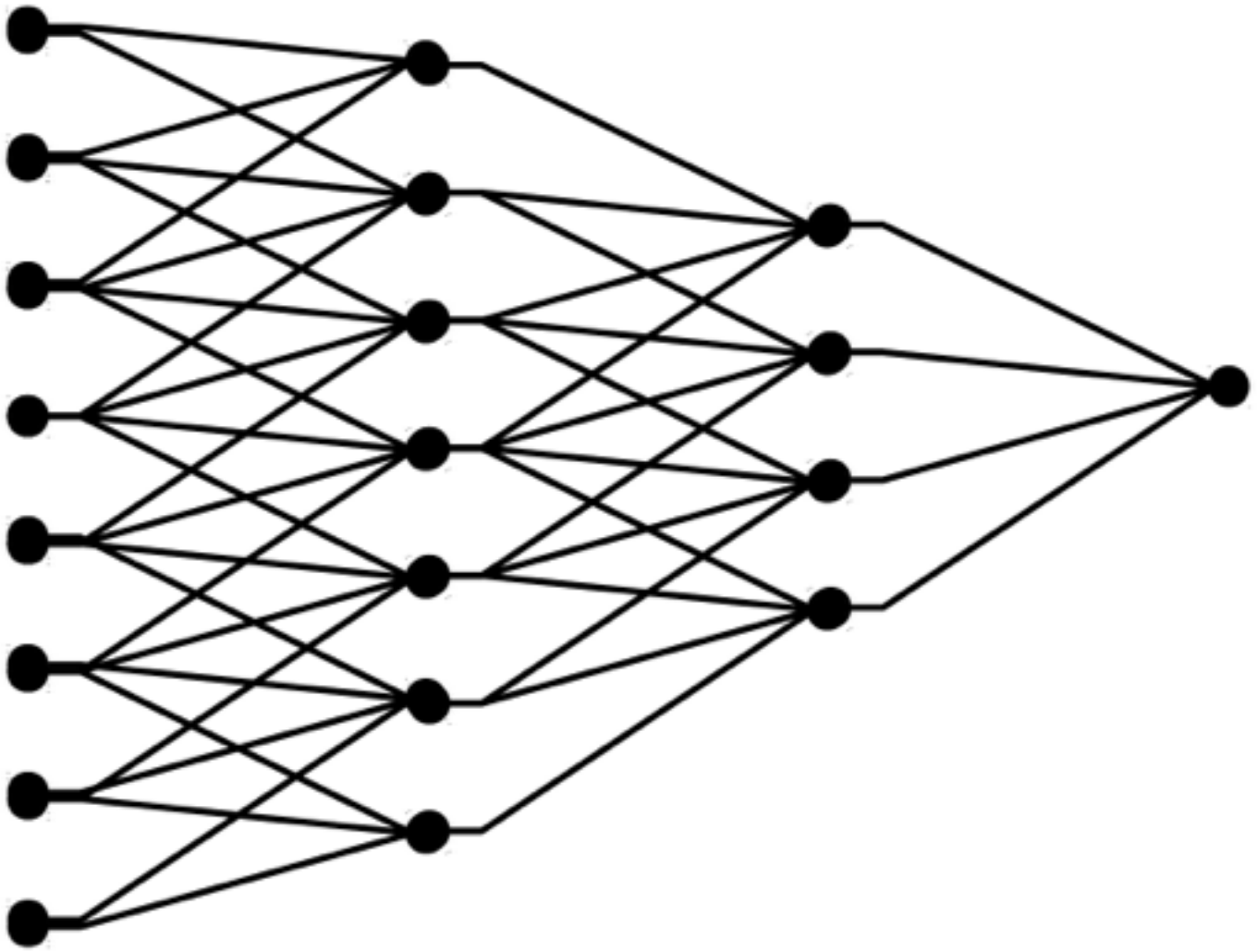
From Logistic Regression To Neural Nets



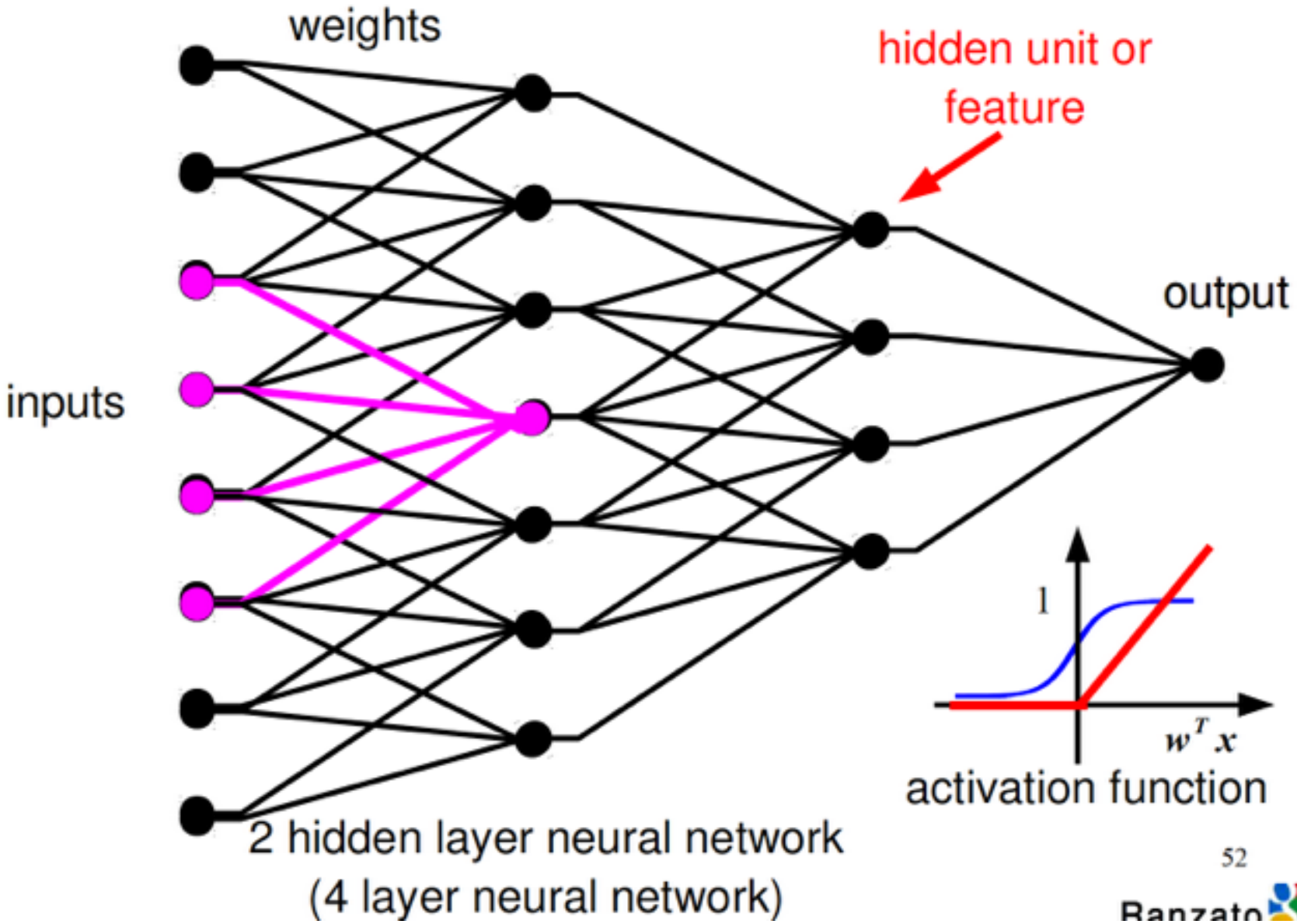
From Logistic Regression To Neural Nets



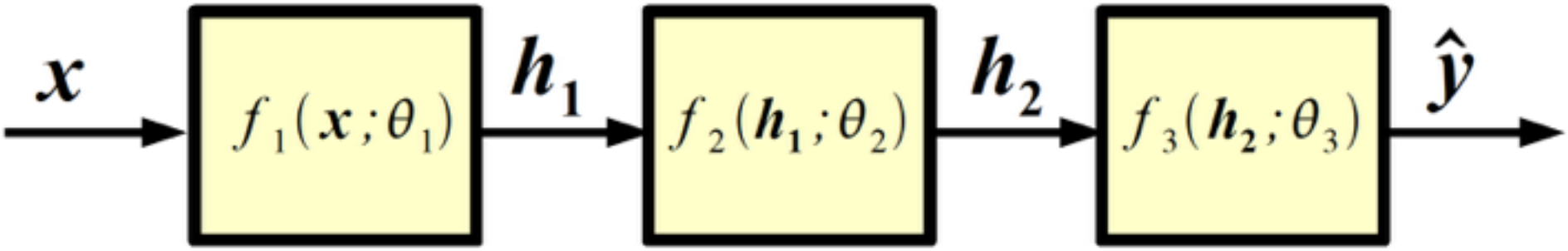
From Logistic Regression To Neural Nets



Neural Network



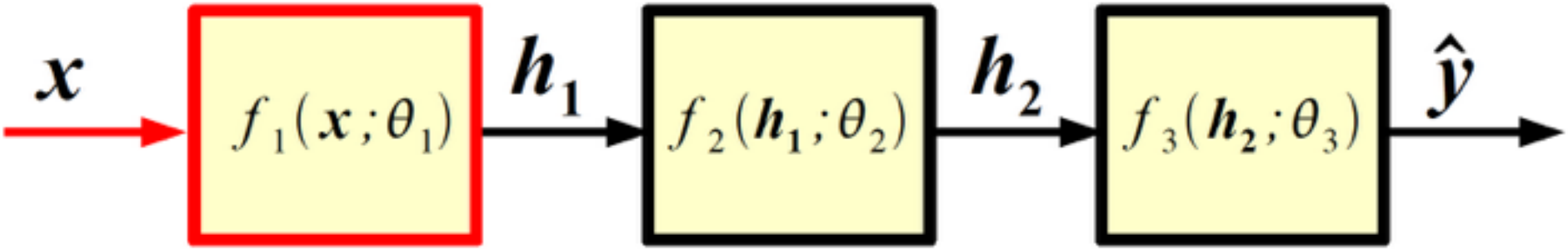
Neural Nets



NOTE: In practice, each module does NOT need to be a logistic regression classifier.

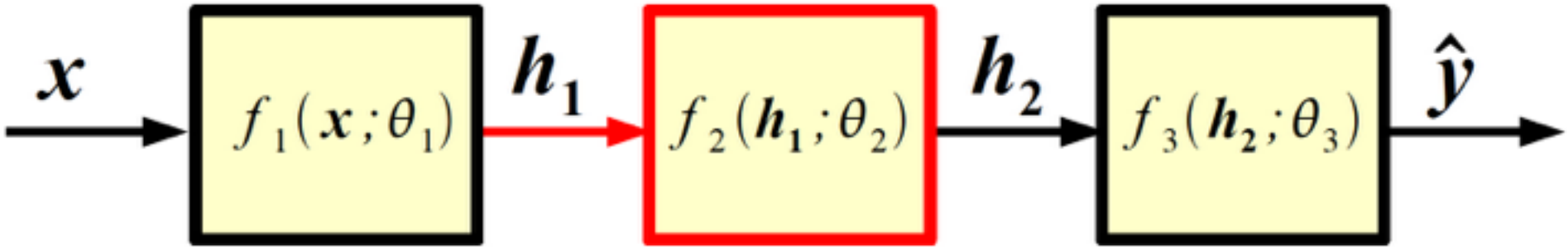
Any (a.e. differentiable) non-linear transformation is potentially good.

Forward Propagation (FPROP)



1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

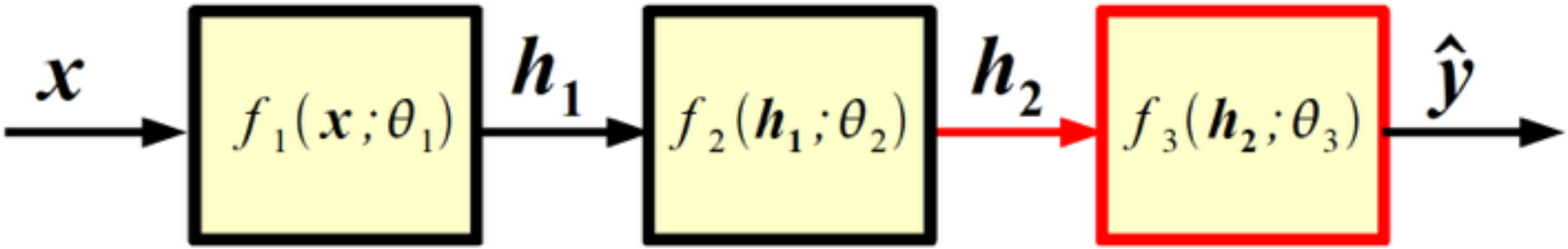
Forward Propagation (FPROP)



1) Given x compute: $h_1 = f_1(x;\theta_1)$

2) Given h_1 compute: $h_2 = f_2(h_1;\theta_2)$

Forward Propagation (FPROP)



1) Given $\boldsymbol{x}$ compute: $\boldsymbol{h_1} = f_1(\boldsymbol{x};\theta_1)$

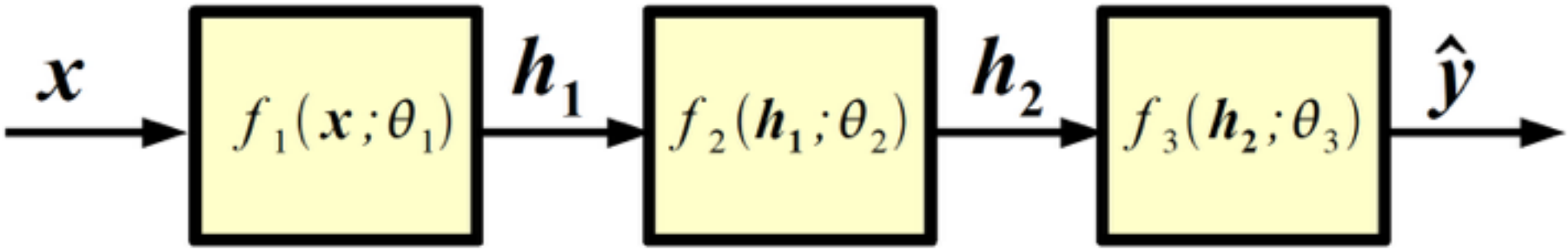
2) Given $\boldsymbol{h_1}$ compute: $\boldsymbol{h_2} = f_2(\boldsymbol{h_1};\theta_2)$

3) Given $\boldsymbol{h_2}$ compute: $\hat{\boldsymbol{y}} = f_3(\boldsymbol{h_2};\theta_3)$

For instance,

$$\hat{y}_i = p(class = i|\boldsymbol{x}) = \frac{e^{W_{3i}\boldsymbol{h}_2 + b_{3i}}}{\sum_k e^{W_{3k}\boldsymbol{h}_2 + b_{3k}}}$$

Forward Propagation (FPROP)



1) Given x compute: $h_1 = f_1(x;\theta_1)$

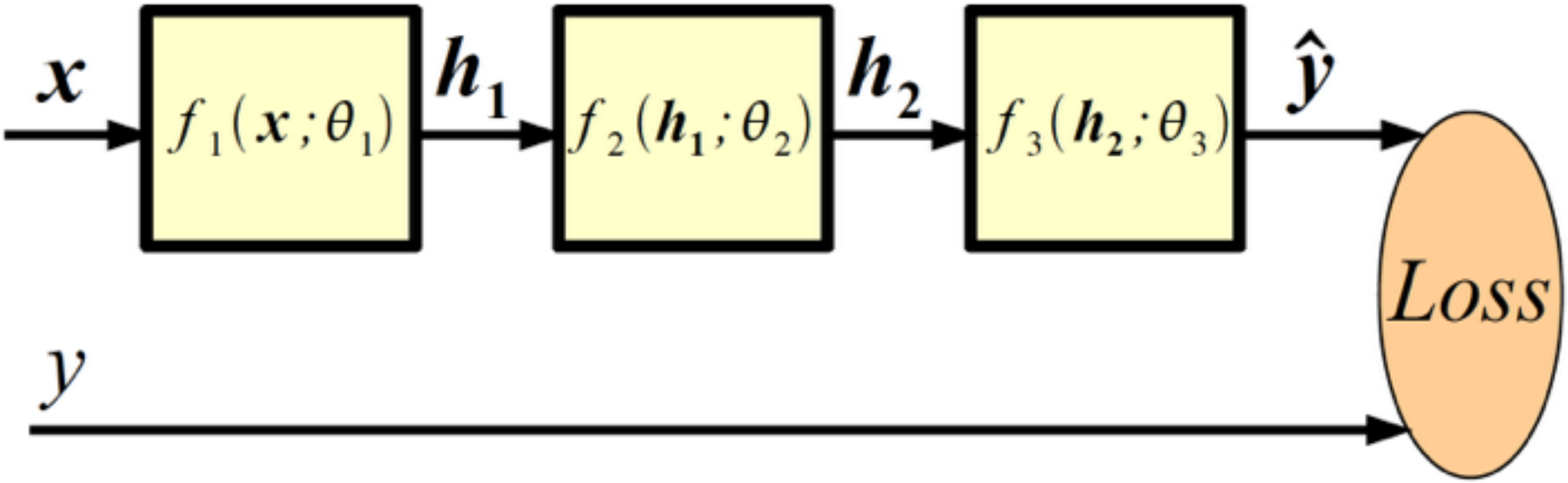
2) Given h_1 compute: $h_2 = f_2(h_1;\theta_2)$

3) Given h_2 compute: $\hat{y} = f_3(h_2;\theta_3)$

This is the typical processing at test time.

At training time, we need to compute an error measure and tune the parameters to decrease the error.

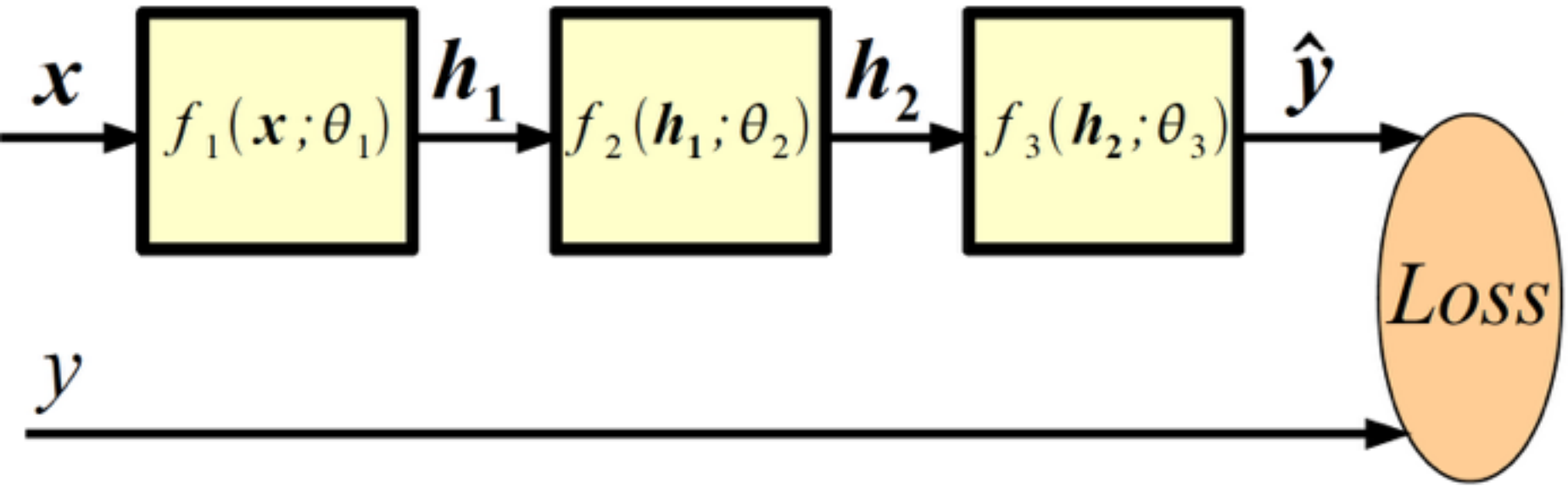
Loss



The measure of how well the model fits the training set is given by a suitable loss function: $L(\boldsymbol{x}, y; \boldsymbol{\theta})$

The loss depends on the input $\boldsymbol{x}$, the target label y, and the parameters $\boldsymbol{\theta}$.

Loss



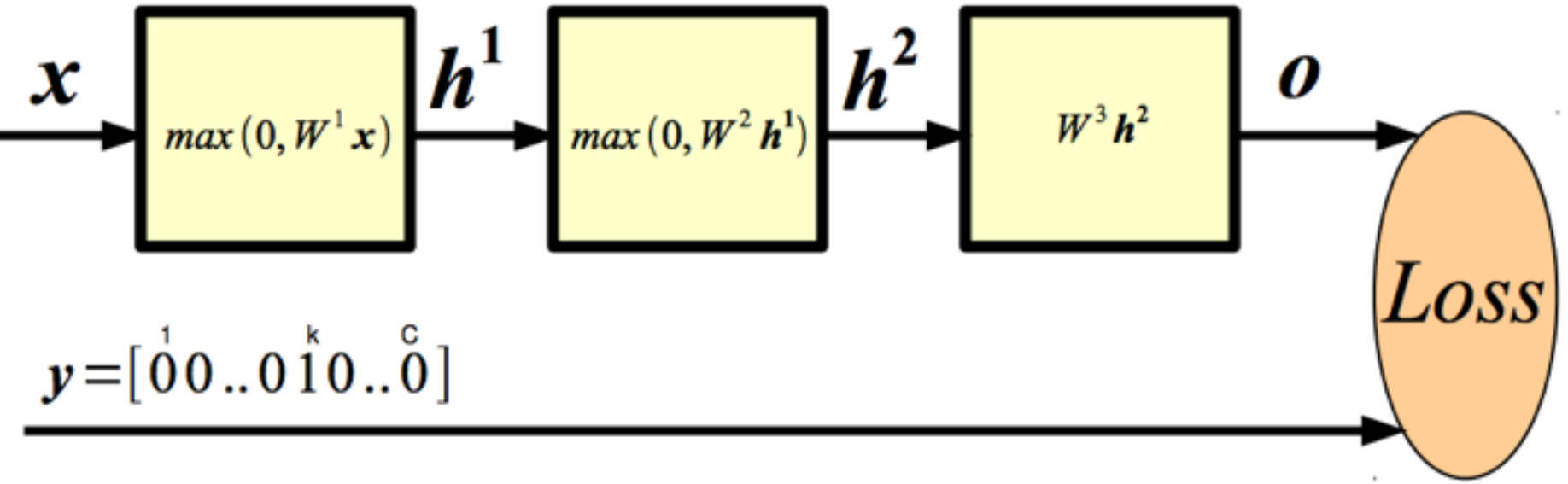
Q.: how to tune the parameters to decrease the loss?

If loss is (a.e.) differentiable we can compute gradients.

We can use chain-rule, a.k.a. **back-propagation**, to compute the gradients w.r.t. parameters at the lower layers.

Rumelhart et al. "Learning internal representations by back-propagating.." Nature 1986

How Good is a Network?



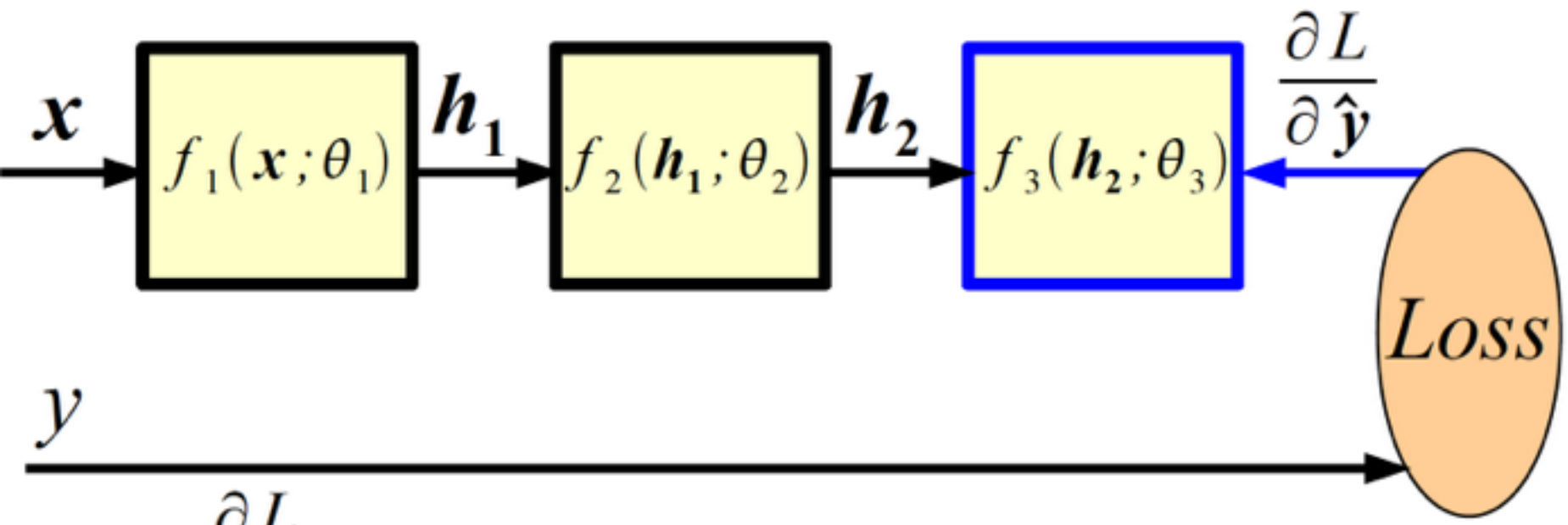
Probability of class k given input (softmax):

$$p(c_k = 1 | \boldsymbol{x}) = \frac{e^{o_k}}{\sum_{j=1}^{C} e^{o_j}}$$

(Per-sample) **Loss**; e.g., negative log-likelihood (good for classification of small number of classes):

$$L(\boldsymbol{x}, y; \boldsymbol{\theta}) = -\sum_j y_j \log p(c_j | \boldsymbol{x})$$

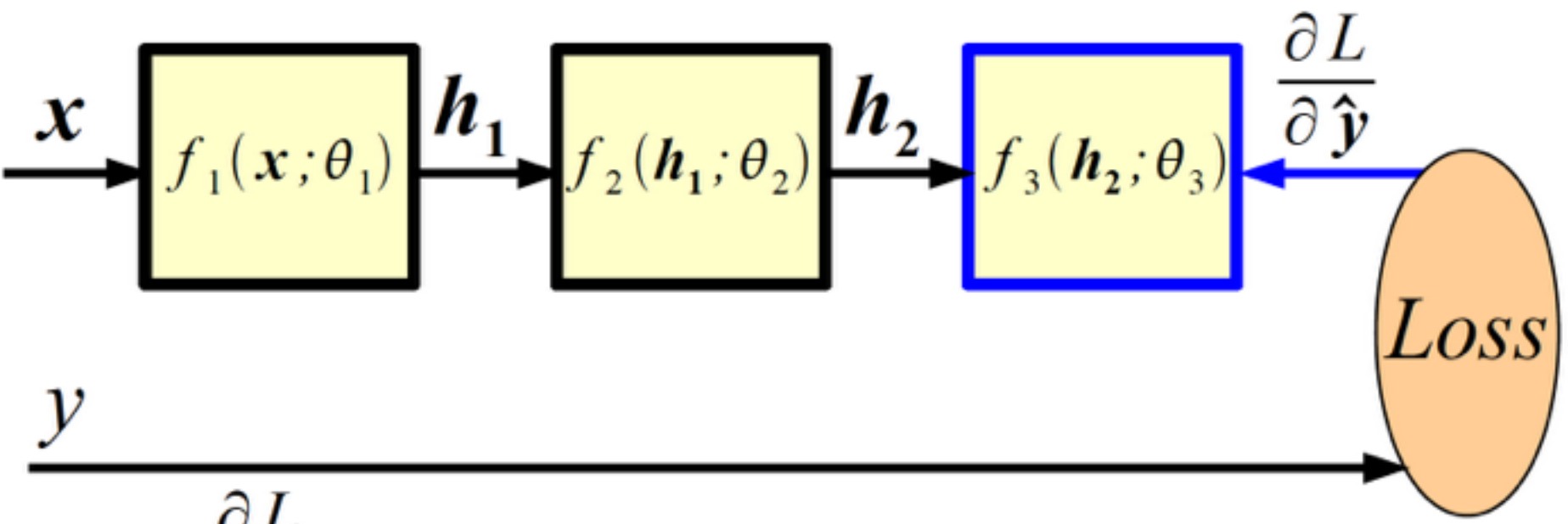
Backward Propagation (BPROP)



Given $\frac{\partial L}{\partial \hat{\mathbf{y}}}$ and assumiing the Jacobian of each module is easy to compute, then we have:

$$\frac{\partial L}{\partial \theta_3} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \theta_3} \qquad \frac{\partial L}{\partial \boldsymbol{h}_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{h}_2}$$

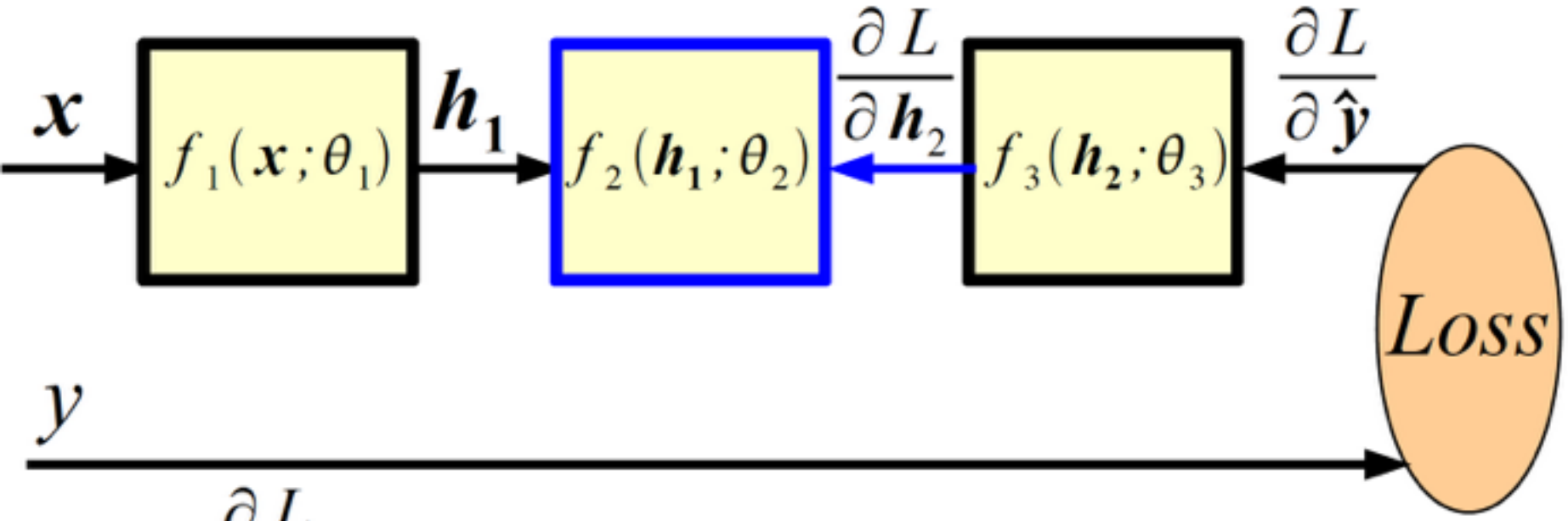
Backward Propagation (BPROP)



Given $\frac{\partial L}{\partial \hat{y}}$ and assumiing the Jacobian of each module is easy to compute, then we have:

$$\frac{\partial L}{\partial \theta_3} = (\hat{y} - y)\ h_2{}' \qquad \frac{\partial L}{\partial h_2} = (\hat{y} - y)\ \theta_3{}'$$

Backward Propagation (BPROP)

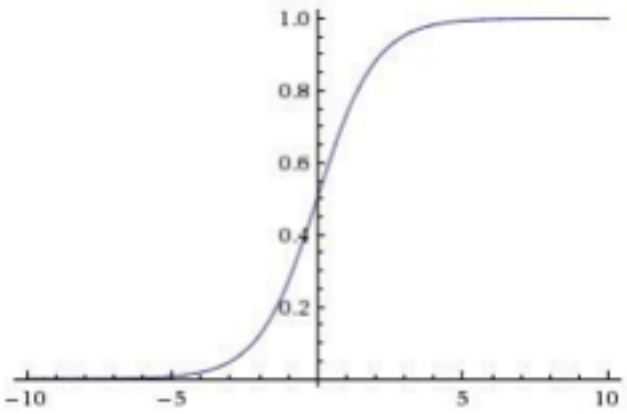


Given $\frac{\partial L}{\partial \boldsymbol{h}_2}$ we can compute now:

$$\frac{\partial L}{\partial \theta_2} = \frac{\partial L}{\partial \boldsymbol{h}_2}\frac{\partial \boldsymbol{h}_2}{\partial \theta_2} \qquad \frac{\partial L}{\partial \boldsymbol{h}_1} = \frac{\partial L}{\partial \boldsymbol{h}_2}\frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{h}_1}$$

Activation Functions

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



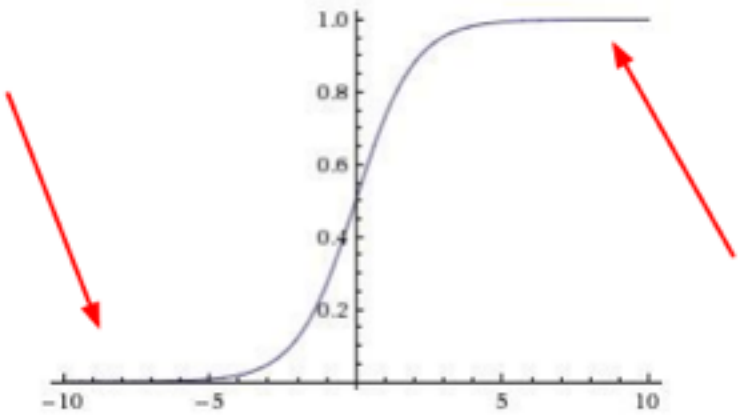
Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

2 BIG problems:

Activation Functions

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



Sigmoid

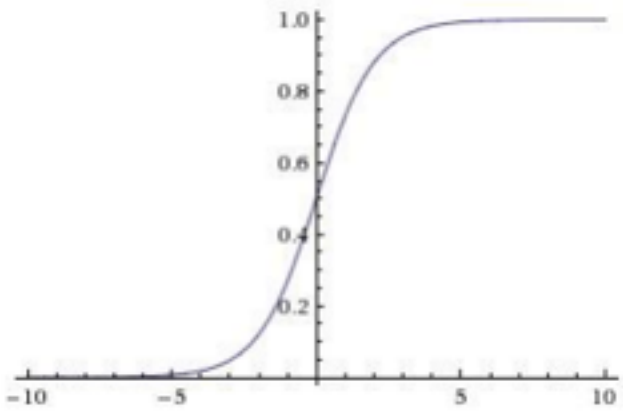
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

2 BIG problems:

1. Saturated neurons “kill” the gradients

Activation Functions

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



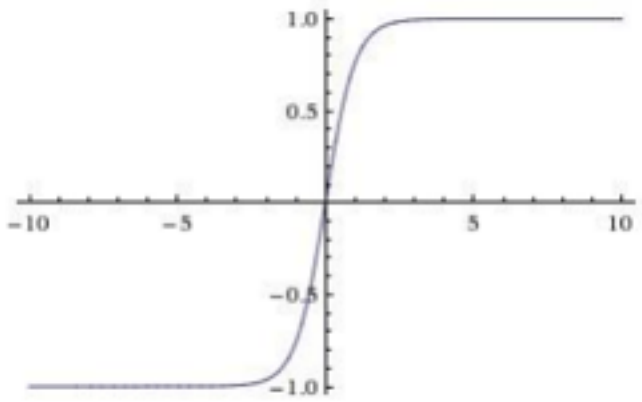
Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

2 BIG problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered

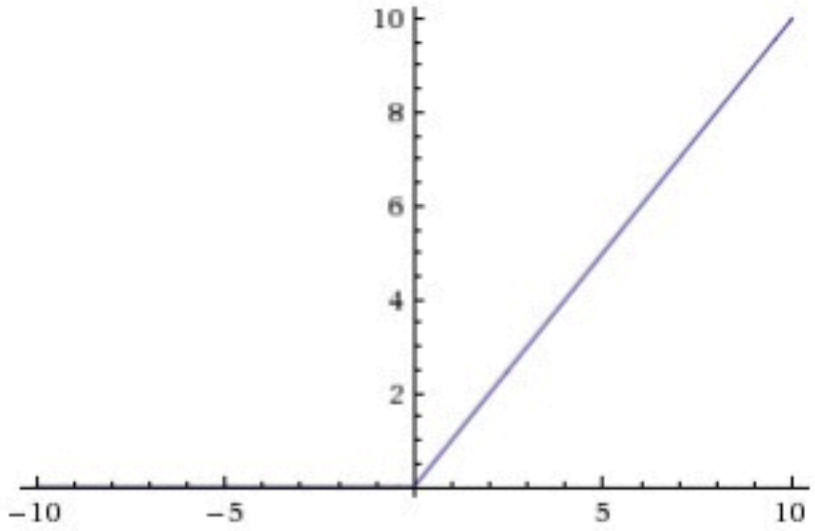
Activation Functions



tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

Activation Functions



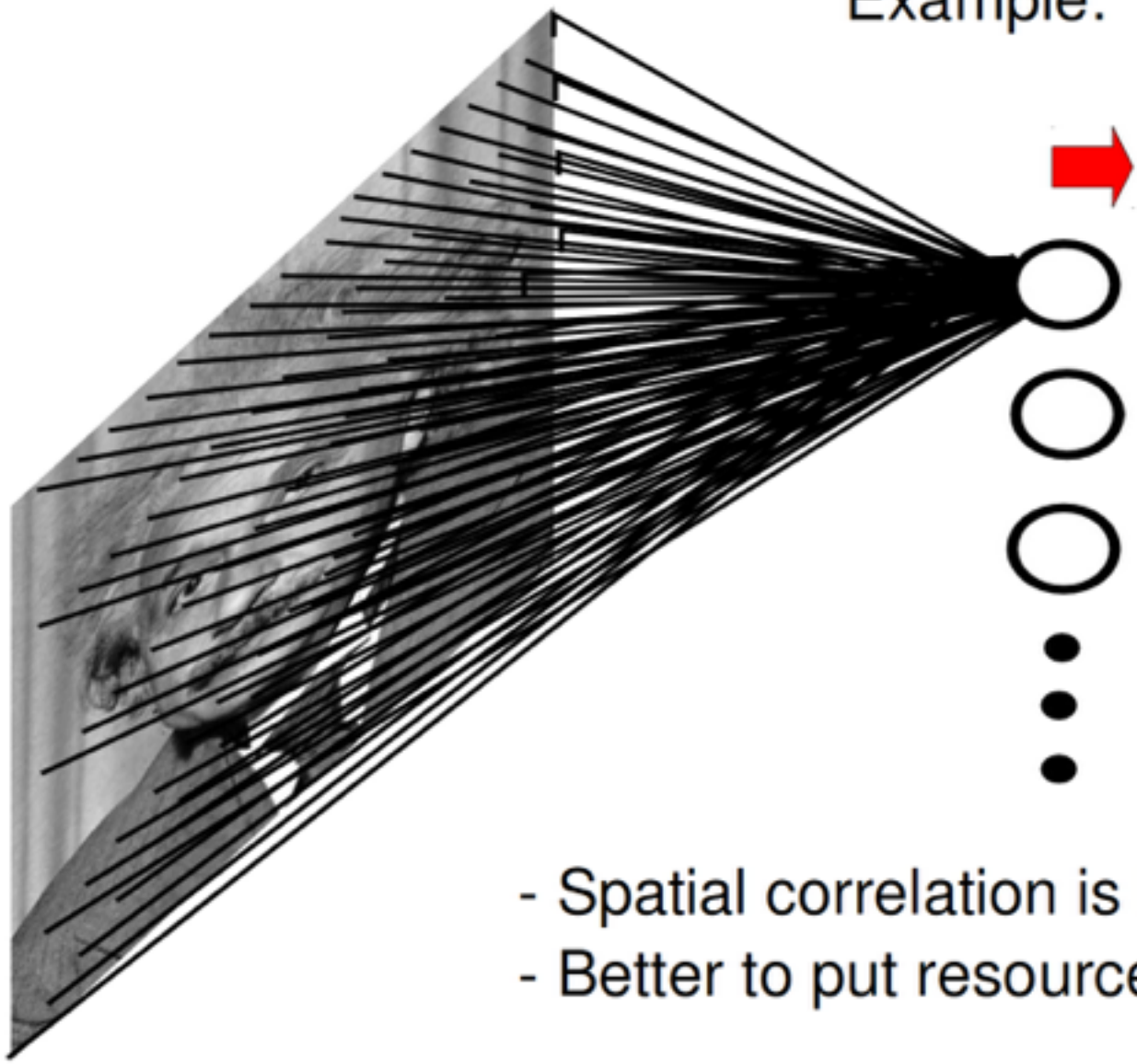
ReLU

FULLY CONNECTED NEURAL NET

Example: 1000x1000 image

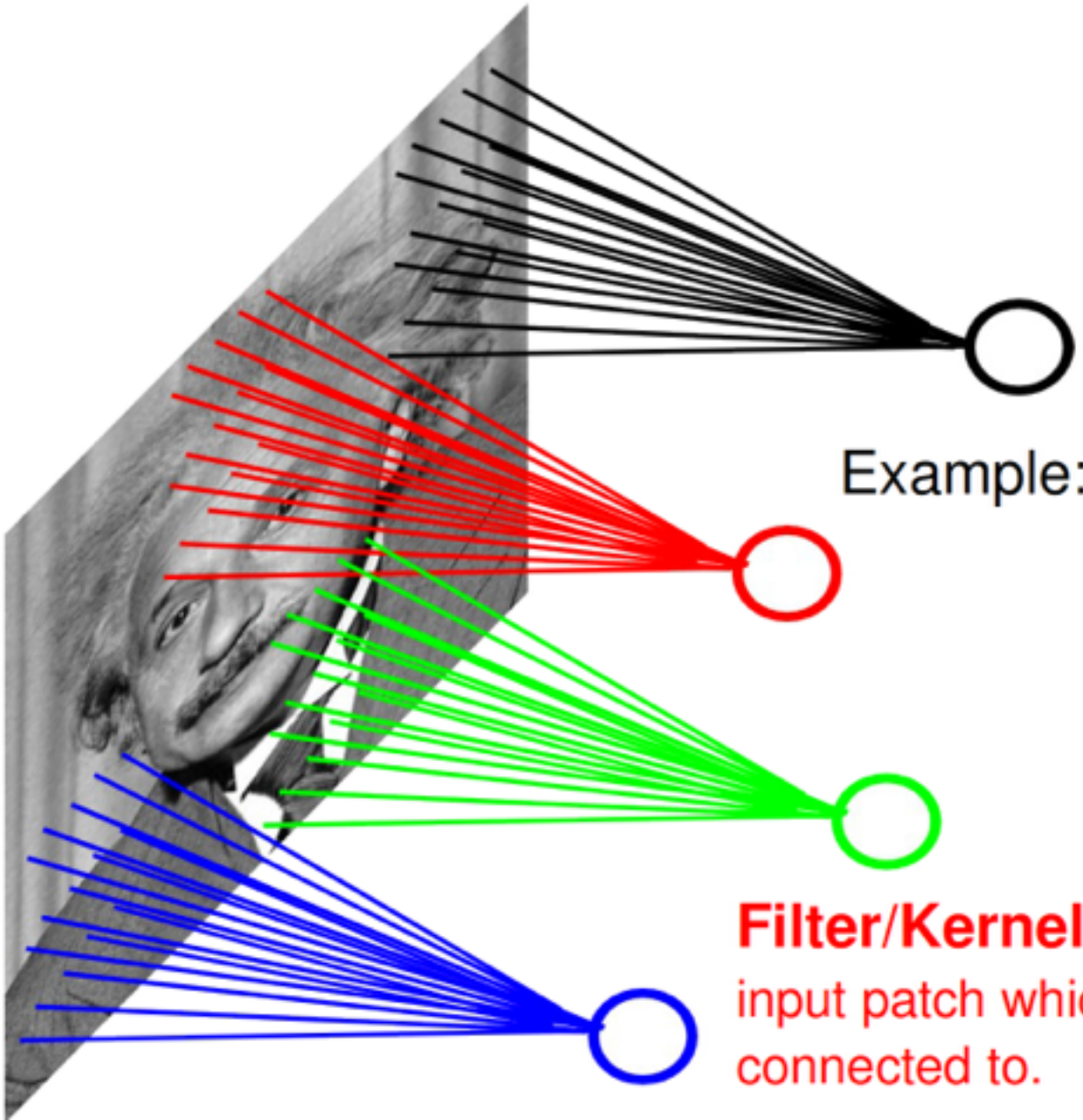
1M hidden units

10^12 parameters!!!



- Spatial correlation is local
- Better to put resources elsewhere!

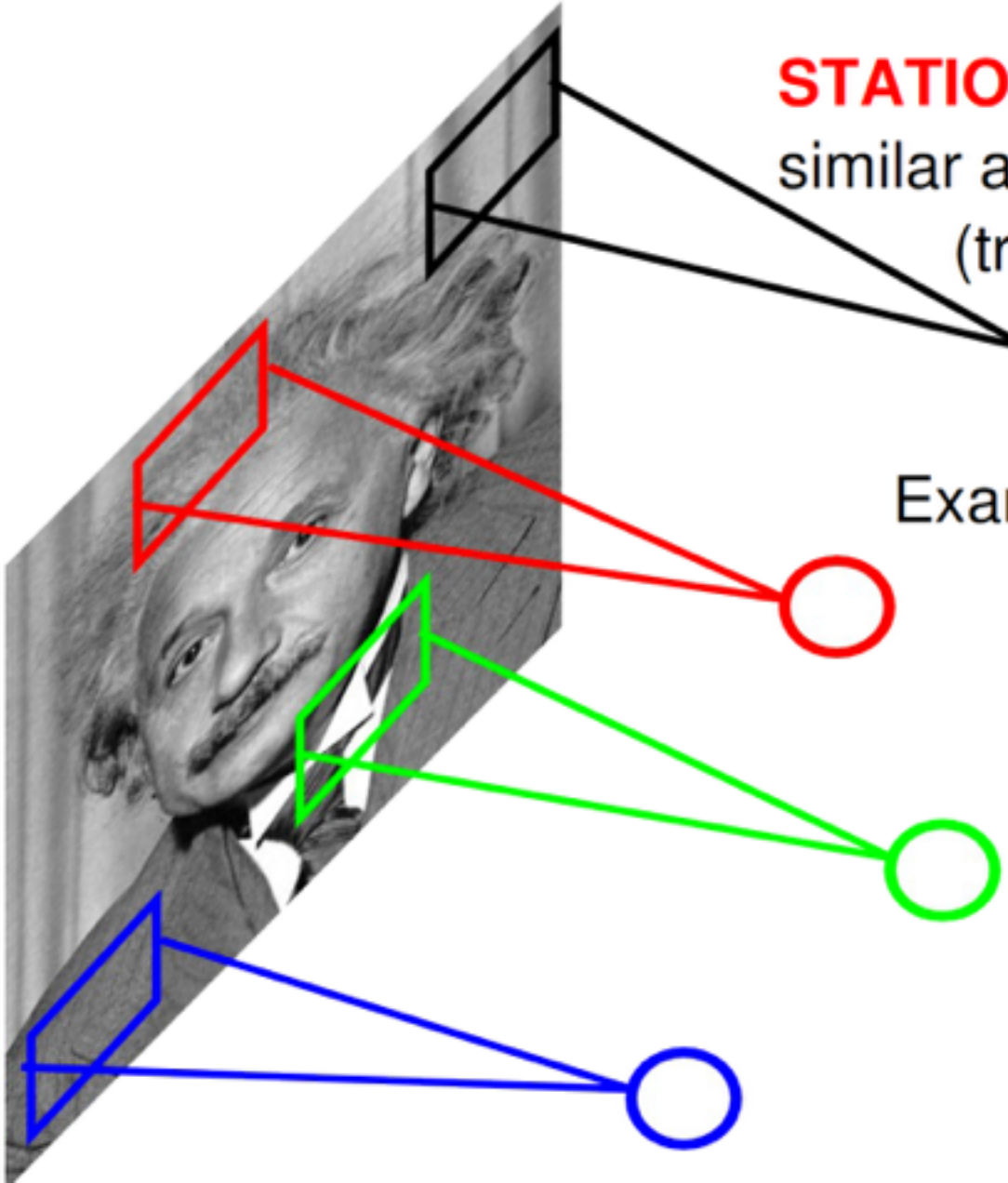
LOCALLY CONNECTED NEURAL NET



Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

Filter/Kernel/Receptive field:
input patch which the hidden unit is connected to.

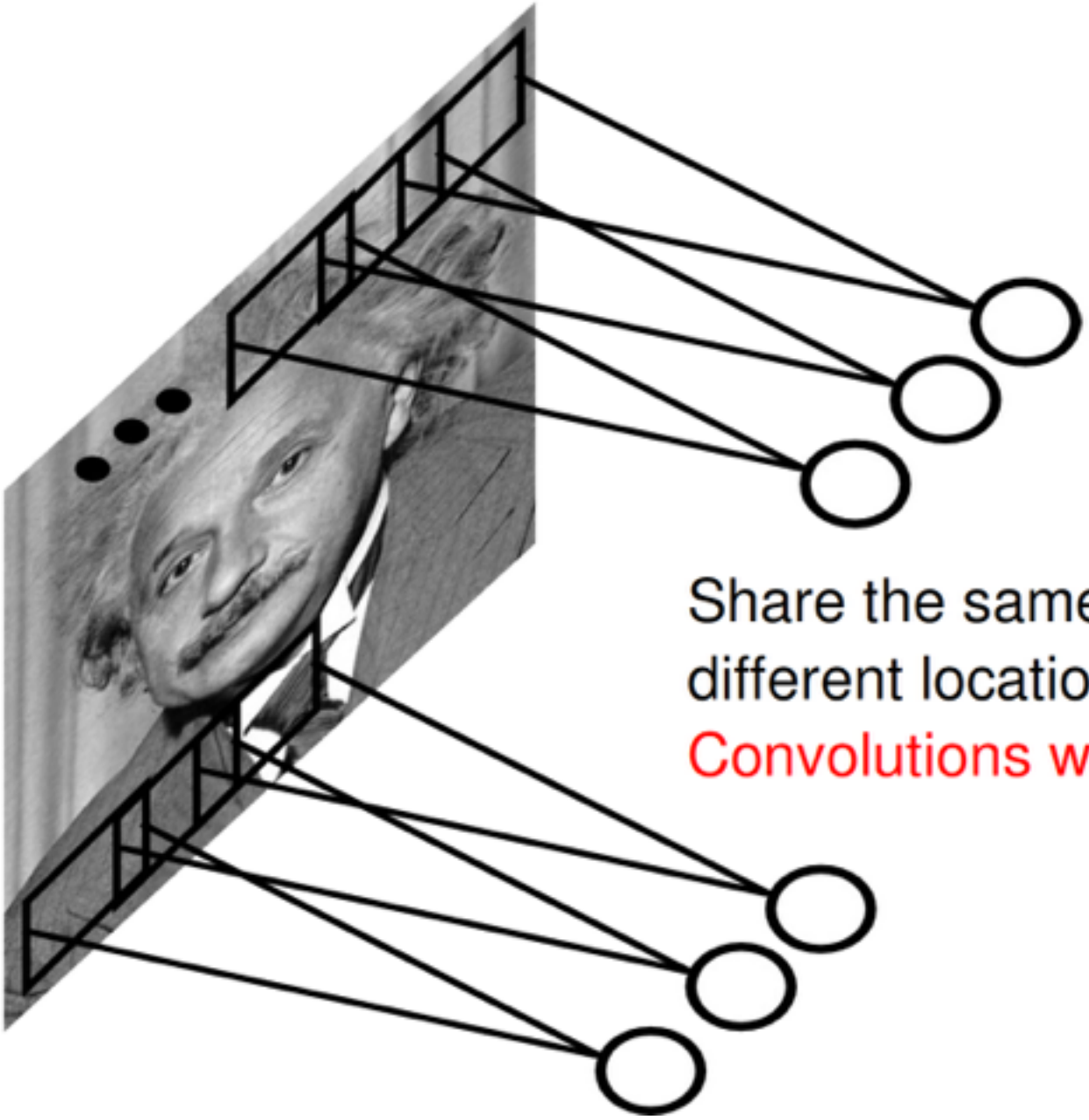
LOCALLY CONNECTED NEURAL NET



STATIONARITY? Statistics are similar at different locations (translation invariance)

Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

CONVOLUTIONAL NET



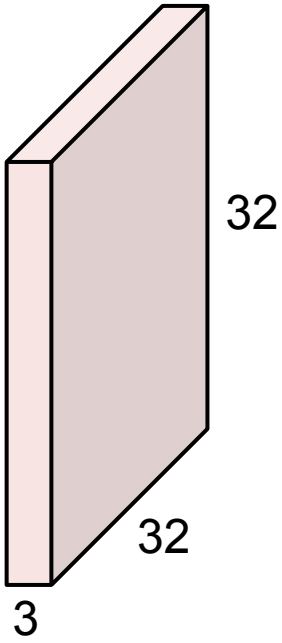
Share the same parameters across different locations:

Convolutions with learned kernels

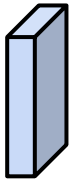
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

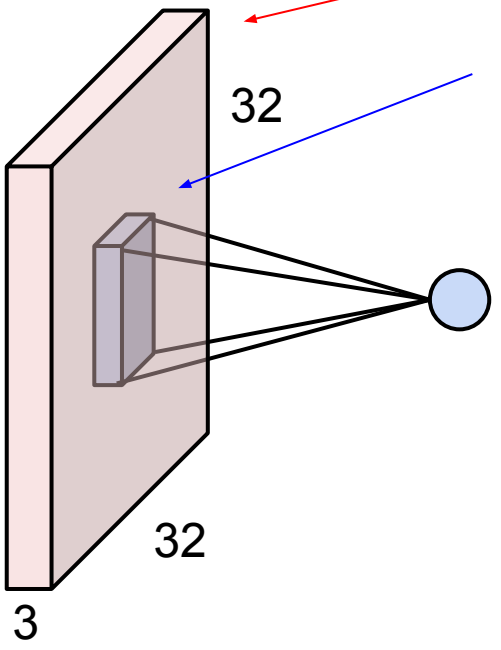


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



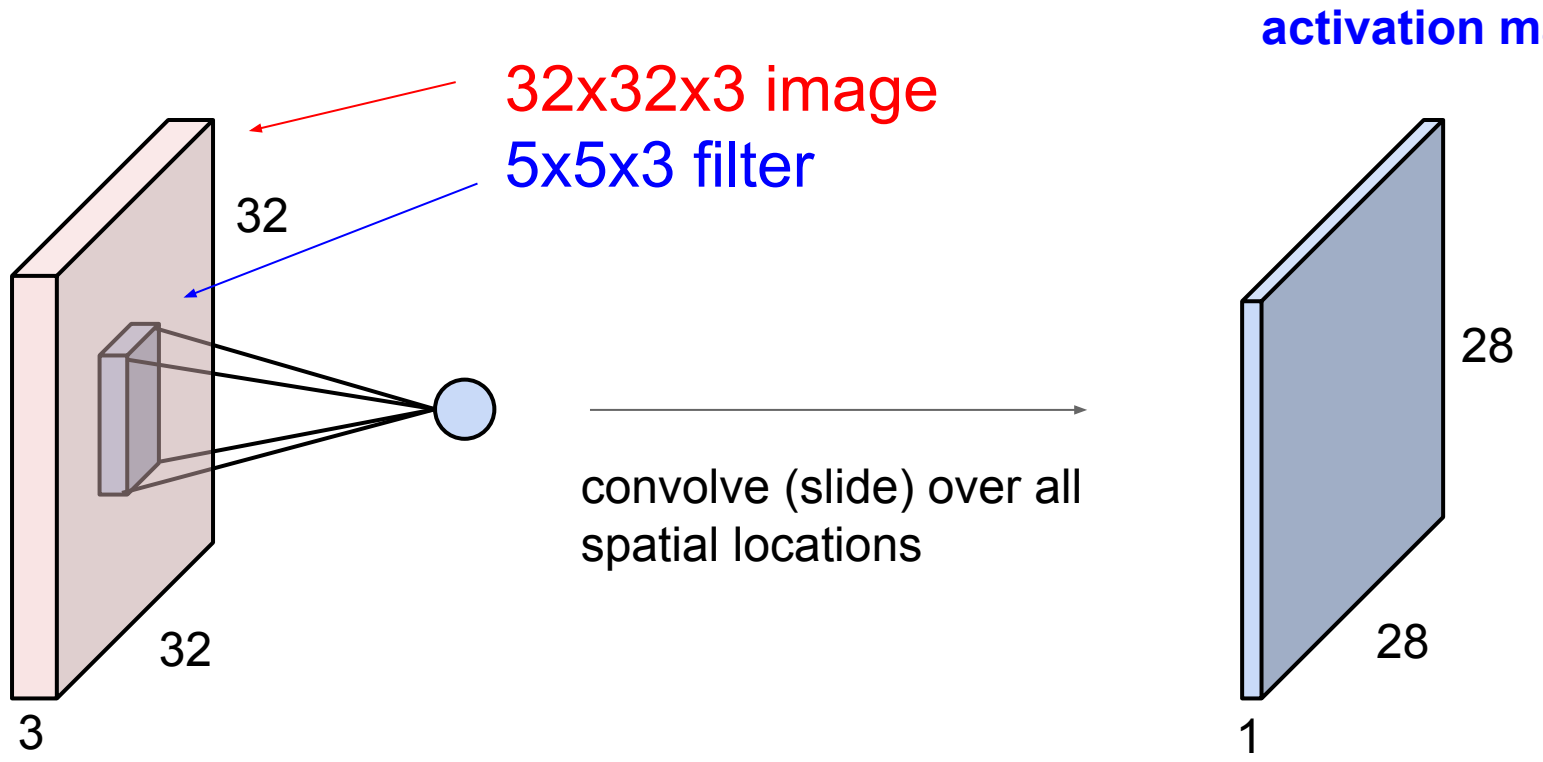
32x32x3 image

5x5x3 filter w

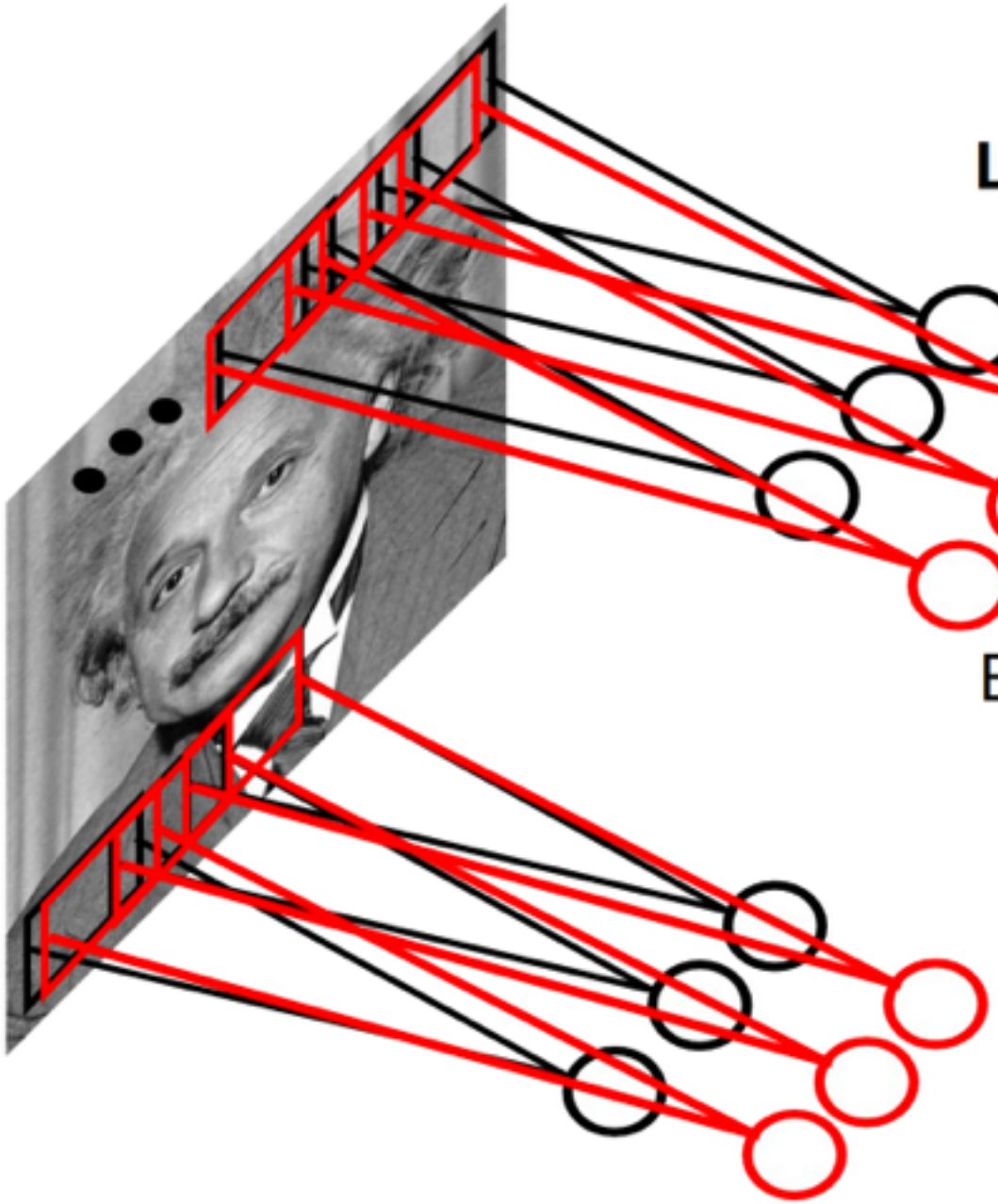
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer



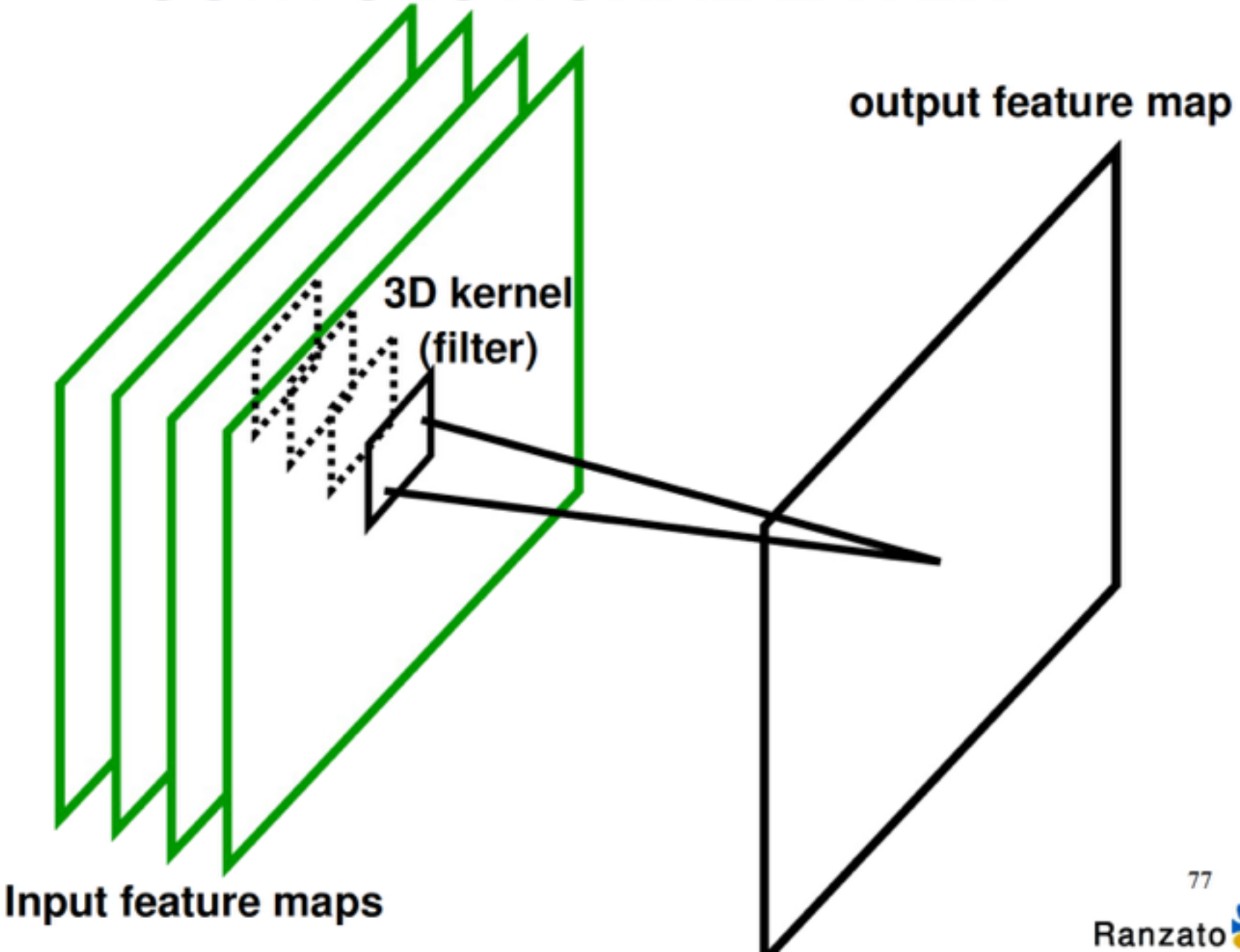
CONVOLUTIONAL NET



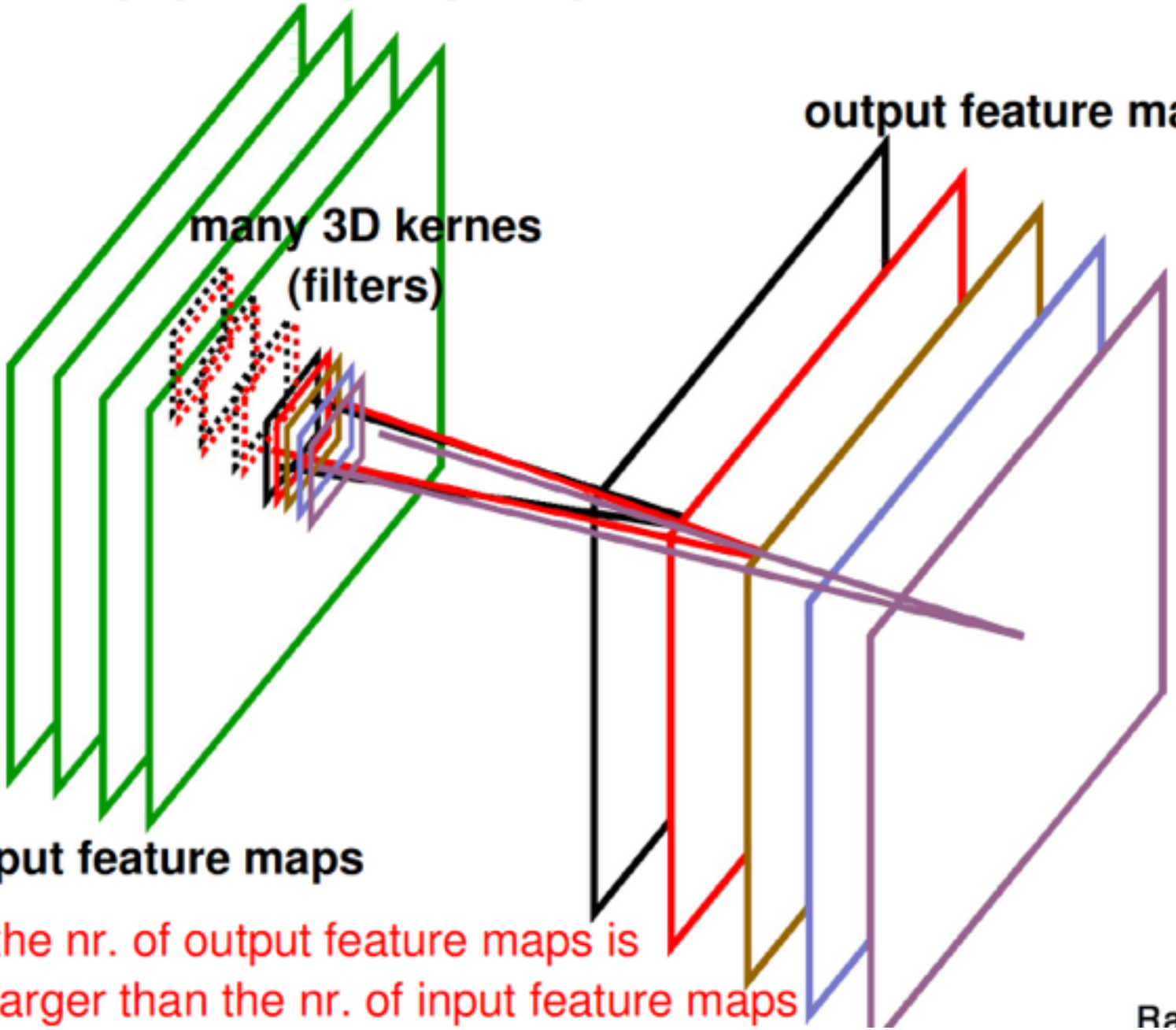
Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

CONVOLUTIONAL LAYER

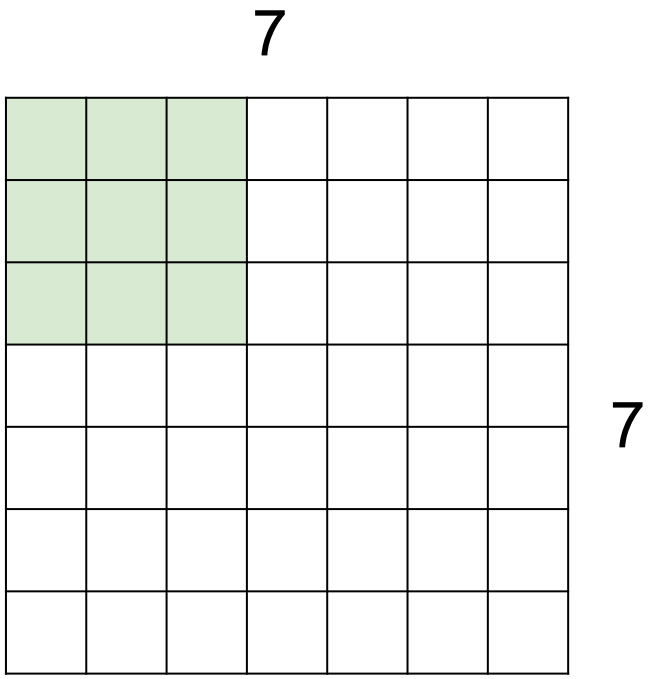


CONVOLUTIONAL LAYER



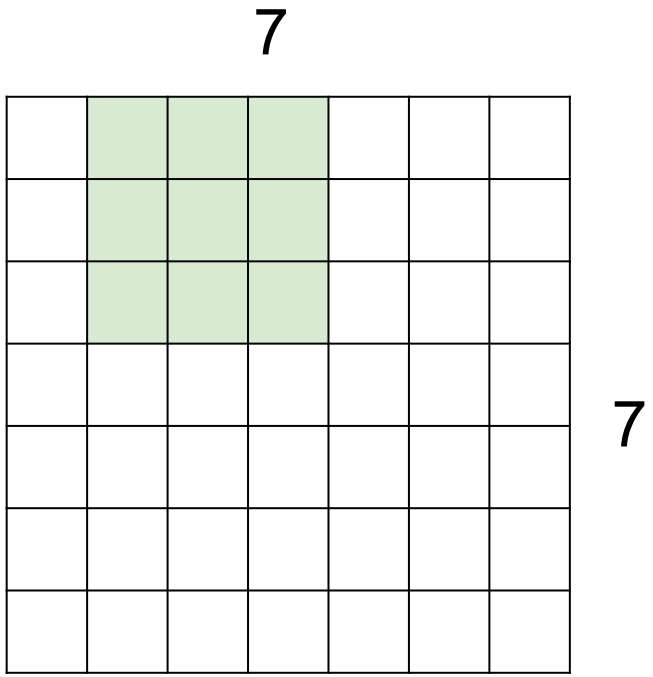
NOTE: the nr. of output feature maps is usually larger than the nr. of input feature maps

Closer look at Spatial Dimensions



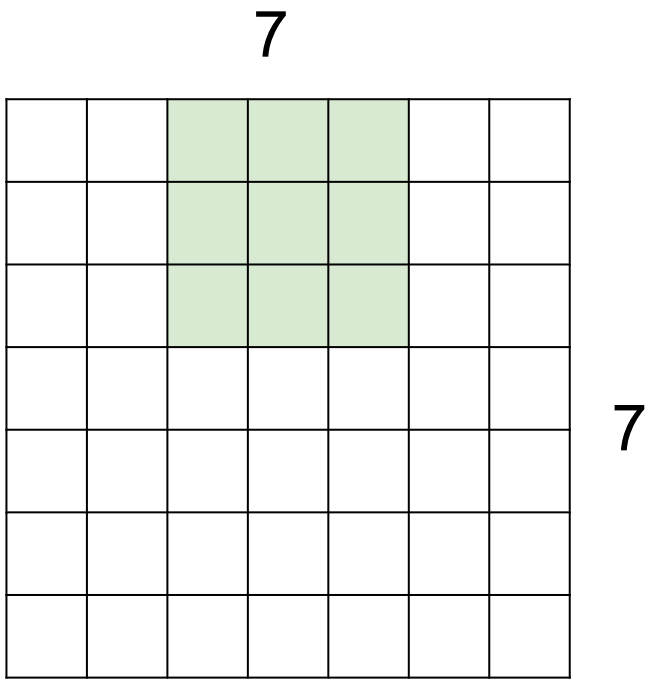
7x7 input (spatially)
assume 3x3 filter

Closer look at Spatial Dimensions



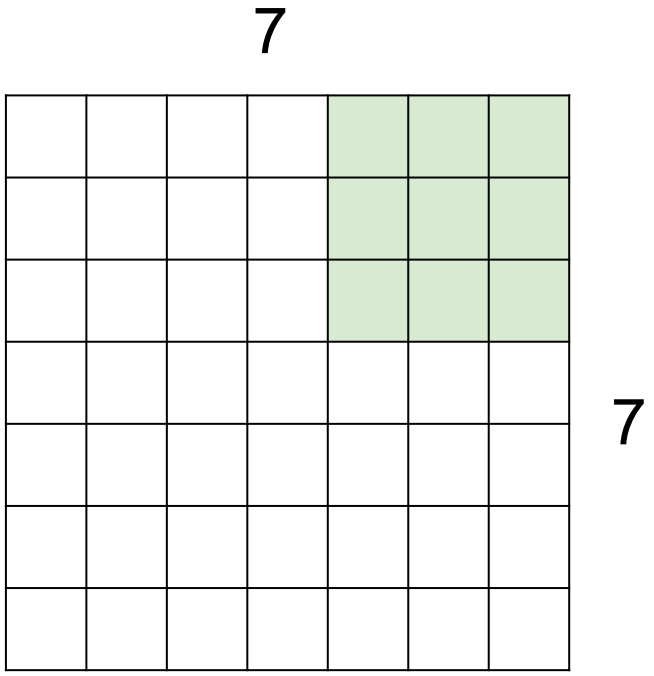
7x7 input (spatially)
assume 3x3 filter

Closer look at Spatial Dimensions



7x7 input (spatially)
assume 3x3 filter

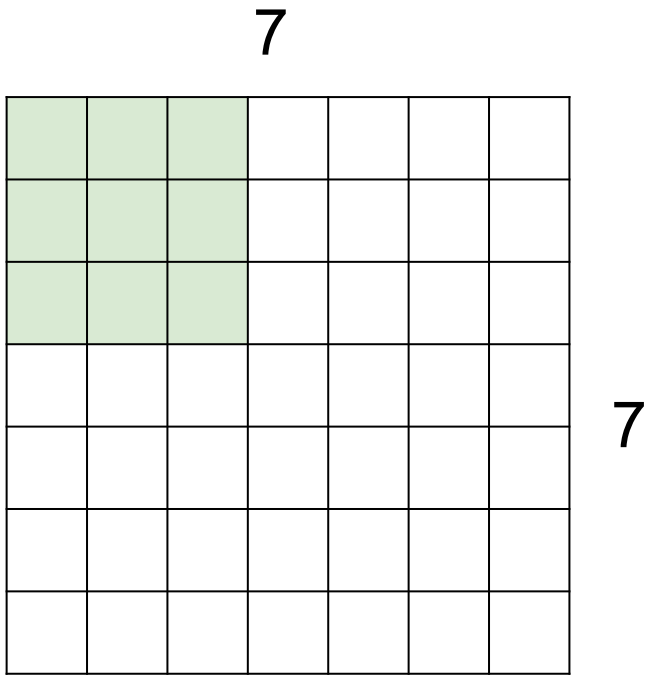
Closer look at Spatial Dimensions



7x7 input (spatially)
assume 3x3 filter

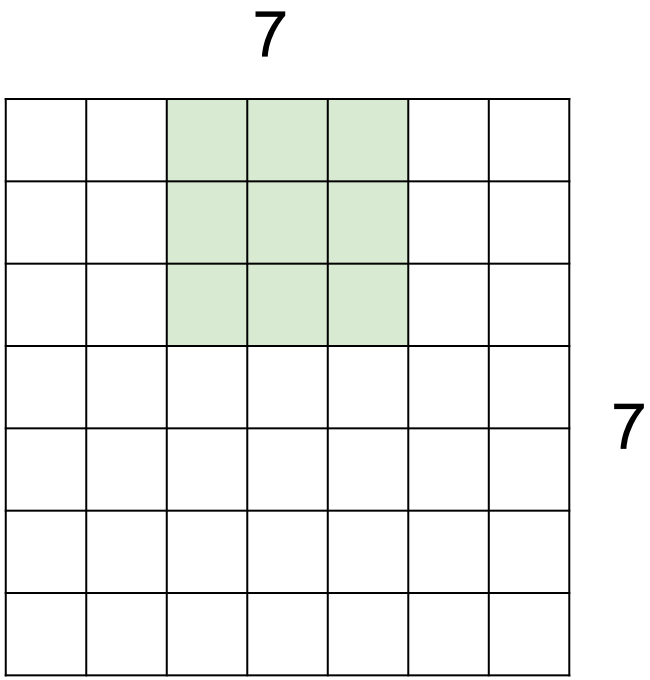
=> 5x5 output

Closer look at Spatial Dimensions



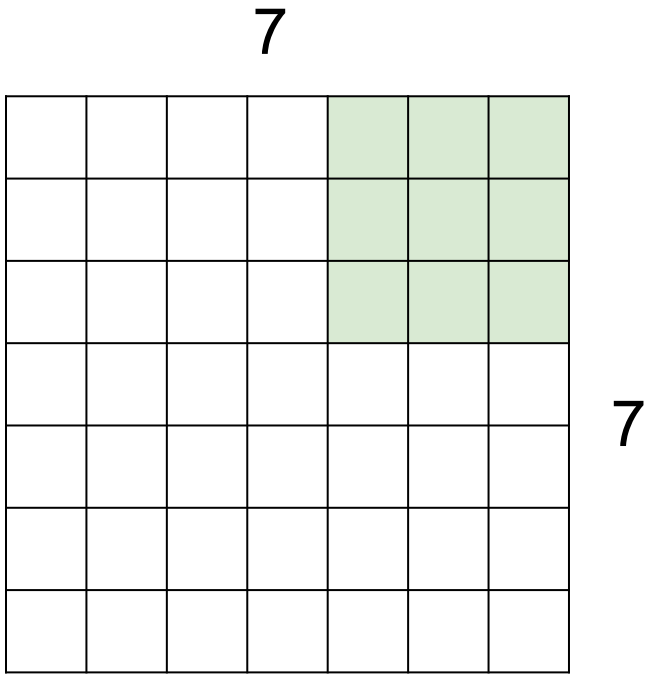
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Closer look at Spatial Dimensions



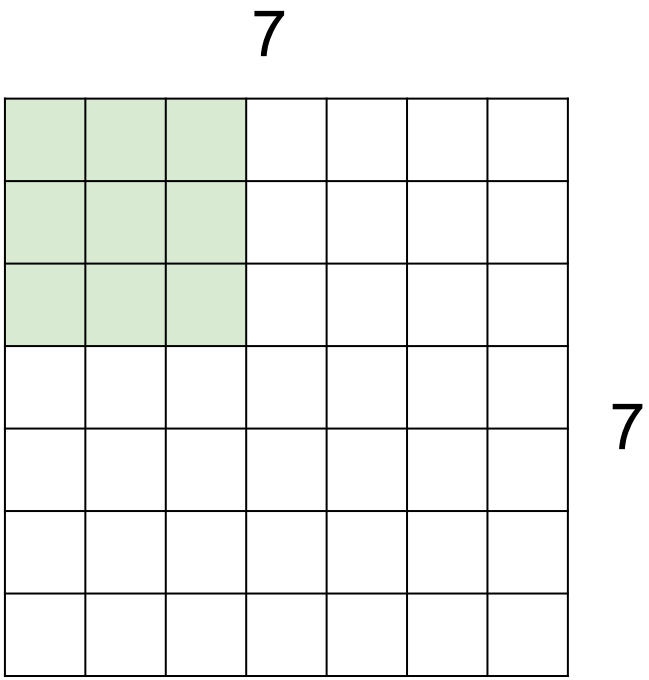
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Closer look at Spatial Dimensions



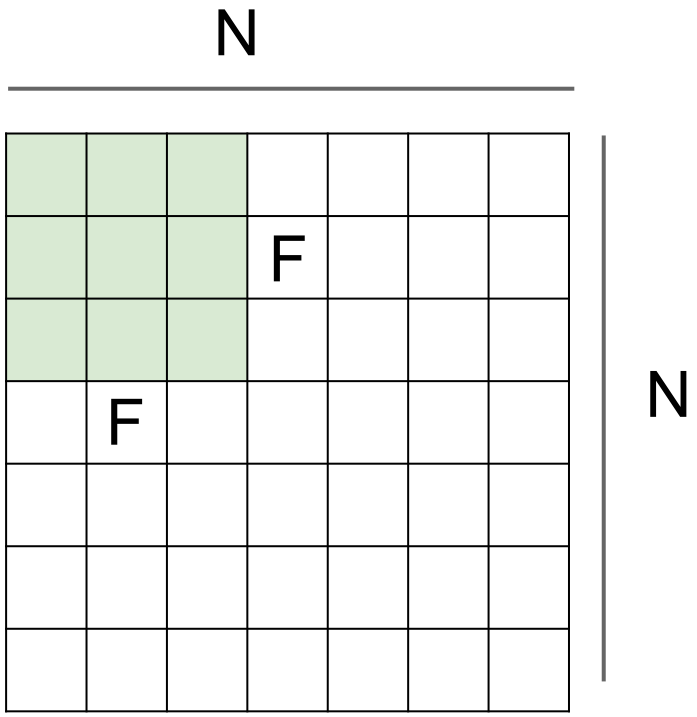
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Closer look at Spatial Dimensions



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

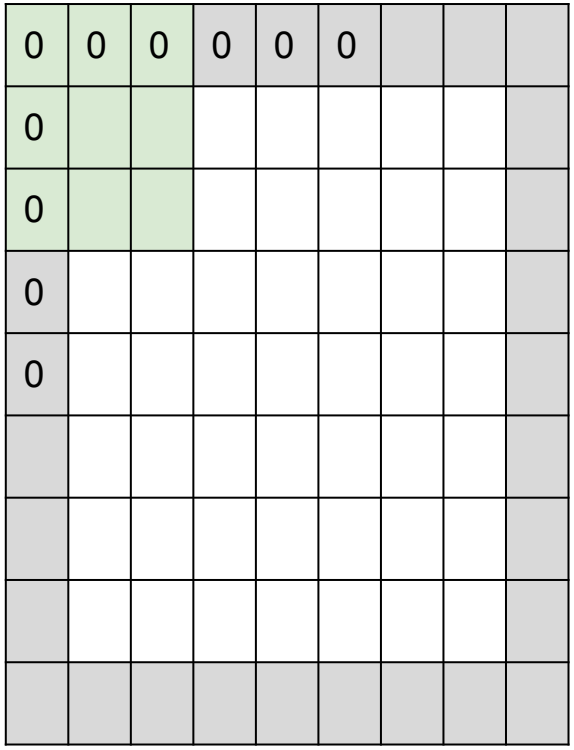
doesn't fit!
cannot apply 3x3 filter on 7x7 input with stride 3.



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

In practice: Common to zero pad the border



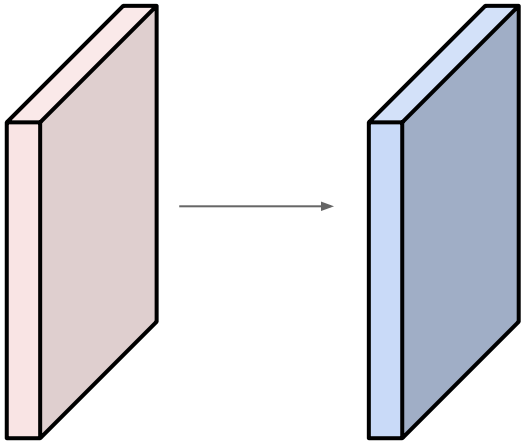
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

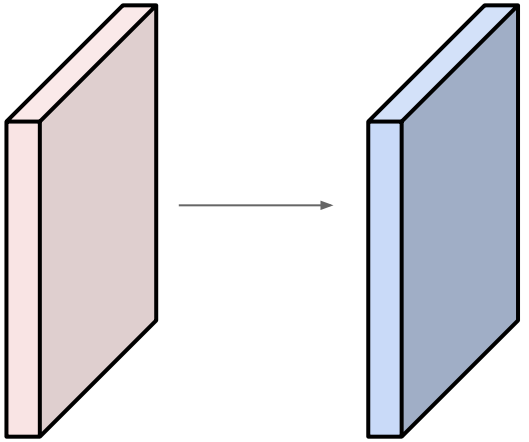
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

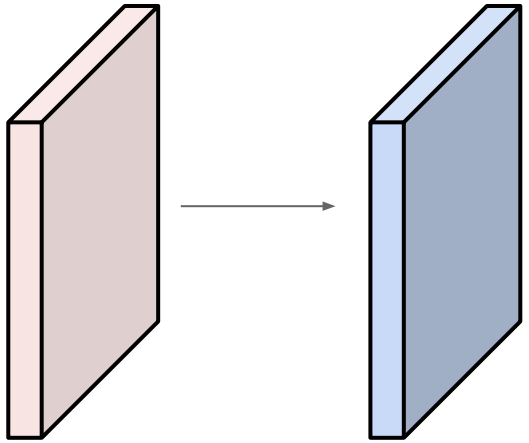
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

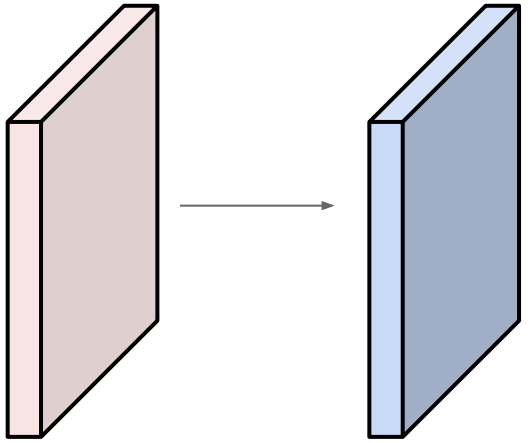
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Examples time:

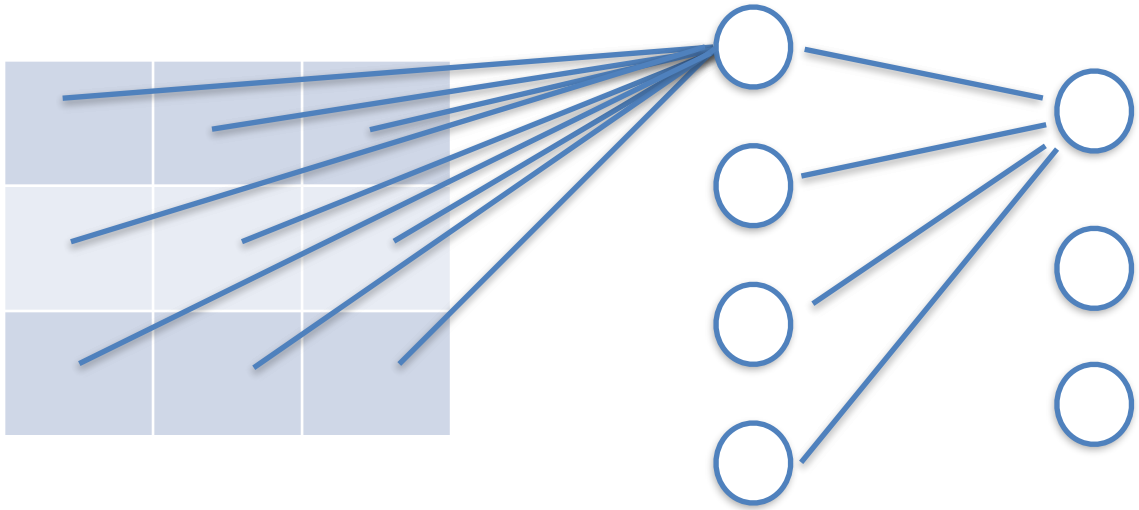


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Fully Connected Layer

Connects to entire input volume similar to DNN



Fully Connected Layer

Number of parameters:

10x10x10 connected to 1000 FC

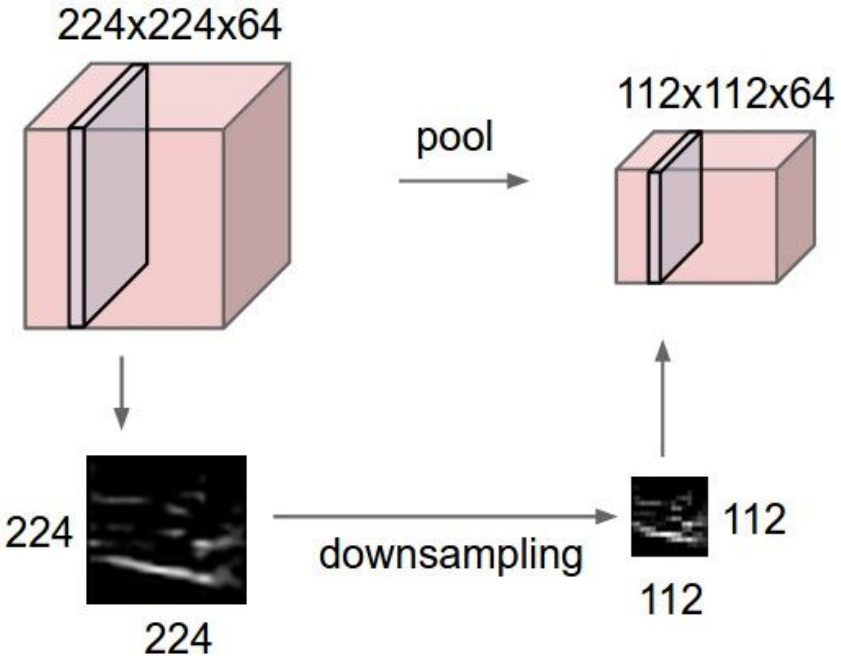
1000 FC to 1000 FC.

Special Layers

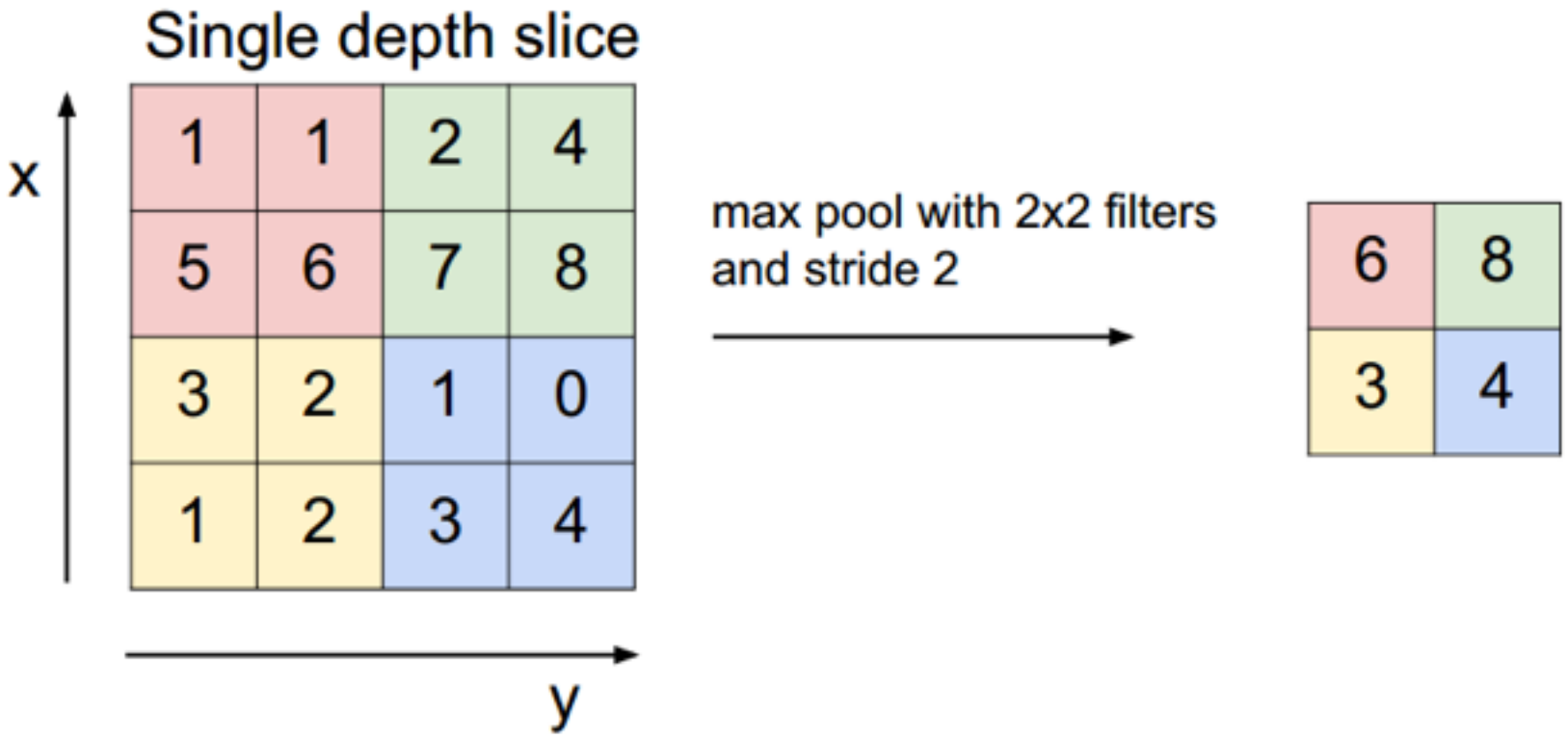
- Pooling
- Contrast Normalization

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

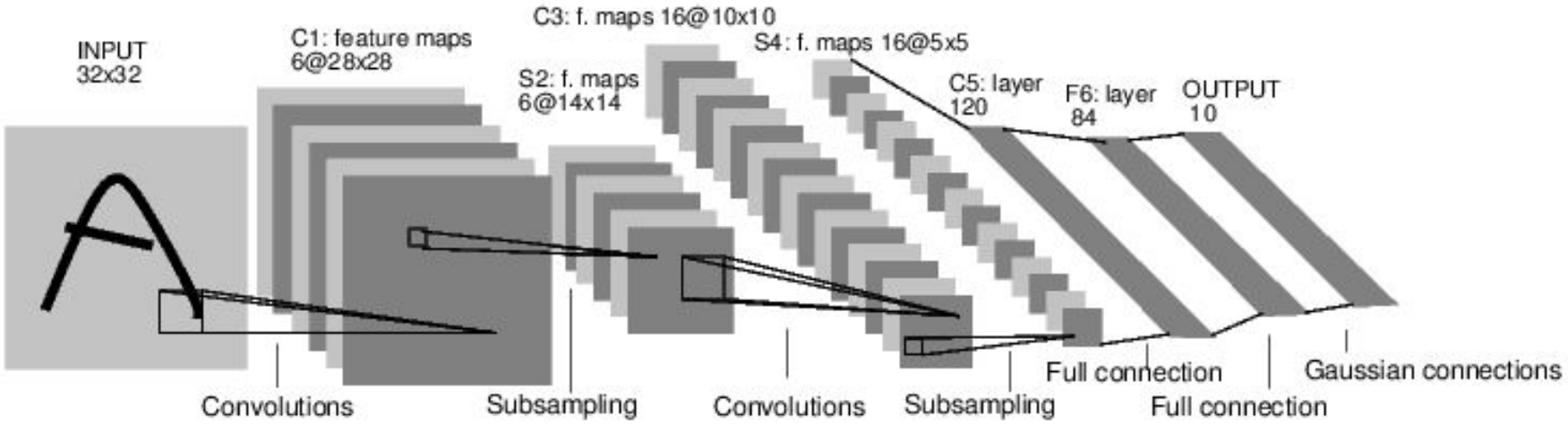
Local Response Normalization

$$b_{x,y}^{i} = a_{x,y}^{i} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^{j})^2 \right)^{\beta}$$

Example Architectures

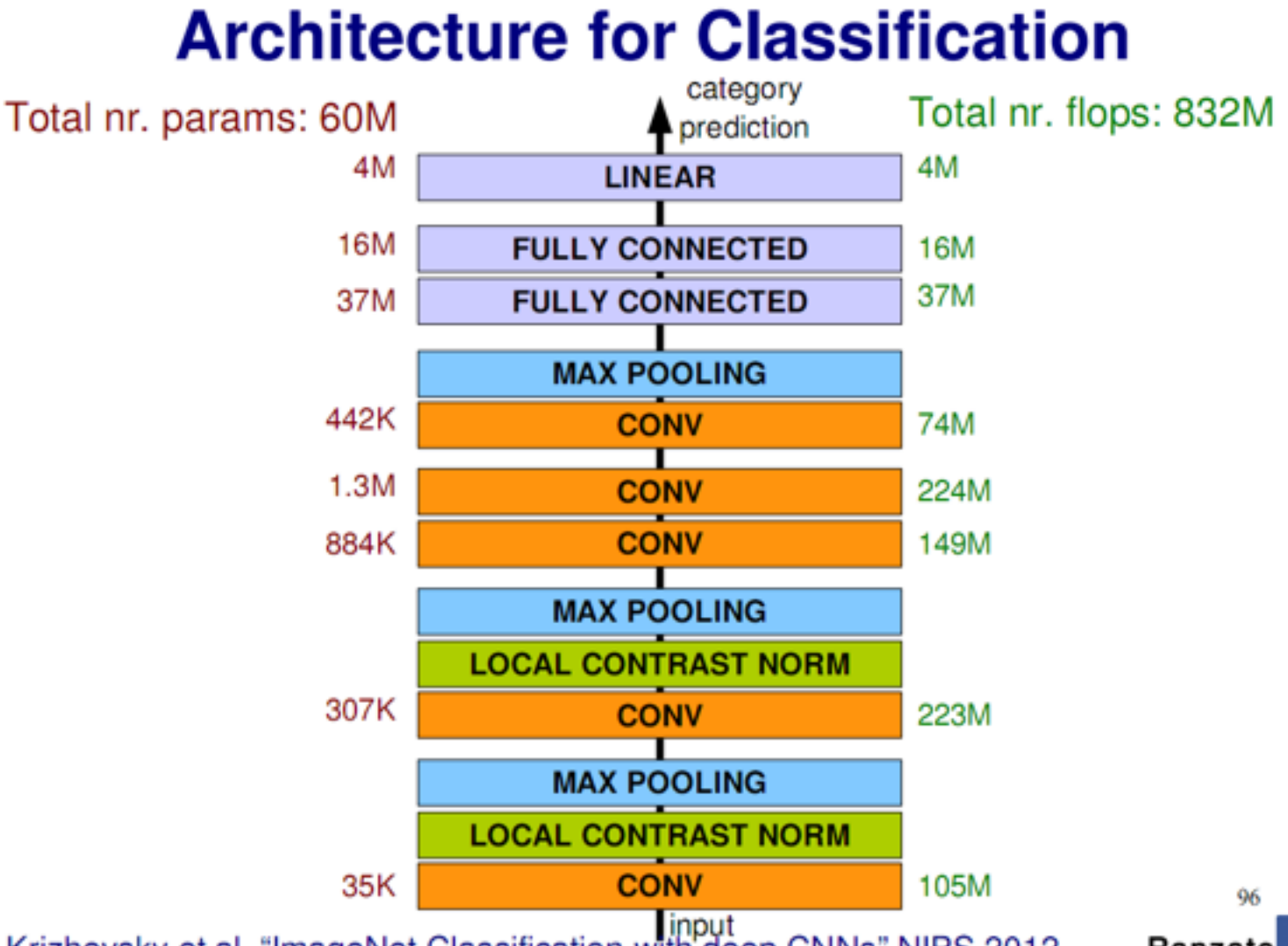
Case Study: LeNet-5

[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

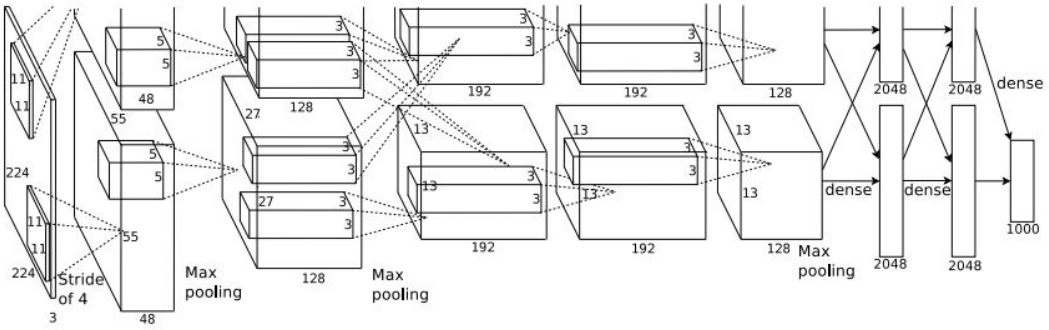
Imagenet Network



Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

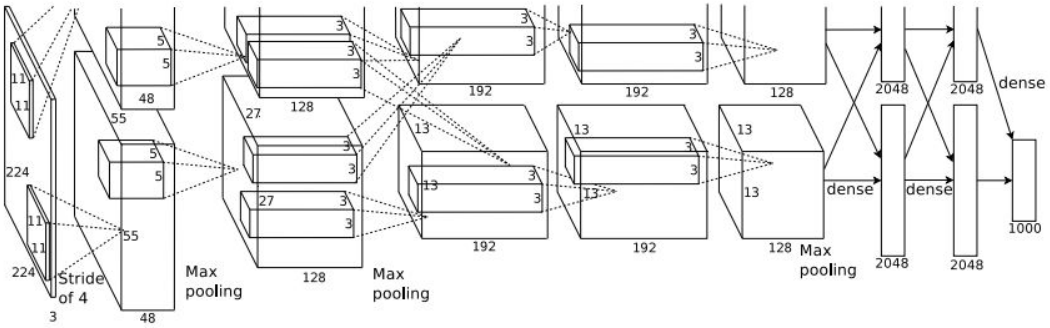
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study: AlexNet

[Krizhevsky et al. 2012]



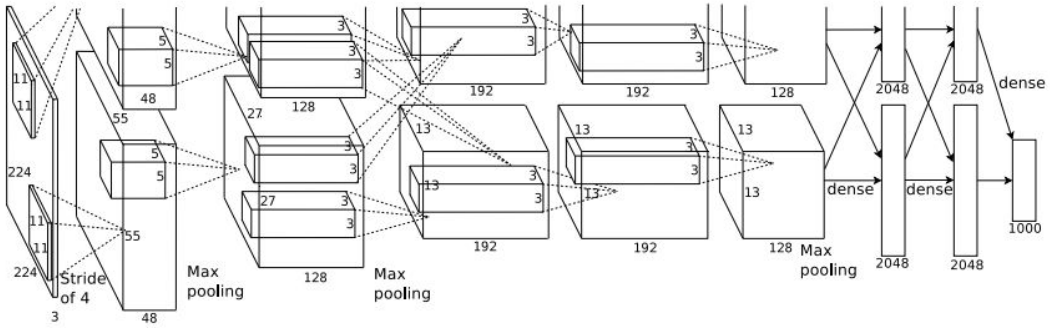
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]

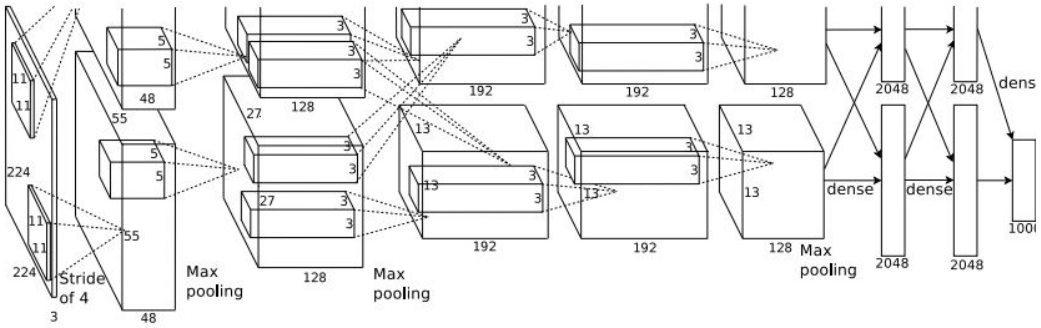


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Case Study: AlexNet

[Krizhevsky et al. 2012]



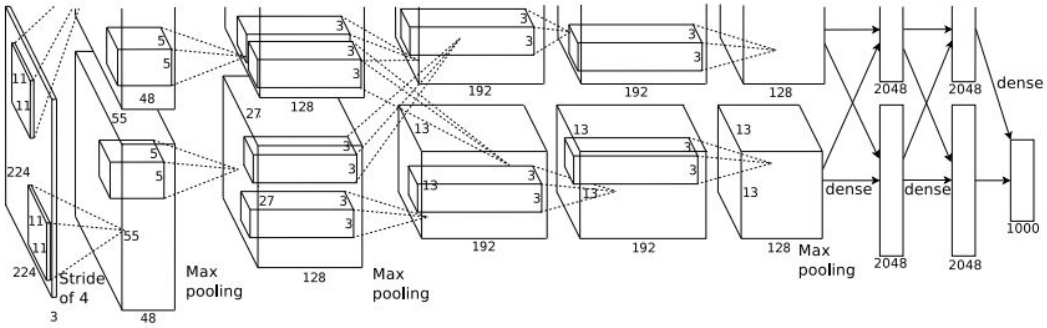
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Case Study: AlexNet

[Krizhevsky et al. 2012]



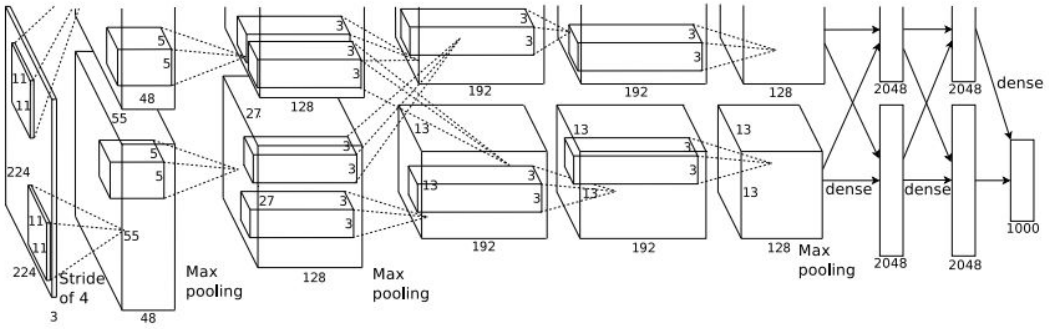
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Case Study: AlexNet

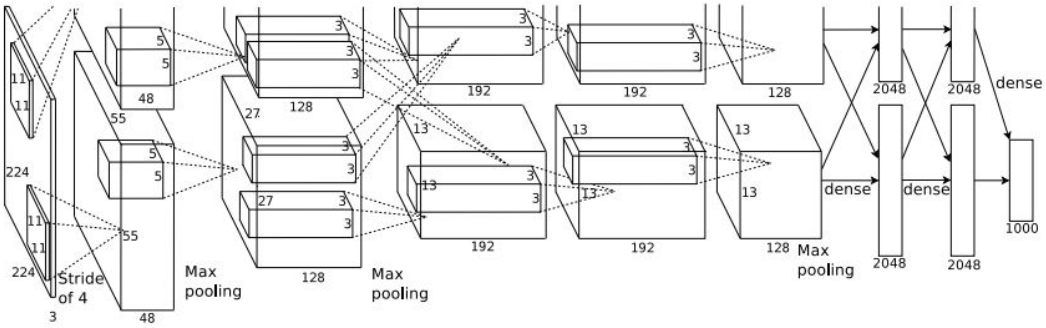
[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

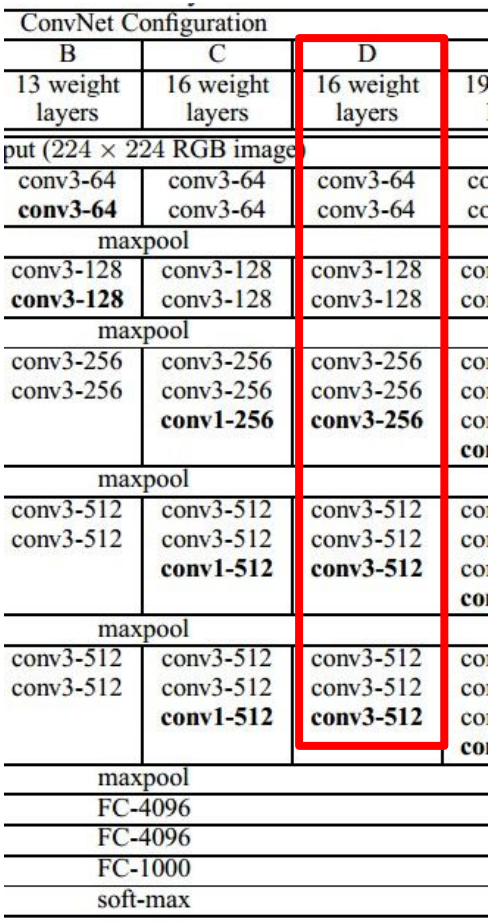
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	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 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

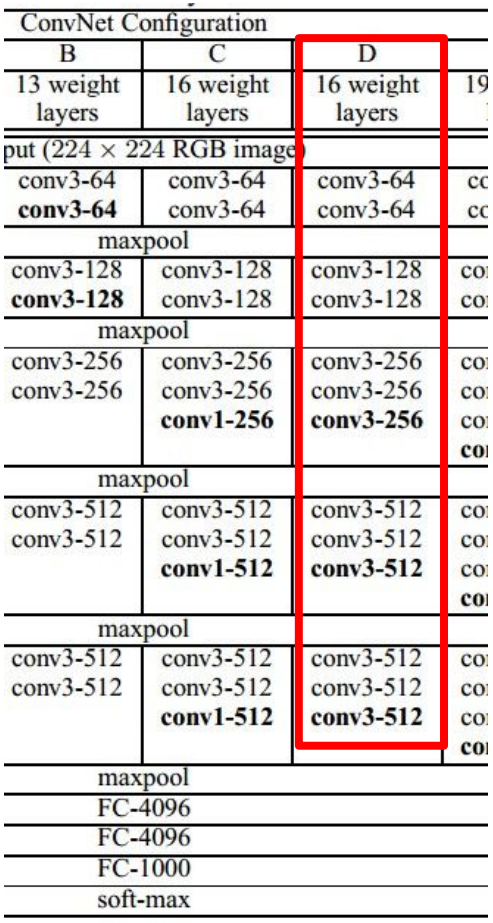


ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	co
conv3-64	conv3-64	conv3-64	co
maxpool			
conv3-128	conv3-128	conv3-128	co
conv3-128	conv3-128	conv3-128	co
maxpool			
conv3-256	conv3-256	conv3-256	co
conv3-256	conv3-256	conv3-256	co
	conv1-256	**conv3-256**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

(not counting biases)

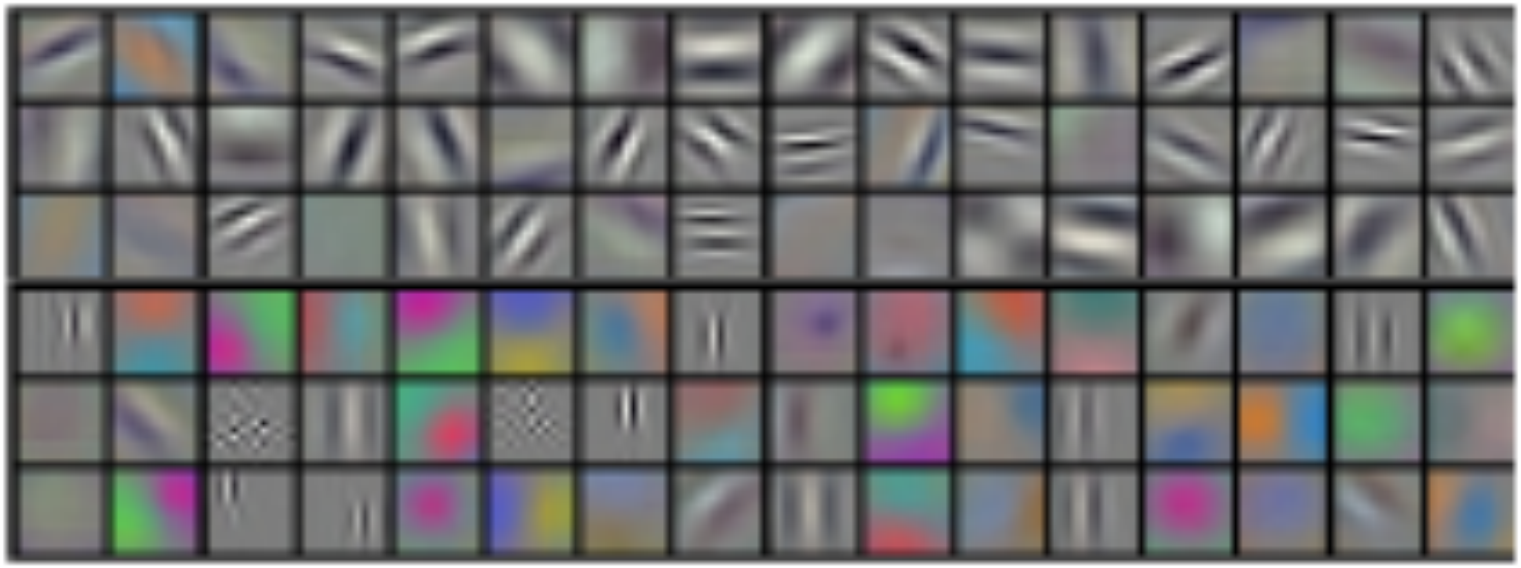
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	co
conv3-64	conv3-64	conv3-64	co
maxpool			
conv3-128	conv3-128	conv3-128	co
conv3-128	conv3-128	conv3-128	co
maxpool			
conv3-256	conv3-256	conv3-256	co
conv3-256	conv3-256	conv3-256	co
	conv1-256	**conv3-256**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

First Layer



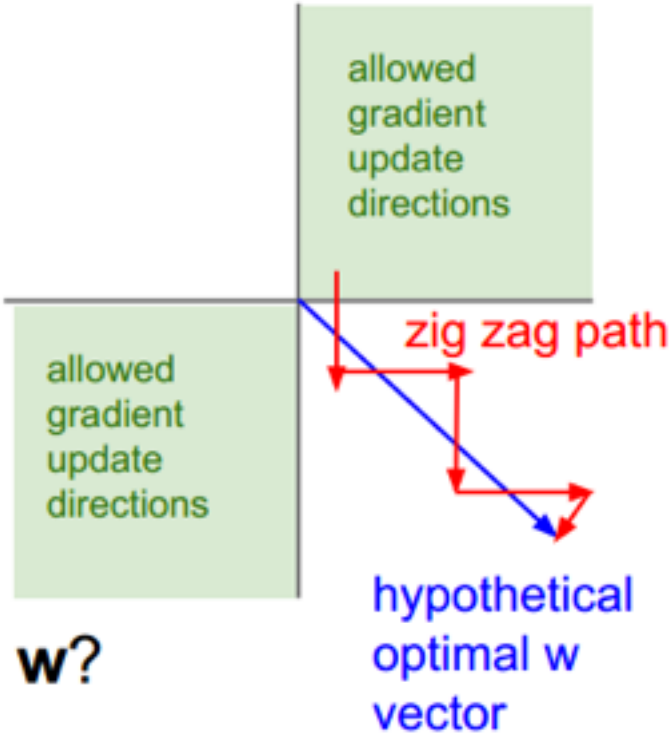
Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$

What can we say about the gradients on **w**?

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients on **w**?

Always all positive or all negative :(

(this is also why you want zero-mean data!)