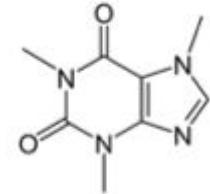


# DIY Deep Learning for Vision: A Full-Day Caffe Tutorial



	Maximally accurate	Maximally specific
espresso		2.23192
coffee		2.19914
beverage		1.93214
liquid		1.89367
fluid		1.85519



[caffe.berkeleyvision.org](http://caffe.berkeleyvision.org)



[github.com/BVLC/caffe](https://github.com/BVLC/caffe)



**BAIR**

BERKELEY ARTIFICIAL INTELLIGENCE RESEARCH

**embedded**  
**VISION**  
ALLIANCE

Evan Shelhamer, Jeff Donahue, Jon Long

# The Highest Bandwidth Input Channel

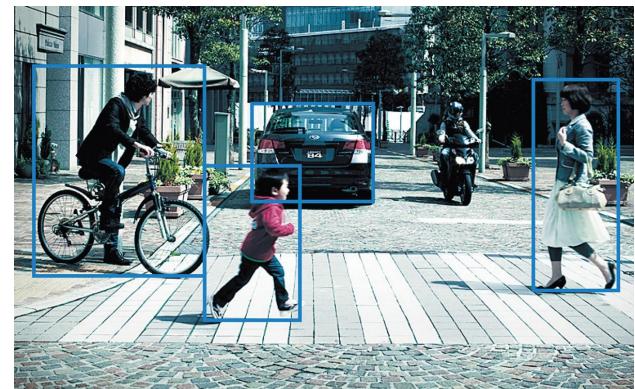
Machines are useful mainly to the extent that they **interact with the physical world**

**Visual information is the richest source** of information about the real world

**Vision is the highest-bandwidth** mode for machines to obtain real-world info

Embedded vision enables our things to be:

- **More responsive**
- **More personal and secure**
- **Safer, more autonomous**
- **Easier to use**



# Tutorial Organization

## Part 1 (Morning)

Introduction to Deep Learning  
and Convolutional Networks:

*A Theoretical and Practical  
Discussion*



## Part 2 (Afternoon)

Hands-On Introduction to the  
Caffe Framework:

*Brewing Networks with Caffe*



# Part 1: Deep Learning for Vision

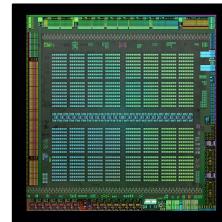
Why Deep Learning?



Dive into Deep Learning



What is DL?  
Why Now?



The Challenge  
of Recognition



Learning &  
Optimization

$$\nabla_{\theta} L$$

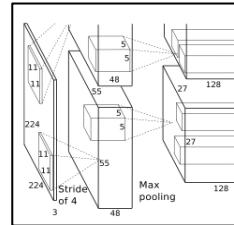
Applications



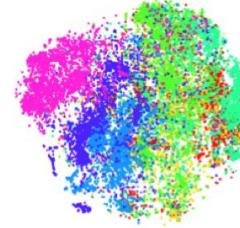
Caffe First Sip



Network Tour

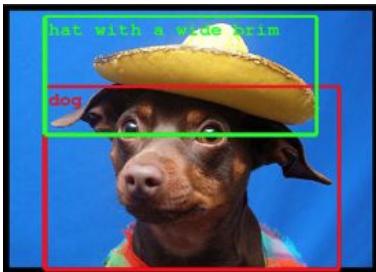


Transfer Learning

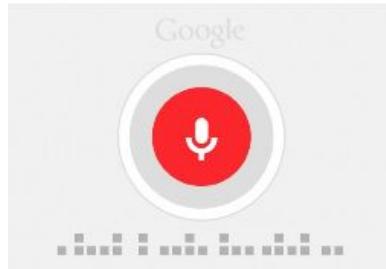


# Why Deep Learning?

## End-to-End Learning for Many Tasks



vision



speech



text



control

# Visual Recognition Tasks

## Classification

- what kind of image?
- which kind(s) of objects?

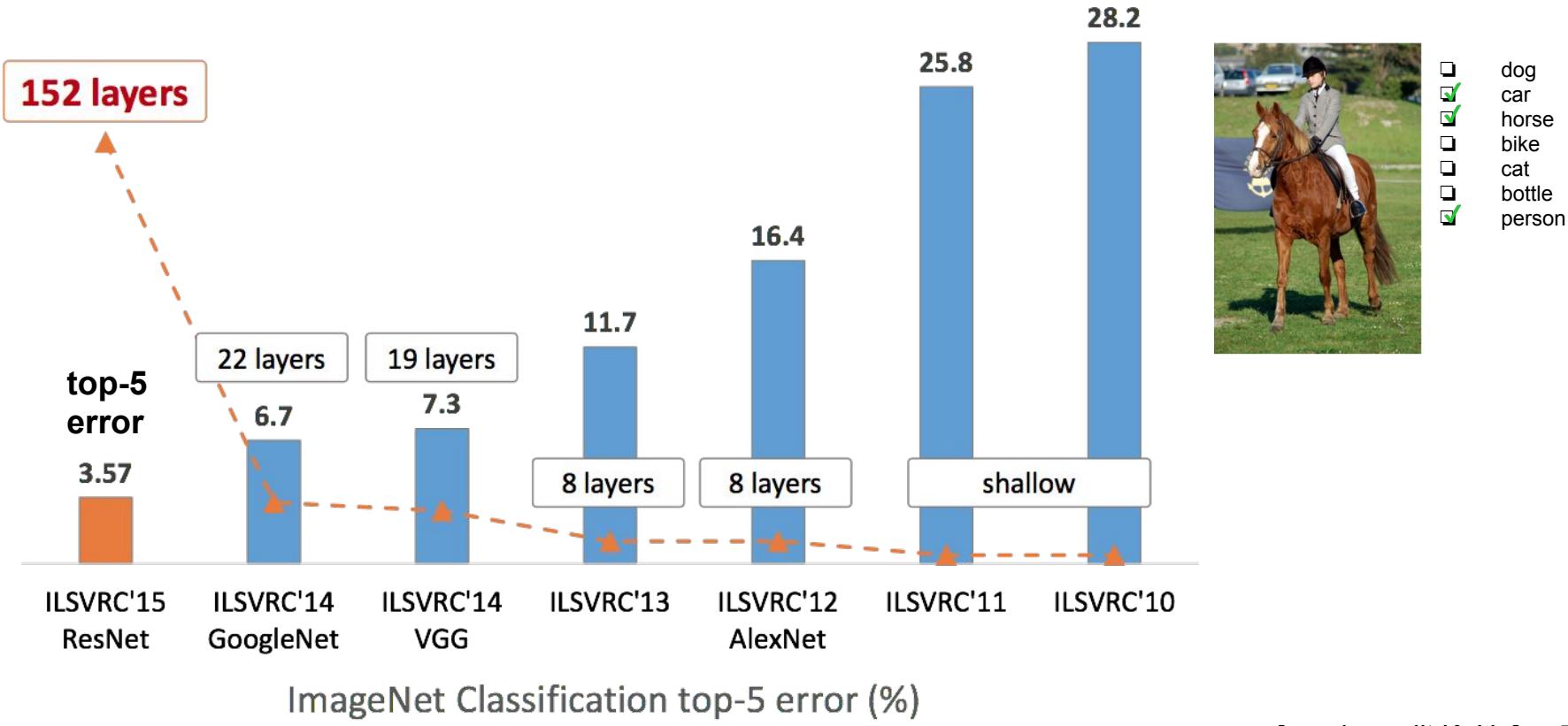


- dog
- car
- horse
- bike
- cat
- bottle
- person

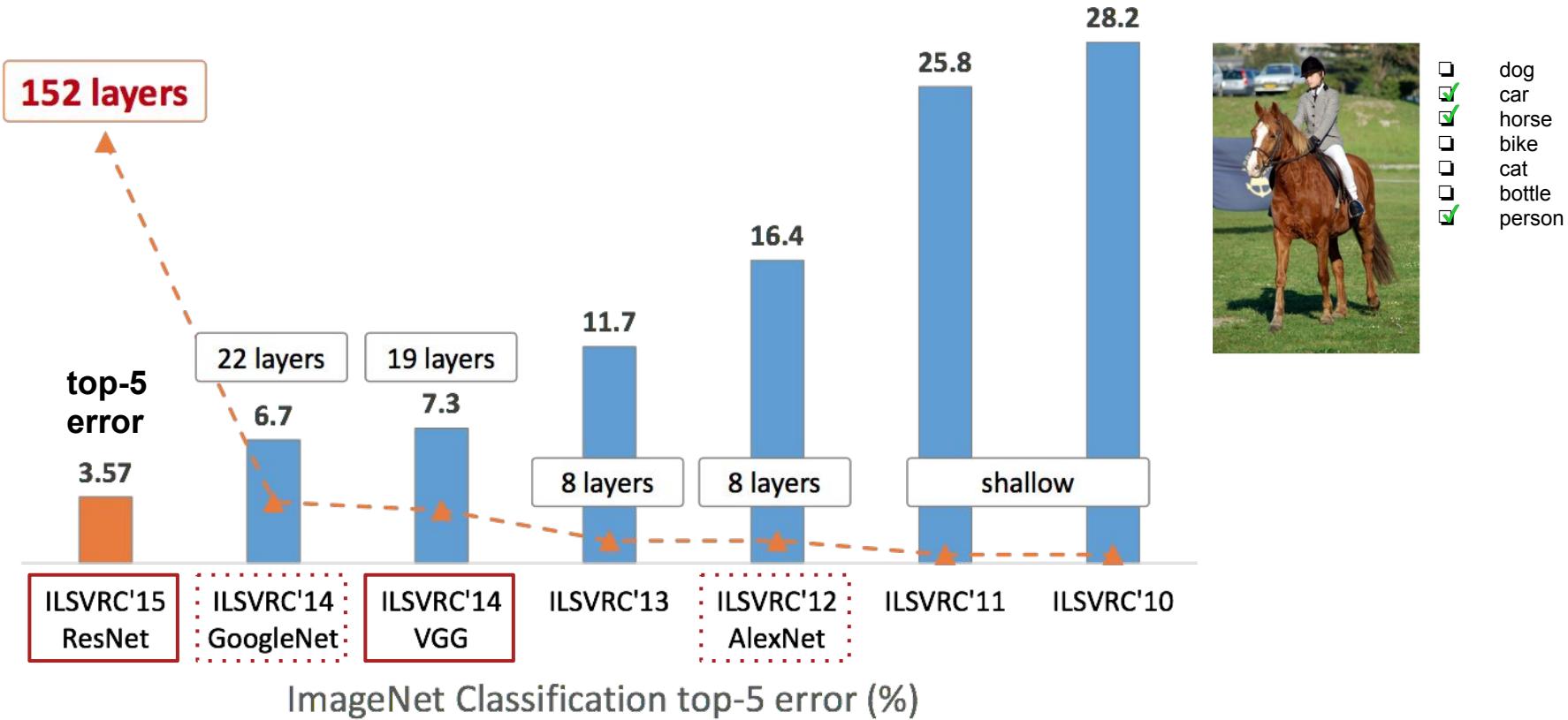
## Challenges

- appearance varies by lighting, pose, context, ...
- clutter
- fine-grained categorization (horse or exact species)

# Image Classification: ILSVRC 2010-2015



# Image Classification: ILSVRC 2010-2015



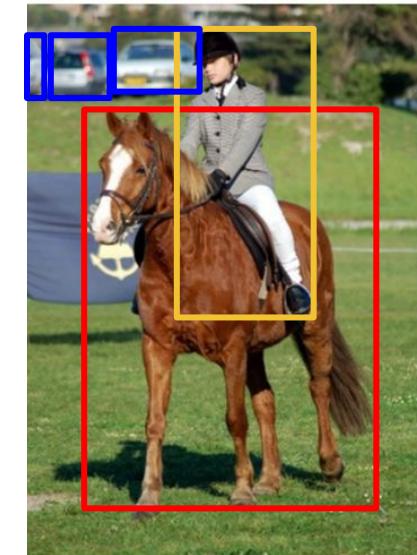
# Visual Recognition Tasks

## Detection

- what objects are there?
- where are the objects?

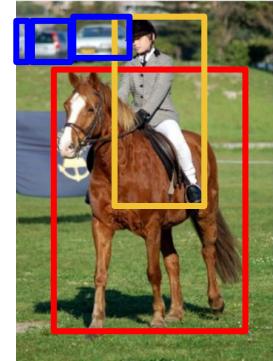
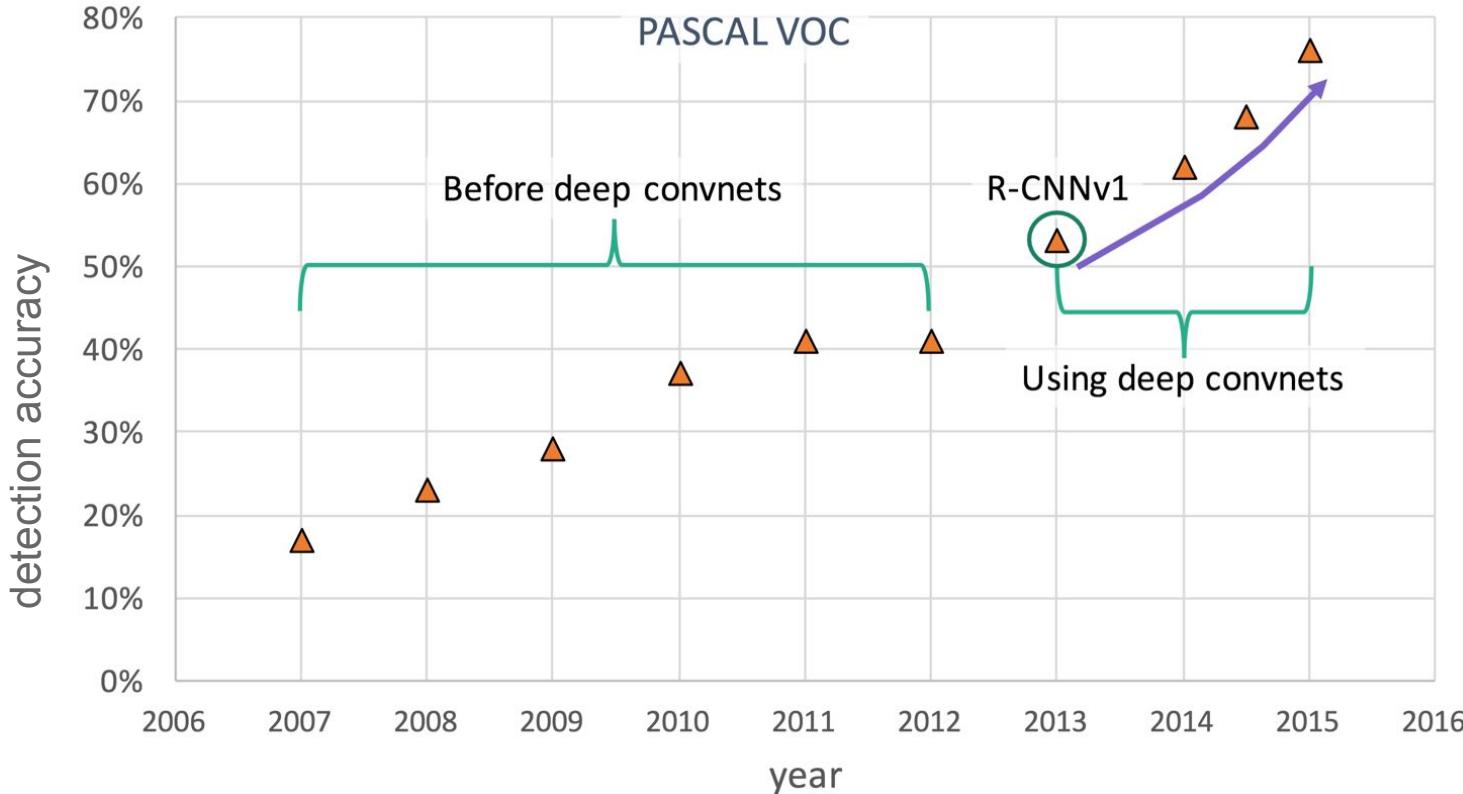
## Challenges

- localization
- multiple instances
- small objects



car   person   horse

# Detection: PASCAL VOC



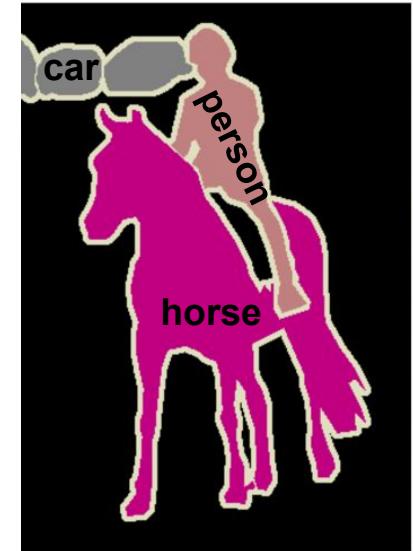
**R-CNN:**  
regions +  
convnets

state-of-the-art,  
in Caffe

# Visual Recognition Tasks

## Semantic Segmentation

- what kind of thing  
is each pixel part of?
- what kind of stuff  
is each pixel?



## Challenges

- tension between  
recognition and localization
- amount of computation

# Segmentation: PASCAL VOC

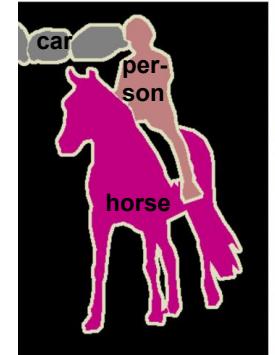
## Leaderboard

MSRA_BoxSup [?]	75.2
Oxford_TVGG_CRF_RNN_COCO [?]	74.7
DeepLab-MSC-CRF-LargeFOV-COCO-CrossJoint [?]	73.9
Adelaide_Context_CNN_CRF_VOC [?]	72.9
DeepLab-CRF-COCO-LargeFOV [?]	72.7
POSTECH_EDeconvNet_CRF_VOC [?]	72.5
Oxford_TVGG_CRF_RNN_VOC [?]	72.0
DeepLab-MSC-CRF-LargeFOV [?]	71.6
MSRA_BoxSup [?]	71.0
DeepLab-CRF-COCO-Strong [?]	70.4
DeepLab-CRF-LargeFOV [?]	70.3
TTI_zoomout_v2 [?]	69.6
DeepLab-CRF-MSC [?]	67.1
DeepLab-CRF [?]	66.4
CRF_RNN [?]	65.2
TTI_zoomout_16 [?]	64.4
Hypercolumn [?]	62.6
<b>FCN-8s [?]</b>	<b>62.2</b>
MSRA_CFM [?]	61.8
TTI_zoomout [?]	58.4
SDS [?]	51.6
NUS_UDS [?]	50.0
TTIC-divmbest-rerank [?]	48.1
BONN_O2PCPMC_FGT_SEGM [?]	47.8
BONN_O2PCPMC_FGT_SEGM [?]	47.5
BONNGC_O2P_CPMC_CSI [?]	46.8
BONN_CMFR_O2P_CPMC_LIN [?]	46.7

deep learning with Caffe

end-to-end networks lead to  
30 points absolute or 50% relative improvement  
and >100x speedup in 1 year!

(papers published for +1 or +2 points)



FCN:  
pixelwise  
convnet

state-of-the-art,  
in Caffe

WHEN A USER TAKES A PHOTO,  
THE APP SHOULD CHECK WHETHER  
THEY'RE IN A NATIONAL PARK...

SURE, EASY GIS LOOKUP.  
GIMME A FEW HOURS.

... AND CHECK WHETHER  
THE PHOTO IS OF A BIRD.

I'LL NEED A RESEARCH  
TEAM AND FIVE YEARS.



xkcd: Tasks

“The Virtually Impossible”

IN CS, IT CAN BE HARD TO EXPLAIN  
THE DIFFERENCE BETWEEN THE EASY  
AND THE VIRTUALLY IMPOSSIBLE.



EXAMPLE PHOTOS



Photo credits

# PARK or BIRD

Want to know if your photo is from a U.S. national park?

Want to know if it contains a bird? Just drag it into the box to the left, and we'll tell you. We'll use the GPS embedded in your photo (if it's there) to see whether it's from a park, and we'll use our super-cool computer vision skills to try to see whether it's a bird (which is a hard problem, but we do a pretty good job at it).

To try it out, just drag any photo from your desktop into the upload box, or try dragging any of our example images. We'll give you your answers below!

Want to know more about PARK or BIRD, including why the heck we did this? Just click here for more info →

PARK?

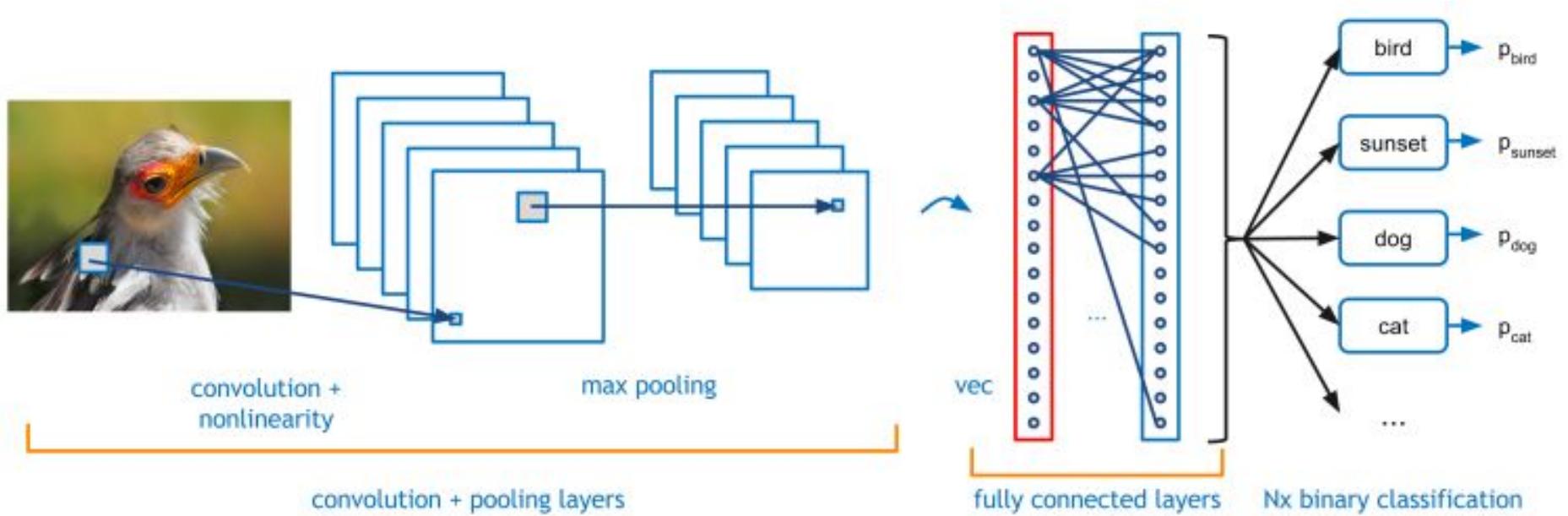
**YES**

Ah yes, [Everglades](#) is truly beautiful.

BIRD?

**YES**

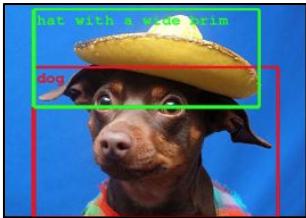
Dude, that is such a bird.



All in a day's work with Caffe

# Part 1: Deep Learning for Vision

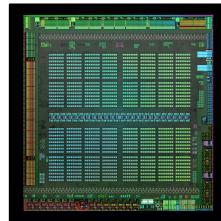
Why Deep Learning?



Dive into Deep Learning



What is DL?  
Why Now?



The Challenge  
of Recognition



Learning &  
Optimization

$$\nabla_{\theta} L$$

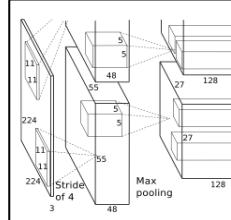
Applications



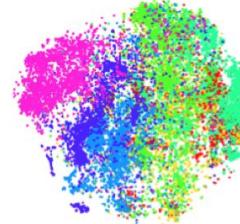
Caffe First Sip



Network Tour



Transfer Learning



# First Dive Into Deep Learning

Deep Learning is

Stacking **Layers**

and

Learning **End-to-End**



# Stacking Layers

A **layer** is a transformation

$$\textcolor{red}{x}' = \mathbf{layer}(x)$$

Deep networks are layered models made by stacking different types of transformation

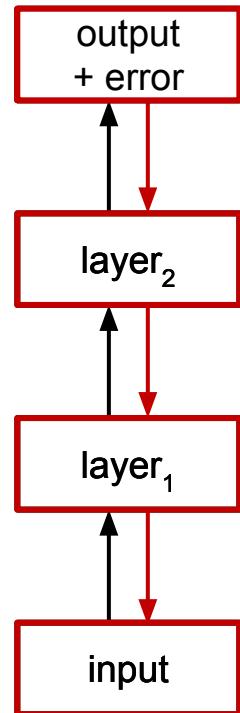
$$\textcolor{red}{x}_2 = \mathbf{layer}_1(x_1)$$

$$\textcolor{red}{x}_3 = \mathbf{layer}_2(x_2)$$

*How do layers stack?*

...

# Layered Networks



Networks run layer-by-layer, composing the input-output transformation of each layer

$$x_1 = \text{layer}_1(\text{input})$$

$$\text{out} = \text{layer}_2(x_1)$$

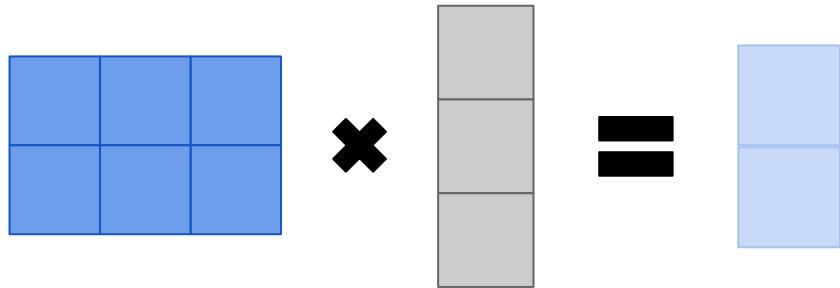
During learning, the **error** is passed back layer-by-layer to tune the transformations

*What kind of layers should we stack?*

# The Simplest Layers

**Matrix Multiplication**

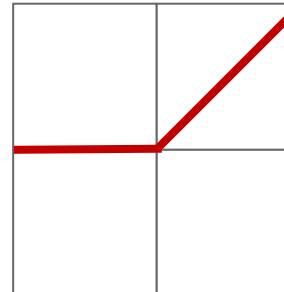
$$x' = Wx + b$$



**Non-linearity**

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

(for example)

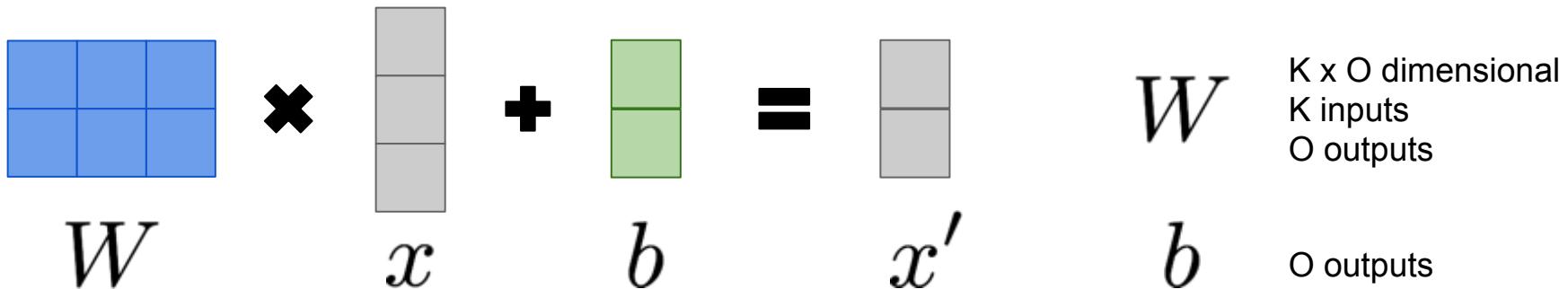


# Matrix Multiplication

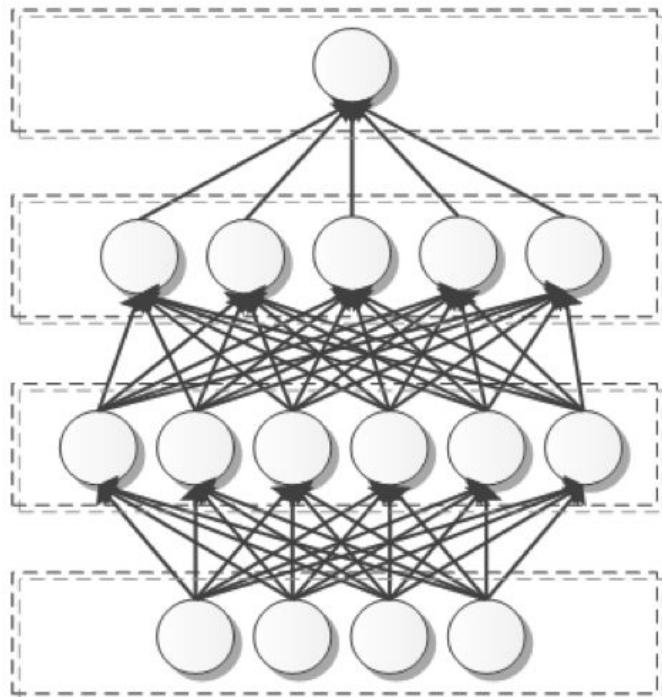
Multiply input  $x$  by weights  $W$  and add bias  $b$

Learns linear transformations

$$x' = Wx + b$$



# Matrix Multiplication == Fully Connected Layer

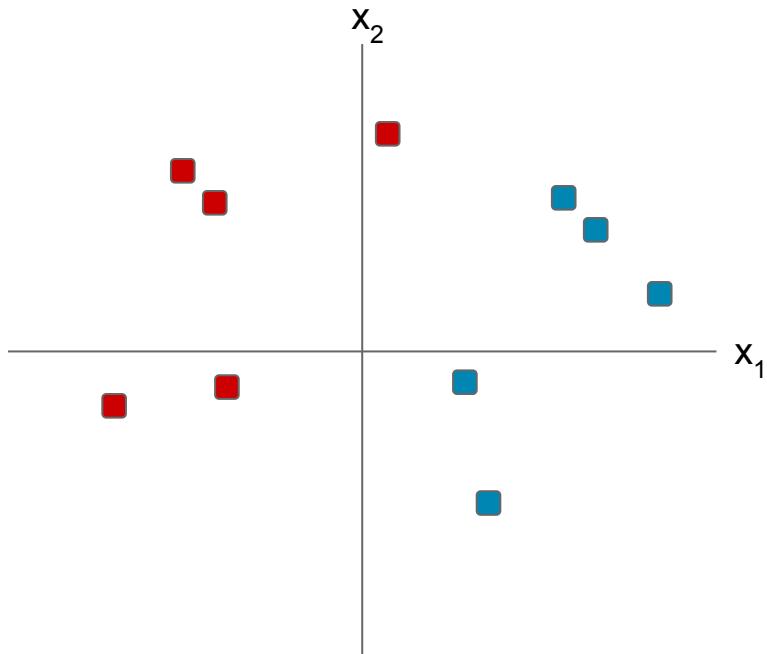


Output is a function of every input,  
or the input and output are  
“fully connected”

Abbreviated as **FC**

# Linear Classification

To classify we need to separate the data into red vs. blue

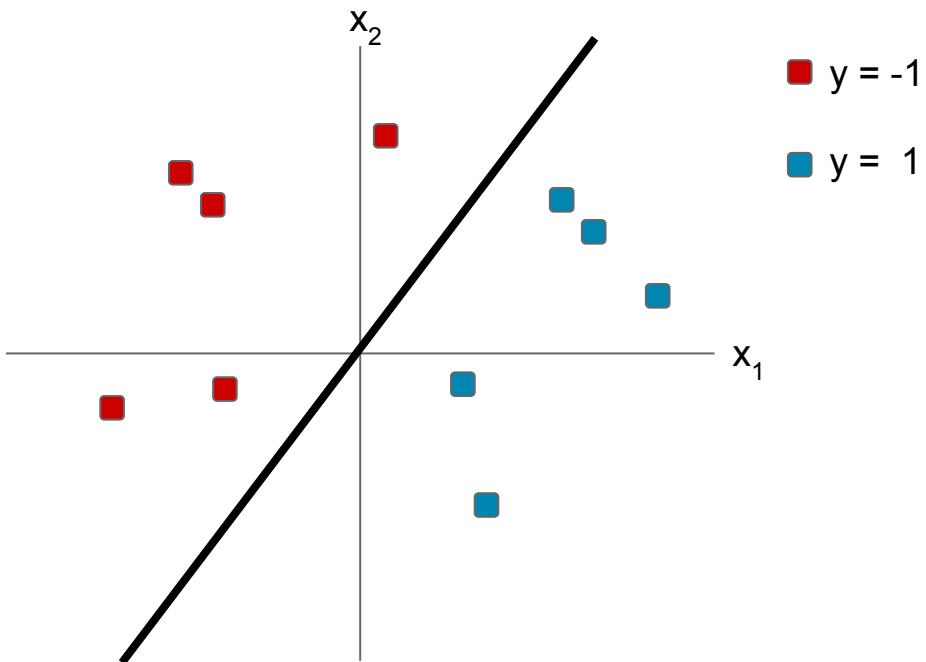


- $y = -1$
  - $y = 1$
- Suppose our data points ( $\mathbf{x}$ ) are 2D and each comes with a label  $y$ , where  $y = -1$  or  $y = 1$
  - Learn a weight vector  $\mathbf{w} = [w_1; w_2]$
  - Predict the class of a given  $\mathbf{x}$  by  $\text{sign}(\mathbf{w}^T \mathbf{x}) = \text{sign}(w_1 x_1 + w_2 x_2)$

?

# Linear Classification

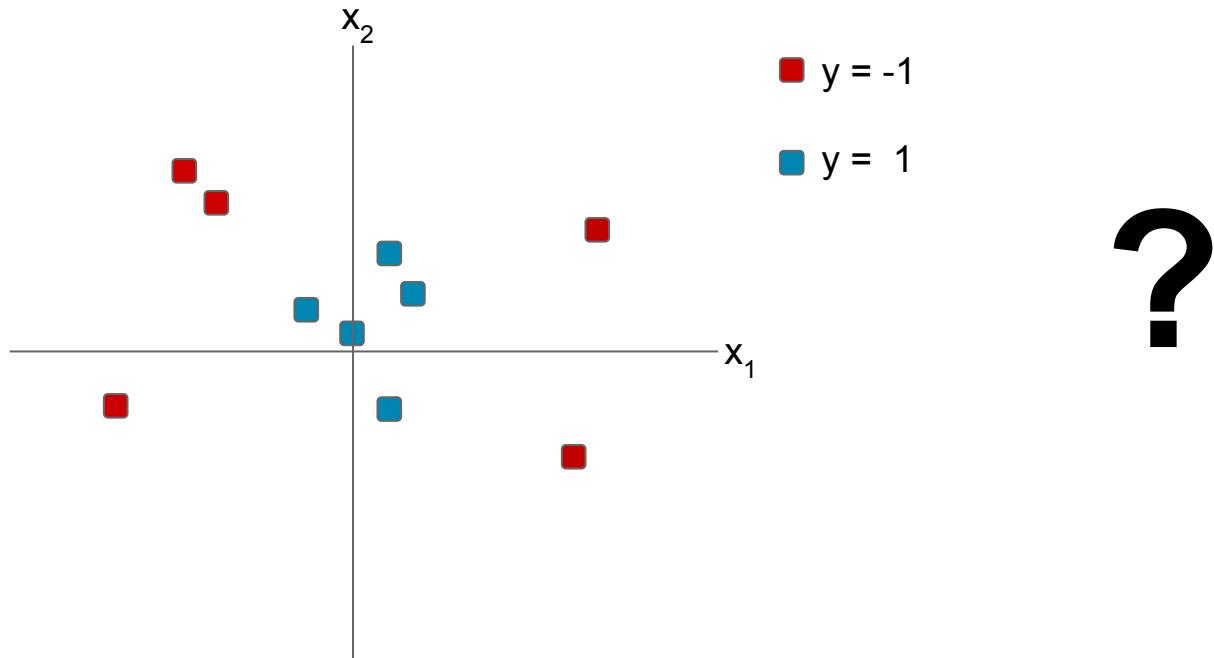
To classify we need to separate the data into red vs. blue



- Suppose our data points ( $\mathbf{x}$ ) are 2D and each comes with a label  $y$ , where  $y = -1$  or  $y = 1$
- Learn a weight vector  $\mathbf{w} = [w_1; w_2]$
- Predict the class of a given  $\mathbf{x}$  by  $\text{sign}(\mathbf{w}^T \mathbf{x}) = \text{sign}(w_1 x_1 + w_2 x_2)$

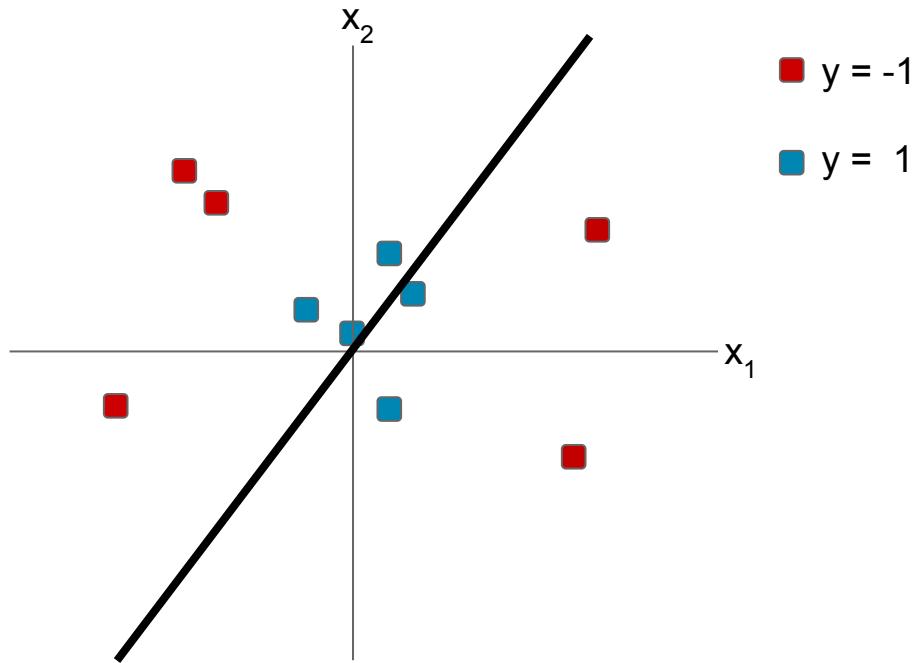
# Linearity is Not Enough

To classify we need to separate the data into red vs. blue



# Linearity is Not Enough

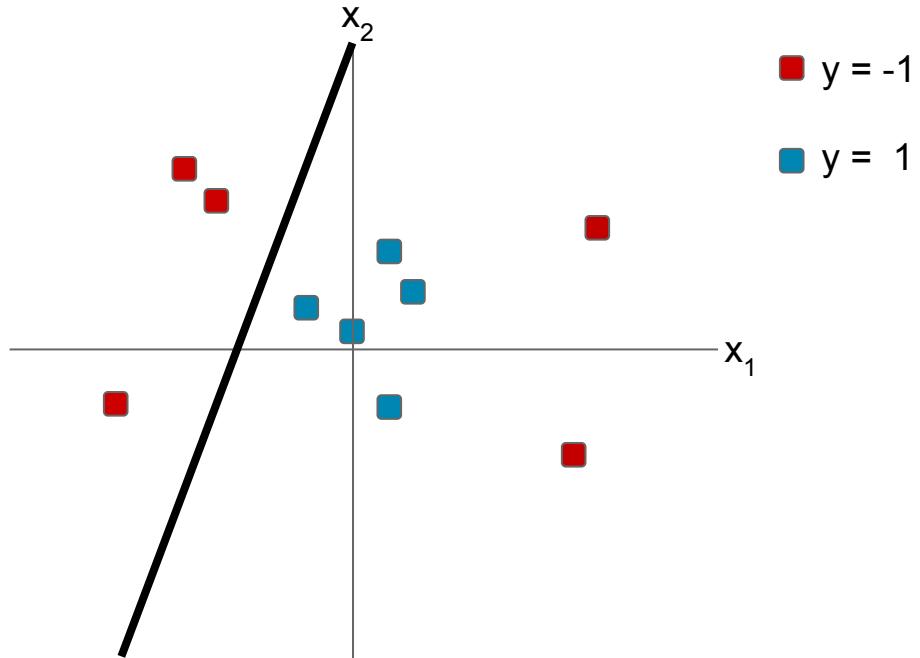
To classify we need to separate the data into red vs. blue



**NO**

# Linearity is Not Enough

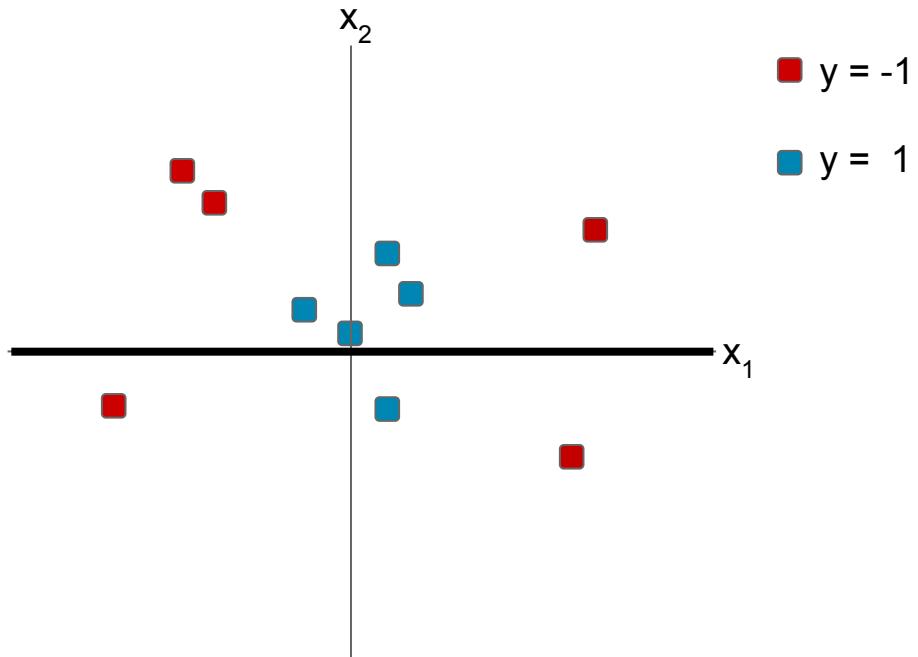
To classify we need to separate the data into red vs. blue



**NO**

# Linearity is Not Enough

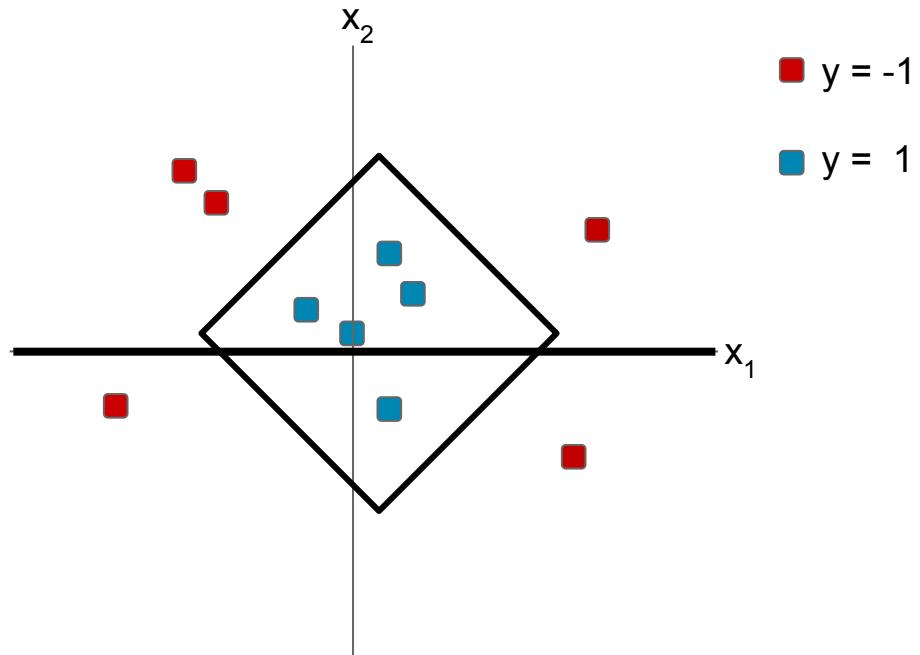
To classify we need to separate the data into red vs. blue



**NO**

# Linearity is Not Enough

To classify we need to separate the data into red vs. blue

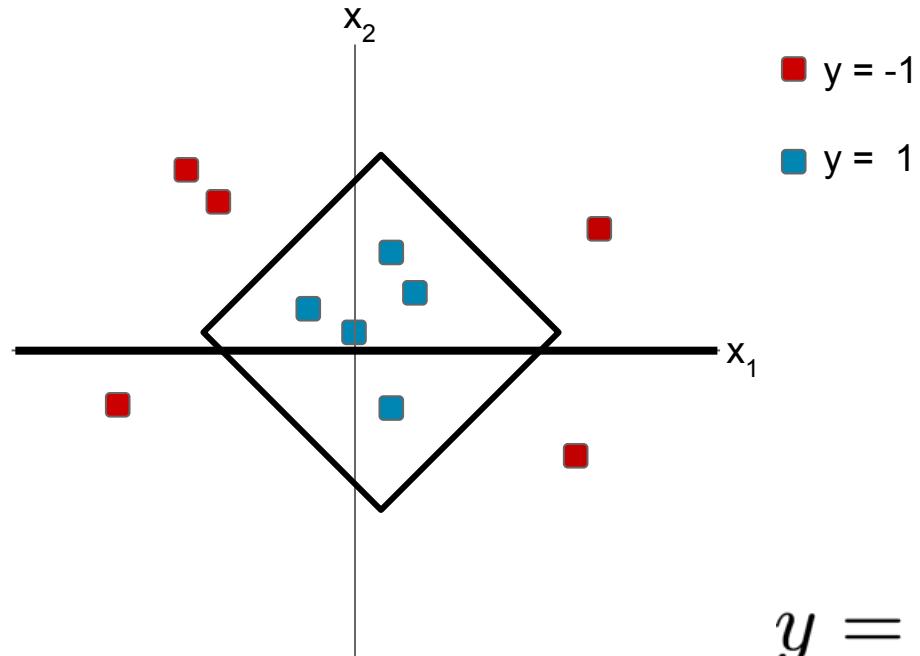


**YES**

Non-linearity!

# Linearity is Not Enough

To classify we need to separate the data into red vs. blue



**YES**

Non-linearity!

$$y = W_2 \max(0, (W_1(x)))$$

# The Limits of Linearity

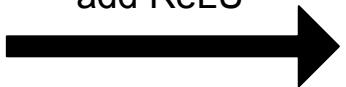
Linear steps **collapse** and stay linear

$$\begin{array}{ccc} x_2 = W_1 x_1 & \longrightarrow & x_3 = W_{\text{both}} x_1 \\ x_3 = W_2 x_2 & & W_{\text{both}} = W_2 \times W_1 \end{array}$$

Linear layers alone do not meaningfully stack

# The Shallowest Deep Net

Deep nets are made by stacking learned linear layers and simple pointwise **non-linear** layers

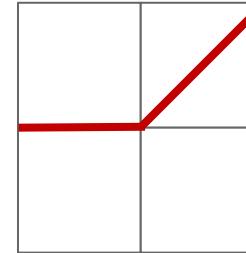
Linear		Non-linear, Deep
$x_2 = W_1 x_1$		$x_2 = W_1 x_1$
$x_3 = W_2 x_2$	 add ReLU	$x_2^\circ = \max(0, x_2)$ $x_3 = W_2 x_2^\circ$

Due to the Rectified Linear Unit (ReLU) non-linearity **max(0, x)**,  $x_3$  cannot be computed as a linear function of

# Non-linearity

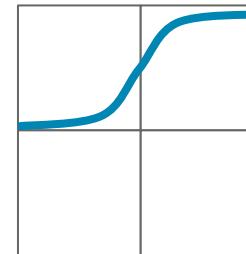
**ReLU**

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



**Sigmoid**

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

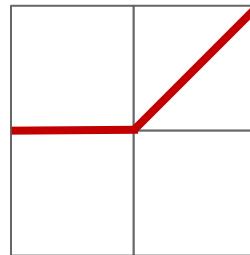


**Non-linearity** is needed to deepen the representation  
Many non-linearities or *activations* to choose from

# Yet More Non-linearities

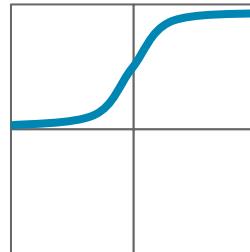
**ReLU**

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



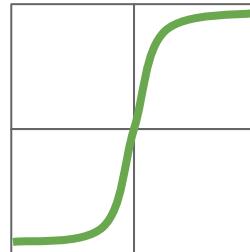
**Sigmoid**

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



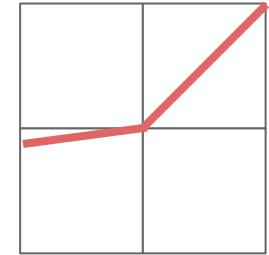
**TanH**

$$x' = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



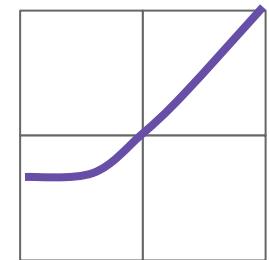
**Leaky ReLU**

$$x' = \max(0.1x, x)$$



**ELU**

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



When in doubt, **ReLU**

Worth trying Leaky ReLU, ELU

Avoid Sigmoid

# Define Your First Net

Let's go non-linear on  
a classification problem

[Try It Out](#)

# Designing for Sight

**Convolutional Networks** or convnets are nets for vision

- functional fit for the visual world  
by compositionality and feature sharing
- learned end-to-end to handle visual detail  
for more accuracy and less engineering

Convnets are the dominant architectures for visual tasks

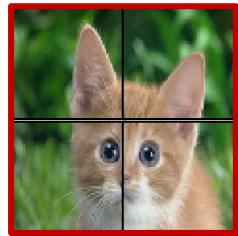
# Visual Structure

**Local Processing:** pixels close together go together  
*receptive fields* capture local detail

**Across Space:** the same what, no matter where  
recognize the same input in different places

# Visual Structure

**Local Processing:** pixels close together go together  
*receptive fields* capture local detail



Can rely on spatial coherence

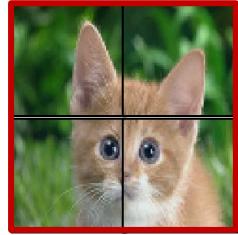


This is not a cat

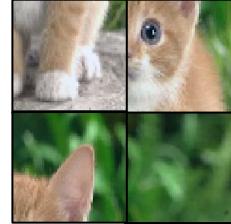
**Across Space:** the same what, no matter where  
recognize the same input in different places

# Visual Structure

**Local Processing:** pixels close together go together  
*receptive fields* capture local detail



Can rely on spatial coherence



This is not a cat

**Across Space:** the same what, no matter where  
recognize the same input in different places



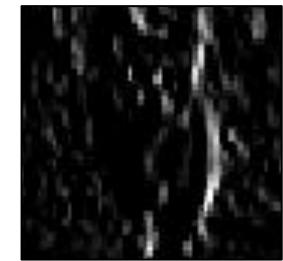
All of these are cats

# Vision Layers

**Convolution/Filtering**  
linear layer for vision



Learned Filter



**Pooling**  
spatial summarization

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

max pool 2x2  
with stride 2



6	8
3	4

# Convolution: A Linear Layer for Vision

Images have *translation invariant* semantics: these are all equally squirrels  
So use the same weights between nodes with the same *spatial* relationship



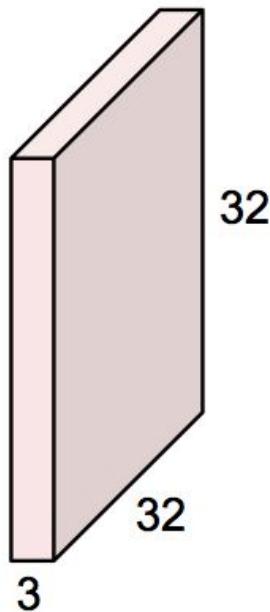
$$\text{output}(x, y) = \sum_{i,j} \text{weight}(i, j) \cdot \text{input}(x + i, y + j)$$

This is **convolution** (or correlation—used interchangeably in vision)

*Convolution* means *fewer parameters* for *more efficient learning*

# A Filter

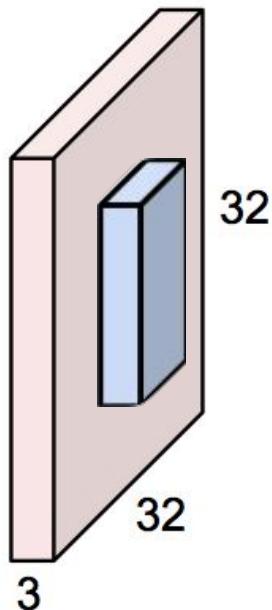
A filter is a **spatially local** and **cross-channel** template  
Convnet filters are **learned**



**input** is **3x32x32** data  
a color image (3 RGB channels) and square (32x32)

# A Filter

A filter is a **spatially local** and **cross-channel** template  
Convnet filters are **learned**



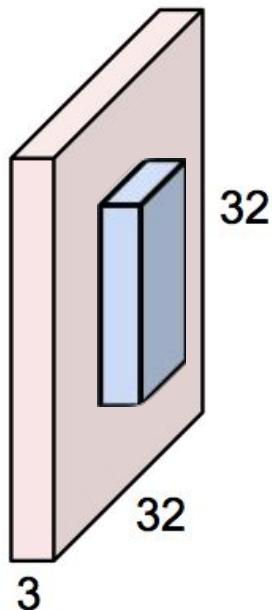
**input** is **3x32x32** data  
a color image (3 RGB channels) and square (32x32)

**filter** is **3x5x5** weights

- spatially local: kernel size is 5x5
- cross-channel: connected across all input channels

# A Filter

A filter is a **spatially local** and **cross-channel** template  
Convnet filters are **learned**



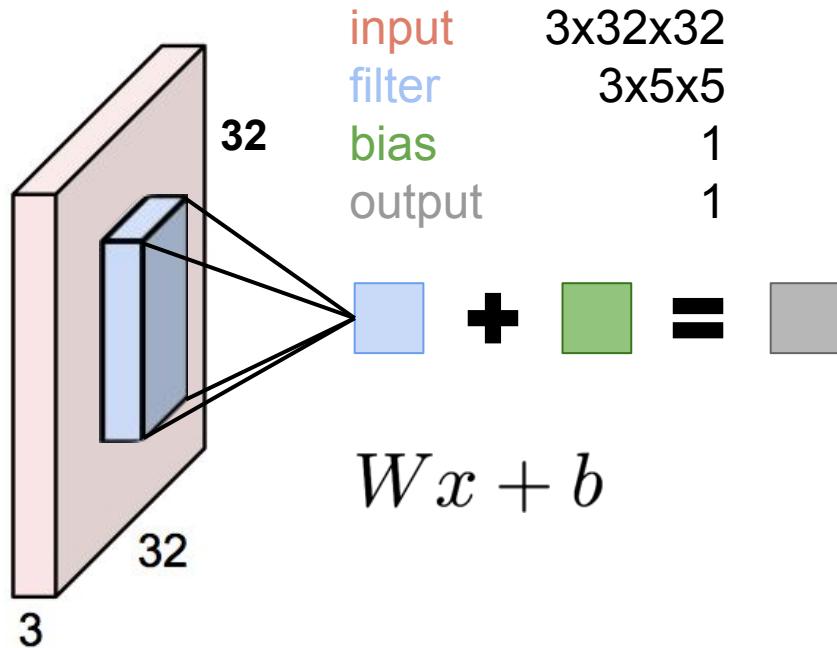
**input** is **3x32x32** data  
a color image (3 RGB channels) and square (32x32)

**filter** is **3x5x5** weights  
- spatially local: kernel size is 5x5  
- cross-channel: connected across all input channels

total parameters:  
 $3 \times 5^2 = 75$  filter weights + 1 bias

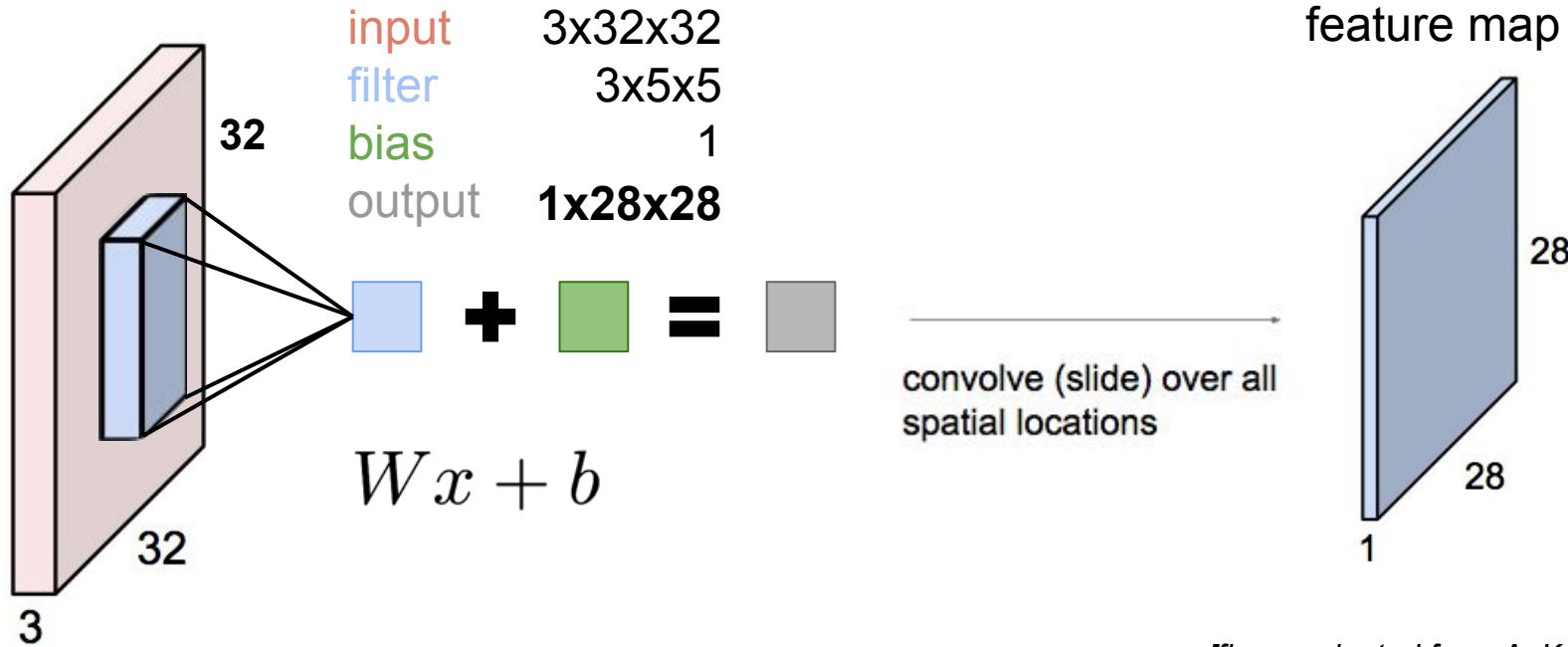
# Convolution

One filter evaluation is a dot product between the input window and weights + bias



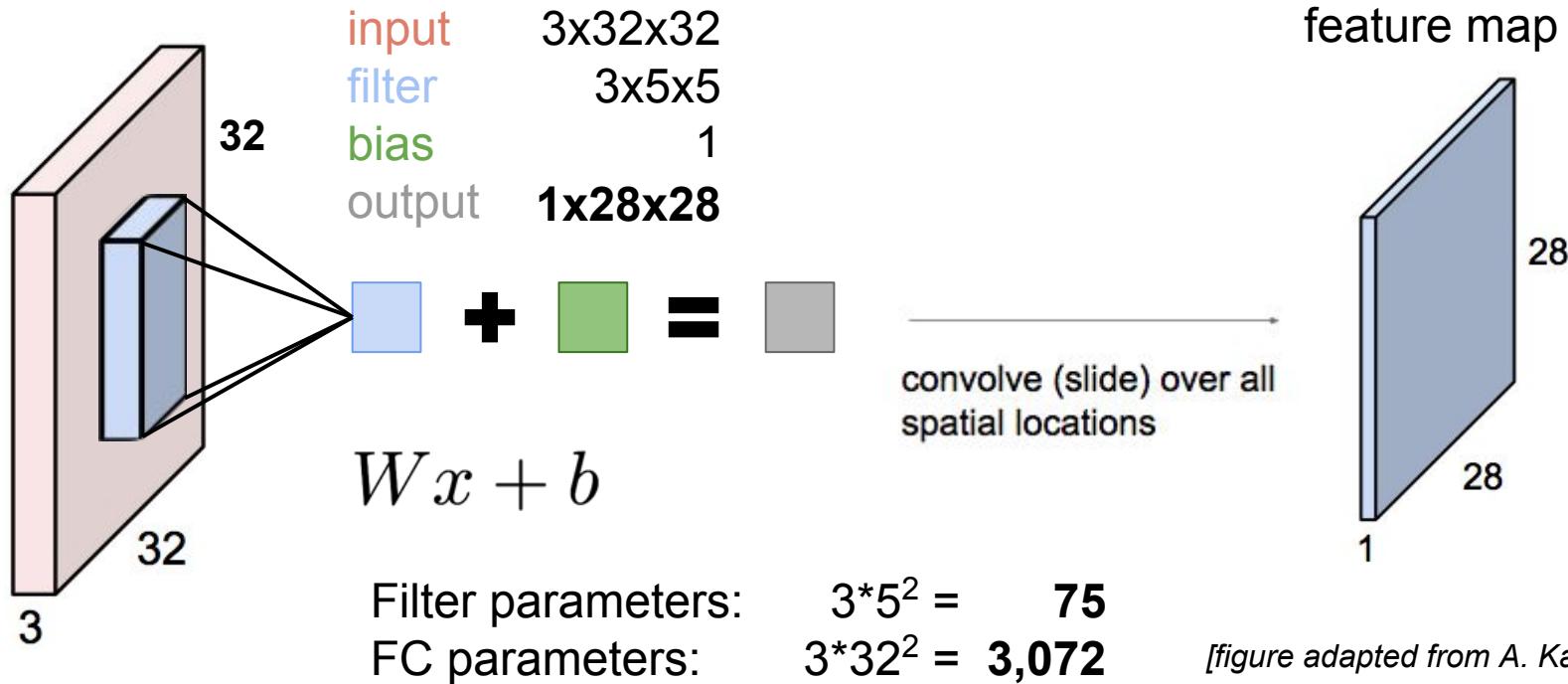
# Convolution

Convolving the filter with the input gives a **feature map**.



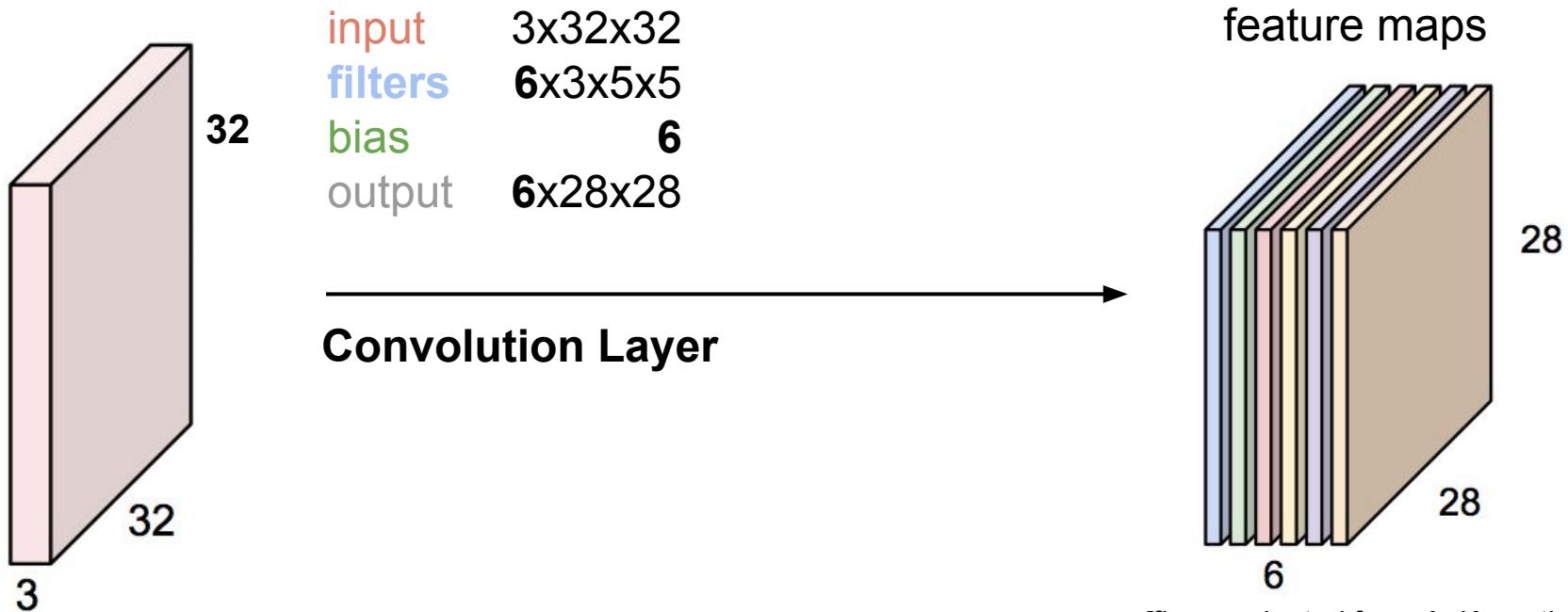
# Convolution

Convolving the filter with the input gives a **feature map**.



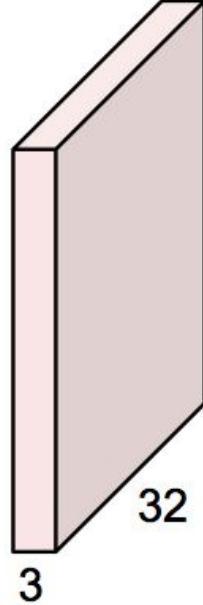
# Convolution Layer (conv)

Convolution layers have multiple **filters** for more modeling capacity



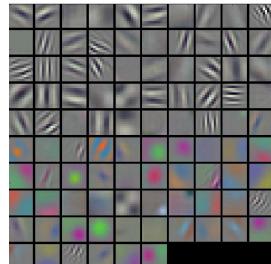
# Convolution Layer (conv)

Convolution layers have multiple **filters** for more modeling capacity



input       $3 \times 32 \times 32$   
filters     $6 \times 3 \times 5 \times 5$   
bias        6  
output      $6 \times 28 \times 28$

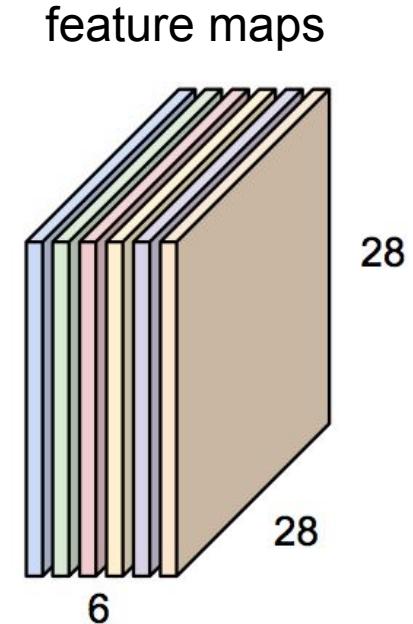
## Convolution Layer



Learned Filters from AlexNet  $\text{conv}_1$

$\text{conv}_1$  has 96 filters for edge, color, and frequency

richer than 3D RGB

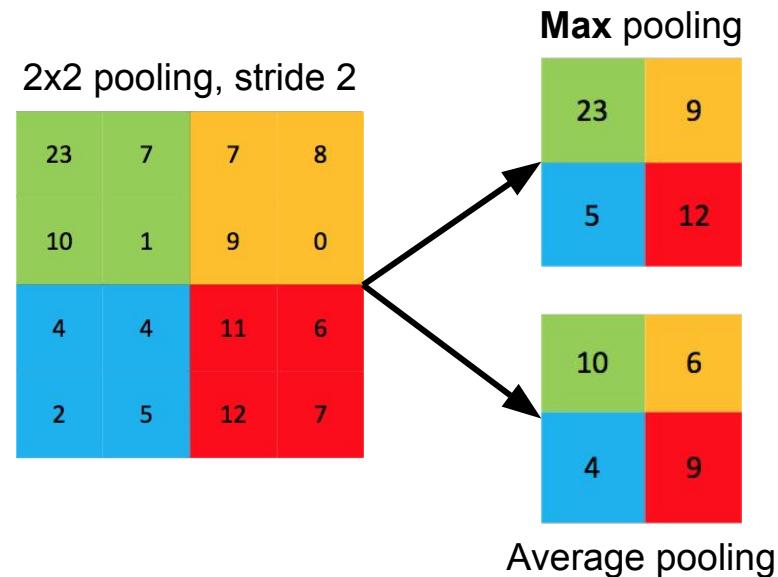


[figure adapted from A. Karpathy] 49

# Pooling (pool)

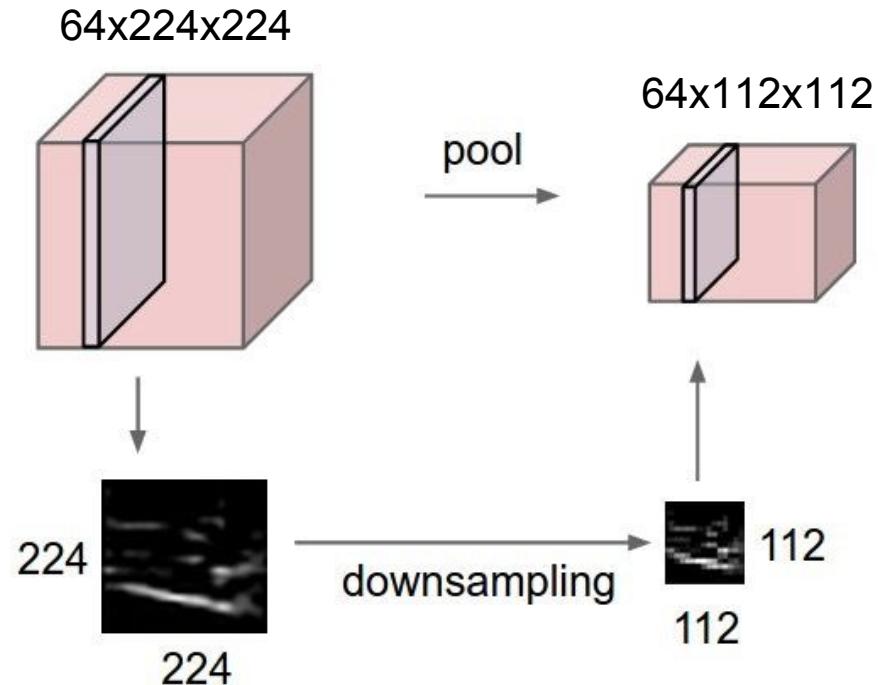
Spatial summary by computing  
**operation over window** with **stride**

- overlapping or non-overlapping
- separate across channels
- Current fashion:  
3x3 max pooling  
with stride 2



# Pooling

- reduce resolution
- increase receptive field size for later layers
- save computation
- add invariance to translation/noise within pooling window

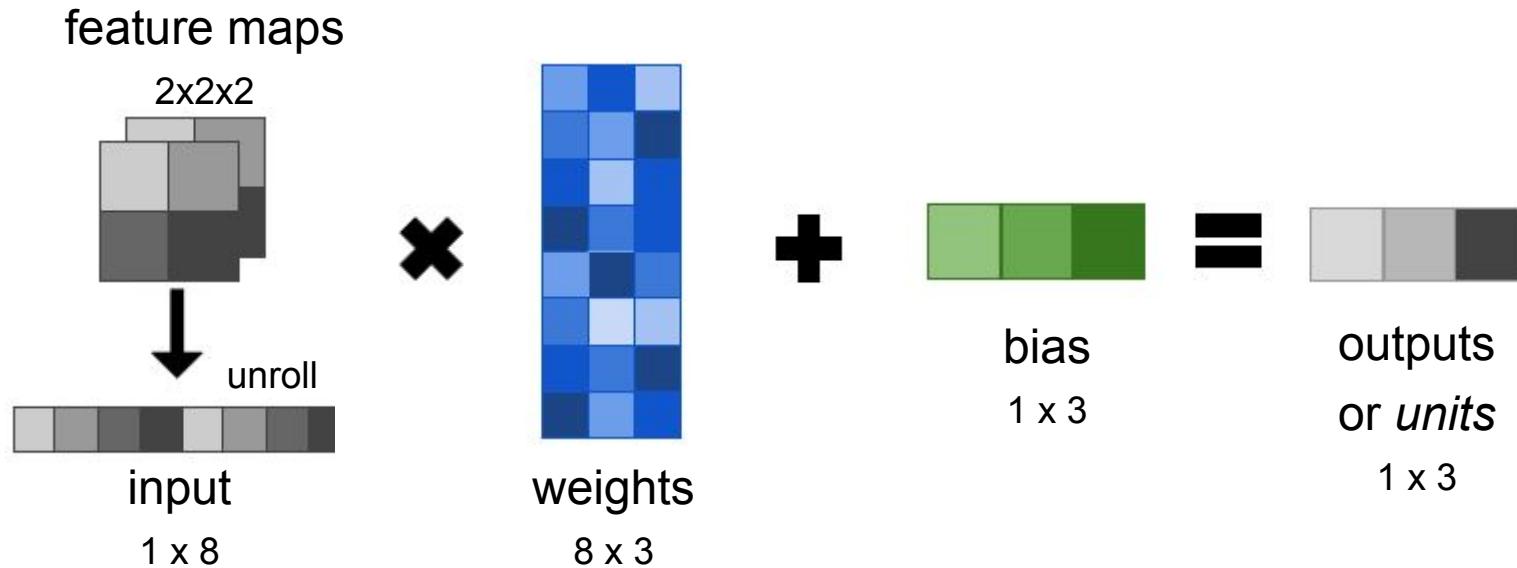


# Fully Connected Layers (FC)

Learn a global feature from the full feature maps

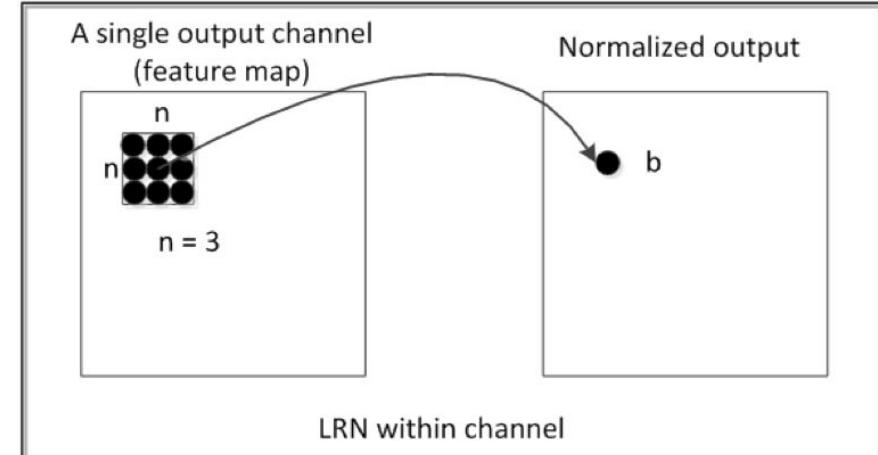
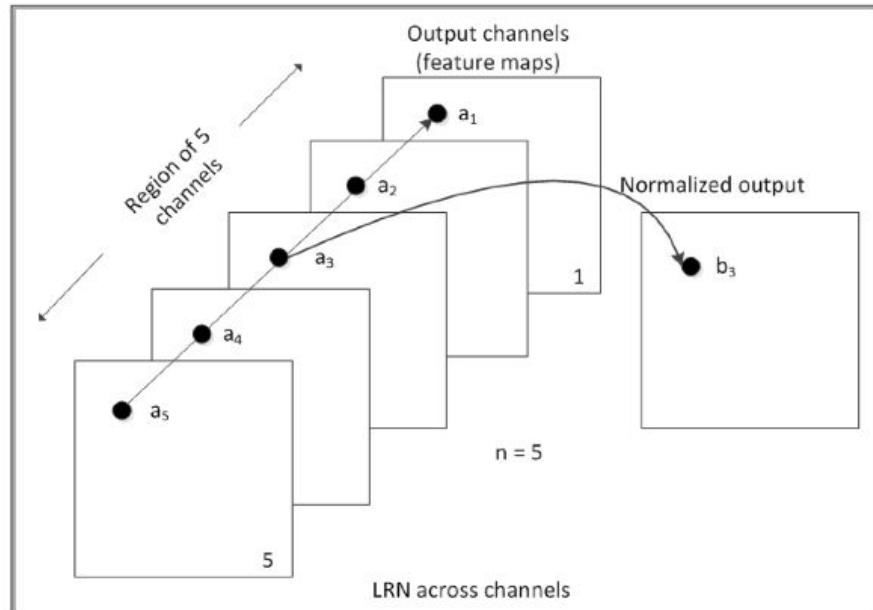
Often found at the end of convnets

Note: this could likewise be done by a large convolution kernel



# Normalization Layers (Deprecated)

Local response normalization was popular for a time but is now deprecated; more recent networks do not include these layers

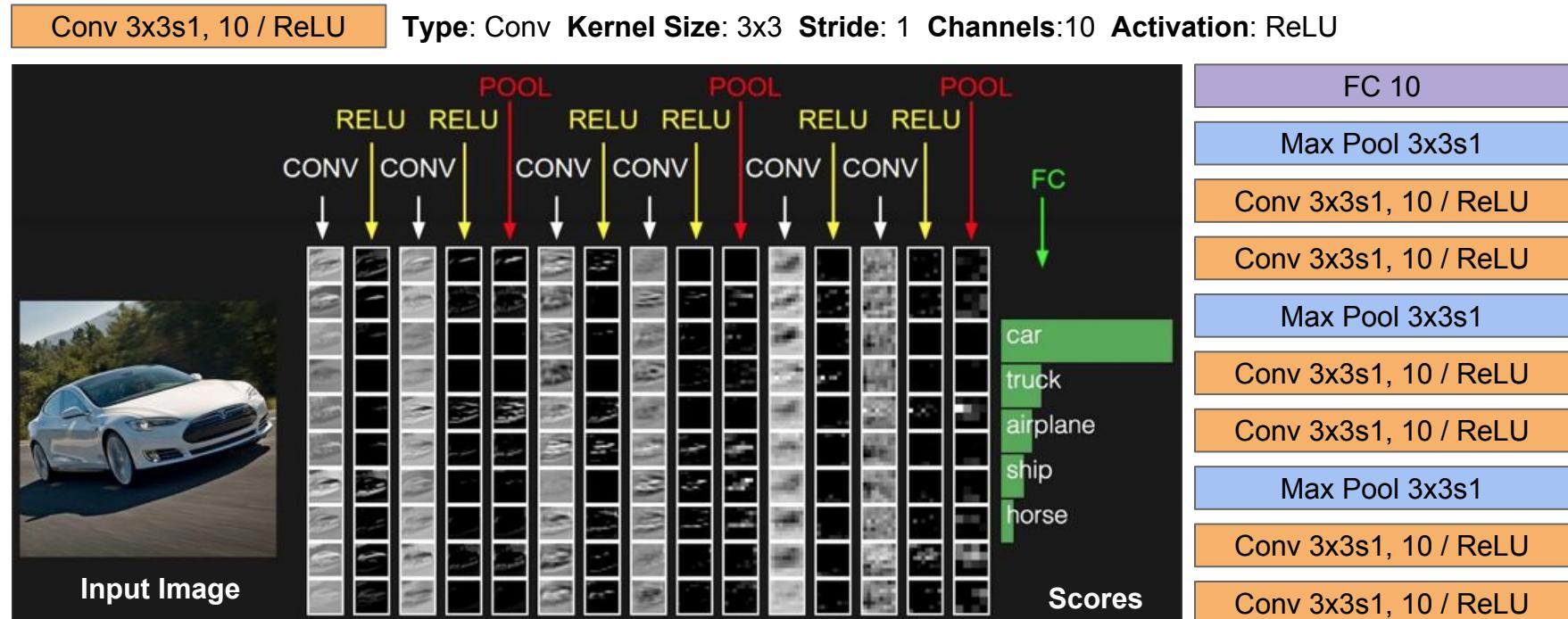


# Layer Review

- layers compute differentiable transformations
- types of layers: **conv**, **ReLU**, pool, FC
- parameters (conv, FC) or not (pool, ReLU)
- arguments like kernel size, stride, etc. (conv, pool) or not (Sigmoid)

# Convnet Architecture

Stack convolution, non-linearity, and pooling until global FC layer classifier



[figure credit A. Karpathy]

# See a Net Learn to See

Let's watch a convnet as it learns  
how to recognize objects in images

[Try It Out](#)

# Part 1: Deep Learning for Vision

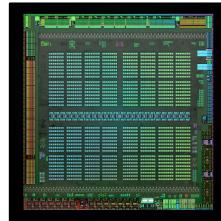
Why Deep Learning?



Dive into Deep Learning



What is DL?  
Why Now?



The Challenge  
of Recognition



Learning &  
Optimization

$$\nabla_{\theta} L$$

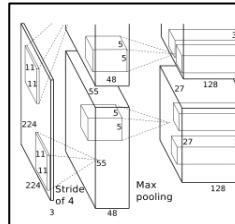
Applications



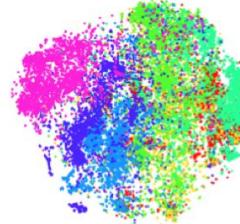
Caffe First Sip



Network Tour



Transfer Learning



# Share a Sip of Brewed Models

[demo.caffe.berkeleyvision.org](http://demo.caffe.berkeleyvision.org)

demo code open-source and bundled



	Maximally accurate	Maximally specific
cat		1.80727
domestic cat		1.74727
feline		1.72787
tabby		0.99133
domestic animal		0.78542

# Scene Recognition <http://places.csail.mit.edu/>



## Predictions:

- **Type of environment:** outdoor
- **Semantic categories:** skyscraper:0.69, tower:0.16, office\_building:0.11,
- **SUN scene attributes:** man-made, vertical components, natural light, open area, nohorizon, glossy, metal, wire, clouds, far-away horizon

# Visual Style Recognition

Karayev et al. *Recognizing Image Style*. BMVC14. Caffe fine-tuning example.

Demo online at <http://demo.vislab.berkeleyvision.org/> (see Results Explorer).

Ethereal



HDR



Melancholy



Minimal



Other Styles:

[Vintage](#)

[Long Exposure](#)

[Noir](#)

[Pastel](#)

[Macro](#)

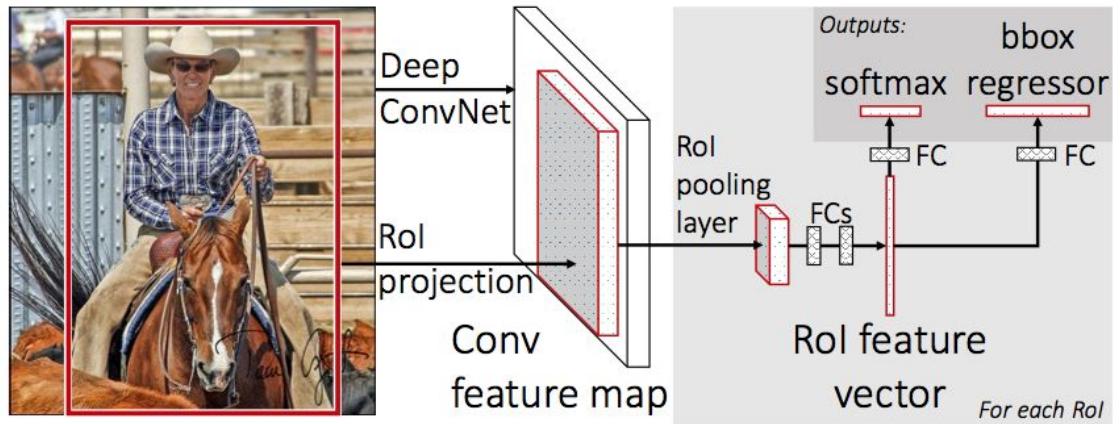
... and so on.

# Object Detection

R-CNNs: Region-based Convolutional Networks

## Fast R-CNN

- convnet for features
- proposals for detection



## Faster R-CNN

- end-to-end proposals and detection
- image inference in 200 ms
- Region Proposal Net + Fast R-CNN

papers + code online

Ross Girshick, Shaoqing Ren, Kaiming He, Jian Sun

# Pixelwise Prediction

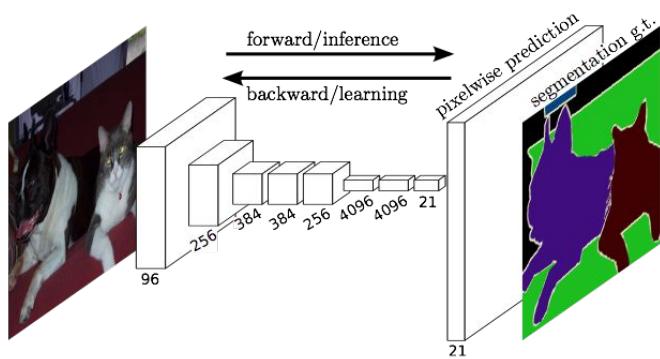
Fully convolutional networks for pixel prediction  
in particular semantic segmentation

- end-to-end learning
- efficient inference and learning  
100 ms per-image prediction
- multi-modal, multi-task

## Applications

- semantic segmentation
- denoising
- depth estimation
- optical flow

CVPR'15 [paper](#) and [code + models](#)

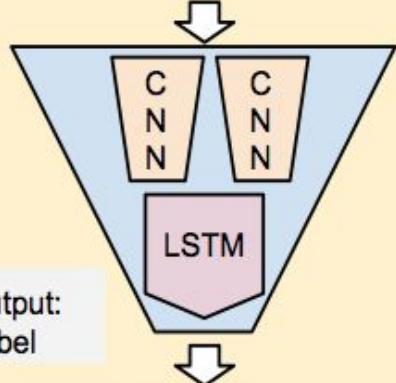


*Jon Long\* & Evan Shelhamer\*,  
Trevor Darrell. CVPR'15*

# Visual Sequence Tasks

## Activity Recognition

Input:  
Sequence  
of Frames



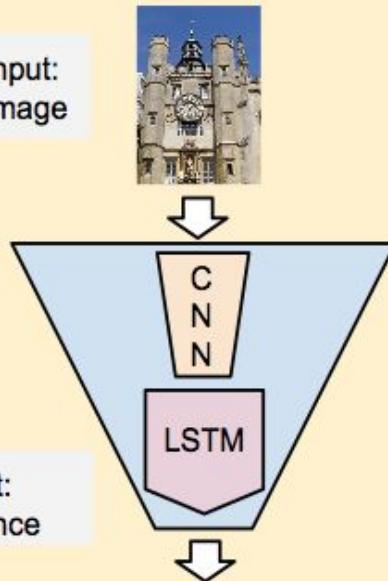
## Image Description

Input:  
Image



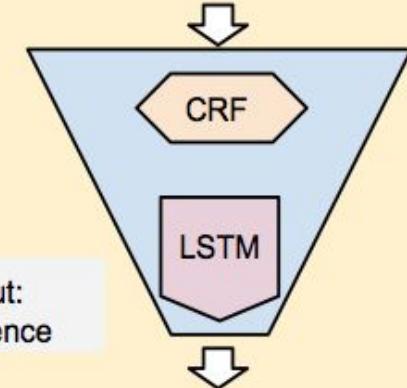
Output:  
Sentence

A large building with a  
clock on the front of it



## Video Description

Input:  
Video

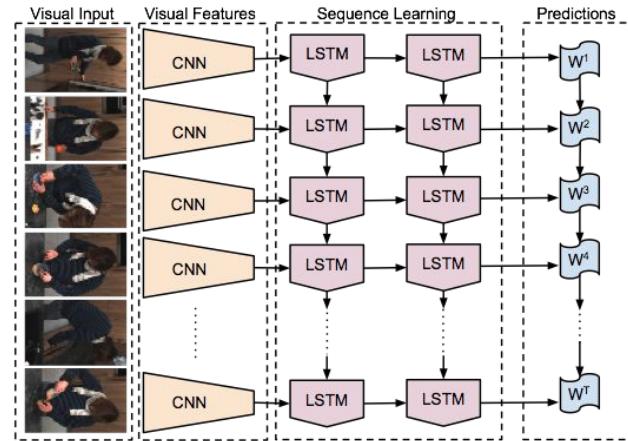


# Recurrent Networks for Sequences

Recurrent Nets and Long Short Term Memories (LSTM)  
are sequential models

- video
- language
- dynamics

learned by backpropagation through time



LRCN: Long-term Recurrent Convolutional Network

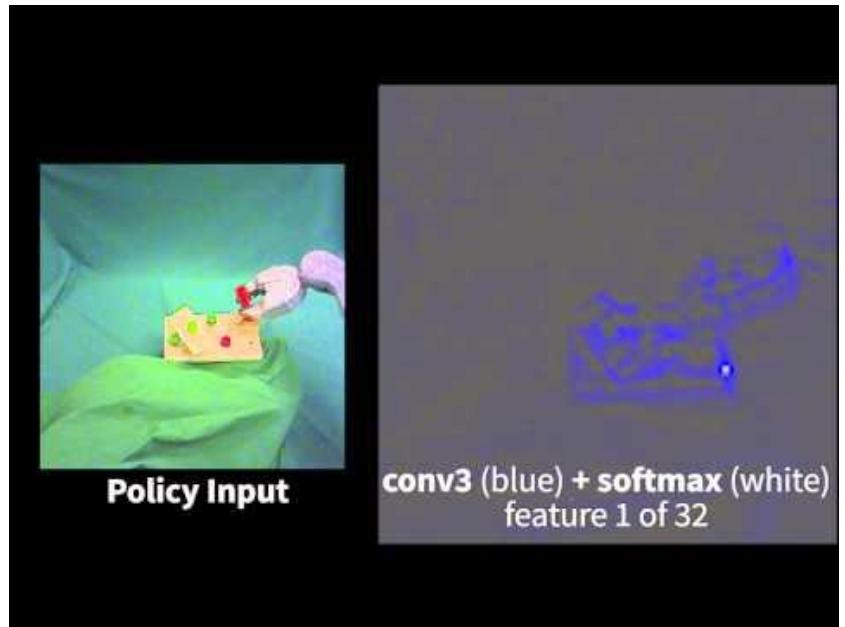
- activity recognition (sequence-in)
- image captioning (sequence-out)
- video captioning (sequence-to-sequence)

**LRCN:**  
recurrent + convolutional  
for visual sequences

# Deep Visuomotor Control

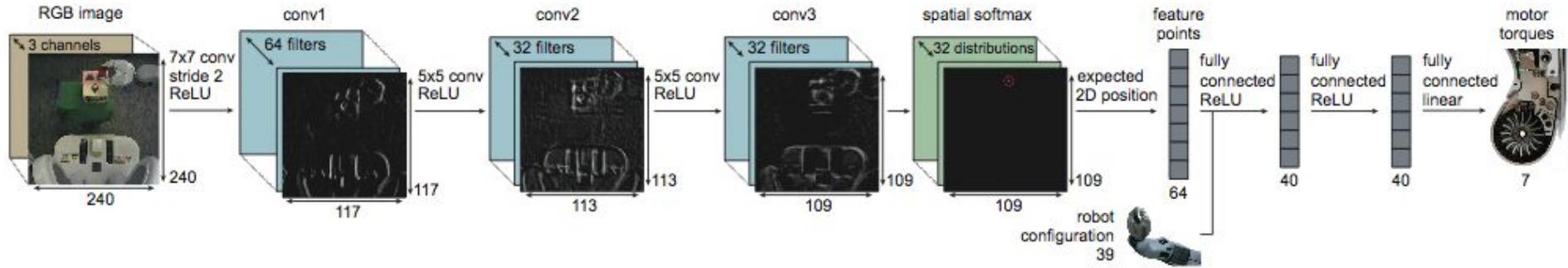


example experiments



feature visualization

# Deep Visuomotor Control Architecture



- multimodal (images & robot configuration)
- runs at 20 Hz - mixed GPU & CPU for real-time control

[paper](#) + [code](#) for guided policy search

Sergey Levine\* & Chelsea Finn\*,  
Trevor Darrell, and Pieter Abbeel

# Embedded Caffe

Caffe runs on embedded CUDA hardware and mobile devices

- same model weights,  
same framework interface
- out-of-the-box on  
CUDA platforms
- OpenCL port  
thanks Fabian Tschopp!  
+ AMD, Intel, and the community
- community Android port  
thanks sh1r0!



CUDA [Jetson TX1, TK1](#)



[OpenCL branch](#)

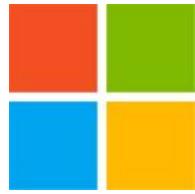


Android [lib](#), [demo](#)



[Ristretto toolkit](#)  
for approximation

# Caffeinated Companies



SIEMENS



... startups, big companies, more ...

# Caffe at Facebook

- in production for **vision at scale**: uploaded photos run through Caffe
- **Automatic Alt Text** for the blind
- **On This Day** for surfacing memories
- objectionable content detection
- contributing back to the community: inference tuning, tools, code review include [fb-caffe-exts](#) thanks Andrew!



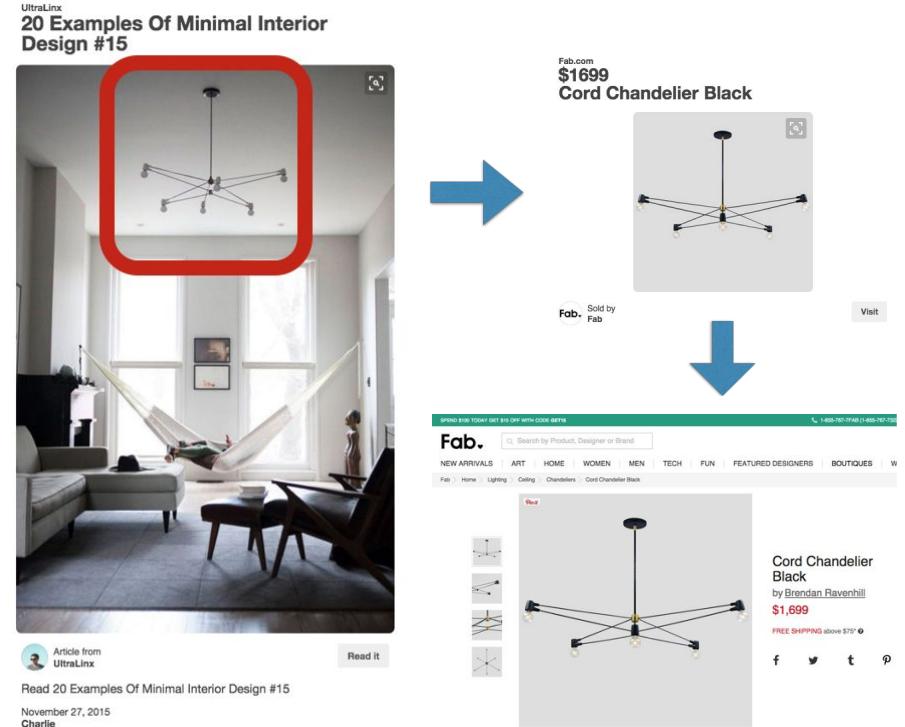
**On This Day**  
highlight content



**Automatic Alt Text**  
recognize photo content  
for accessibility

# Caffe at Pinterest

- in production for **vision at scale**: uploaded photos run through Caffe
- deep learning for visual search: **retrieval over billions of images** in <250 ms
- ~4 million requests/day
- built on an open platform of Caffe, FLANN, Thrift, ...



[example credit Andrew Zhai, Pinterest]

# Caffe at Yahoo! Japan

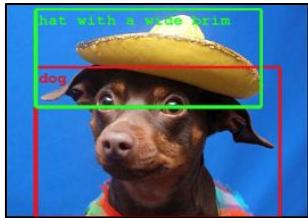
- curate news and restaurant photos for recommendation
- arrange user photo albums



**News Image Recommendation**  
select and crop images for news

# Part 1: Deep Learning for Vision

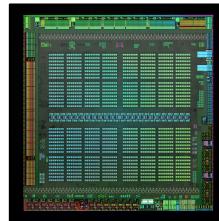
Why Deep Learning?



Dive into  
Deep Learning



What is DL?  
Why Now?



The Challenge  
of Recognition



Learning &  
Optimization

$$\nabla_{\theta} L$$

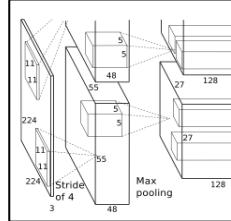
Applications



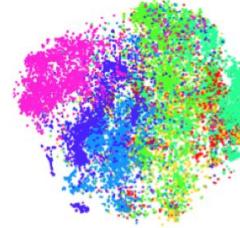
Caffe First Sip



Network Tour



Transfer Learning

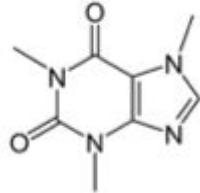


# First Sip of Caffe

We'll be using **Caffe** for our hands-on experience

[see notebook](#)

# Coffee Break (15 minutes)



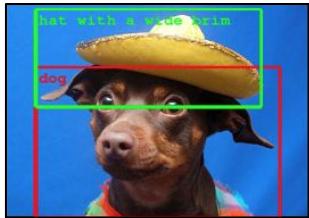
[caffe.berkeleyvision.org](http://caffe.berkeleyvision.org)

 [github.com/BVLC/caffe](https://github.com/BVLC/caffe)



# Part 1: Deep Learning for Vision

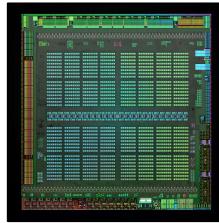
Why Deep Learning?



Dive into Deep Learning



What is DL?  
Why Now?



The Challenge  
of Recognition



Learning &  
Optimization

$$\nabla_{\theta} L$$

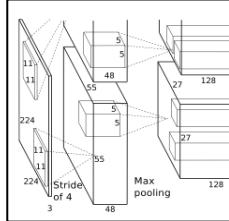
Applications



Caffe First Sip



Network Tour



Transfer Learning

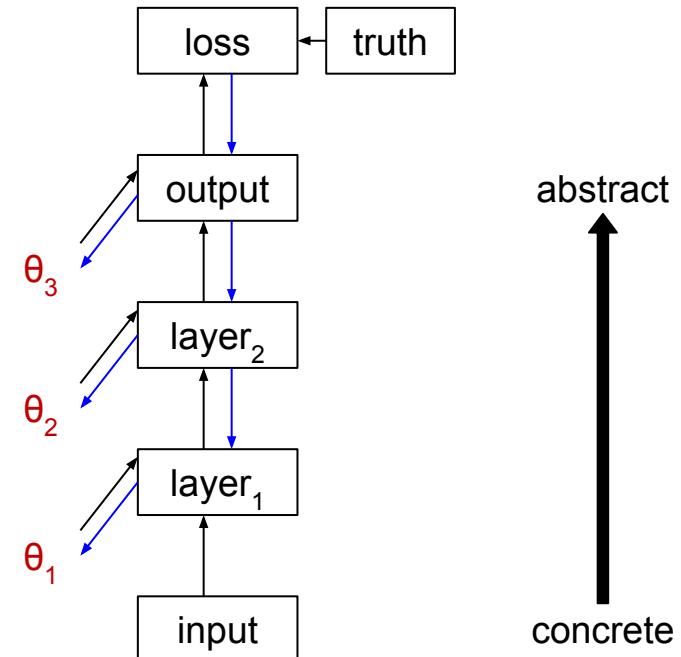


# What is Deep Learning?

Compositional Models  
Learned End-to-End

## Hierarchy of Representations

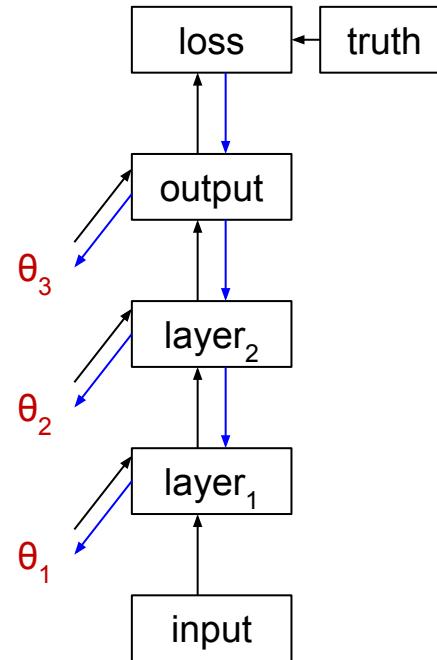
- vision: pixel, motif, part, object
- text: character, word, clause, sentence
- speech: audio, band, phone, word



# What is Deep Learning?

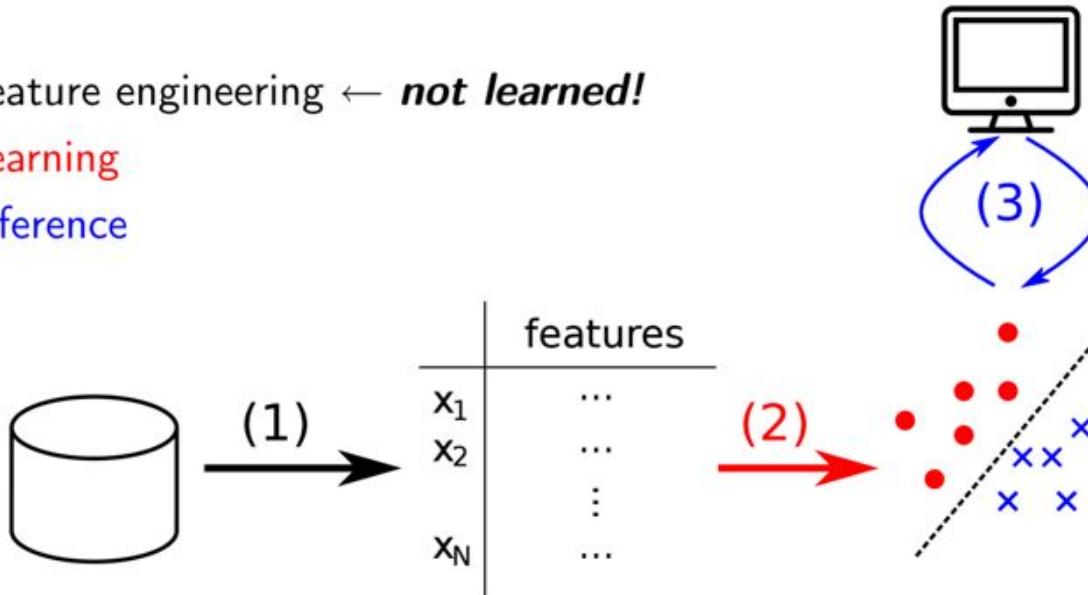
Compositional Models  
Learned End-to-End

**Back-propagation** jointly learns  
all of the **model parameters** to  
optimize the output for the task  
—more on this later!



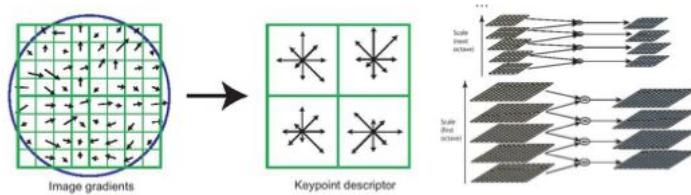
# Shallow Learning

1. Feature engineering  $\leftarrow$  ***not learned!***
2. **Learning**
3. Inference

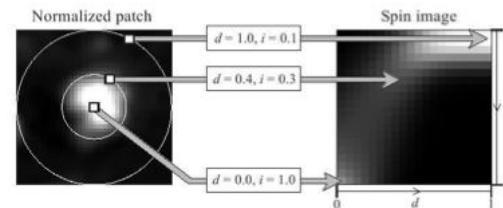


Separation of hand engineering and machine learning

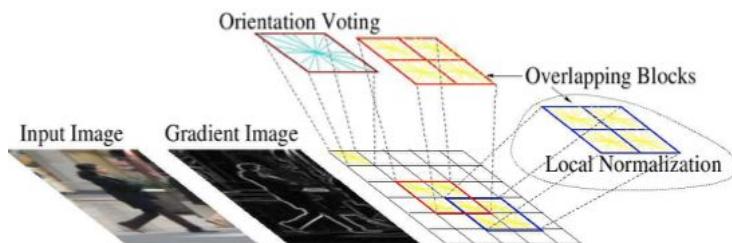
# Hand-Engineered Features



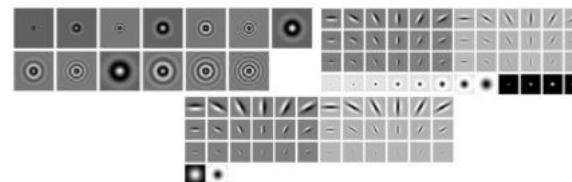
SIFT



Spin image



HoG



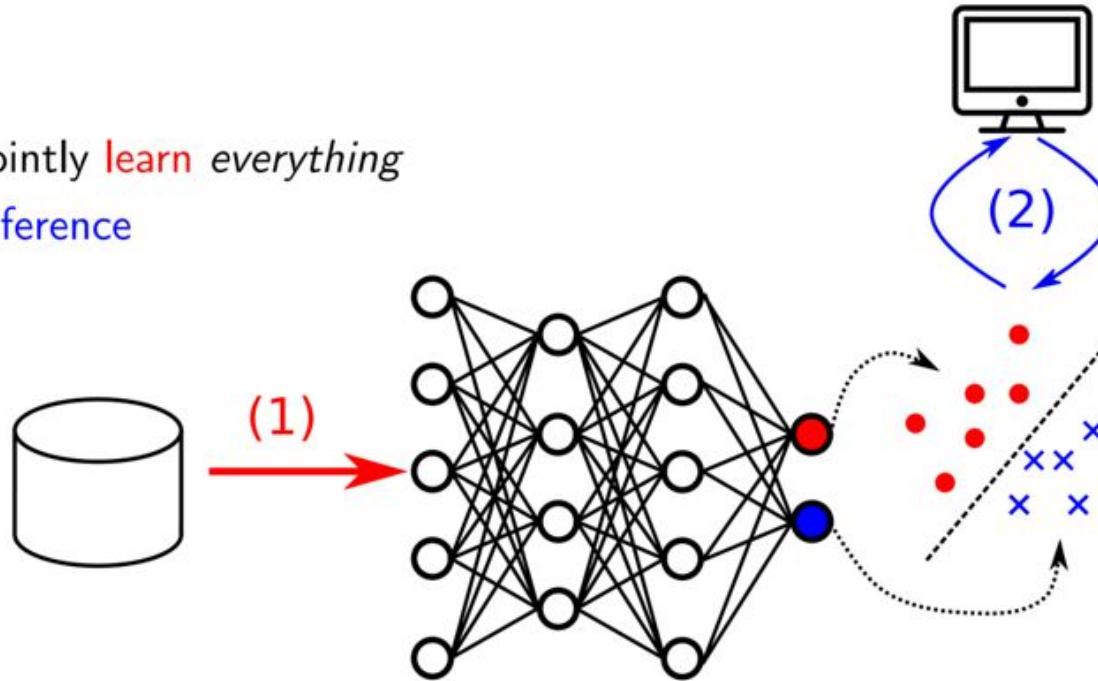
Textons

[figure credit R. Fergus]

Features from years of vision expertise by the whole community are now surpassed by *learned* representations and these *transfer across tasks*

# Deep Learning

1. Jointly **learn** *everything*
2. **Inference**



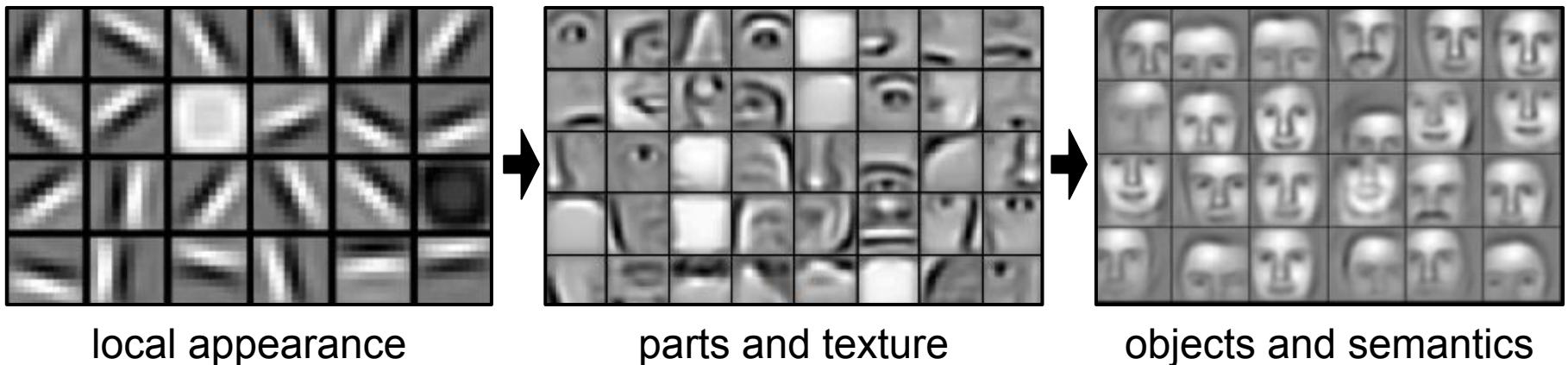
*The data decides* –Yoshua Bengio

[slide credit K. Cho]

80

# End-to-End Learning Representations

The visual world is too vast and varied  
to fully describe by hand



Learn the representation from data

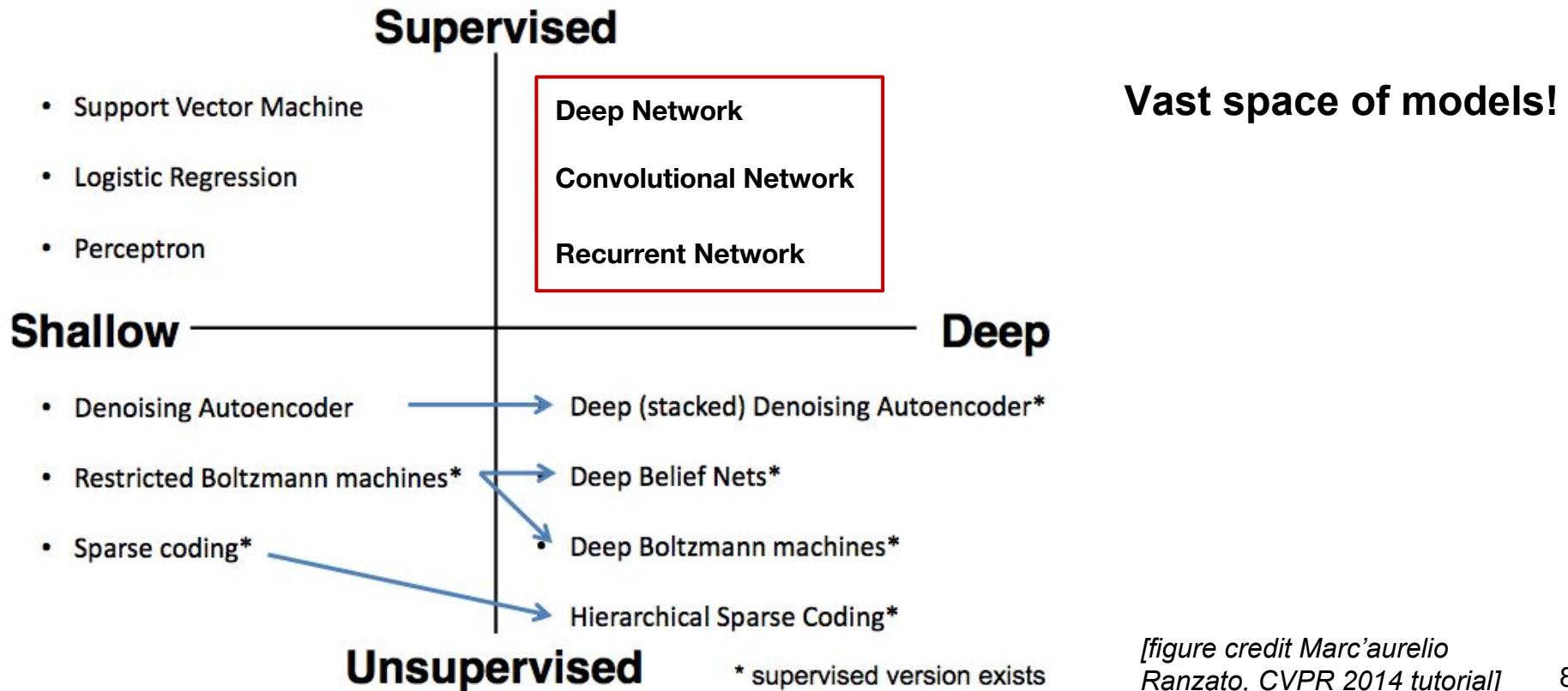
# End-to-End Learning Tasks

The visual world is too vast and varied  
to fully describe by hand



Learn the task from data

# Types of Learning

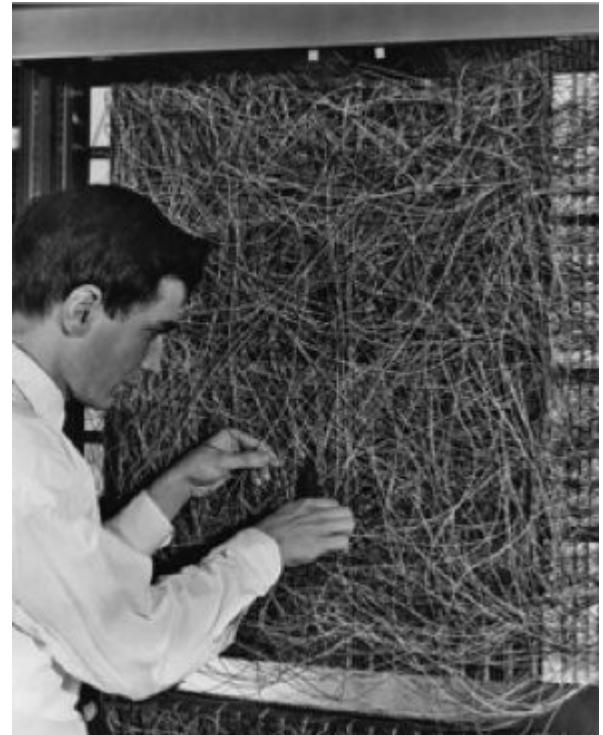


[figure credit Marc'aurelio Ranzato, CVPR 2014 tutorial]

# History

- 1958 Rosenblatt proposed perceptrons
- 1980 Neocognitron (Fukushima, 1980)
- 1982 Hopfield network, SOM (Kohonen, 1982), Neural PCA (Oja, 1982)
- 1985 Boltzmann machines (Ackley et al., 1985)
- 1986 Multilayer perceptrons and backpropagation (Rumelhart et al., 1986)
- 1988 RBF networks (Broomhead&Lowe, 1988)
- 1989 Autoencoders (Baldi&Hornik, 1989), Convolutional network (LeCun, 1989)
- 1992 Sigmoid belief network (Neal, 1992)
- 1993 Sparse coding (Field, 1993)
- 2000s Sparse, Probabilistic, and Layer-wise models (Hinton, Bengio, Ng)
- 2012 DL popularized in vision by contest victory (Krizhevsky et al. 2012)

Is deep learning 4, 20, or 50 years old? What's changed?



Rosenblatt's Perceptron

# Why Now?

## 1. Data

ImageNet et al.: millions of *labeled* (crowdsourced) images

## 2. Compute

GPUs: terabytes/s memory bandwidth, teraflops compute

## 3. Technique

new optimization know-how,

new variants on old architectures,

new tools for rapid experimentation

# Why Now? Data

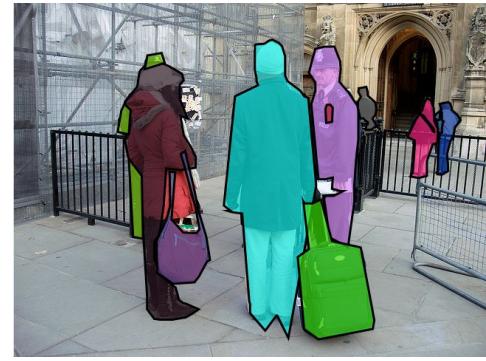
For example:



>10 million labeled images  
>1 million with *bounding boxes*

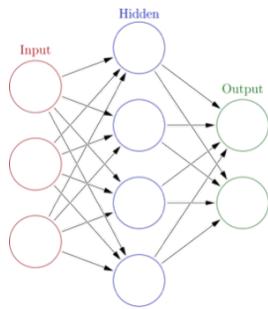


>300,000 images with *labeled and segmented* objects



# Why Now? GPUs

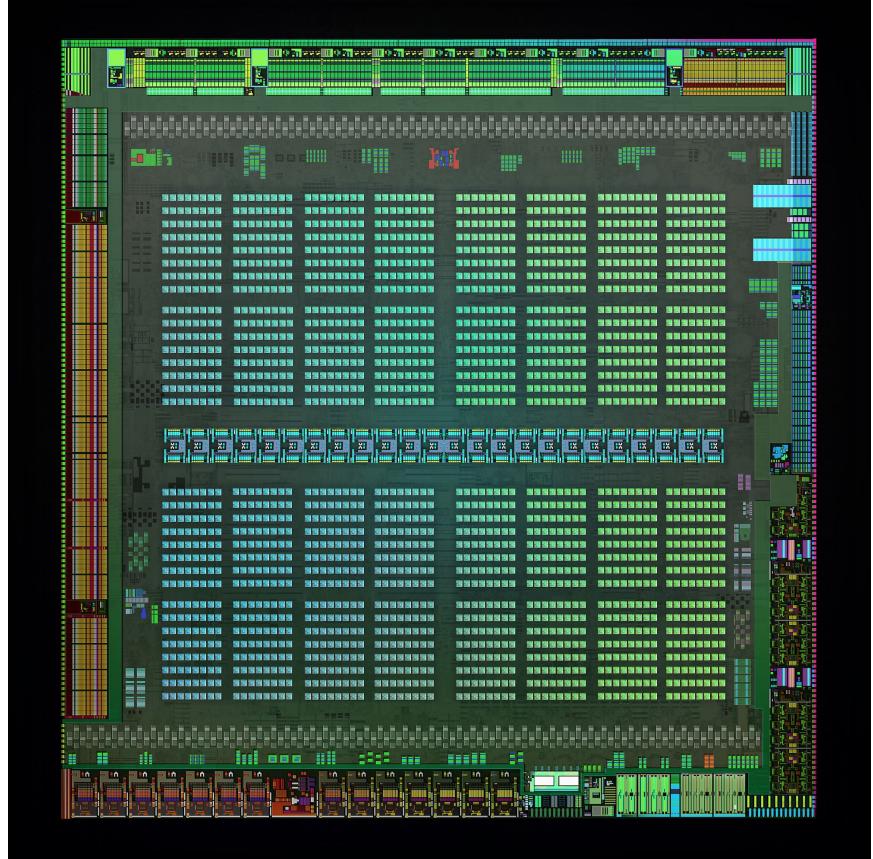
Parallel processors  
for parallel models:



**Inherent Parallelism**  
same op, different data

**Bandwidth**  
lots of data in and out

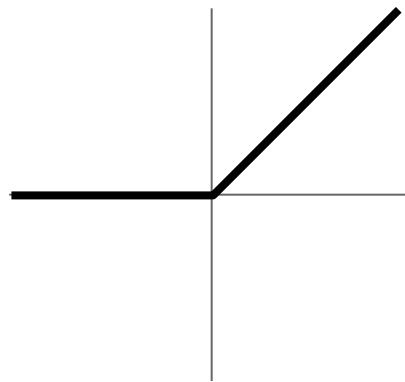
**Tuned Primitives**  
cuDNN and cuBLAS  
for deep nets                    for matrices



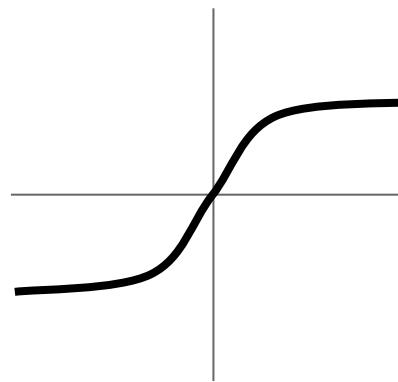
# Why Now? Technique

Non-convex and high-dimensional learning is okay  
with the right design choices

e.g. non-saturating non-linearities

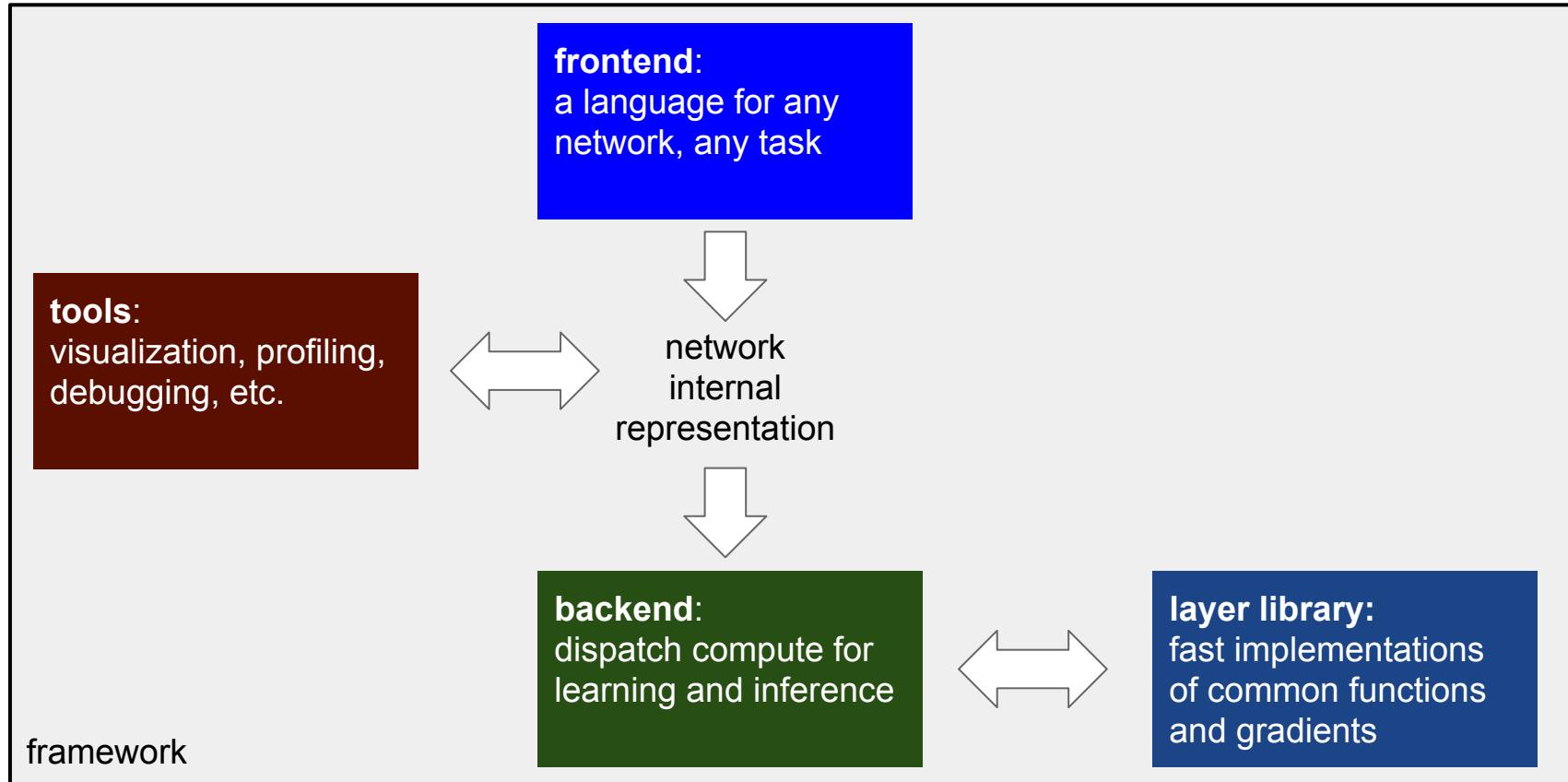


instead of

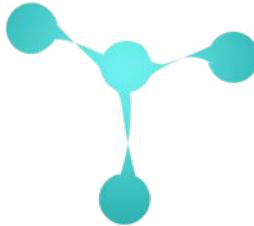
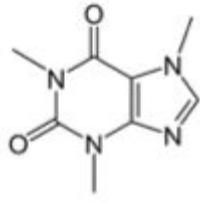


Learning by Stochastic Gradient Descent (SGD) with momentum and other variants — more later!

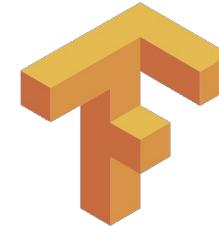
# Why Now? Deep Learning Frameworks



# Deep Learning Frameworks



theano



## Caffe

Berkeley / BVLC  
C++ / CUDA,  
Python, MATLAB

## Torch

Facebook + NYU  
Lua (C++)

## Theano

U. Montreal  
Python

## TensorFlow

Google  
Python (C++)

all open source  
we like to brew our networks with **Caffe**

# Not So “Neural”

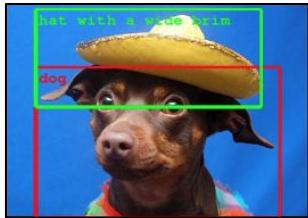
These models are not how the brain works  
*We don't know how the brain works!*



- This isn't a problem (except for neuroscientists)
- Be wary of neural realism hype or “it just works because it's like the brain”
- **network**, not neural network
- **unit**, not neuron

# Part 1: Deep Learning for Vision

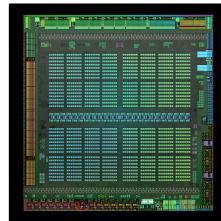
Why Deep Learning?



Dive into Deep Learning



What is DL?  
Why Now?



The Challenge  
of Recognition



Learning &  
Optimization

$$\nabla_{\theta} L$$

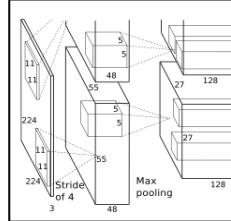
Applications



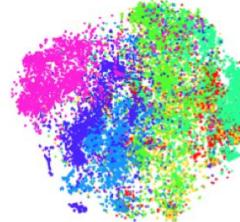
Caffe First Sip



Network Tour



Transfer Learning



# The Challenge of Recognition



What does this pattern depict?

# The Challenge of Recognition



What does this pattern depict?



Same data!

# The Challenge of Recognition



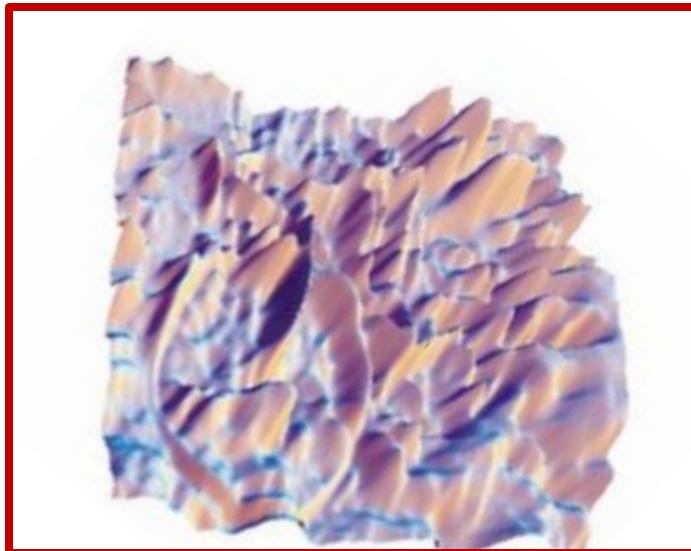
What does this pattern depict?  
*intensity as elevation*



Same data!  
*intensity as brightness*

# The Challenge of Recognition

## Machine Recognition

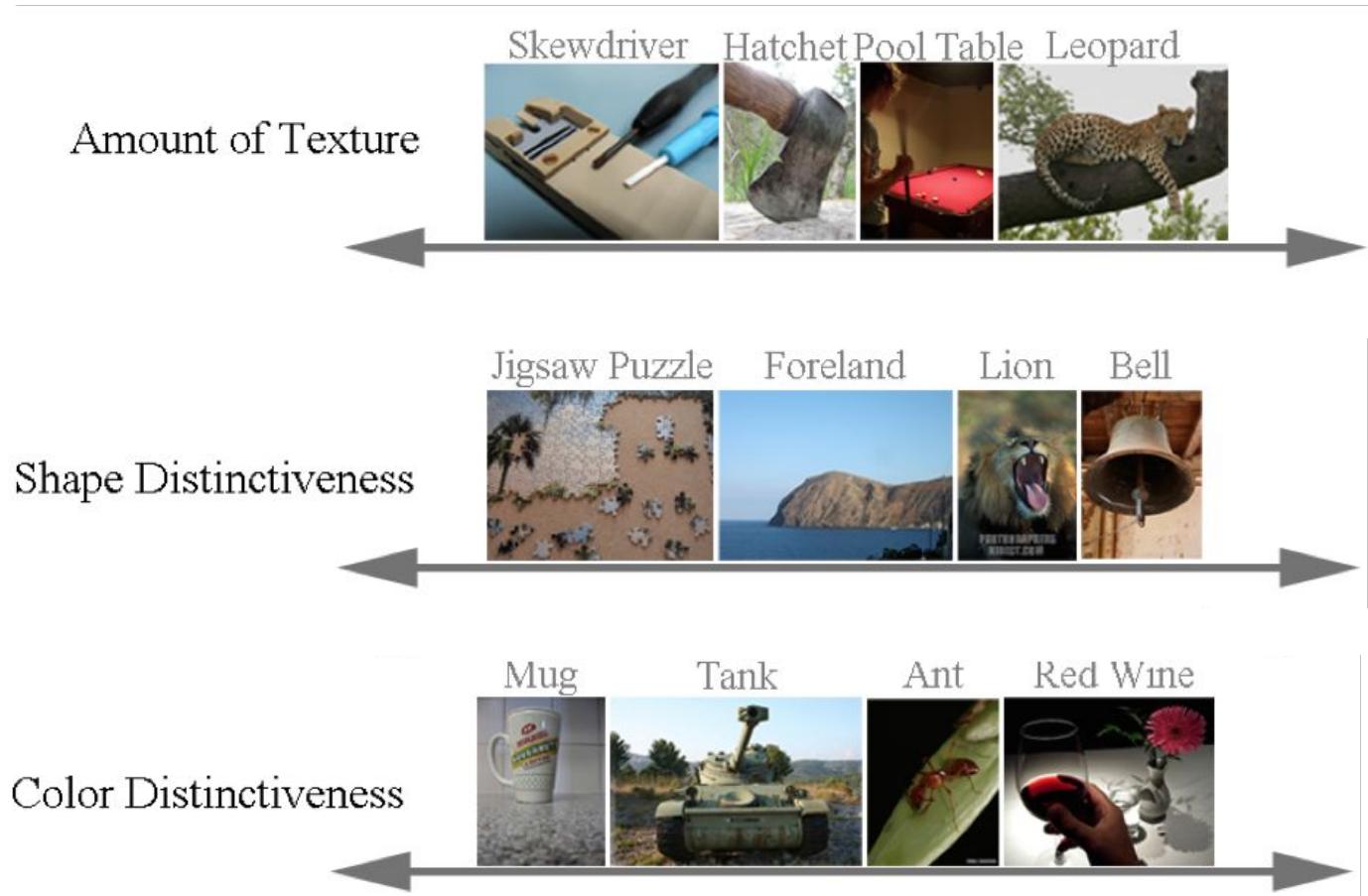


What does this pattern depict?  
*intensity as elevation*



Same data!  
*intensity as brightness*





## PASCAL



bird



cat



dog

## ILSVRC



flamingo

cock

ruffed grouse

quail

partridge

...



Egyptian cat

Persian cat

Siamese cat

tabby

lynx

...



dalmatian

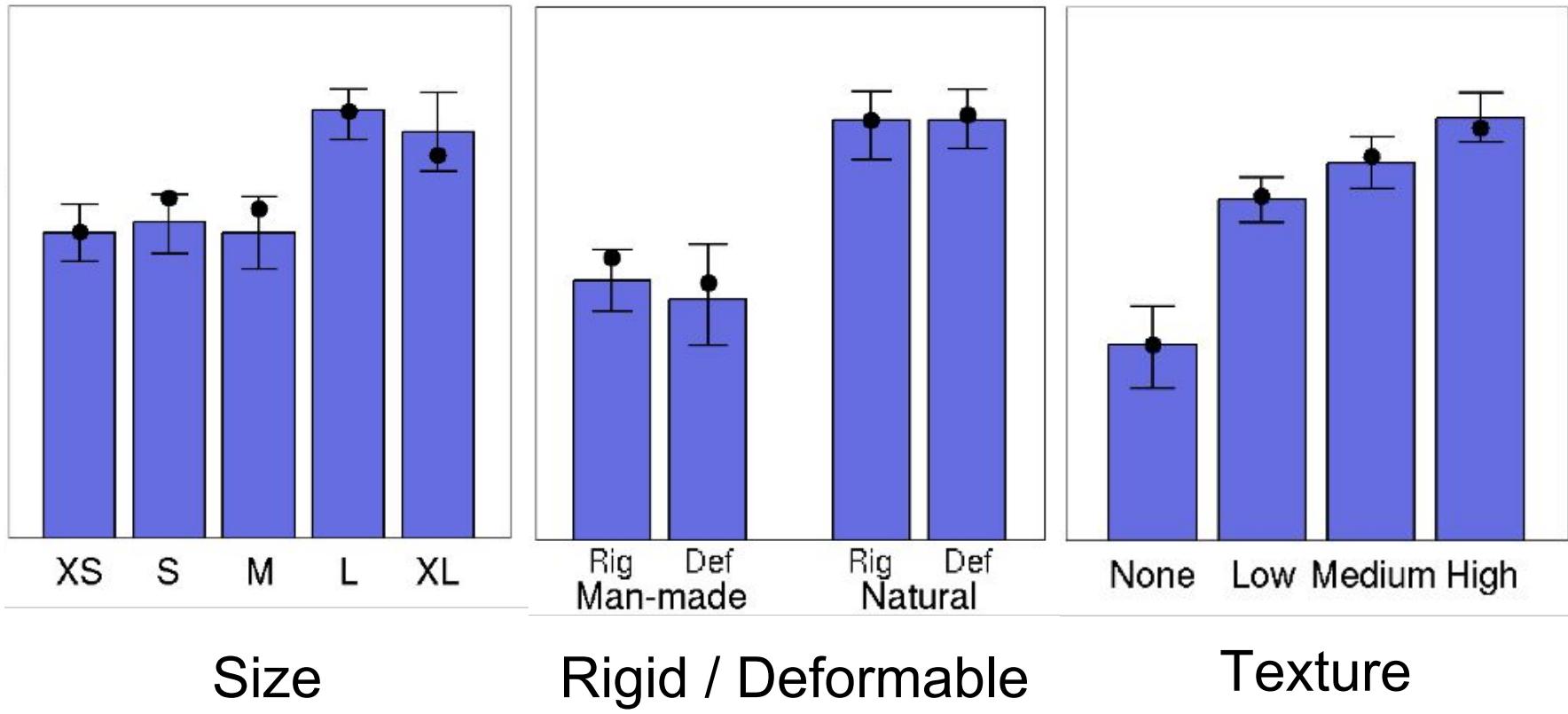
keeshond

miniature schnauzer

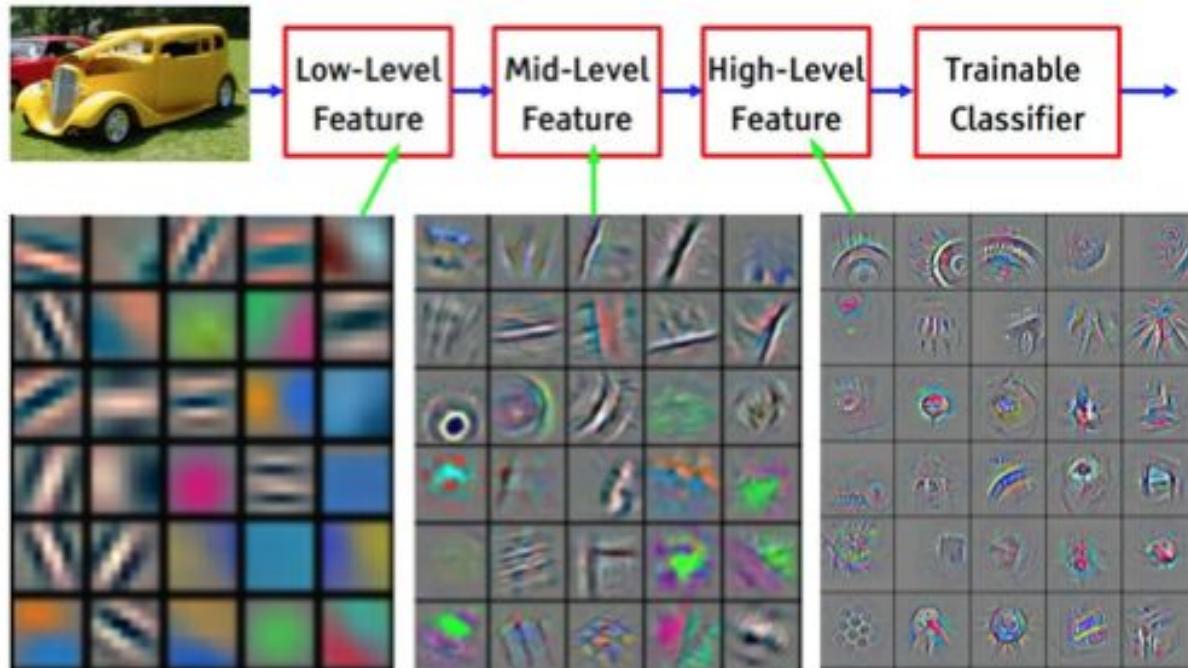
standard schnauzer

giant schnauzer

# Image Classification Accuracy (on ILSVRC)



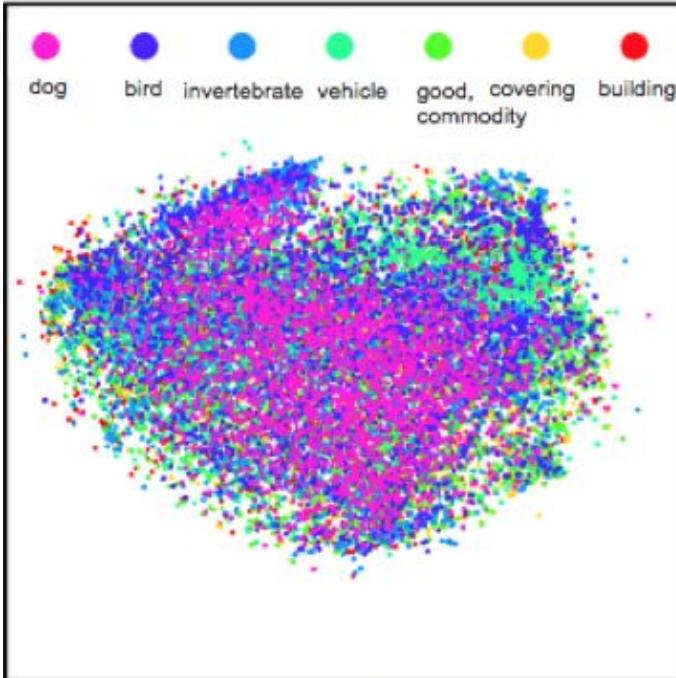
# What Does a Deep Net Learn?



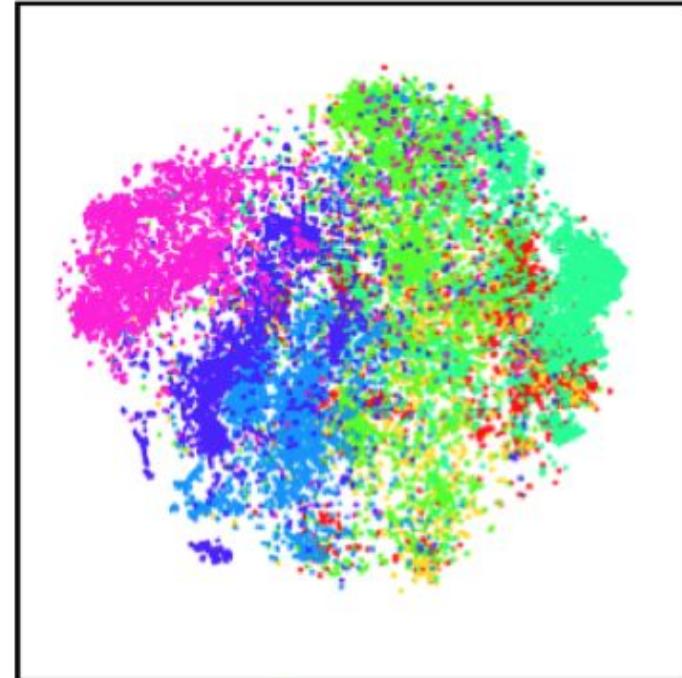
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# Feature Embedding

## The Unreasonable Effectiveness of Deep Features



Low-level: Pool<sub>1</sub>



High-level: FC<sub>6</sub>

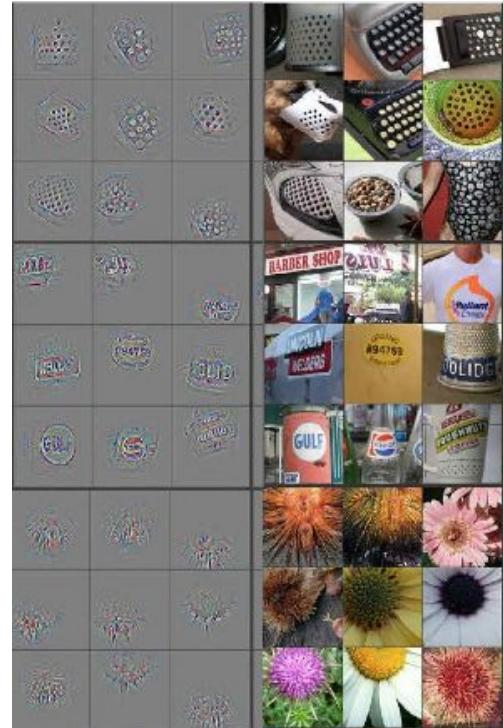
Classes separate in the deep representations and transfer to many tasks.  
[DeCAF] [Zeiler-Fergus]

# Feature Mining

Strongest Feature Responses are Visually Rich



[R-CNN]



Rich visual structure of features deep in hierarchy

# Shallow Layers Learn Simple Features

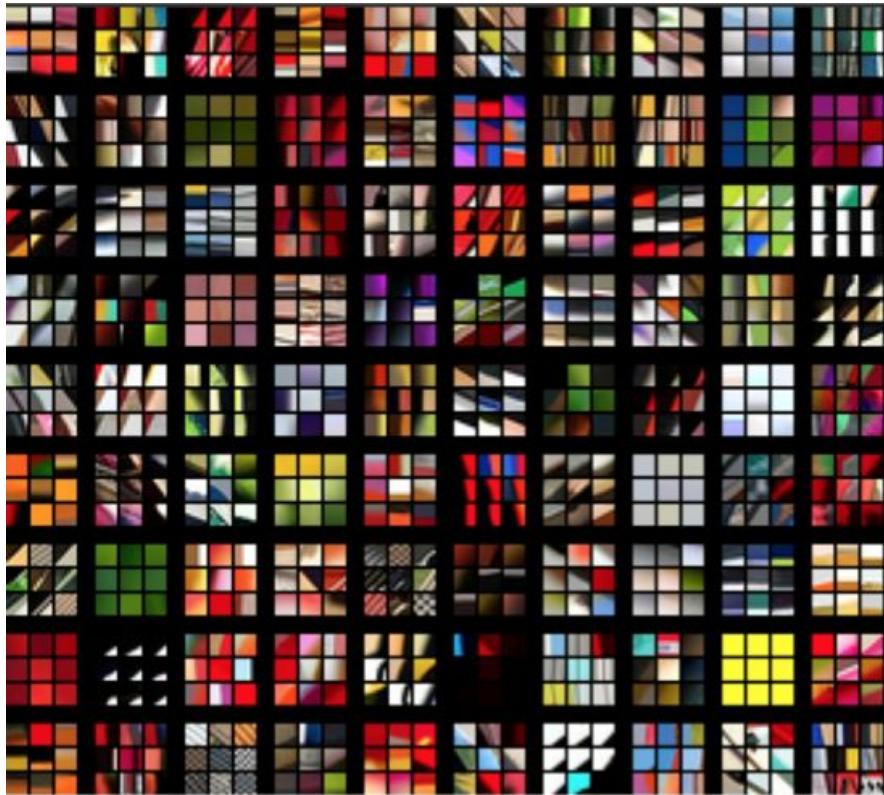
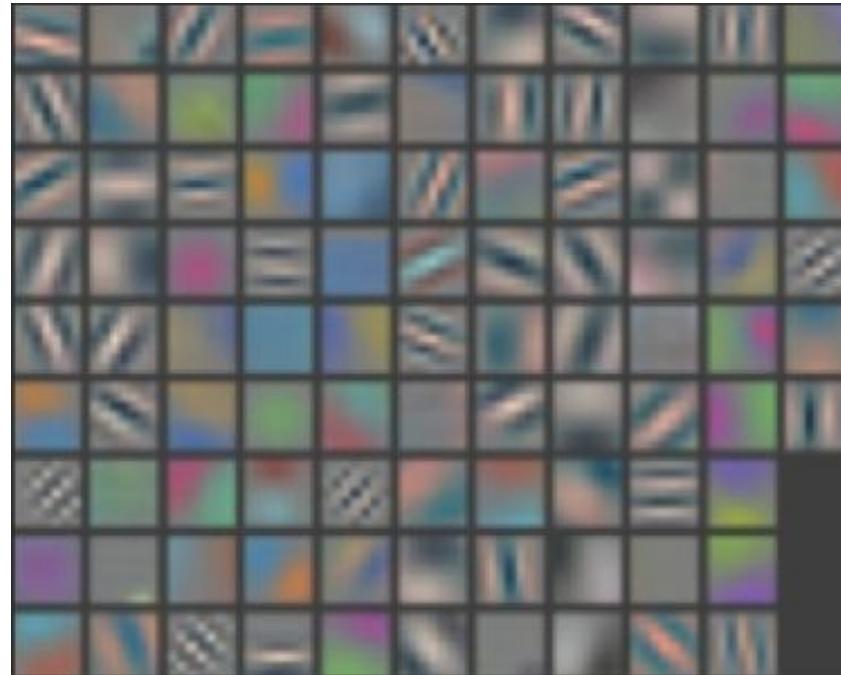


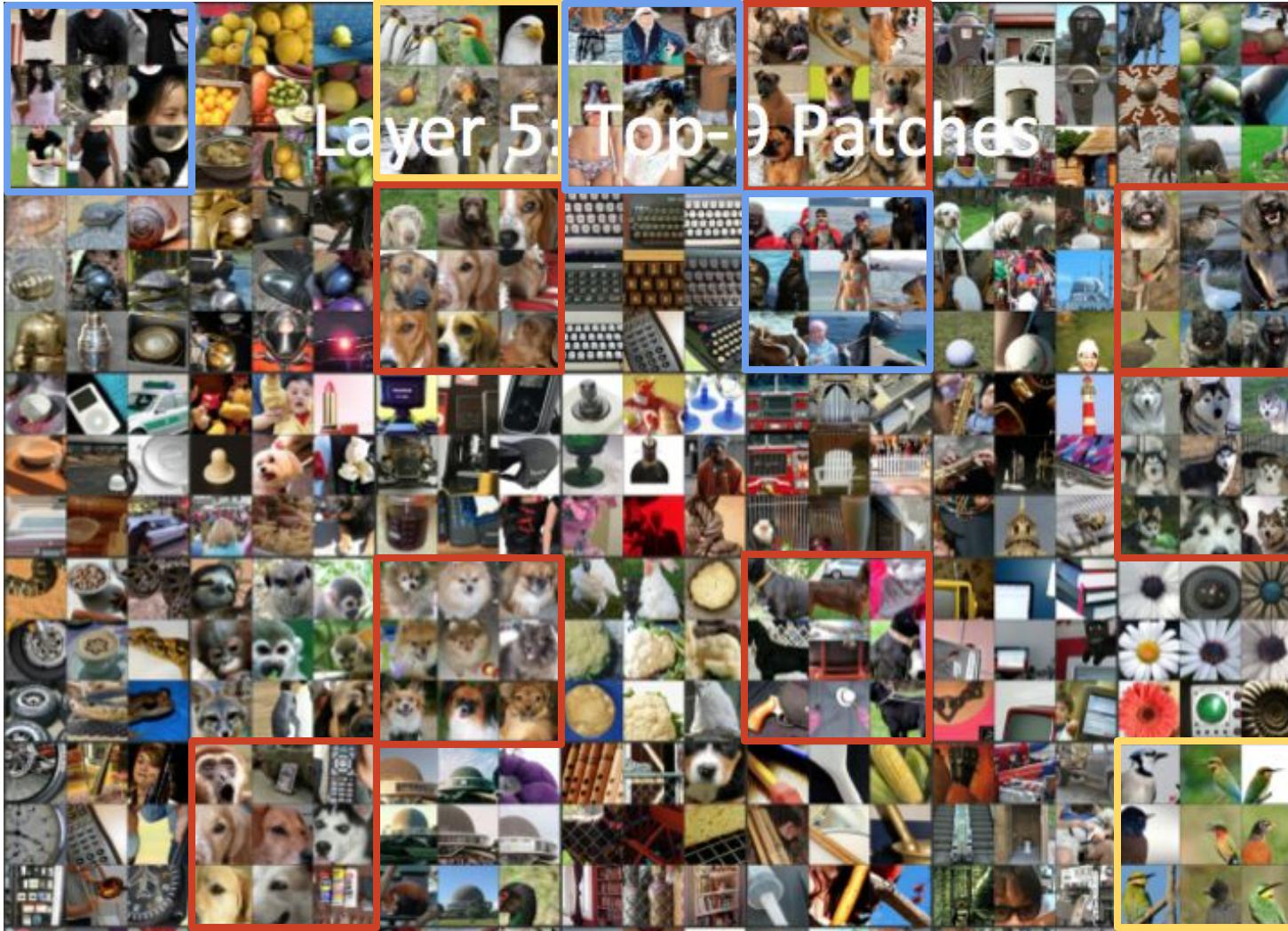
image patches that strongly activate 1st layer filters



$\text{conv}_1$  1st layer filters

[figure credit M. Zeiler]

## Layer 5: Top-9 Patches



$\text{conv}_5$

visual complexity  
increases from  
“shallow” to “deep”

dog person bird

deep responses have  
*semantic* consistency

[figure credit M. Zeiler]

# Part 1: Deep Learning for Vision

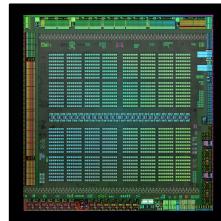
Why Deep Learning?



Dive into Deep Learning



What is DL?  
Why Now?



The Challenge  
of Recognition



Learning &  
Optimization

$$\nabla_{\theta} L$$

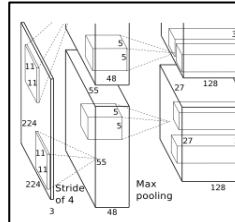
Applications



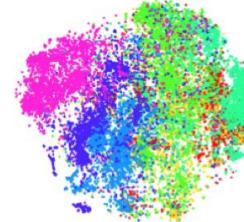
Caffe First Sip



Network Tour



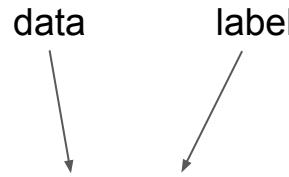
Transfer Learning



# Supervised Learning

Given labeled data:

$$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$$



Goal: find a function  $f$  such that

$$y_n = f(x_n)$$

for all  $n$ , “as well as possible”

# Supervised Loss

What does “as well as possible” mean?

Pick a *loss function*  $\ell(y, \hat{y})$ : how wrong is it to predict  $\hat{y}$  when the true label is  $y$ ?

*Minimize* the total loss over all data:

$$L = \sum_n \ell(y_n, \hat{y}_n)$$

E.g.  $\ell(y, \hat{y}) = \|y - \hat{y}\|^2$  “Euclidean Loss” or everyday linear regression

# Parametric Learning

How do we find the label-prediction function  $f$ ?

Parametric answer: pick it from a family determined by a set of *parameters*  $\theta$ :

$$f(x) = f(x; \theta)$$

matrix      vector



E.g.  $f(x; \theta) = \theta x$  “linear prediction”

For us:  $f$  is a *network*,  $\theta$  is a set of *weights*

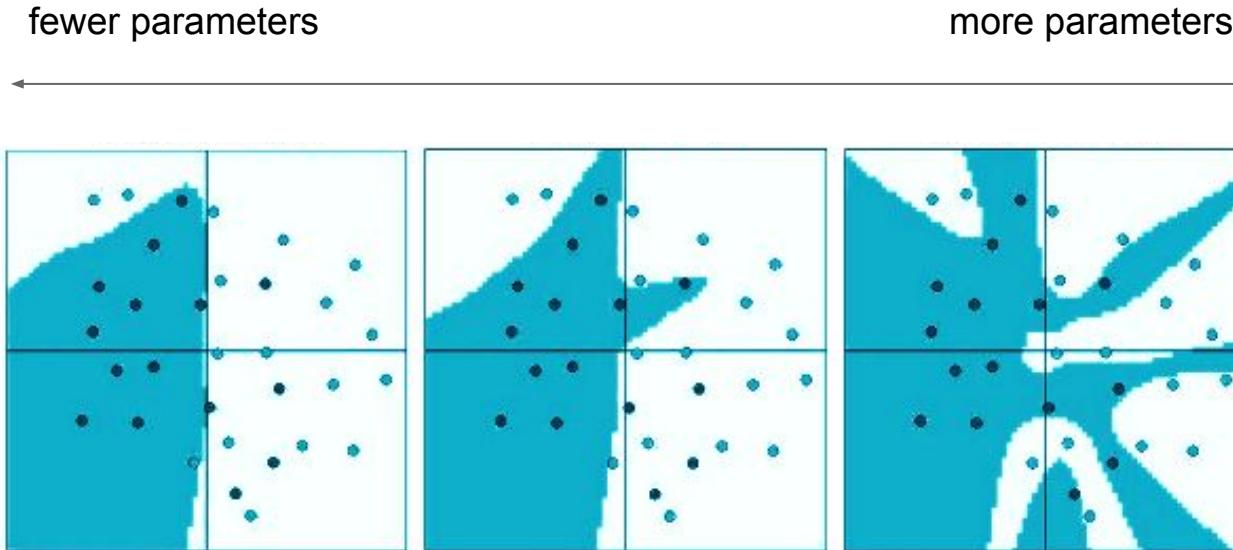
# Parametric Supervised Learning

Altogether: our goal is to find  $\theta$  in order to

$$\text{minimize } L(\theta) = \sum_n \ell(y_n, \hat{y}_n) = \sum_n \ell(y_n, f(x_n; \theta))$$

The diagram illustrates the components of the loss function  $L(\theta)$ . It shows the formula  $\text{minimize } L(\theta) = \sum_n \ell(y_n, \hat{y}_n) = \sum_n \ell(y_n, f(x_n; \theta))$ . Arrows point from "loss" to the first term  $\ell(y_n, \hat{y}_n)$ , from "true label" to  $y_n$ , and from "predicted label" to  $\hat{y}_n$ . Another arrow points from "sum over data" to the summation symbol. A double-headed arrow connects "model (network)" and "parameters (weights)".

# Underfitting and Overfitting

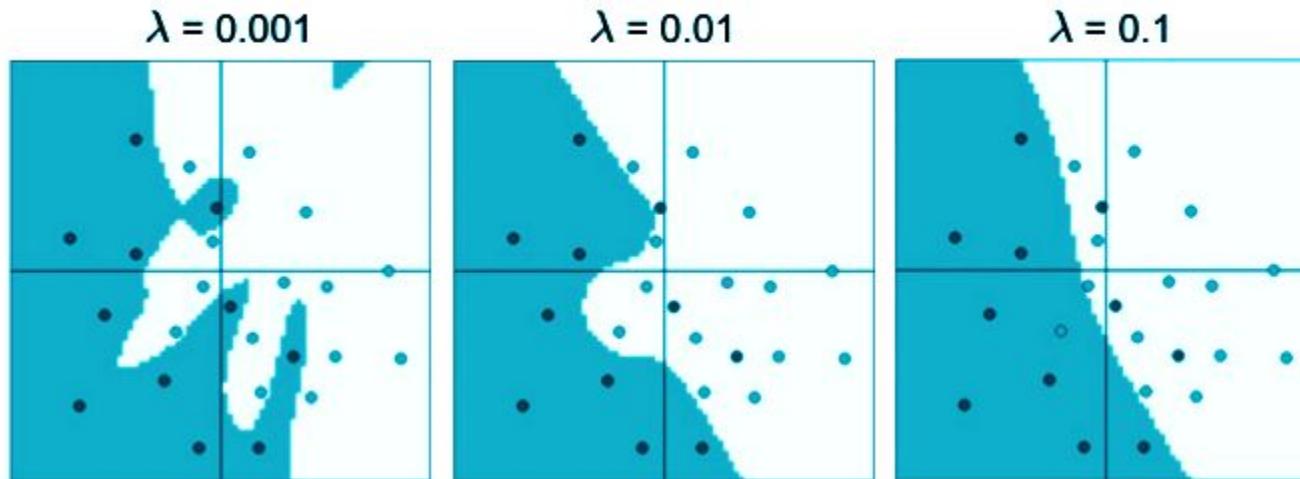


*underfitting:*  
not enough parameters  
to model the data

*overfitting:*  
enough parameters to  
memorize the training  
set without generalizing

# Regularization

How can we prevent overfitting without reducing the number of parameters?



Add a *regularization penalty* to our loss: “complicated” solutions are worse

# Regularization: Weight Decay and Dropout

Weight Decay: minimize  $L(\theta) + \lambda \|\theta\|^2$  to pull weights toward zero

$\lambda$  (scalar) is an optimization setting... pick it empirically  
aka “L<sup>2</sup> regularization”

Dropout: during training, randomly set a fraction  $p$  of activations to zero

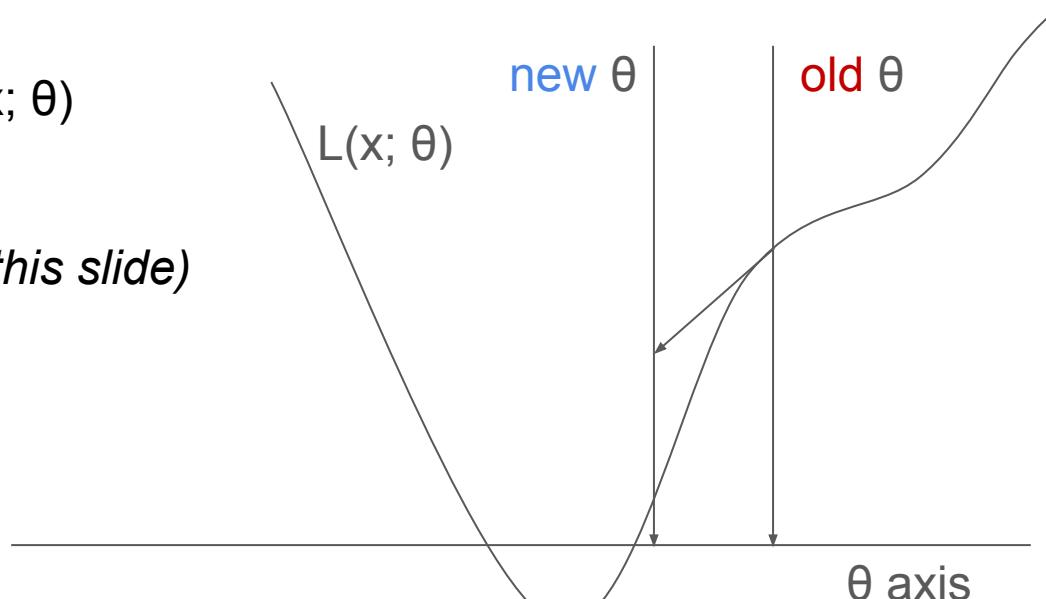
$p$  is an optimization setting (often 0.5)  
forces model to be robust to noise

# Gradient Descent: Intuition

Want to minimize “loss” function  $L(x; \theta)$

$\theta$  (vector): parameter to update

$x$  (vector): input data (*fixed on this slide*)

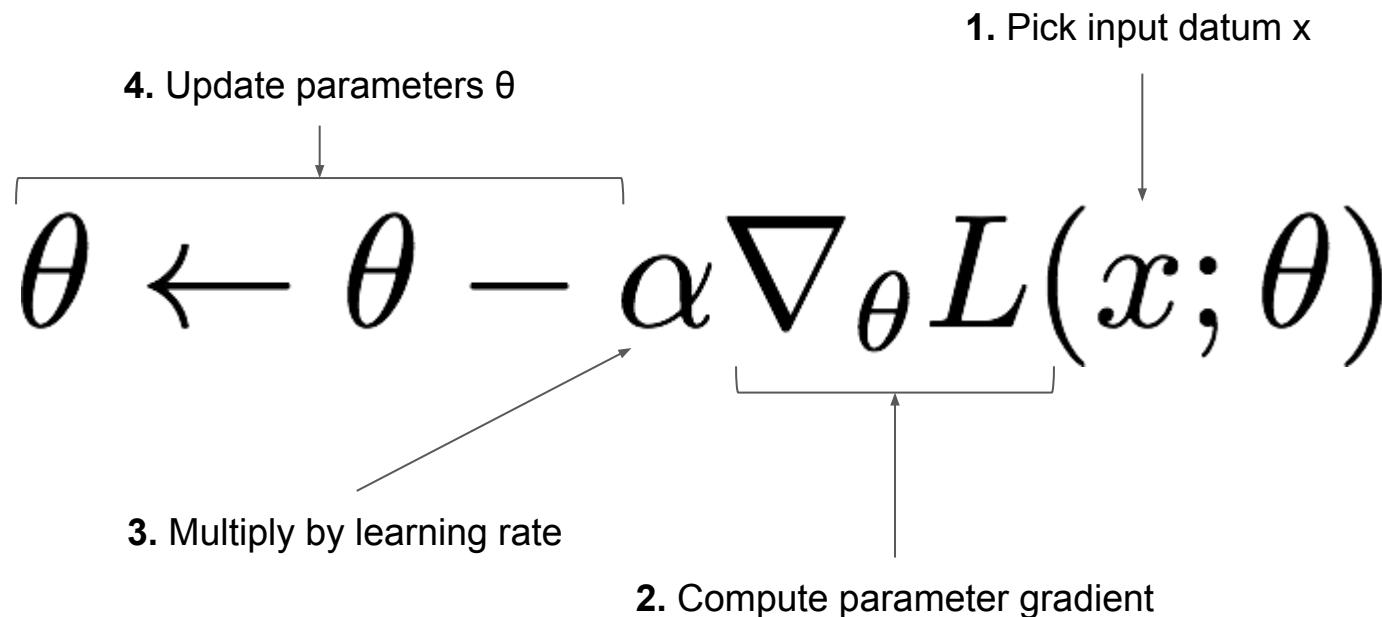


Move in the direction of the gradient

The gradient tells you, for each element of the network parameters, how the loss changes in response to a change in that parameter.

# Stochastic Gradient Descent (SGD)

Want to minimize “loss” function  $L(x; \theta)$



# Why “Stochastic”?

The gradient depends on the choice of input datum  $x$

Choose  $x$  *randomly* (or just cycle through all data in a fixed order)

(The alternative is to average the gradient over all available data,  
“batch gradient descent”:

$$\theta \leftarrow \theta - \alpha \sum_i \nabla_{\theta} L(x_i; \theta)$$

That's too slow for big data!)

# SGD with Weight Decay and Momentum

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} L(x; \theta)$$

# SGD with Weight Decay and Momentum

$$\theta \leftarrow \theta - \alpha (\nabla_{\theta} L(x; \theta) - \lambda \theta)$$



weight decay  
(*regularization*)

# SGD with Weight Decay and Momentum

$$\theta \leftarrow \theta - \alpha (\nabla_{\theta} L(x; \theta) - \lambda \theta) + p[\text{last update}]$$

weight decay  
(regularization)

momentum  
( $p$  is a number less than 1)

There are many other variants:

Adam, RMSprop, AdaDelta, AdaGrad, Nesterov, ...

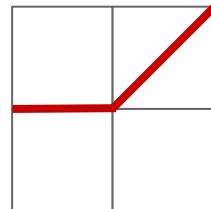
# Layer Gradients

## Matrix Multiply

$$y_i = \sum_j A_{ij} x_j$$

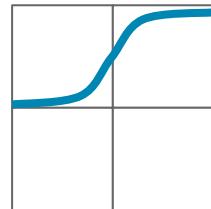
## ReLU

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



## Sigmoid

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



## Gradients

$$\frac{\partial y_i}{\partial x_j} = A_{ij} \quad \frac{\partial y_i}{\partial A_{ij}} = x_j$$

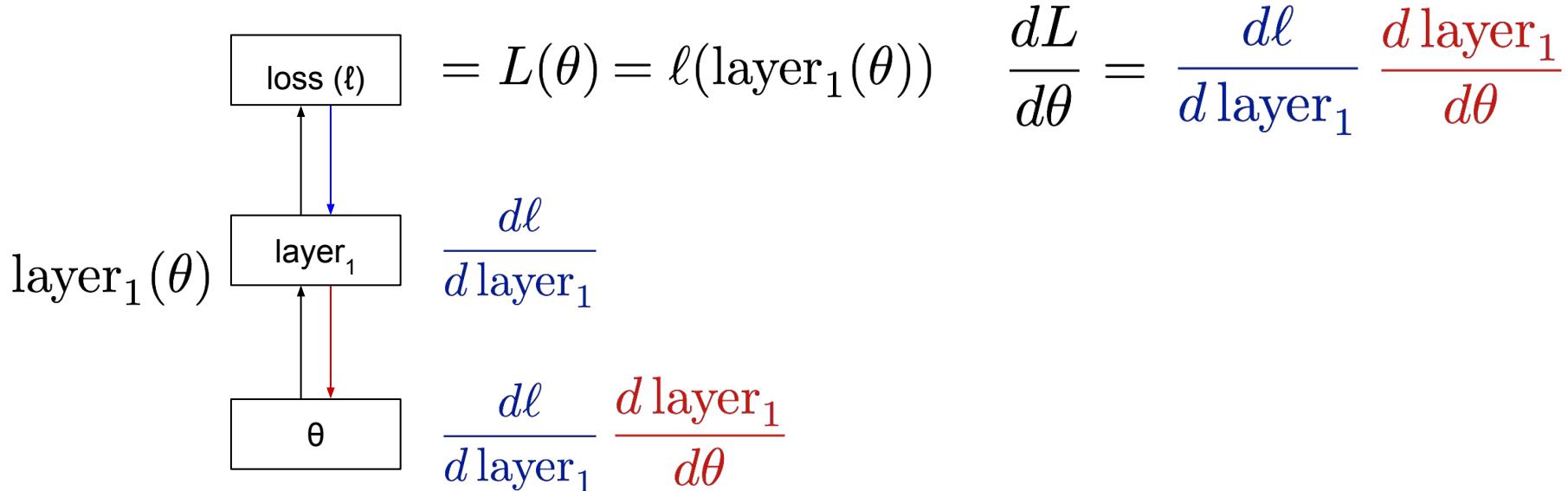
$$\frac{dy}{dx} = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$

$$\frac{dy}{dx} = e^{-x} y^2$$

# Back-propagation: The Chain Rule

A net is a composition of layer functions

The gradient of a net is the product of layer gradients



# Back-propagation in a Bigger Net

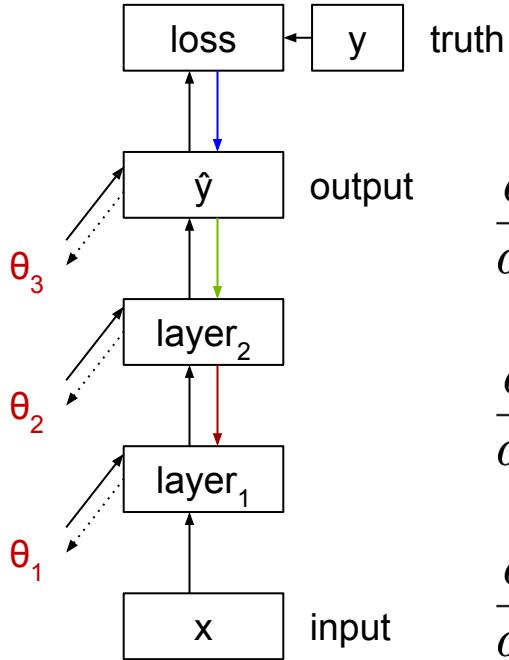
Forward pass

$$L = \text{loss}(\hat{y}, y)$$

$$\hat{y} = \text{layer}_3(x_2; \theta_3)$$

$$x_2 = \text{layer}_2(x_1; \theta_2)$$

$$x_1 = \text{layer}_1(x; \theta_1)$$



Backward pass

$$\frac{dL}{d\hat{y}}$$

$$\frac{dL}{d\theta_3} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{d\theta_3}$$

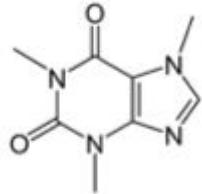
$$\frac{dL}{d\theta_2} = \frac{dL}{dx_2} \frac{dx_2}{d\theta_2}$$

$$\frac{dL}{d\theta_1} = \frac{dL}{dx_1} \frac{dx_1}{d\theta_1}$$

$$\frac{dL}{dx_2} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dx_2}$$

$$\frac{dL}{dx_1} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dx_1}$$

# Lunch Break (45 minutes)



[caffe.berkeleyvision.org](http://caffe.berkeleyvision.org)

 [github.com/BVLC/caffe](https://github.com/BVLC/caffe)



# Part 1: Deep Learning for Vision

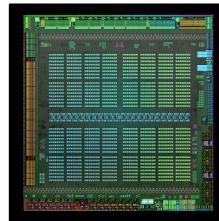
Why Deep Learning?



Dive into Deep Learning



What is DL?  
Why Now?



The Challenge  
of Recognition



Learning &  
Optimization

$$\nabla_{\theta} L$$

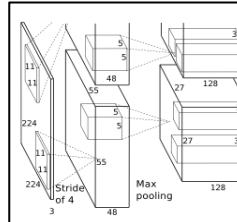
Applications



Caffe First Sip



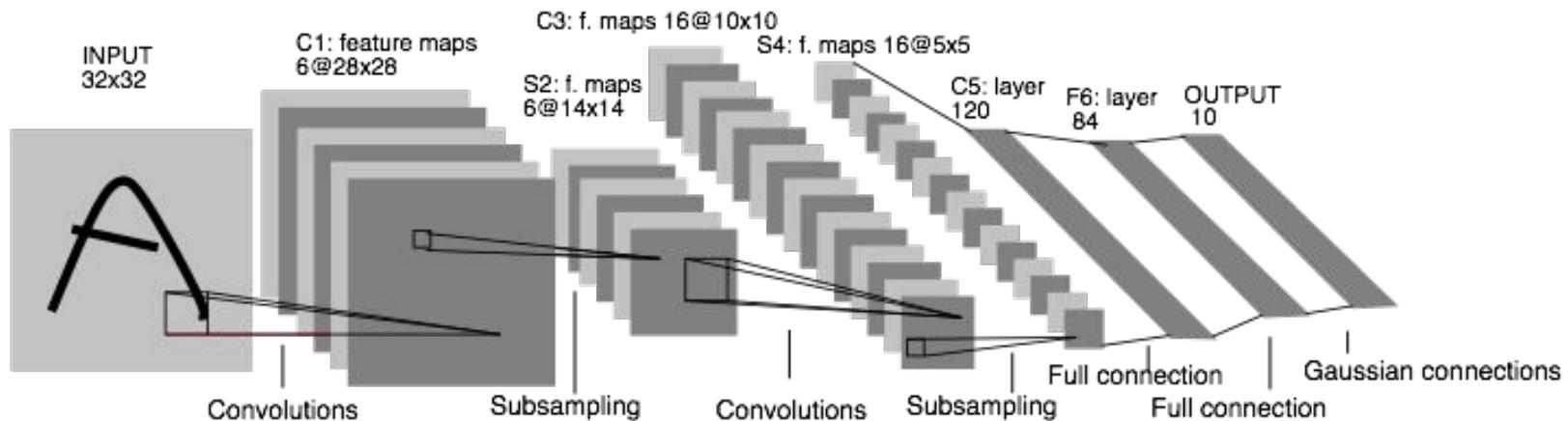
Network Tour



Transfer Learning



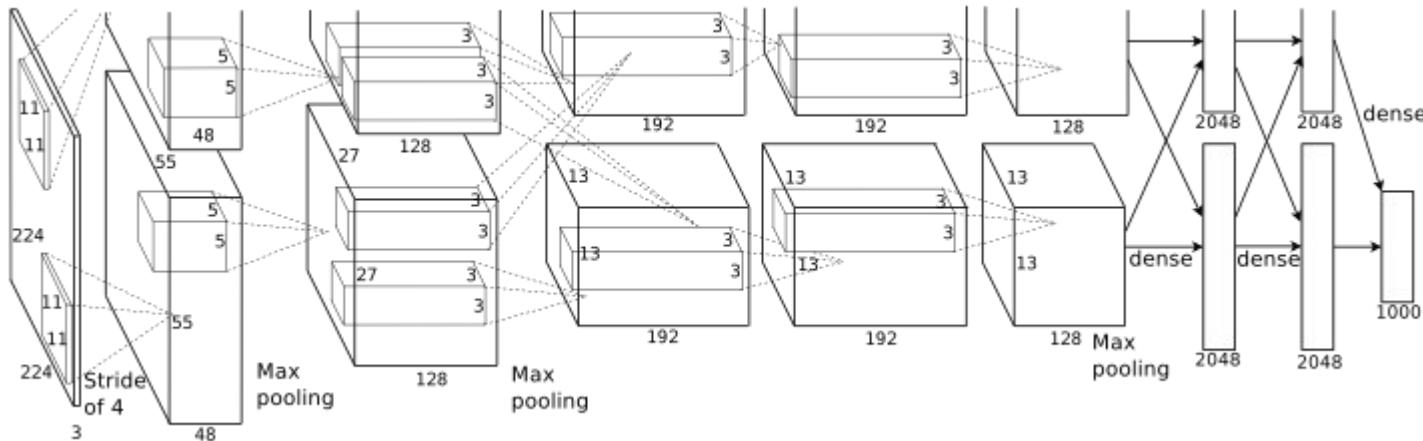
# Convolutional Networks: 1989



LeNet: a layered model composed of convolution and subsampling layers followed by a holistic representation and ultimately a classifier for handwritten digits [LeNet]

Note: channel dimension goes up as spatial dimension goes down... still a common pattern today

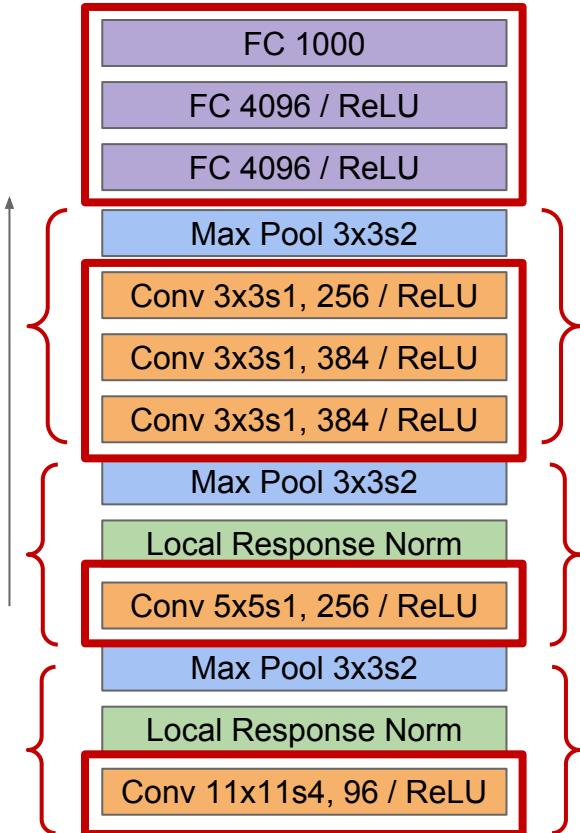
# Convolutional Networks: 2012



AlexNet: a layered model composed of convolution, subsampling, and further operations followed by a holistic representation and all-in-all a landmark classifier on ILSVRC12 [AlexNet]

- + data
- + gpu
- + non-saturating non-linearity
- + regularization

# Convnet Design Patterns



## FC-ReLU:

stack at end of the net to learn output  
majority of the learned parameters

## Conv-Pool:

1+ conv are followed by pooling to subsample  
spatial size shrinks; receptive field grows

## Conv-ReLU:

all conv are followed by non-linearity  
in this case ReLU

# Convnet Computation: 2012 & 2014

params	AlexNet	FLOPs
4M	FC 1000	4M
16M	FC 4096 / ReLU	16M
37M	FC 4096 / ReLU	37M
	Max Pool 3x3s2	
442K	Conv 3x3s1, 256 / ReLU	74M
1.3M	Conv 3x3s1, 384 / ReLU	112M
884K	Conv 3x3s1, 384 / ReLU	149M
	Max Pool 3x3s2	
	Local Response Norm	
307K	Conv 5x5s1, 256 / ReLU	223M
	Max Pool 3x3s2	
	Local Response Norm	
35K	Conv 11x11s4, 96 / ReLU	105M

AlexNet inference for a single image (3x227x227 input):

- **725M FLOPS**
- **60M** parameters (60,965,224 to be exact)
- **408 mb** GPU memory in Caffe  
**<12 gb** for batch size of 1,500
- **<1ms / image** on Titan X with cuDNN v4  
for batch size  $\geq 256$

Compare GoogleNet (ILSVRC14 winner):

- **2x** FLOPs
- **0.1x** the parameters
- **14%** more accurate

Architecture matters!

But the computational primitives are the same.

# Convolutional Nets: 2014



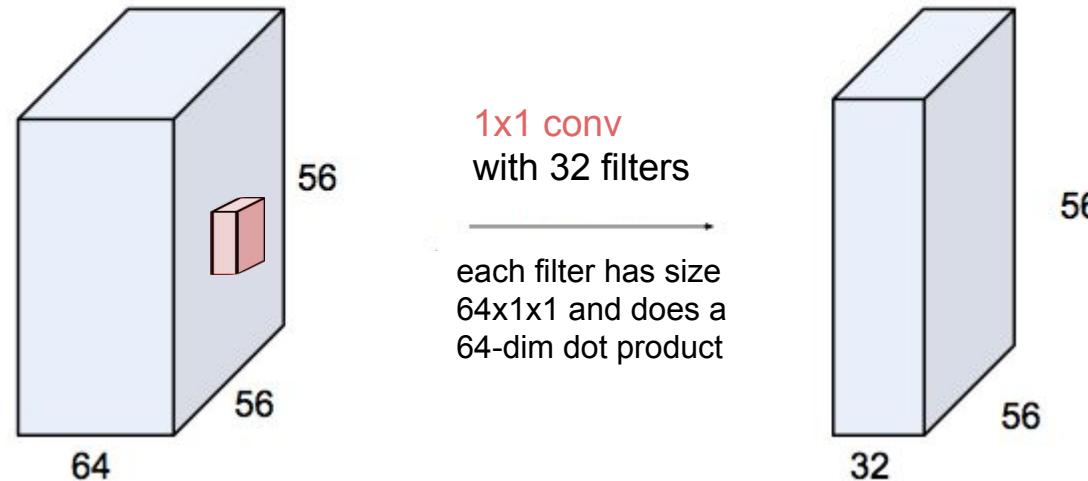
**GoogLeNet** ILSVRC14 Winner: ~6.6% Top-5 error

- composition of multi-scale dimension-reduced “Inception” modules
- no FC layers and only 5 million parameters

- + depth
- + auxiliary classifiers
- + dimensionality reduction

# 1x1 Convolution

- reduce channel dimension to control 1. parameter count 2. computation
- stack with non-linearity for deeper net
- found in many of the latest nets

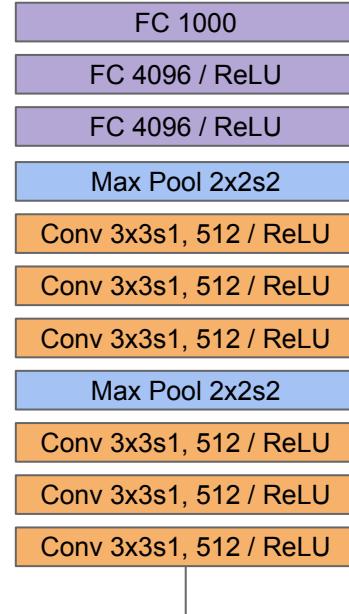
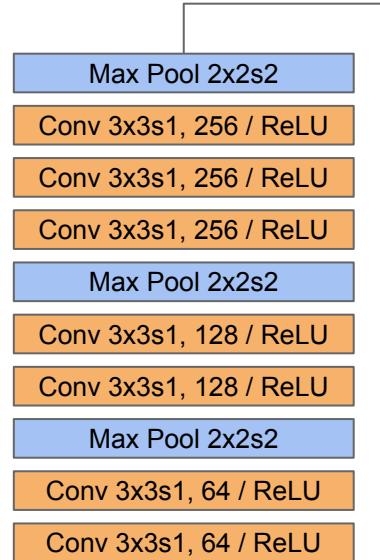


# Convolutional Nets: 2014

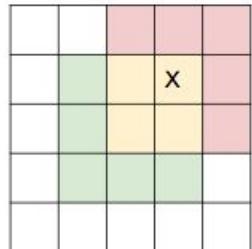
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
LRN		conv3-64	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128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

**VGG16** ILSVRC14 Runner-up: ~7.3% Top-5 error

- 13 layers of 3x3 convolution interleaved with max pooling + 3 fully-connected layers
- simple architecture, *good for transfer learning*
- 155 million params and more expensive to compute



stack 2  
3x3 conv



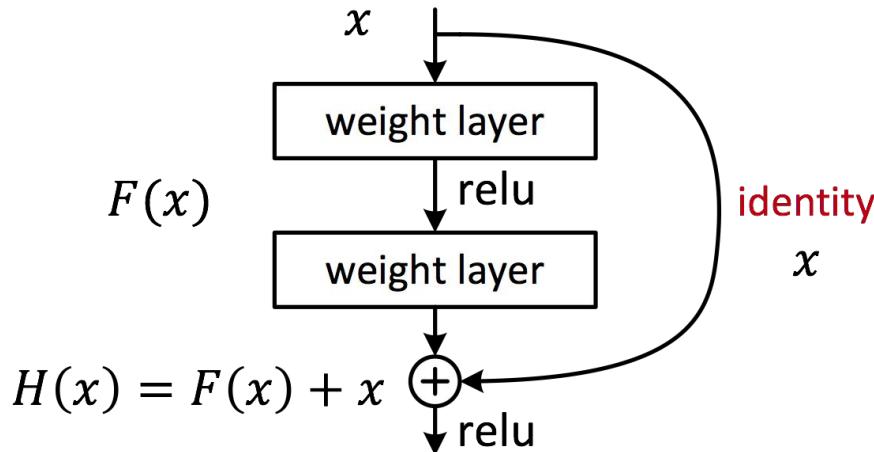
for a 5x5  
receptive field

[figure credit  
A. Karpathy]

- + depth
- + fine-tuning deeper and deeper
- + stacking small filters

# Convolutional Nets: 2015

Learn **residual** mapping w.r.t. **identity**



ILSVRC15 and COCO15 Winner: **MSRA ResNet**

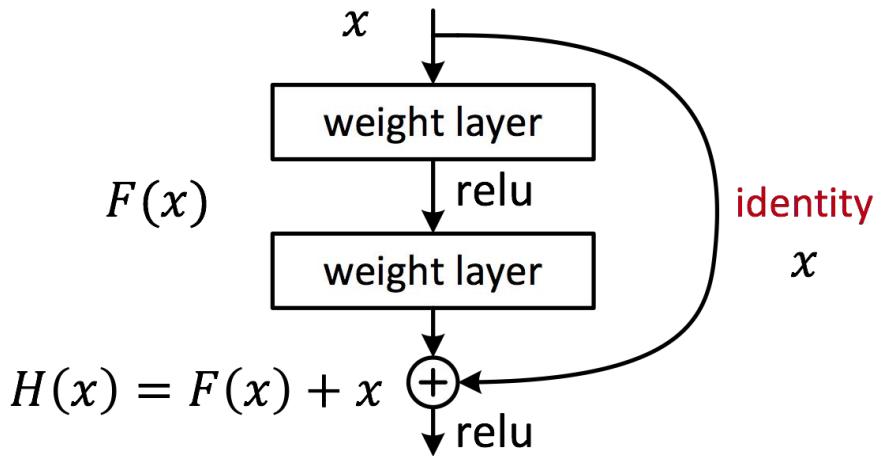
- classification
- detection
- segmentation

- very deep 100+ layer nets
- skip connections across layers
- batch normalization

Kaiming He, et al.

[Deep Residual Learning for Image Recognition](#)  
arXiv 1512.03385. Dec. 2015.

# Convolutional Nets: 2015



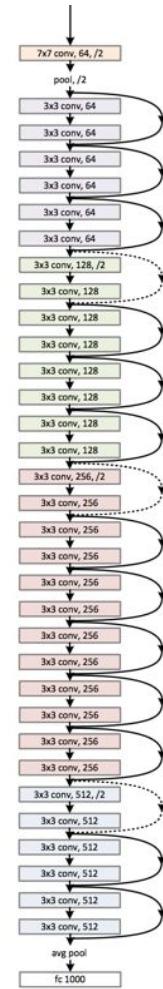
ILSVRC15 Winner **3.5% Top-5 error** and

COCO15 Winner with **>10% lead** for detection and segmentation

- MSRA Residual Net (ResNet): **101 and 152 layer networks**
- skip and sum layers to form residuals
- batch normalization (optimization trick)

**MSRA ResNet**

(~5x the layers shown here)



# Part 1: Deep Learning for Vision

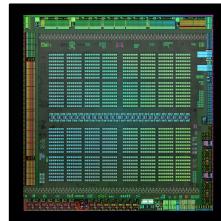
Why Deep Learning?



Dive into Deep Learning



What is DL?  
Why Now?



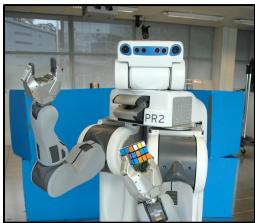
The Challenge  
of Recognition



Learning &  
Optimization

$$\nabla_{\theta} L$$

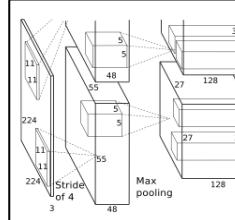
Applications



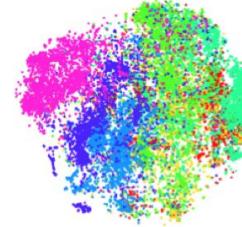
Caffe First Sip



Network Tour

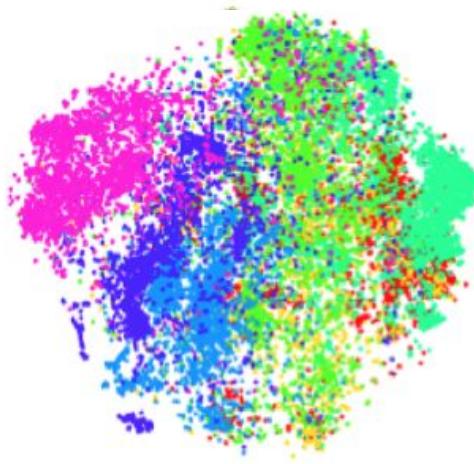


Transfer Learning



# Take a Pre-trained Model and Fine-tune to New Datasets...

Lots of Data



IM<sub>GENET</sub>



Your Data

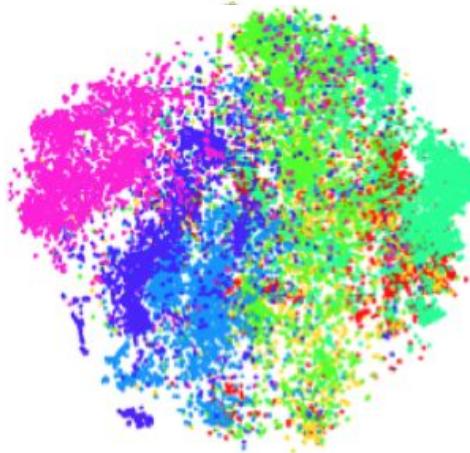


Style  
Recognition

Dogs vs.  
Cats  
top 10 in  
10 minutes

# Take a Pre-trained Model and Fine-tune to New Tasks...

Lots of Data

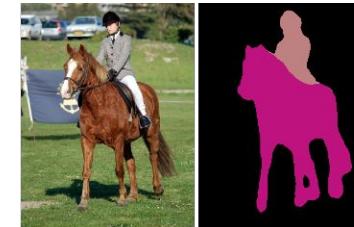


IM<sub>3</sub>GENET

Your Task



Detection



Segmentation

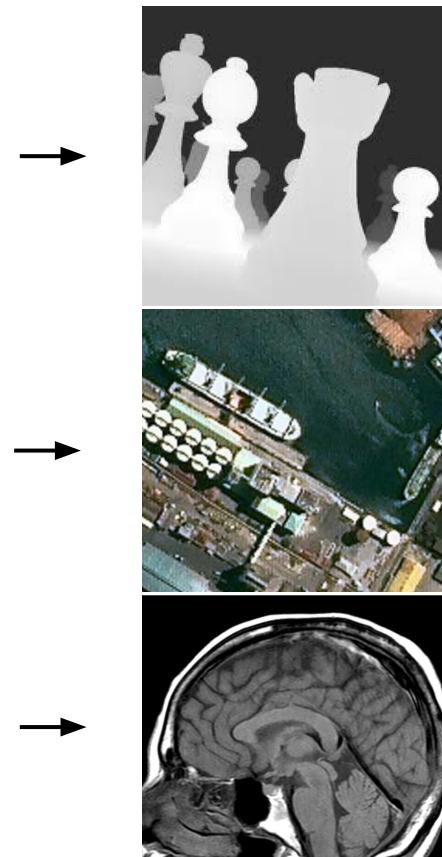
# Take a Pre-trained Model and Fine-tune to New Modalities...

Lots of Data



IMAGENET

Your Modality



Depth/  
Range

Remote  
Sensing

Medical  
Imaging

# End of Part 1

To learn more:

- [Deep Learning Intro](#) by Nielsen
- [Stanford cs231n](#) Convnet Class
- [Deep Learning Book \(draft\)](#)  
by Goodfellow, Bengio, Courville
- [caffe.berkeleyvision.org](#)

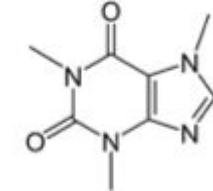
# Part 2: Hands-On Introduction to the Caffe Framework: *Brewing Networks with Caffe*

What is caffe?

Running a pre-trained network

Defining, learning, and testing networks

Fine-tuning for transfer learning



[caffe.berkeleyvision.org](http://caffe.berkeleyvision.org)



[github.com/BVLC/caffe](https://github.com/BVLC/caffe)



# What is Caffe?

**Open framework, models, and worked examples**  
for deep learning

- 4.5 years old
- 7,000+ citations, 250+ contributors, 24,000+ stars
- 15,000+ forks, >1 pull request / day average at peak
- focus has been vision, but also handles  
sequences, reinforcement learning, speech + text



Prototype



Train



Deploy

# What is Caffe?

**Open framework, models, and worked examples**  
for deep learning

- Pure C++ / CUDA architecture for deep learning
- Command line, Python, MATLAB interfaces
- Fast, well-tested code
- Tools, reference models, demos, and recipes
- Seamless switch between CPU and GPU



Prototype



Train



Deploy

# Caffe is a Community

project pulse

 BVLC / **caffe**

 Unwatch ▾ 1,205

 Unstar 8,498

 Fork 4,821

January 19, 2016 – February 19, 2016

Period: 1 month ▾

## Overview



 22  
Merged Pull Requests

 23  
Proposed Pull Requests

 52  
Closed Issues

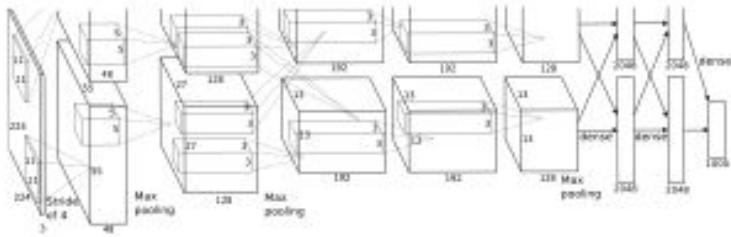
 38  
New Issues

Excluding merges, **20 authors** have pushed **19 commits** to master and **53 commits** to all branches. On master, **44 files** have changed and there have been **2,268 additions** and **162 deletions**.

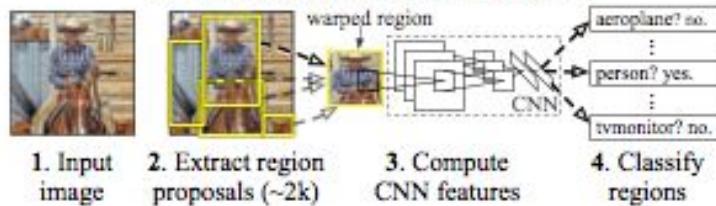


# Reference Models

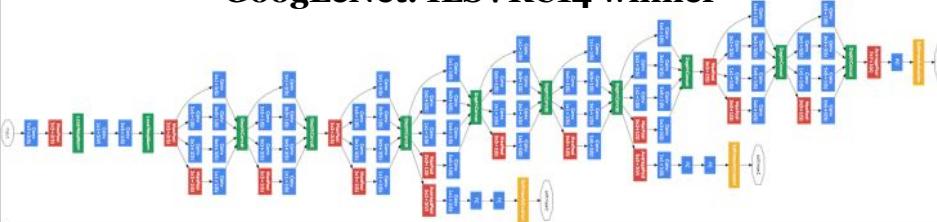
AlexNet: ImageNet Classification



R-CNN: *Regions with CNN features*



GoogLeNet: ILSVRC14 winner



Caffe offers the

- model definitions
- optimization settings
- pre-trained weights

so you can start right away

The BVLC models are licensed for unrestricted use

The community shares models in our [Model Zoo](#)

# Open Model Collection

The Caffe [Model Zoo](#) open collection of deep models to share innovation

- MSRA ResNet ILSVRC15 winner **in the zoo**
- VGG ILSVRC14 + Devil models **in the zoo**
- MIT Places scene recognition model **in the zoo**
- Network-in-Network / CCCP model **in the zoo**

help disseminate and reproduce research

bundled tools for loading and publishing models

**Share Your Models!** with your citation + license of course

# Brewing by the Numbers...

Speed with Krizhevsky's 2012 model:

- 2 ms/image on K40 GPU
- <1 ms inference with Caffe + cuDNN v4 on Titan X
- 72 million images/day with batched IO
- 8-core CPU: ~20 ms/image Intel optimization in progress

9k lines of C++ code (20k with tests)

# System Requirements

Linux or OS X

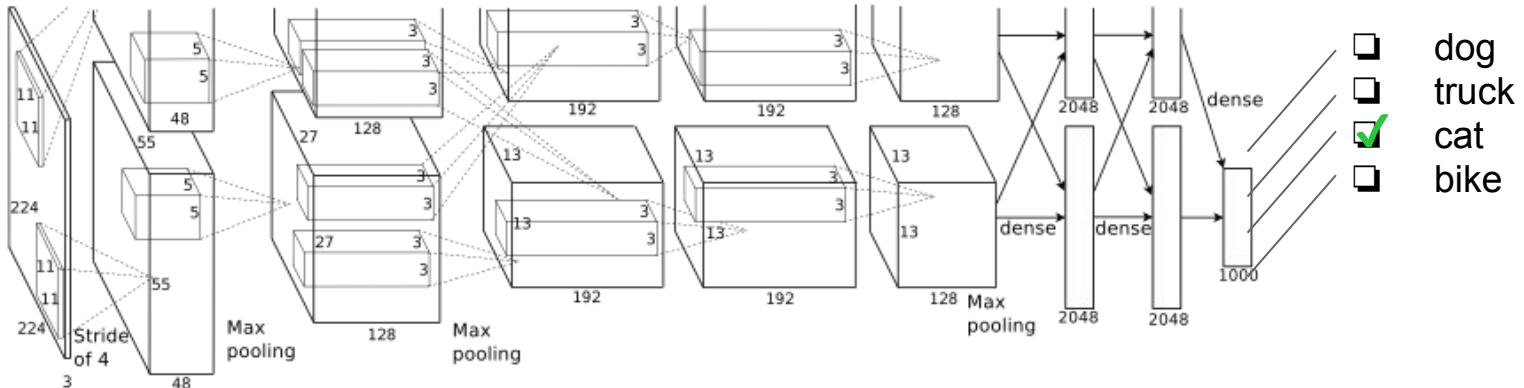
- Ubuntu best, RHEL ok
- Apt and Homebrew packaging in-progress

GPU

- CUDA >3.0 capability Titan X, K80, K40, GTX980, ...
- CPU-only mode for deployment

Windows and OpenCL ports—see [installation guide](#)

# Classification



Before we get into the details, let's take a peek at image classification using deep convnets in Caffe [AlexNet]

# Classification

Instant recognition the Caffe way  
[see notebook](#)

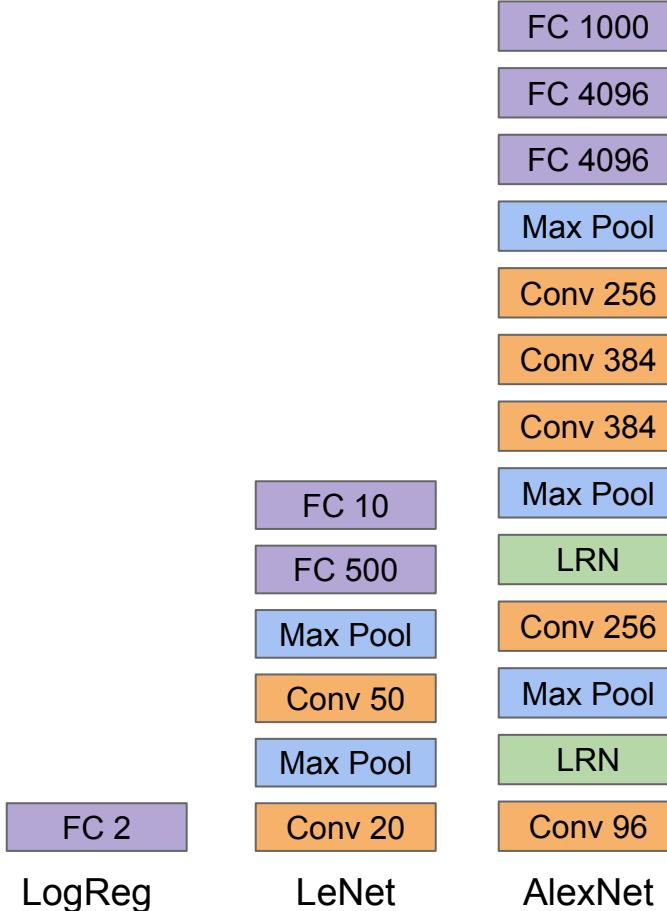
1. Load pre-trained network (CaffeNet)
2. Classify an image
3. Explore output and intermediate features
4. Compare CPU and GPU computation

# Net

A network is a set of layers  
and their connections:

```
name: "dummy-net"  
layer { name: "data" ...}  
layer { name: "conv" ...}  
layer { name: "pool" ...}  
    ... more layers ...  
layer { name: "loss" ...}
```

- Caffe creates and checks the net from the definition.
- Data and derivatives flow through the net as *blobs* – an array interface



# Layer Protocol

**Forward:** make output given input.

**Backward:** make gradient of output

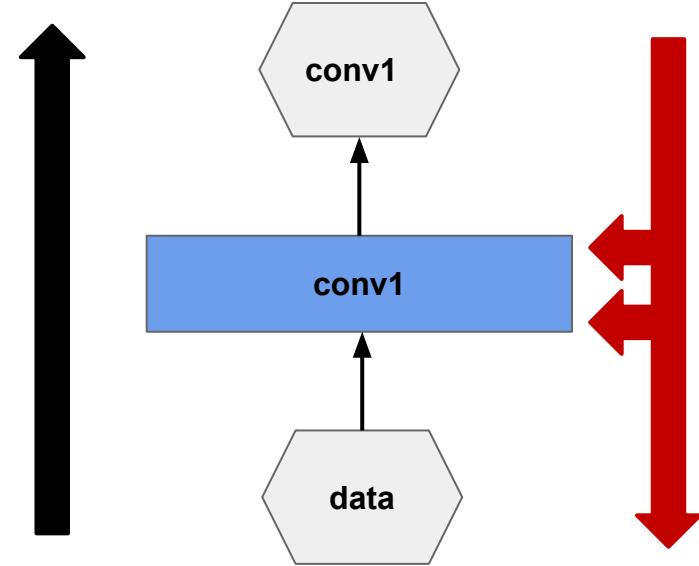
- w.r.t. bottom
- w.r.t. parameters (if needed)

**Setup:** run once for initialization.

**Reshape:** set dimensions.

## *Compositional Modeling*

The Net's forward and backward passes  
are composed of the layers' steps



[Layer Development Checklist](#)

```
import caffe
import numpy as np

class EuclideanLoss(caffe.Layer):

    def setup(self, bottom, top):
        # check input pair
        if len(bottom) != 2:
            raise Exception("Need two inputs to compute distance.")

    def reshape(self, bottom, top):
        # check input dimensions match
        if bottom[0].count != bottom[1].count:
            raise Exception("Inputs must have the same dimension.")
        # difference is shape of inputs
        self.diff = np.zeros_like(bottom[0].data, dtype=np.float32)
        # loss output is scalar
        top[0].reshape(1)

    def forward(self, bottom, top):
        self.diff[...] = bottom[0].data - bottom[1].data
        top[0].data[...] = np.sum(self.diff**2) / bottom[0].num / 2.

    def backward(self, top, propagate_down, bottom):
        for i in range(2):
            if not propagate_down[i]:
                continue
            if i == 0:
                sign = 1
            else:
                sign = -1
            bottom[i].diff[...] = sign * self.diff / bottom[i].num
```

# Layer Protocol == Class Interface

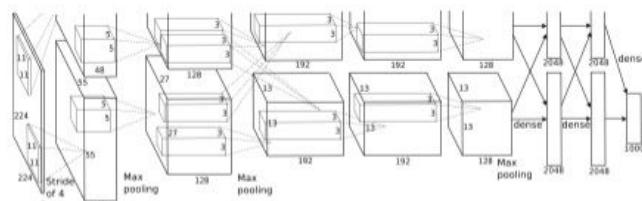
Define a class in C++ or Python to extend Layer

Include your new layer type in a network and keep brewing

```
layer {
  type: "Python"
  python_param {
    module: "layers"
    layer: "EuclideanLoss"
  }
}
```

# Forward / Backward The Essential Net Computations

Forward:  
inference  $f_W(x)$



"espresso"  
+ loss

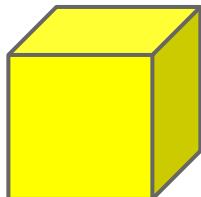
$\nabla f_W(x)$  Backward:  
learning

Caffe models are complete machine learning systems for inference and learning  
The computation follows from the model definition: define the model and run

# Memory Interface: Blob

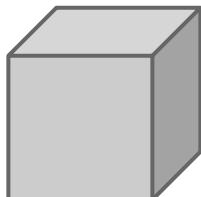
Blobs are N-D arrays for storing and communicating information

- hold data, derivatives, and parameters
- lazily allocate memory
- shuttle between CPU and GPU



**Data**

Number x  $K$  Channel x Height x Width  
256 x 3 x 227 x 227 for ImageNet train input



**Parameter: Convolution Weight**

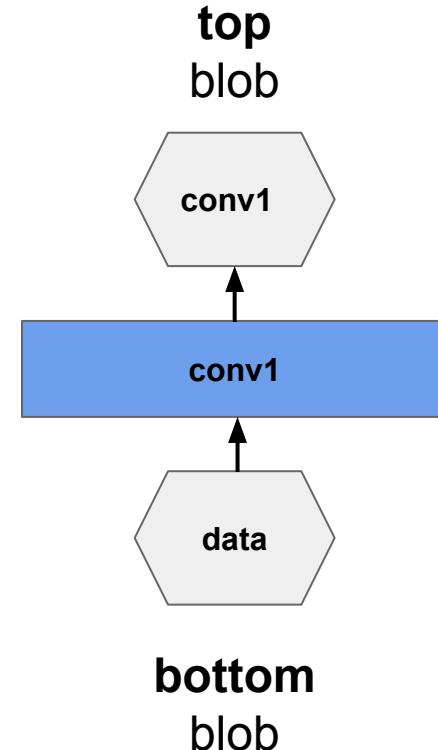
$N$  Output x  $K$  Input x Height x Width  
96 x 3 x 11 x 11 for CaffeNet conv1



**Parameter: Convolution Bias**

96 x 1 x 1 x 1 for CaffeNet conv1

```
name: "conv1"
type: "Convolution"
bottom: "data"
top: "conv1"
... definition ...
```



# Loss Defines the Task

What kind of model is this?

?

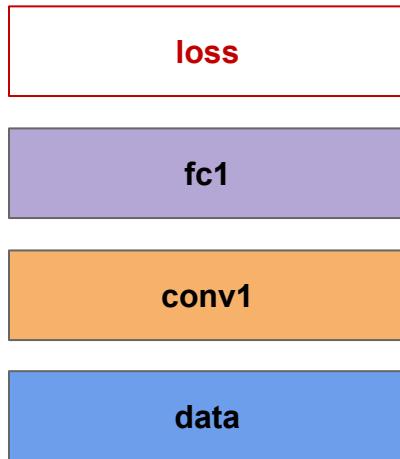
fc1

conv1

data

# Loss Defines the Task

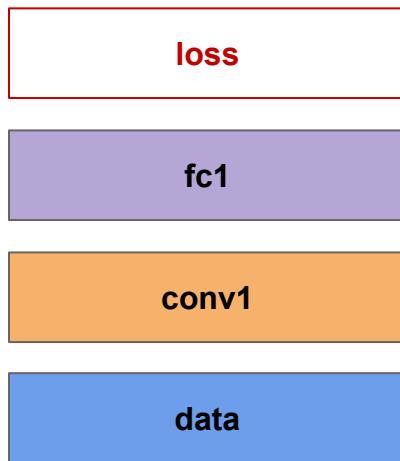
What kind of model is this?



Define the task by the **loss**

# Loss Defines the Task

What kind of model is this?



Classification	SoftmaxWithLoss
HingeLoss	
Linear Regression	EuclideanLoss
Attributes / Multiclassification	SigmoidCrossEntropyLoss
Others...	
New Task	
NewLoss	

Define the task by the **loss**

# Solving: Training a Net

Optimization like model definition is configuration

**train\_net**: "lenet\_train.prototxt"

**type**: SGD

**base\_lr**: 0.01

**momentum**: 0.9

**weight\_decay**: 0.0005

**max\_iter**: 10000

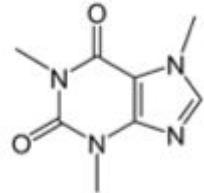
**snapshot\_prefix**: "lenet\_snapshot"

All you need to run things  
on the GPU

> caffe train -solver lenet\_solver.prototxt **-gpu 0**

SGD + momentum **SGD** · Nesterov's Accelerated Gradient **Nesterov**  
**Adaptive Solvers Adam** · RMSProp · AdaDelta · AdaGrad

# Coffee Break (15 minutes)

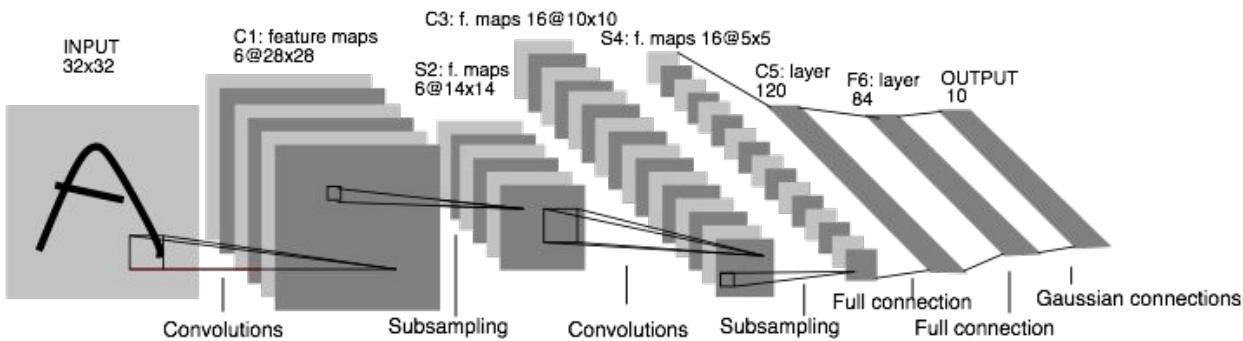


[caffe.berkeleyvision.org](http://caffe.berkeleyvision.org)

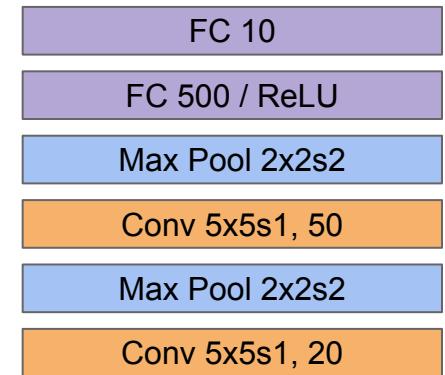
 [github.com/BVLC/caffe](https://github.com/BVLC/caffe)



# Learning LeNet



The original LeNet: a typical conv-pool to FC architecture  
in 1989 style with small channel dimensions and sigmoid non-linearity



our LeNet  
in 60 seconds

Define, train, and test a handwritten digit classifier like the famous LeNet convnet [LeNet]

# Learning LeNet

Back to the future of visual recognition  
[see notebook](#)

1. Define and instantiate the network and solver
2. Train the net on MNIST data to recognize digits
3. Examine progress during learning

# Model Format: Protobuf

- Strongly typed format
- Auto-generates code
- Developed by Google
- Defines Net / Layer / Solver schemas in **caffe.proto**

```
message ConvolutionParameter {  
    // The number of outputs for the layer  
    optional uint32 num_output = 1;  
    // whether to have bias terms  
    optional bool bias_term = 2 [default = true];  
}
```

```
layer {  
    name: "conv1"  
    type: "Convolution"  
    bottom: "data"  
    top: "conv1"  
    convolution_param {  
        num_output: 20  
        kernel_size: 5  
        stride: 1  
        weight_filler {  
            type: "xavier"  
        }  
    } }  
}
```

# Model Zoo Format

 [readme.md](#) [Raw](#)

---

name: FCN-32s Fully Convolutional Semantic Segmentation on PASCAL-Context  
caffemodel: fcn-32s-pascalcontext.caffemodel  
caffemodel\_url: <http://dl.caffe.berkeleyvision.org/fcn-32s-pascalcontext.caffemodel>  
sha1: adbbd504c280e2b8966fc32e32ada2a2ecf13603

**gist\_id: 80667189b218ad570e82**

---

This is a model from the [paper](#):

Fully Convolutional Networks for Semantic Segmentation  
Jonathan Long, Evan Shelhamer, Trevor Darrell  
arXiv:1411.4038

Gists on github hold model definition, license, url for weights, and hash of Caffe commit that guarantees compatibility

# Take a Pre-trained Model and Fine-tune to New Tasks

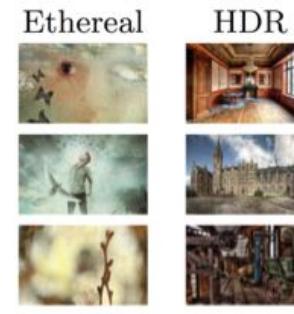
[DeCAF, Zeiler-Fergus, R-CNN,

OverFeat]

Lots of Data



Your Task



Style  
Recognition

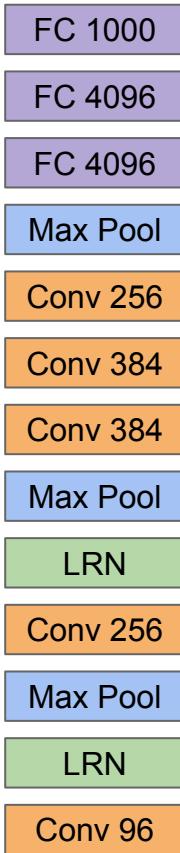


© kaggle.com

Dogs vs.  
Cats  
top 10 in  
10 minutes

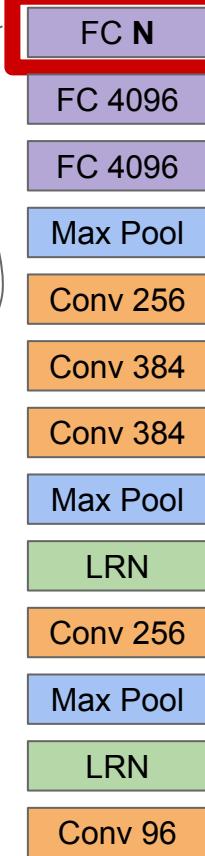
# Transfer Learning With Convnets

(1) Train on ImageNet



(2a) *Small dataset.*  
Fix all weights (treat convnet as a feature extractor); retrain only the classifier

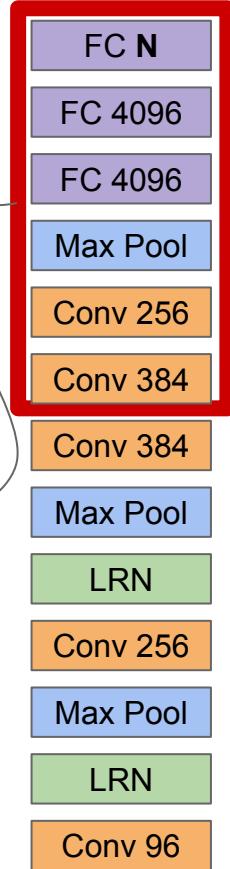
Swap the softmax layer at the end, replace with appropriate number of classes **N**



(2b) *Medium-sized or larger dataset.*

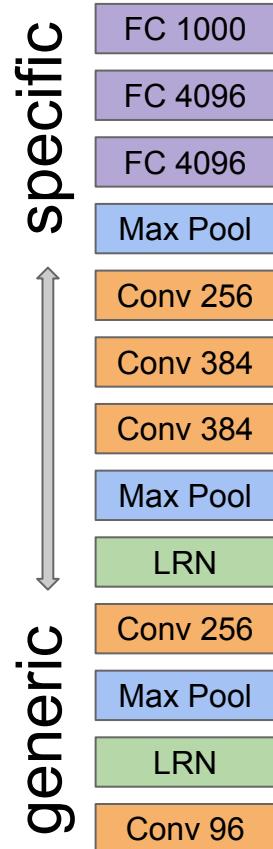
**Fine-tune** the network. Using the ImageNet-learned weights as an initialization, train the full network, or just some of the last layers

Retrain a bigger portion of the network, or even all of it



Tip: use only ~1/10th the original learning rate when fine-tuning the top (output) layer and ~1/100th for other layers

# From the Source to Other Tasks



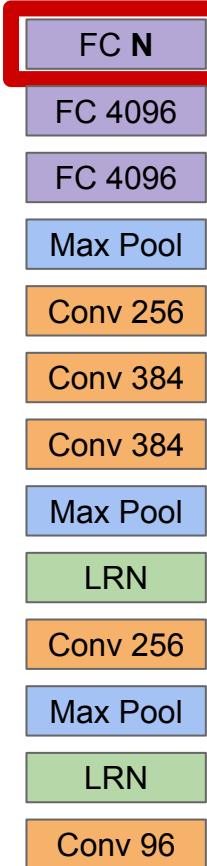
the  
**target**  
dataset  
has...

compared with the source, the **target** dataset is a...

	<b>very similar dataset</b>	<b>very different dataset</b>
<b>very little data</b>	Learn linear classifier on features from top layer(s)	You're in trouble ... try a linear classifier on different layers
<b>lots of data</b>	Finetune a few layers	Finetune more layers

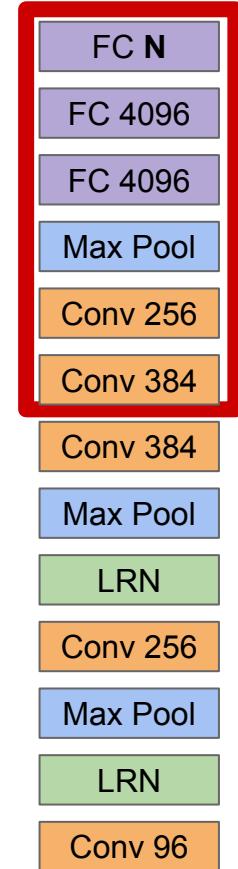
# A Two-stage Approach to Fine-tuning

(1) First, train only the new classifier to convergence, without modifying the network or weights otherwise



(2) Then, train more of the network, or even all of it

- (1) prevents the pretrained network from being subjected to noisy updates from the randomly initialized classifier, so that in (2), the rest of the network can be usefully tuned for the target task



# From ImageNet to Style (1/2)

Simply change a few lines in the model definition

```
layer {
    name: "data"
    type: "Data"
    data_param {
        source: "ilsvrc12_train_lmdb"
        mean_file: "../../data/ilsvrc12"
    }
    ...
}
...
layer {
    name: "fc8"
    type: "InnerProduct"
    inner_product_param {
        num_output: 1000
    }
}
```

```
layer {
    name: "data"
    type: "Data"
    data_param {
        source: "style_train_lmdb"
        mean_file: "../../data/ilsvrc12"
    }
    ...
}
...
layer {
    name: "fc8-style"
    type: "InnerProduct"
    inner_product_param {
        num_output: 20
    }
}
```

new name =  
new params

Input:  
A different source

Last Layer:  
A different classifier

# From ImageNet to Style (2/2)

```
> caffe train -solver models/finetune_flickr_style/solver.prototxt  
      -weights bvlc_reference_caffenet.caffemodel
```

Step-by-step in pycaffe:

```
pretrained_net = caffe.Net(  
    "net.prototxt", "net.caffemodel")  
  
solver = caffe.SGDSolver("solver.prototxt")  
solver.net.copy_from(pretrained_net)  
solver.solve()
```



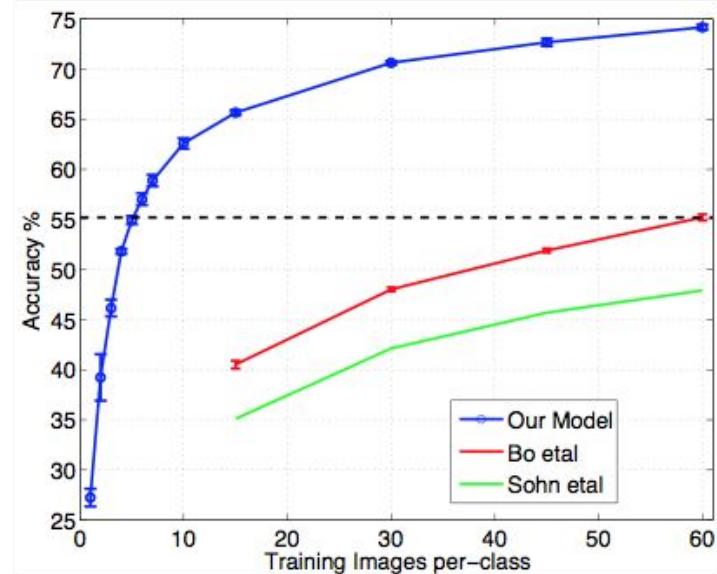
# When to Fine-tune?

Almost always

- More robust optimization – good initialization helps
- Needs less data
- Faster learning

State-of-the-art results in

- classification
- detection
- segmentation
- more



# Fine-tuning Tricks

## Learn the last layer first

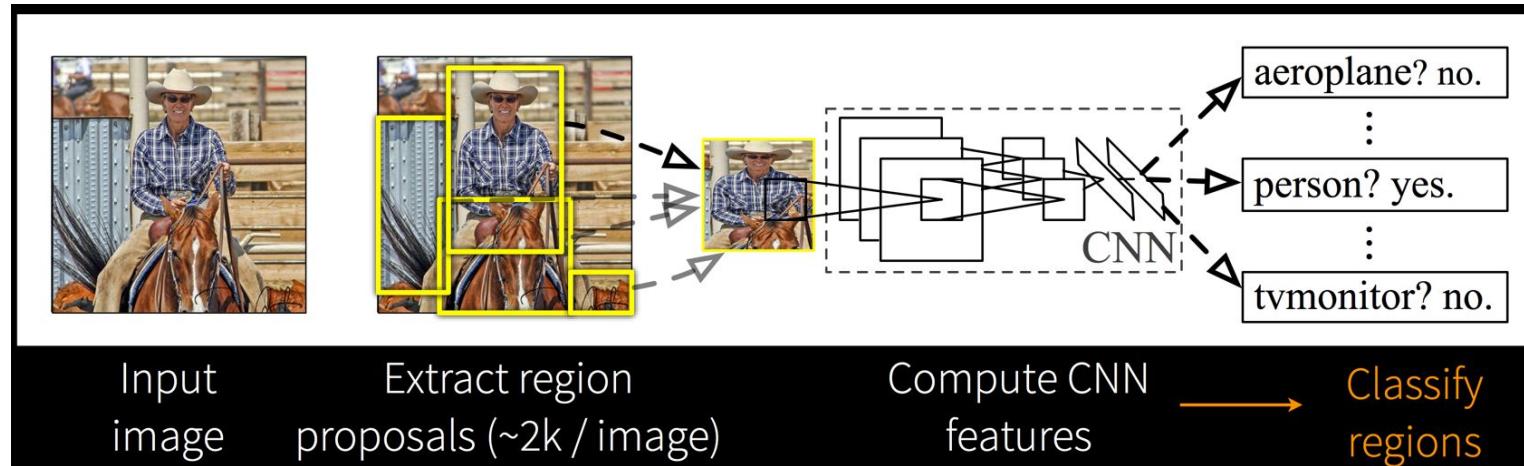
- Caffe layers have local learning rates: `param { lr_mult: 1 }`
- Freeze all but the last layer for fast optimization and avoiding divergence by setting `lr_mult: 0` to fix a parameter
- Stop if good enough, or keep fine-tuning

## Reduce the learning rate

- Drop the solver learning rate by 10x, 100x
- Preserve the initialization from pre-training and avoid divergence

# Fine-tuning for Detection

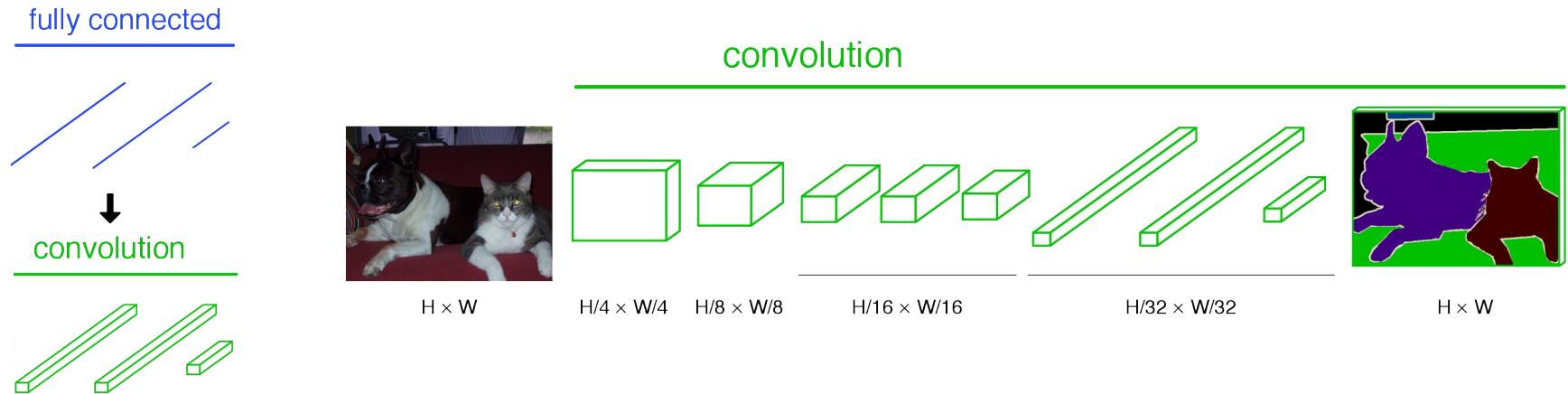
R-CNN approach: cast detection into classification of *region proposals*; fine-tune as usual



Later work (*Faster R-CNN*) incorporates region proposals into net: end-to-end approach is faster and more accurate

# Fine-tuning for Segmentation

FCN approach: cast net in *spatially dense* “fully convolutional” form;  
fine-tune as usual



Infer and learn on whole images at a time:  
end-to-end approach is faster and more accurate

# Fine-tuning

Transferring features to style recognition  
[see notebook](#)

1. Learn a classifier on pre-trained features
2. Fine-tune by end-to-end optimization
3. Compare from scratch and fine-tuned nets

# Recipe for Brewing

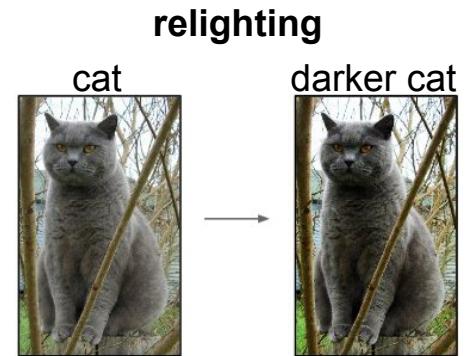
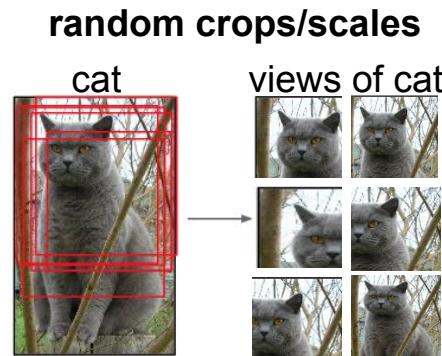
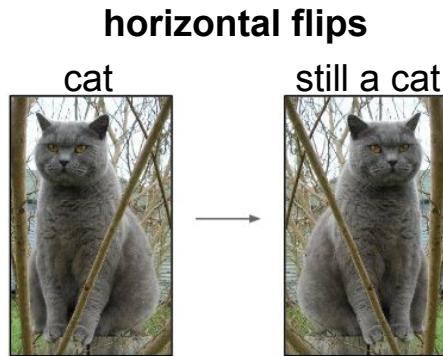
- Convert the data to Caffe-format
  - python layer, Imdb, leveldb, hdf5 / .mat, list of images, etc.
- Define the Net
- Configure the Solver
- `caffe train -solver solver.prototxt -gpu 0`
- Examples are your friends
  - `caffe/examples/mnist, cifar10, imagenet`
  - `caffe/examples/*.ipynb`
  - `caffe/models/*`

# Contents of the `caffe` Directory

- **model schema**
  - src/caffe/proto/**caffe.proto**
- **layers**
  - src/caffe/layers/\*.cpp, cu}
  - include/caffe/layers/\*.hpp
- **solvers**
  - src/caffe/solvers/\*.cpp, cu}
  - include/caffe/sgd\_solvers.hpp
- **examples** notebooks + scripts
  - examples/\*.ipynb for hands-on notebooks
  - examples/mnist for quick-start scripts
- **docs** local edition of tour, API, web site
- **models** reference models + model zoo
- **tools, scripts**
  - command line tool
  - visualization

# Data Augmentation

transform the training data, without changing its truth



... and anything else you can come up with! (random combinations of the above, or shearing, rotations, ...)

# Debugging Training (Sanity Checks)

- my loss is **diverging** or **getting stuck**!
- verify these sanity checks before exploring optimization tricks on next slides
  - check your **input data** and labels
    - look at the values actually being fed into the net (reasonable mean/variance? zero mean = probably good; zero variance = almost certainly bad)
    - if the data and/or labels are images (or image-like), **visualize** them
  - did you initialize multiplicative weights to non-zero values?
    - if not, you may see that the loss is exactly constant
    - add `debug_info: true` to a Caffe solver: prints the mean of absolute activation (output) values for each layer
      - if any of these values is 0, you may have initialized weights to 0 in one or more layers
  - if you wrote any new layers, use a **gradient checker** to ensure their “backward” implementation is correct

# Debugging Training (Diverging Loss)

- what should I do if...
- ...my **loss diverges**? (increases by order of magnitude, goes to inf. or NaN)
  - lower the learning rate
  - raise momentum (with corresponding learning rate drop)
  - raise weight decay
  - raise batch size
  - use gradient clipping (limit the L2 norm of the gradient to a particular value at each iteration; shrink it to that norm if greater)
  - try another solver: momentum SGD, ADAM, RMSProp, ...
  - try a smaller initialization (e.g., for a Gaussian init., lower the stdev.)

# Debugging Training (Static Loss 1/3)

- what should I do if...
- ...my **loss doesn't improve / gets stuck / drops slowly?**
  - raise the learning rate
  - (maybe) lower momentum, weight decay, and/or batch size
  - try another solver: momentum SGD, ADAM, RMSProp, ...
  - transfer a pre-trained (e.g. on ImageNet) initialization, if possible
  - use a larger initialization (in particular, *make sure you didn't zero-initialize any multiplicative weights* in intermediate layers)
  - use a “smarter” initialization (e.g., for linear layers followed by ReLUs, try the `msra` initialization in Caffe)

# Debugging Training (Static Loss 2/3)

- what should I do if...
- ...my **loss doesn't improve / gets stuck / drops slowly?**
  - remove some layers to make the network shallower
    - at least to start!
    - a strategy for model design: begin with a simple, trainable network; “deepen” it by adding new layers one-by-one
  - modify the architecture to improve gradient flow:
    - batch normalization
    - residual learning [ResNet]
    - intermediate losses [GoogLeNet]
    - other tricks

# Debugging Training (Static Loss 3/3)

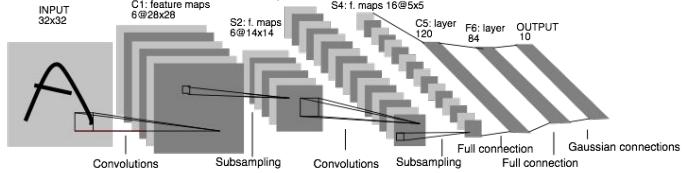
- what should I do if...
- ...my **loss doesn't improve / gets stuck / drops slowly?**
  - be patient! (go outside?)
    - deep learning can take a long time
    - training AlexNet in 2012: 12 days  
although this is down to 1 day in 2015!
      - loss hovers around the chance value of  $\ln(1000) \approx 6.908$  for the first 1000+ iterations (~1 hour on 2012 GPU)
    - training ResNet-152 in 2015: 1-2 months (on 8 GPUs!)
  - the best configurations (net architectures, solvers) at convergence are often *not* the ones that train fastest early on
    - some tricks to speed up learning can be “greedy” rather than ultimately beneficial

# Debugging Training (Overfitting/Underfitting)

- what should I do if...
- ...my **net overfits**? (train accuracy is much higher than test accuracy)
  - ideally, collect (and label) more data! but this can be expensive, so...
  - use data augmentation (and if possible, generate data)
  - in a word, **regularize!** specifically...
    - increase weight decay (and/or other weight regularization)
    - use dropout (or increase proportion of activations dropped out)
    - try a shallower/simpler model
      - fewer layers
      - fewer units in each layer
      - (for convnets) restrict (reduce) receptive field sizes
- ...my **net underfits**? (train accuracy is much lower than expected)
  - use less regularization (reverse any above regularization suggestions)
  - try a deeper/larger model (more layers, more units, ...)

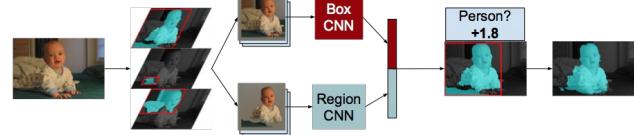
# Network Graph

Many current deep models have linear structure

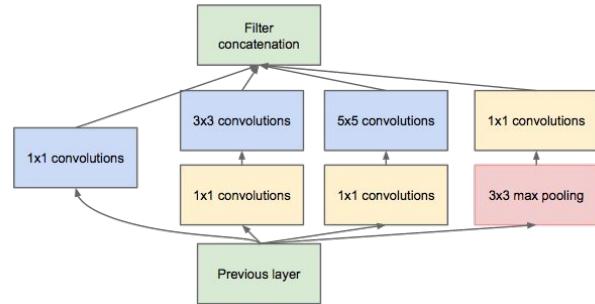


but Caffe nets can have any directed acyclic graph (DAG) structure

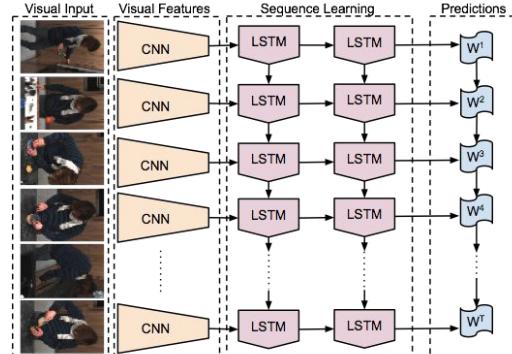
Define bottoms and tops  
and Caffe will connect the net



two-stream detection + segmentation



GoogLeNet Inception Module



LRCN joint vision-sequence model 183

# Weight Sharing

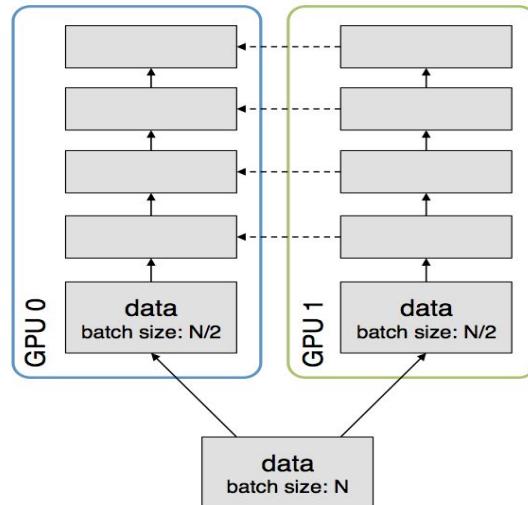
- Just give the parameter blobs explicit names using the param field
- Layers specifying the same param name will share that parameter, accumulating gradients accordingly

```
layer {
    name: 'innerproduct1'
    type: 'InnerProduct'
    inner_product_param {
        num_output: 10
        bias_term: false
        weight_filler {
            type: 'gaussian'
            std: 10
        }
    }
param { name: 'sharedweights' }
bottom: 'data1'
top: 'innerproduct1'
}
layer: {
    name: 'innerproduct2'
    type: 'InnerProduct'
    inner_product_param {
        num_output: 10
        bias_term: false
    }
param { name: 'sharedweights' }
bottom: 'data2'
top: 'innerproduct2'
}
```

# Parallelism

Train in parallel across GPUs

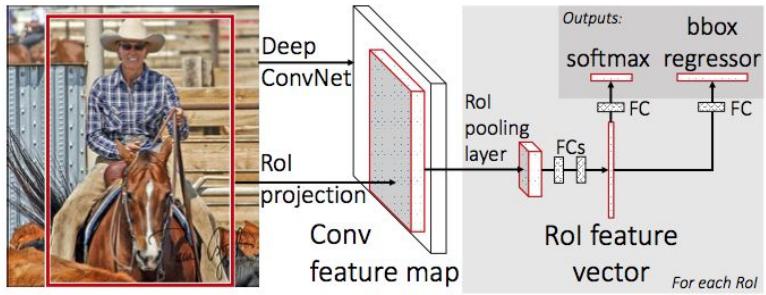
- data-parallelism included  
caffe train -gpu 0,1 ...
- further parallelism possible,  
with effort



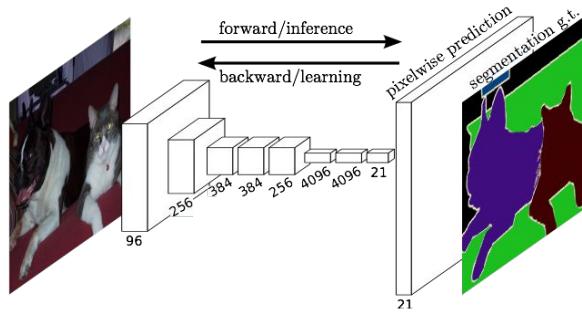
**data parallelism**  
split data across processors

# Extensions Tasks and Future

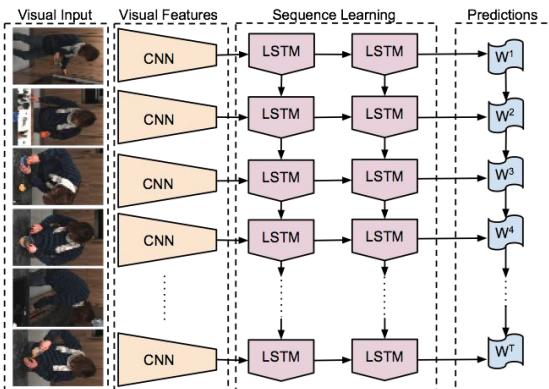
## Detection



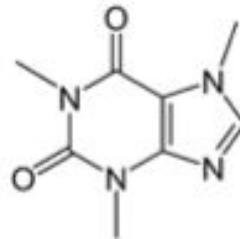
## Pixelwise Prediction



## Sequences



## Framework Future



# Framework Future

1.0 is coming stability, documentation, packaging

Dedicated Branches for OpenCL and Windows

Halide and Module interface for prototyping and experimenting

Performance Tuning for GPU (CUB) and CPU (nppack)

Widening the Circle continued and closer collaborative development

# Help Brewing

## Documentation

- [tutorial documentation](#)
- [hands-on examples](#)

## Modeling, Usage, and Installation

- [caffe-users group](#)
- [gitter.im chat](#)

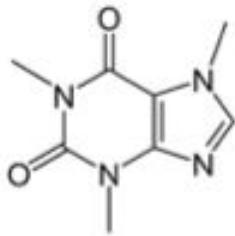
## Convolutional Nets

- [CS231n online convnet class](#)  
by Andrej Karpathy and  
Fei-Fei Li
- [Deep Learning Online](#)  
by Michael Nielsen
- [Deep Learning Book](#)  
by Goodfellow, Bengio,  
Courville

# Next Steps

Today you've seen the essentials of deep learning for vision with convolutional networks and Caffe

Next Up:



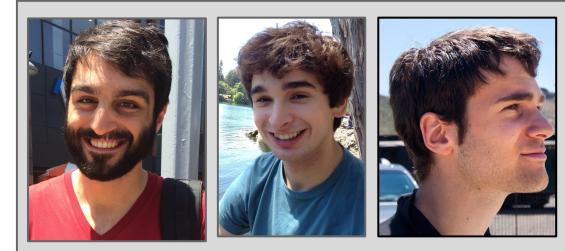
[caffe.berkeleyvision.org](http://caffe.berkeleyvision.org)

 [github.com/BVLC/caffe](https://github.com/BVLC/caffe)

Check out Caffe on github



[Run Caffe through Docker](#)  
and NVIDIA Docker for GPU



Reach out to [Caffe Coldpress](#)  
for R&D consulting

# Acknowledgements



Thank you to the Berkeley Vision and Learning Center and its sponsors



Thank you to NVIDIA  
for GPUs, cuDNN collaboration,  
and hands-on cloud instances



Thank you to our 250+  
open source contributors  
and vibrant community!

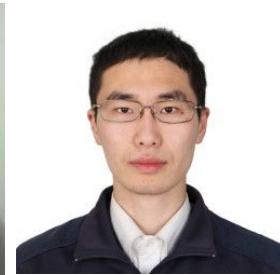
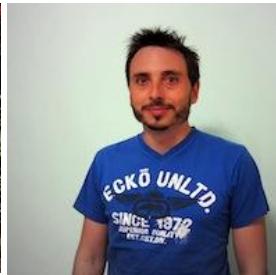


Thank you to A9 and AWS  
for a grant for Caffe dev and  
reproducible research



Thank you to the EVA  
for partnering with us  
to make this tutorial

# Thanks to the whole Caffe Crew



Yangqing Jia, Evan Shelhamer, Jeff Donahue, Jonathan Long,  
Sergey Karayev, Ross Girshick, Sergio Guadarrama, Ronghang Hu, Trevor Darrell  
and our [open source contributors!](#)

# References

- [ DeCAF ] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell.  
DeCAF: A deep convolutional activation feature for generic visual recognition. ICML14.
- [ R-CNN ] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. CVPR14.
- [ Zeiler-Fergus ] M. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. ECCV14.
- [ LeNet ] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. IEEE98.
- [ AlexNet ] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. NIPS12.
- [ OverFeat ] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun.  
Overfeat: Integrated Recognition, Localization and Detection using Convolutional Networks. ICLR14.
- [ Karayev14 ] [S. Karayev](#), [M. Trentacoste](#), [H. Han](#), [A. Agarwala](#), [T. Darrell](#), [A. Hertzmann](#), [H. Winnemoeller](#). Recognizing Image Style. BMVC14.
- [ Russakovsky15 ] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. IJCV15.
- [ Szegedy15 ] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich.  
Going Deeper with Convolutions. CVPR15.
- [ Simonyan15 ] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR15.
- [ He15 ] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2015

**END**