

6.874, 6.802, 20.390, 20.490, HST.506

Deep Learning in the Life Sciences

# Lecture 3:

# Convolutional Neural Networks

## Prof. Manolis Kellis

Slides credit: **6.S191**, Dana Erlich, Param Vir Singh, David Gifford, Alexander Amini, Ava Soleimany, @TessFerrandez's totally awesome Coursera Notes, and many more outstanding online resources



# Today: Convolutional Neural Networks (CNNs)

## 1. Scene understanding and object recognition for machines (and humans)

- Scene/object recognition challenge. Illusions reveal primitives, conflicting info
- Human neurons/circuits. Visual cortex layers==abstraction. General cognition

## 2. Classical machine vision foundations: features, scenes, filters, convolution

- Spatial structure primitives: edge detectors & other filters, feature recognition
- Convolution: basics, padding, stride, object recognition, architectures

## 3. CNN foundations: LeNet, *de novo* feature learning, parameter sharing

- Key ideas: *learn* features, hierarchy, re-use parameters, back-prop filter learning
- CNN formalization: representations(Conv+ReLU+Pool)\*N layers + Fully-connected

## 4. Modern CNN architectures: millions of parameters, dozens of layers

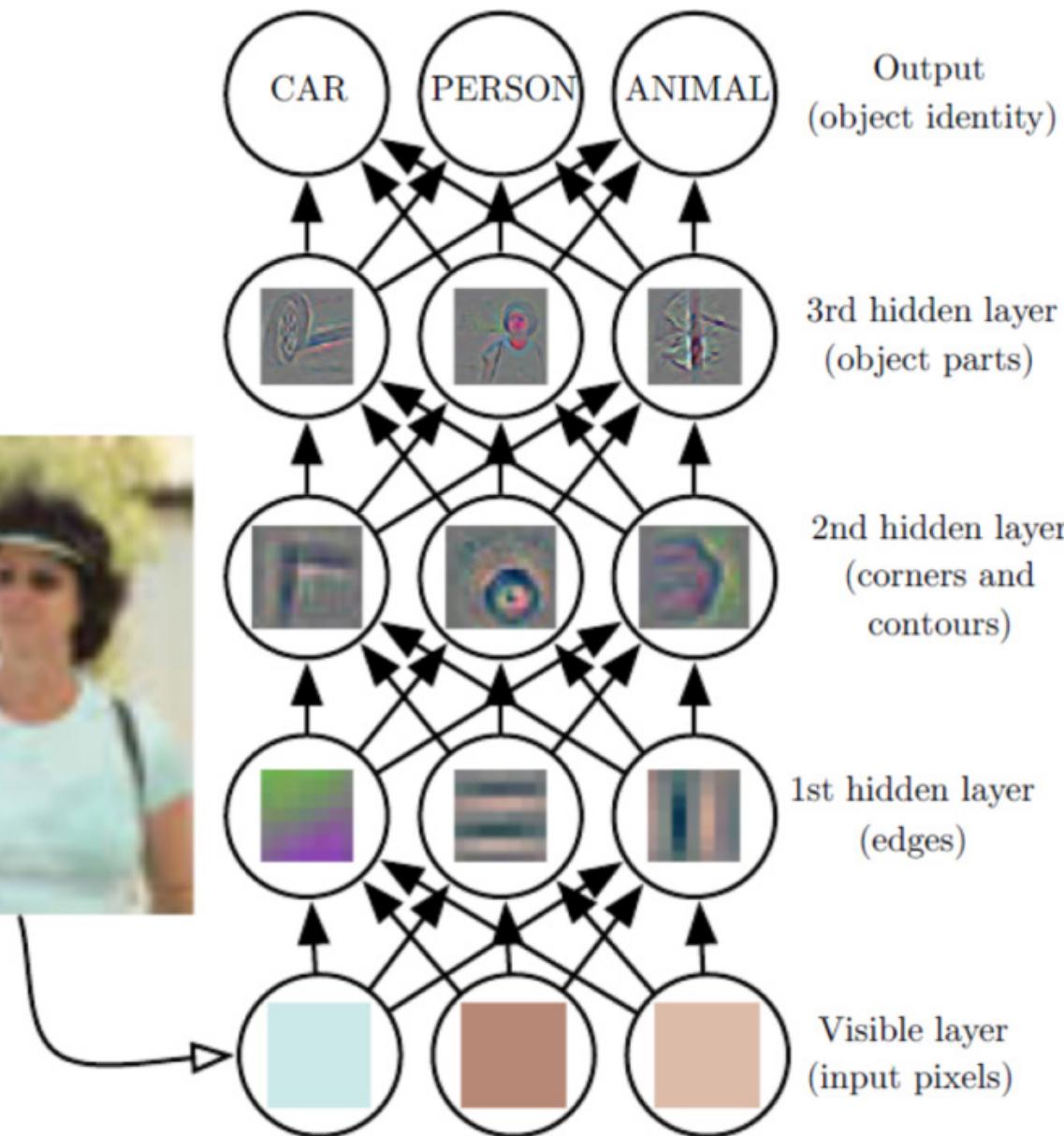
- Feature invariance is hard: apply perturbations, learn for each variation
- ImageNet progression of best performers
- AlexNet: First top performer CNN, 60M parameters (from 60k in LeNet-5), ReLU
- VGGNet: simpler but deeper (8→19 layers), 140M parameters, ensembles
- GoogleNet: new primitive=inception module, 5M params, no FC, efficiency
- ResNet: 152 layers, vanishing gradients → fit residuals to enable learning

## 5. Countless applications: General architecture, enormous power

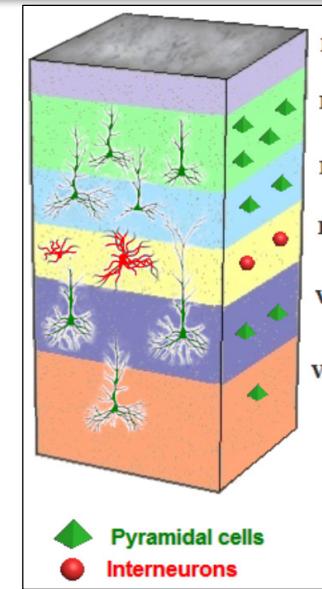
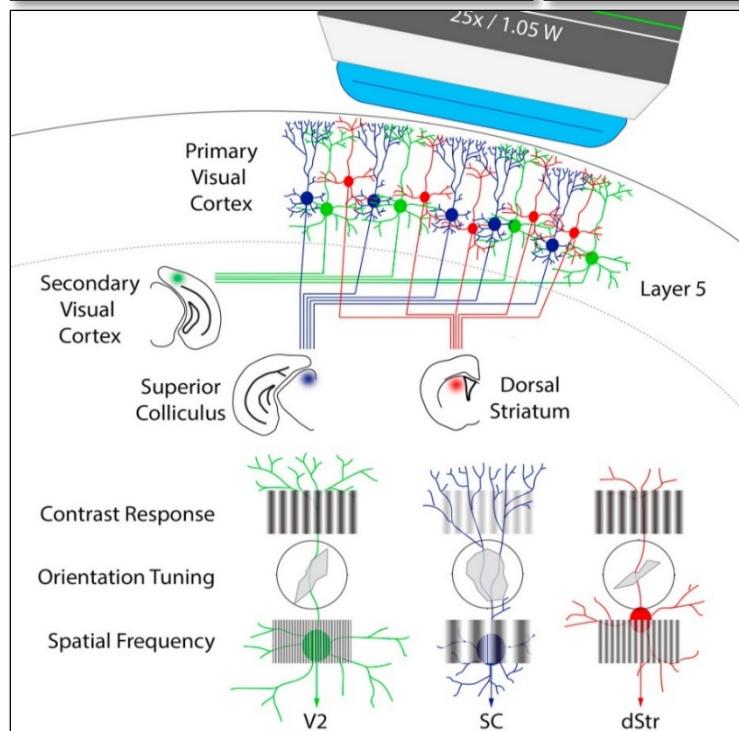
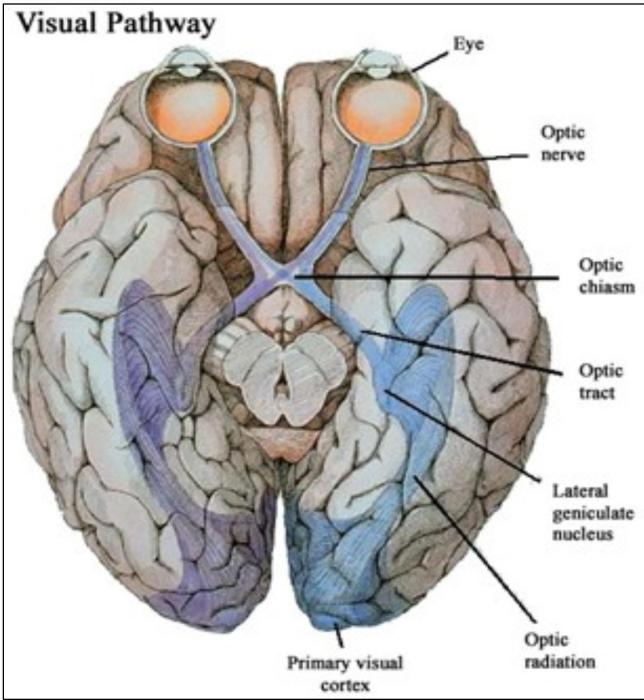
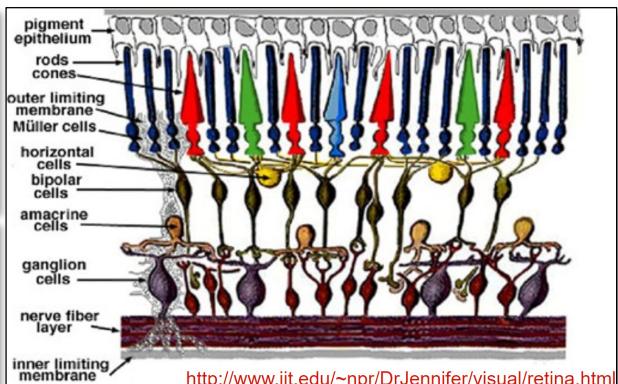
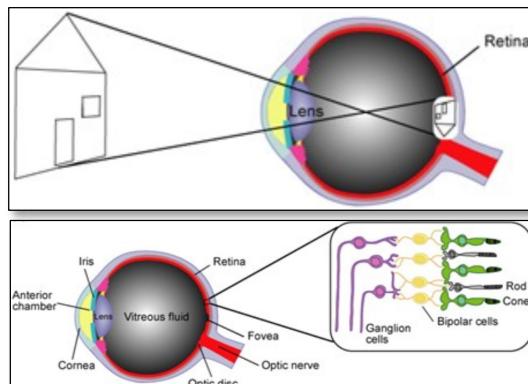
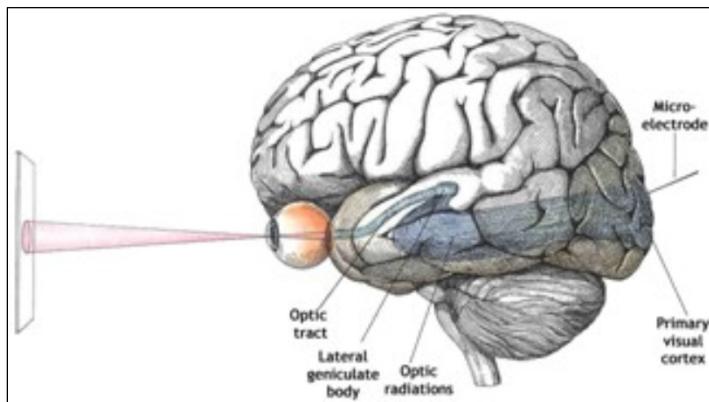
- Semantic segmentation, facial detection/recognition, self-driving, image colorization, optimizing pictures/scenes, up-scaling, medicine, biology, genomics

Convolutional neural networks  
inside our brains

# Human Vision $\Leftrightarrow$ many layers of abstraction $\Leftrightarrow$ Deep learning

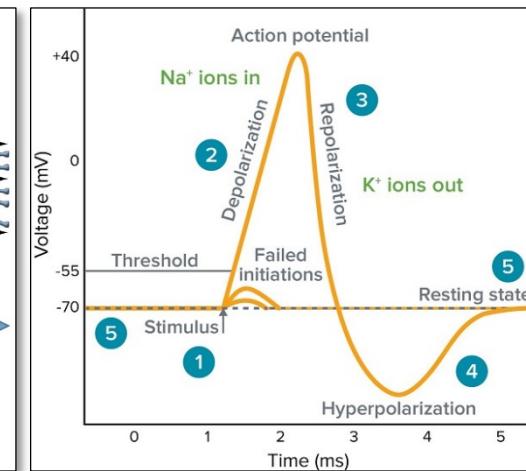
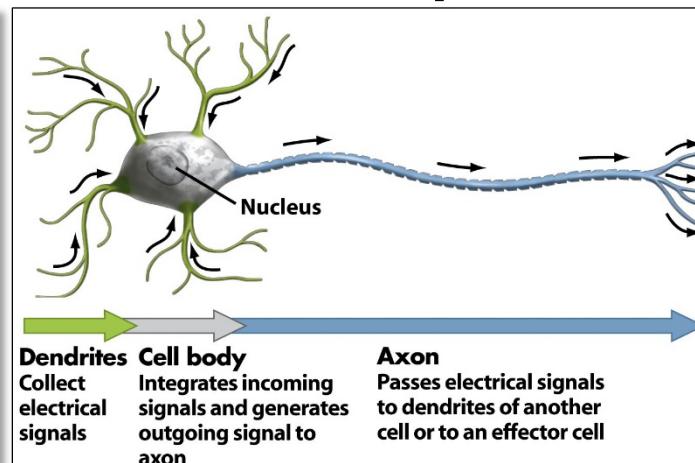
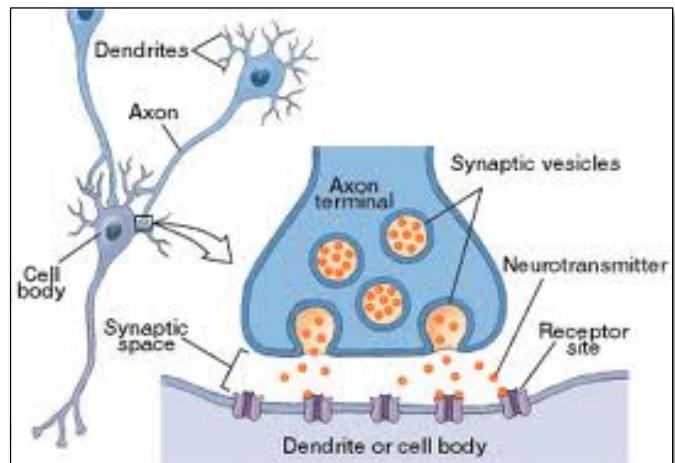


# CNN inspiration in the 50s/60s: human/animal visual cortex

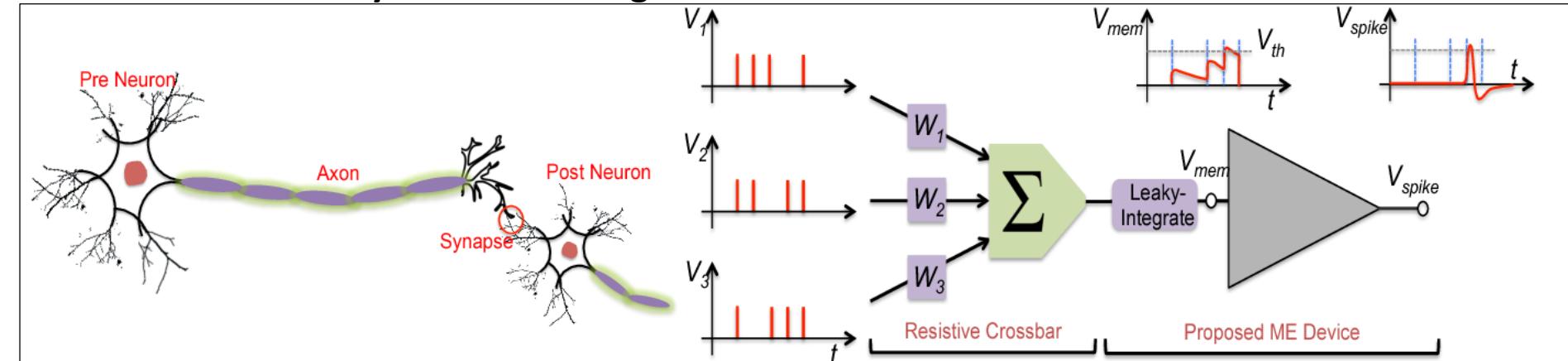


- Hubel/Wisell 1968 cat/monkey: (1) Receptive fields = local computation. (2) Simple cells = edge/orientation detectors. (3) Complex cells = position invariance/pooling
- Layers: pixels, edges (bands given slant, contrast edges), shapes, primitives, scenes
- Hierarchical abstractions, simple building blocks, local computation, learning, invariance

# Primitives: Neurons, action potentials, networks

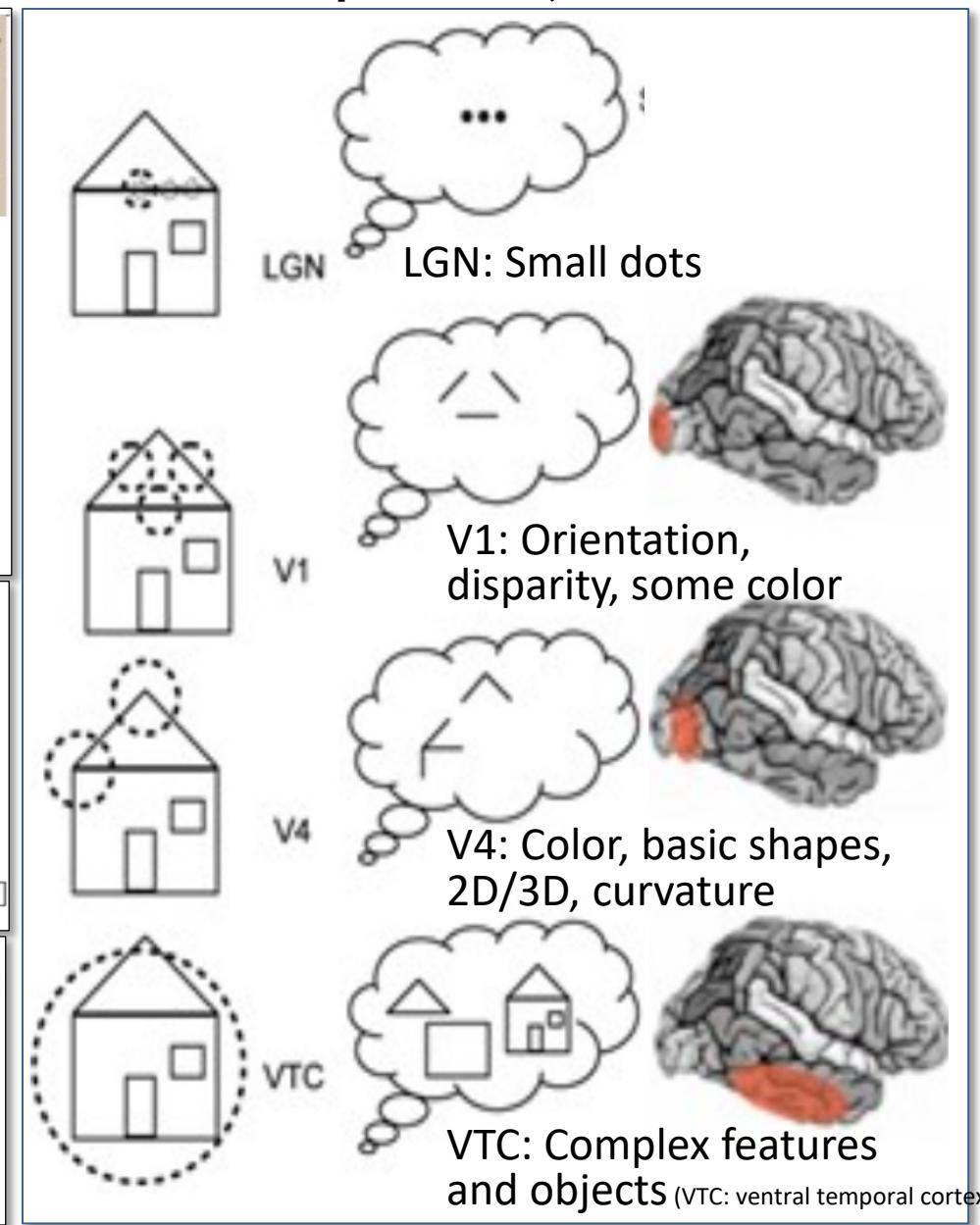
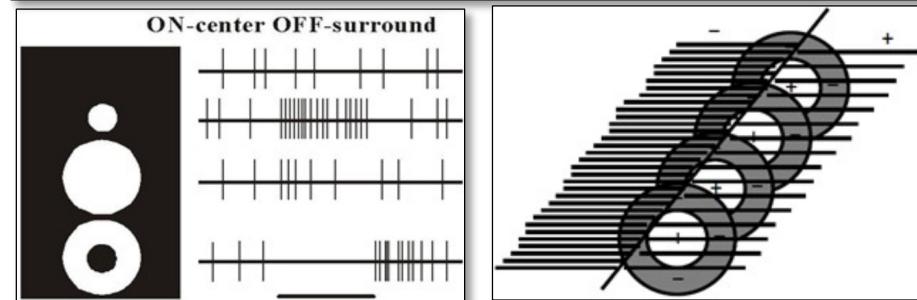
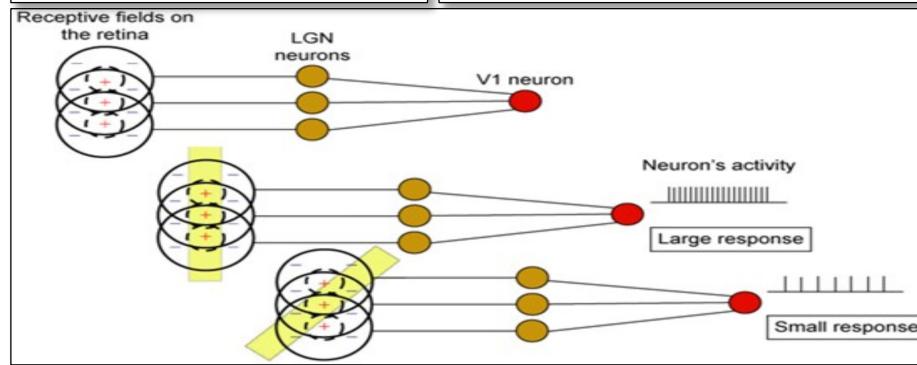
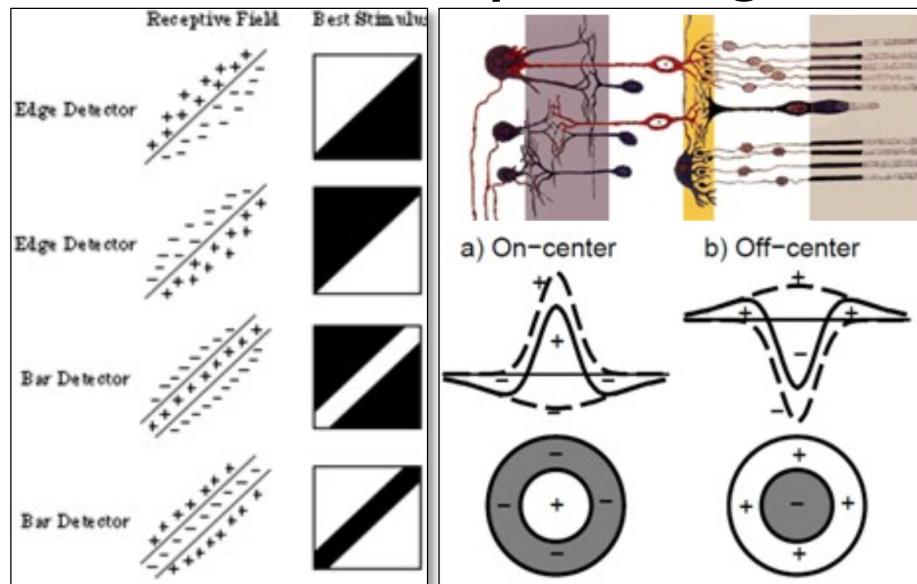


- Chemical accumulation across dendritic connections
- Pre-synaptic axon  
→ post-synaptic dendrite  
→ neuronal cell body
- Each neuron receives multiple signals from its many dendrites
- When **threshold** crossed, it fires
- Its axon then sends outgoing signal to downstream neurons
- Weak stimuli ignored
- **Activation function**: signal threshold crossed
- **Non-linearity** within each neuronal level



- Neurons connected into **circuits** (neural networks): **emergent** properties, **learning**, **memory**
- Simple **primitives** arranged in simple, repetitive, and extremely **large** and **deep** networks
- 86 billion neurons, each connects to 10k neurons, 1 quadrillion ( $10^{12}$ ) connections
- Human brain surprisingly large and powerful given 3lb weight, tiny energy consumption

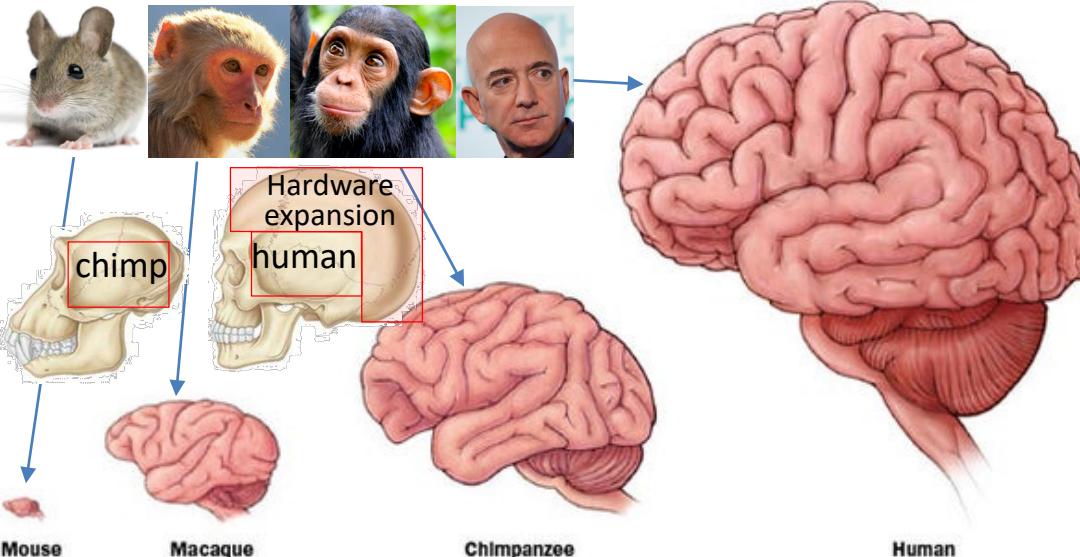
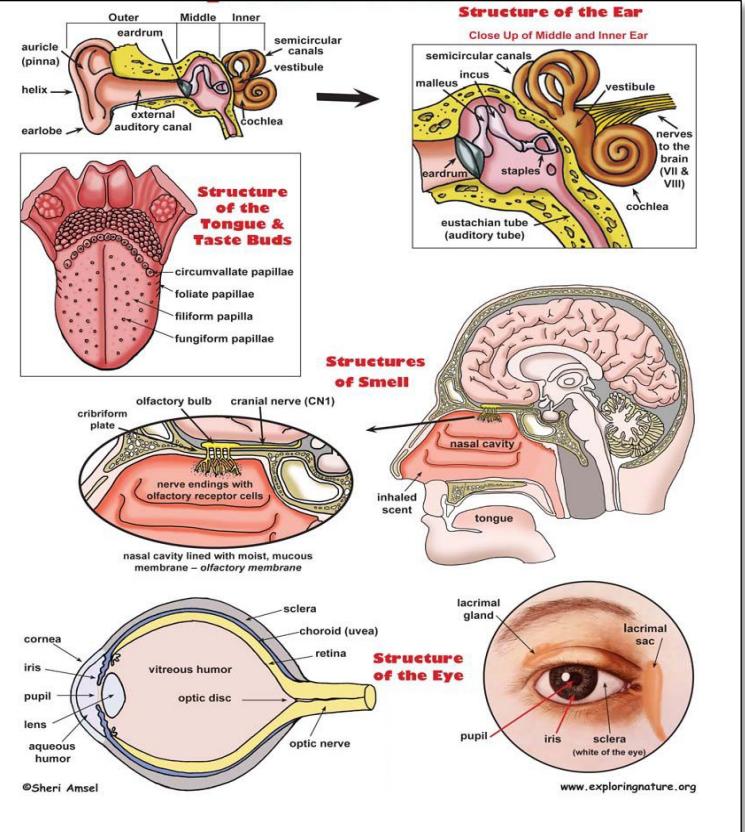
# Abstraction layers: edges, bars, dir., shapes, objects, scenes



- Primitives of visual concepts encoded in neuronal connection in early cortical layers

- Deep: Abstraction layers  $\leftrightarrow$  visual cortex layers
- Complex concepts from simple parts, hierarchy

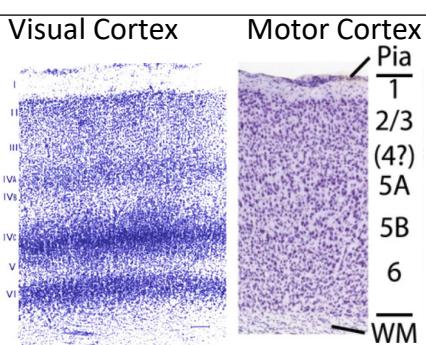
# General “learning machine”, reused widely



- Massive recent expanse of human brain has re-used a relatively simple but general learning architecture

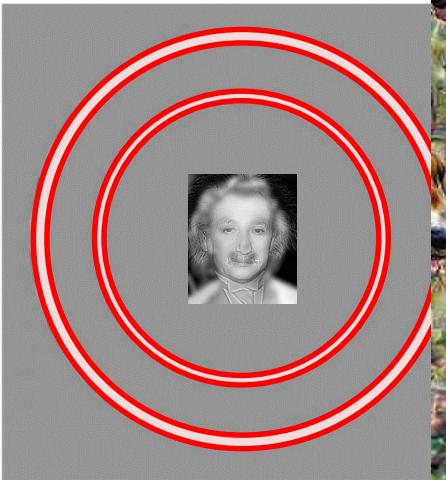


- Learning not fully-general, but well-adapted to our world
- Humans co-opted this circuitry to many new applications
- Modern tasks accessible to any homo sapiens (70k years!)
- ML still similar to animals: room for architecture novelty!



- Interchangeable circuitry
- Auditory cortex learns to ‘see’ if sent visual signals
- Injury area tasks shift to uninjured areas

# Visual illusions send conflicting signals at different filters/layers



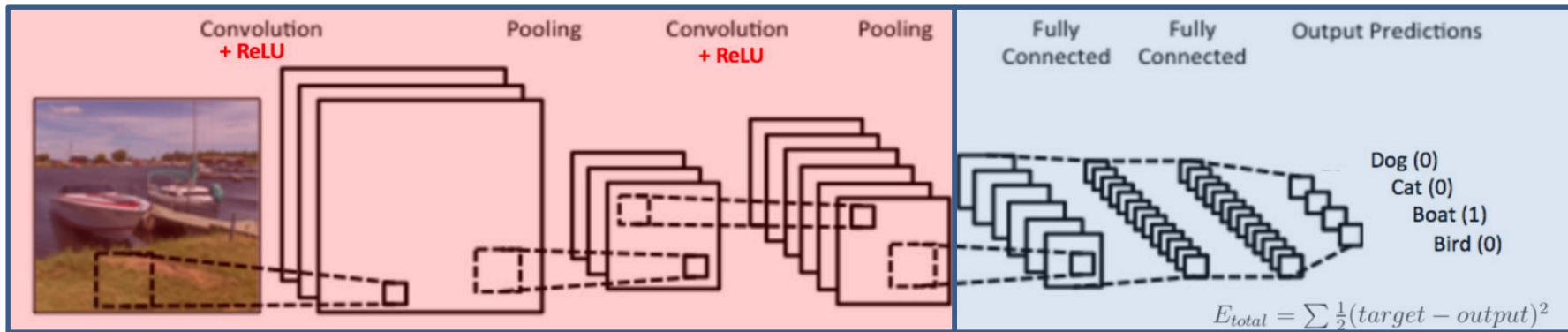
- Visual illusions reveal brain **primitives**, building blocks, computations, architecture
- Deep learning can exploit such **conflicting primitives** to create strong experiences, or for **adversarial** 'confusions' of ML systems

# Key ingredients of a CNN

# Many similarities with the brain

Property	Human Visual System Property	Deep Learning CNN Building Block
Locality	Low-level neurons respond to local patches (receptive field)	Local computation of convolutional filters (not a fully-connected network)
Filters	Specialized neurons carry out low-level detection operation	Low-level filters carry out the same operation throughout the network
Layers / abstraction	Layers of neurons learn increasingly abstract ‘concepts’	Layers of hidden units, abstract concepts learned from simpler parts / building blocks
Threshold	Neurons fire after cross activation threshold → non-linearity	Activation functions introduce non-linearities → expand universe of functions
Pooling	Higher-level neurons invariant to exact position, sum/max of prev.	Max/Avg pooling layers: positional invariance reduced # parameters, speed up compute
Multimodal	Different neurons extract different features of image	Multiple filters applied simultaneously, each captures different aspects of original image
Saturation	Neurons ‘tired’ after activation, signal quiets down	Limiting weight of individual hidden units, dropout learning, regularization
Reinforcement	Useful connections strengthened over time	Back-propagation, adjusting weights across the hierarchy
Feed-forward edges	Neurons with long connections from lower levels to higher ones	Residual networks (ResNets) feed lower-level signal, avoid vanishing gradients

# Key idea: Representation learning



'Modern' Deep learning:  
Hierarchical Representation Learning  
Feature extraction

'Classical' Fully-connected  
Neural Networks  
Classification

In deep learning, the two tasks are coupled:

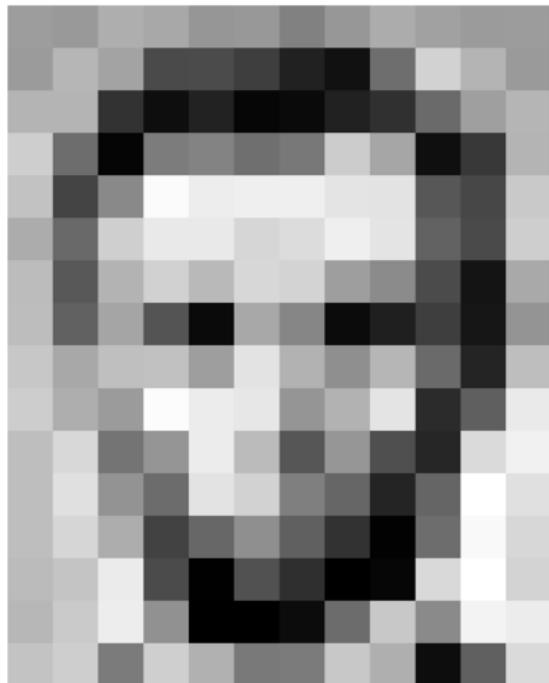
- the **classification task** “drives” the **feature extraction**
- **Extremely powerful and general paradigm**
  - **Be creative!** The field is still at its infancy!
  - New application domains (e.g. beyond images) can have **structure** that current architectures **do not capture/exploit**
  - Genomics/biology/neuroscience can help drive development of **new architectures**

# Today: use these primitives to ‘learn’ complex scenes



# CNNs = Translating pixels to concepts

What you see



Input Image

What you both see

157	153	174	168	150	152	129	151	172	161	155	156
156	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	84	6	10	33	48	106	159	181
206	109	5	124	191	111	120	204	166	15	56	180
194	68	137	251	257	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	209	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Input Image + values

What the computer "sees"

157	153	174	168	150	152	129	151	172	161	155	156
156	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	84	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	209	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixel intensity values  
("pix-el"=picture-element)

An image is just a matrix of numbers [0,255]. i.e., 1080x1080x3 for an RGB image.

Question: is this Lincoln? Washington? Jefferson? Obama?

How can the computer answer this question?

**Can I just do classification on the 1,166400-long image vector directly?**

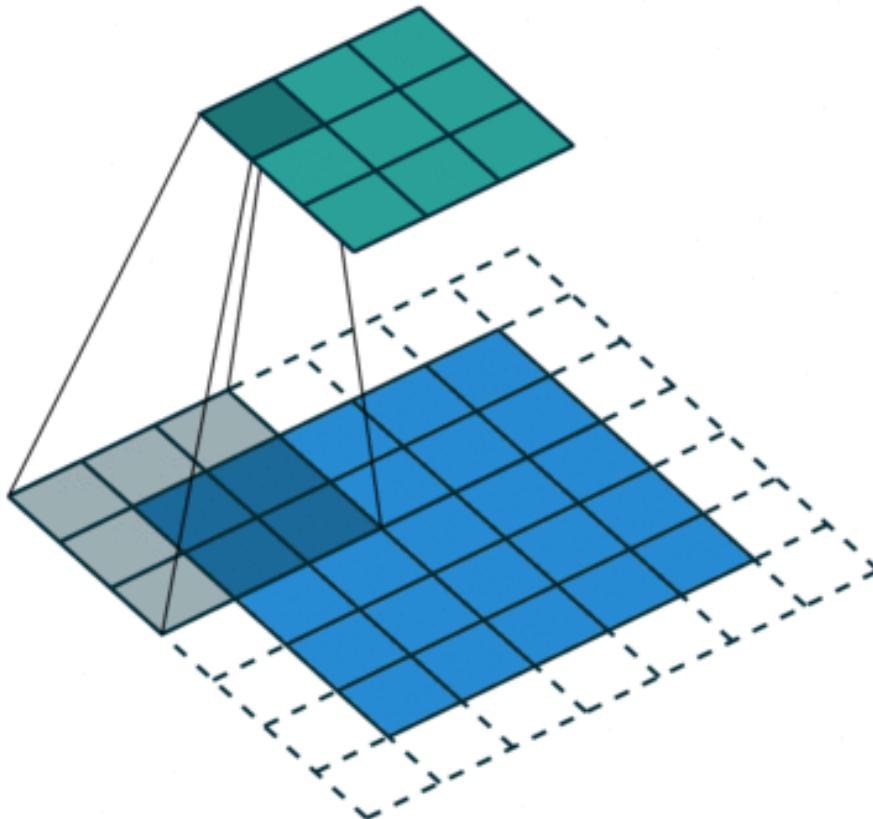
**No. Instead: exploit image spatial structure. Learn patches. Build them up**

Convolutions:  
Spatial structure, local  
computation, shared parameters

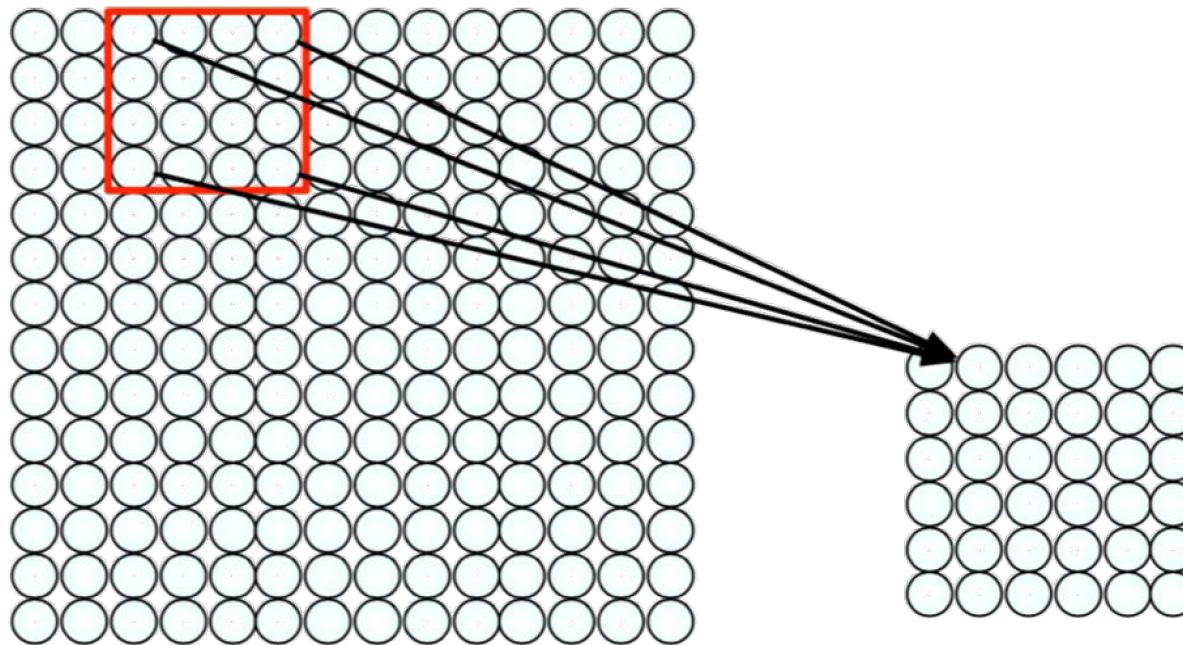
## Key idea: re-use parameters

Convolution shares parameters

Example 3x3 convolution on a 5x5 image

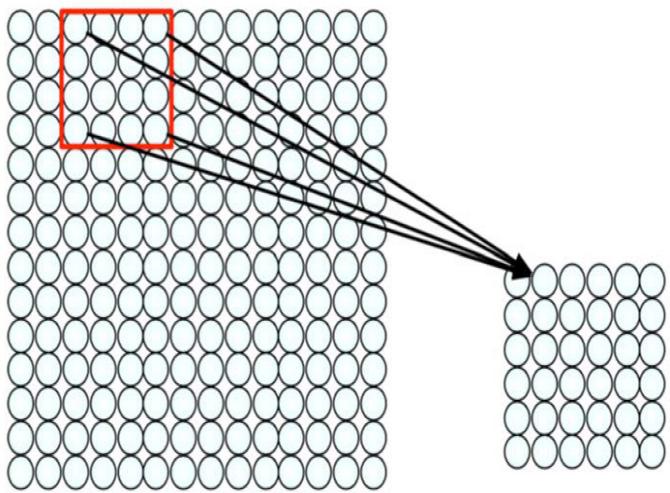


# Feature Extraction with Convolution



- 1) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) **Spatially share** parameters of each filter

# Feature Extraction with Convolution

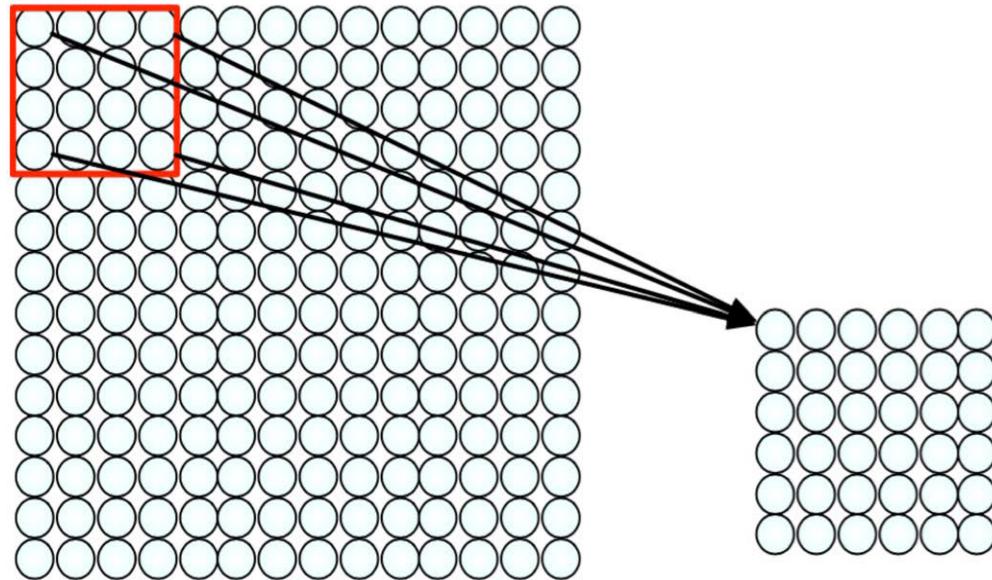


- Filter of size  $4 \times 4$  : 16 different weights
- Apply this same filter to  $4 \times 4$  patches in input
- Shift by 2 pixels for next patch

This “patchy” operation is **convolution**

- 1) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) **Spatially share** parameters of each filter

# Convolutional Layers: Local Connectivity



`tf.keras.layers.Conv2D`

For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

4x4 filter:  
matrix of  
weights  $w_{ij}$

$$\sum_{i=1}^4 \sum_{j=1}^4 w_{ij} x_{i+p,j+q} + b$$

for neuron  $(p,q)$  in hidden layer

- 1) applying a window of weights
- 2) computing linear combinations
- 3) activating with non-linear function

“Representations”

Filters extract Features

# Convolution fundamentals

## CONVOLUTION

### FUNDAMENTALS

### COMPUTER VISION

IMAGE  
CLASSIFICATION



CAT OR  
NOT-CAT

OBJECT  
DETECTION



WHERE IS  
THE CAR?

NEURAL  
STYLE  
TRANSFER



PAINT ME  
LIKE PICASSO

PROBLEM: IMAGES CAN BE BIG  
 $1000 \times 1000 \times 3$  (RGB) = 3M

WITH 1000 HIDDEN UNITS WE  
NEED  $3M + 1000 = 3B$  PARAMS

SOLUTION: USE CONVOLUTIONS  
IT'S LIKE SCANNING OVER YOUR  
IMG WITH A MAGNIFYING GLASS  
OR FILTER

ALSO SOLVES THE PROBLEM  
THAT THE CAT IS NOT  
ALWAYS IN THE SAME  
LOCATION IN THE IMB

## CONVOLUTION

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

INPUT 6x6 IMAGE

$$3+1+2+0+0+0-1-8-2 = -5$$

(3x1)

1	0	-1
1	0	-1
1	0	-1

CONVOLUTION

\*

=

-5	-4	0	8
-16	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

OUTPUT 4x4 IMAGE

VERTICAL  
EDGE DETECTOR

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

INPUT 6x6 IMAGE

\*

=

1	0	-1
1	0	-1
1	0	-1

FILTER 3x3

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

OUTPUT 4x4 IMAGE

DETECTED  
EDGE IN THE MIDDLE

THIS IS LIKE ADDING  
AN 'INSTA' FILTER THAT  
JUST SHOWS OUTLINES

WE COULD HARD-CODE FILTERS · JUST LIKE WE  
CAN HARD-CODE HEURISTIC RULES ... BUT.... A MUCH BETTER  
WAY IS TO TREAT THE FILTER# AS PARAMS  
TO BE LEARNED

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

# Convolution operation is element wise multiply and add

1	0	1
0	1	0
1	0	1

Filter / Kernel

1 <small><math>\times 1</math></small>	1 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	0	0
0 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	1 <small><math>\times 0</math></small>	1	0
0 <small><math>\times 1</math></small>	0 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

# Producing Feature Maps



Original



Sharpen



Edge Detect



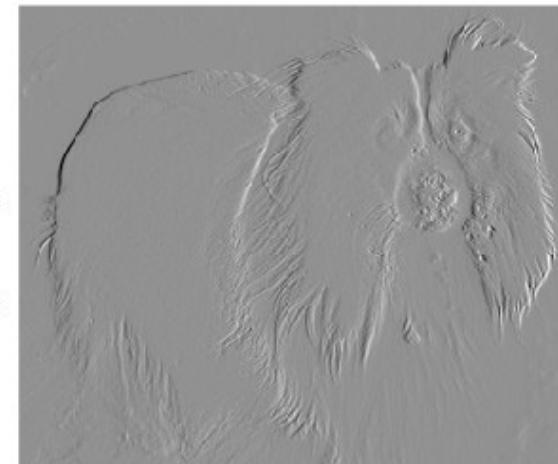
“Strong” Edge  
Detect

# A simple pattern: Edges

## How can we detect edges with a kernel?



Input



Output

1	-1
---	----

Filter

# Simple Kernels / Filters

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Representation Learning:  
Learning convolutional filters:  
extracting common ‘features’

# High Level Feature Detection

Let's identify key features in each image category



Nose, Eyes, Mouth



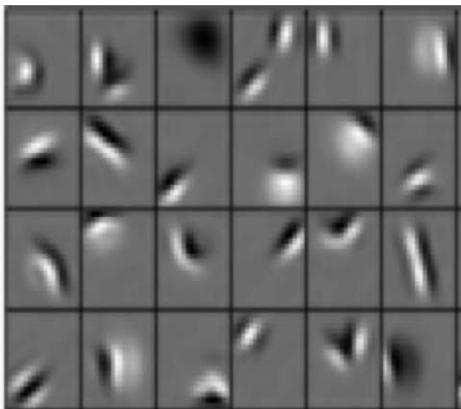
Wheels, License Plate,  
Headlights



Door, Windows, Steps

**Key idea:**  
**learn hierarchy of features**  
**directly from the data**  
(rather than hand-engineering them)

Low level features



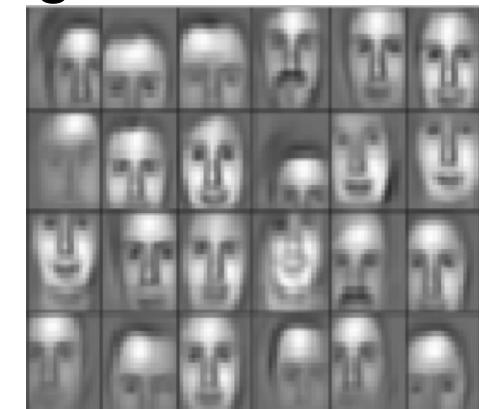
Edges, dark spots

Mid level features



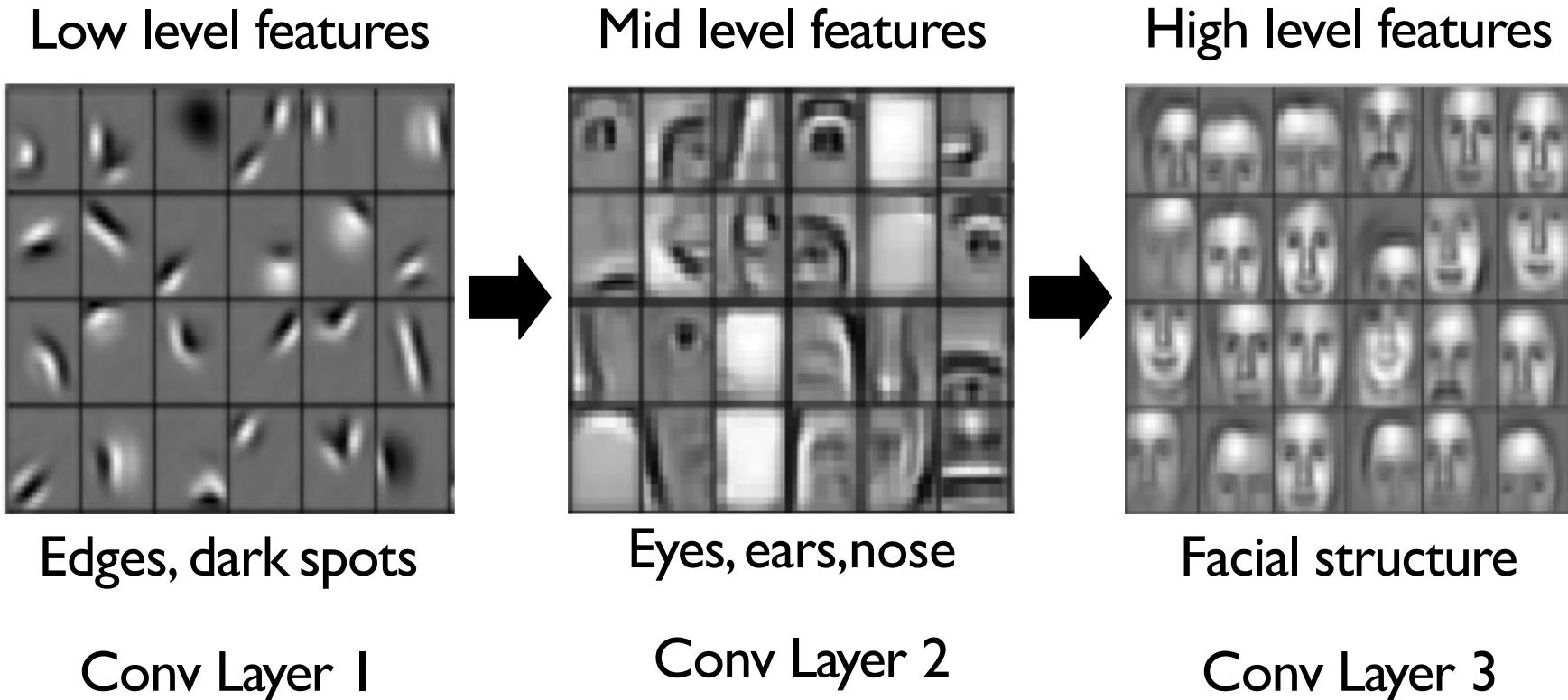
Eyes, ears, nose

High level features



Facial structure

# Representation Learning in Deep CNNs

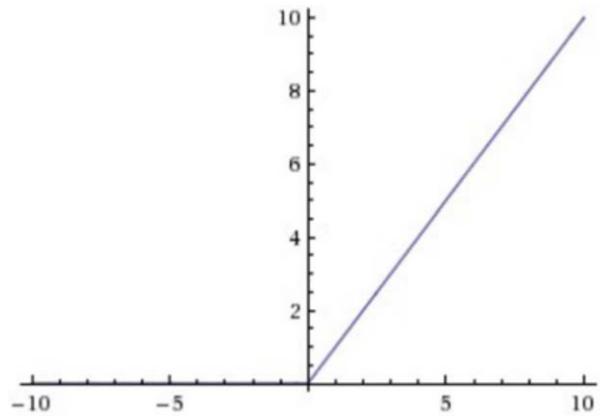
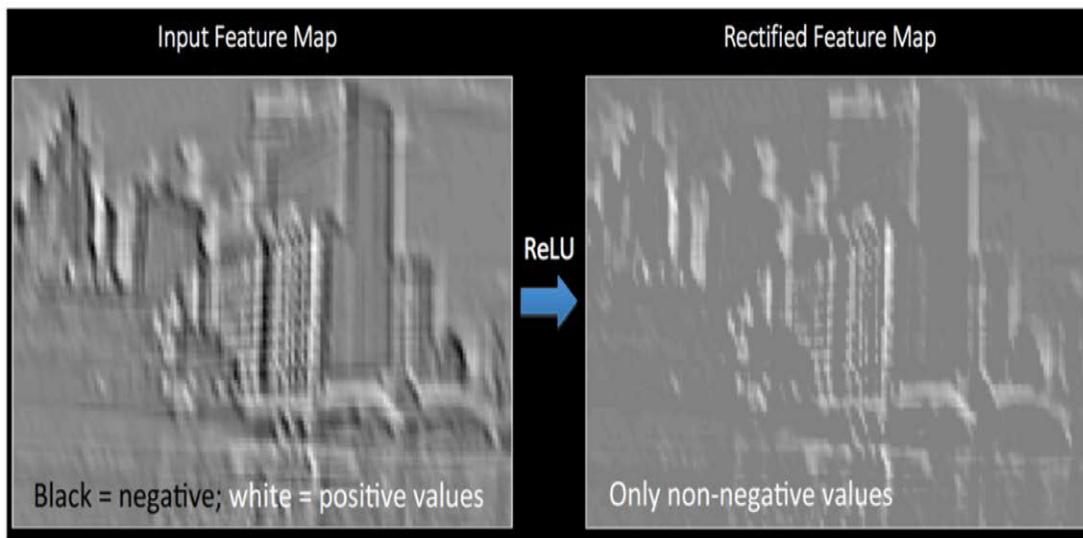


# Detection: Non-Linearities

# Introducing Non-Linearity

- Apply after every convolution operation  
(i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero.
- **Non-linear operation**

Rectified Linear Unit  
(ReLU)



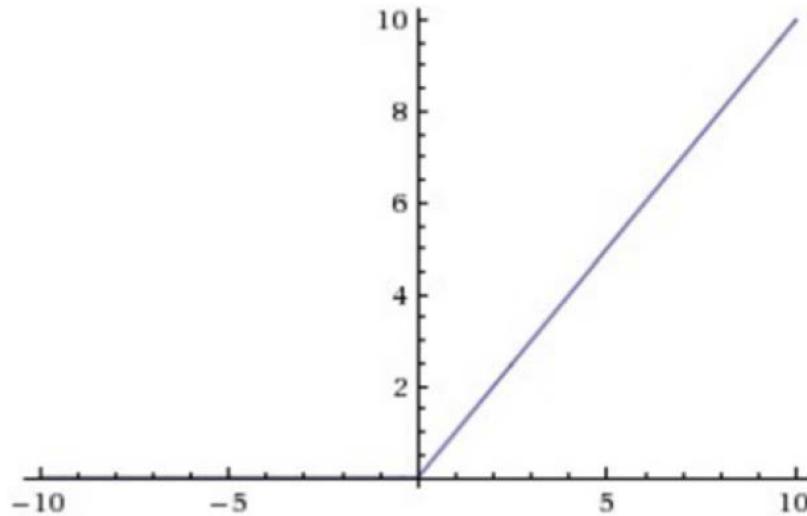
$$g(z) = \max(0, z)$$



`tf.keras.layers.ReLU`

# The REctified Linear Unit (RELU) is a common non-linear **detector** stage after convolution

```
x = tf.nn.conv2d(x, w, strides=[1, strides, strides, 1], padding='SAME')
x = tf.nn.bias_add(x, b)
x= tf.nn.relu(x)
```



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

When will we backpropagate through this?  
Once it “dies” what happens to it?

# Pooling layers: Positional invariance

# Why Pooling

POOLING (MAX)

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

FIND MAX VAL  
IN SECTION

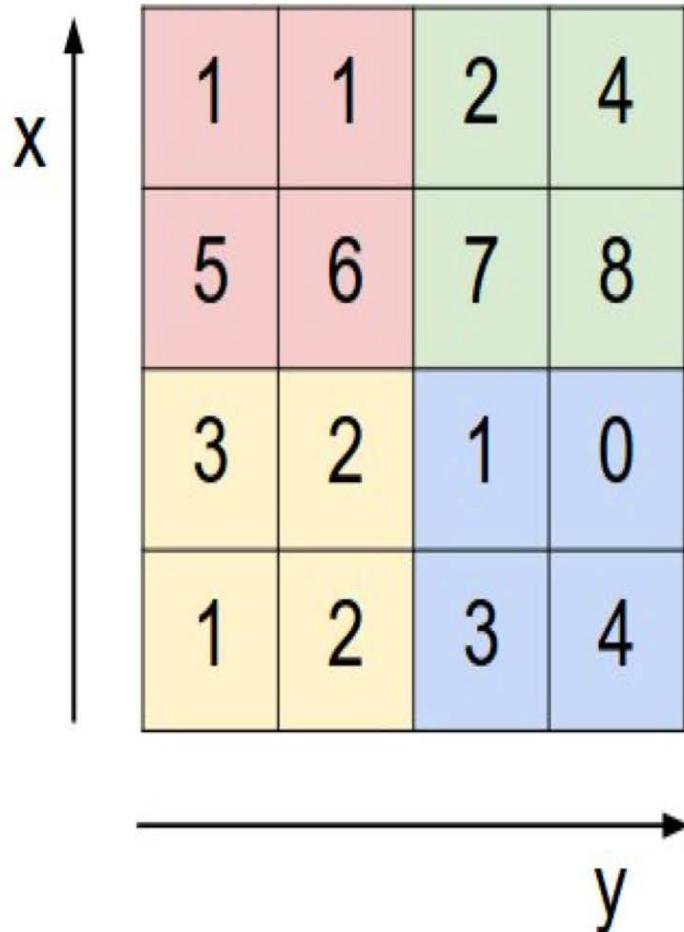
$$f=2 \\ s=2$$

9	2
6	3

HYPERPARAMS

- \* REDUCES SIZE OF REPRES.
- \* SPEEDS UP COMPUTATION
- \* MAKES SOME OF THE DETECTED FEAT. MORE ROBUST

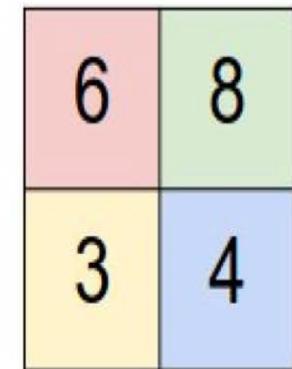
# Pooling



max pool with 2x2 filters  
and stride 2



```
tf.keras.layers.Max  
Pool2D(  
    pool_size=(2, 2),  
    strides=2)
```

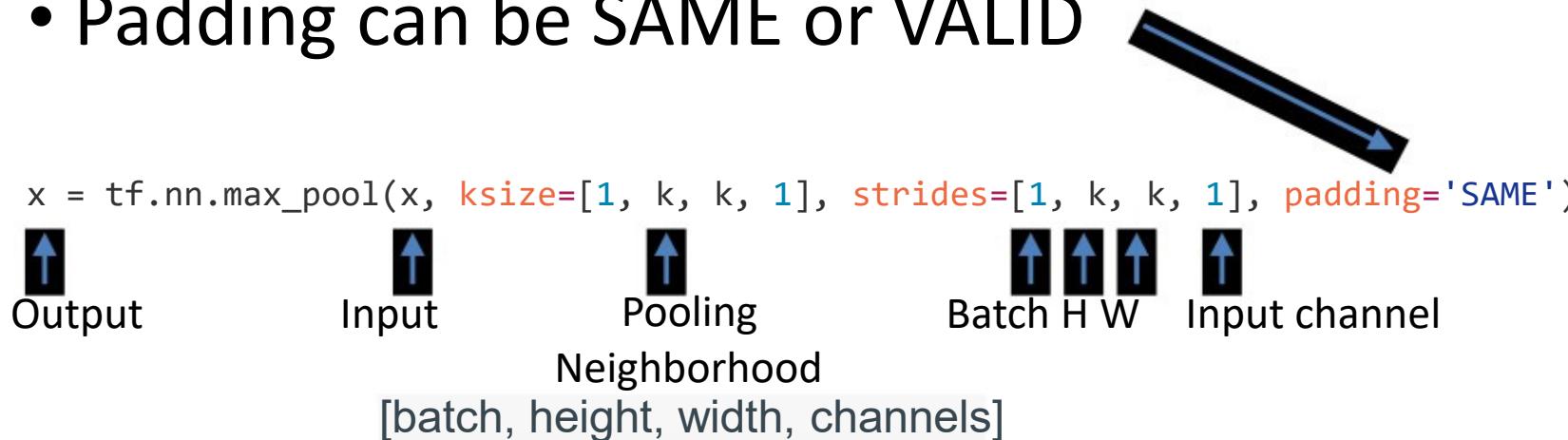


- 1) Reduced dimensionality
- 2) Spatial invariance

Max Pooling, average pooling

# Pooling reduces dimensionality by giving up spatial location

- **max pooling** reports the maximum output within a defined neighborhood
- Padding can be SAME or VALID

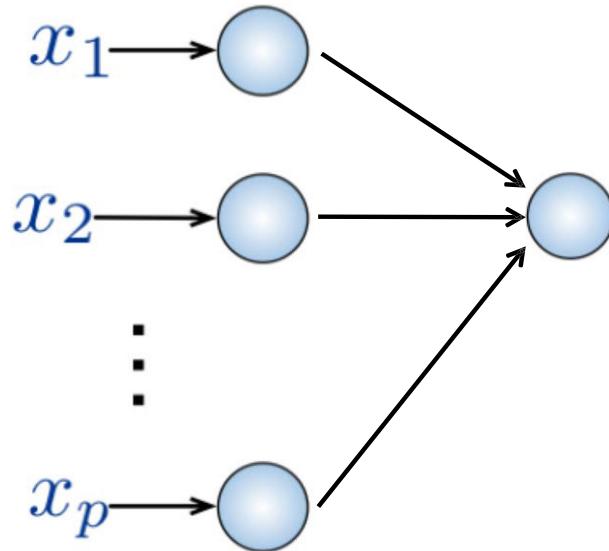


# Classification: fully-connected layers

# Fully Connected Neural Network

## Input:

- 2D image
- Vector of pixel values

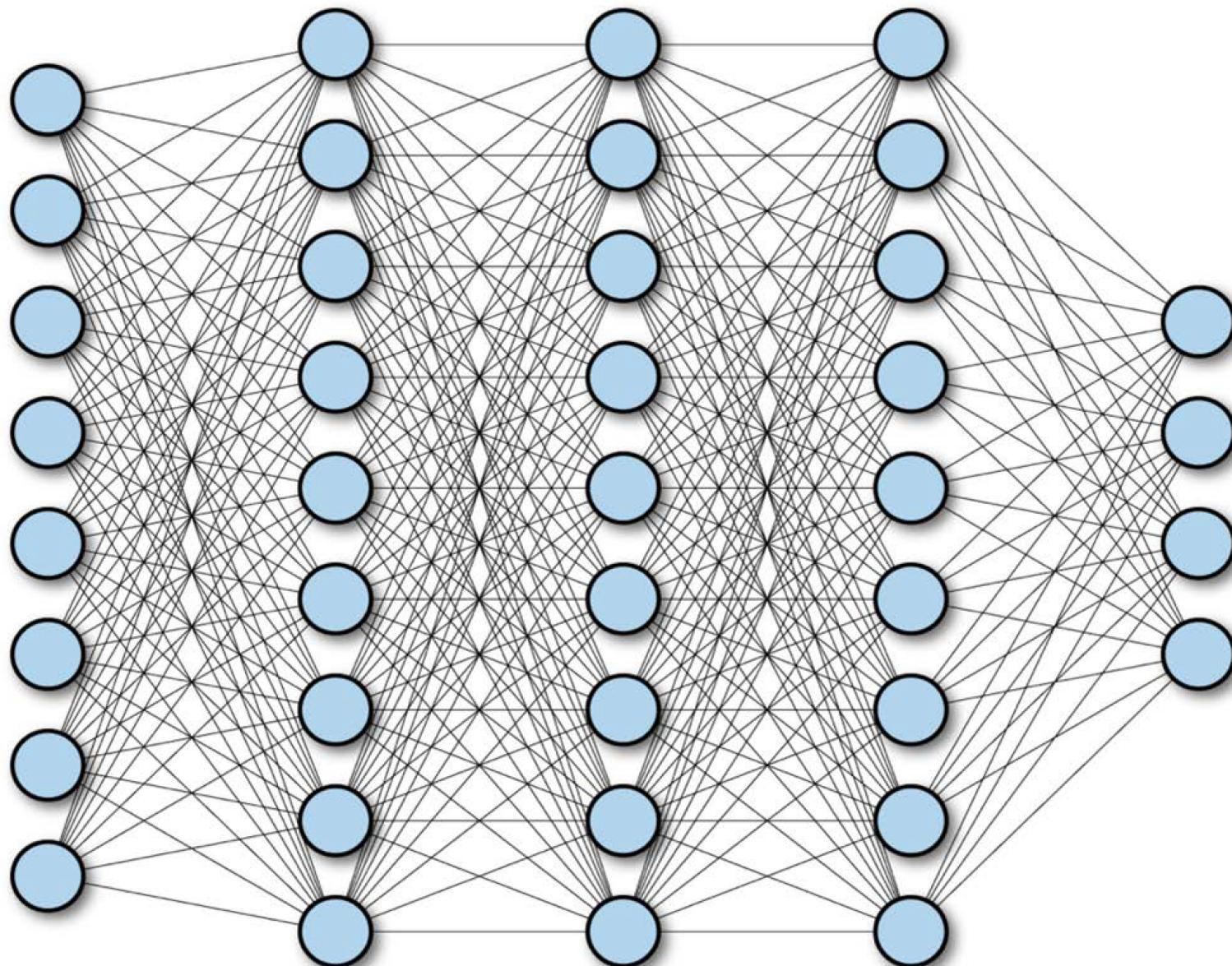


## Fully Connected:

- Each neuron in hidden layer connected to all neurons in input layer
- No spatial information
- Many, many parameters

**Key idea:** Use spatial structure in input to inform architecture of the network

# Fully Connected Neural Network



# Edge cases (literally): Practical issues of convolutions

# Padding

## PADDING

PROBLEM: IMAGES SHRINK

$$6 \times 6 \rightarrow 3 \times 3 \rightarrow 4 \times 4$$

PROBLEM: EDGES GET LESS 'LOVE'

SOLUTION: PAD W. A BORDER  
OF 0s BEFORE CONVOLVING

0	0	0	0	0	0	0	0
0	3	0	1	2	7	4	0
0	1	5	8	9	3	1	0
0	2	7	2	5	1	3	0
0	0	1	3	1	7	8	0
0	4	8	1	6	2	8	0
0	2	4	5	2	3	9	0
0	0	0	0	0	0	0	0

TWO COMMONLY USED  
PADDING OPTIONS

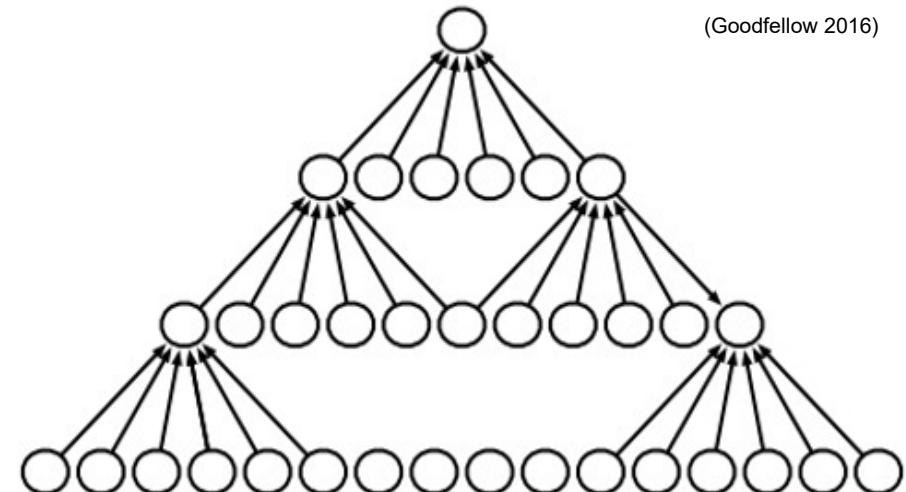
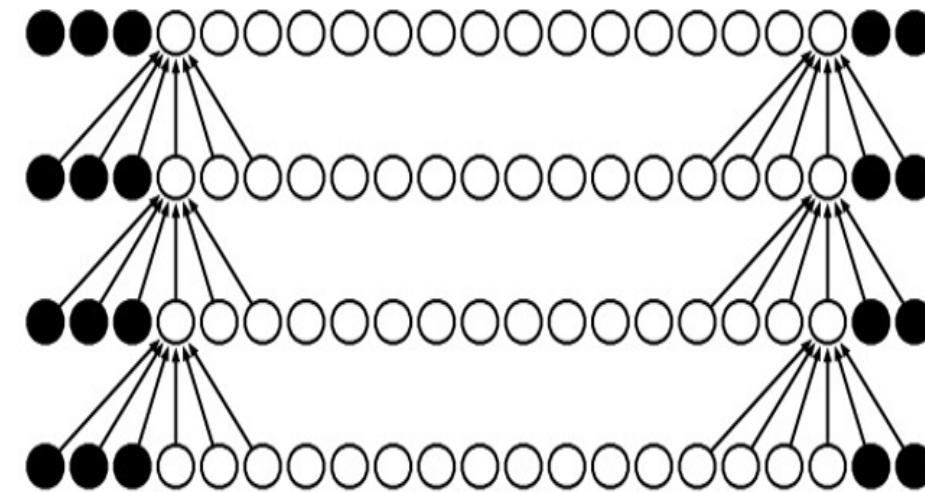
(HOW MUCH TO PAD)

'VALID'  $\Rightarrow P=0$  NO  
PADDING

'SAME'  $\Rightarrow P=\frac{f-1}{2}$  OUTPUT  
SIZE = INPUT  
SIZE  
FILTER  
SIZE

# Zero Padding Controls Output Size

(Goodfellow 2016)



- **Same convolution:** zero pad input so output is same size as input dimensions

- **Full convolution:** zero pad input so output contains at least one input value (expands output)

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

```
x = tf.nn.conv2d(x, W, strides=[1,strides,strides,1], padding='SAME')
```



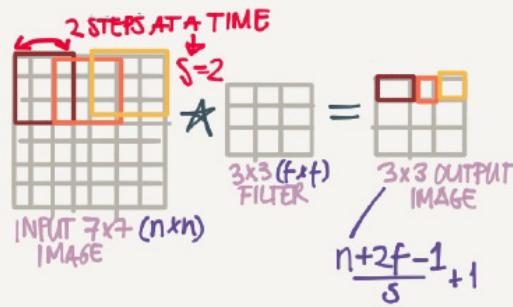
- TF convolution operator takes stride and zero fill option as parameters
- Stride is distance between kernel applications in each dimension
- Padding can be SAME or VALID

# Edge cases (literally): Practical issues of convolutions

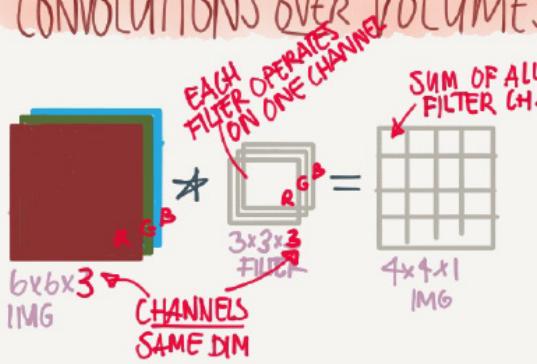
# Stride

## STRIDE

WHAT PACE YOU SCAN WITH



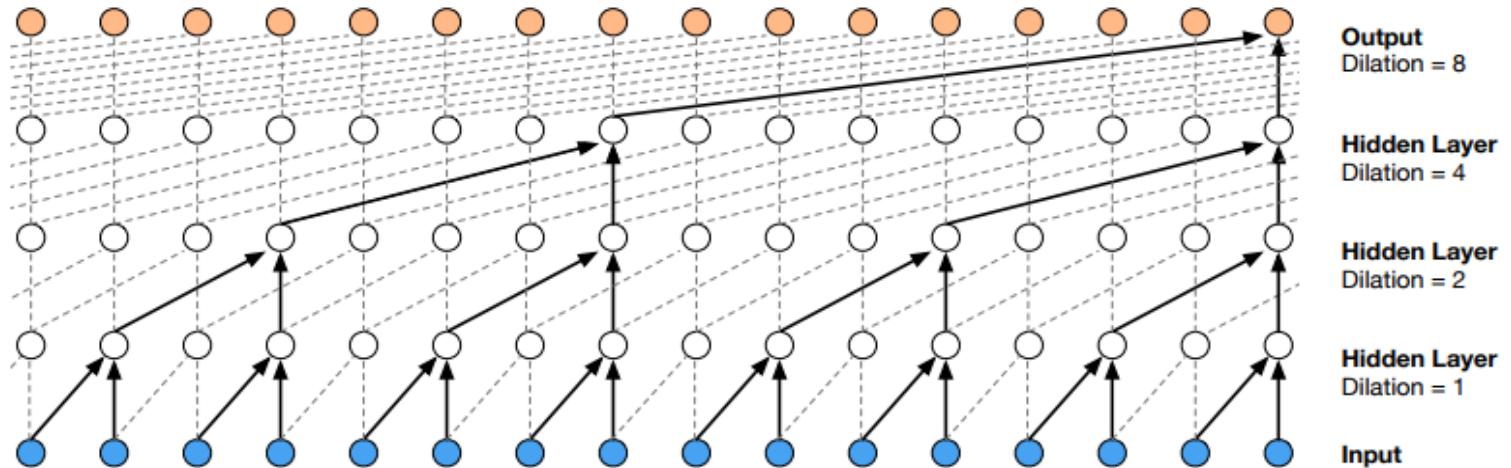
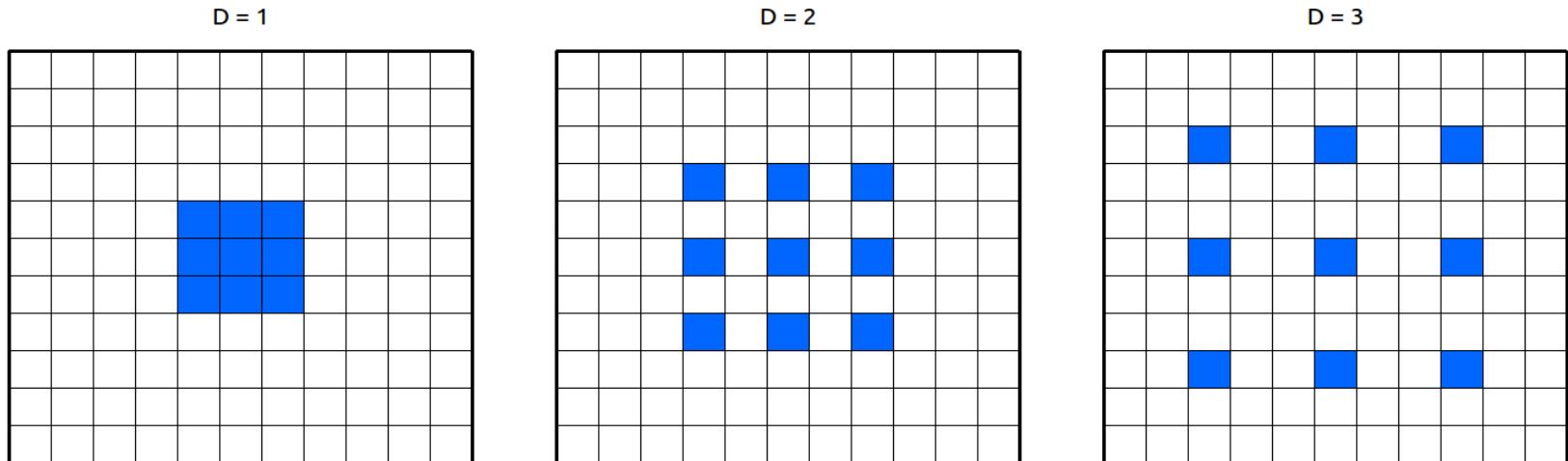
## CONVOLUTIONS OVER VOLUMES



THIS ALLOWS US TO DETECT FEATURES  
IN COLOR IMAGES FOR EXAMPLE

MAYBE WE WANT TO FIND ALL  
EDGES OR MAYBE ORANGE BLOBS

# Dilated Convolution



# Real-world Feature Invariance:

## Data augmentation

# Feature invariance to perturbation is hard

Viewpoint variation



Scale variation



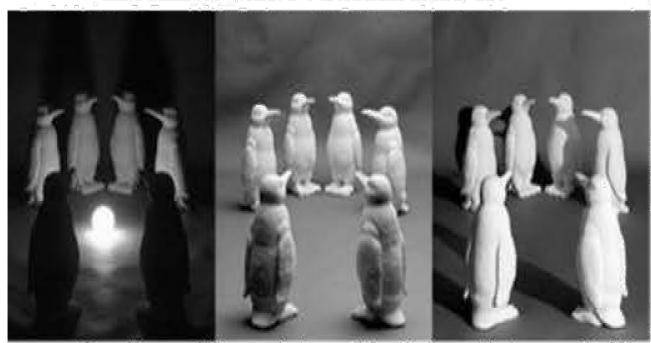
Deformation



Occlusion



Illumination conditions



Background clutter



Intra-class variation



# X or X?

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

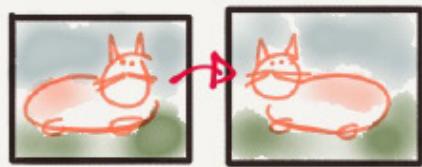


-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

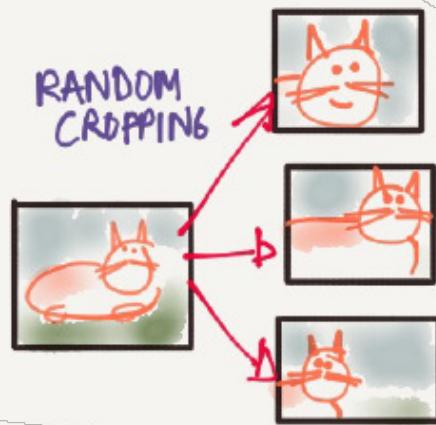
Image is represented as matrix of pixel values... and computers are literal!  
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

# DATA AUGMENTATION

WE ALMOST ALWAYS NEED MORE DATA TO TRAIN ON



MIRRORING



RANDOM CROPPING

ROTATION  
SHEARING  
LOCAL WARPING  
...



COLOR SHIFTING

# How can computers recognize objects?



Challenge:

- Objects can be anywhere in the scene, in any orientation, rotation, color hue, etc.
- How can we overcome this challenge?

Answer:

- Learn a ton of features (millions) from the bottom up
- Learn the convolutional filters, rather than pre-computing them

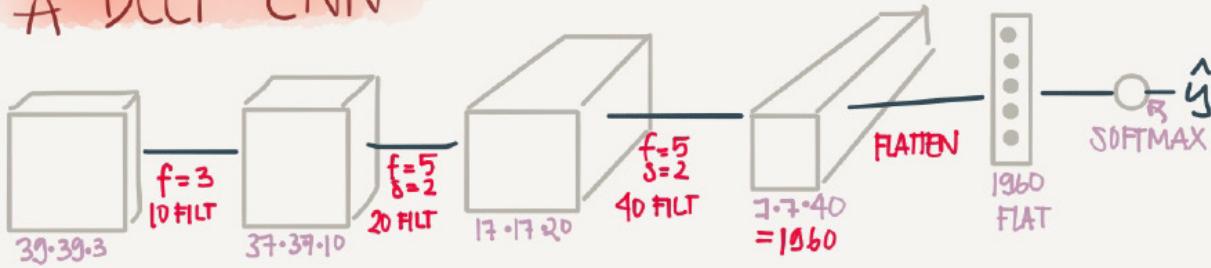
CNNs: Putting all their  
ingredients together

# Many similarities with the brain

Property	Human Visual System Property	Deep Learning CNN Building Block
Locality	Low-level neurons respond to local patches (receptive field)	Local computation of convolutional filters (not a fully-connected network)
Filters	Specialized neurons carry out low-level detection operation	Low-level filters carry out the same operation throughout the network
Layers / abstraction	Layers of neurons learn increasingly abstract ‘concepts’	Layers of hidden units, abstract concepts learned from simpler parts / building blocks
Threshold	Neurons fire after cross activation threshold → non-linearity	Activation functions introduce non-linearities → expand universe of functions
Pooling	Higher-level neurons invariant to exact position, sum/max of prev.	Max/Avg pooling layers: positional invariance reduced # parameters, speed up compute
Multimodal	Different neurons extract different features of image	Multiple filters applied simultaneously, each captures different aspects of original image
Saturation	Neurons ‘tired’ after activation, signal quiets down	Limiting weight of individual hidden units, dropout learning, regularization
Reinforcement	Useful connections strengthened over time	Back-propagation, adjusting weights across the hierarchy
Feed-forward edges	Neurons with long connections from lower levels to higher ones	Residual networks (ResNets) feed lower-level signal, avoid vanishing gradients

# Building blocks of deep convolutional networks

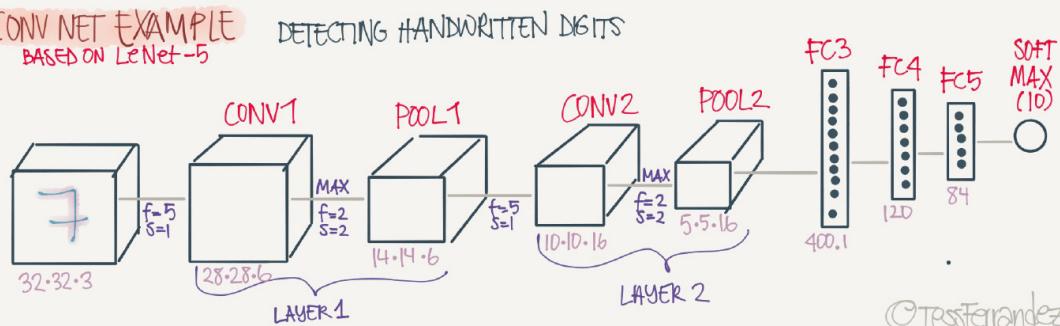
## A DEEP CNN



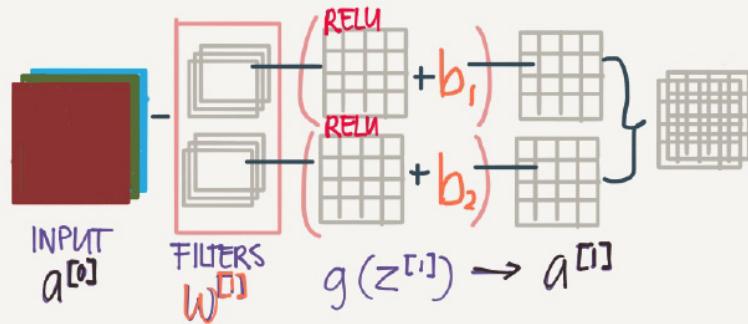
A LOT OF THE WORK IS FIGURING OUT HYPERPARAMS  
= #FILTERS, STRIDE, PADDING ETC

TYPICALLY SIZE → TREND DOWN  
# FILTERS → TREND UP

## CONV NET EXAMPLE BASED ON LeNet-5

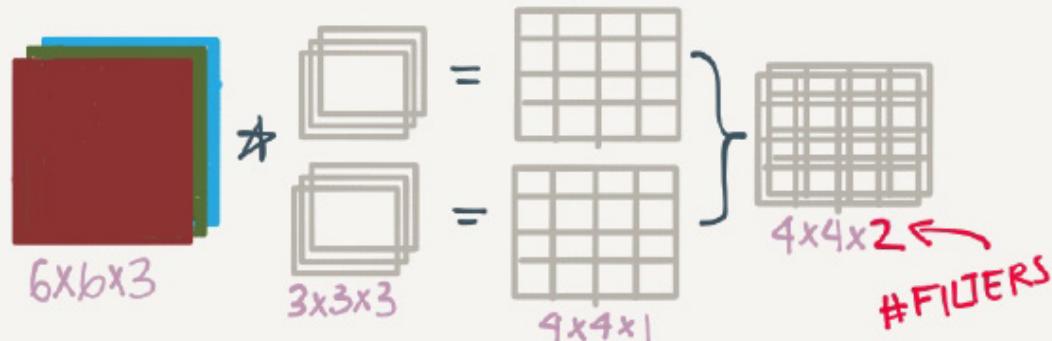


## ONE CONV. NET LAYER



## MULTIPLE FILTERS

### DETECTING MULTIPLE FEATURES AT A TIME



**NOTE** IT DOESN'T MATTER HOW BIG THE INPUT IS - THE LEARNABLE PARAMS  $W$  &  $b$  ONLY DEPEND ON THE # OF FILTERS AND THEIR SIZES.

$$W = 3 \cdot 3 \cdot 3 \cdot 2 = 54 \quad \left. \begin{array}{l} \text{56 PARAMS} \\ \text{TO LEARN} \end{array} \right\}$$
$$b = 2$$

© TessFerrandez

# Putting it all together

```
import tensorflow as tf

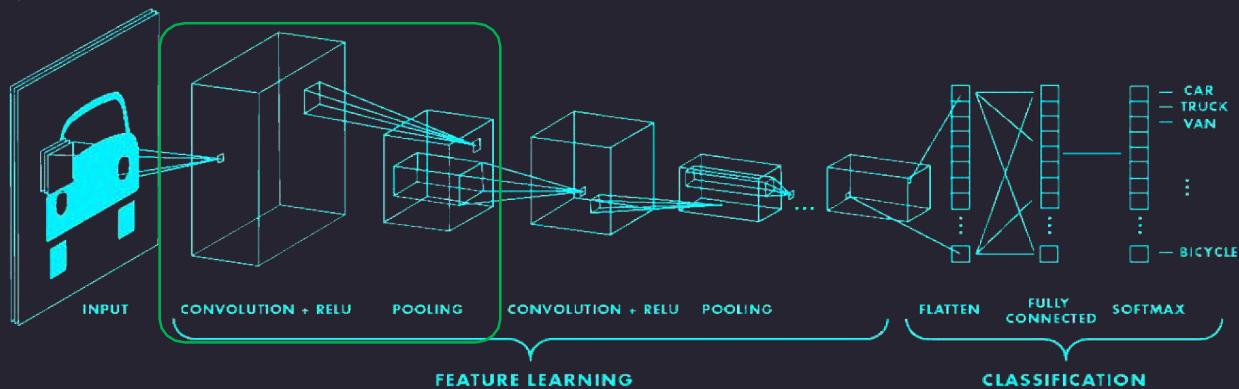
def generate_model():
    model = tf.keras.Sequential([
        # first convolutional layer
        tf.keras.layers.Conv2D(32, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    ])

    # second convolutional layer
    tf.keras.layers.Conv2D(64, filter_size=3, activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

    # fully connected classifier
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')

    # 10 outputs
])

return model
```



# LeNet-5

- *Gradient Based Learning Applied To Document Recognition - Y. Lecun, L. Bottou, Y. Bengio, P. Haffner; 1998*
- Helped establish how we use CNNs today
- Replaced manual feature extraction

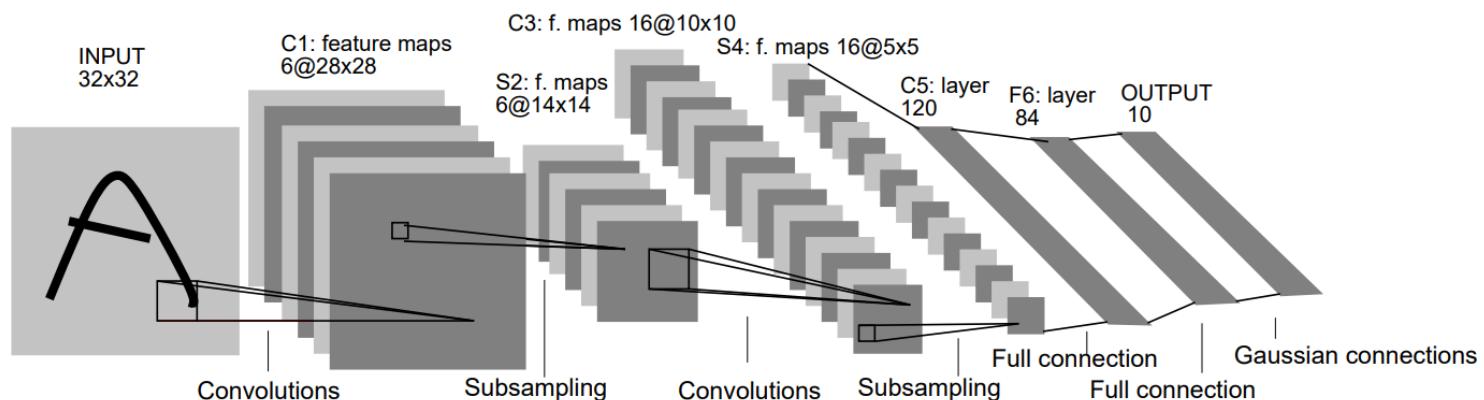
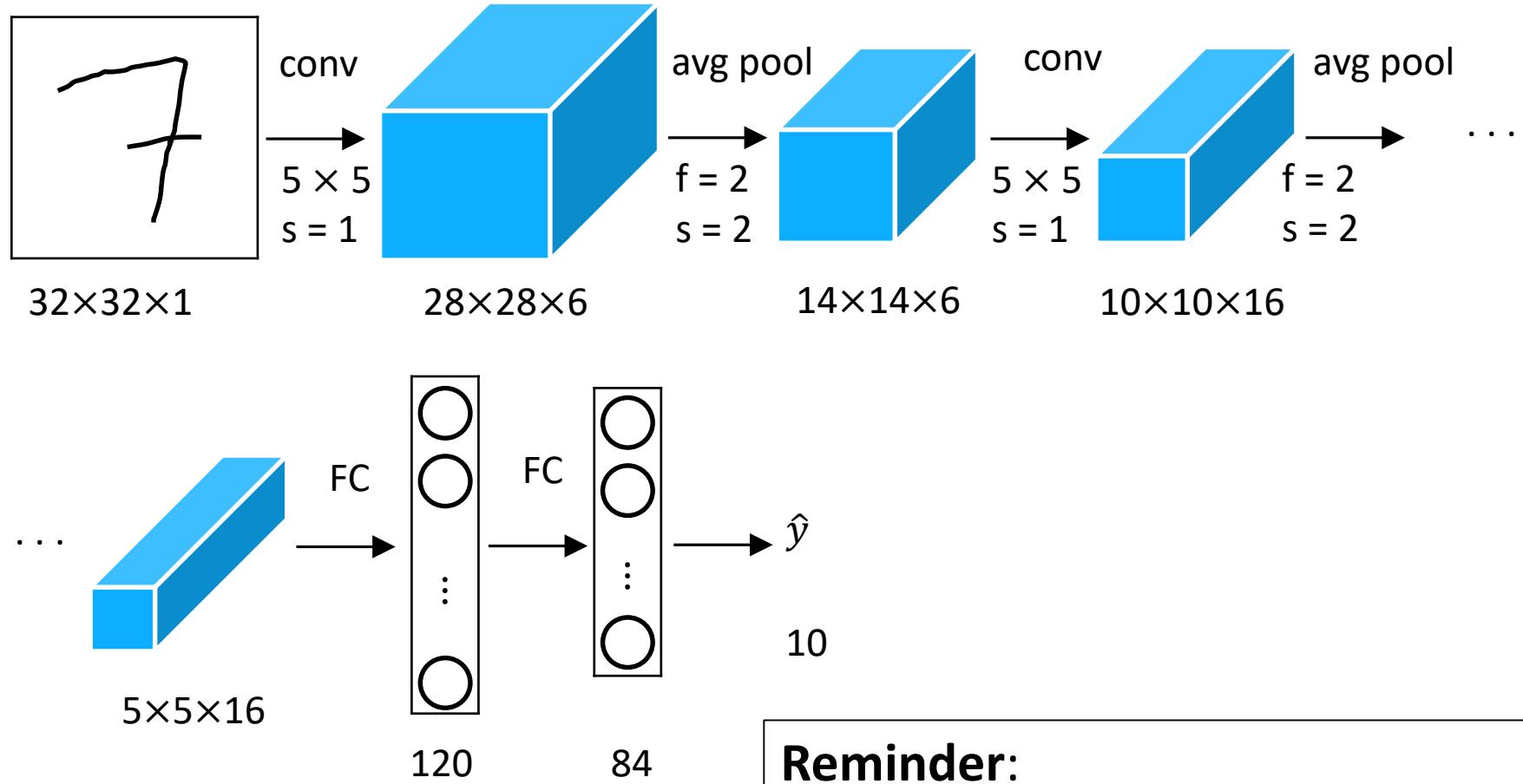


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

# LeNet-5



**Reminder:**

Output size =  $(N+2P-F)/\text{stride} + 1$



# LeNet-5

- Only 60K parameters
  - As we go deeper in the network:  $N_H \downarrow$ ,  $N_W \downarrow$ ,  $N_C \uparrow$
  - General structure:  
 $\text{conv} \rightarrow \text{pool} \rightarrow \text{conv} \rightarrow \text{pool} \rightarrow \text{FC} \rightarrow \text{FC} \rightarrow \text{output}$
- 

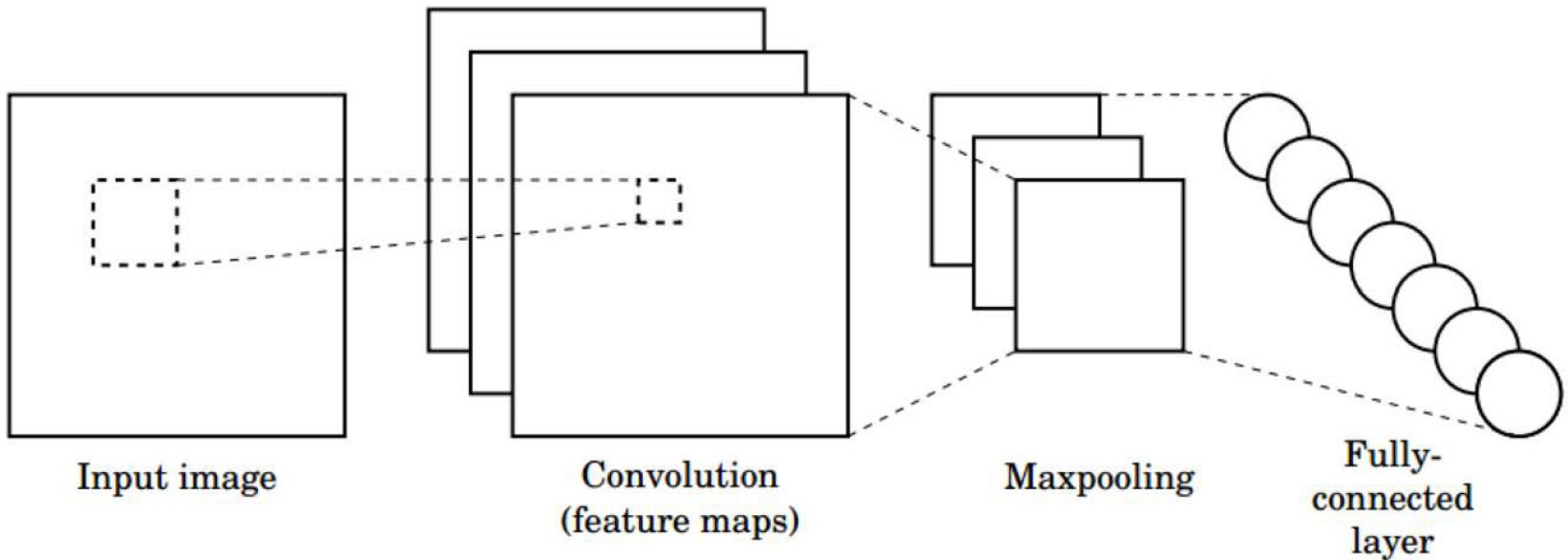
- Different filters look at different channels
- Sigmoid and Tanh nonlinearity

# Backpropagation of convolution

$$\begin{matrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{matrix} = \text{Convolution} \left( \begin{matrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{matrix}, \begin{matrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{matrix} \right)$$

$$\begin{matrix} \partial E / \partial F_{11} & \partial E / \partial F_{12} \\ \partial E / \partial F_{21} & \partial E / \partial F_{22} \end{matrix} = \text{Convolution} \left( \begin{matrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{matrix}, \begin{matrix} \partial E / \partial O_{11} & \partial E / \partial O_{12} \\ \partial E / \partial O_{21} & \partial E / \partial O_{22} \end{matrix} \right)$$

# CNNs for Classification



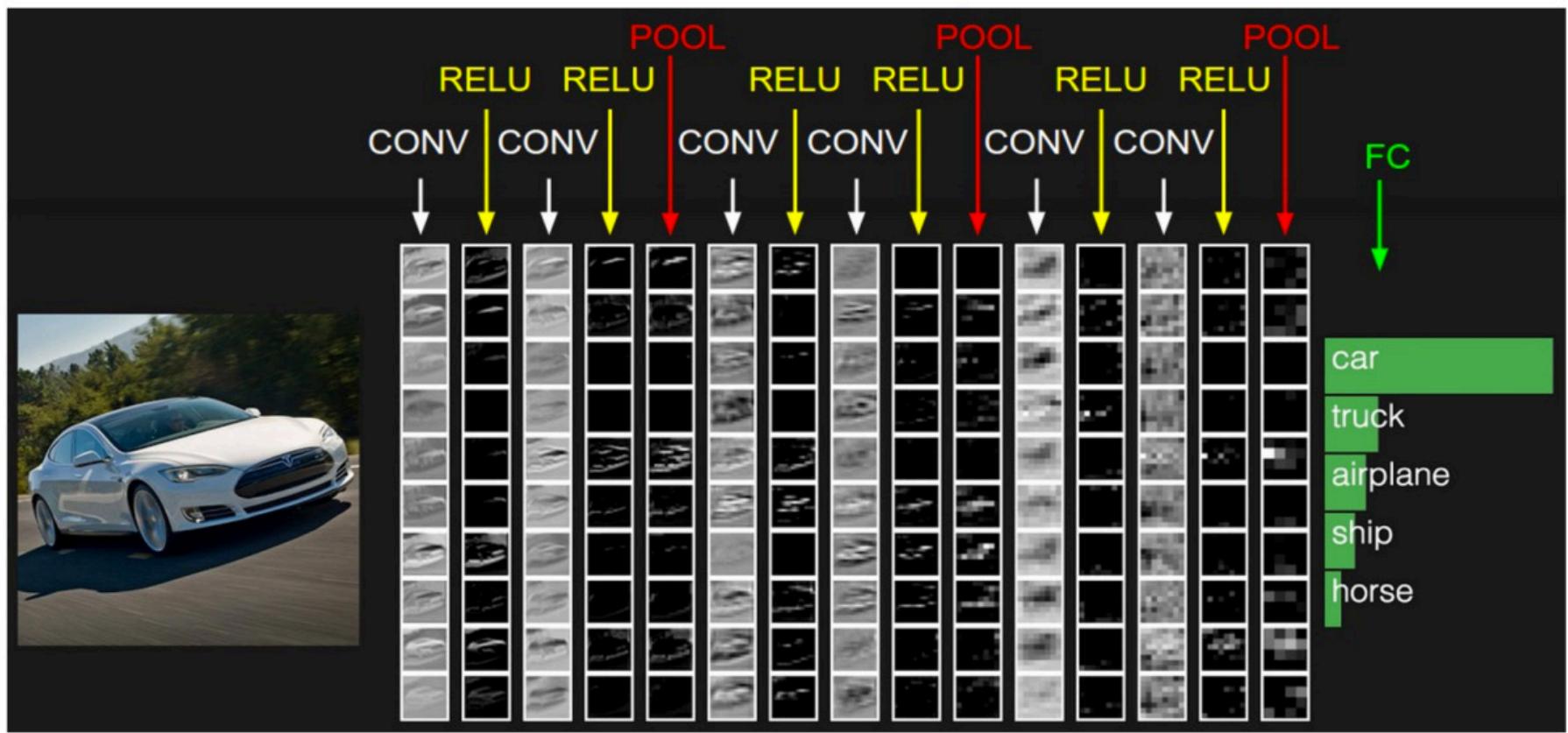
- 1. Convolution:** Apply filters to generate feature maps.
- 2. Non-linearity:** Often ReLU.
- 3. Pooling:** Downsampling operation on each feature map.

**Train model with image data.**

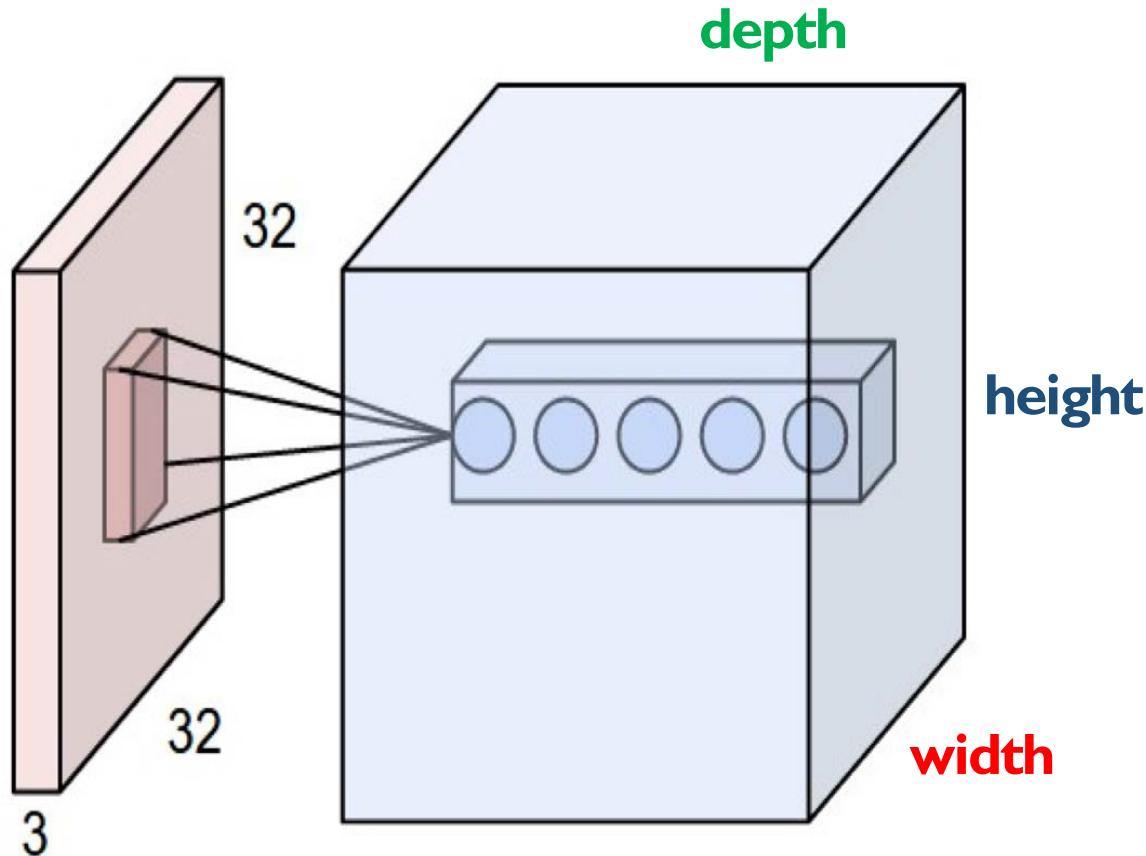
**Learn weights of filters in convolutional layers.**

tf.keras.layers.Conv2D  
tf.keras.activations.\*  
tf.keras.layers.MaxPool2D

# Example – Six convolutional layers



# CNNs: Spatial Arrangement of Output Volume



**Layer Dimensions:**

$$h \times w \times d$$

where h and w are spatial dimensions  
d (depth) = number of filters

**Stride:**

Filter step size

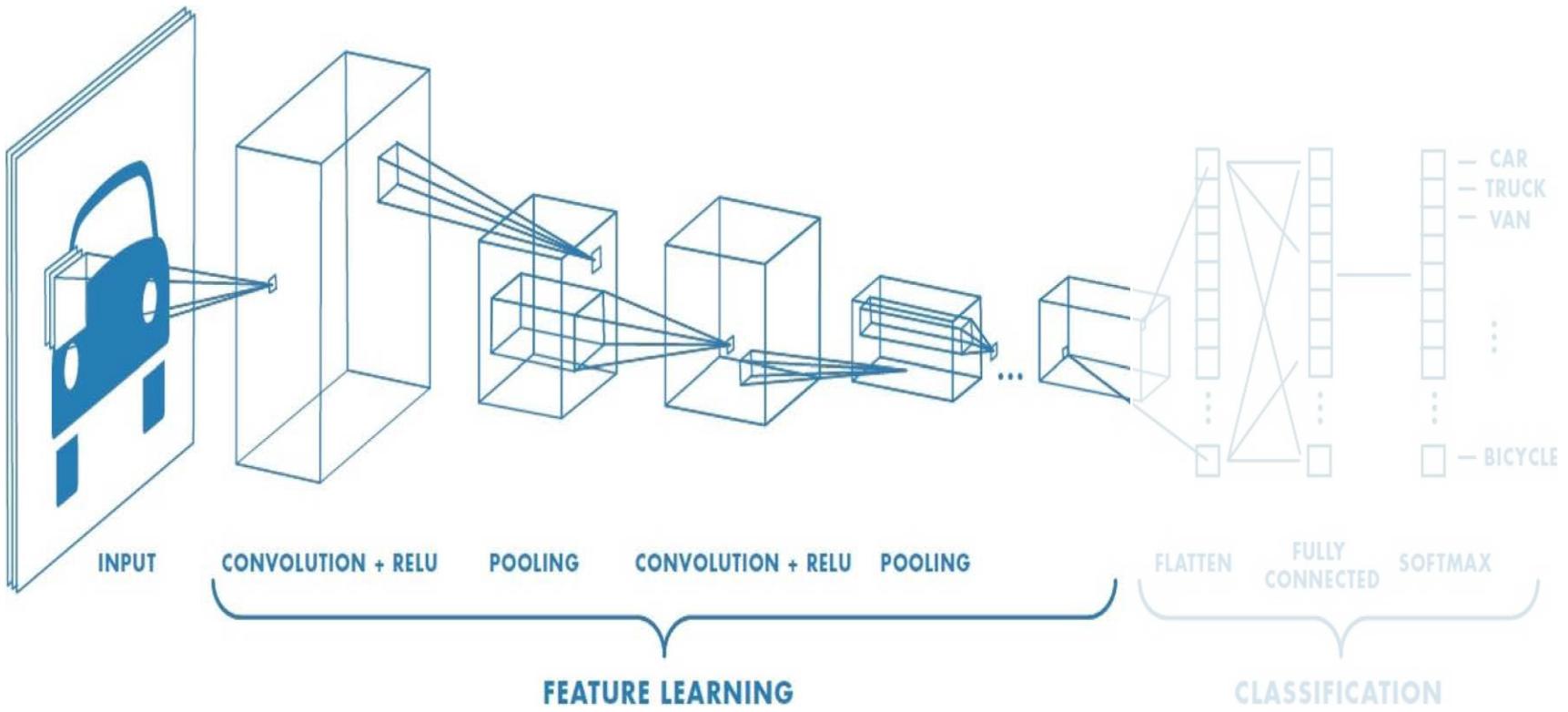
**Receptive Field:**

Locations in input image that a node is path connected to



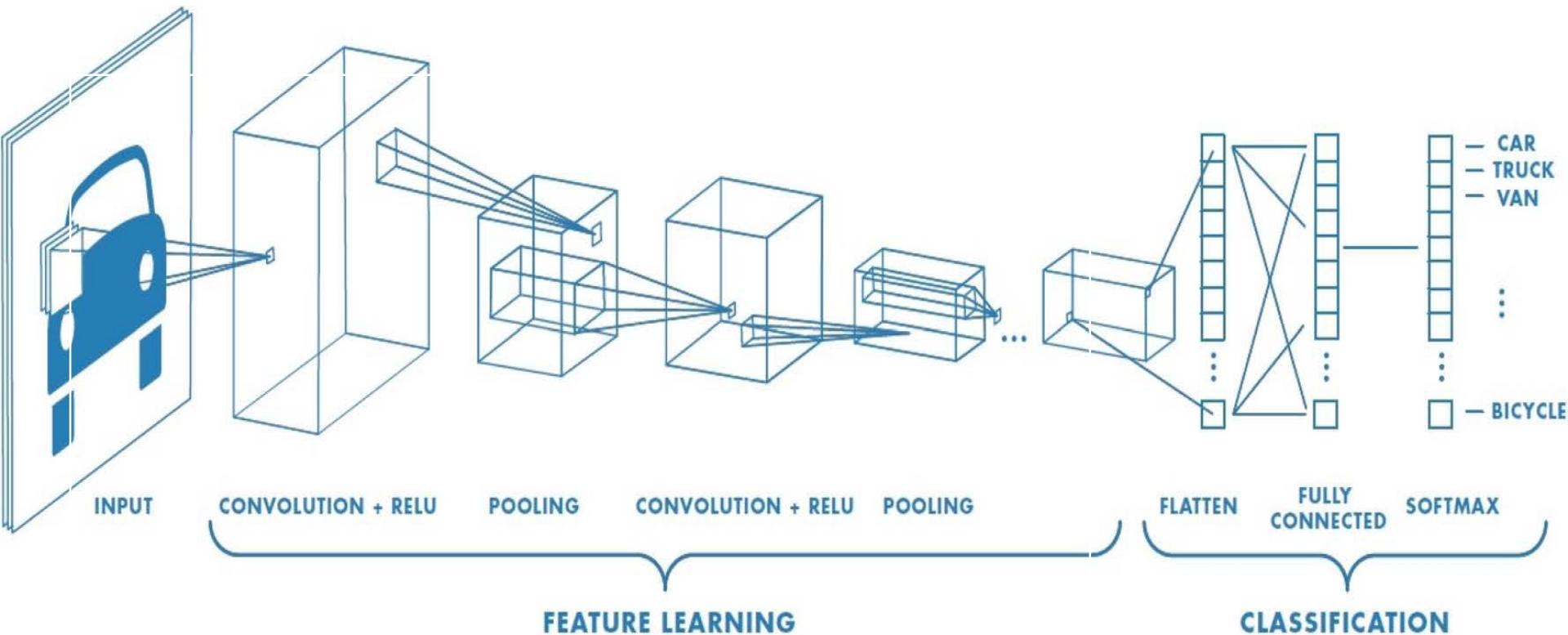
```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

# CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

# CNNs for Classification: Class Probabilities

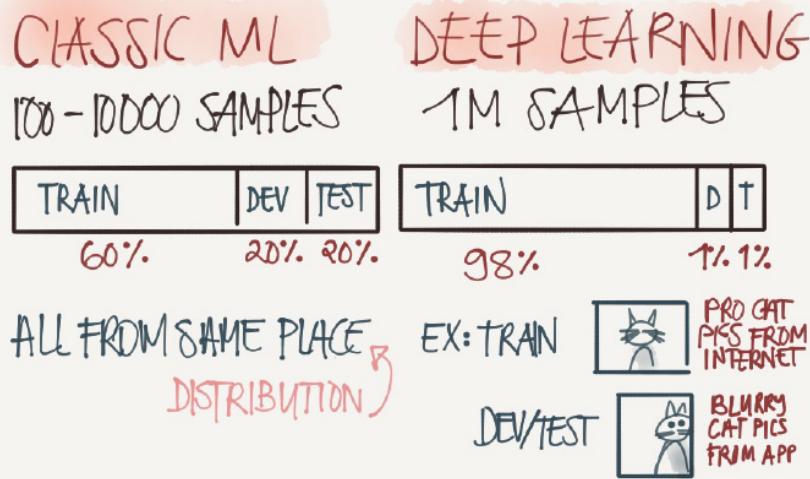


$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

# The art of CNN training

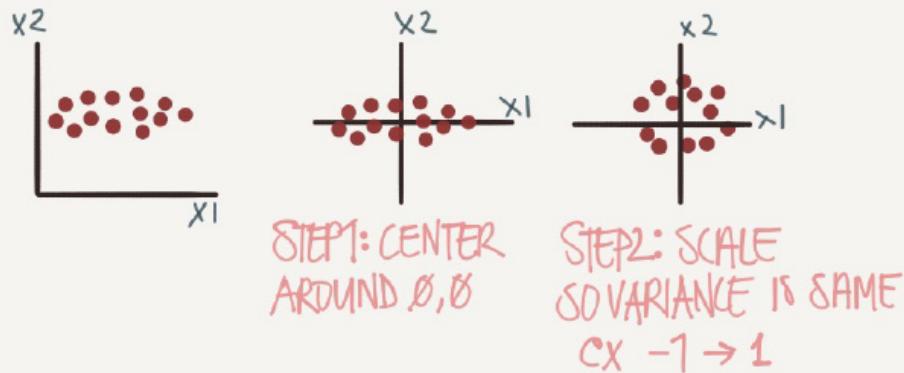
# Foundations of CNN training



- Needs lots of data for training

# Normalization matters

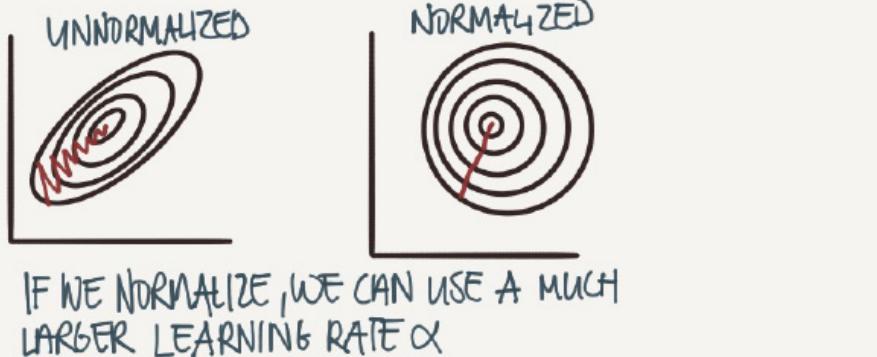
## NORMALIZING INPUTS



TIP

USE SAME AVG/VAR TO  
NORMALIZE DEV/TEST

## WHY DO WE DO THIS?



# Vanishing / exploding gradients

DEALING WITH  
VANISHING/EXPLODING  
GRADIENTS

Ex: DEEP NW (L LAYERS)

$$\hat{y} = \underbrace{w^{[L-1]} w^{[L-2]} \dots w^0}_x + b$$

IF  $w = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \Rightarrow 0.5^{L-1} \Rightarrow$  VANISHING

OR  $w = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \Rightarrow 1.5^{L-1} \Rightarrow$  EXPLODING

IN BOTH CASES GRADIENT DESCENT  
TAKES A VERY LONG TIME

PARTIAL SOLUTION: CHOOSE INITIAL  
VALUES CAREFULLY

$$w^{[L]} = \text{rand} * \sqrt{\frac{2}{n^{L-1}}} \quad (\text{FOR RELU})$$

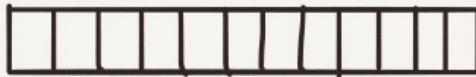
#inputs

$$\text{XAVIER } \sqrt{\frac{1}{n^{L-1}}} \quad (\text{FOR TANH})$$

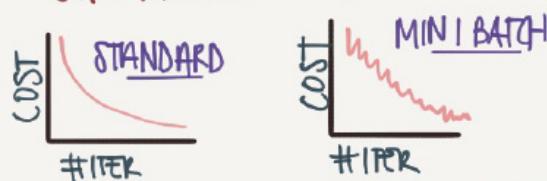
SETS THE VARIANCE

# Mini-batch gradient descent

## MINI-BATCH GRAD. DESCENT



SPLIT YOUR DATA INTO MINI-BATCHES  
& DO GRAD DESCENT AFTER EACH BATCH.  
THIS WAY YOU CAN PROGRESS AFTER  
JUST A SHORT WHILE



## CHOOSING THE MINIBATCH SIZE

SIZE =  $m$  → BATCH GRAD DESC.

SIZE = 1 → STOCHASTIC GRAD DESC



TIP

IF YOU HAVE < 2000 SAMPLES

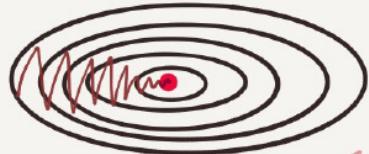
USE SIZE = 2000

OTHERWISE, USE 64, 128, 256...

SO X+y FITS IN CPU/GPU CACHE

# Optimizing training

## GRADIENT DESCENT W. MOMENTUM



SLOWER LEARNING  
FASTER LEARNING

WE WANT TO REDUCE  
OSCILLATION ↓ SO WE GET TO THE  
GOAL FASTER

SOLUTION: SMOOTH OUT THE  
CURVE BY TAKING AN EXPONENTIALLY  
WEIGHTED AVERAGE OF THE  
DERIVATIVES (i.e. LAST ONE HAS MORE)  
IMPORTANCE

## RMSProp - ROOT MEAN SQUARED



SLOWER LEARNING  
FASTER LEARNING

NORMALIZE GRADIENT USING A MOVING AVG.

$$S_{dw} = \beta S_{dw} + (1-\beta) dW^2$$

$$S_{db} = \beta S_{db} + (1-\beta) db^2$$

$$W = W - \alpha \frac{dW}{\sqrt{S_{dw}}} \quad b = b - \alpha \frac{db}{\sqrt{S_{db}}}$$

## ADAM OPTIMIZATION

COMBO OF GD W  
MOMENTUM & RMSProp

## LEARNING RATE DECAY

IDEA: USE A LARGE  $\alpha$  IN THE  
BEGINNING. THEN DECREASE AS  
WE GET CLOSER TO GOAL

OPTION 1:  $\alpha = \frac{1}{1 + \text{DECAYRATE} \cdot \text{EPOCH}} \alpha_0$

EXPONENTIAL:  $\alpha = 0.95^{\text{EPOCH}} \alpha_0$

OPTION 3:  $\alpha = \frac{k}{\sqrt{\text{EPOCH}}} \alpha_0$

OPTION 4:  $\alpha = \frac{k}{\sqrt{t}} \alpha_0$

OPTION 5:   
DISCRETE STAIRCASE  $\alpha$  EPOCHS

OPTION 6: MANUAL

EPOCH = 1 PASS THROUGH THE DATA

# Hyperparameter Tuning

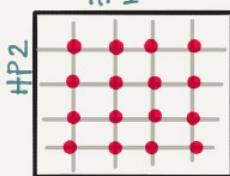
## HYPERPARAM TUNING

WHICH HYPERPARAMS ARE MOST IMPORTANT?

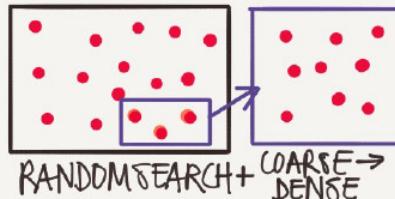
- $\alpha$  LEARNING RATE
- # HIDDEN UNITS
- MINIBATCH SIZE
- $\beta$  MOMENTUM, TURN = 0.9
- # LAYERS
- LEARNING RATE DECAY
- $\beta_1 = 0.9$   $\beta_2 = 0.999$   $\epsilon = 10^{-8}$  (ADAM)

## TESTING VALUES

CLASSIC ML



GRID SEARCH  
SOLUTION



PROBLEM: ONE ITERATION TAKES A LONG TIME & IN 16 GO'S WE HAVE ONLY TRIED 4 $\alpha$  - BUT 4 DIFF  $\epsilon$  ←  
NOT AS IMPORTANT

MY PANDA IS ACTUALLY A MISCLASSIFIED CAT BECAUSE I CAN'T DRAW PANDAS

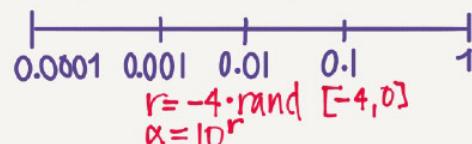
PANDA VS CAVIAR  
BABY'S IT ONE MODEL & TUNE  
SPAWN LOTS OF MODELS W DIFF HP

USE AN APPROPRIATE SCALE

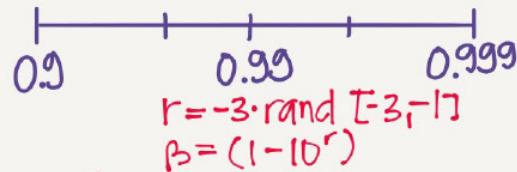
# HIDDEN UNITS



$\alpha$  LEARNING RATE



$\beta$  EXP WEIGHT AVE



TIP  
RE-EVALUATE YOUR HYP. PARAMS EVERY FEW MONTHS

## MISC. EXTRAS

### BATCH NORMALIZATION

NORMALIZE LAYER OUTPUT

- SPEEDS UP TRAINING
- MAKES WEIGHTS DEEPER IN NW MORE ROBUST (COVARIATE SHIFT)
- SLIGHT REGULARIZING EFFECT

### MULTICLASS CLASSIFIC.



C = # CLASSES = 4

### SOFTMAX ACTIVATION

$$t = e^{z^{[i]}}$$

$$a^{[i]} = \frac{t}{\sum t_i}$$

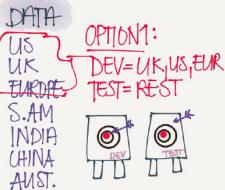
p(OTHER | x)  
p(CAT | x)  
p(FISH | x)  
p(B.C | x)

EX:  $z^{[i]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$     $t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$    SUM: 1

$$\Rightarrow o^{[i]} = \frac{t}{176.3} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.02 \\ 0.14 \end{bmatrix}$$

11.4% PROB IT'S A BABY CHICK  
© TessFerrandez

# Train / Dev / Test sets



# Importance of train/dev sets

## TRAIN vs DEV/TEST MISMATCH

### AVAILABLE DATA

200K PRO CAT PICS FROM INTERNET

10K BLURRY CAT PICS FROM APP  
**WHAT WE CARE ABT**

HOW DO WE SPLIT → TRAIN/DEV/TEST?

OPTION 1: SHUFFLE ALL

205k (TRAIN)	D	T
	2.5k	

PROBLEM: DEV/TEST IS NOW  
MOSTLY WEB IMB (**NOT REPR.**  
(OF END SCENARIO))

SOLUTION: LET DEV/TEST COME  
FROM APP. THEN SHUFFLE 5K  
OF APP PICS IN WEB FOR TRAIN

205k	25	25
WEB+APP	APP	APP

### BIAS & VARIANCE IN MISMATCHED TRAIN/DEV

HUMANS ~0%  
TRAIN 1% ↘  
DEV ERR 10%

IS THIS DIFF  
DUE TO THE MODEL  
NOT GENERALIZING  
OR IS DEV DATA  
MUCH HARDER

A: CREATE A TRAIN-DEV SET  
THAT WE DON'T TRAIN ON

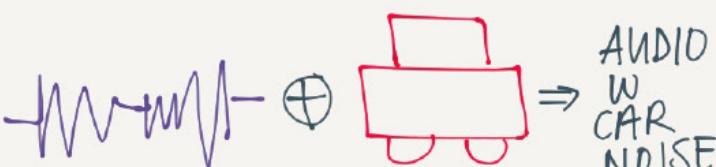
TRAIN	F	D	T

	A	B	C	D
TRAIN	1%	1%	10%	10%
TRAIN-DEV	9%	15%	11%	11%
DEV	10%	10%	12%	20%
VARIANCE			BIAS	BIAS + DATA MISMATCH
TRAIN-DEV MISMATCH				

### ADDRESSING DATA MISMATCH

EX. CIAR GPS • TRAINING DATA IS 10.000H  
OF GENERAL SPEECH DATA

1. CARRY OUT MANUAL ERROR ANALYSIS  
TO UNDERSTAND THE DIFFERENCE  
(EX NOISE, STREET NUMBERS)
2. TRY TO MAKE TRAIN MORE SIMILAR  
TO DEV OR GATHER MORE DEV-LIKE  
TRAIN-DATA



### NOTE

BE CAREFUL • IF YOU  
ONLY HAVE 1 HR OF  
CAR NOISE & APPLY IT TO 10K HR  
SPEECH YOU MAY OVERFIT TO  
THE CAR NOISE

# Metrics for performance

## SETTING YOUR GOAL

\* GOAL SHOULD BE A SINGLE #

	PRECISION	RECALL
A	95%	90%
B	98%	85%

IS A OR  
B BEST?

	PRECISION	RECALL	F1
A	95%	90%	92.4%
B	98%	85%	91%

F1 = HARMONIC MEAN BETW.  
RECALL & PRECISION

\* DEFINE OPTIMIZING VS  
SATISFYING METRICS

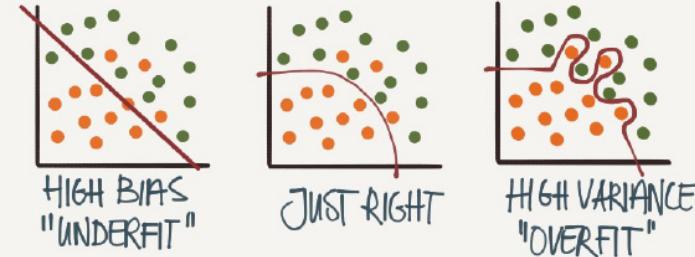
	ACCURACY	RUNTIME
A	90%	80ms
B	92%	95ms
C	95%	1500ms

MAXIMIZE ACC.  
GIVEN TIME  $\leq$  100ms

ACCURACY =  
OPTIMIZING  
RUNTIME =  
SATISFYING

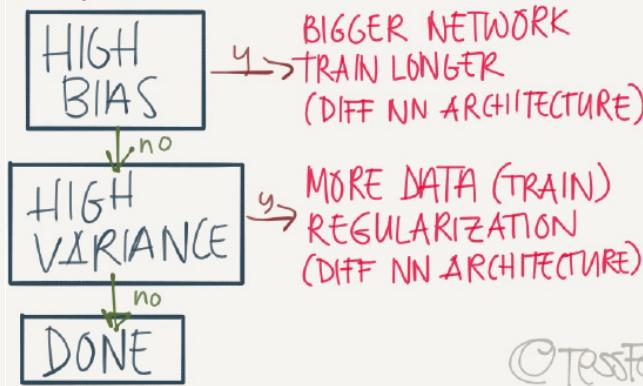
# Bias vs. Variance

## BIAS/VARIANCE



	ERROR			
TRAIN	1%	15%	15%	0.5%
TEST	11%	16%	30%	1%
HIGH VARIANCE	HIGH BIAS	HIGH BIASE & VARIANCE	LOW BIASE & VARIANCE	
ASSUMING HUMANS GET 0% ERROR				

## THE ML RECIPE



@TessFe

# Bayes Optimal Error; Suprassing Human Performance

BAYES OPTIMAL ERROR

HUMAN LEVEL PERF

MEDICAL IMB CLASS

TYPICAL HUMAN 3%

TYPICAL DOCTOR 1%

EXPERIENCED DR. 0.7%

TEAM OF EXP DRs. 0.5%

WHY DOES ACC SLOW DOWN WHEN WE SURPASS HUMAN LEVEL PERF?

HUMAN LEV PERF  
(PROXY FOR BAYES)

1. OFTEN CLOSE TO BAYES
2. A HUMAN CAN NO LONGER HELP IMPROVE (INSIGHTS)
3. DIFFICULT TO ANALYSE BIAS/VARIANCE

CAT CLASSIFICATION

	A	B
HUMAN	1%	7.5%
TRAIN ERR	8%	8%
DEV ERR	10%	10%
FOCUS ON BIAS		FOCUS ON VARIANCE

HUMAN TRAIN BIGGER NETW.

| AVOIDABLE BIAS } TRAIN LONGER / BETTER OPT. (RMSPROP, ADAM)  
| VARIANCE } CHANGE NN ARCH OR HYPERPARAMS

TRAIN | VARIANCE } MORE DATA (TRAIN)  
| AVOIDABLE BIAS } REGULARIZATION  
DEV } NN ARCHITECTURE

	A	B
HUMAN	0.5	0.5
TRAIN ERR	0.6	0.3
DEV ERR	0.8	0.4
AVOID. BIAS	0.1	?

DON'T KNOW IF WE OVERFIT  
OR IF WE'RE CLOSE TO BAYES

OPTIONS TO PROCEED ARE UNCLEAR

# Error Analysis

## ERROR

### ANALYSIS

YOU HAVE 10% ERRORS, SOME ARE DOGS MIS-CLASSIFIED AS CATS. SHOULD YOU TRAIN ON MORE DOG PICS?

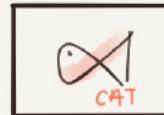
1. PICK 100 MIS-LABELED
2. COUNT ERROR REASONS

Dog	Blurry	Insta Filter	Big Cat	...
1	1		1	
2				1
3		1		
...				
100			1	
5	...			

5% OF ALL ERRORS

FOCUSING ON DOGS. THE BEST WE CAN HOPE FOR IS 9.5% ERROR

YOU FIND SOME INCORRECT LABELED DATA IN THE DEV SET. SHOULD YOU FIX IT?



DL ALGORITHMS ARE PRETTY ROBUST TO RANDOM ERRORS. BUT NOT TO SYSTEMATIC ERR. (EX. ALL WHITE CATS INCORRECTLY LABELED AS MICE)

ADD EXTRA COL. IN ERROR ANALYSIS AND USE SAME CRITERIA

**NOTE:** IF YOU FIX DEV YOU SHOULD FIX TEST AS WELL.

FOR NEW PROJ:  
BUILD 1ST SYSTEM QUICK & ITERATE

EX: SPEECH RECOGNITION



WHAT SHOULD YOU FOCUS ON?

NOISE  
ACCENTS  
FAR FROM MIKE

1. START QUICKLY DEV/TEST METRICS
2. GET TRAIN-SET
3. TRAIN
4. BIAS/VARIANCE ANAL
5. ERROR ANALYSIS
6. PRIORITIZE NEXT STEP

# Regularization

# Regularization

## REGULARIZATION PREVENTING OVERFITTING

### L2 REGULARIZATION

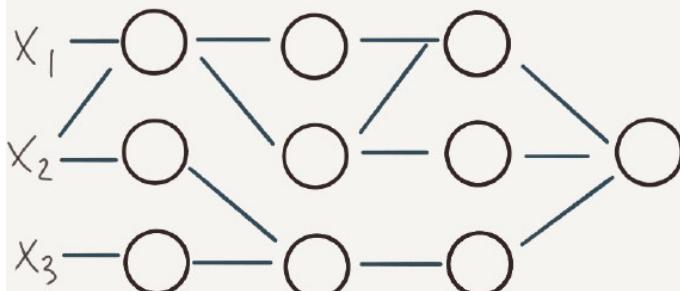
$$\text{COST: } J(w, b) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}_i, y_i) + \frac{\lambda}{2m} \|w\|_2^2$$

EUCLIDEAN NORM

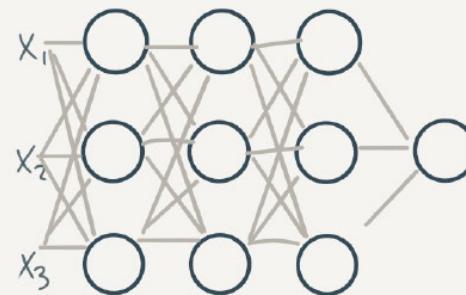
### L1 REGULARIZATION

$$\text{COST: } J(w, b) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}_i, y_i) + \frac{\lambda}{m} \|w\|_1$$

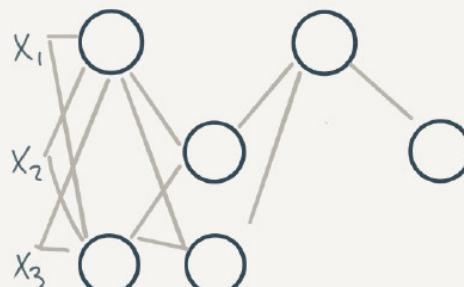
BOTH PENALIZE LARGE WEIGHTS  $\Rightarrow$   
SOME WILL BE CLOSE TO  $0 \Rightarrow$   
SIMPLER NETWORKS



## DROPOUT



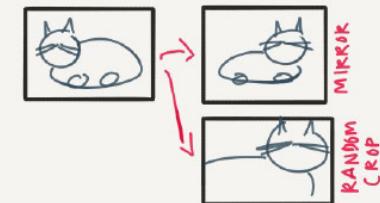
FOR EACH ITERATION  $\in$  SAMPLE  
SOME NODES ARE RANDOMLY  
DROPPED (BASED ON KEEP-PROB)



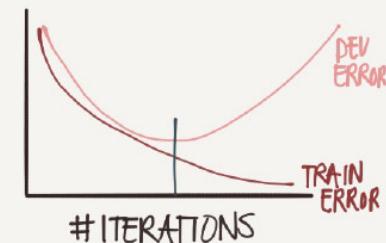
WE GET SIMPLER NWs  
 $\in$  LESS CHANCE TO RELY ON  
SINGLE FEATURES

## OTHER REGULARIZATION TECHNIQUES

DATA AUGMENTATION  
GENERATE NEW PICS FROM EXISTING



## EARLY STOPPING



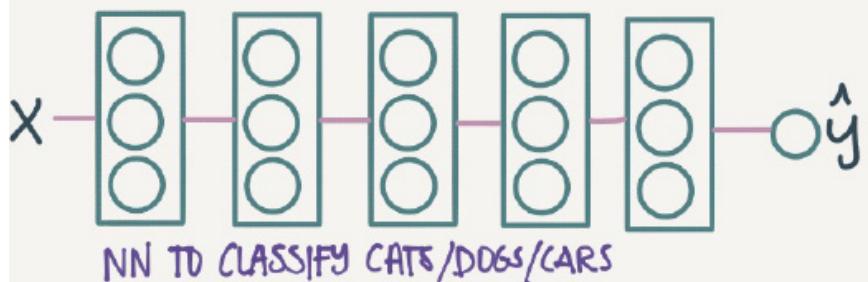
PROBLEM: AFFECTS BOTH  
BIAS & VARIANCE

# Extended Learning

# Transfer learning

## TRANSFER LEARNING

PROBLEM: YOU WANT TO  
CLASSIFY SOME MEDICAL IMB.  
YOU HAVE AN NN THAT  
CLASSIFIES CATS



**OPTION 1:** YOU ONLY HAVE A FEW  
RADIOLOGY IMAGES

SOLUTION: INIT W. WEIGHTS FROM CAT NN  
ONLY RETRAIN LAST LAYER(S) ON RADIOLOGY  
IMAGES

**OPTION 2:** YOU HAVE LOTS OF RADIOLOGY IMB

SOLUTION: INIT WITH WEIGHTS FROM CAT NN  
RETRAIN ALL LAYERS

# Multi-task learning

## MULTI TASK LEARNING

TRAINING ON MULT. TASKS AT ONCE

DETECT  
CAR  
STOP SIGN  
PEDESTR.  
TRAFFIC LIGHT



UNLIKE SOFTMAX - MANY THINGS CAN BE TRUE

$$\text{COST: } J(w, b) = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^4 p(x_i^{(j)}, y_j^{(i)})$$

SUMMING OVER ALL  
OUTPUT OPTIONS

WE COULD HAVE JUST TRAINED 4 NN'S INSTEAD BUT.. MT LEARNING MAKES SENSE WHEN

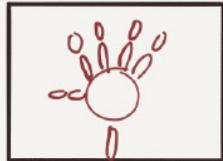
- A. THE LEARNING DATA YOU HAVE FOR THE DIFF TASKS IS QUITE SIMILAR - & THE AMOUNTS (E.G. 1K CARS, 1K STOP SIGNS)
- B. THE SUM OF THE DATA ALLOWS YOU TO TRAIN A BIG ENOUGH NN TO DO WELL ON ALL TASKS

IN REALITY TRANSFER LEARNING IS USED MORE OFTEN

# End-to-End Learning

## END-TO-END LEARNING

FROM X-RAY OF CHILDS HAND  
TELL ME THE AGE OF THE CHILD



TYPICAL SURN:

1. LOCATE BONES TO FIND LENGTHS  
USING ML
2. TRAIN MODEL TO PREDICT  
AGE BASED ON BONE LENGTH

## END - TO - END

RADIOLOGY  $\longrightarrow$  CHILD  
IMG AGE

PROS:

- LET'S THE DATA SPEAK  
(MAYBE IT FINDS RELATIONS WE'RE UNAWARE OF)

- LESS HAND-DESIGNING OF  
COMPONENTS NEEDED

CONS:

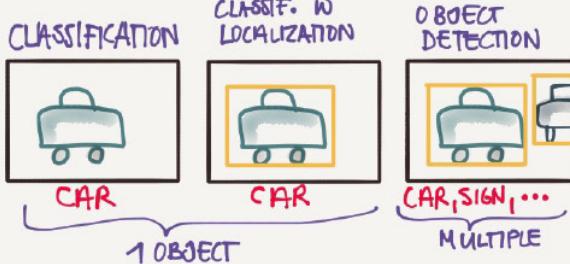
- NEEDS LARGE AMT OF DATA ( $X \rightarrow Y$ )  
Labeled
- EXCLUDES POTENTIALLY USEFUL  
HAND-MADE COMPONENTS

© Tess Ferrandez

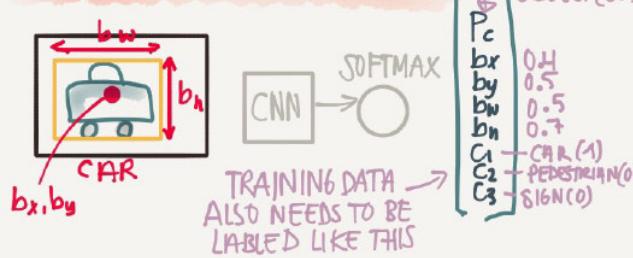
# CNN applications

# Detection, localization, landmarks

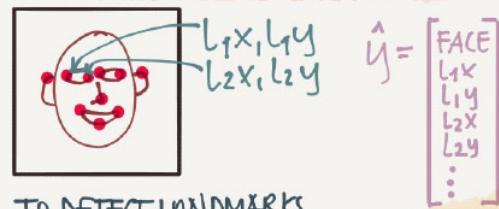
## DETECTION ALGORITHMS



## OBJECT LOCALIZATION



## LANDMARK DETECTION



TO DETECT LANDMARKS IN THE FACE (CORNER OF MOUTH ETZ) LABEL THE X, Y COORDS OF THE LANDMARK

USED FOR SENTIMENT ANALYSIS & FOR EFFECTS LIKE PLACING CROWN ON HEAD ETZ.

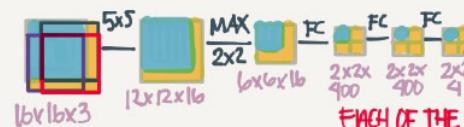
## SLIDING WINDOWS DETECTION



1. CREATE TIGHTLY CROPPED IMG OF CARS (LOTS)
2. SLIDE A WINDOW OVER THE IMG. & CLASSIFY THIS WINDOW CAR(1/0) AGAINST YOUR OTHER CARS
3. REPEAT WITH SLIGHTLY LARGER WINDOW SIZE

PROBLEM: VERY EXPENSIVE (TO COMPUTE)

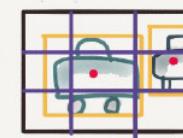
SINCE ADT WINDOWS SHARE A LOT OF THE COMPUTATIONS WE CAN DO THIS MUCH CHEAPER W CONVOLUTIONS



NOW WE JUST PASS THROUGH ONCE AND CALC ALL AT THE SAME TIME

EACH OF THE 4 VALS ARE RESULTS FOR EACH OF THE 4 WINDOWS

## YOLO - You Only Look Once

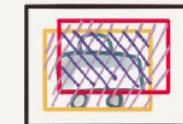


IN PRACTICE WE MIGHT HAVE A  $19 \times 19$  GRID

1. SPLIT IMG INTO  $X(g)$  GRID CELLS
  2. FOR EACH CELL, SAY IF IT CONTAINS CAR + BOUNDING BOX (IF CELL CONTAINS THE MID POINT)
- $$X \cdot g = 3 \times 3 \times 8$$

HOW DO YOU KNOW HOW GOOD IT IS?

HOW GOOD IS THE RED SQUARE?



$$IOU = \frac{\text{SIZE OF INTERSECTION}}{\text{SIZE OF UNION}}$$

## INTERSECTION OVER UNION

GENERALLY IF  $IOU \geq 0.5$  IT IS REGARDED AS CORRECT

WHAT IF MULTIPLE SQUARES CLAIM THE SAME CAR?

## NON-MAX SUPPRESSION

IF TWO BOUNDING BOXES HAVE A HIGH IOU - PICK THE ONE W HIGHEST  $P_c$  - GET RID OF THE REST.

## ANCHOR BOXES

ANCHOR BOXES LET YOU ENCODE MULTIPLE OBJECTS IN THE SAME SQUARE

# Face Recognition

## FACE RECOGNITION

FACE  
VERIFICATION



IS THIS PETE?  
99% ACC  $\Rightarrow$   
PRETTY GOOD

FACE  
RECOGNITION



WHO IS THIS?  
(OUT OF K PERSONS)  
IF K = 100 NEED  
MUCH HIGHER THAN  
99%

## ONE-SHOT LEARNING

NEED TO BE ABLE TO RECOGNIZE  
A PERSON EVEN THOUGH YOU ONLY  
HAVE ONE SAMPLE IN YOUR DB.

YOU CAN'T TRAIN A CNN WITH  
A SOFTMAX (EACH PERSON) BECAUSE  
A) YOU DON'T HAVE ENOUGH SAMPLES  
B) IF A NEW PERSON JOINS YOU  
NEED TO RETRAIN THE NETWORK

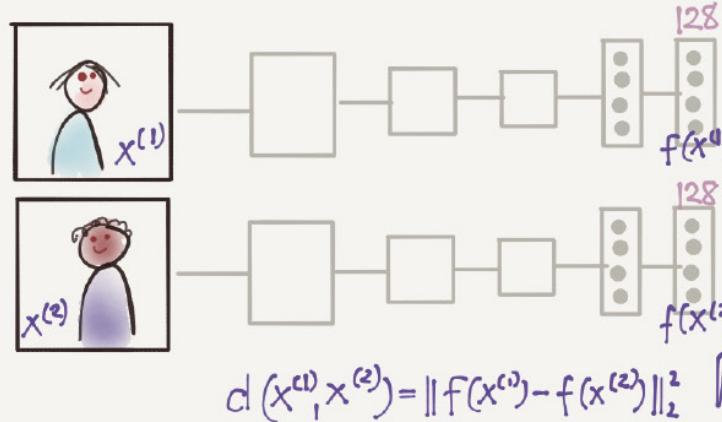
**SOLUTION** LEARN A SIMILARITY  
FUNCTION

$$d(\text{img}1, \text{img}2) = \text{degree of difference}$$

BUT HOW DO YOU LEARN THIS?

## SIAMESE NETWORK

## DeepFace

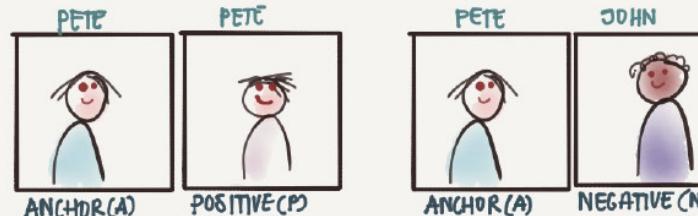


LEARN THE PARAMS OF  
THE NN SUCH THAT  
- IF  $x^{(1)}, x^{(2)}$  ARE THE SAME  
PERSON  $\cdot d(x^{(1)}, x^{(2)}) \Rightarrow$  SMALL  
- IF  $x^{(1)}, x^{(2)}$  ARE DIFFERENT  
PEOPLE  $\cdot d(x^{(1)}, x^{(2)}) \Rightarrow$  LARGE

WE CAN ACCOMPLISH  
THIS WITH THE TRIPLET  
LOSS FUNCTION

## TRIPLET LOSS

## FaceNet



$$\text{WANT } \|f(A) - f(P)\|^2 \leq \|f(A) - f(N)\|^2 \Rightarrow d(A, P) - d(A, N) \leq 0$$

$d(A, P) - d(A, N) + \alpha \leq 0$

HOW DO WE CHOOSE TRIPLETS  
TO TRAIN ON?

- IF A/P ARE VERY SIMILAR, & A/N ARE VERY DIFFERENT  
TRAINING IS VERY EASY.

SELECT A/N THAT ARE PRETTY SIMILAR TO TRAIN A GOOD NET

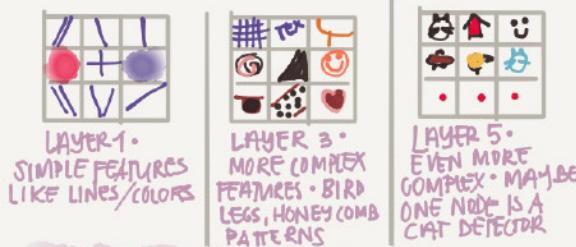
SOME BIG COMPANIES  
HAVE ALREADY TRAINED  
NETWORKS ON LARGE  
AMTS OF PHOTOS SO  
YOU MAY JUST  
WANT TO REUSE  
THEIR WEIGHTS

# Style transfer

## NEURAL STYLE TRANSFER



WE CAN VISUALIZE WHAT A NETWORK LEARNS BY LOOKING AT WHAT IMAGES (PARTS) ACTIVATED EACH UNIT MOST



BUT HOW DOES THIS HELP US GENERATE AN IMAGE IN THE STYLE OF ANOTHER?

IDEA:

1. GENERATE A RANDOM IM $G$
2. OPTIMIZE THE COST FUNCTION

$$J(G) = \alpha J_{\text{CONTENT}}(C, G) + \beta J_{\text{STYLE}}(S, G)$$

HOW SIMILAR ARE  $C \& G$       HOW SIMILAR ARE  $S \& G$

3. UPDATE EACH PIXEL

## CONTENT COST FUNCTION

- USE A PRE-TRAINED CONVNET (ex VGG)
- SELECT A HIDDEN LAYER SOMEWHERE IN THE MIDDLE  
LATER  $\Rightarrow$  COPIES LARGER FEATURES
- LET  $a^{(l)(c)}$  &  $a^{(l)(k)}$  BE THE ACTIVATIONS
- IF  $a^{(l)(c)} \approx a^{(l)(k)}$  ARE SIMILAR THEY HAVE SIMILAR CONTENT  
*BECAUSE THEY BOTH TRIGGER THE SAME HIDDEN UNITS*

HOW DO WE TELL IF THEY ARE SIMILAR?

$$J_{\text{CONTENT}}(C, G) = \frac{1}{2} \| a^{(l)(c)} - a^{(l)(k)} \|_F^2$$

## CAPTURING THE STYLE



USING THE STYLE IM $G$  AND THE ACTIVATIONS IN A LAYER. LOOK THROUGH THE ACTIVATIONS IN THE DIFFERENT CHANNELS TO SEE HOW CORRELATED THEY ARE

WHEN WE SEE PATTERNS LIKE THIS



DO WE USUALLY SEE IT WITH PATCHES LIKE THESE?



## STYLE MATRIX

CREATE A MATRIX OF HOW CORRELATED THE ACTIVATIONS ARE, FOR EACH POS  $(x, y)$ ,  $\epsilon$  CHANNEL PAIR  $(k, k')$  FOR THE STYLE IM $G$  & GENERATED

$$G_{kk'} = \sum_{i=1}^{n_h} \sum_{j=1}^{n_w} a_{ijk} \cdot a_{ijk'}$$

## THE STYLE COST FUNCTION

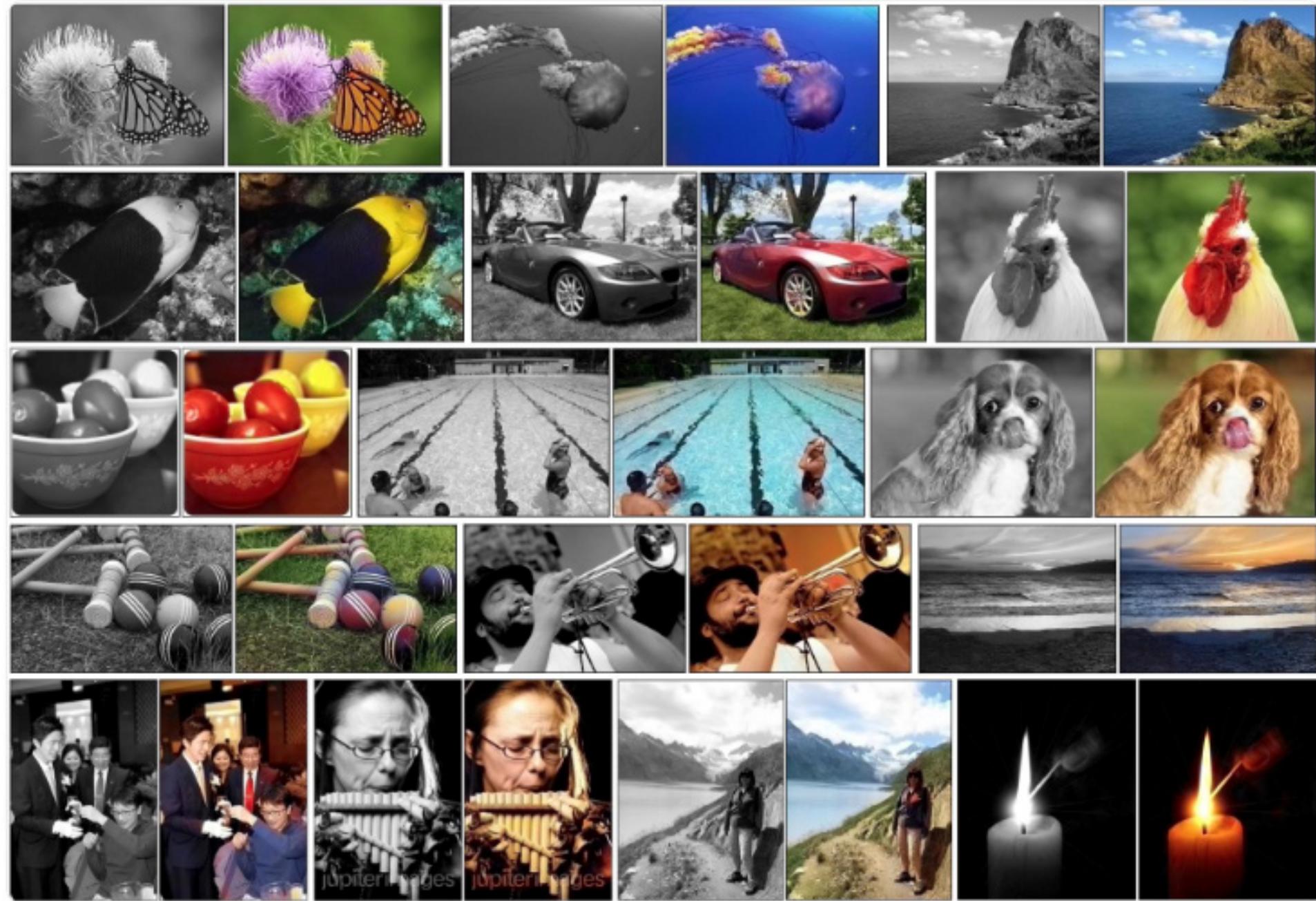
$$J(S, G) = \| G^{(s)} - G^{(g)} \|_F^2$$

FROBENIUS NORM

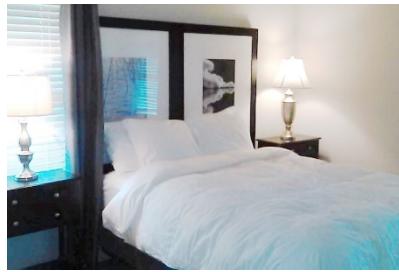
TO GET MORE VISUALLY PLEASING IMAGES IF YOU CALC  $J(S, G)$  OVER MULTIPLE LAYERS



# Automatic Colorization of Black and White Images



# Optimizing Images



Post Processing Feature Optimization  
(Color Curves and Details)

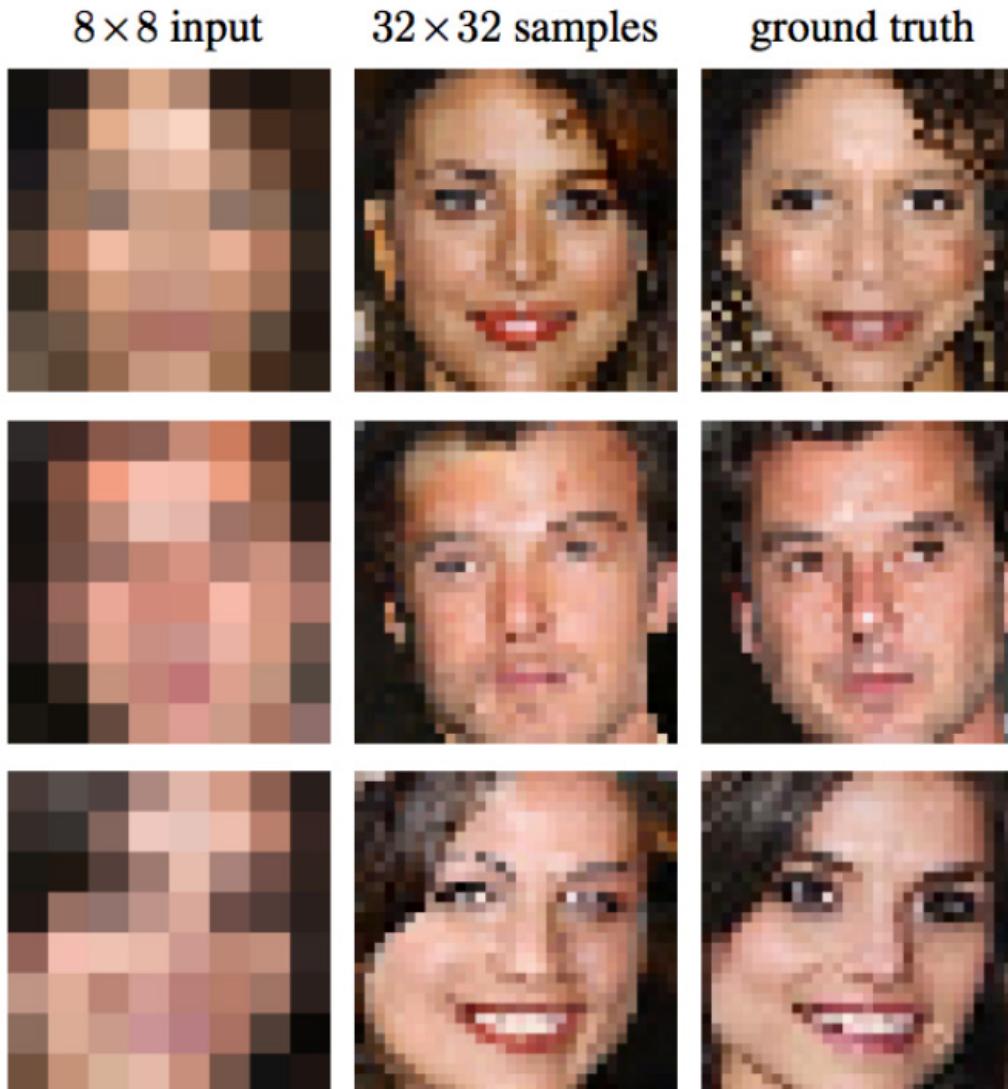


Post Processing Feature Optimization  
(Illumination)



Post Processing Feature Optimization  
(Color Tone: Warmness)

# Up-scaling low-resolution images



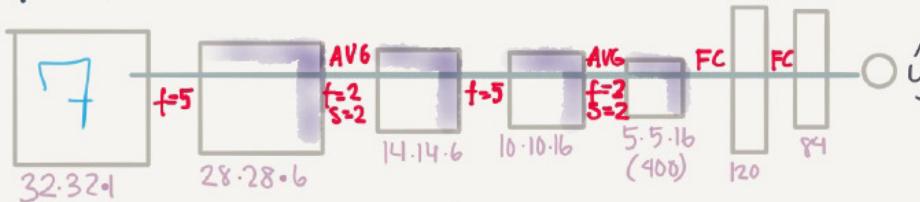
8x8 pixel photos were inputted into a Deep Learning network which tried to guess what the original face looked like. As you can see it was fairly close (the correct answer is under "ground truth").

Next-generation models  
explode # of parameters

# CLASSIC CONV. NETS

## LeNet-5

DOCUMENT CLASSIFICATION



TRENDS: HEIGHT/WIDTH GO DOWN  
CHANNELS GO UP

COMMON PATTERN: A COUPLE OF CONV(1<sup>t</sup>)/POOL LAYERS FOLLOWED BY A FEW FC

OLD STUFF: USED AVG POOLING INSTEAD OF MAX  
PADDING WAS NOT VERY COMMON  
IT USED SIGMOID/TANH INSTEAD OF RELU

## AlexNet

IMAGE CLASSIFICATION

$\approx 60 \text{ M PARAMETERS}$



- SIMILAR TO LeNet BUT MUCH BIGGER
- USES RELU
- THE NN THAT GOT RESEARCHERS INTERESTED IN VISION AGAIN

## VGG-16

ALL CONV. LAYERS HAVE SAME PARAMS  
 $f=3x3, s=1, p=\text{SAME}$   
AND POOLING LAYER  $2x2, s=2$



$\approx 138 \text{ M PARAMETERS}$

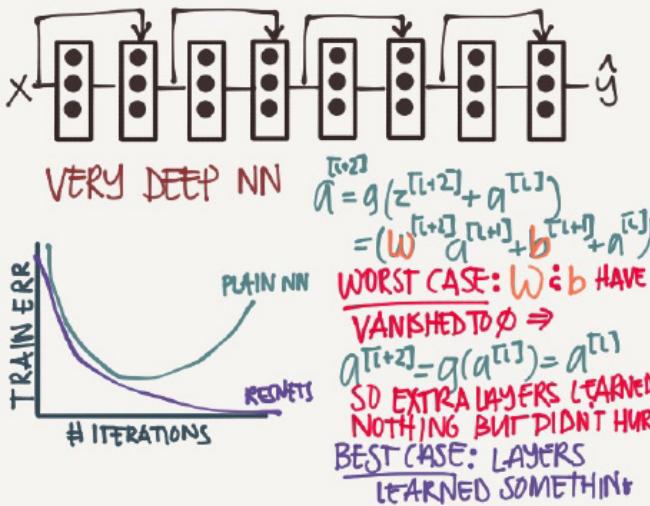
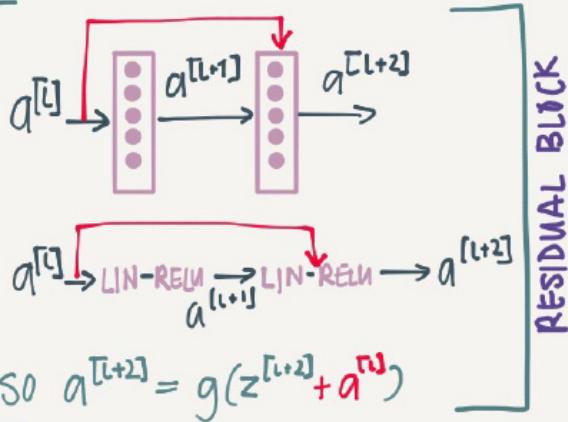
- VERY DEEP
- EASY ARCHITECTURE
- # FILTERS DOUBLE 64, 128, 256, 512

# Residual Networks

## ResNets

PROBLEM: DEEP NN OFTEN SUFFER PROBLEMS IN VANISHING OR EXPLODING GRADIENTS

### SOLUTION) RESIDUAL NETS



# Network-in-Network: $1 \times 1$ convolution

## NETWORK IN NETWORK ( $1 \times 1$ CONVOLUTION)

$$\begin{matrix} 6 & 5 & 3 & 2 \\ 4 & 1 & 9 & 5 \\ 5 & 8 & 2 & 4 \\ 0 & 3 & 6 & 1 \end{matrix} \star \boxed{2} = \begin{matrix} 12 & 10 & 6 & 4 \\ 8 & 2 & 18 & 10 \\ 10 & 16 & 4 & 8 \\ 0 & 6 & 12 & 2 \end{matrix}$$

$1 \times 1$  CONVOLUTION

IT SEEMS PRETTY USELESS, BUT IT ACTUALLY SERVES 2 PURPOSES

### 1. NETWORK IN A NETWORK

A diagram showing a dark blue 3D volume tensor labeled  $6 \times 6 \times 32$ . It is multiplied by a red  $1 \times 1 \times 32$  filter, resulting in a gray  $6 \times 6 \times \# \text{FILTERS}$  grid.

LEARN COMPLEX, NON-LINEAR RELATIONSHIPS ABOUT A SLICE OF A VOLUME

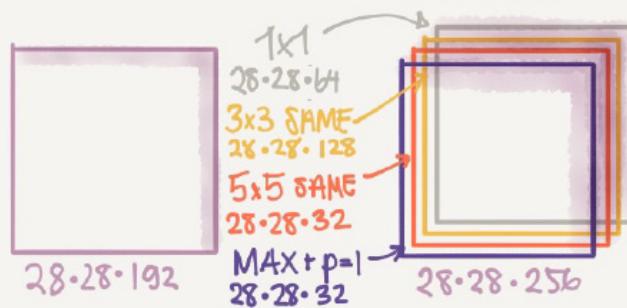
### 2. REDUCING # CHANNELS

A diagram showing a white  $28 \times 28 \times 192$  volume tensor multiplied by a purple  $1 \times 1 \times 92$  filter, resulting in a white  $28 \times 28 \times 32$  volume tensor.

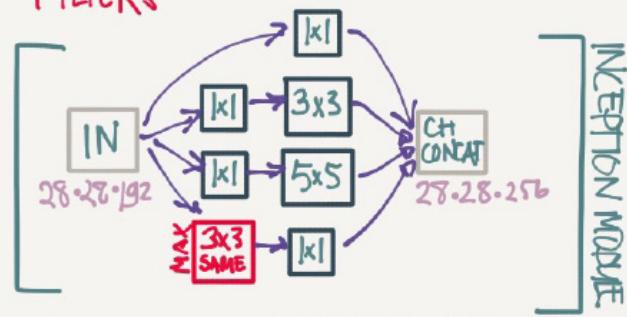
# Inception networks

## INCEPTION NETWORKS

INSTEAD OF CHOOSING A  $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$   
OR A POOLING LAYER - CHOOSE ALL

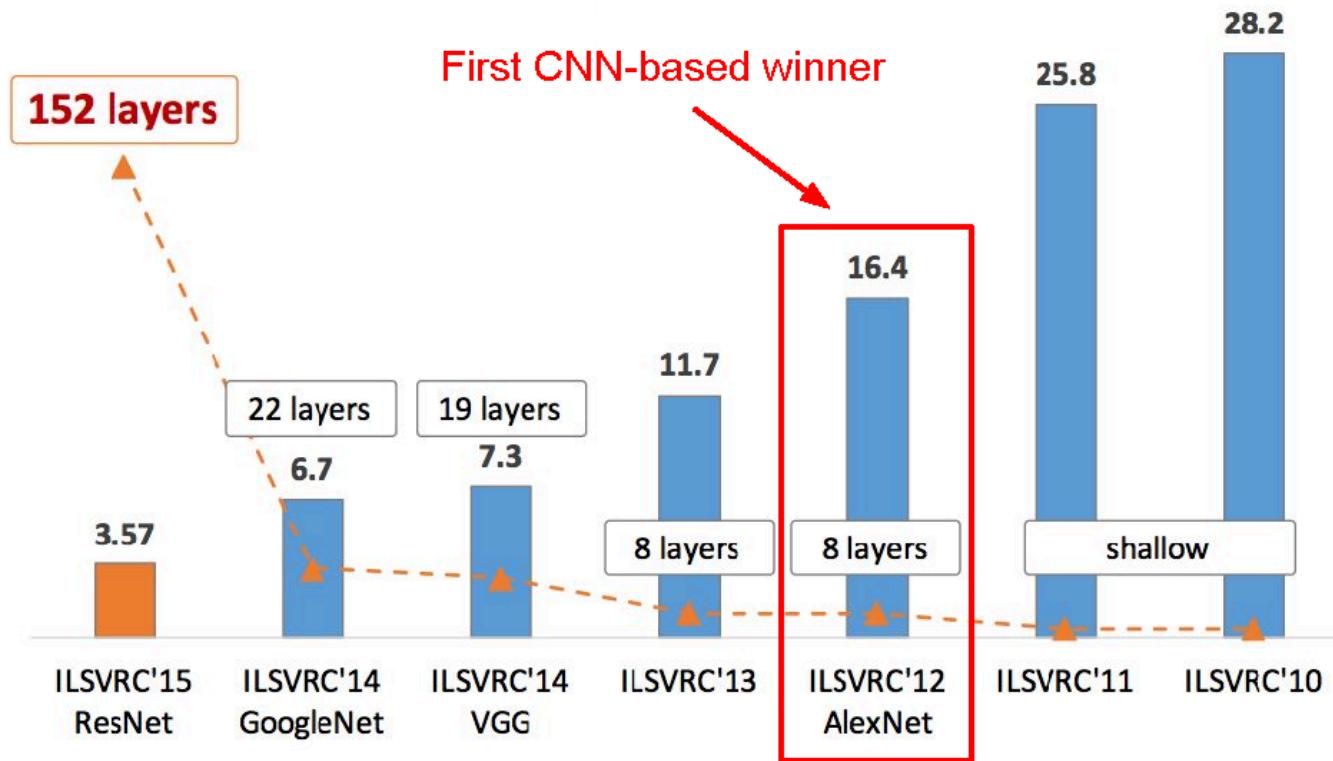


PROBLEM: VERY EXPENSIVE TO COMPUTE  
SOLUTION: SHRINK THE # CHANNELS IN  
A  $1 \times 1$  CONV BEFORE APPLYING ALL THE  
FILTERS



TO BUILD AN INCEPTION NETWORK  
YOU MAINLY STACK A BUNCH OF  
INCEPTION MODULES

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



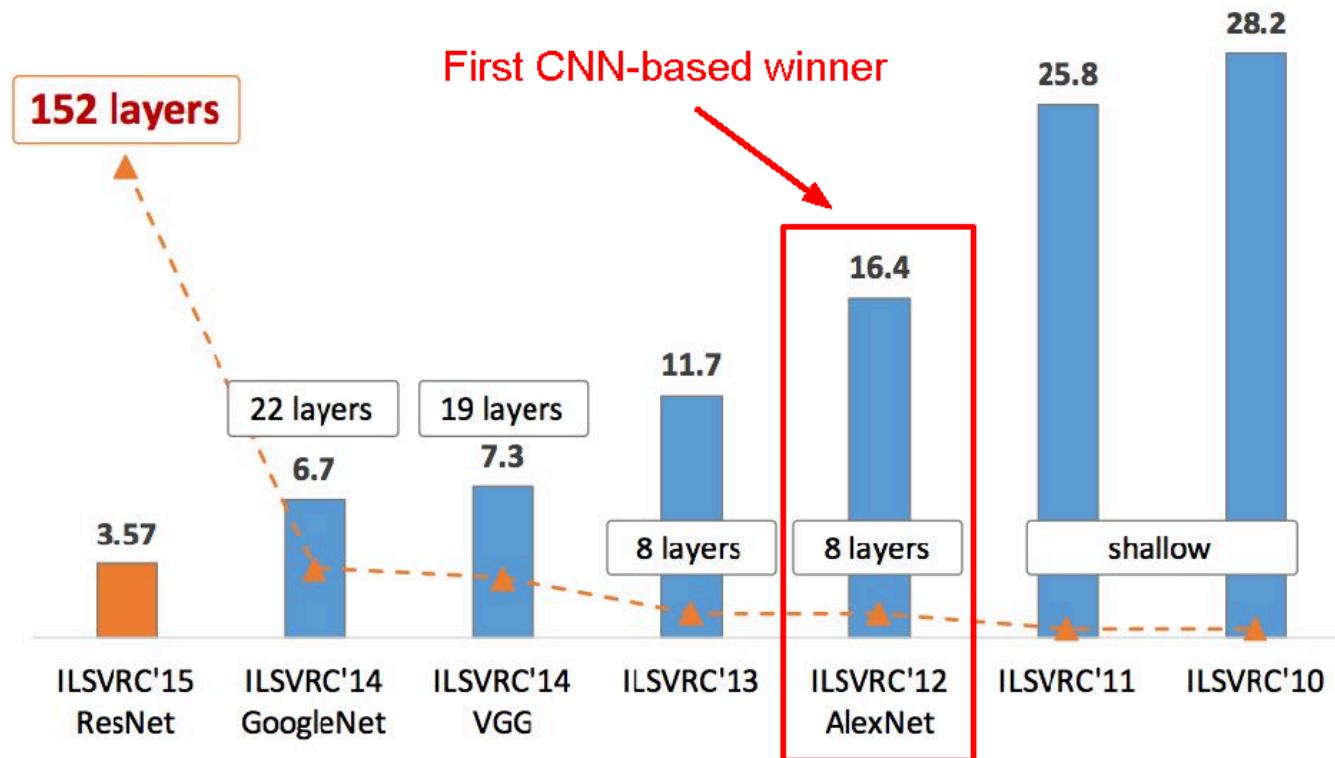
# AlexNet

- *ImageNet Classification with Deep Convolutional Neural Networks - Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton; 2012*
- Facilitated by GPUs, highly optimized convolution implementation and large datasets (ImageNet)
- One of the largest CNNs to date
- Has 60 Million parameter compared to 60k parameter of LeNet-5

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- The annual “Olympics” of computer vision.
- Teams from across the world compete to see who has the best computer vision model for tasks such as classification, localization, detection, and more.
- **2012** marked **the first year where a CNN was used** to achieve a top 5 test error rate of 15.3%.
- The next best entry achieved an error of 26.2%.

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



## Architecture

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

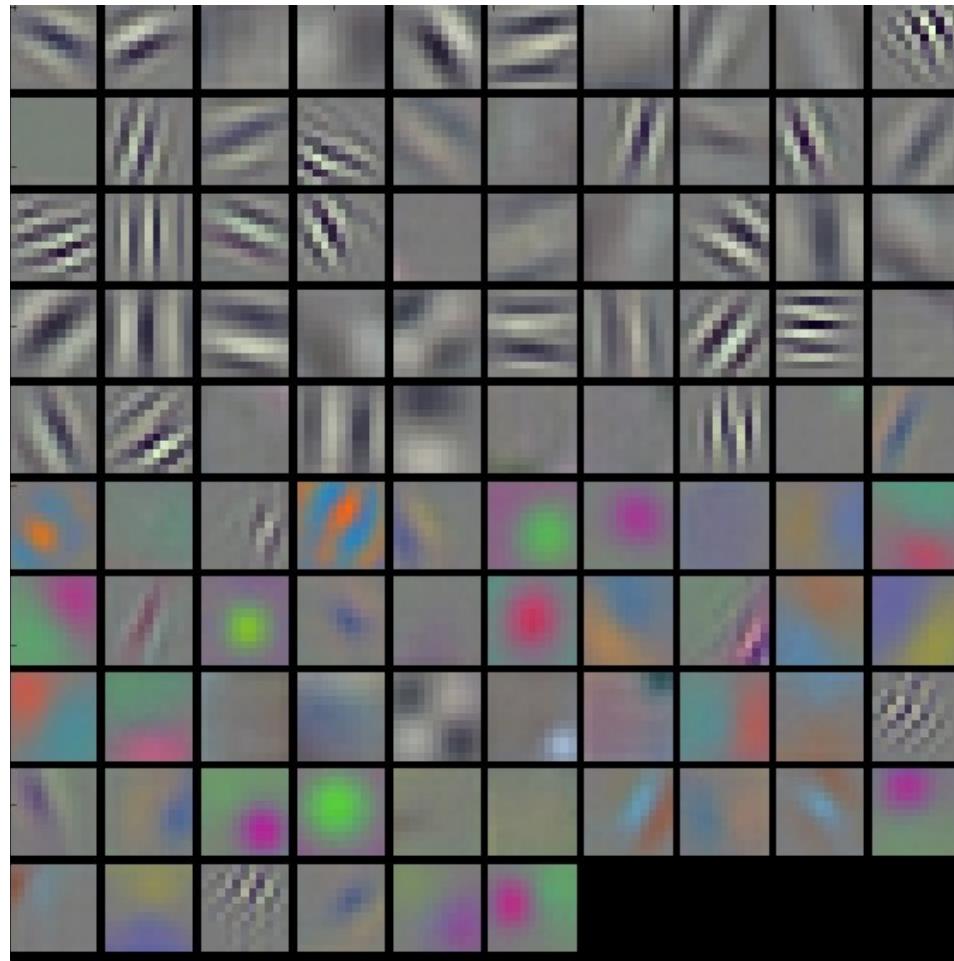
FC7

FC8

# AlexNet

- Input: 227x227x3 images (224x224 before padding)
- First layer: 96 11x11 filters applied at stride 4
- **Output volume size?**  
$$(N-F)/s+1 = (227-11)/4+1 = 55 \rightarrow [55 \times 55 \times 96]$$
- **Number of parameters in this layer?**  
$$(11 \times 11 \times 3) \times 96 = 35K$$

# AlexNet



[Krizhevsky et al., 2012]

## Architecture

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

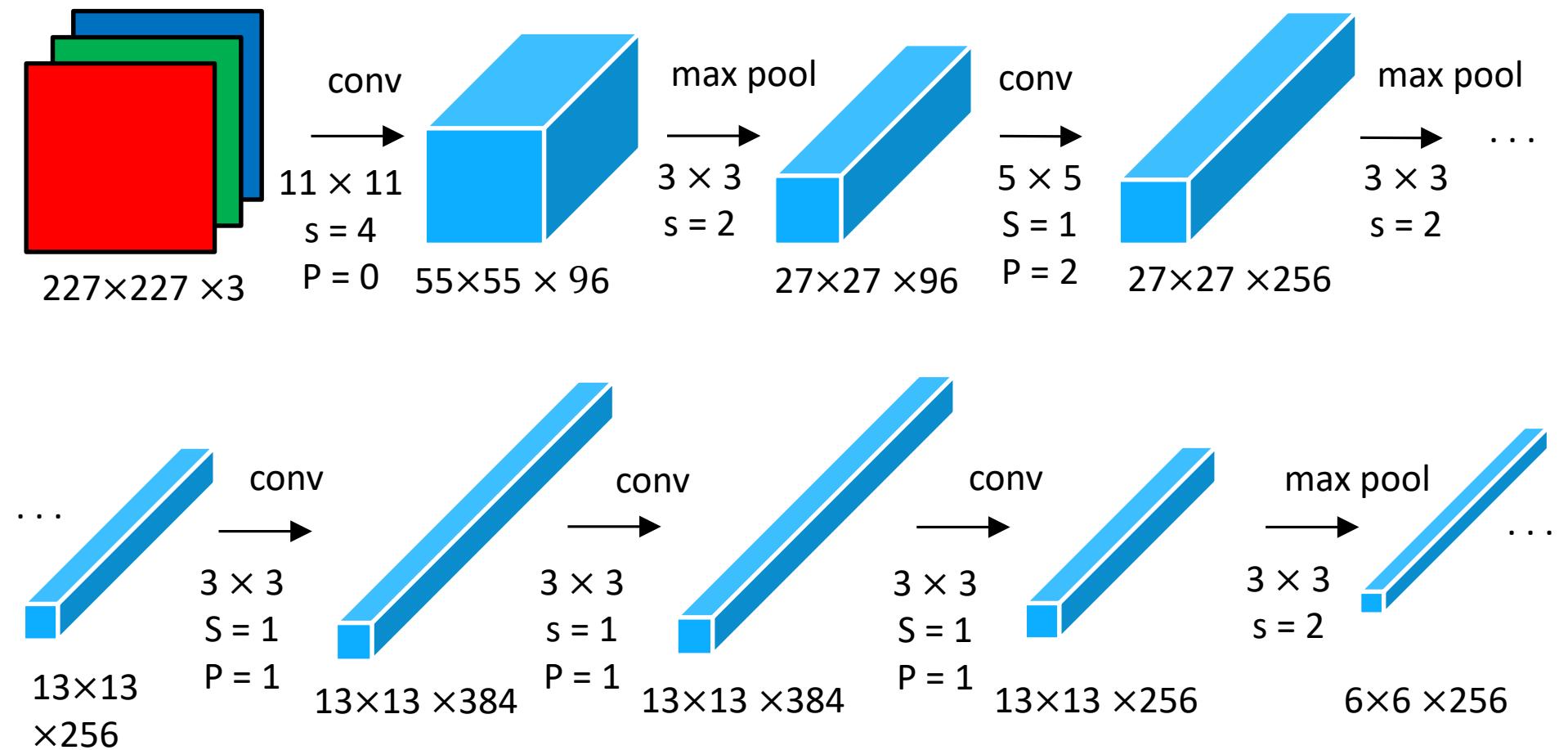
FC7

FC8

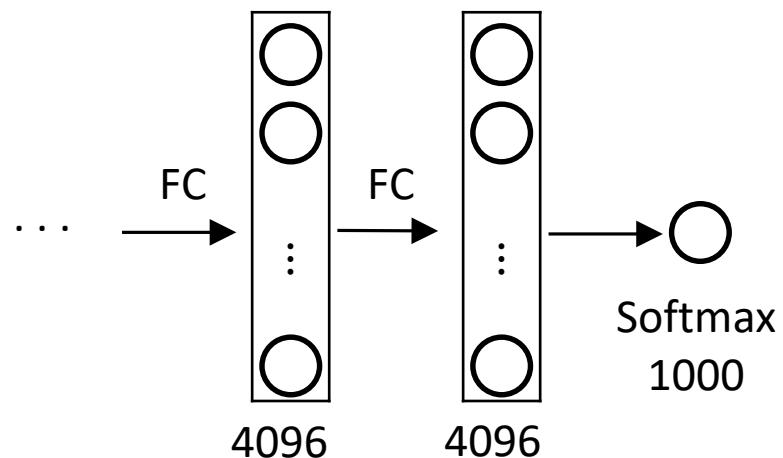
# AlexNet

- Input: 227x227x3 images (224x224 before padding)
- After CONV1: 55x55x96
- Second layer: 3x3 filters applied at stride 2
- **Output volume size?**  
$$(N-F)/s+1 = (55-3)/2+1 = 27 \rightarrow [27x27x96]$$
- **Number of parameters in this layer?**  
0!

# AlexNet



# AlexNet



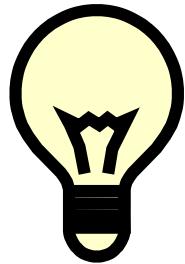
# AlexNet

## Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble

# AlexNet

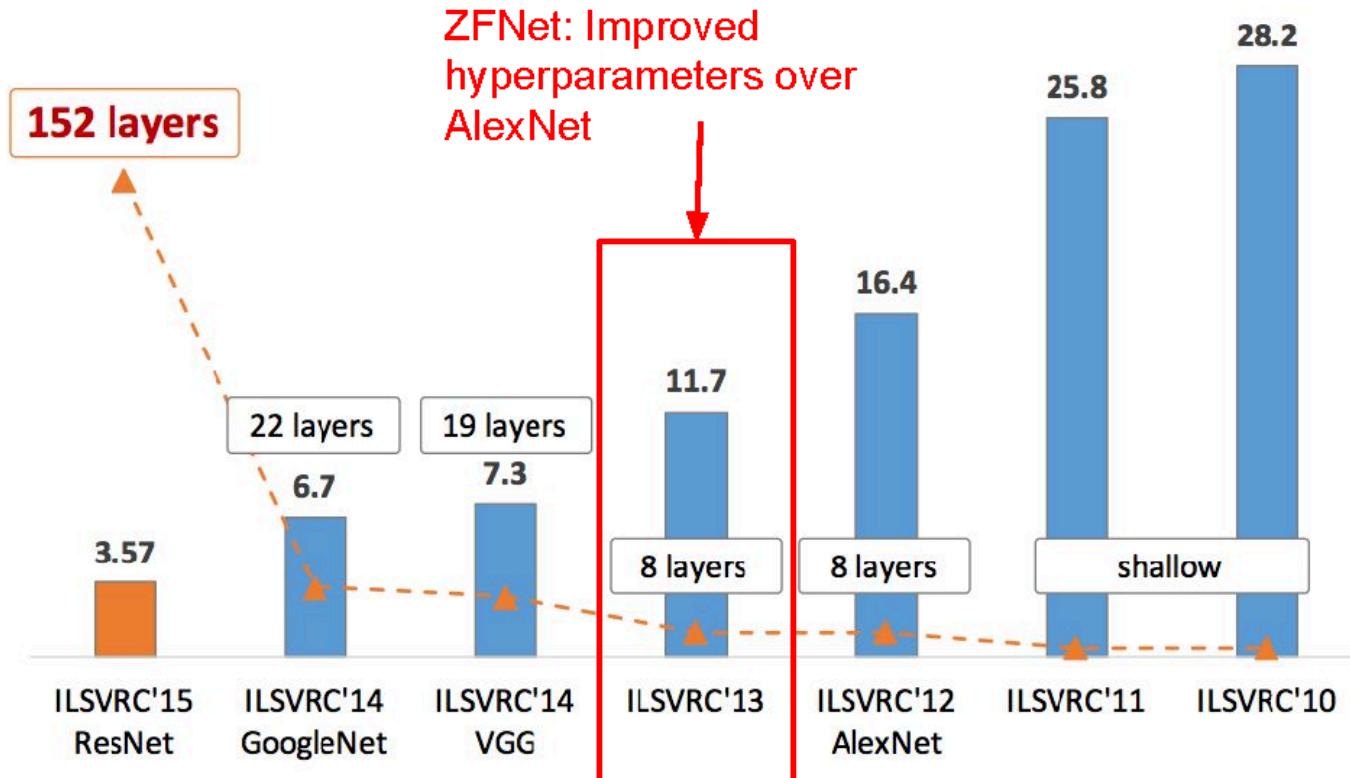
- Trained on GTX 580 GPU with only 3 GB of memory.
- Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.
- CONV1, CONV2, CONV4, CONV5:  
Connections only with feature maps on same GPU.
- CONV3, FC6, FC7, FC8:  
Connections with all feature maps in preceding layer,  
communication across GPUs.



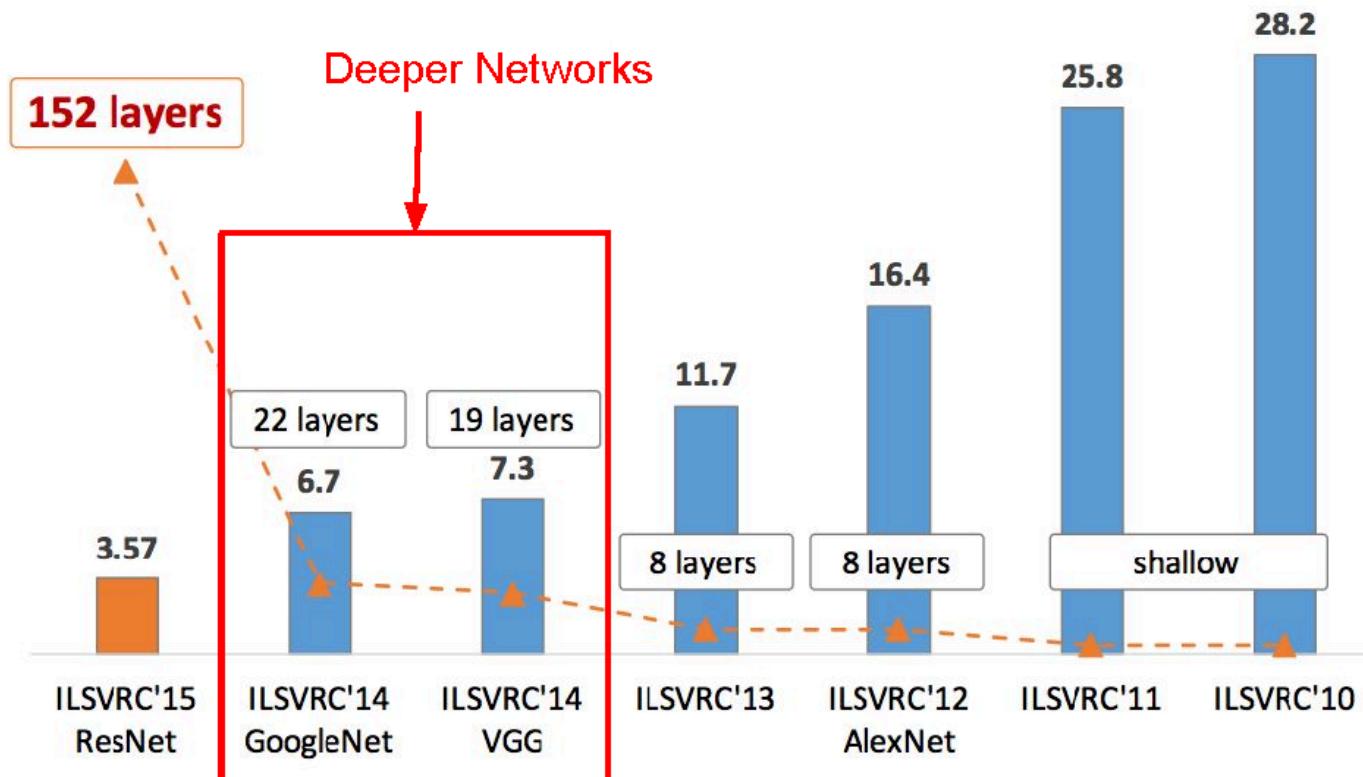
# AlexNet

AlexNet was the coming out party for CNNs in the computer vision community. This was **the first time a model performed so well on a historically difficult ImageNet dataset**. This paper illustrated the benefits of CNNs and backed them up with record breaking performance in the competition.

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# VGGNet

- *Very Deep Convolutional Networks For Large Scale Image Recognition - Karen Simonyan and Andrew Zisserman; 2015*
- The runner-up at the ILSVRC 2014 competition
- Significantly deeper than AlexNet
- 140 million parameters

Input

3x3 conv, 64

3x3 conv, 64

Pool 1/2

3x3 conv, 128

3x3 conv, 128

Pool 1/2

3x3 conv, 256

3x3 conv, 256

Pool 1/2

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool 1/2

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool 1/2

FC 4096

FC 4096

FC 1000

Softmax

# VGGNet

- **Smaller filters**  
Only 3x3 CONV filters, stride 1, pad 1 and 2x2 MAX POOL , stride 2
- **Deeper network**  
AlexNet: 8 layers  
VGGNet: 16 - 19 layers
- ZFNet: 11.7% top 5 error in ILSVRC'13
- VGGNet: 7.3% top 5 error in ILSVRC'14

# VGGNet

- **Why use smaller filters? (3x3 conv)**

Stack of three 3x3 conv (stride 1) layers has the same effective receptive field as one 7x7 conv layer.

- **What is the effective receptive field of three 3x3 conv (stride 1) layers?**

7x7

But deeper, more non-linearities

And fewer parameters:  $3 * (3^2 C^2)$  vs.  $7^2 C^2$  for  $C$  channels per layer

Input

3x3 conv, 64

3x3 conv, 64

Pool

3x3 conv, 128

3x3 conv, 128

Pool

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

FC 4096

FC 4096

FC 1000

Softmax

# VGGNet

## VGG16:

TOTAL memory:  $24M * 4 \text{ bytes} \approx 96\text{MB} / \text{image}$

TOTAL params: 138M parameters

Input	memory: 224*224*3=150K	params: 0
3x3 conv, 64	memory: 224*224*64=3.2M	params: $(3*3*3)*64 = 1,728$
3x3 conv, 64	memory: 224*224*64=3.2M	params: $(3*3*64)*64 = 36,864$
Pool	memory: 112*112*64=800K	params: 0
3x3 conv, 128	memory: 112*112*128=1.6M	params: $(3*3*64)*128 = 73,728$
3x3 conv, 128	memory: 112*112*128=1.6M	params: $(3*3*128)*128 = 147,456$
Pool	memory: 56*56*128=400K	params: 0
3x3 conv, 256	memory: 56*56*256=800K	params: $(3*3*128)*256 = 294,912$
3x3 conv, 256	memory: 56*56*256=800K	params: $(3*3*256)*256 = 589,824$
3x3 conv, 256	memory: 56*56*256=800K	params: $(3*3*256)*256 = 589,824$
Pool	memory: 28*28*256=200K	params: 0
3x3 conv, 512	memory: 28*28*512=400K	params: $(3*3*256)*512 = 1,179,648$
3x3 conv, 512	memory: 28*28*512=400K	params: $(3*3*512)*512 = 2,359,296$
3x3 conv, 512	memory: 28*28*512=400K	params: $(3*3*512)*512 = 2,359,296$
Pool	memory: 14*14*512=100K	params: 0
3x3 conv, 512	memory: 14*14*512=100K	params: $(3*3*512)*512 = 2,359,296$
3x3 conv, 512	memory: 14*14*512=100K	params: $(3*3*512)*512 = 2,359,296$
3x3 conv, 512	memory: 14*14*512=100K	params: $(3*3*512)*512 = 2,359,296$
Pool	memory: 7*7*512=25K	params: 0
FC 4096	memory: 4096	params: $7*7*512*4096 = 102,760,448$
FC 4096	memory: 4096	params: $4096*4096 = 16,777,216$
FC 1000	memory: 1000	params: $4096*1000 = 4,096,000$

# VGGNet

## Details/Retrospectives :

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as AlexNet
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks
- Trained on 4 Nvidia Titan Black GPUs for **two to three weeks.**



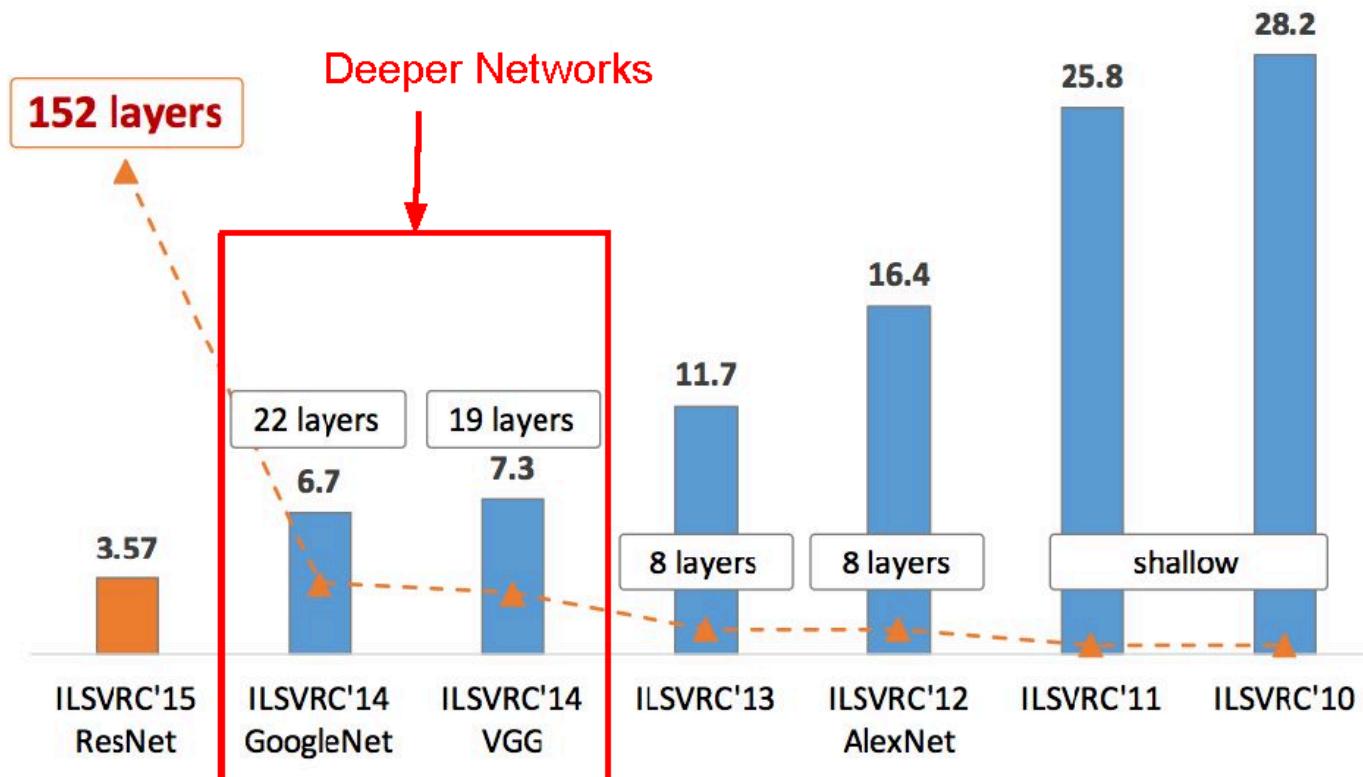
# VGGNet

VGG Net reinforced the notion that **convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work.**

Keep it deep.

Keep it simple.

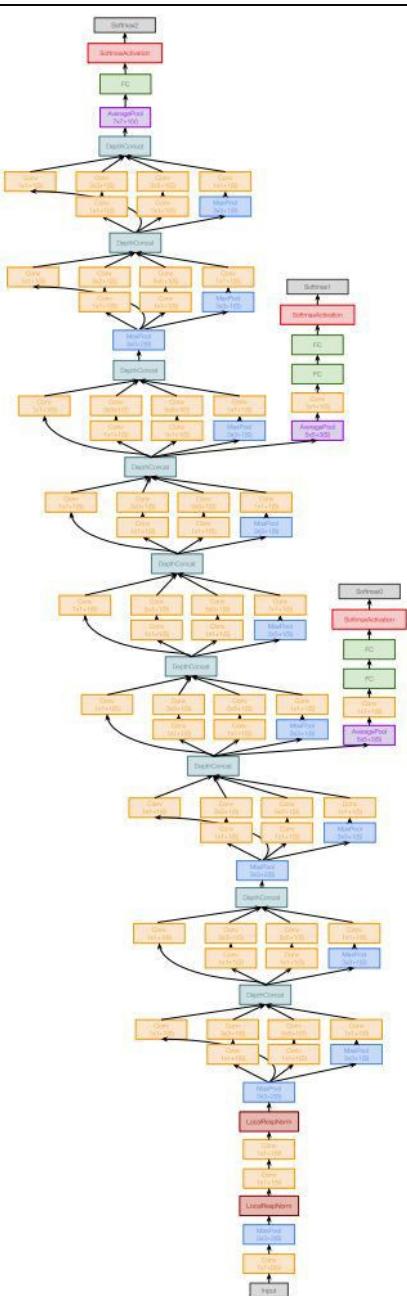
# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# GoogleNet

- *Going Deeper with Convolutions - Christian Szegedy et al.; 2015*
- ILSVRC 2014 competition winner
- Also significantly deeper than AlexNet
- x12 less parameters than AlexNet
- Focused on computational efficiency

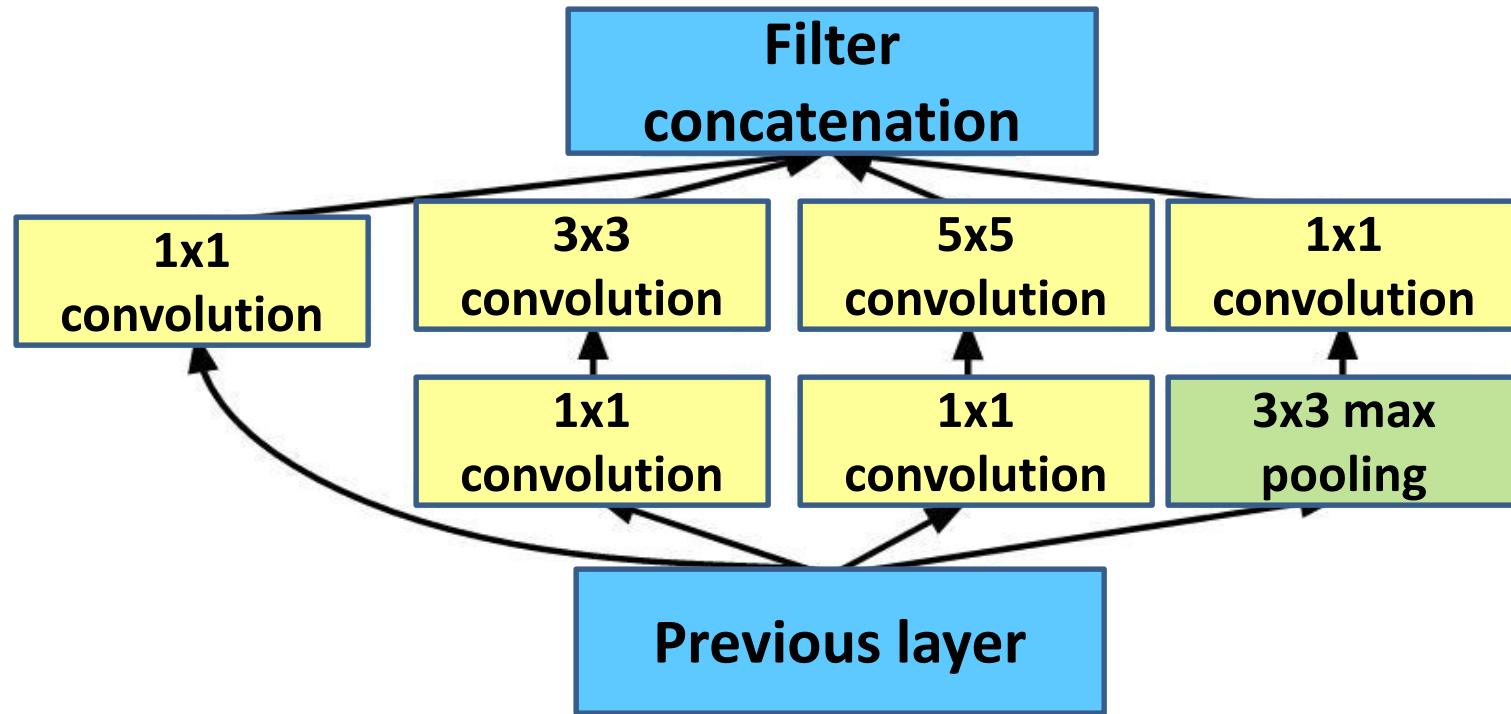
# GoogleNet



- 22 layers
- Efficient “**Inception**” module - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure
- No FC layers
- Only 5 million parameters!
- ILSVRC’14 classification winner (6.7% top 5 error)

# GoogleNet

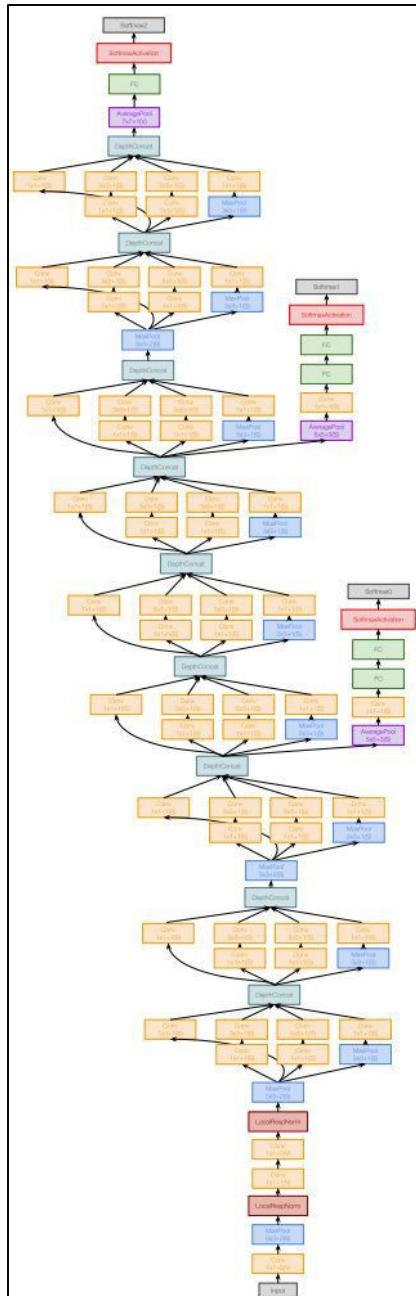
“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other



# GoogleNet

## Details/Retrospectives :

- Deeper networks, with computational efficiency
- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

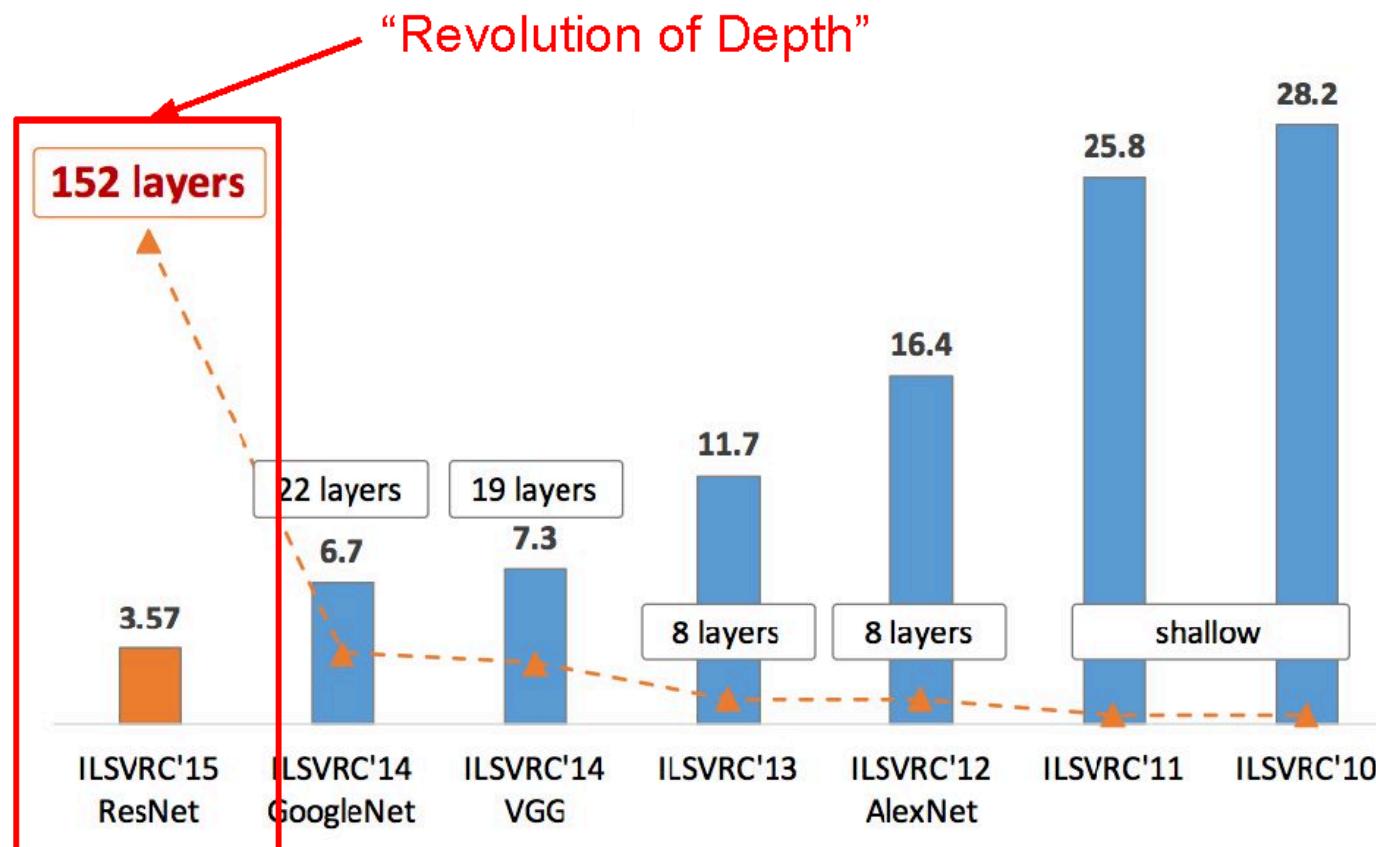




# GoogleNet

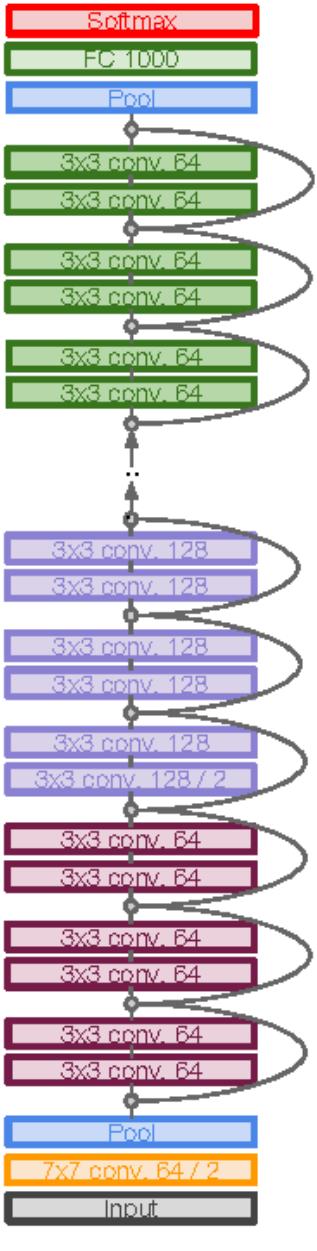
Introduced the idea that CNN layers **didn't always have to be stacked up sequentially**. Coming up with the Inception module, the authors showed that a creative structuring of layers can lead to improved performance and **computationally efficiency**.

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# ResNet

- *Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015*
- Extremely deep network – 152 layers
- Deeper neural networks are more difficult to train.
- Deep networks suffer from vanishing and exploding gradients.
- Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

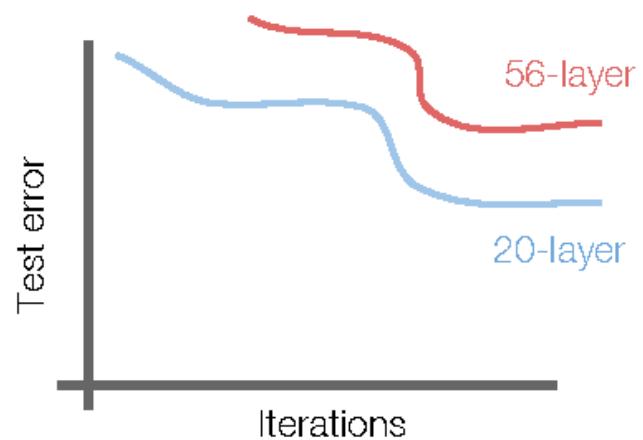
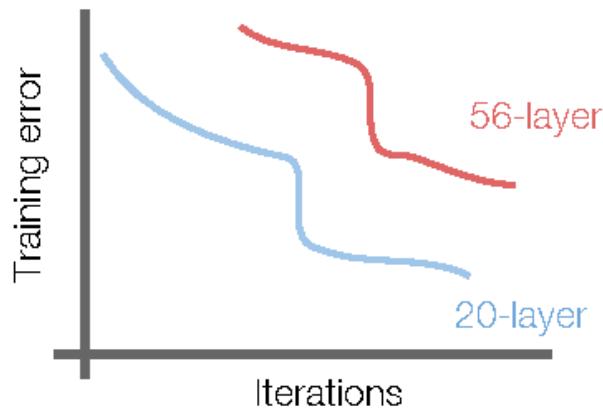


# ResNet

- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)  
Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

# ResNet

- What happens when we continue stacking deeper layers on a convolutional neural network?



- 56-layer model performs worse on both training and test error  
-> The deeper model performs worse (not caused by overfitting)!

# ResNet

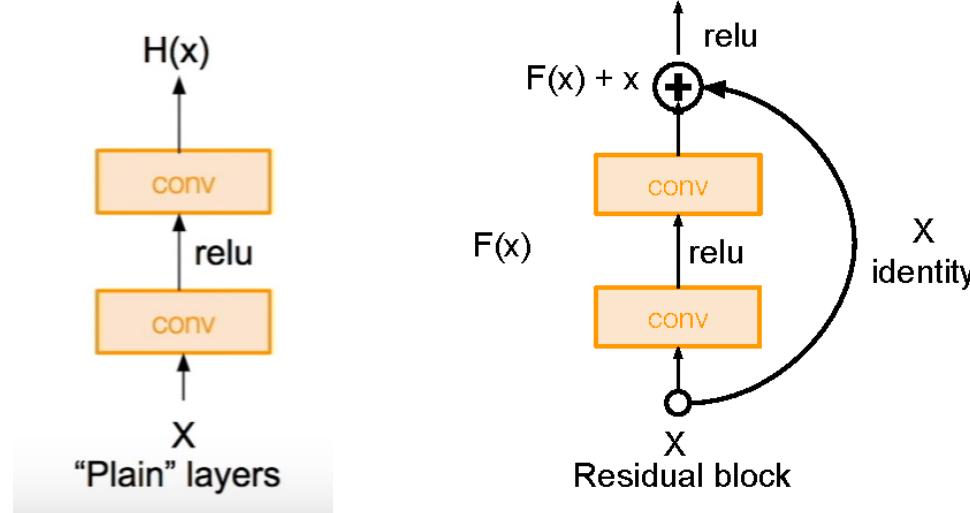
- **Hypothesis:** The problem is an optimization problem. Very deep networks are harder to optimize.
- **Solution:** Use network layers to fit residual mapping instead of directly trying to fit a desired underlying mapping.
- We will use **skip connections** allowing us to take the activation from one layer and feed it into another layer, much deeper into the network.
- Use layers to fit residual  $F(x) = H(x) - x$  instead of  $H(x)$  directly

# ResNet

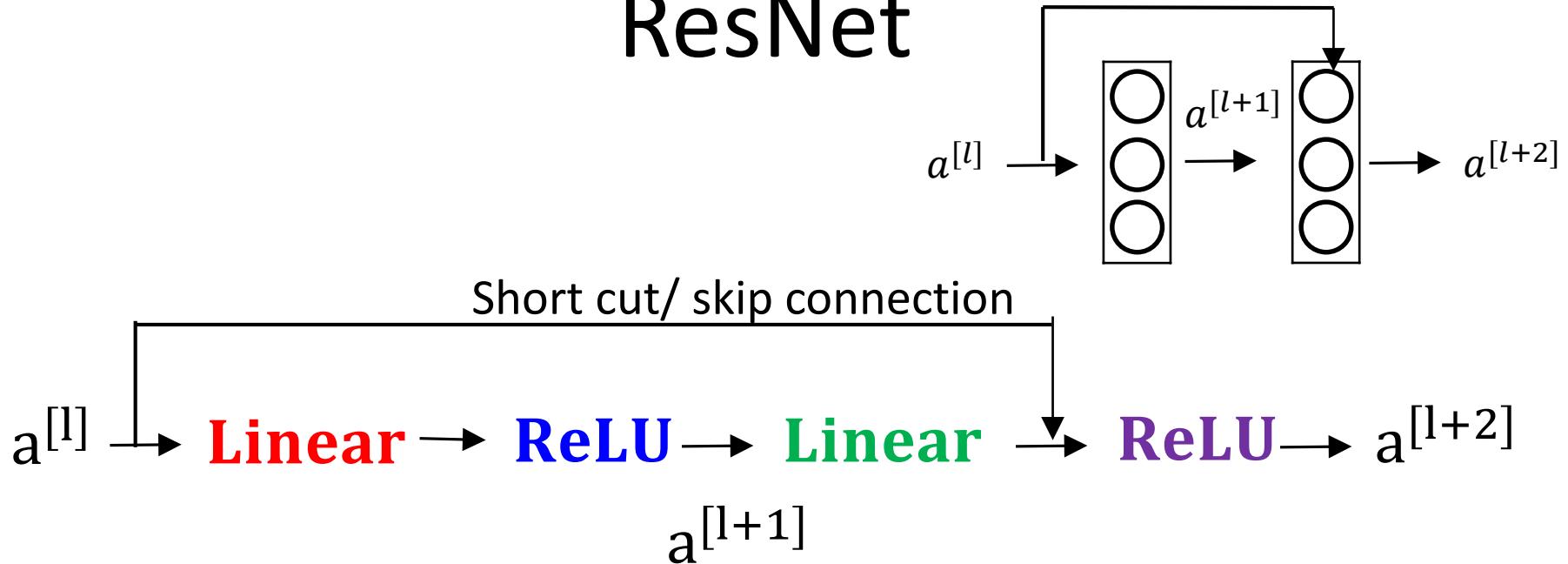
## Residual Block

Input  $x$  goes through conv-relu-conv series and gives us  $F(x)$ . That result is then added to the original input  $x$ . Let's call that  $H(x) = F(x) + x$ .

In traditional CNNs,  $H(x)$  would just be equal to  $F(x)$ . So, instead of just computing that transformation (straight from  $x$  to  $F(x)$ ), we're computing the term that we have to *add*,  $F(x)$ , to the input,  $x$ .



# ResNet

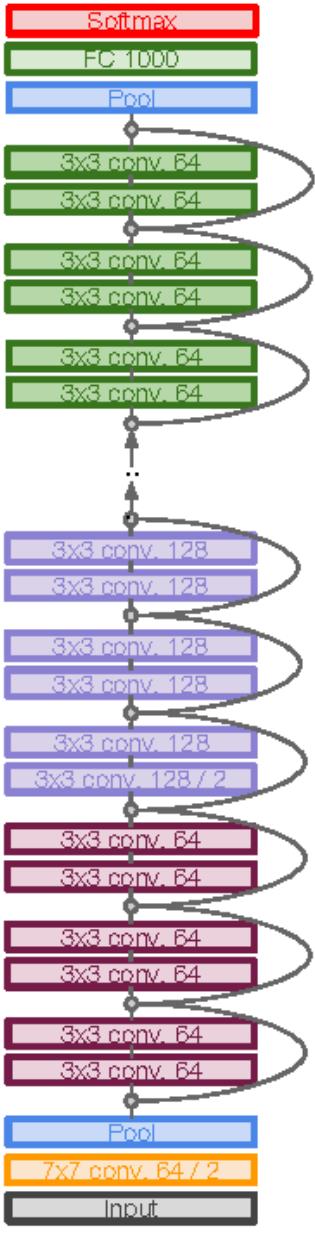


$$\mathbf{z}^{[l+1]} = \mathbf{W}^{[l+1]} \mathbf{a}^{[l]} + \mathbf{b}^{[l+1]} \quad \mathbf{z}^{[l+2]} = \mathbf{W}^{[l+2]} \mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]}$$

$$\mathbf{a}^{[l+1]} = g(\mathbf{z}^{[l+1]})$$

$$\mathbf{a}^{[l+2]} = g(\mathbf{z}^{[l+2]})$$

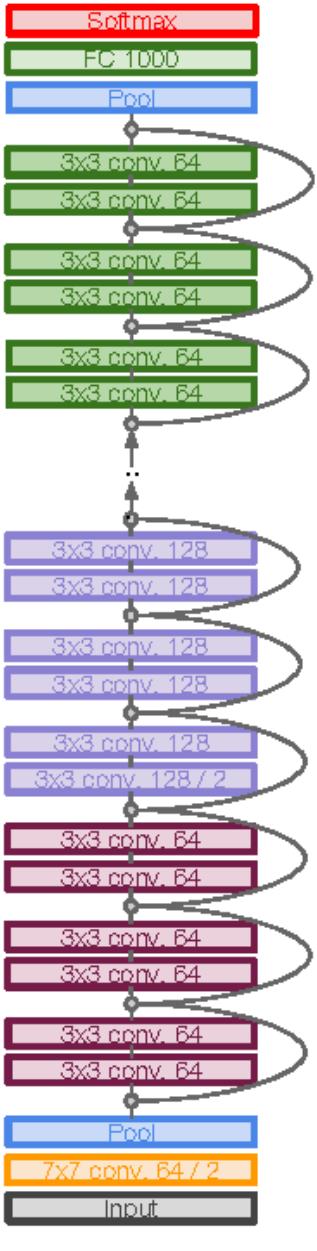
$$\mathbf{a}^{[l+2]} = g(\mathbf{z}^{[l+2]} + \mathbf{a}^{[l]}) = g(\mathbf{W}^{[l+2]} \mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]} + \mathbf{a}^{[l]})$$



# ResNet

## Full ResNet architecture:

- Stack residual blocks
- Every residual block has two  $3 \times 3$  conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



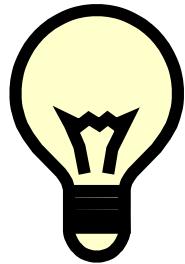
# ResNet

- Total depths of 34, 50, 101, or 152 layers for ImageNet
- For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)

# ResNet

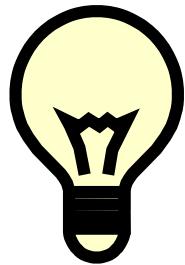
## Experimental Results:

- Able to train very deep networks without degrading
- Deeper networks now achieve lower training errors as expected

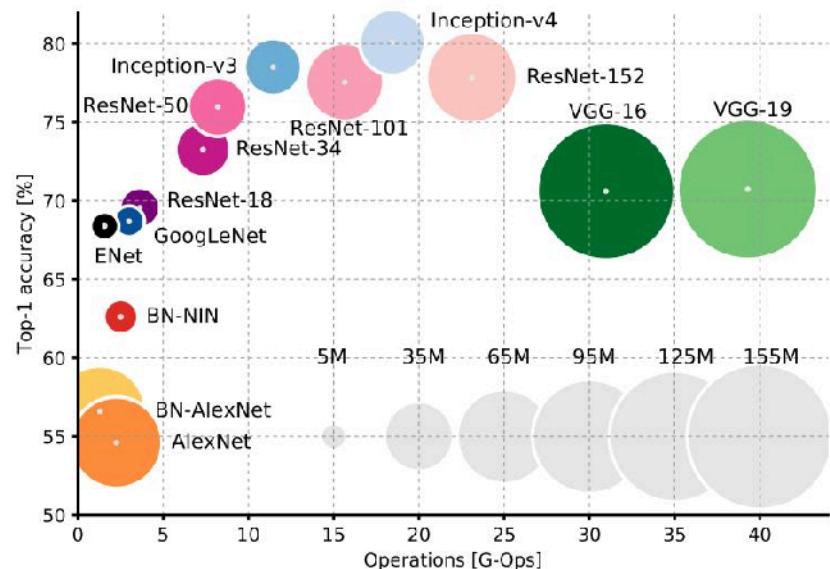
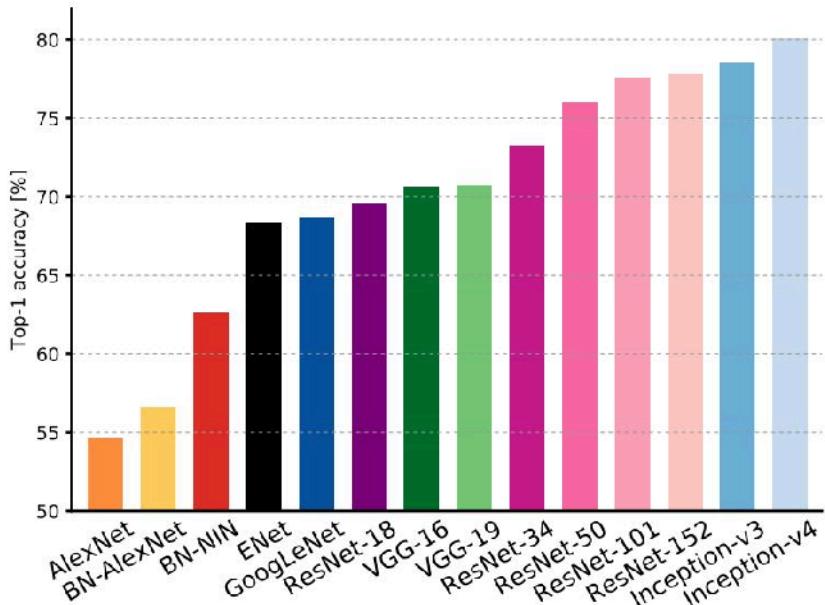


# ResNet

The **best** CNN architecture that we currently have and is a great innovation for the idea of residual learning.  
Even better than human performance!

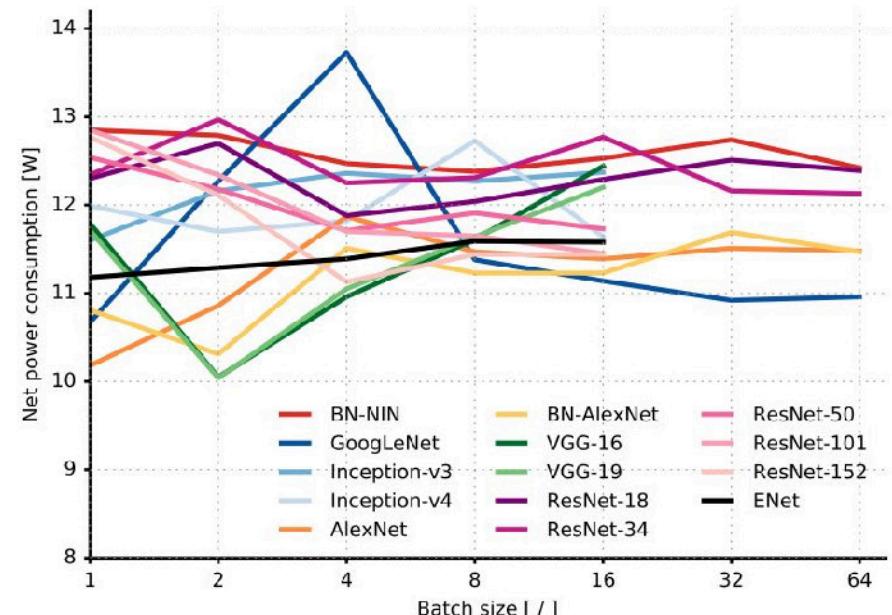
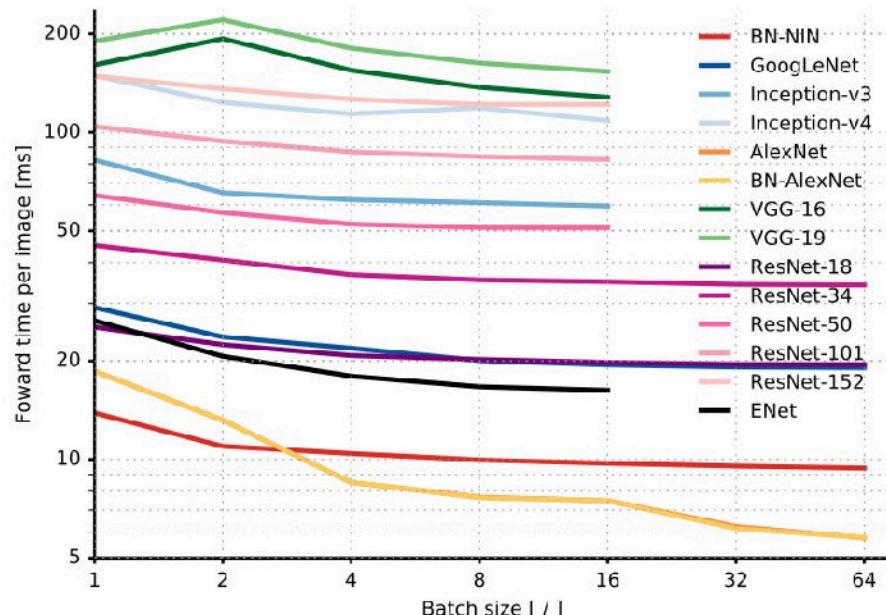


# Accuracy comparison

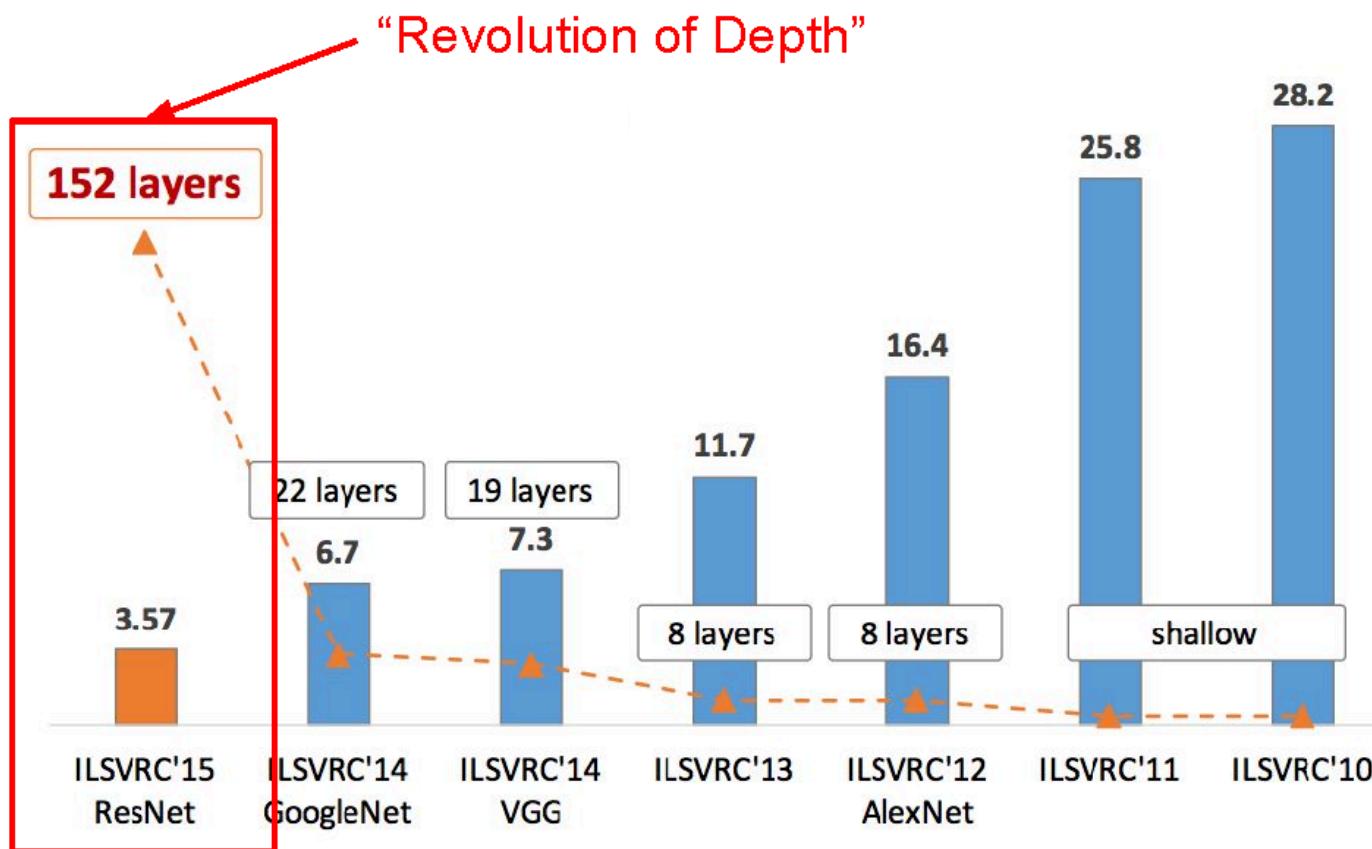




# Forward pass time and power consumption

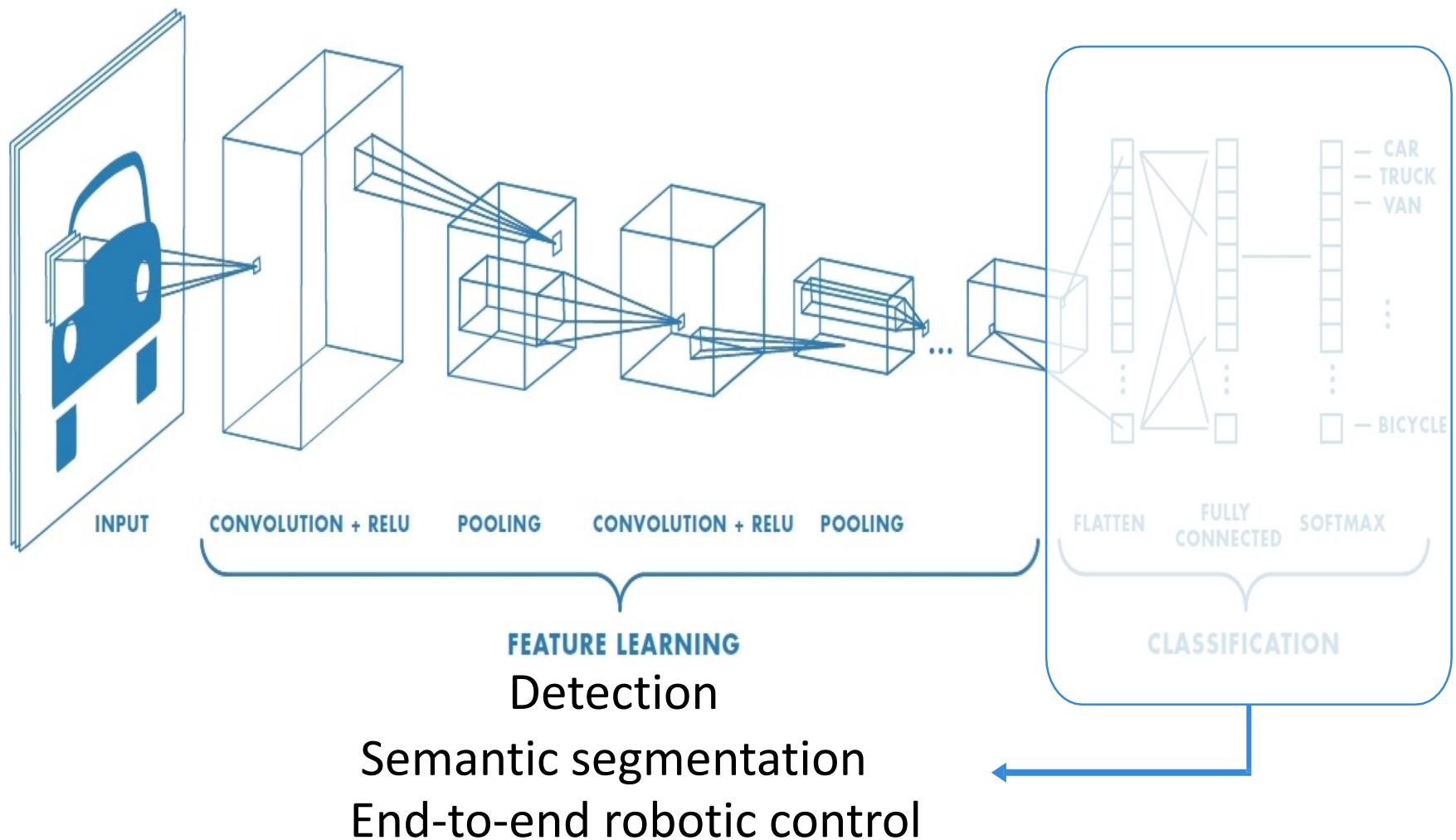


# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# Countless applications

# An Architecture for Many Applications



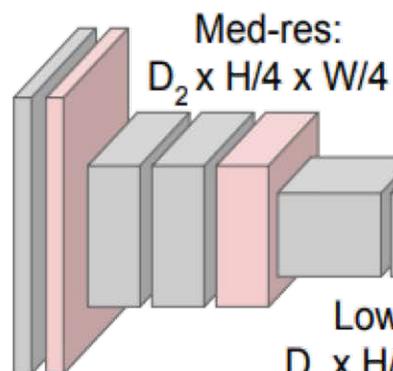
# Semantic Segmentation: Fully Convolutional Networks

FCN: Fully Convolutional Network.

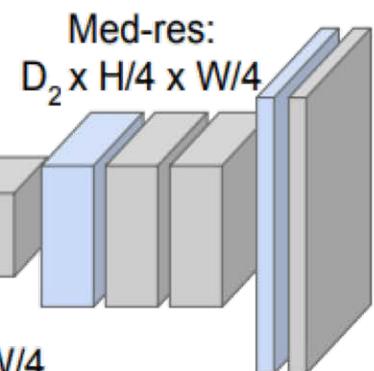
Network designed with all convolutional layers, with **downsampling** and **upsampling** operations



Input:  
 $3 \times H \times W$



High-res:  
 $D_1 \times H/2 \times W/2$



High-res:  
 $D_1 \times H/2 \times W/2$

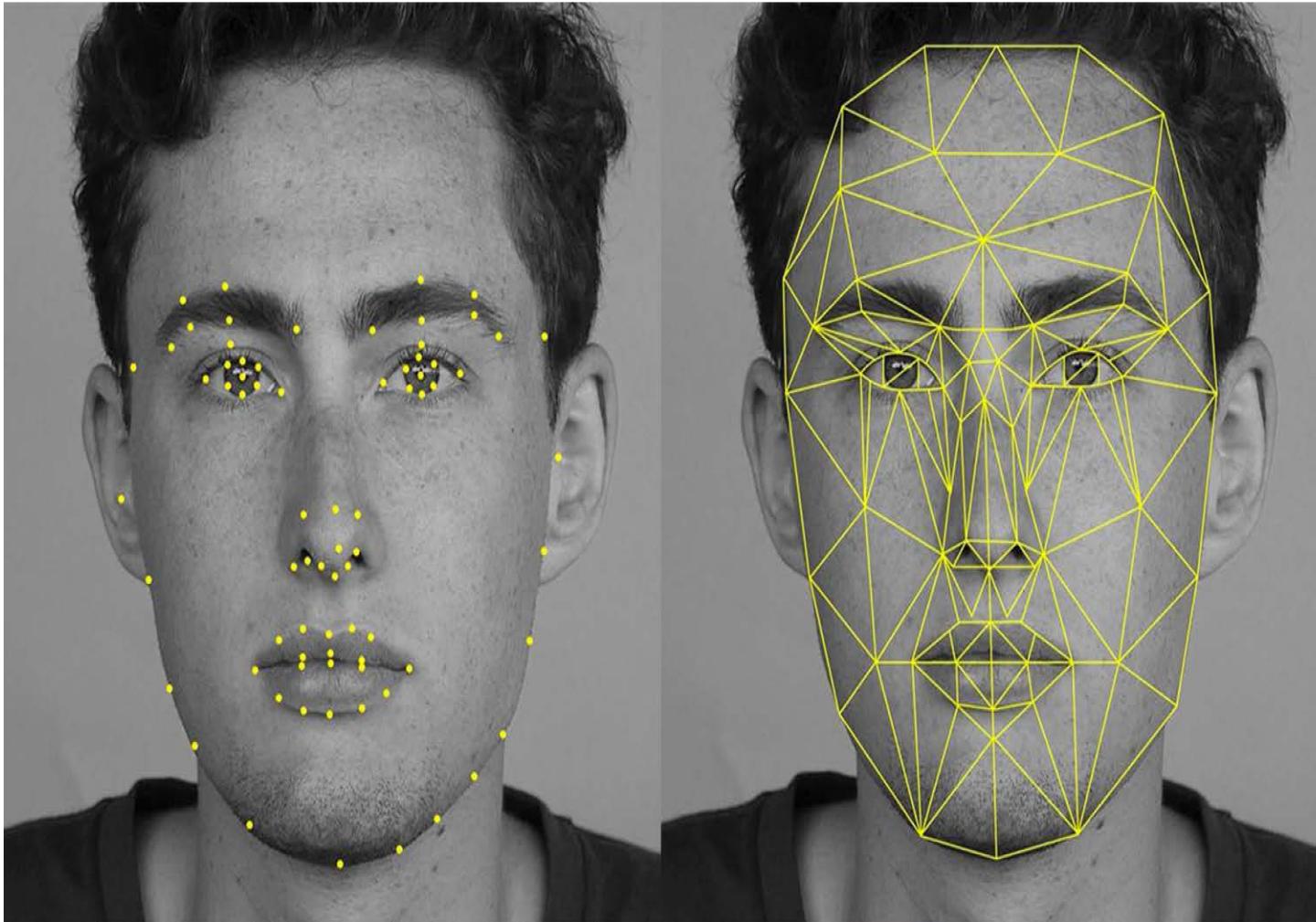
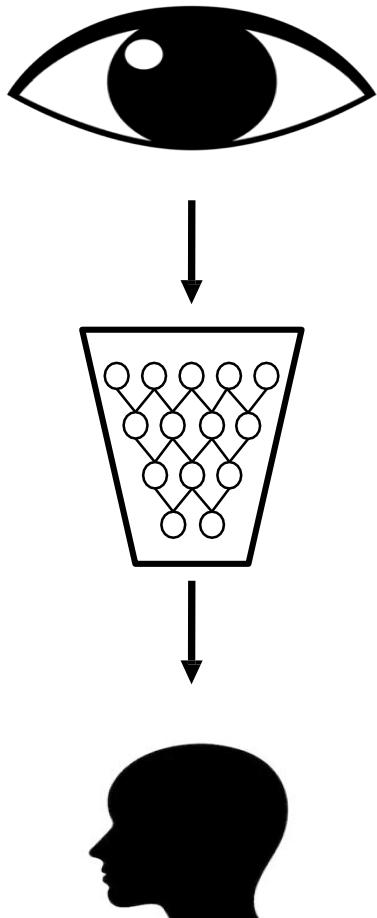


Predictions:  
 $H \times W$

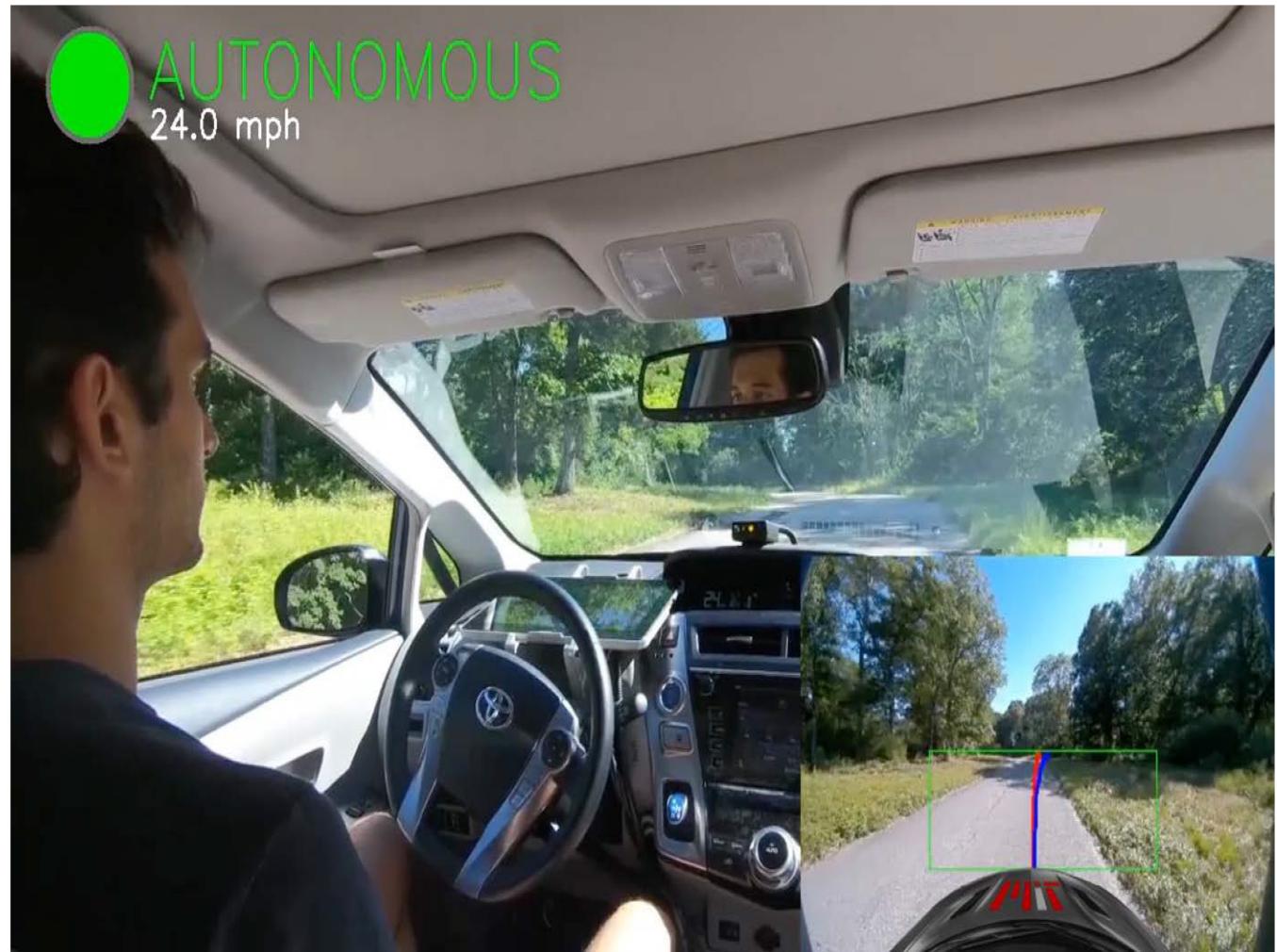
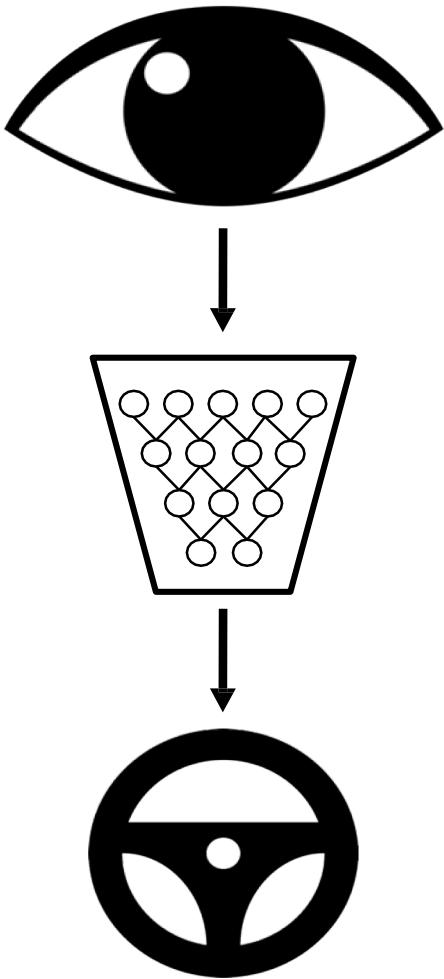


`tf.keras.layers.Conv2DTranspose`

# Facial Detection & Recognition



# Self-Driving Cars

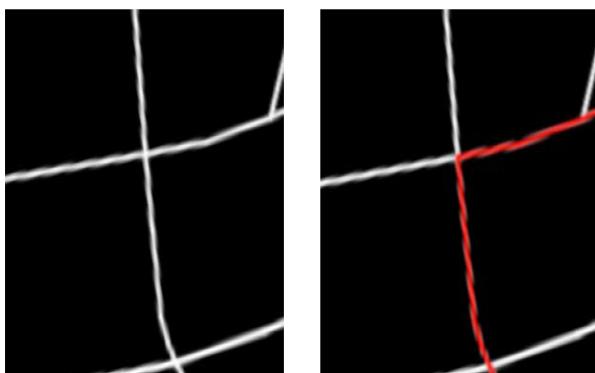


# Self-Driving Cars: Navigation from Visual Perception

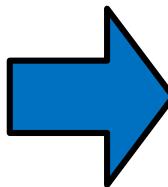
**Raw Perception**  
 $I$   
(ex.camera)



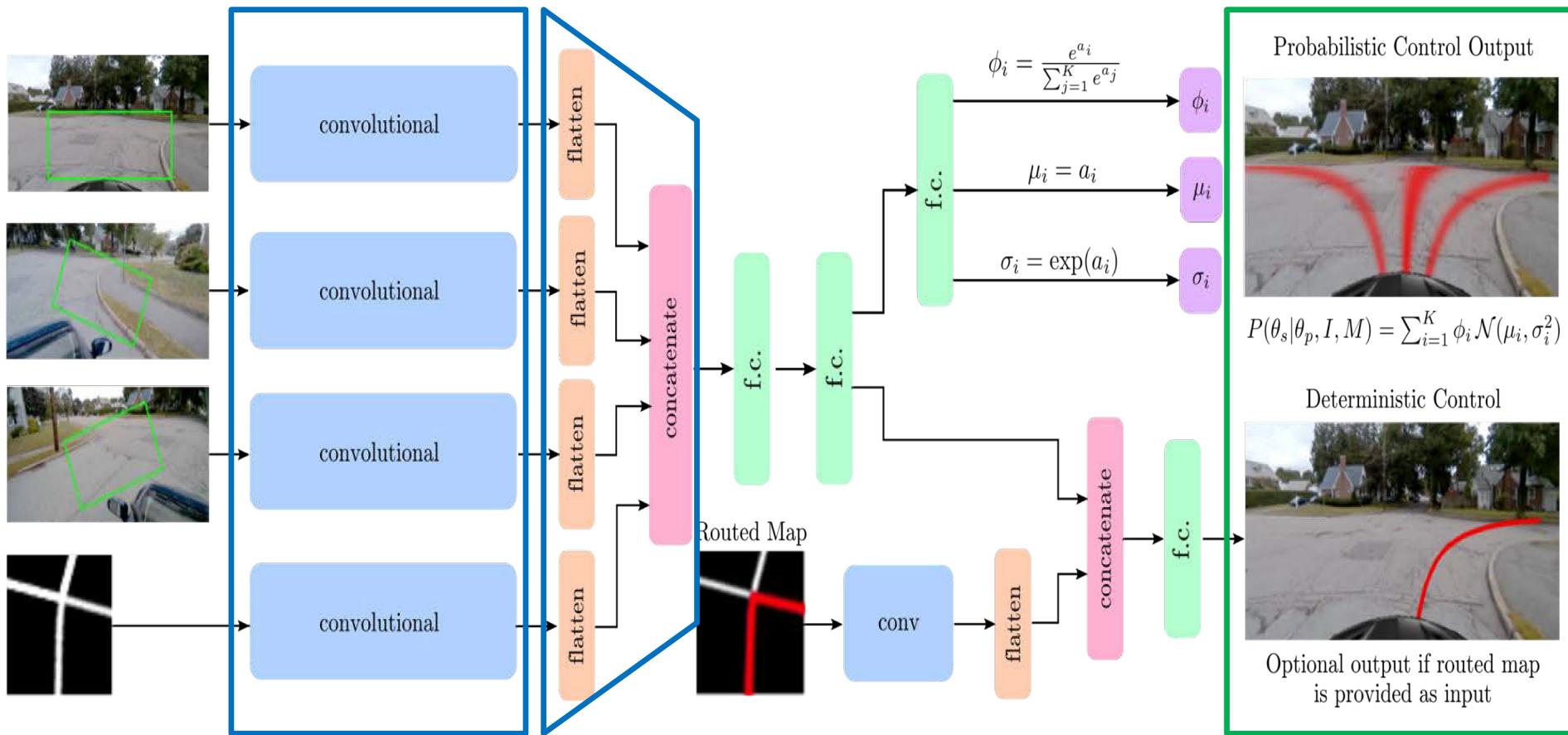
**Coarse Maps**  
 $M$   
(ex.GPS)



**Possible Control Commands**

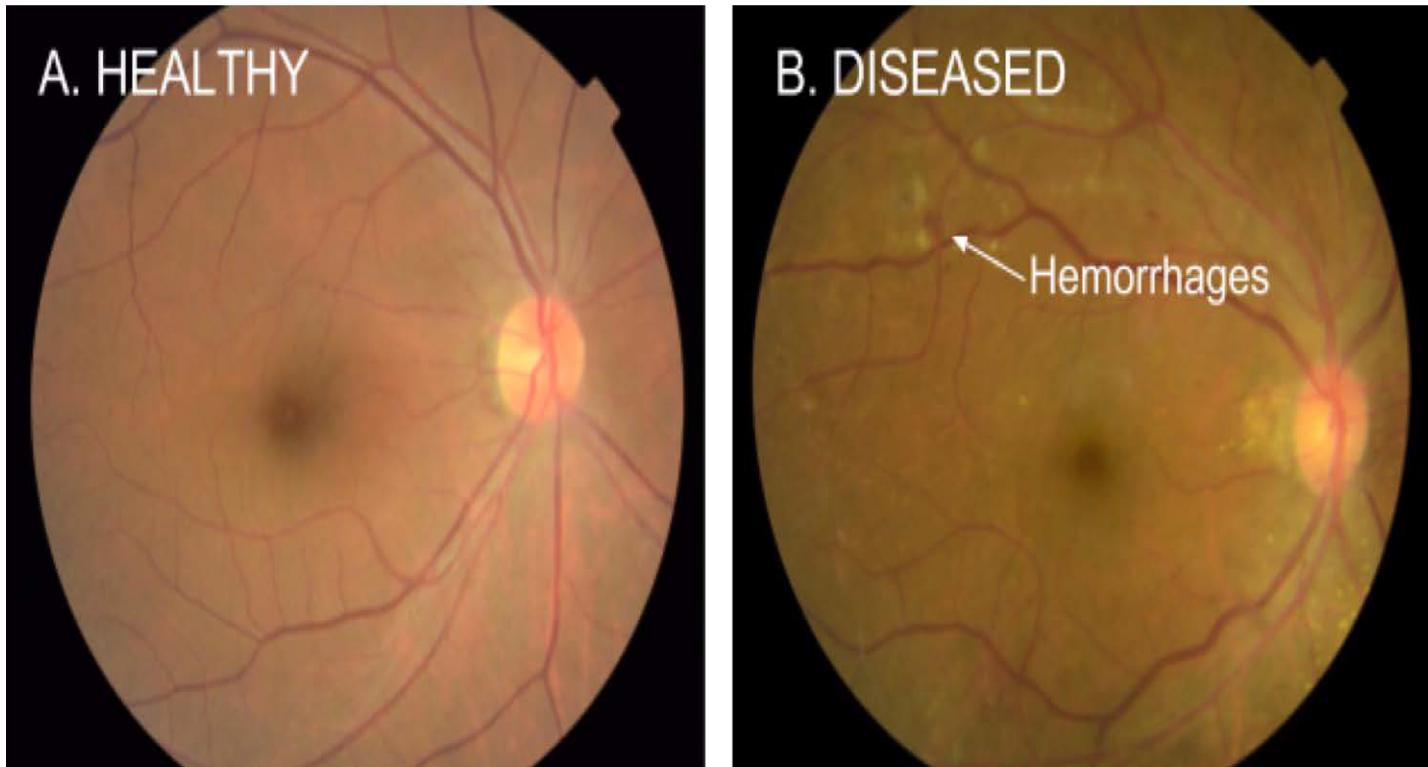
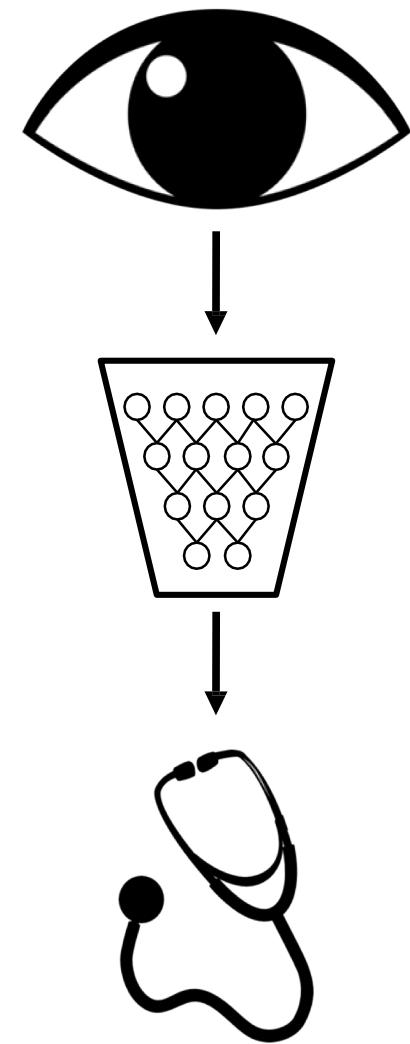


# End-to-End Framework for Autonomous Navigation



Entire model trained end-to-end  
**without any human labelling or annotations**

# Medicine, Biology, Healthcare

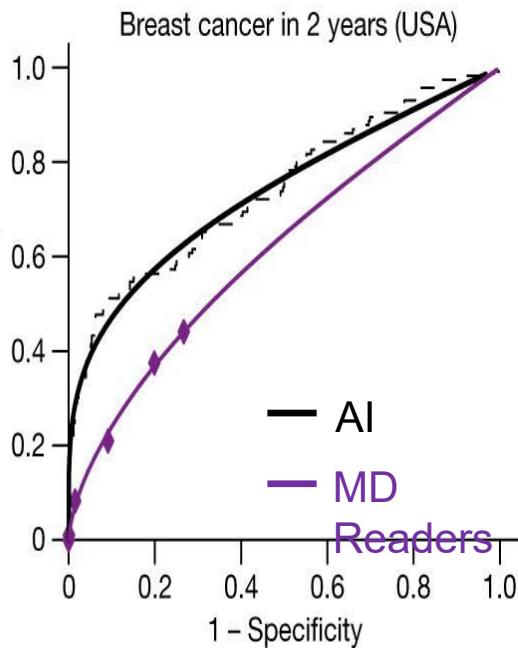


# Breast Cancer Screening

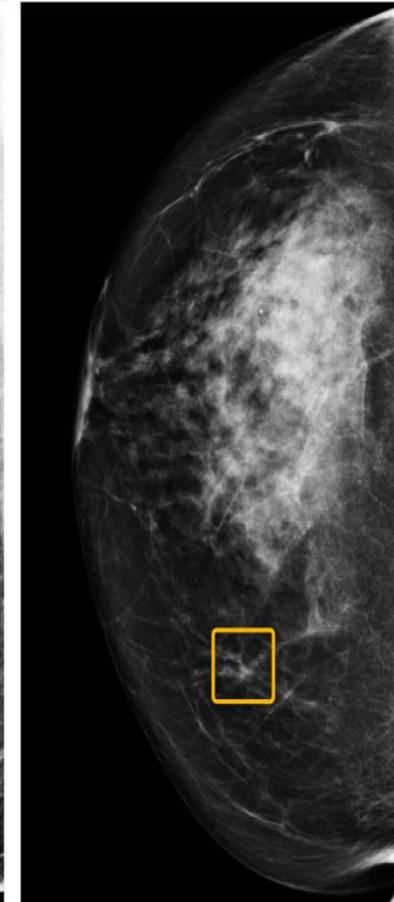
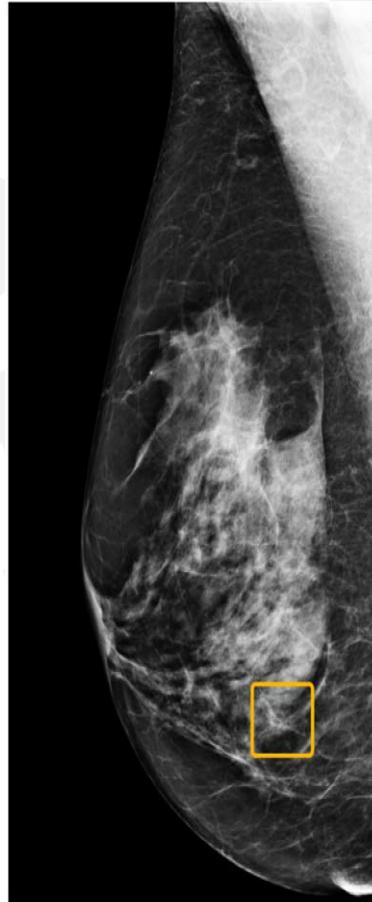
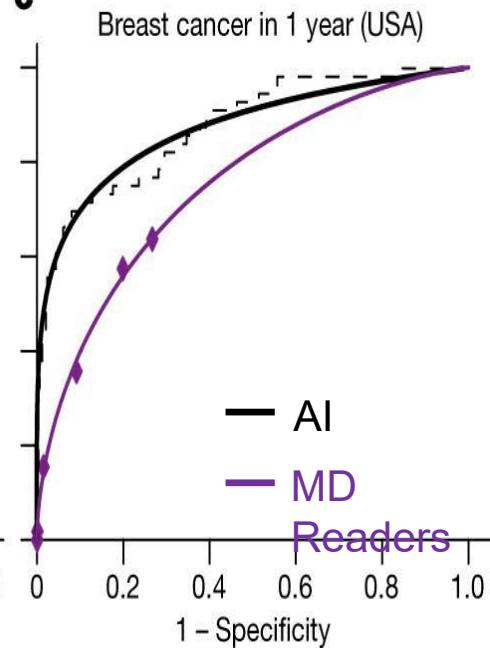
## International evaluation of an AI system for breast cancer screening

nature

b



c

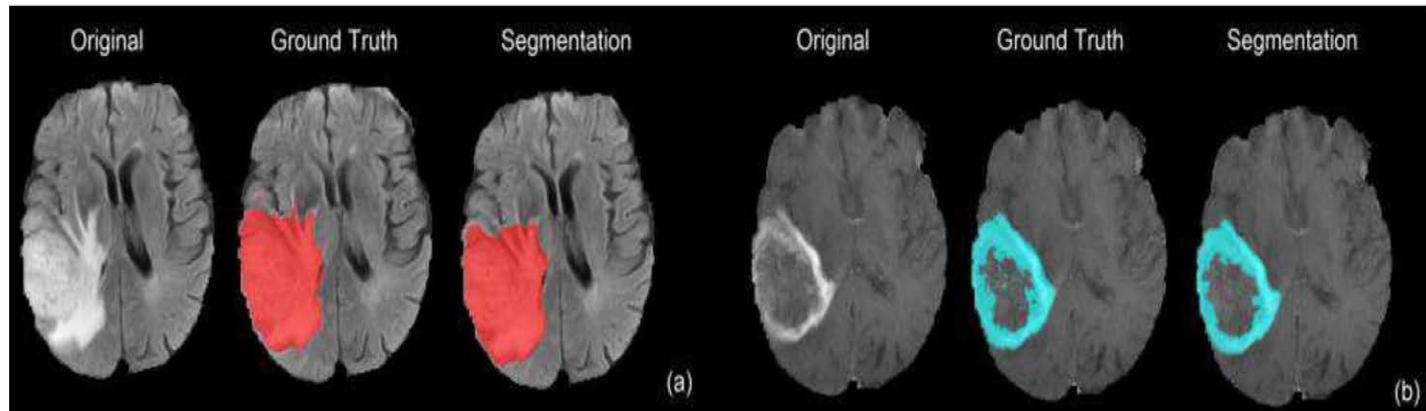


CNN-based system outperformed expert radiologists at detecting breast cancer from mammograms

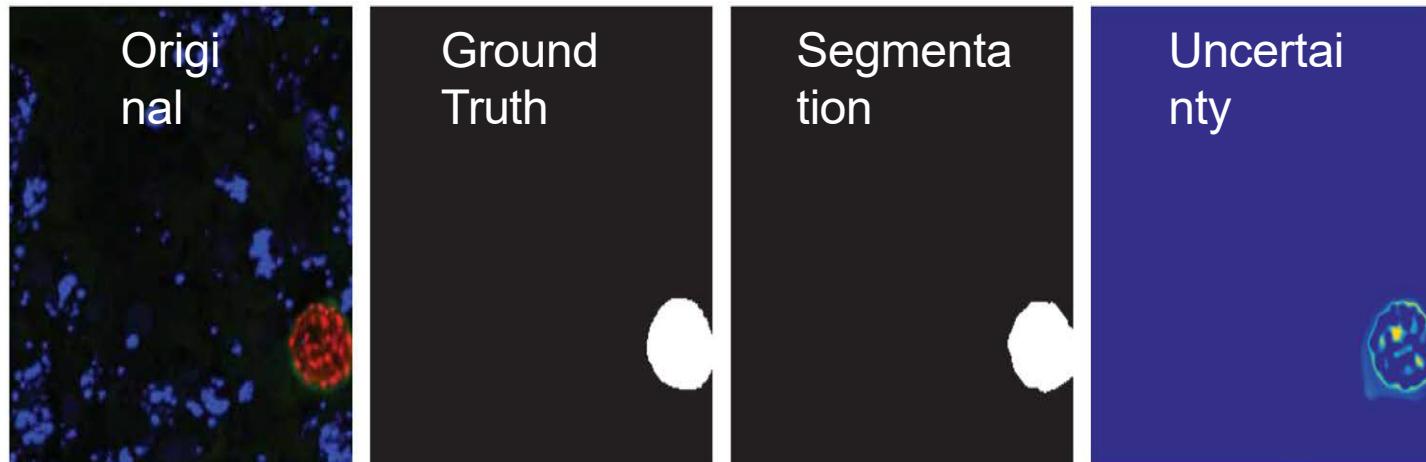
Breast cancer case missed by radiologist but detected by AI

# Semantic Segmentation: Biomedical Image Analysis

Brain Tumors  
Dong+ *MIUA*  
2017.

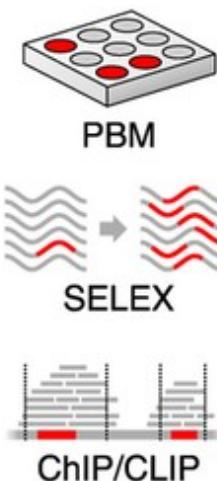


Malaria Infection  
Soleimany+ *arXiv*  
2019.



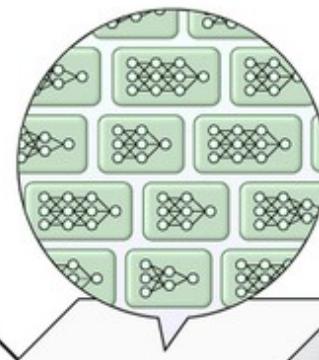
# DeepBind

## 1. High-throughput experiments



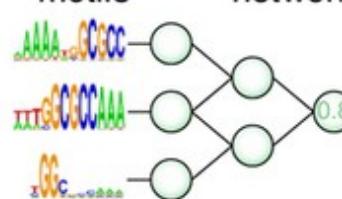
## 2. Massively parallel deep learning

### Automatic model training



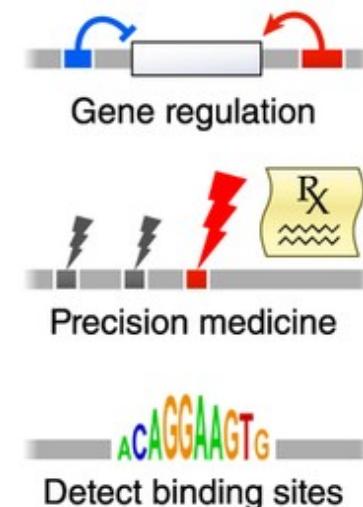
Large-scale data sets

### New motifs Prediction network



DeepBind models

## 3. Community needs



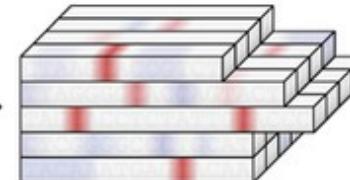
### Current batch of inputs

CTAACGCCGGTCT  
TTAGGGGCACCACTACT  
TAGCACCTCTATTGCACCC  
CTCGGGGGCCCTGCAAT  
TACAAATGAGCACAA

Convolve

Motif detectors

### Motif scans

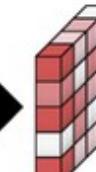


Rectify

Thresholds

Pool

### Features

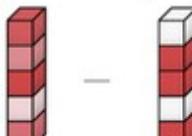


Neural network

Weights

Outputs

Targets



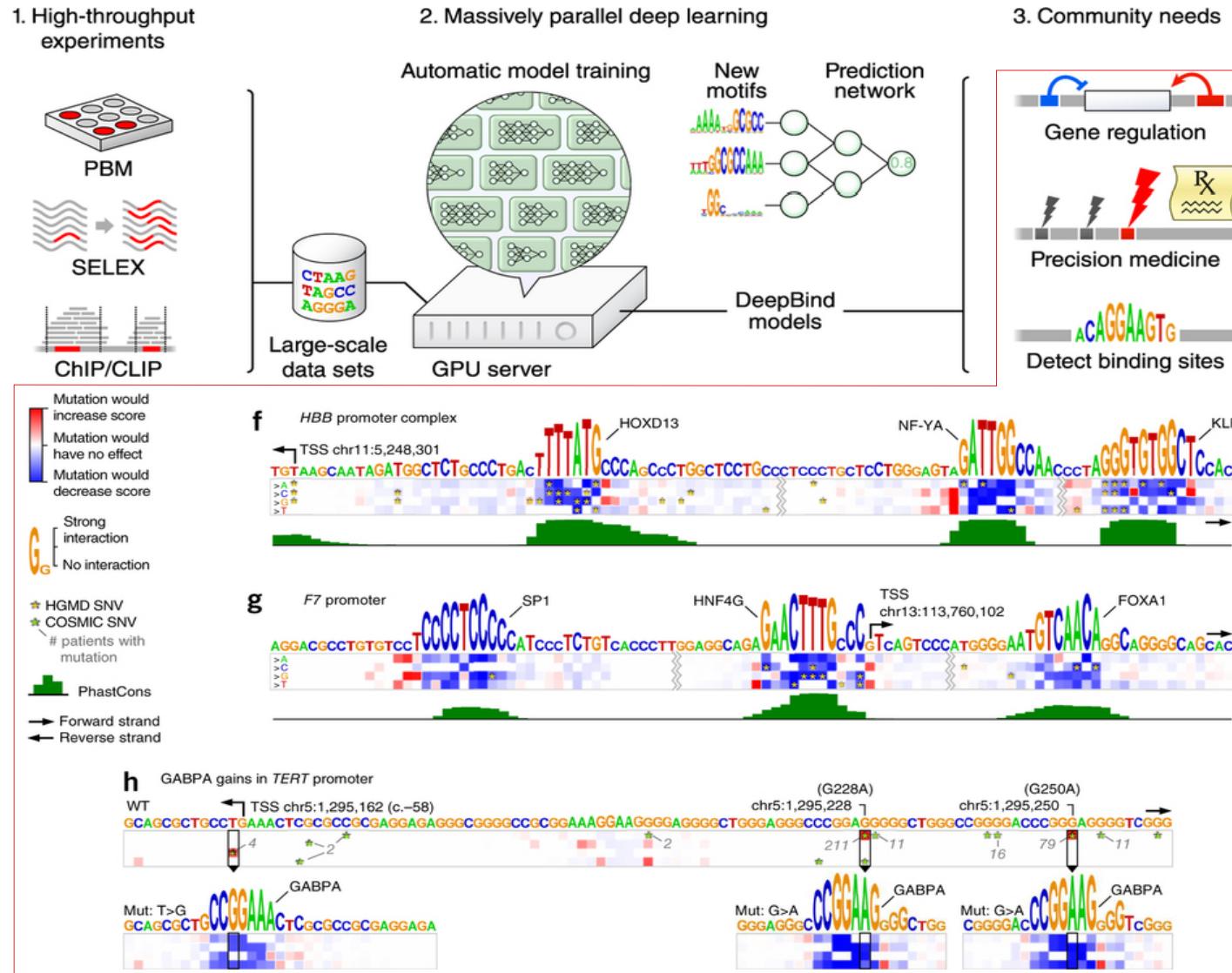
### Current model parameters

Update +

### Parameter updates

Backprop

# Predicting disease mutations



[Alipanahi et al., 2015]

# Today: Convolutional Neural Networks (CNNs)

## 1. Scene understanding and object recognition for machines (and humans)

- Scene/object recognition challenge. Illusions reveal primitives, conflicting info
- Human neurons/circuits. Visual cortex layers==abstraction. General cognition

## 2. Classical machine vision foundations: features, scenes, filters, convolution

- Spatial structure primitives: edge detectors & other filters, feature recognition
- Convolution: basics, padding, stride, object recognition, architectures

## 3. CNN foundations: LeNet, *de novo* feature learning, parameter sharing

- Key ideas: *learn* features, hierarchy, re-use parameters, back-prop filter learning
- CNN formalization: representations(Conv+ReLU+Pool)\*N layers + Fully-connected

## 4. Modern CNN architectures: millions of parameters, dozens of layers

- Feature invariance is hard: apply perturbations, learn for each variation
- ImageNet progression of best performers
- AlexNet: First top performer CNN, 60M parameters (from 60k in LeNet-5), ReLU
- VGGNet: simpler but deeper (8→19 layers), 140M parameters, ensembles
- GoogleNet: new primitive=inception module, 5M params, no FC, efficiency
- ResNet: 152 layers, vanishing gradients → fit residuals to enable learning

## 5. Countless applications: General architecture, enormous power

- Semantic segmentation, facial detection/recognition, self-driving, image colorization, optimizing pictures/scenes, up-scaling, medicine, biology, genomics

# Deep Learning for Computer Vision: Summary

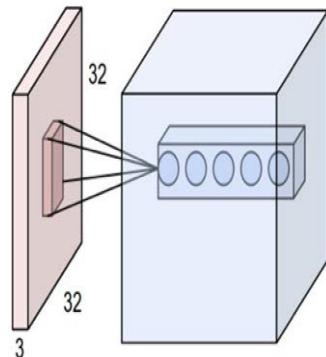
## Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction



## CNNs

- CNN architecture
- Application to classification
- ImageNet



## Applications

- Segmentation, image captioning, control
- Security, medicine, robotics

