

6.874, 6.802, 20.390, 20.490, HST.506

Computational Systems Biology

Deep Learning in the Life Sciences

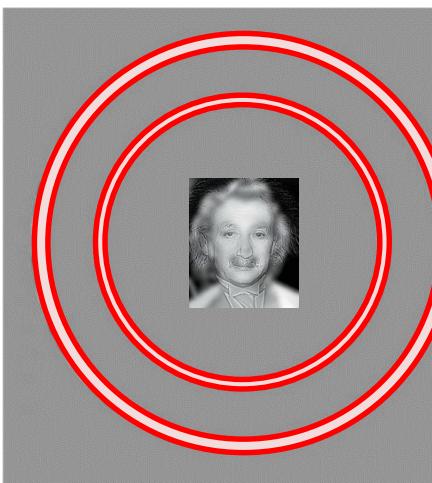
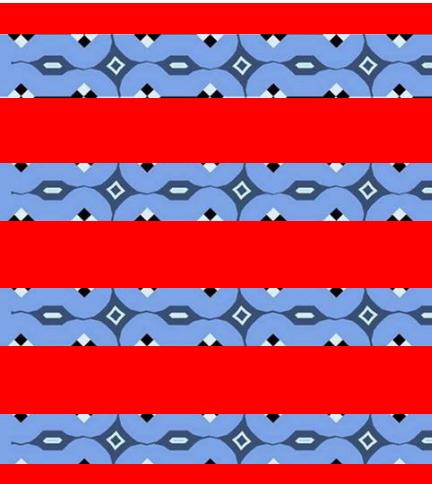
# Lecture 3: Convolutional Neural Networks

Prof. Manolis Kellis

# What do you see?



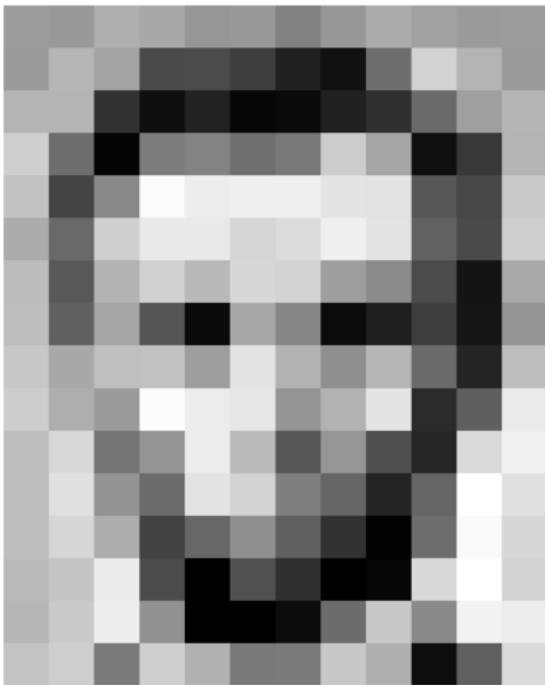
# How do you see?



# How can we help computers see?

# What computers ‘see’: Images as Numbers

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	105	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?

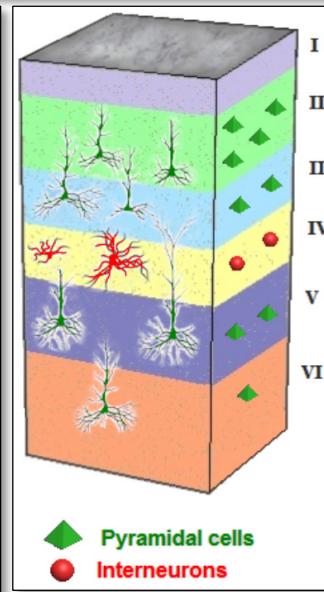
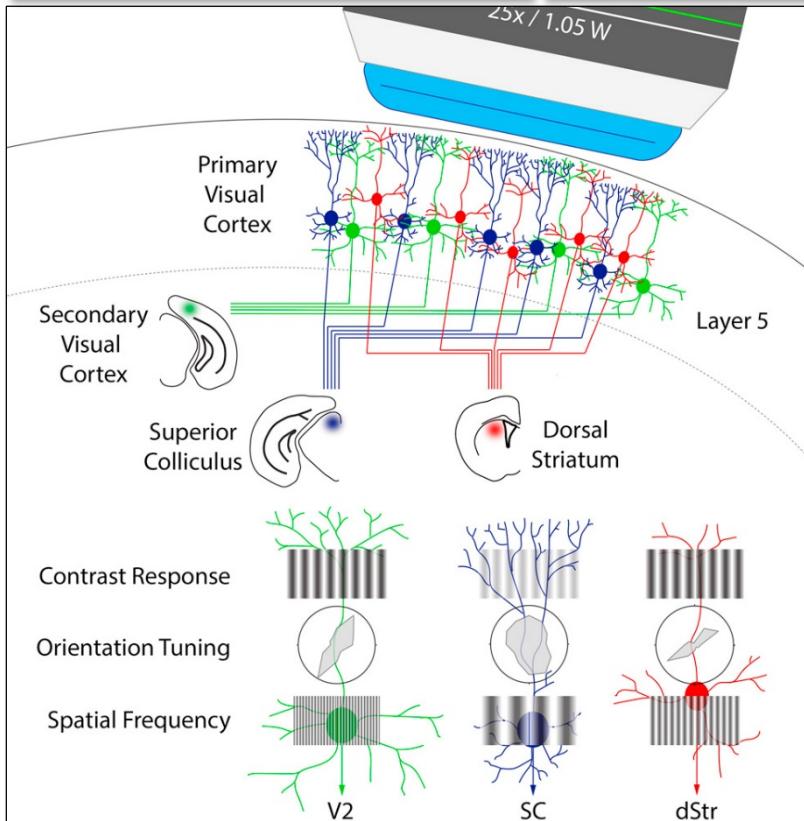
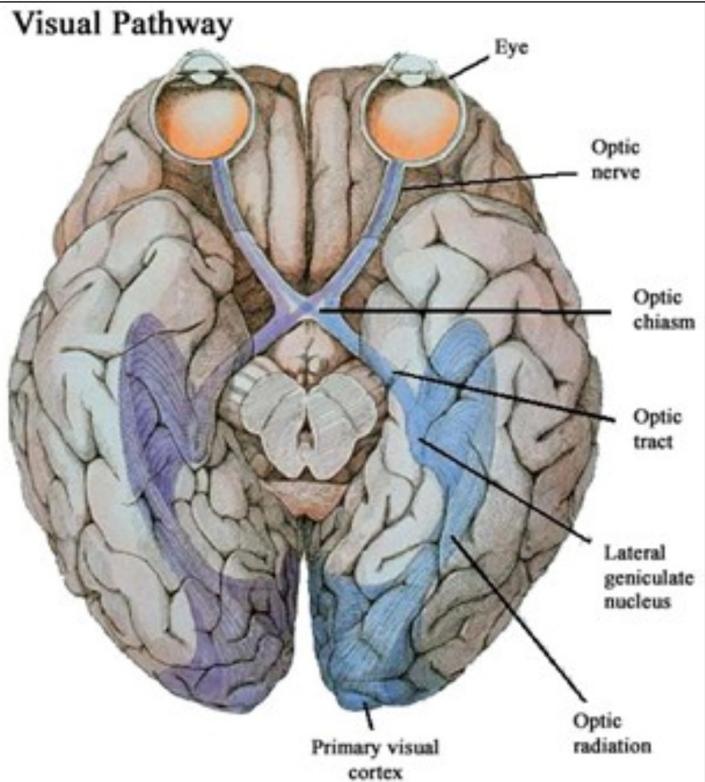
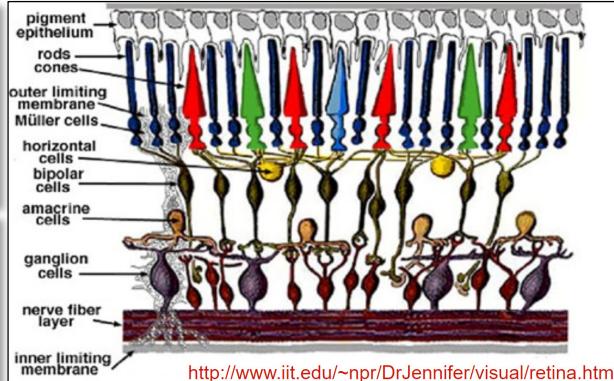
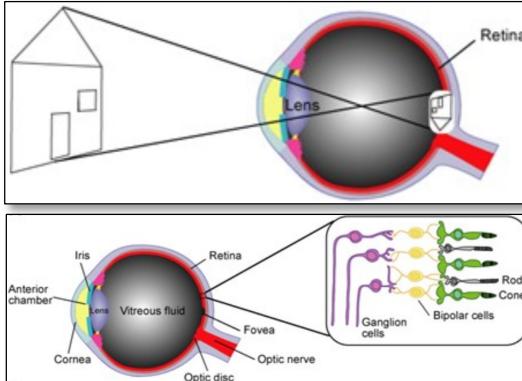
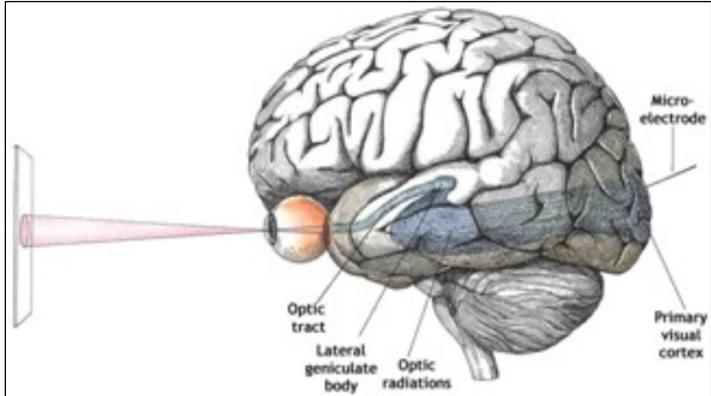
How can the computer answer this question?

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

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

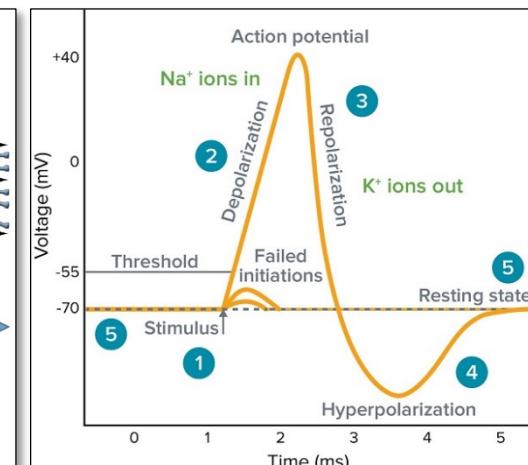
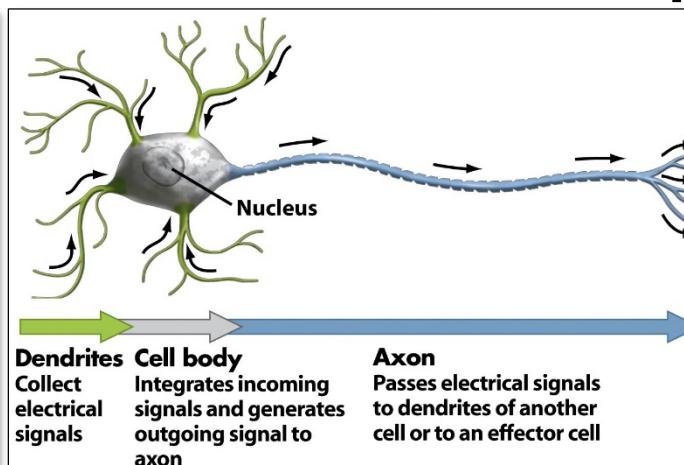
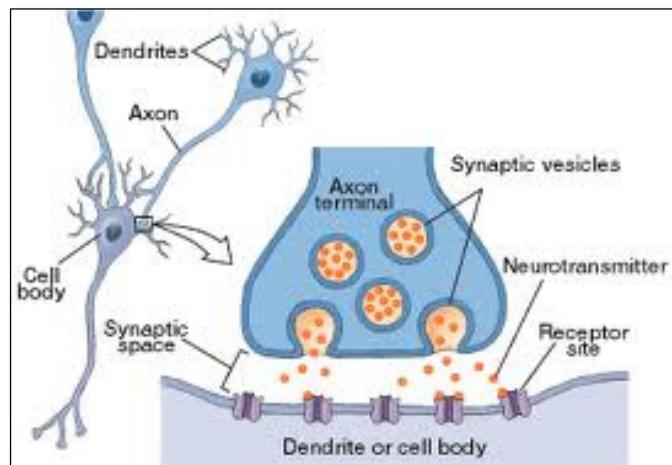
Classical machine vision roots  
in study of human/animal brains

# Inspiration: animal visual cortex



- Layers of neurons: pixels, edges, shapes, primitives, scenes

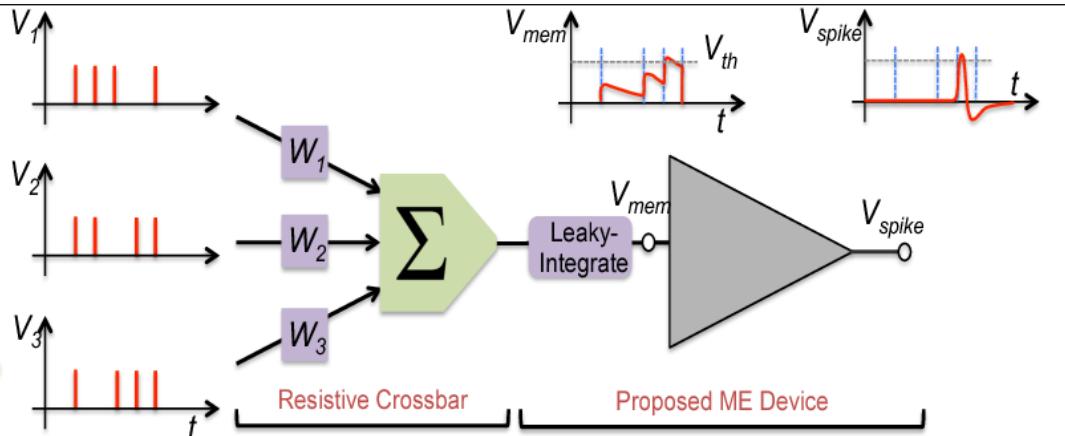
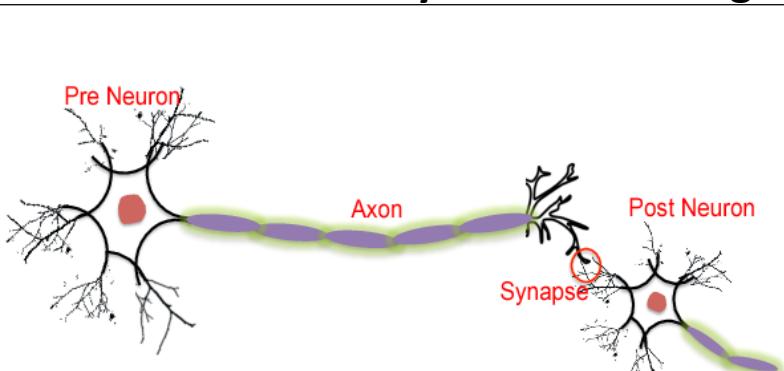
# Primitives: Neurons & action potentials



- Chemical accumulation across dendritic connections
- Pre-synaptic axon  
→ post-synaptic dendrite  
→ neuronal cell body

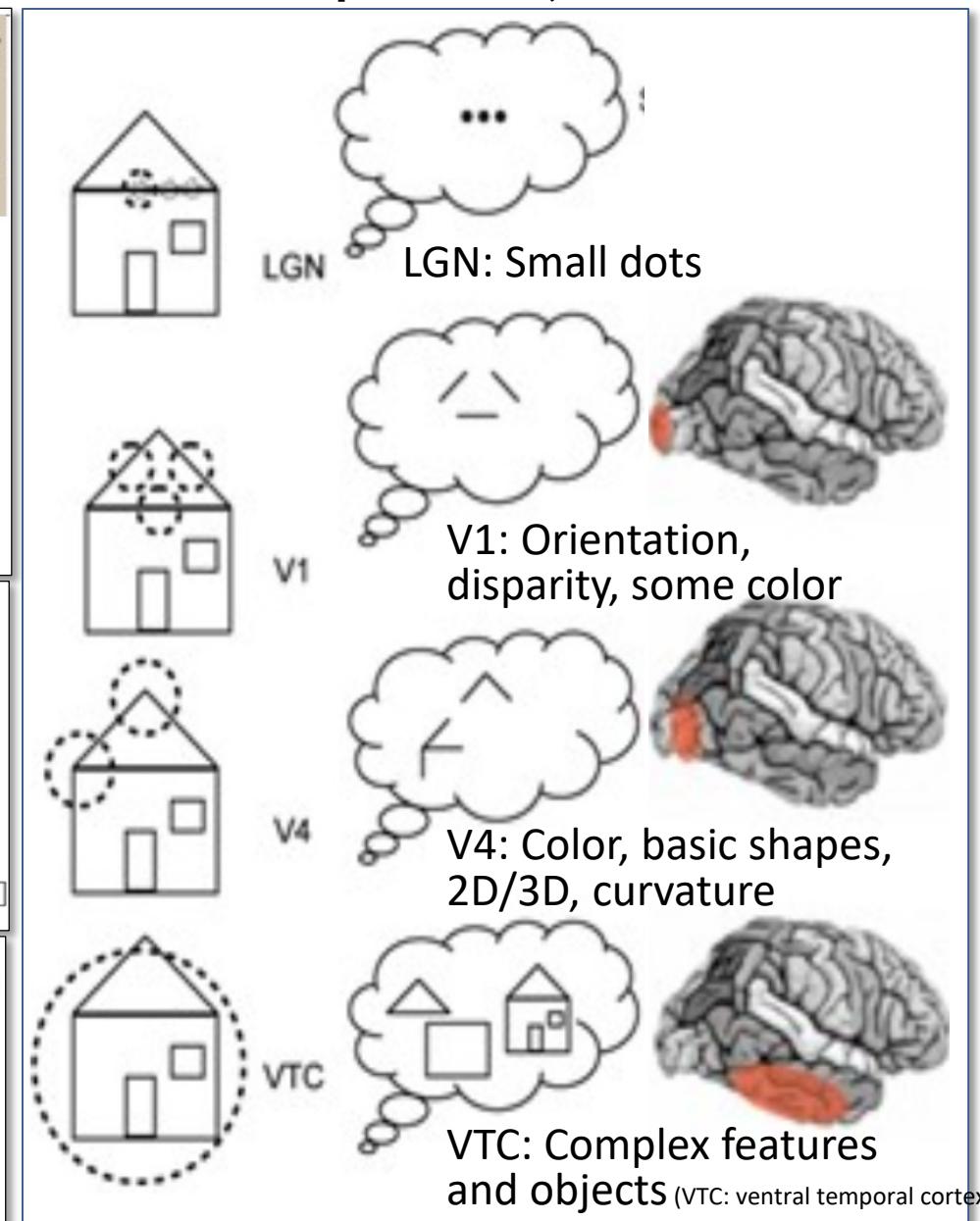
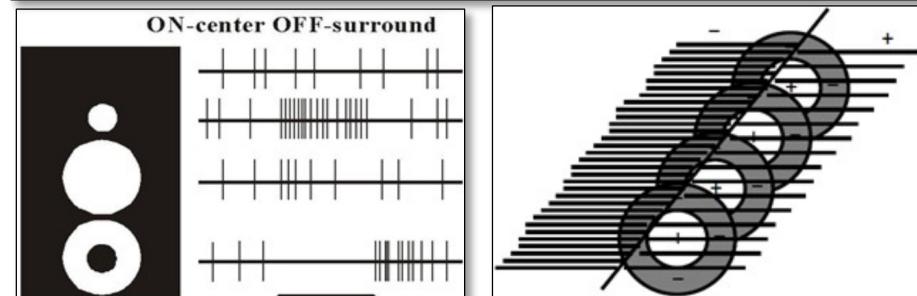
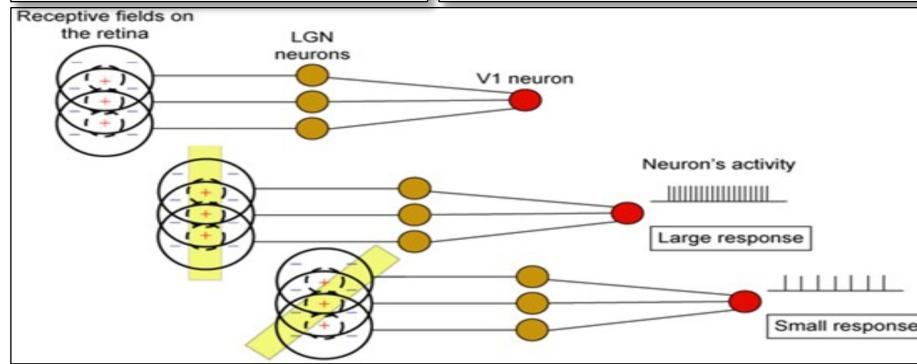
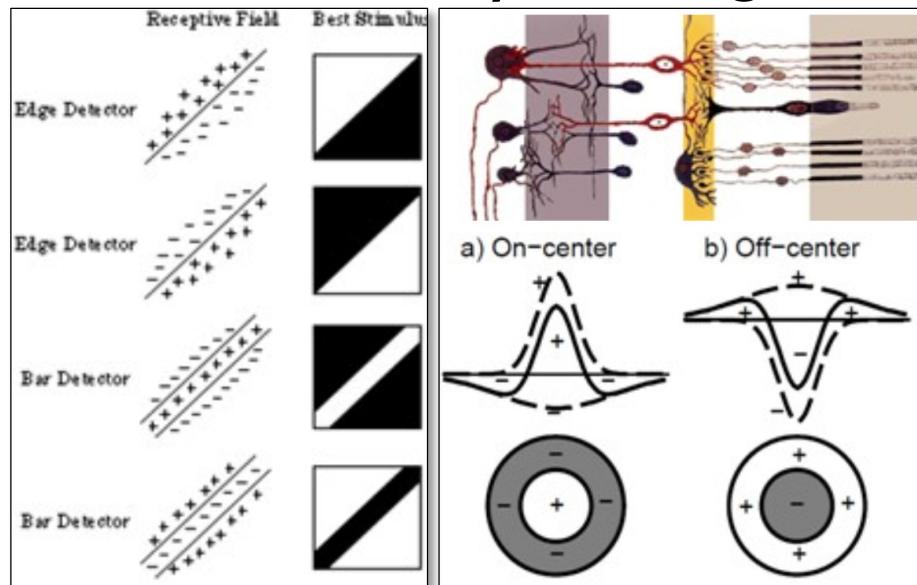
- Each neuron receives multiple signals from its many dendrites
- When threshold crossed, it fires
- Its axon then sends outgoing signal to downstream neurons

- Weak stimuli ignored
- Sufficiently strong cross activation threshold
- 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 networks
- 86 billion neurons, each connects to 10k neurons, 1 quadrillion ( $10^{12}$ ) connections

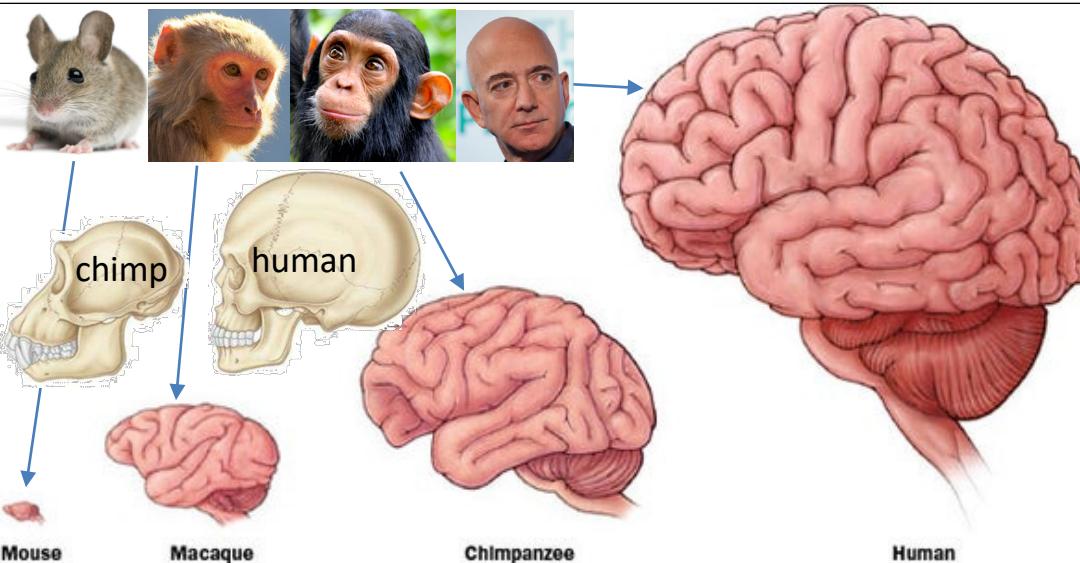
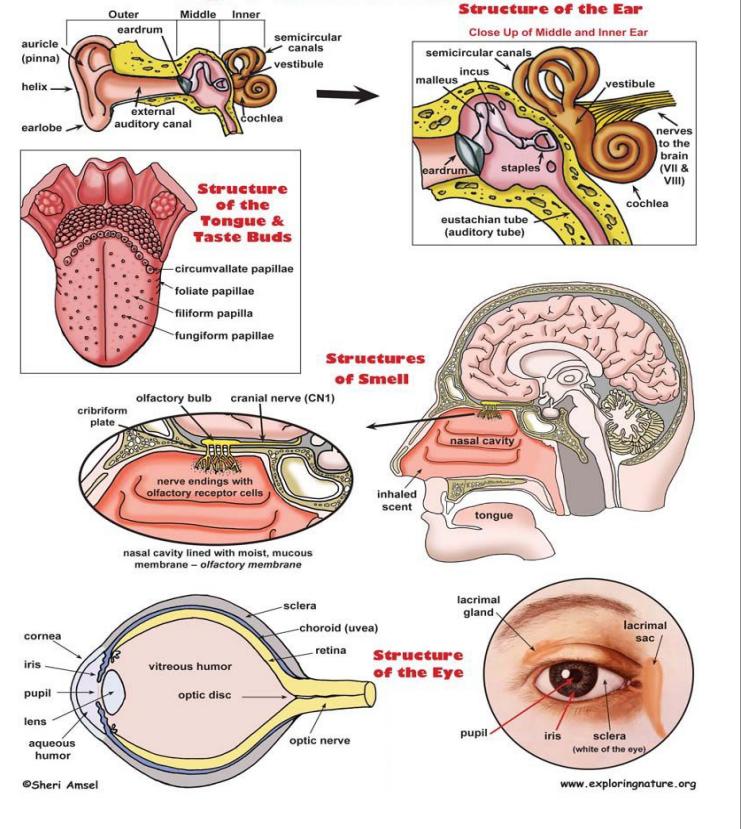
# Abstraction layers: edges, bars, dir., shapes, objects, scenes



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

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

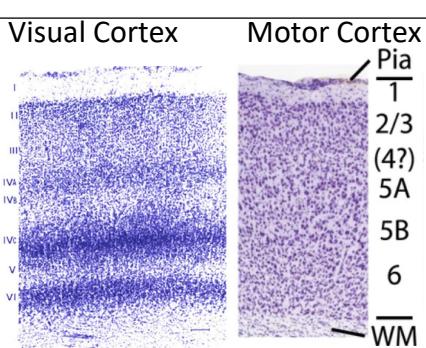
# General “learning machine”, reused widely



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



- Not fully-general learning, 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 primitives not too different from animals: more to come?

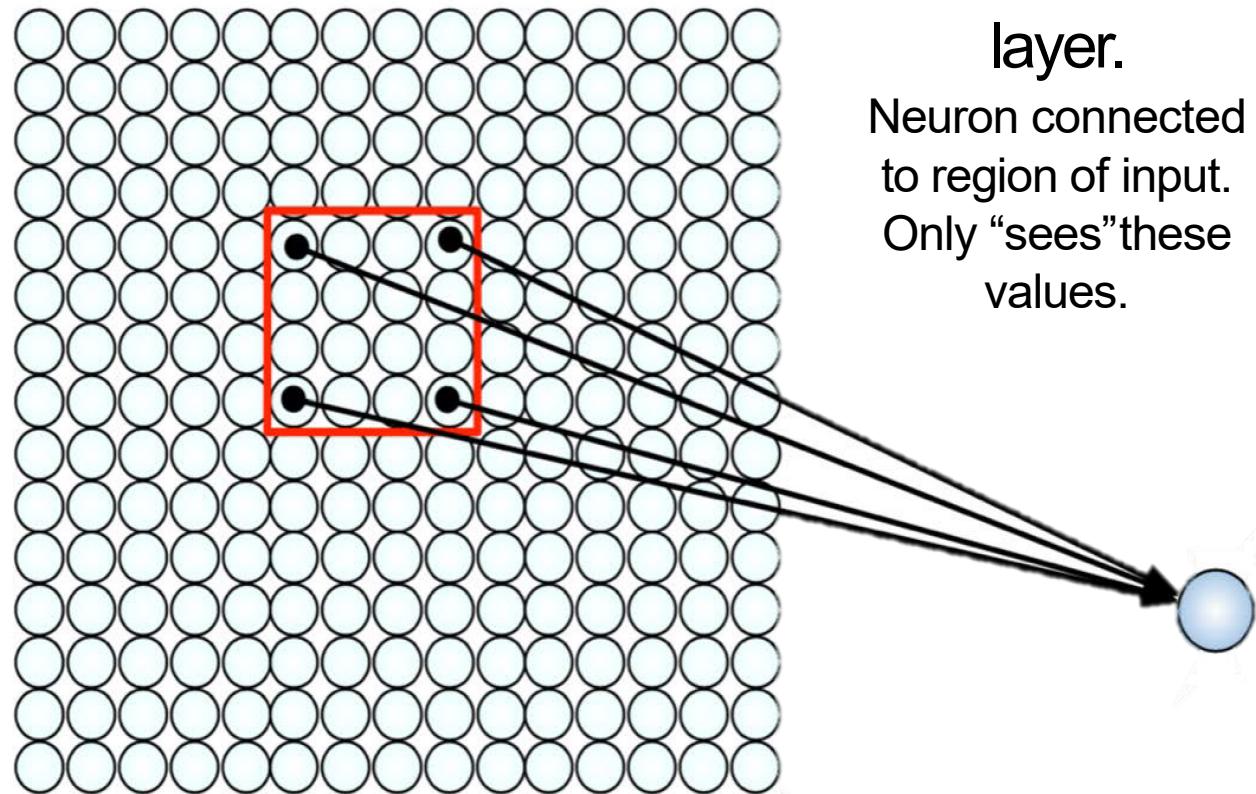


- Interchangeable circuitry
- Auditory cortex learns to ‘see’ if sent visual signals
- Injury area tasks shift to uninjured areas

Spatial structure  
for image recognition

# Using Spatial Structure

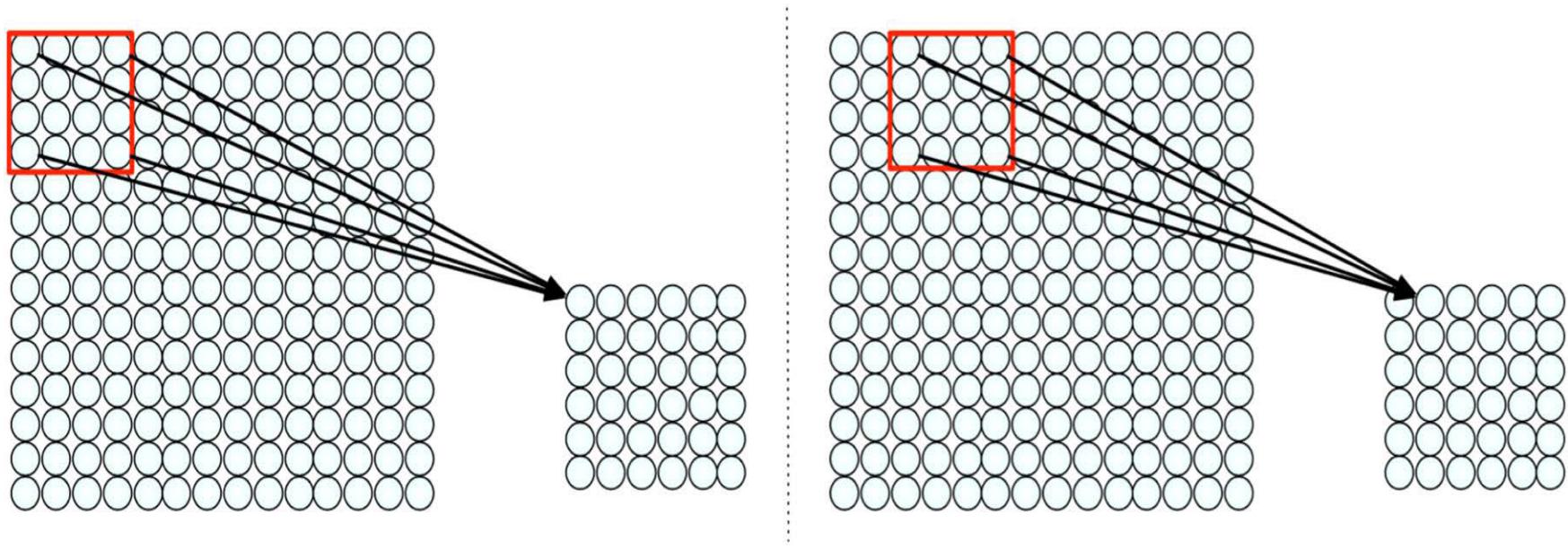
**Input:** 2D  
image.  
Array of pixel  
values



**Idea:** connect  
patches of input to  
neurons in hidden  
layer.

Neuron connected  
to region of input.  
Only “sees”these  
values.

# Using Spatial Structure

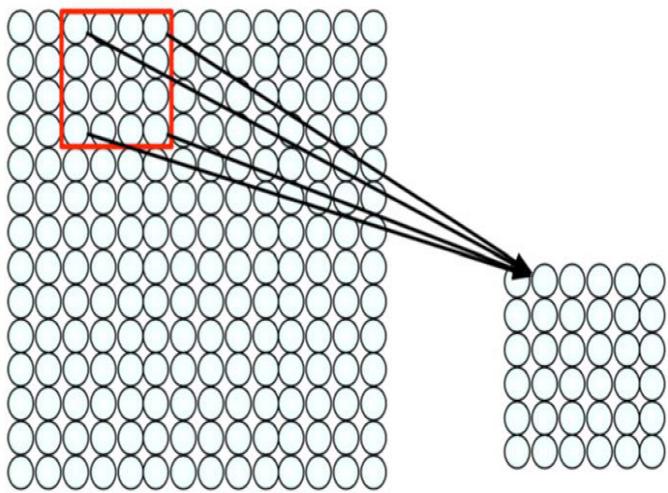


Connect patch in input layer to a single neuron in subsequent layer.

Use a sliding window to define connections.

*How can we **weight** the patch to detect particular features?*

# 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

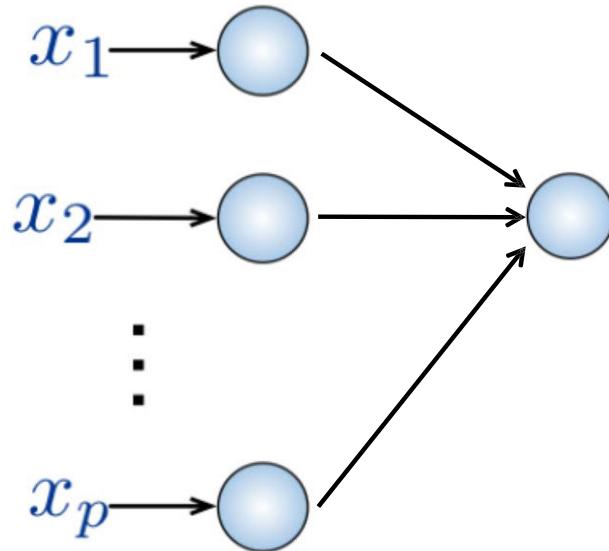
This “patchy” operation is **convolution**

- 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

# 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

# High Level Feature Detection

Let's identify key features in each image category



Nose, Eyes, Mouth

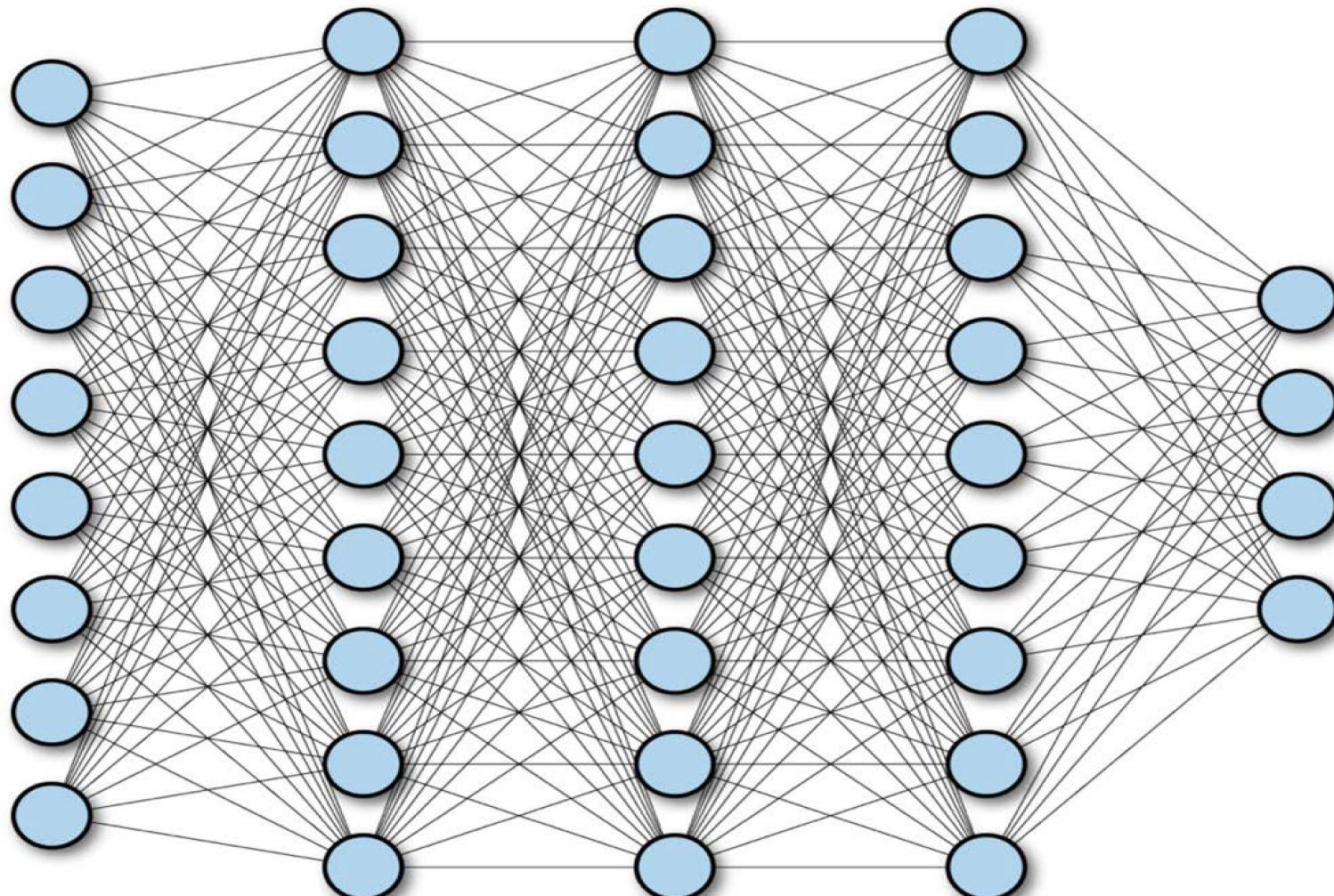


Wheels, License Plate,  
Headlights



Door, Windows, Steps

# Fully Connected Neural Network



# Convolutions and filters

# 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

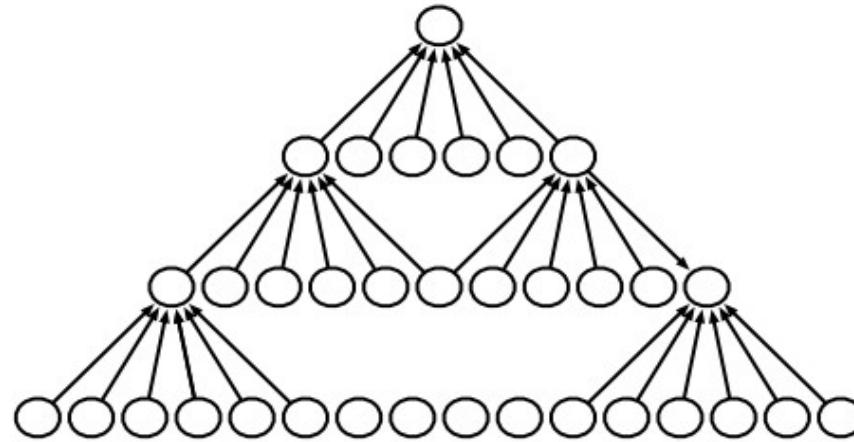
There are three approaches to edge cases in convolution

- **Valid convolution:** output only when entire kernel is contained in input (shrinks output)
- **Same convolution:** zero pad input so output is same size as input dimensions
- **Full convolution:** zero pad input so output is produced whenever an output value contains at least one input value (expands output)

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

# Zero Padding Controls Output Size

Valid -  
no padding



Same -  
add zeros

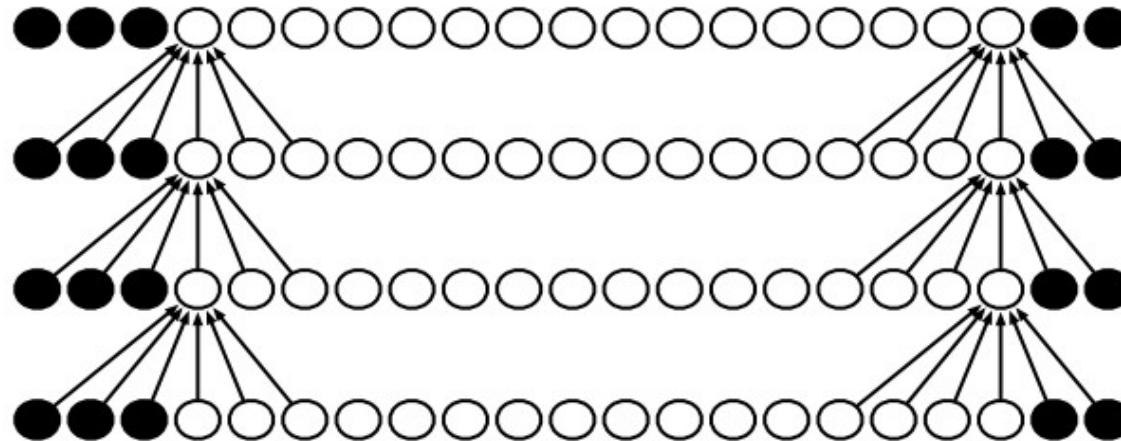
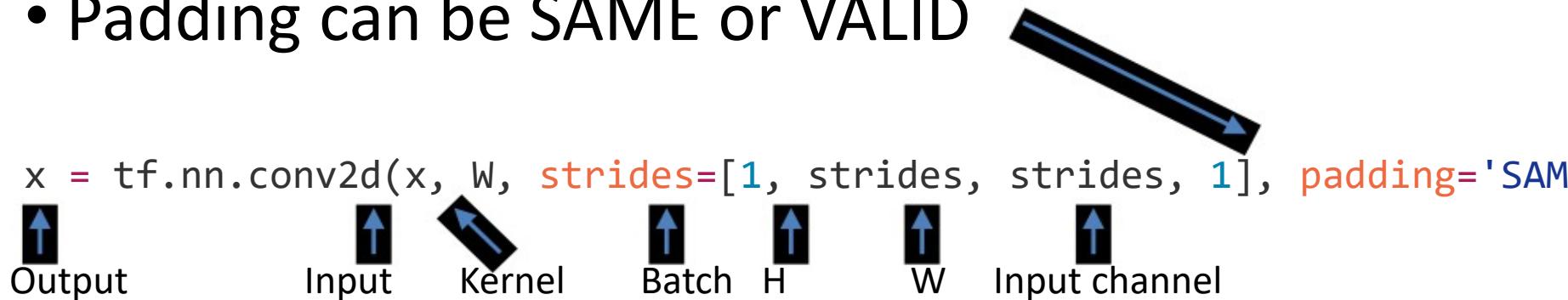


Figure 9.13

# 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



# Producing Feature Maps



Original



Sharpen



Edge Detect



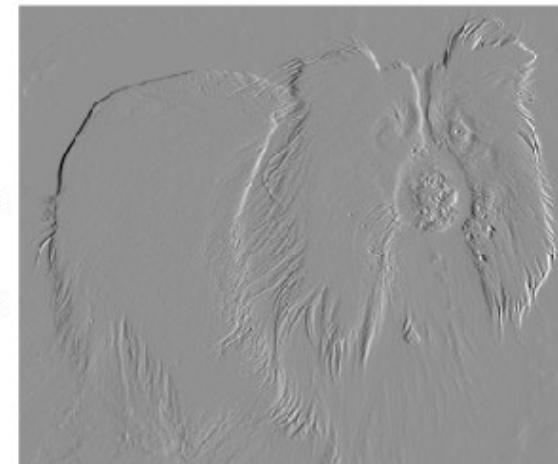
“Strong” Edge  
Detect

# 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)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

# 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



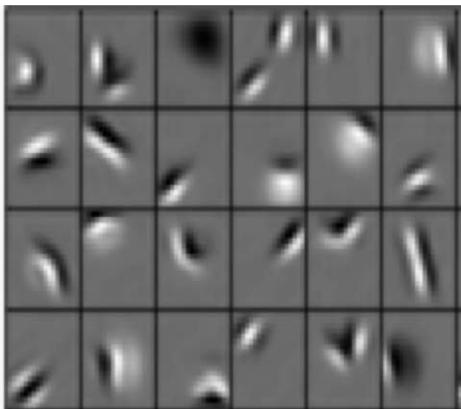
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	1	1	1	-1	1	-1	-1
-1	-1	-1	-1	1	-1	1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

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.

# Learning Visual Features *de novo*

**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

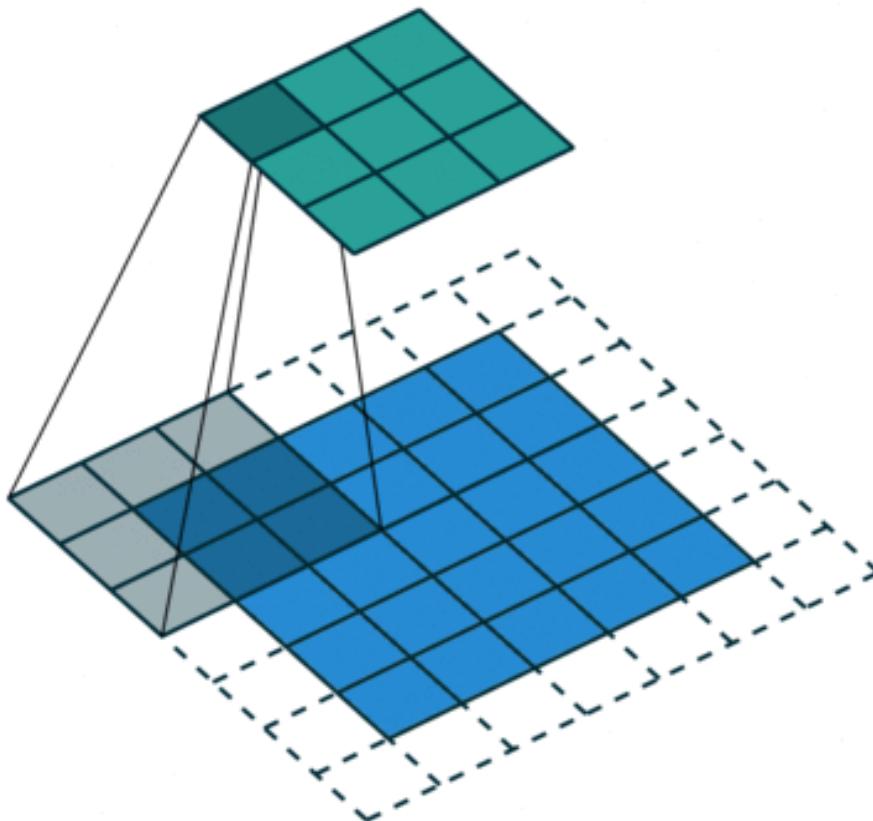


Facial structure

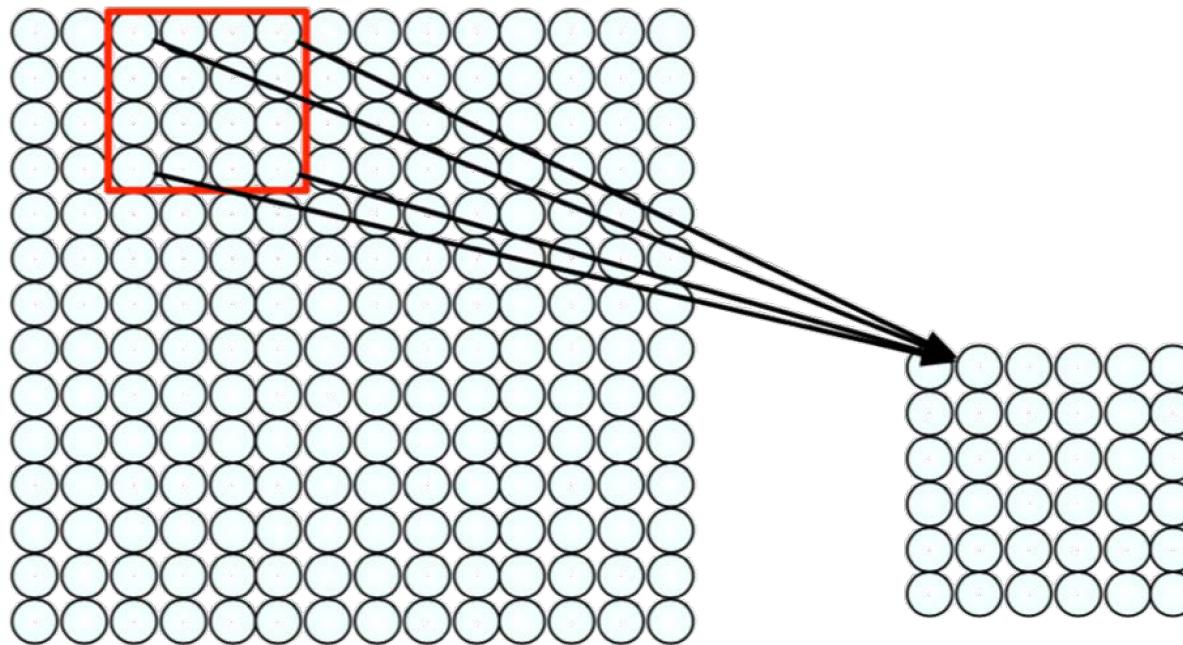
## Key idea: re-use parameters

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

# 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

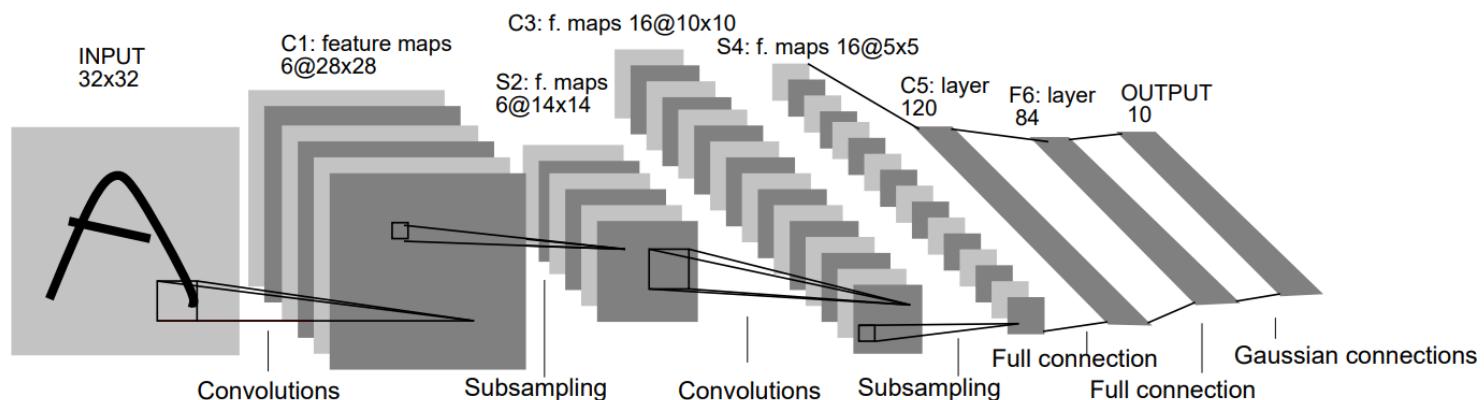
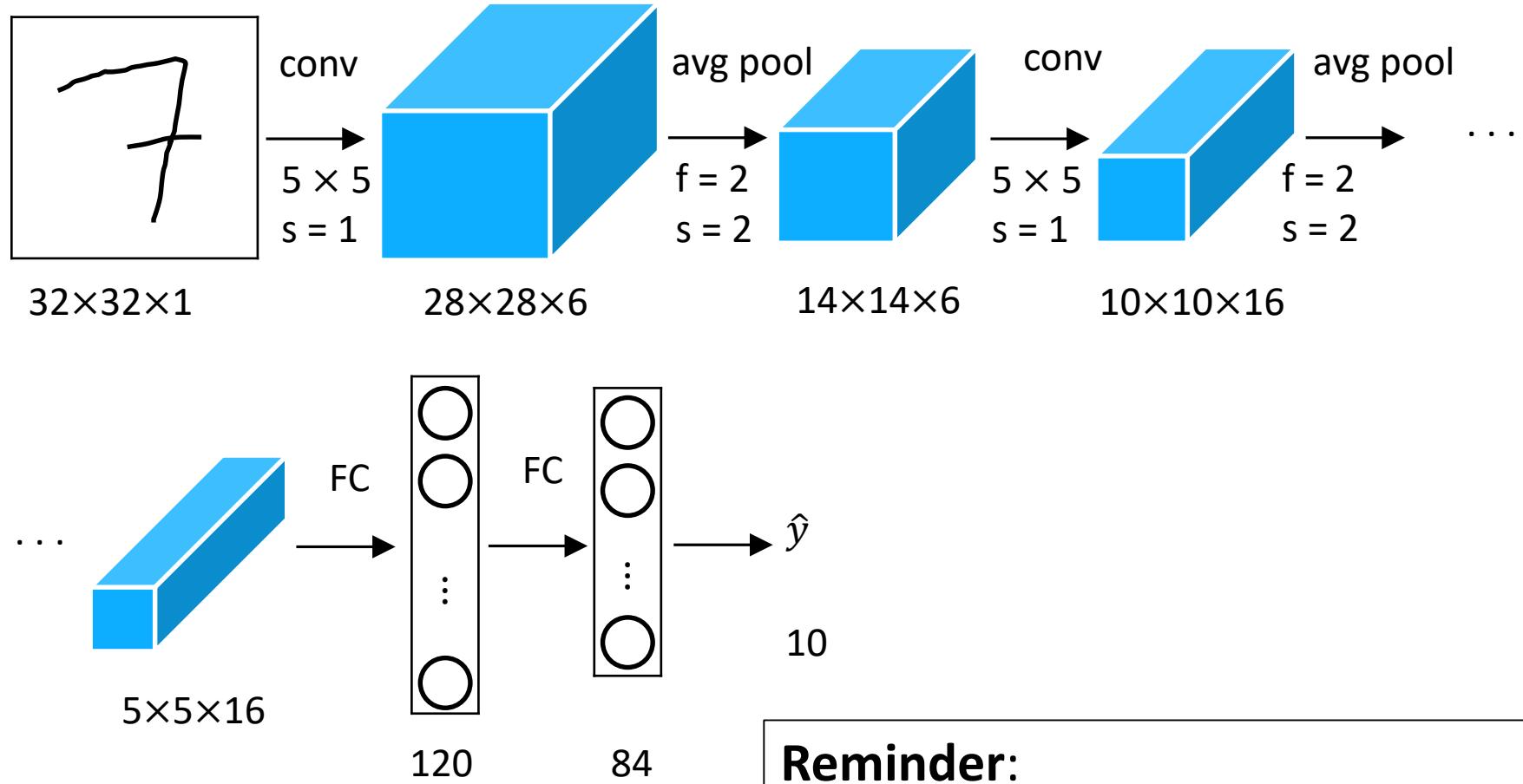


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

# LeNet-5



**Reminder:**

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



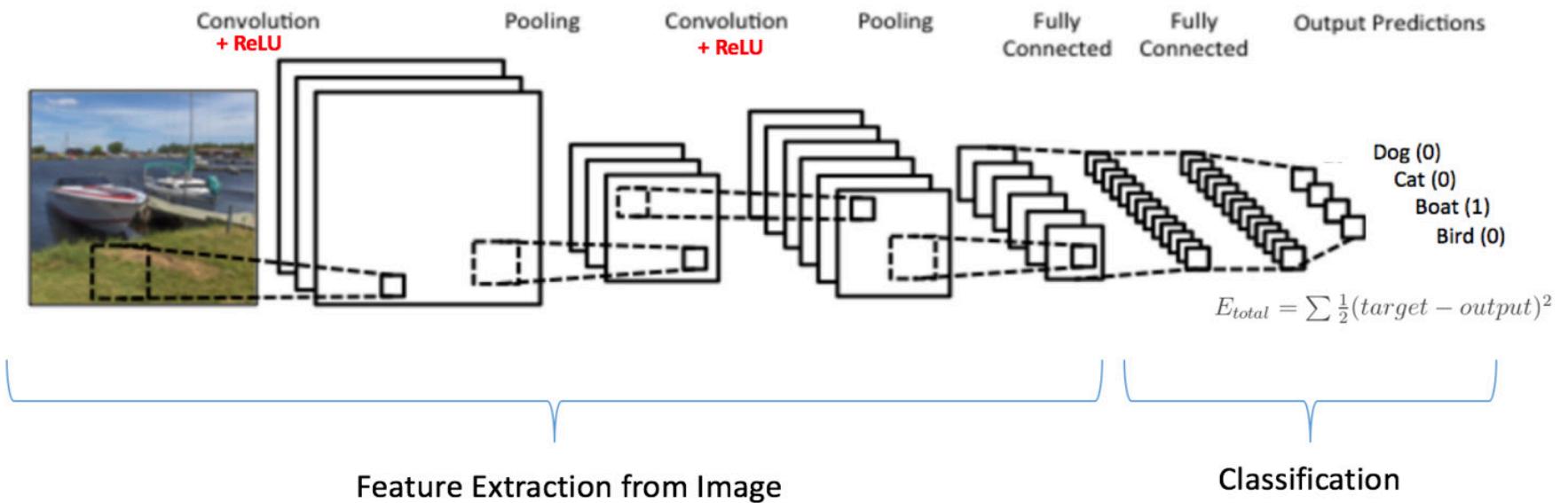
# LeNet-5

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

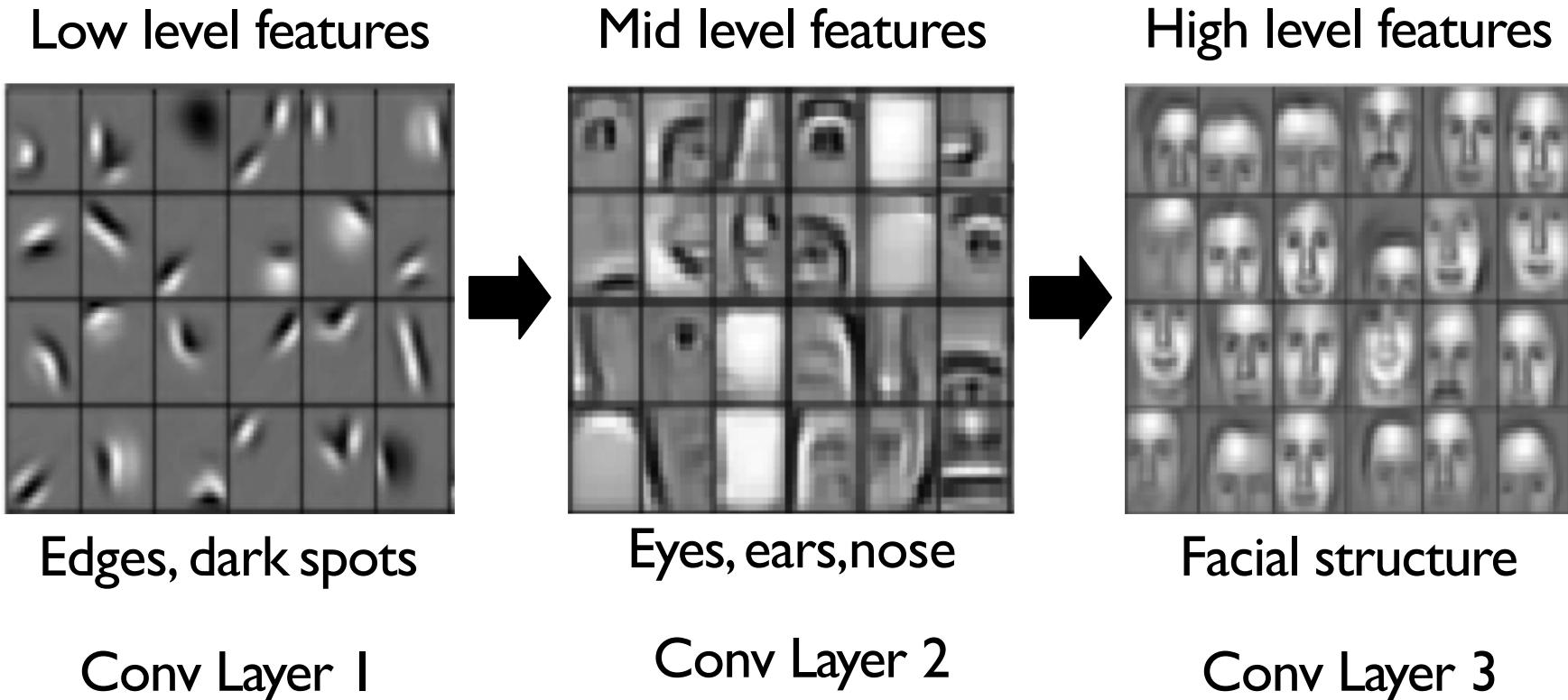
- Different filters look at different channels
- Sigmoid and Tanh nonlinearity

# Convolutional Neural Networks (CNNs)

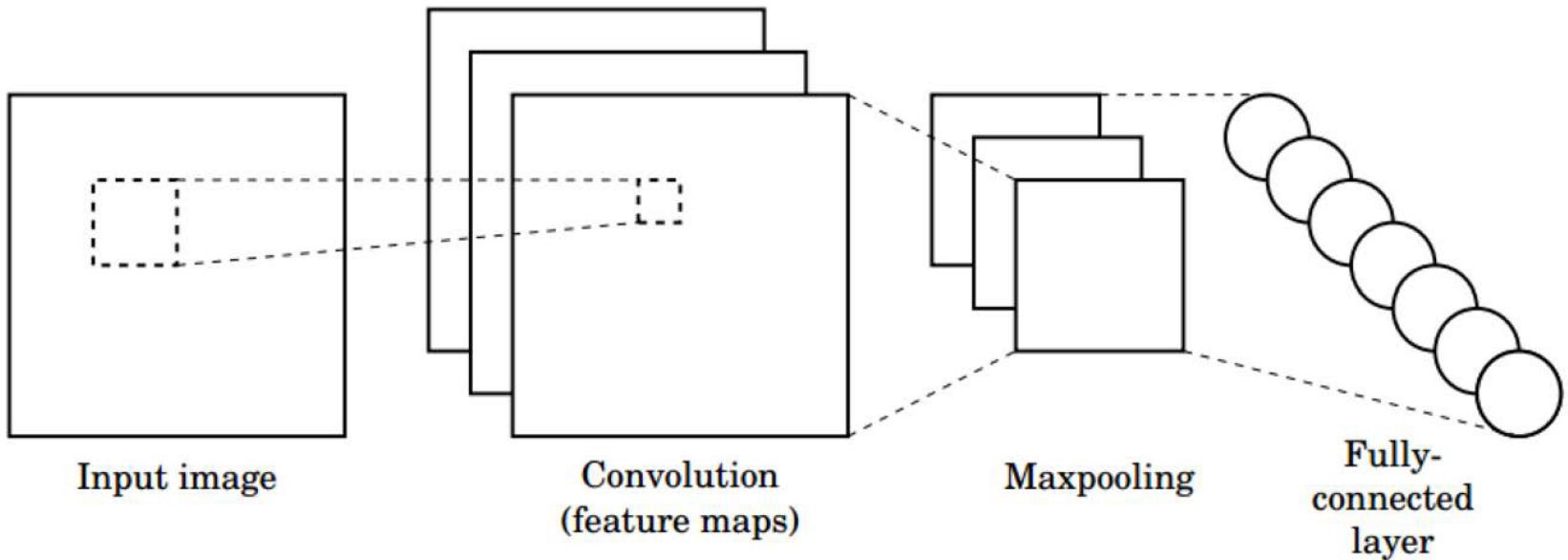
# An image classification CNN



# Representation Learning in Deep CNNs



# CNNs for Classification



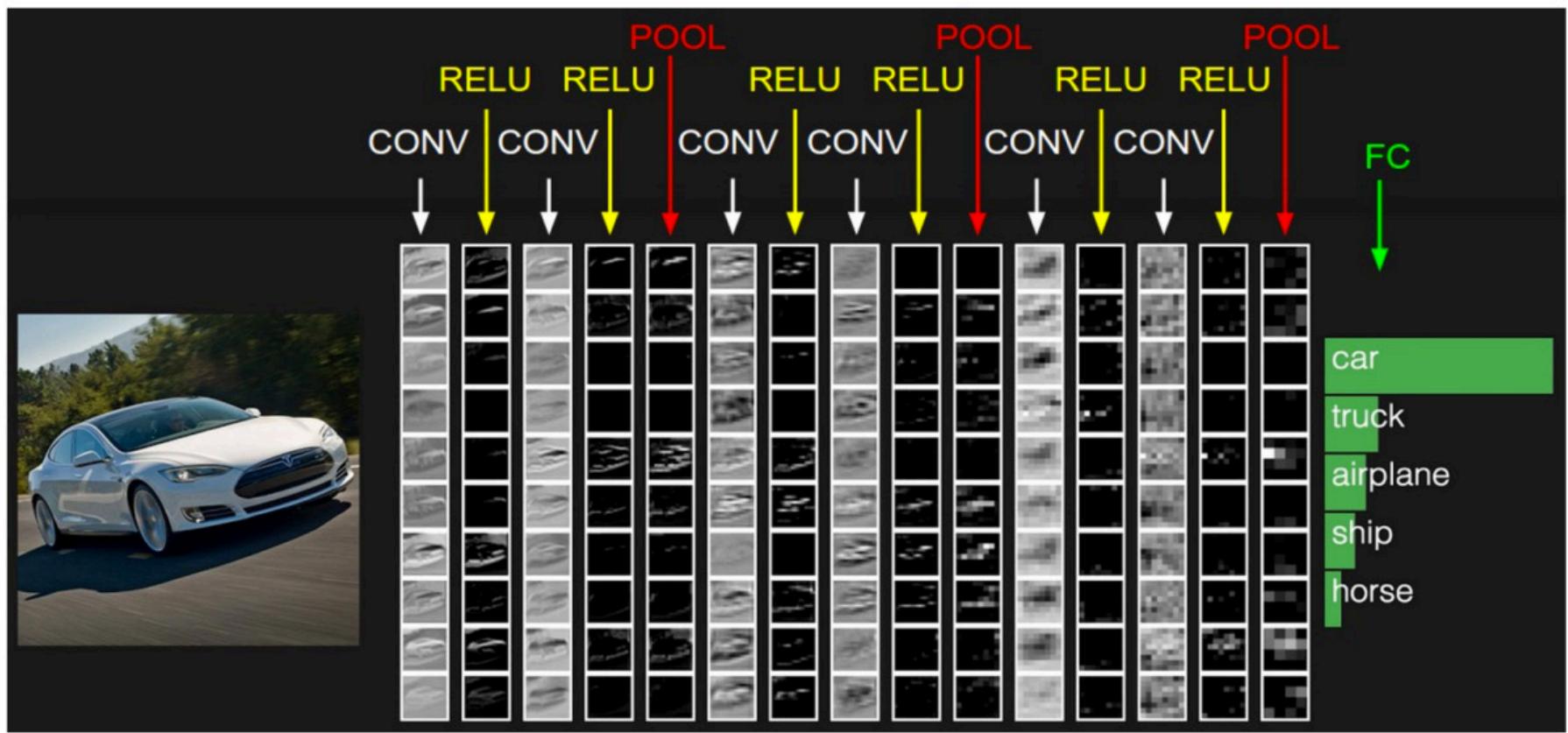
- 1. Convolution:** Apply filters to generate feature maps.
- 2. Non-linearity:** Often ReLU.
- 3. Pooling:** Downsampling operation on each feature map.

**Train model with image data.**

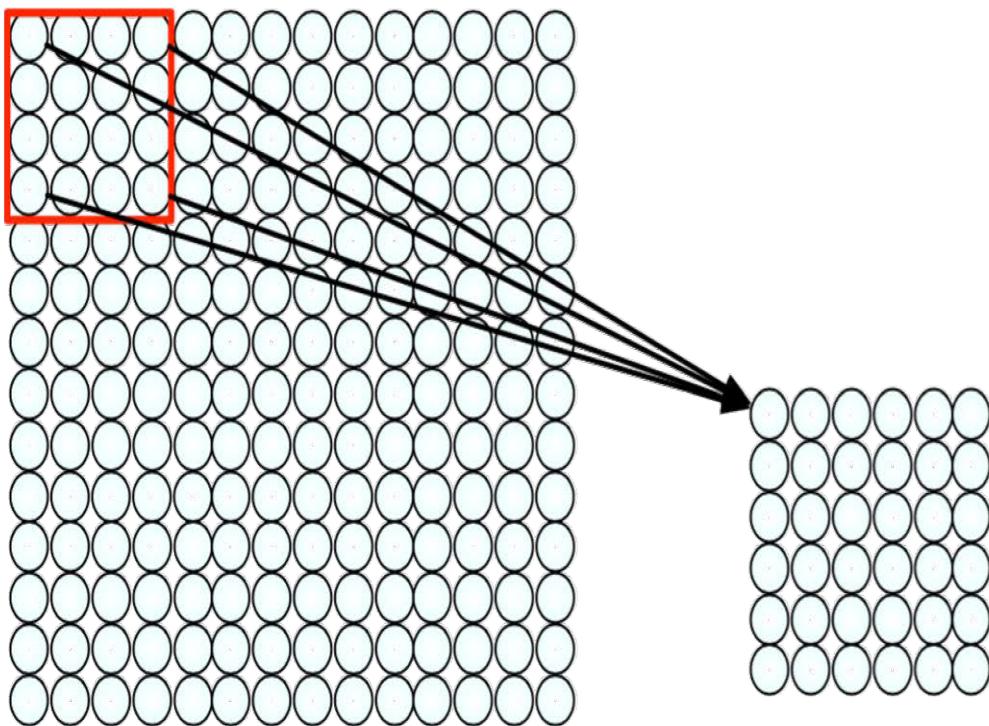
**Learn weights of filters in convolutional layers.**

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

# Example – Six convolutional layers



# Convolutional Layers: Local Connectivity

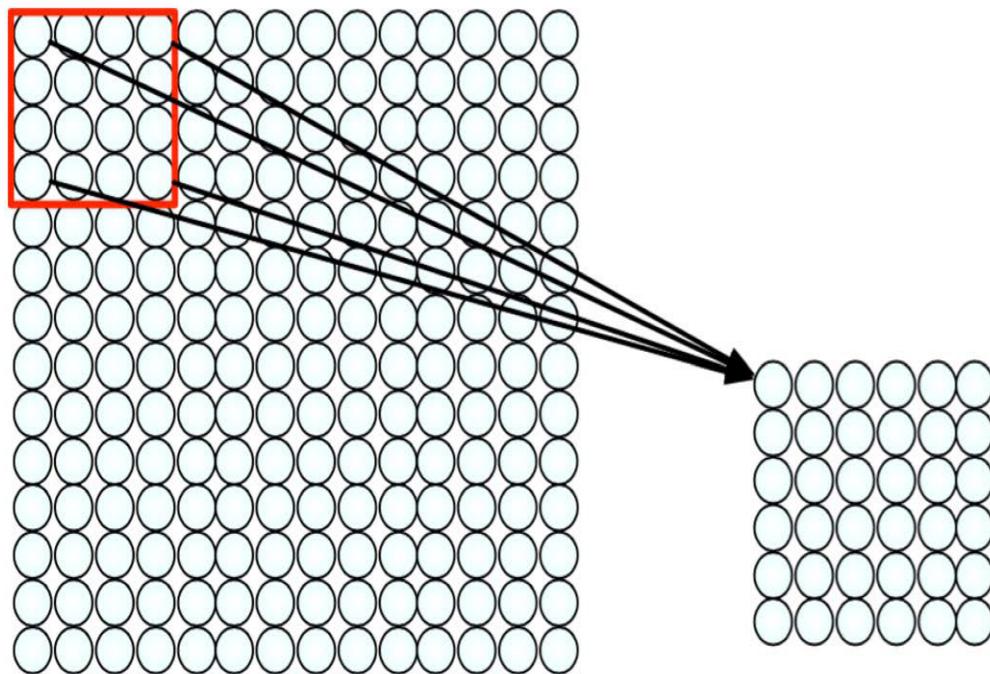


 `tf.keras.layers.Conv2D`

For a neuron in  
hidden layer:

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

# Convolutional Layers: Local Connectivity



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



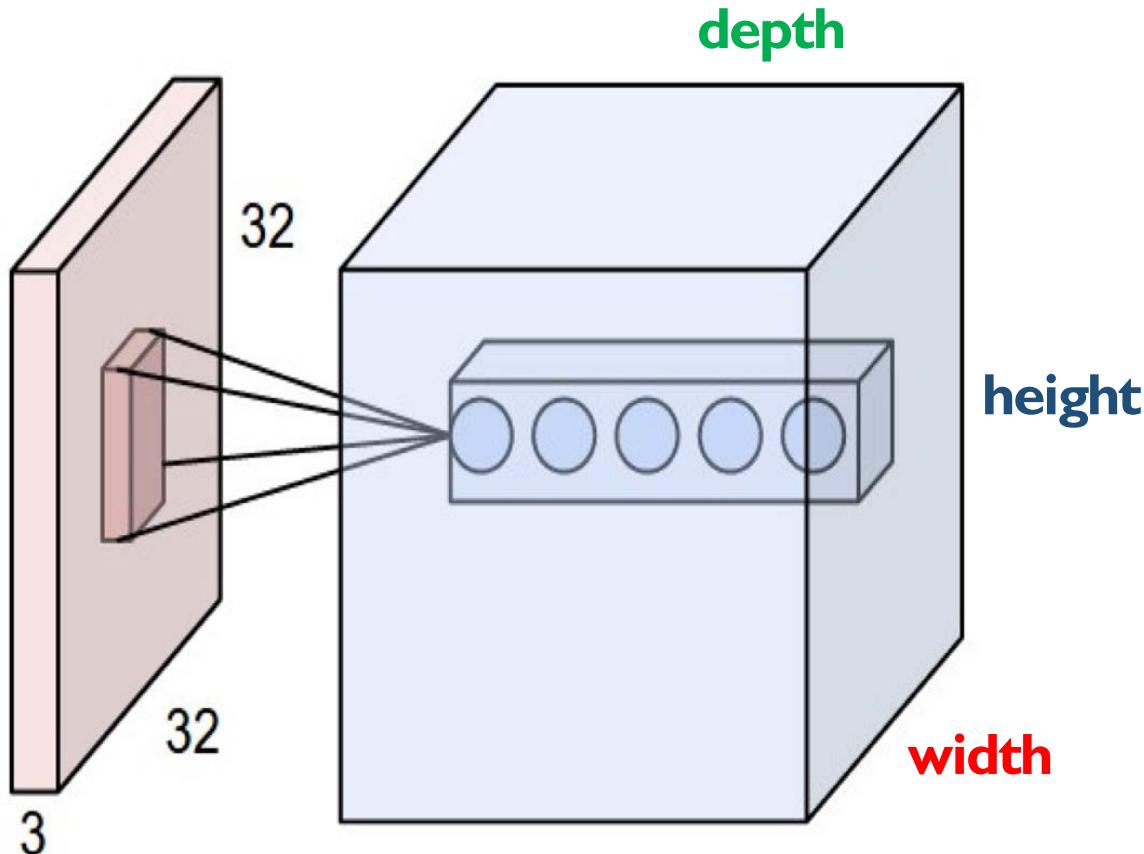
`tf.keras.layers.Conv2D`

For a neuron in  
hidden layer:

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

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

# 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:**

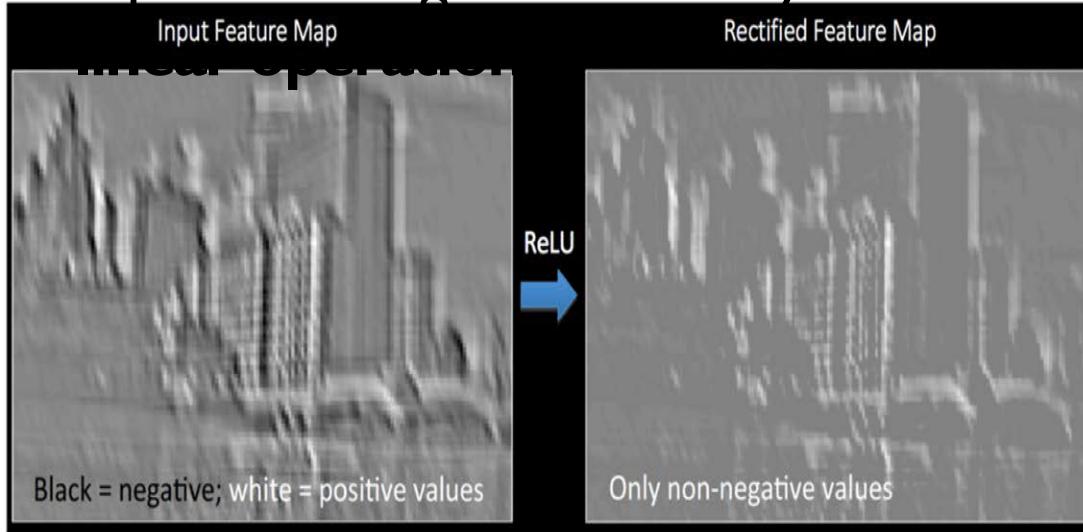
Locations in input image that a node is path connected to



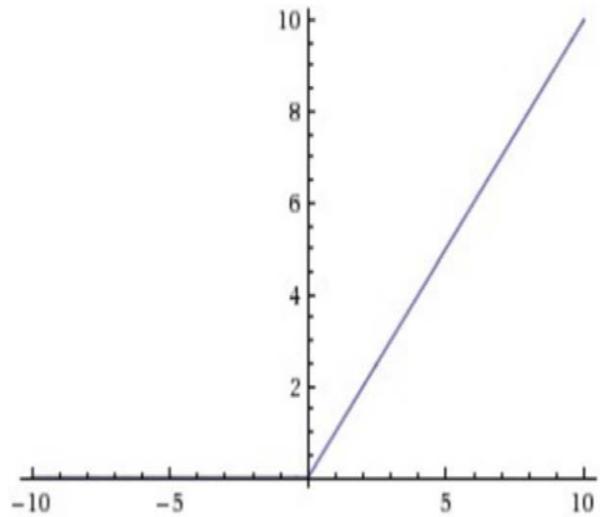
```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

# 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-linearity**



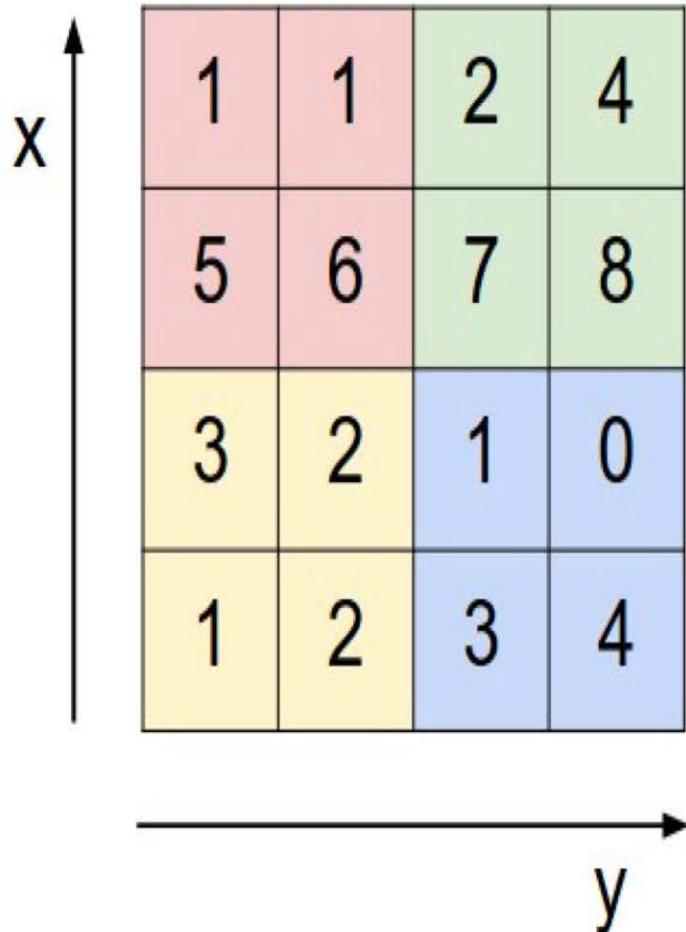
Rectified Linear Unit  
(ReLU)



$$\text{ReLU}(x) = \max(0, x)$$

 `tf.keras.layers.ReLU`

# Pooling



max pool with 2x2 filters  
and stride 2



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

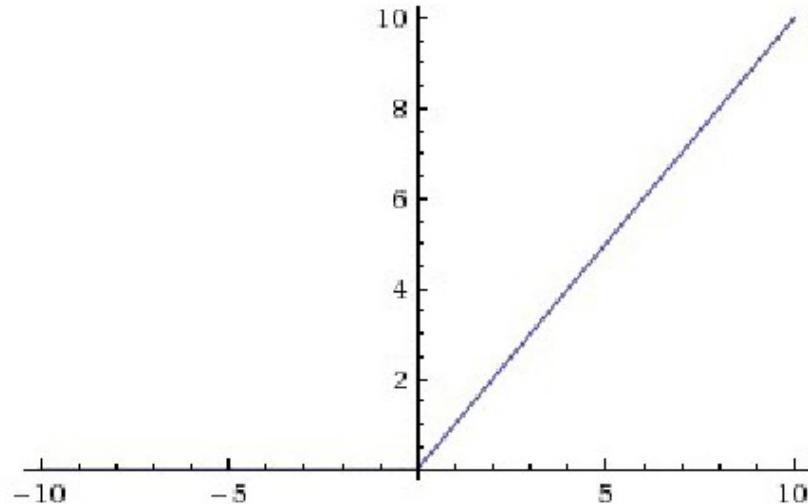


- 1) Reduced dimensionality
- 2) Spatial invariance

Max Pooling, average pooling

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)
```

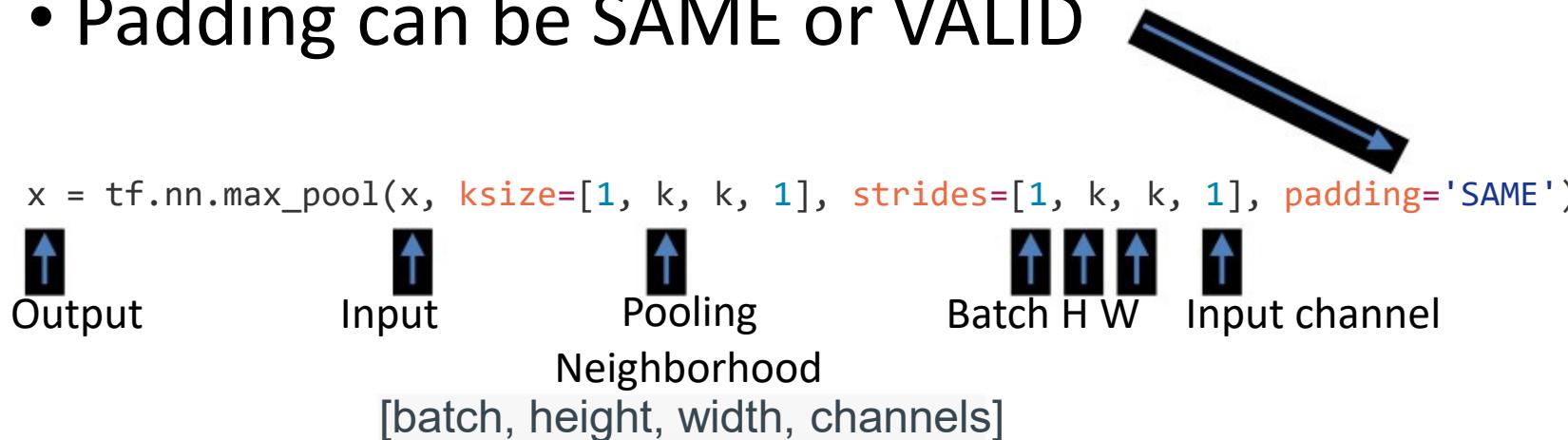


$$f(x) = \max(0, x)$$

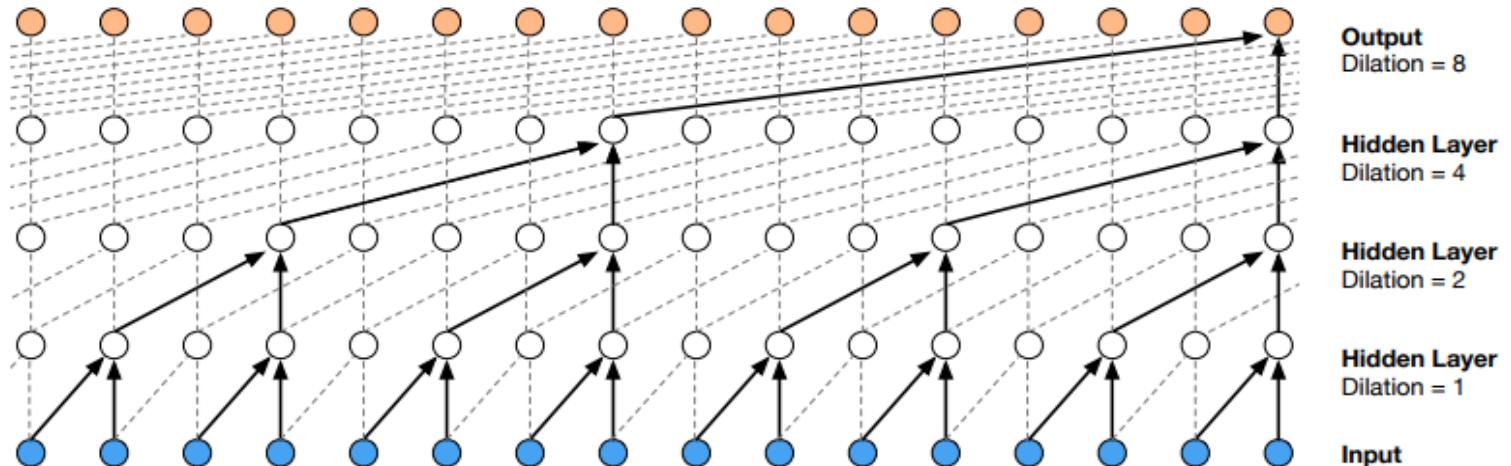
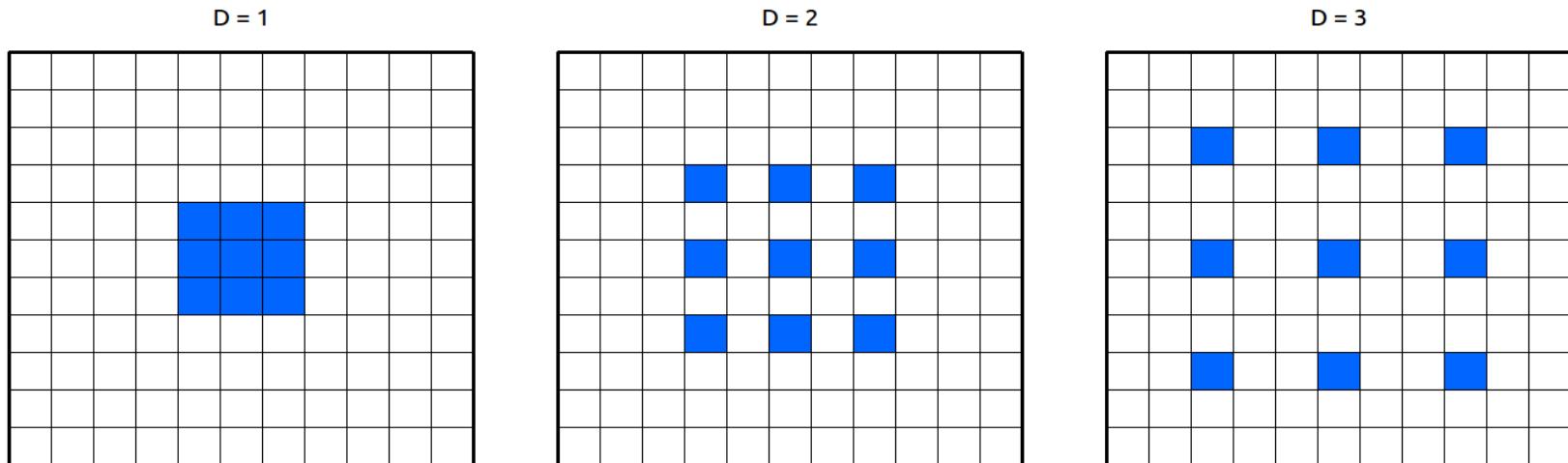
When will we backpropagate through this?  
Once it “dies” what happens to it?

# Pooling reduces dimensionality by giving up spatial location

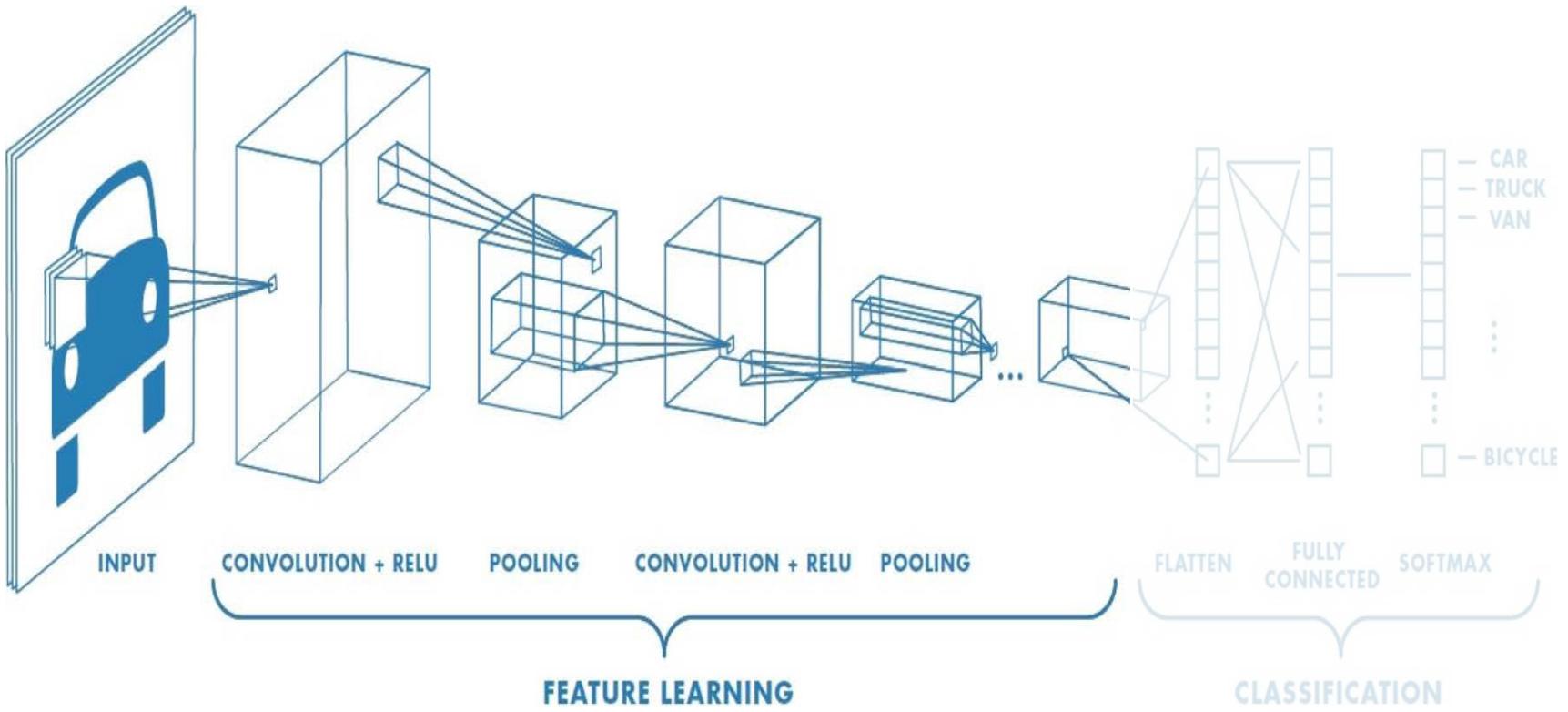
- **max pooling** reports the maximum output within a defined neighborhood
- Padding can be SAME or VALID



# Dilated Convolution

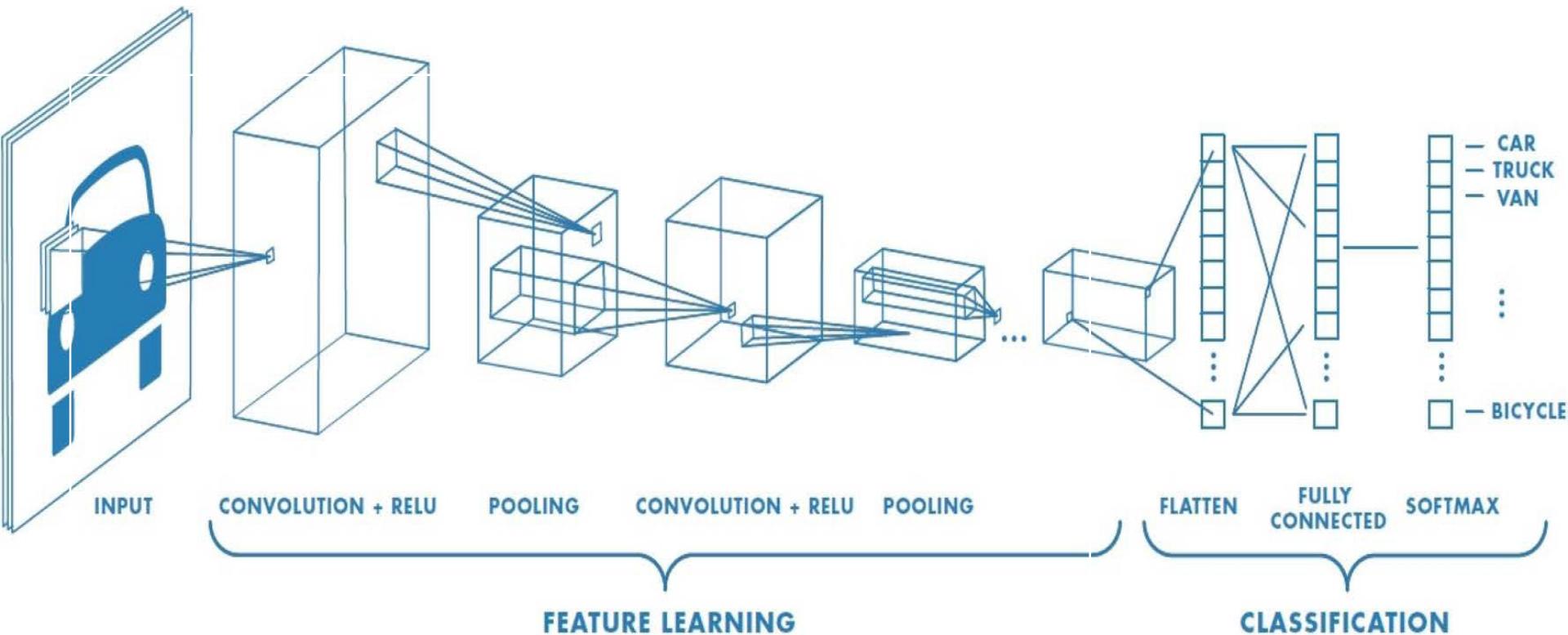


# CNNs for Classification: Feature Learning



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

# Putting it all together

```
import tensorflow as tf

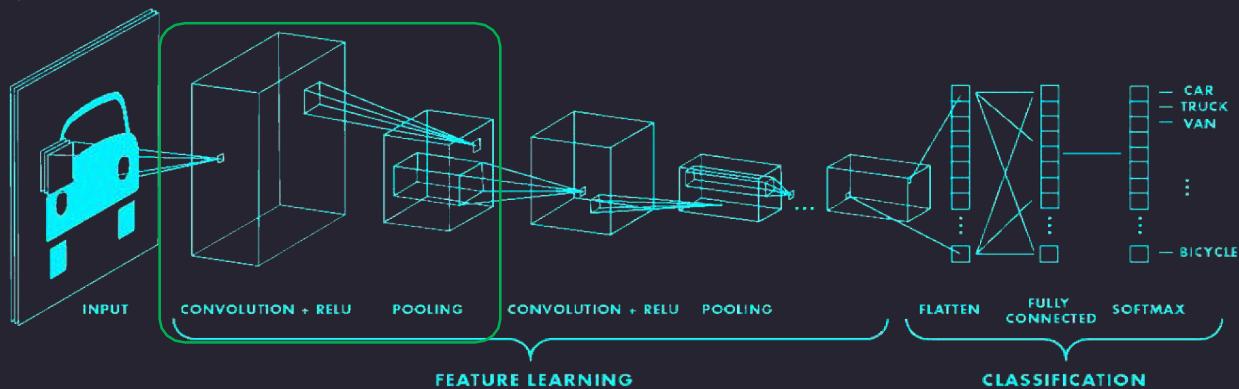
def generate_model():
    model = tf.keras.Sequential([
        # first convolutional layer
        tf.keras.layers.Conv2D(32, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    ])

    # 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
```



Real-world feature invariance  
is hard

# How can computers recognize objects?



# 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

# Feature invariance to perturbation is hard

Viewpoint variation



Scale variation



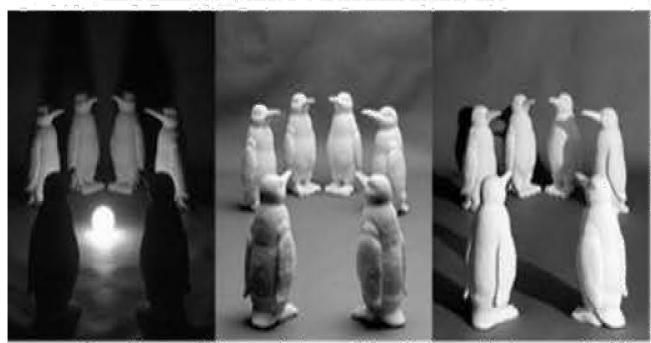
Deformation



Occlusion



Illumination conditions



Background clutter



Intra-class variation



Next-generation models  
explode # of parameters

# 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

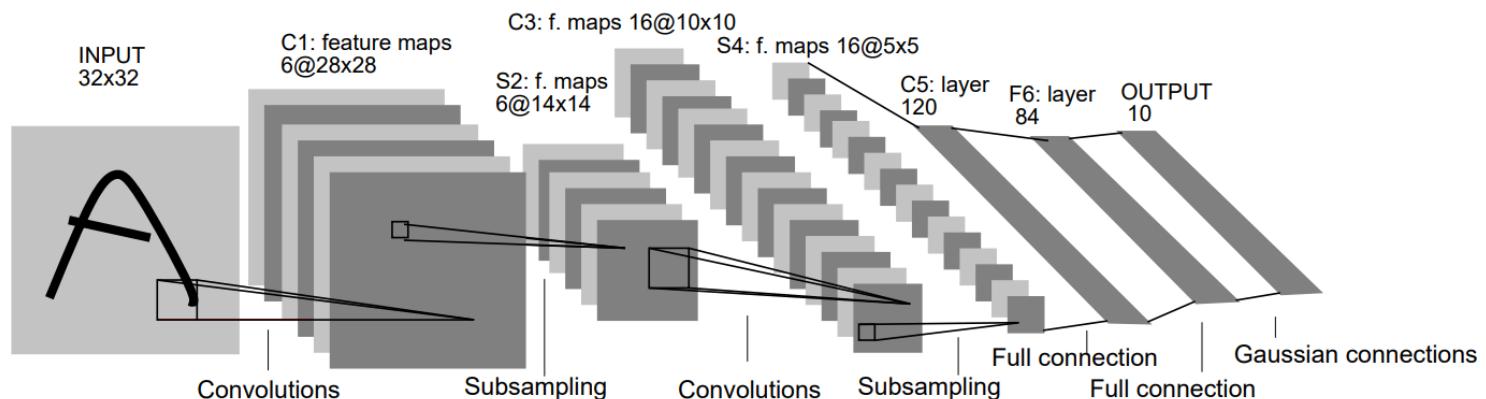
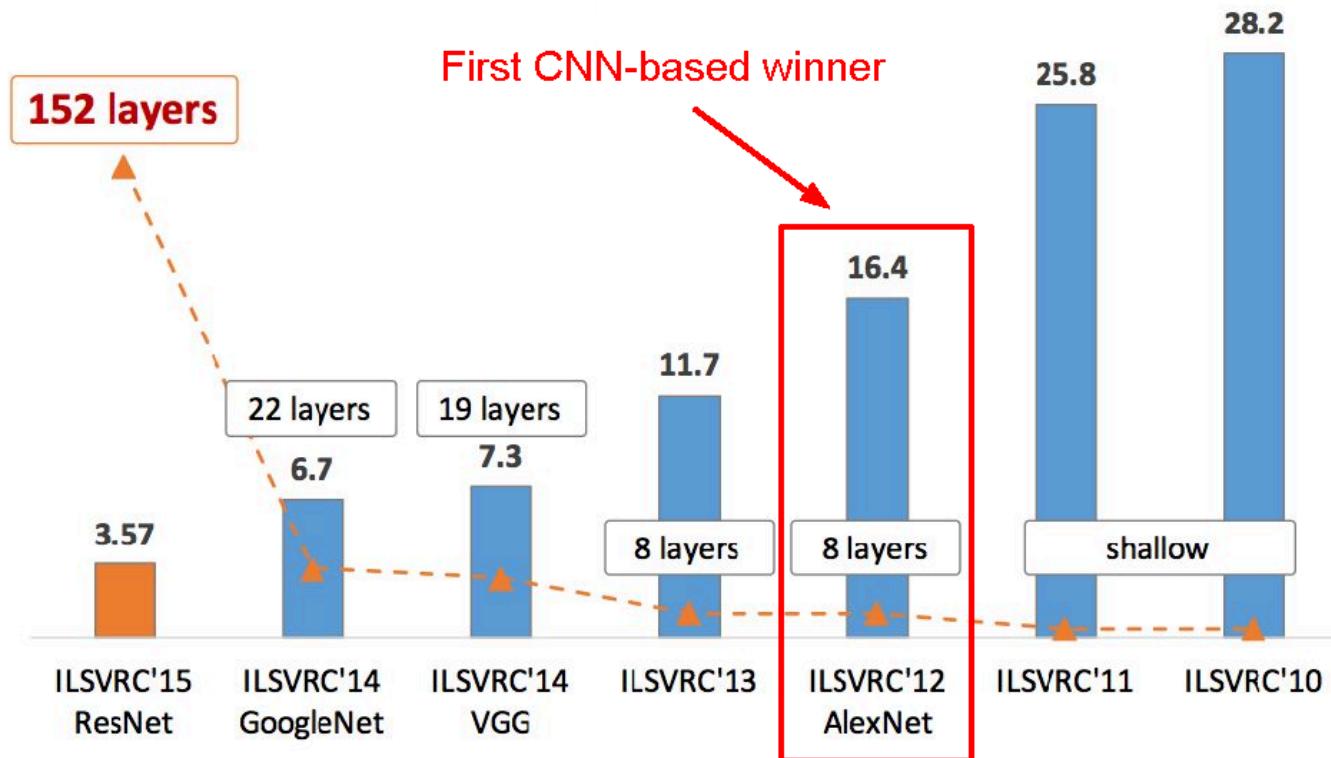


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



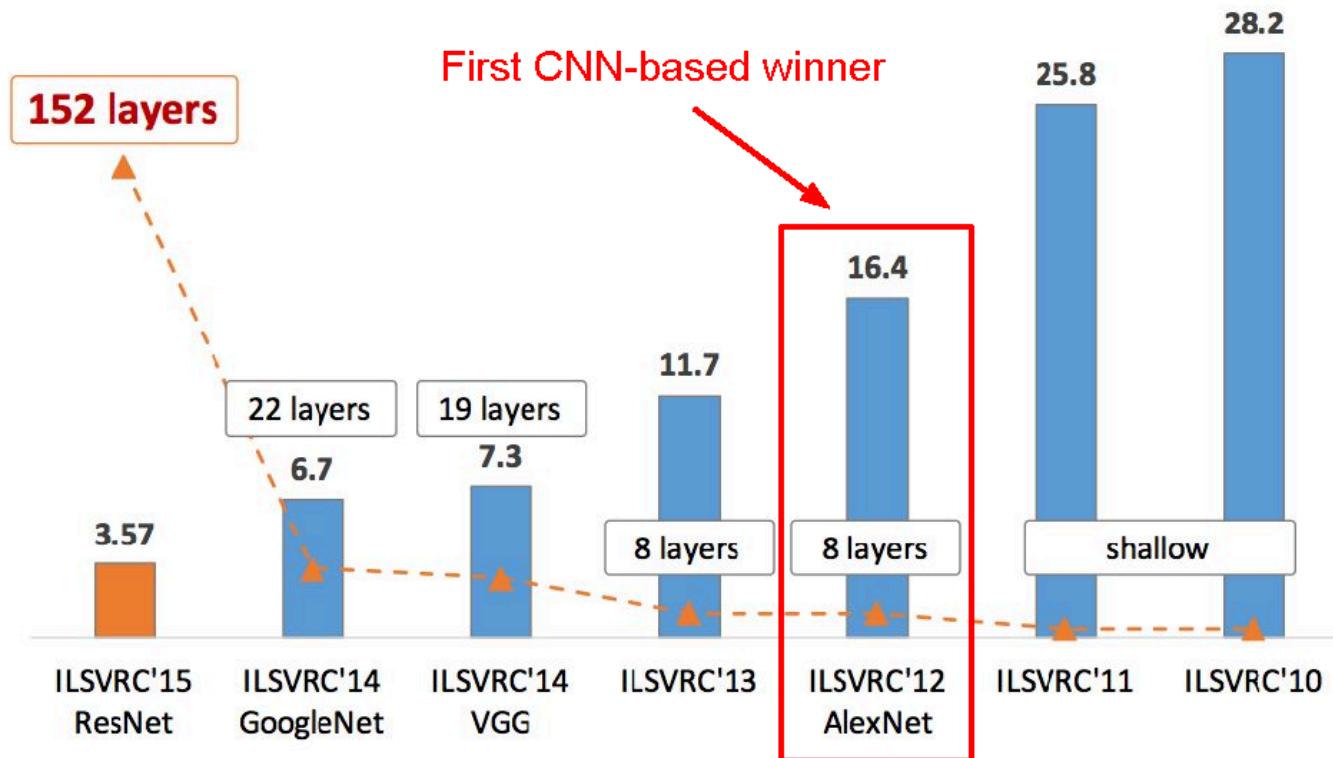
# AlexNet

- *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

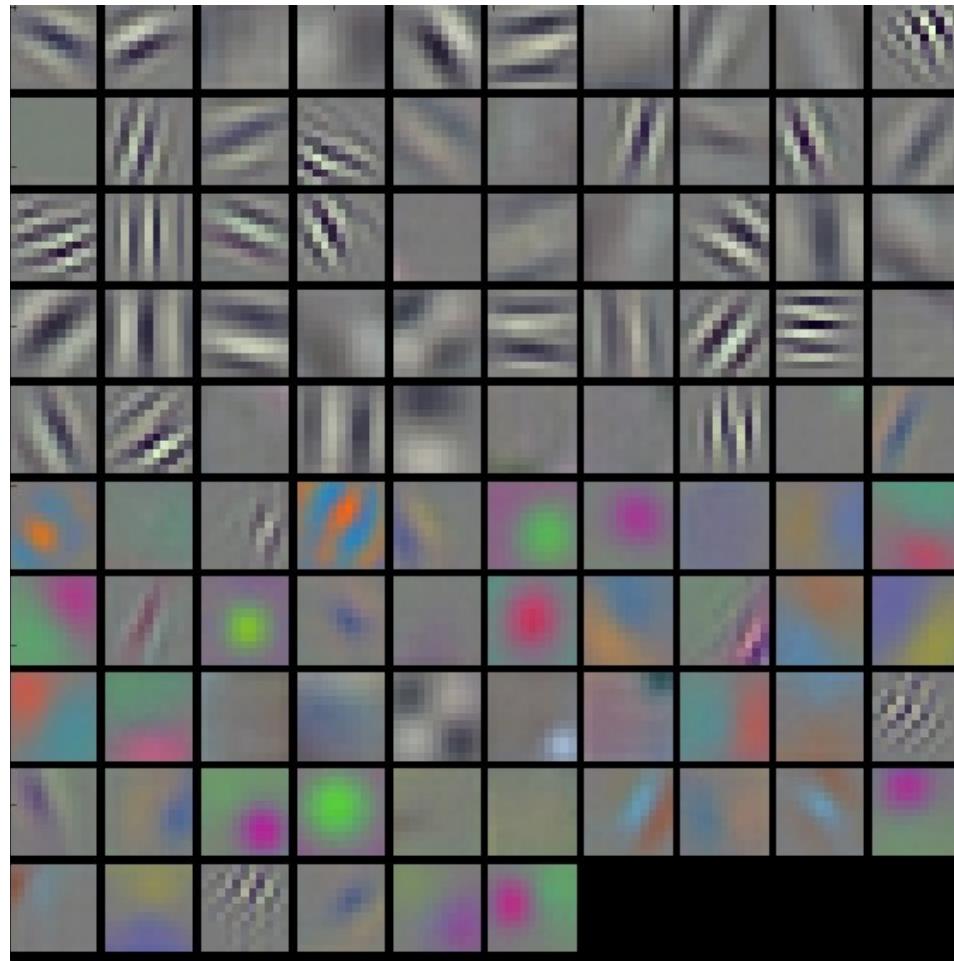
FC7

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

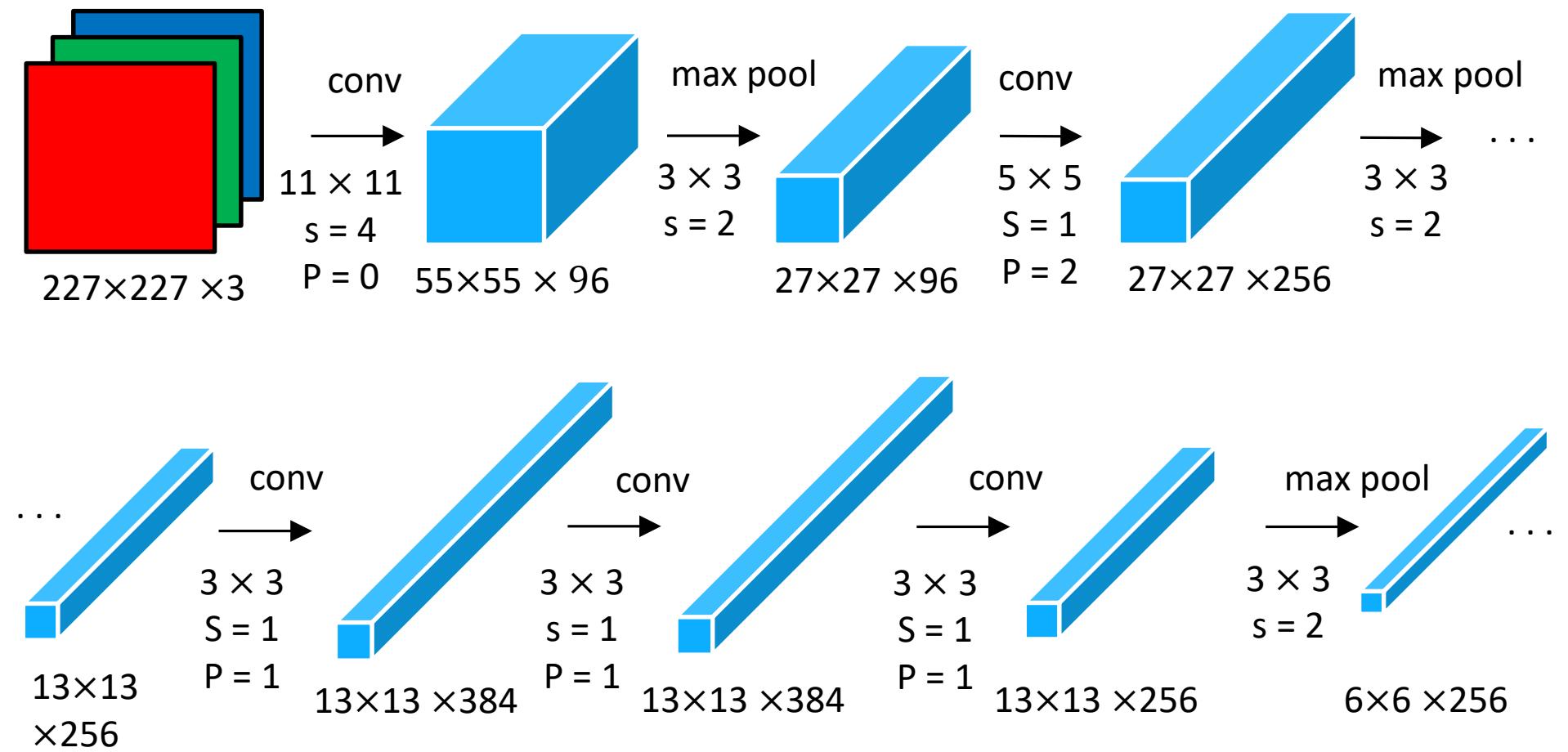
FC7

FC8

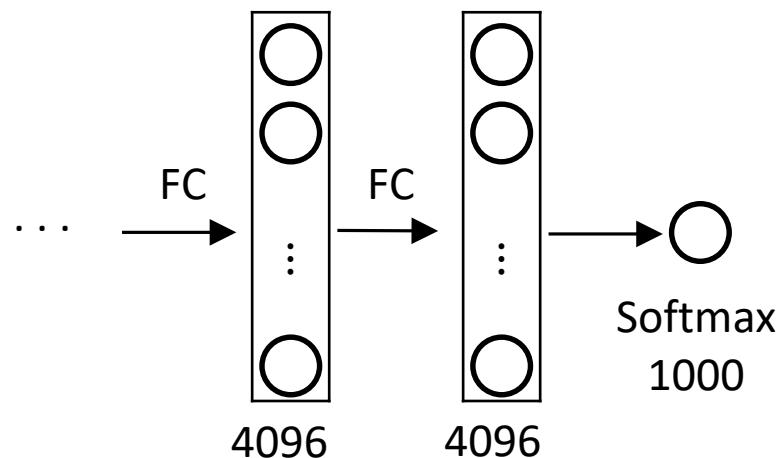
# AlexNet

- 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

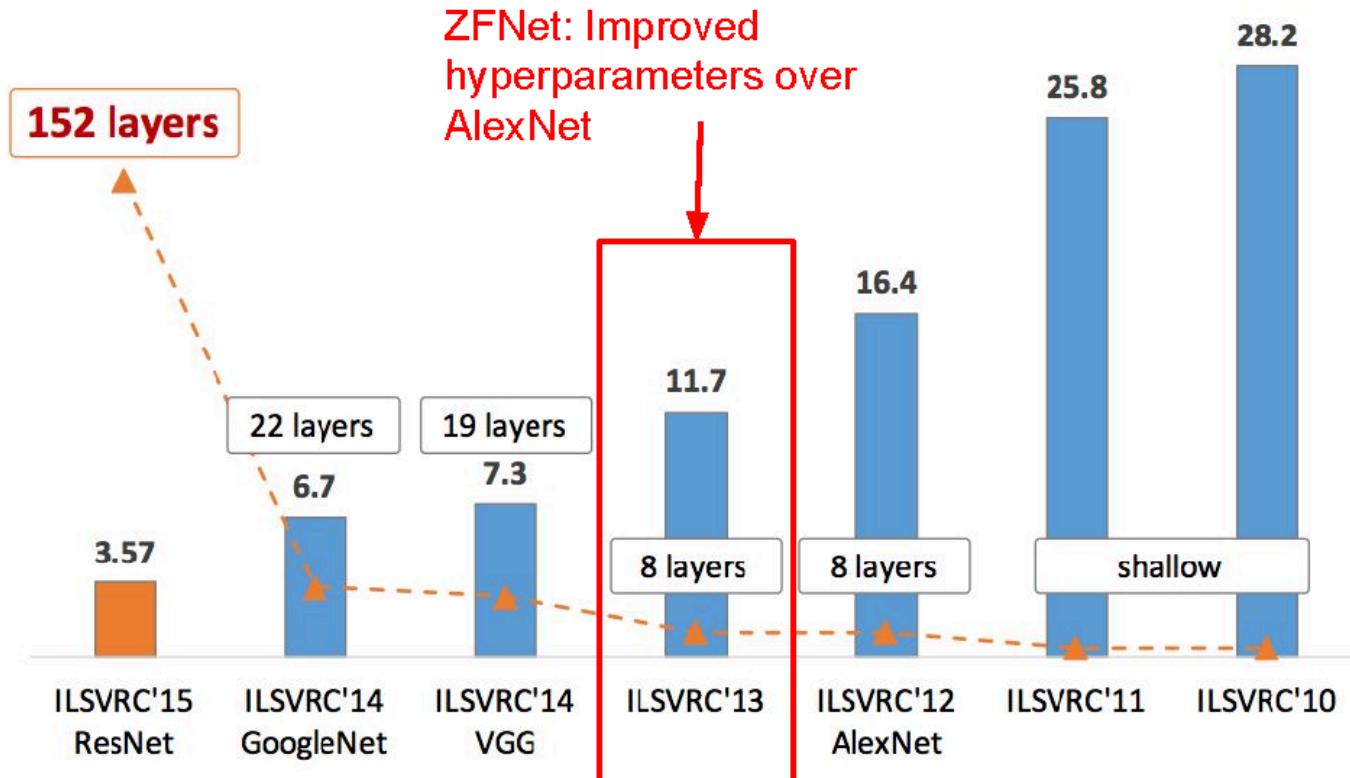
- Trained on GTX 580 GPU with only 3 GB of memory.
- Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.
- CONV1, CONV2, CONV4, CONV5:  
Connections only with feature maps on same GPU.
- CONV3, FC6, FC7, FC8:  
Connections with all feature maps in preceding layer,  
communication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



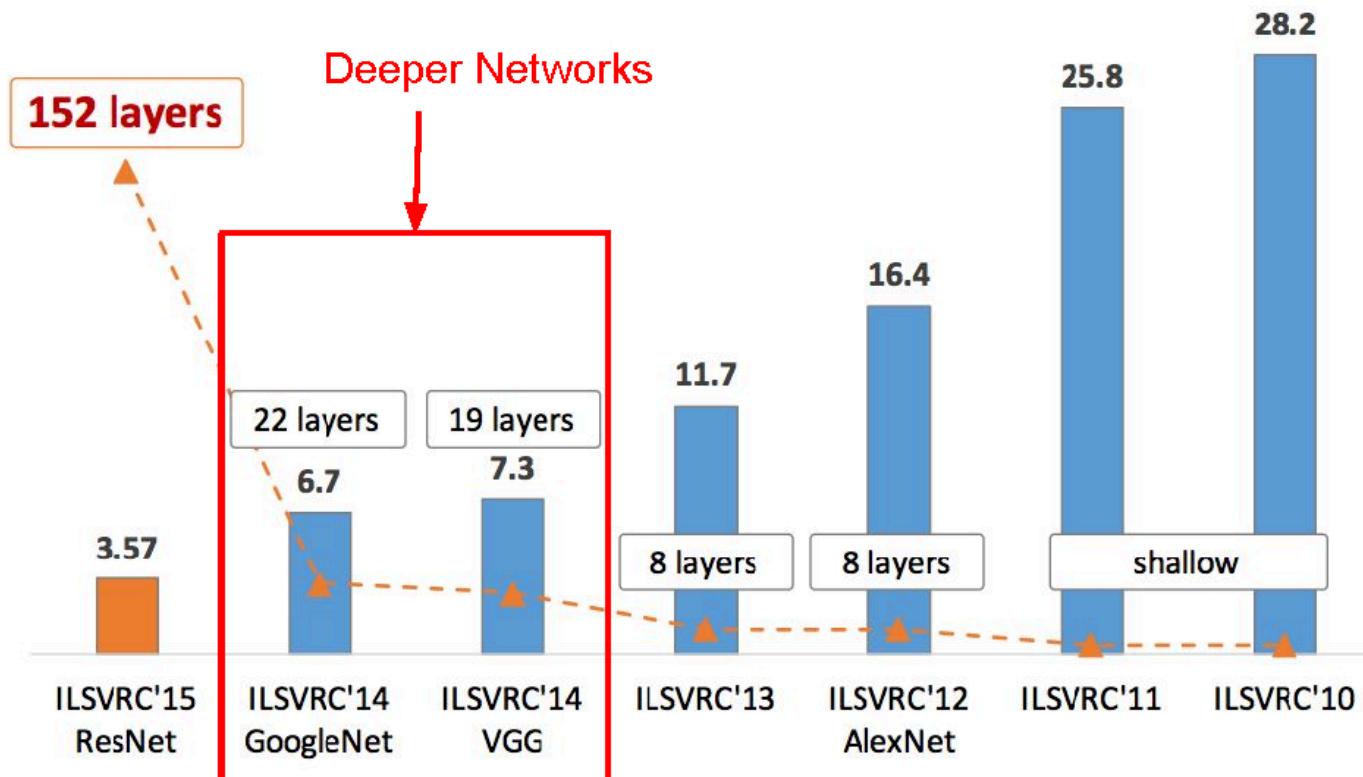
# AlexNet

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

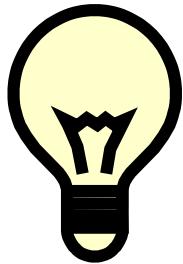
TOTAL params: 138M parameters

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		Simonyan and Zisserman, 2014]

# 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.**



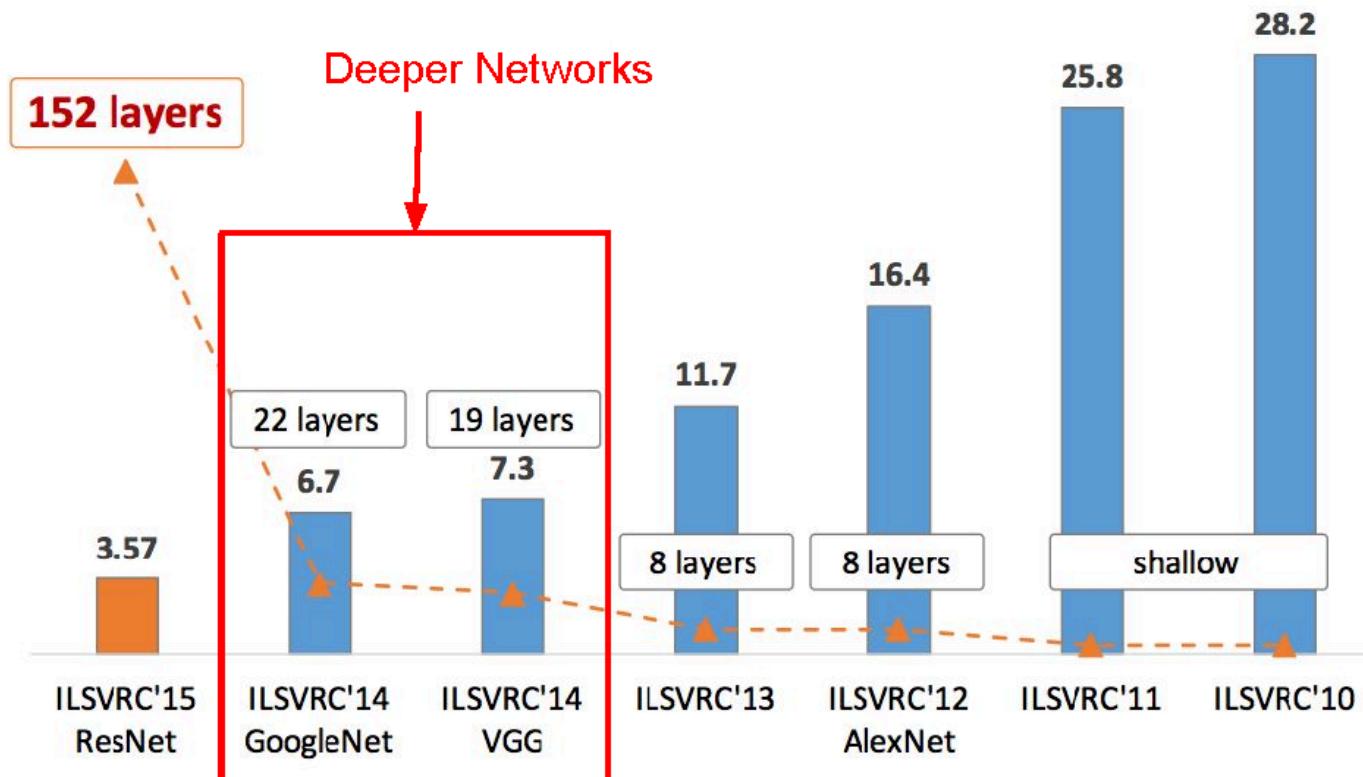
# VGGNet

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.

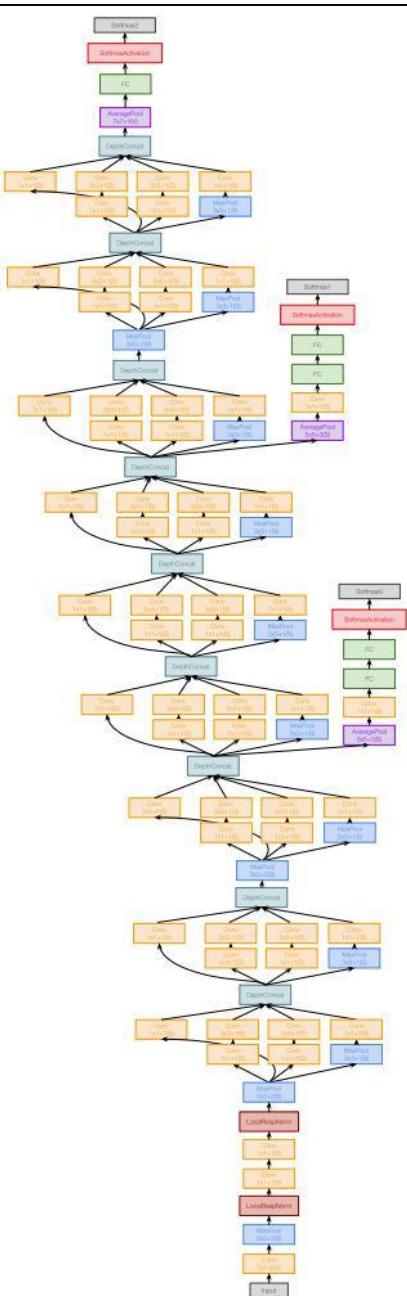
# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# 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

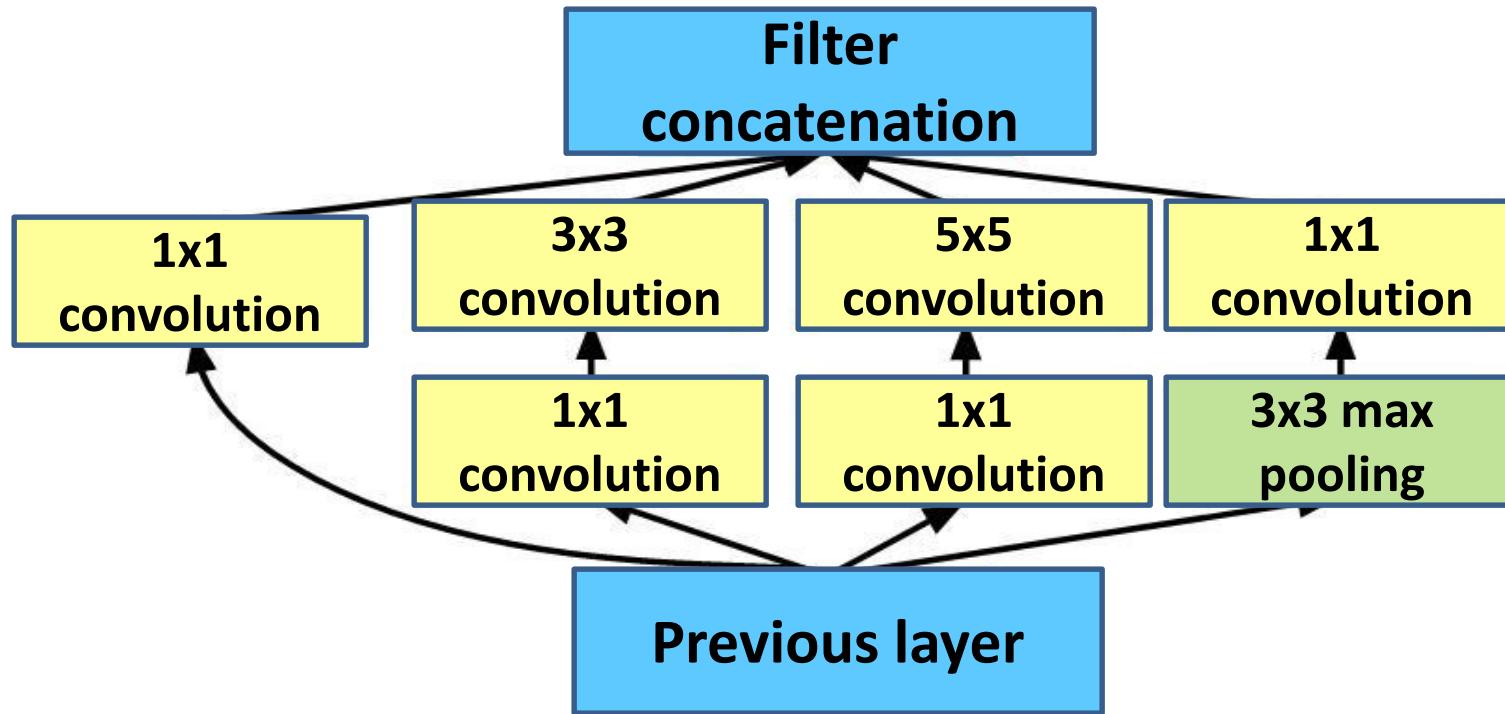
# GoogleNet



- 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

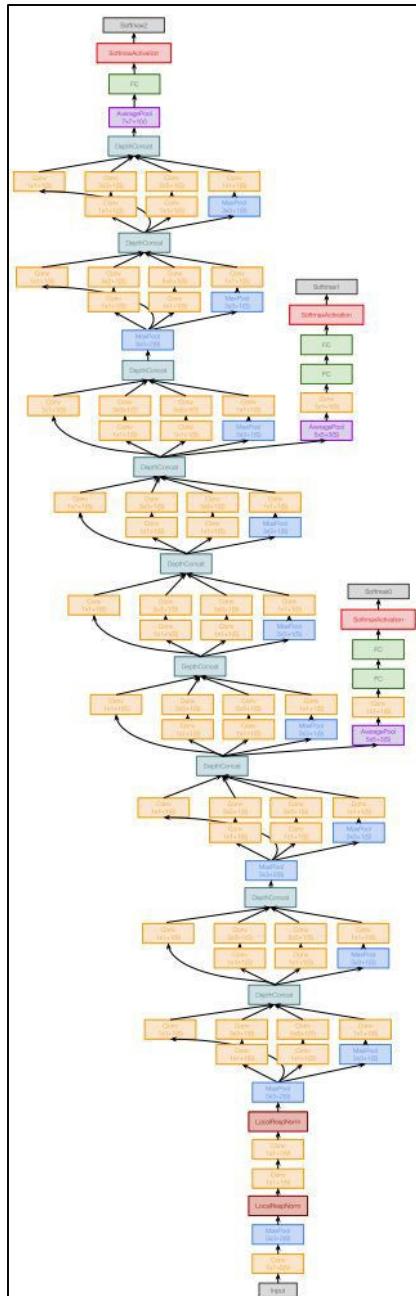
“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other

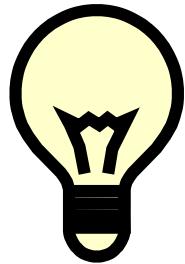


# GoogleNet

## 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)

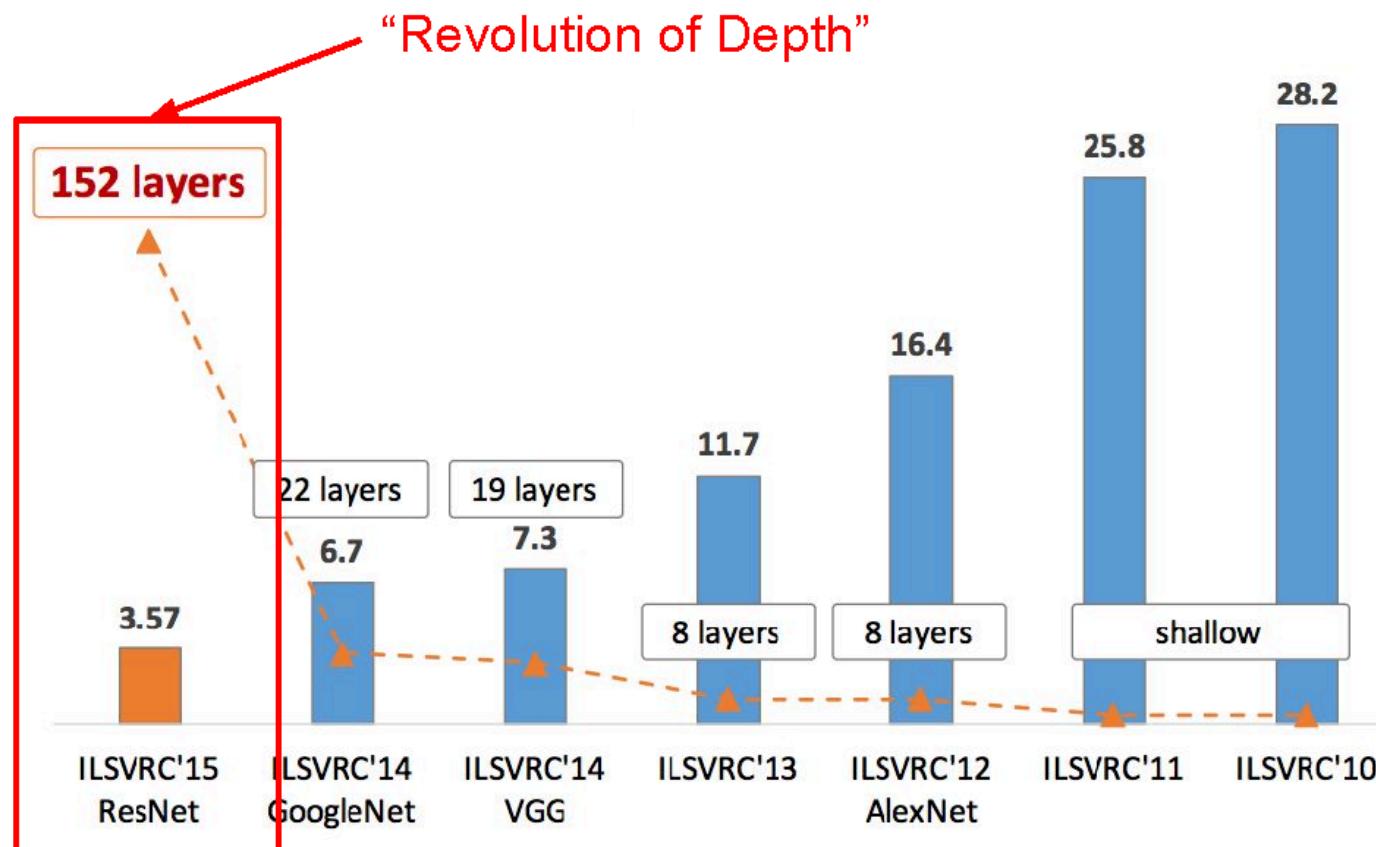




# GoogleNet

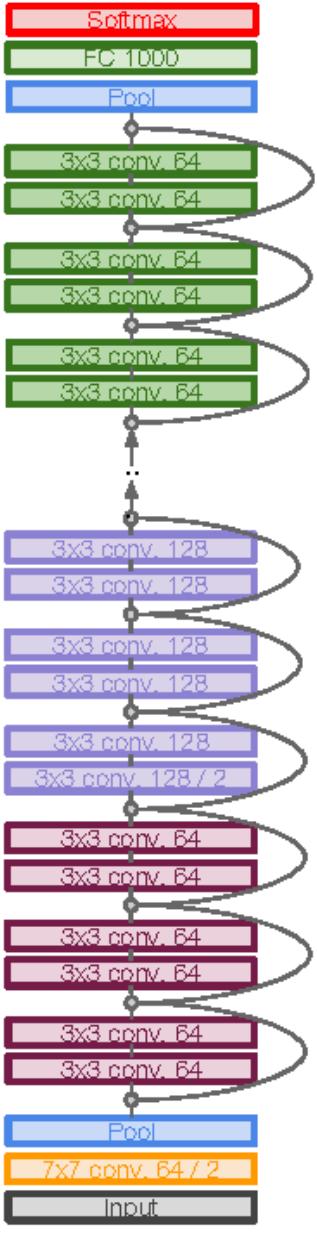
Introduced the idea that CNN layers **didn't always have to be stacked up sequentially**. Coming up with the Inception module, the authors showed that a creative structuring of layers can lead to improved performance and **computationally efficiency**.

# 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.

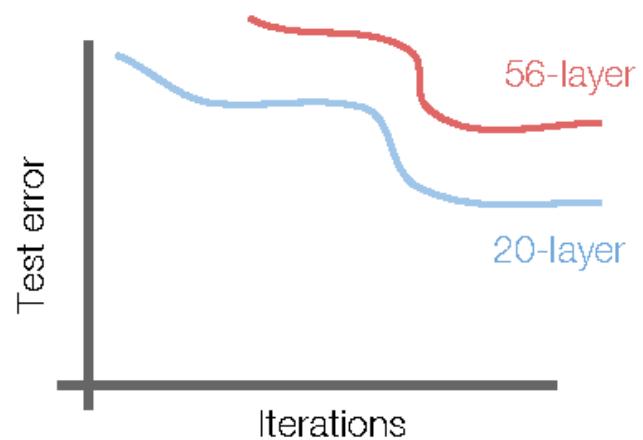
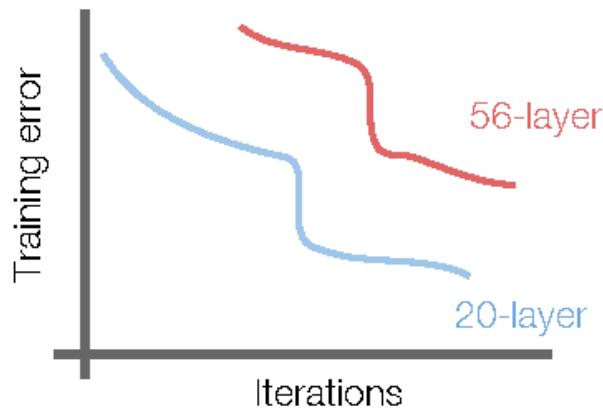


# ResNet

- 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

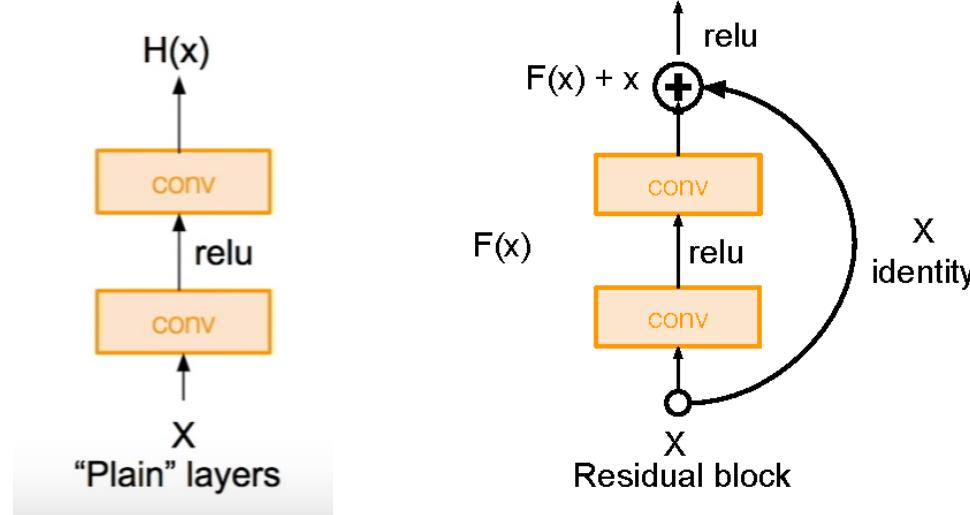
- **Hypothesis:** The problem is an optimization problem. Very deep networks are harder to optimize.
- **Solution:** Use network layers to fit residual mapping instead of directly trying to fit a desired underlying mapping.
- We will use **skip connections** allowing us to take the activation from one 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

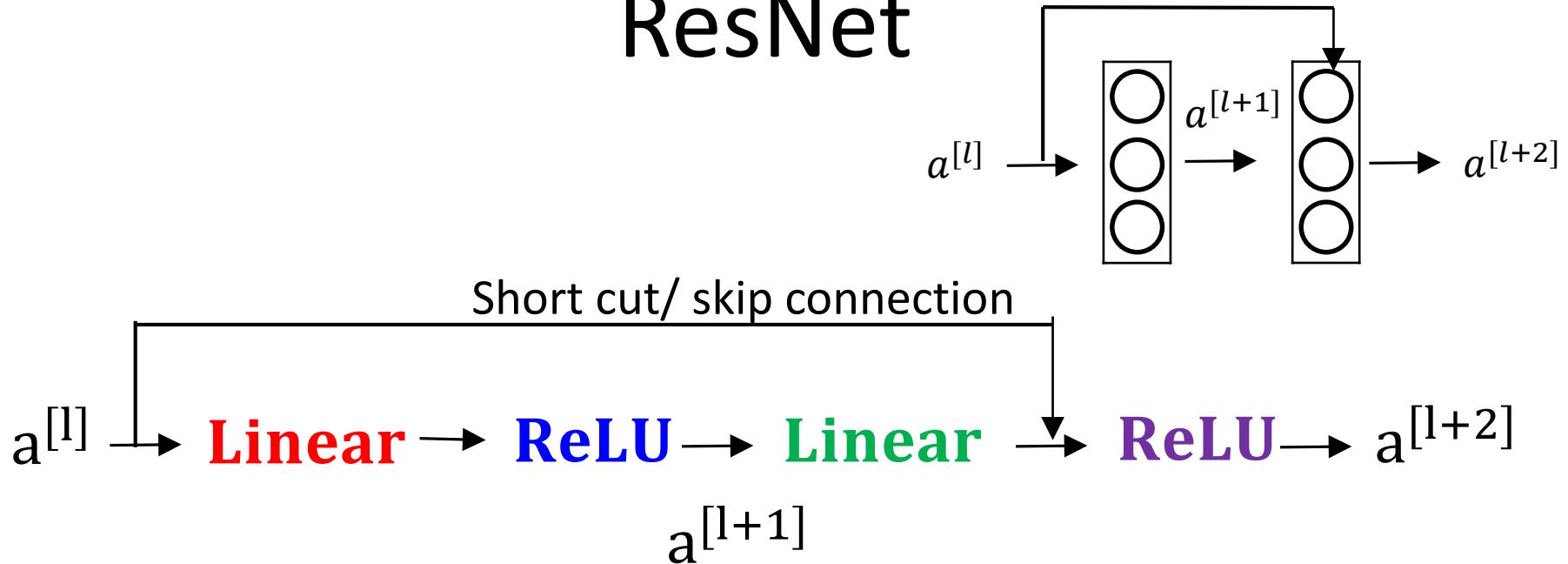
## Residual Block

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

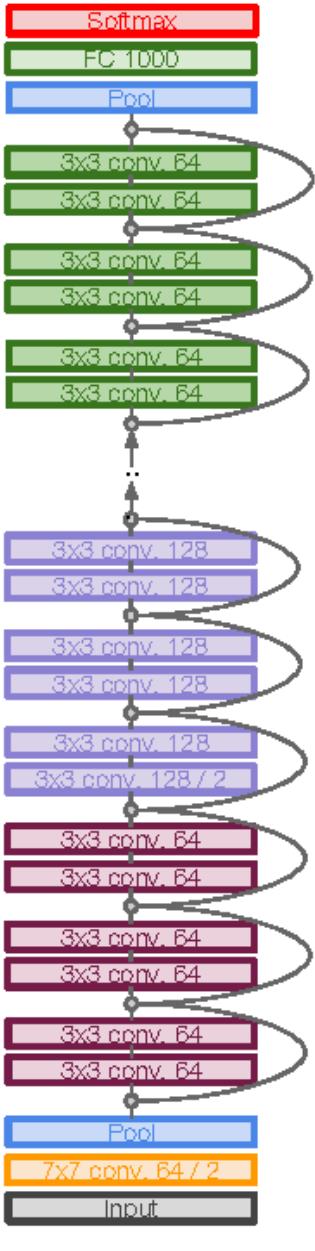


$$\mathbf{z}^{[l+1]} = \mathbf{W}^{[l+1]} \mathbf{a}^{[l]} + \mathbf{b}^{[l+1]} \quad \mathbf{z}^{[l+2]} = \mathbf{W}^{[l+2]} \mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]}$$

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

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

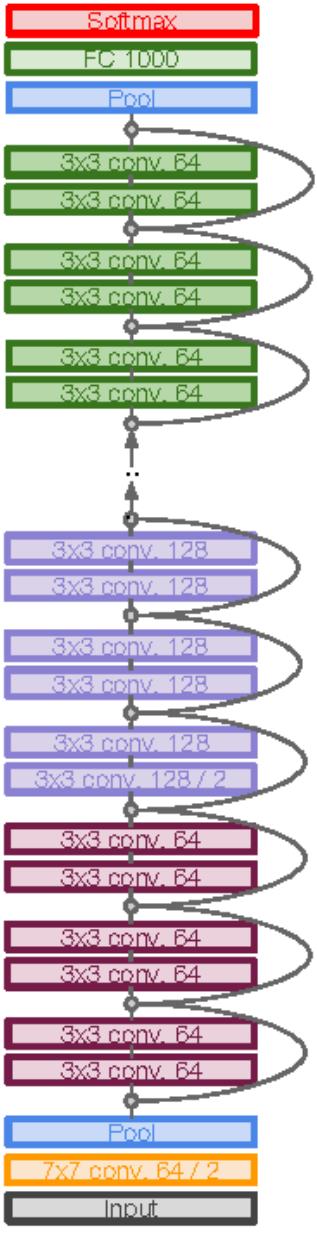
$$\mathbf{a}^{[l+2]} = g(\mathbf{z}^{[l+2]} + \mathbf{a}^{[l]}) = g(\mathbf{W}^{[l+2]} \mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]} + \mathbf{a}^{[l]})$$



# 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

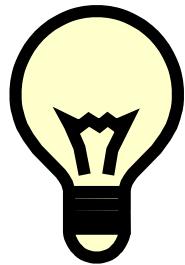
## Experimental Results:

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

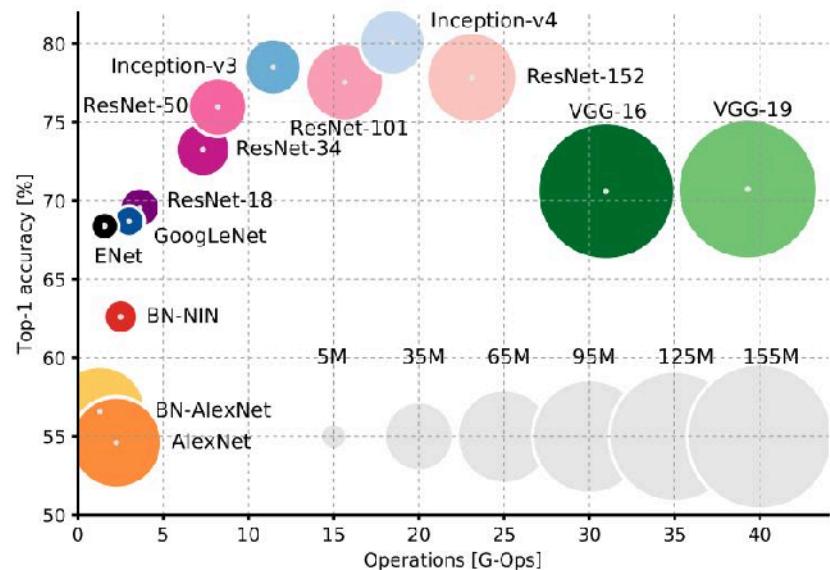
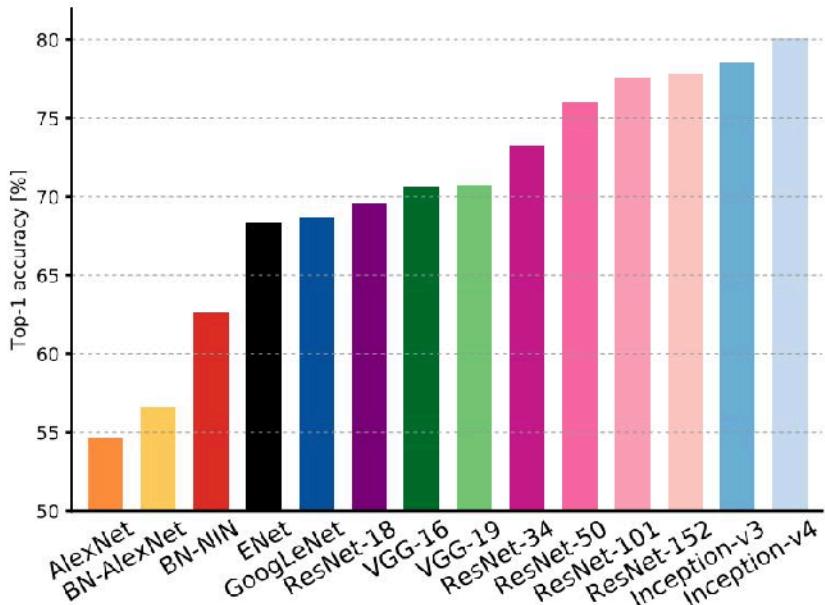


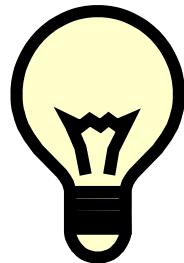
# ResNet

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

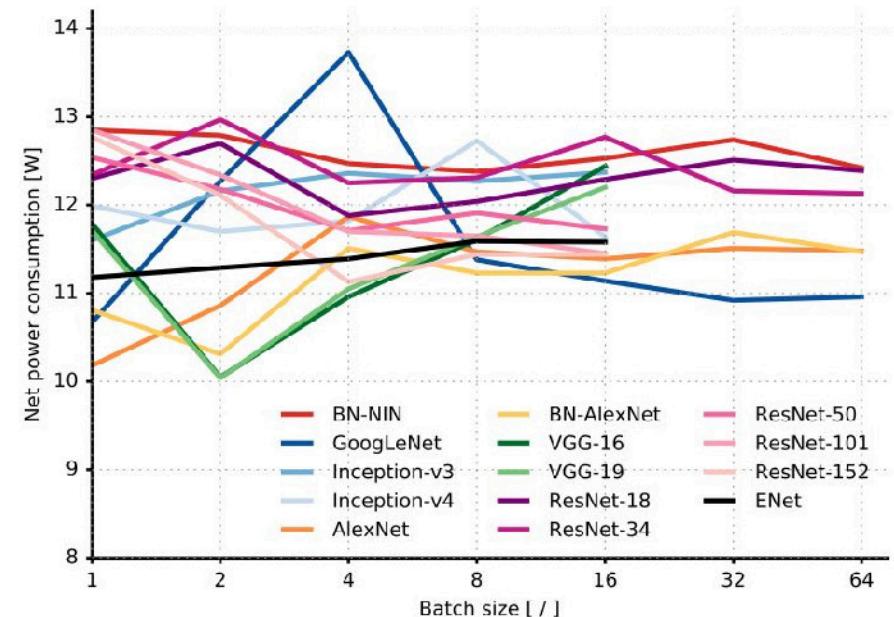
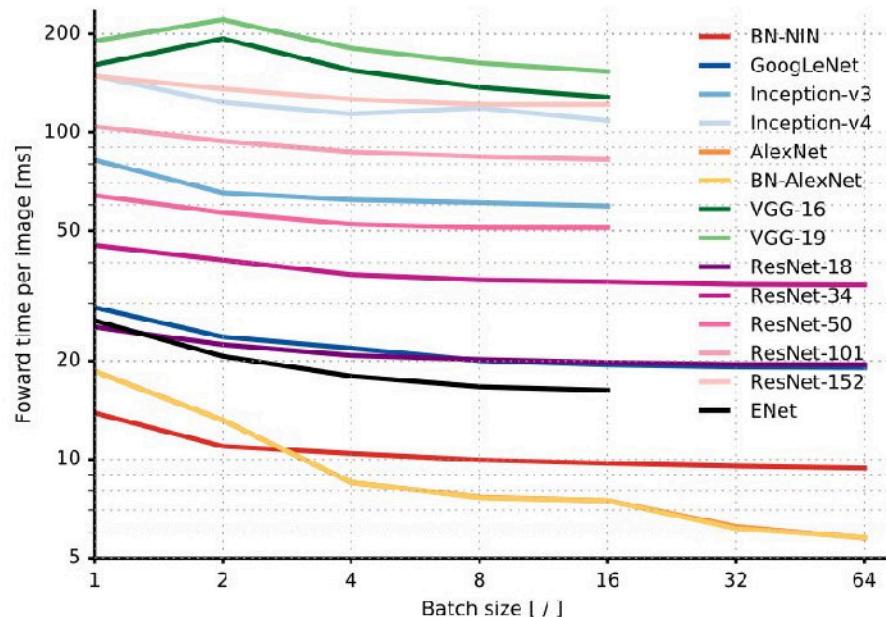


# Accuracy comparison

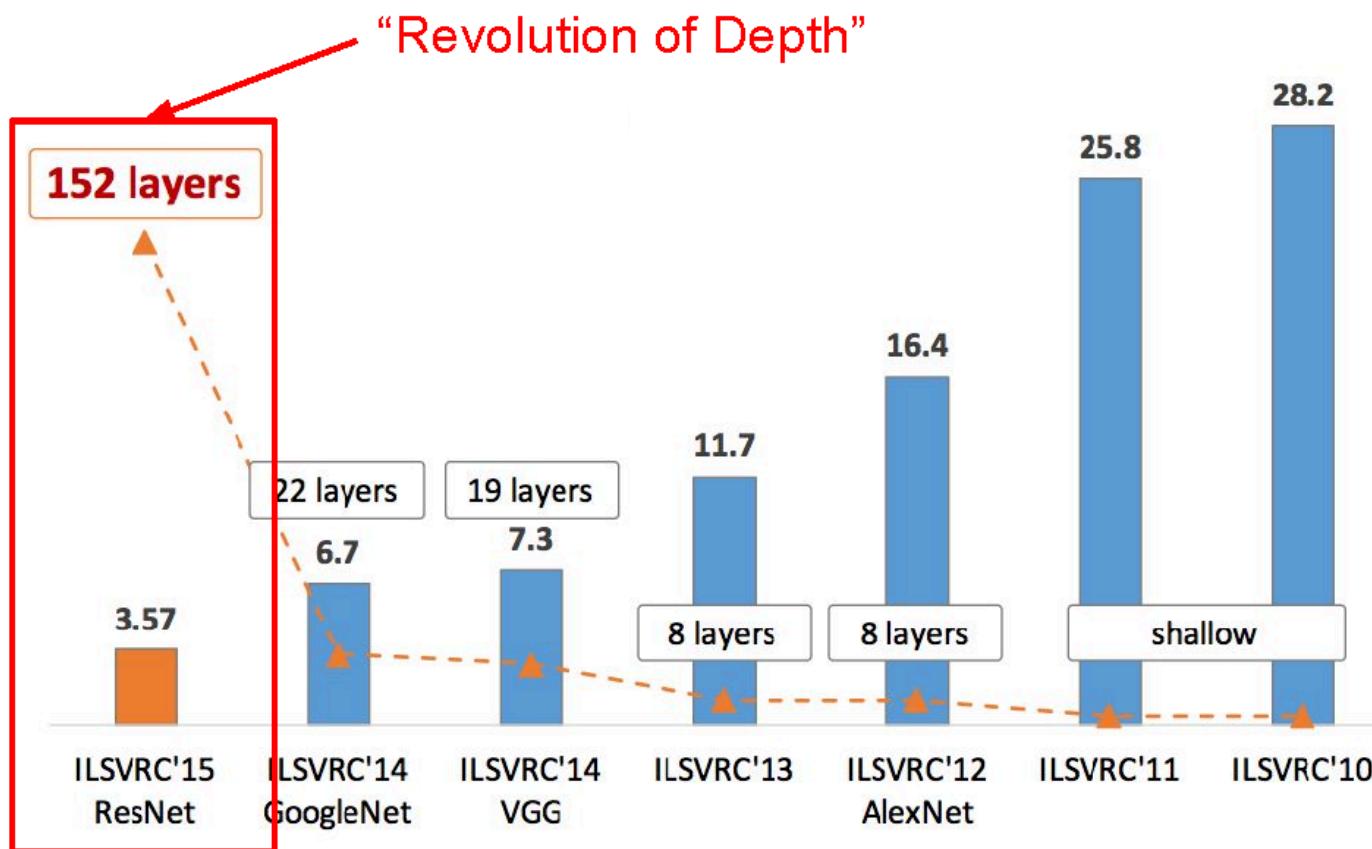




# Forward pass time and power consumption

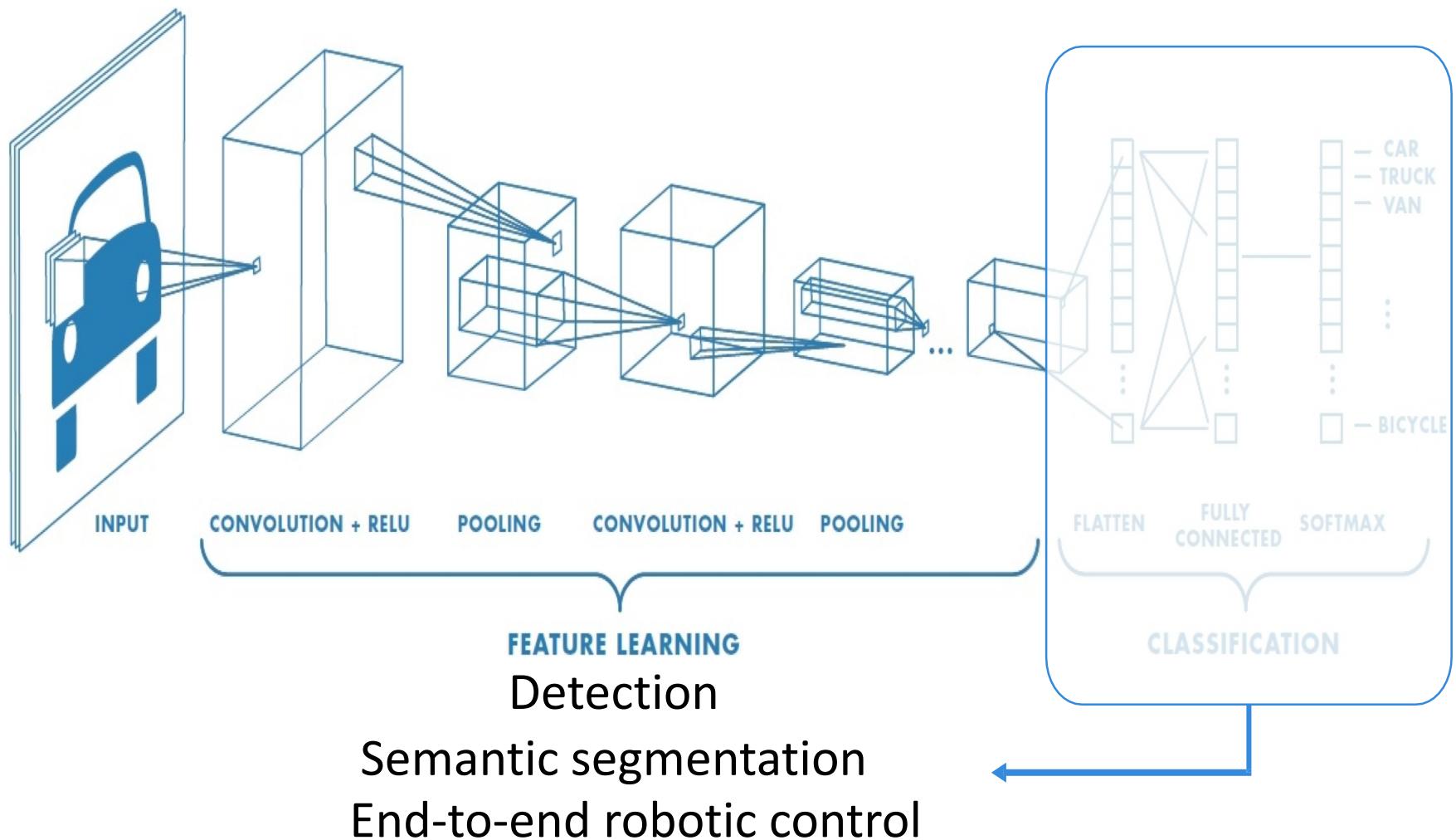


# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# Countless applications

# An Architecture for Many Applications



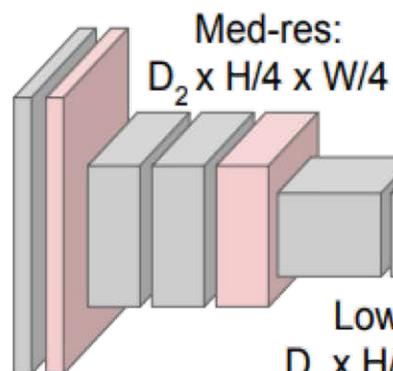
# Semantic Segmentation: Fully Convolutional Networks

FCN: Fully Convolutional Network.

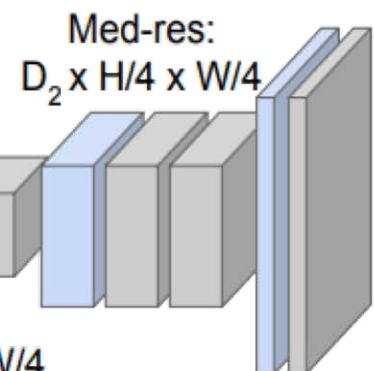
Network designed with all convolutional layers, with **downsampling** and **upsampling** operations



Input:  
 $3 \times H \times W$



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



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

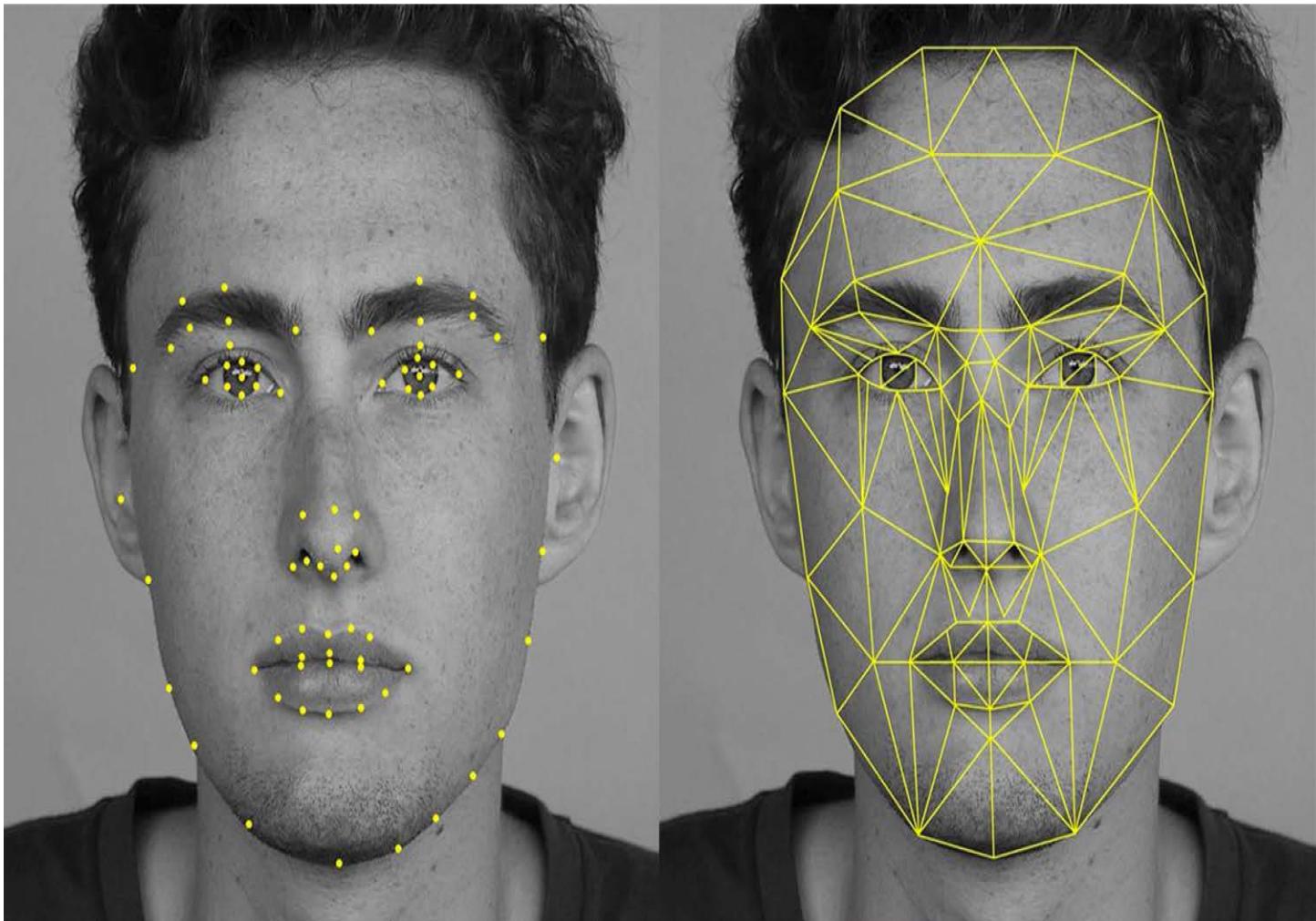
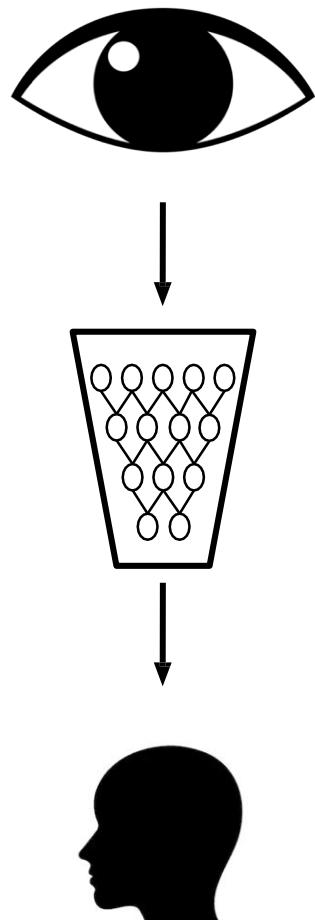


Predictions:  
 $H \times W$

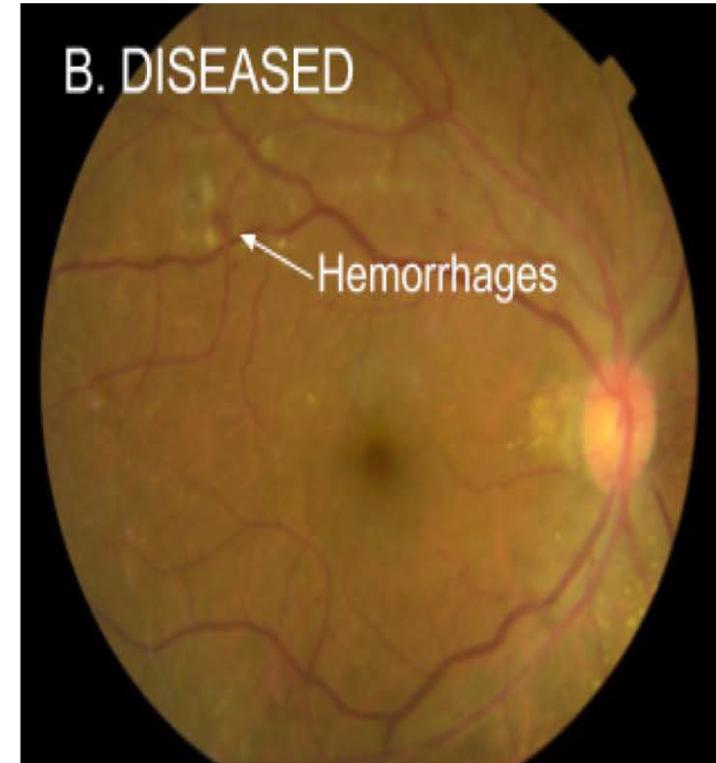
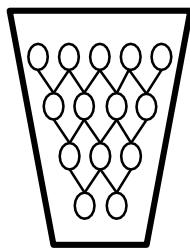
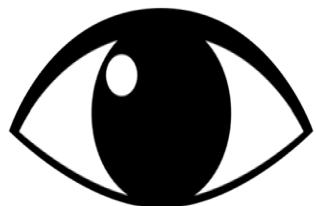


`tf.keras.layers.Conv2DTranspose`

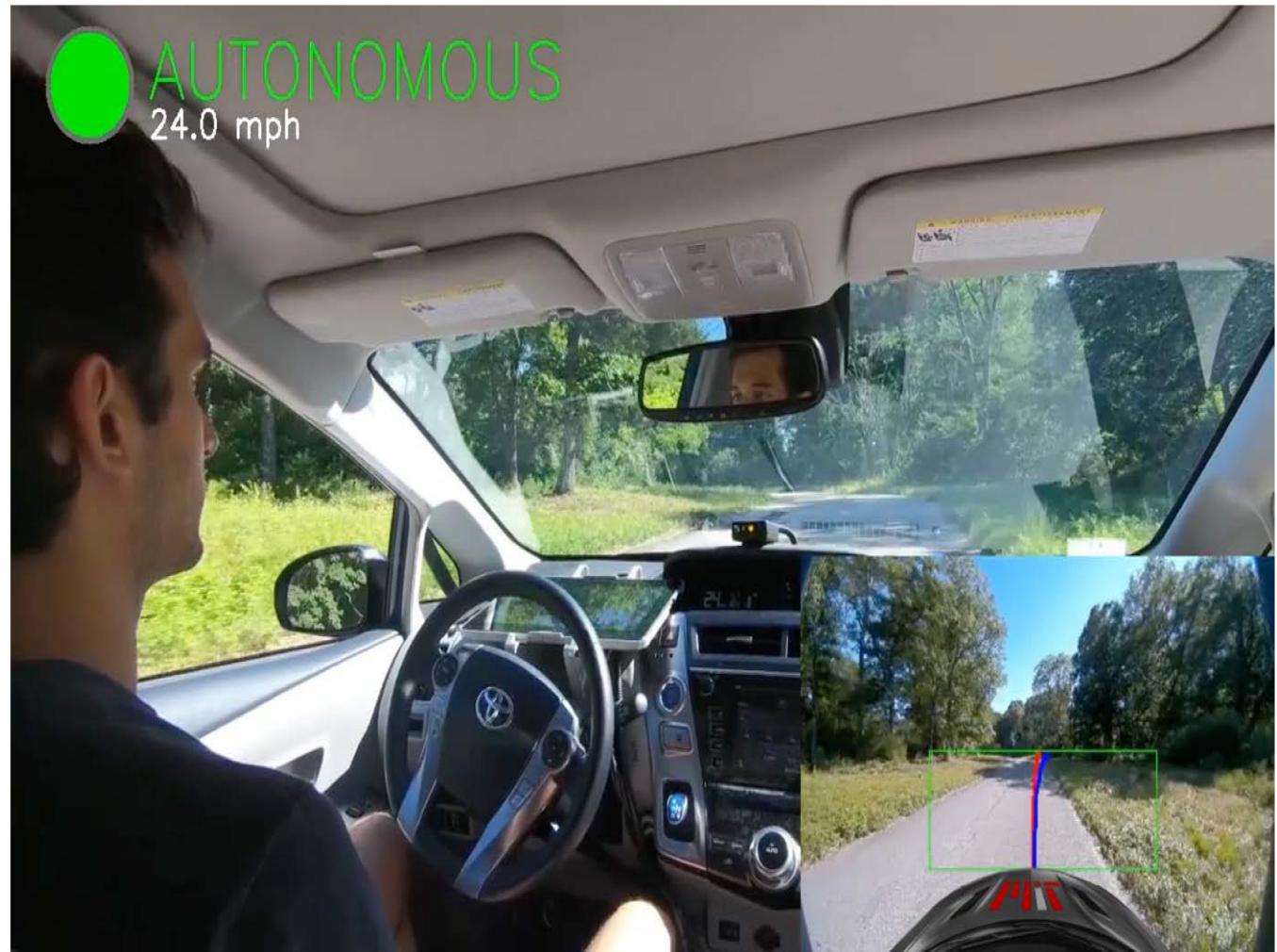
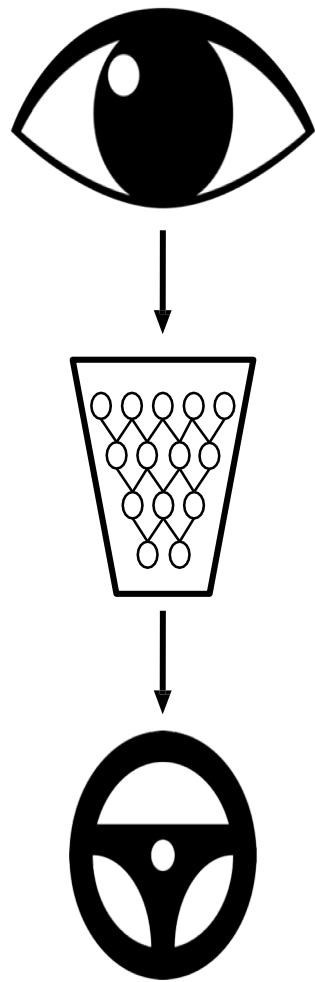
# Facial Detection & Recognition



# Medicine, Biology, Healthcare

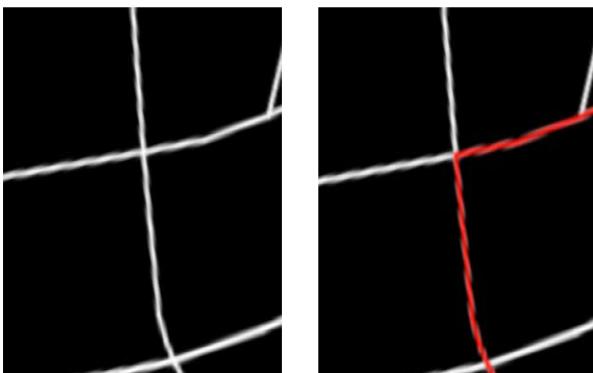


# Self-Driving Cars



# Self-Driving Cars: Navigation from Visual Perception

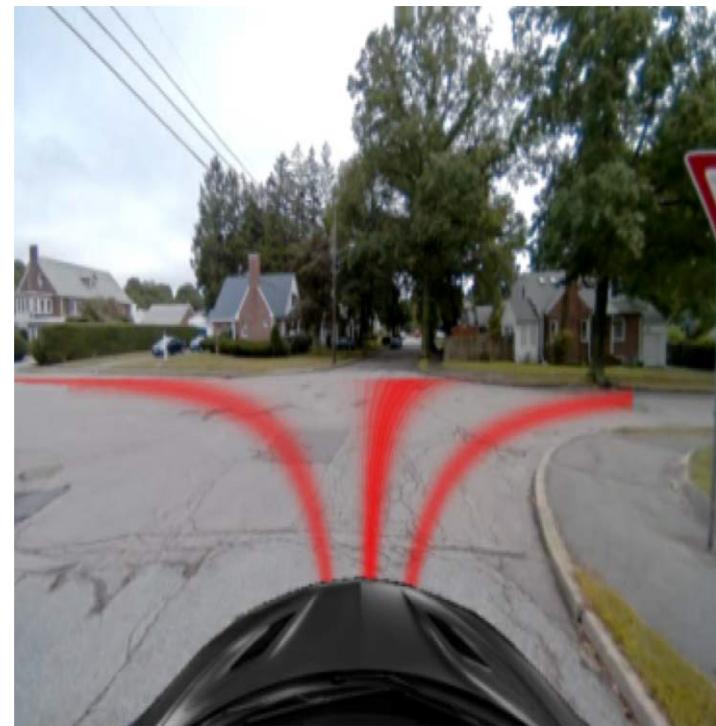
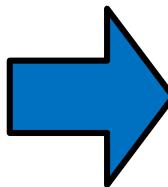
**Raw  
Perception**  
!  
(ex.camera)



**Coarse  
Maps**  
"  
(ex.GPS)

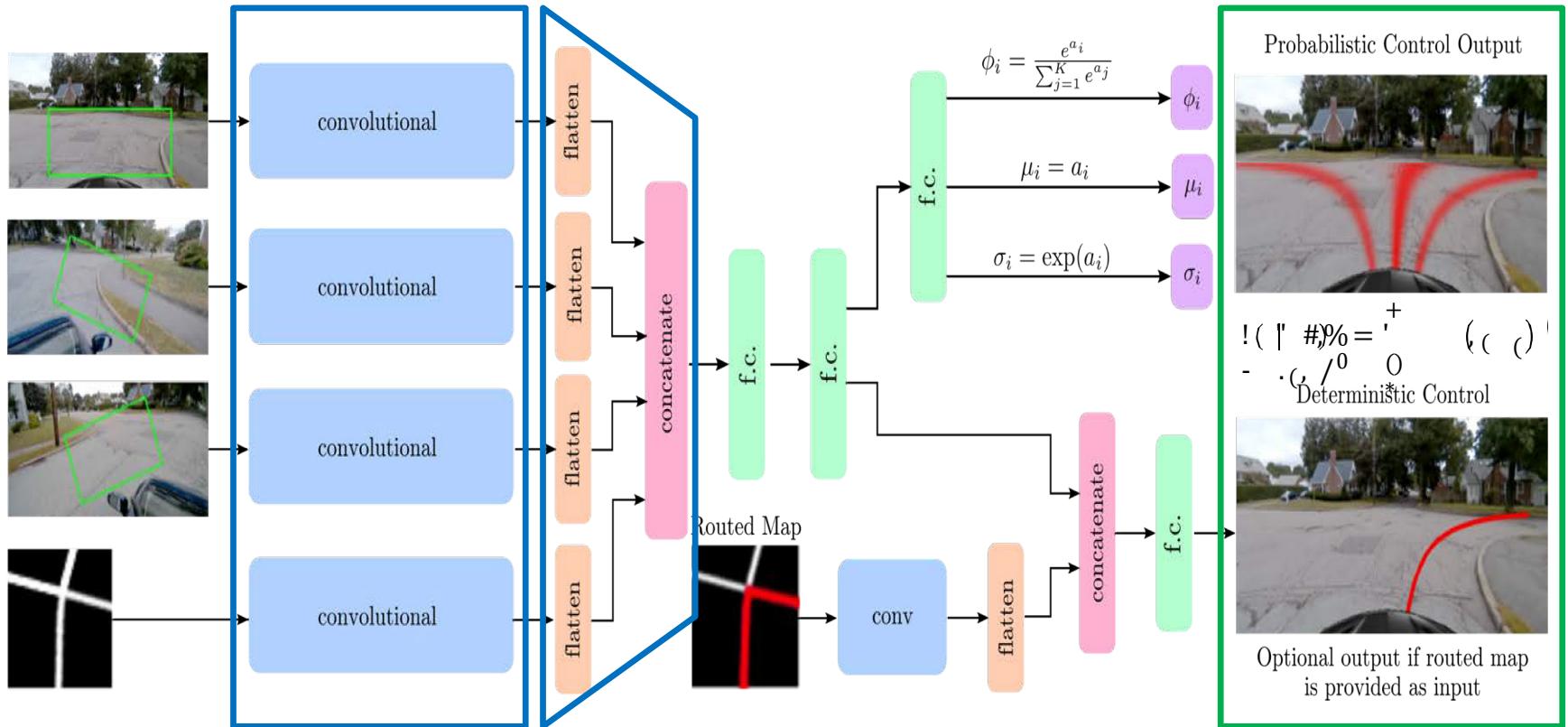


**Possible Control Commands**

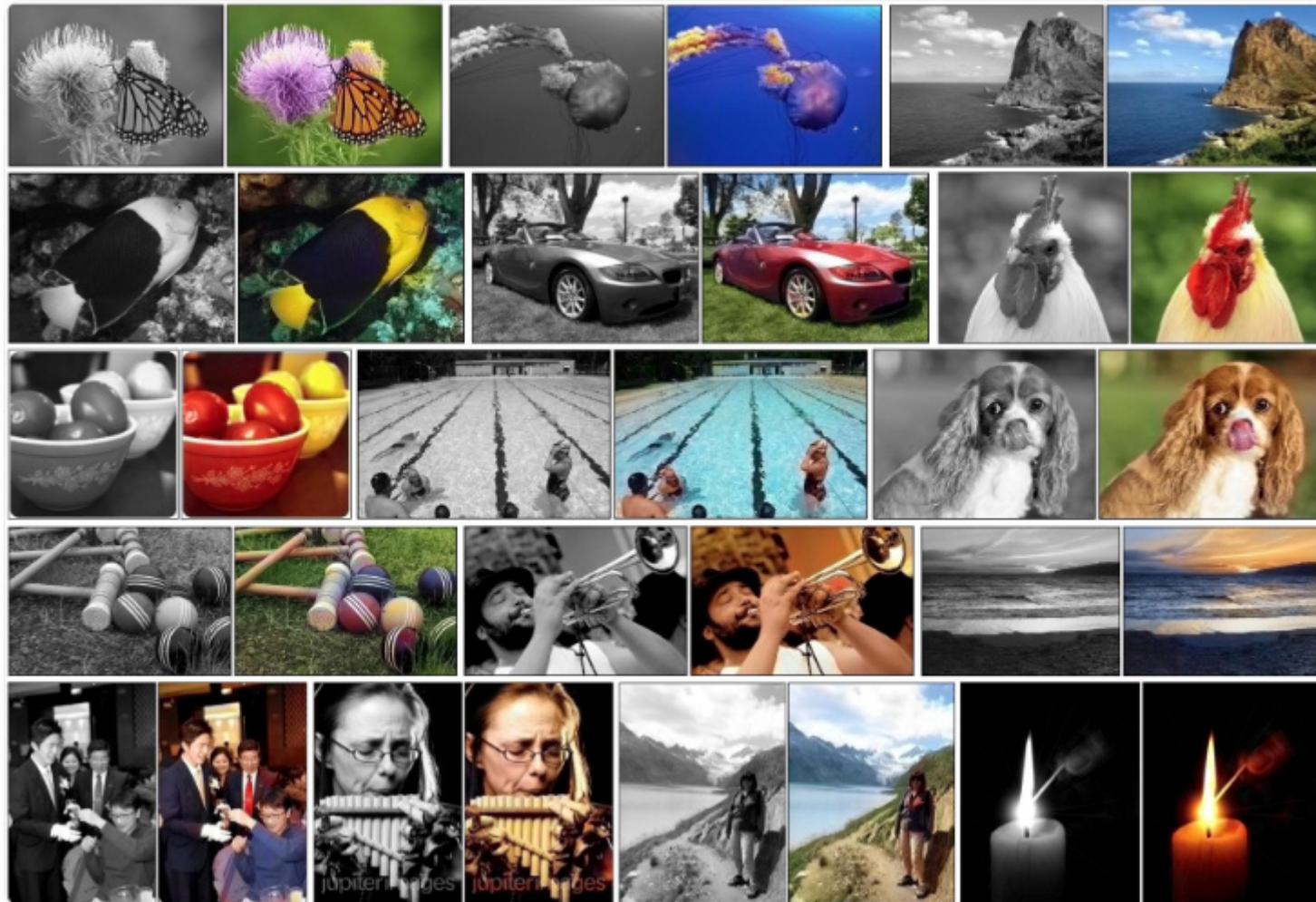


# End-to-End Framework for Autonomous Navigation

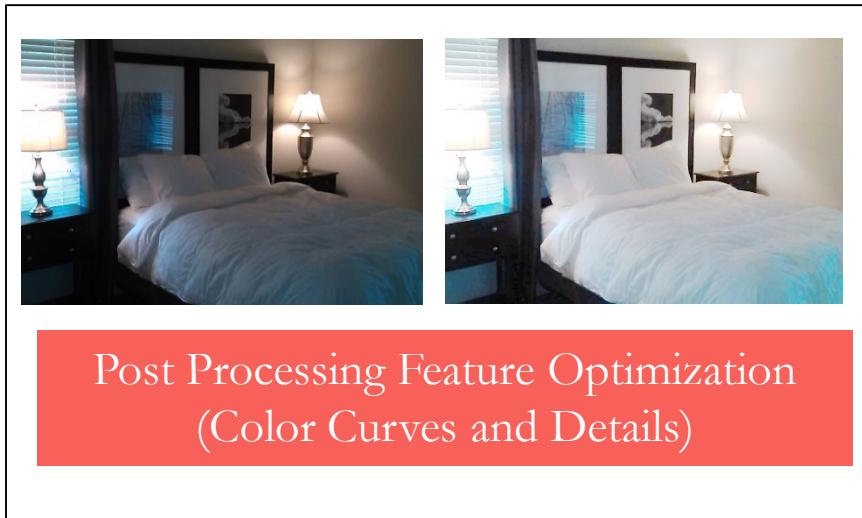
Entire model is trained end-to-end **without any human labelling or annotations**



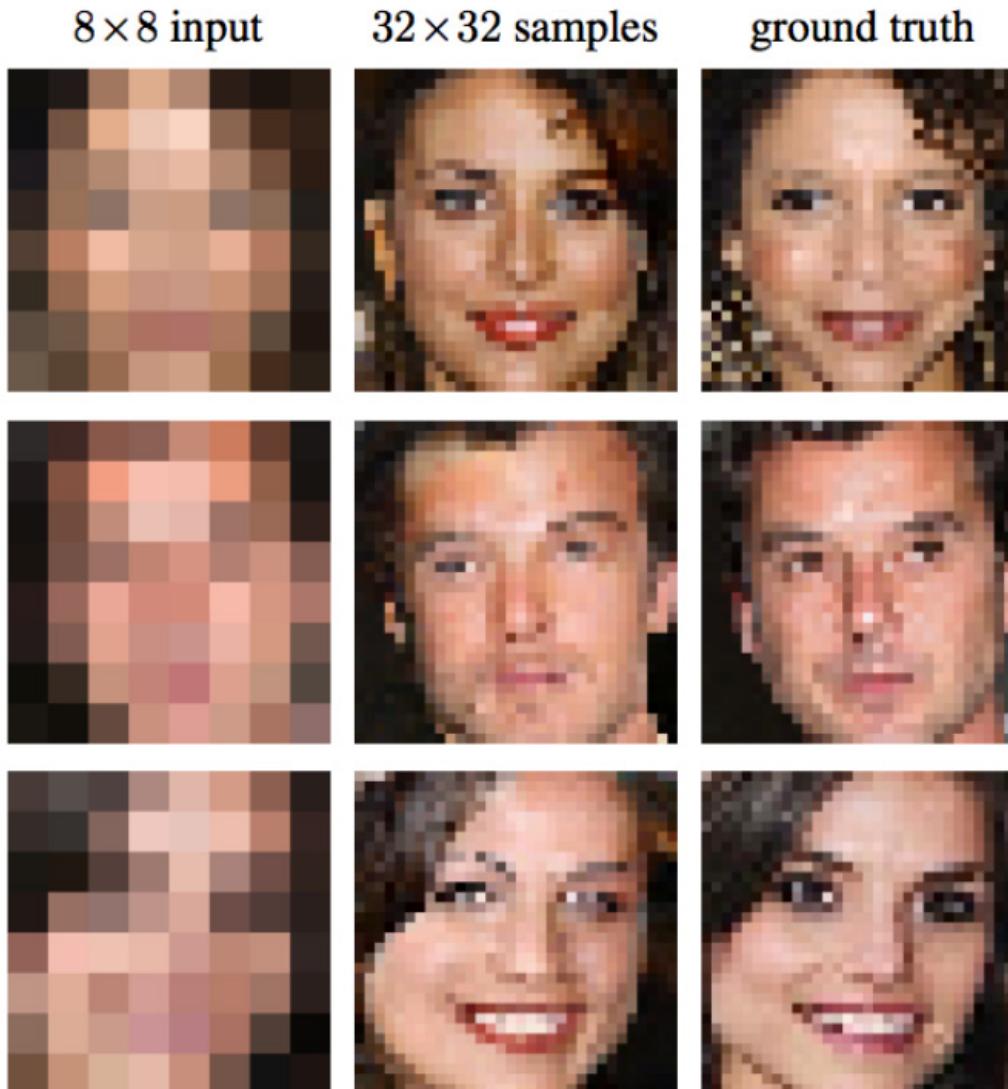
# Automatic Colorization of Black and White Images



# Optimizing Images



# Up-scaling low-resolution images



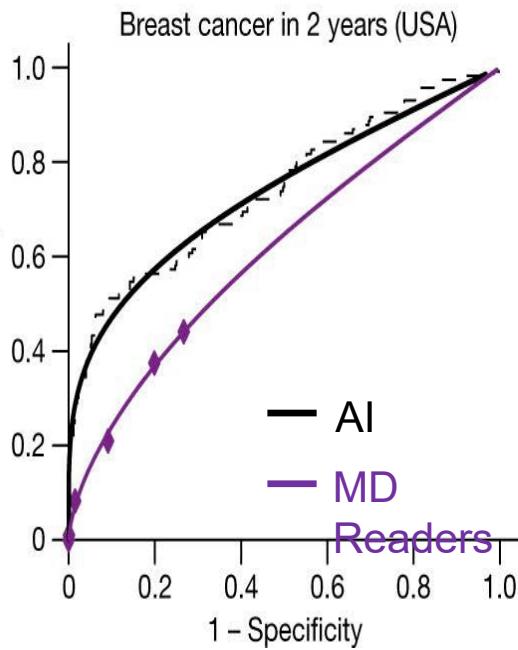
8x8 pixel photos were inputted into a Deep Learning network which tried to guess what the original face looked like. As you can see it was fairly close (the correct answer is under "ground truth").

# Breast Cancer Screening

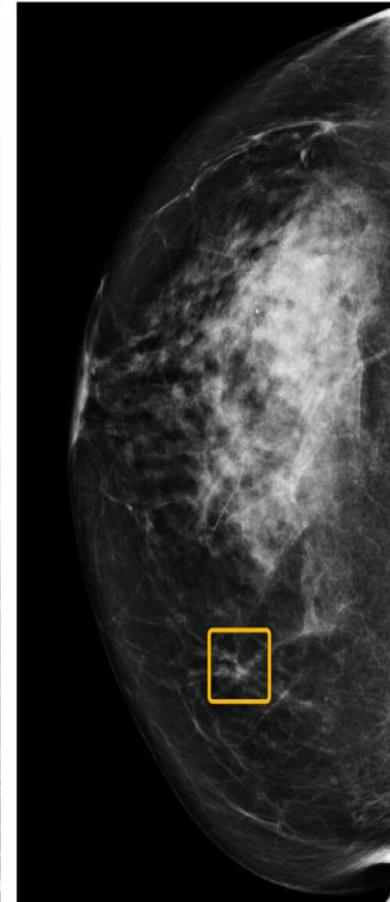
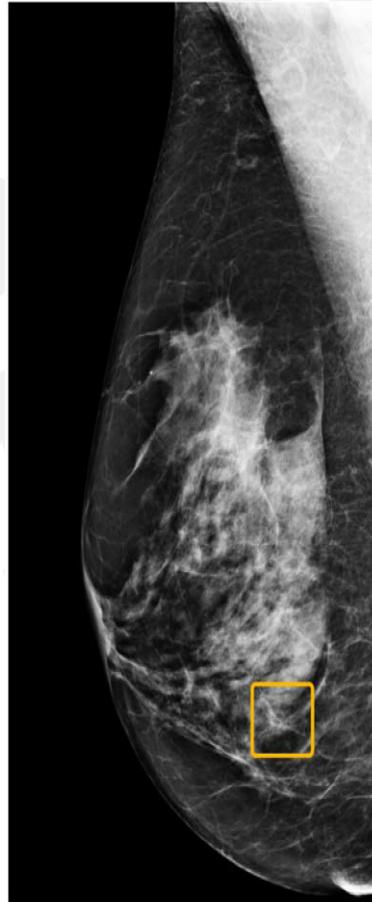
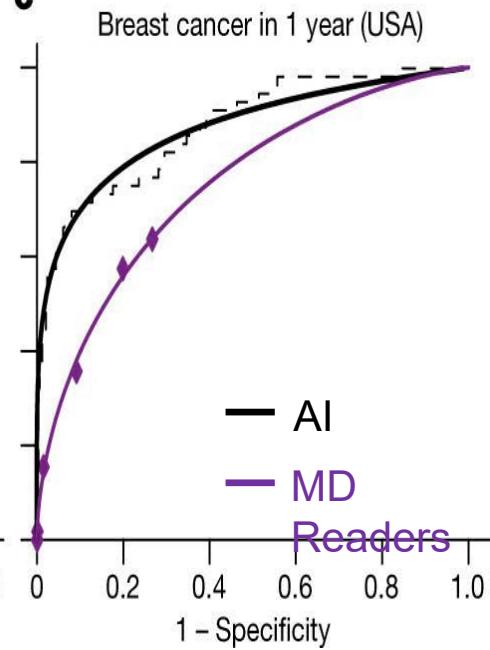
## International evaluation of an AI system for breast cancer screening

nature

b



c

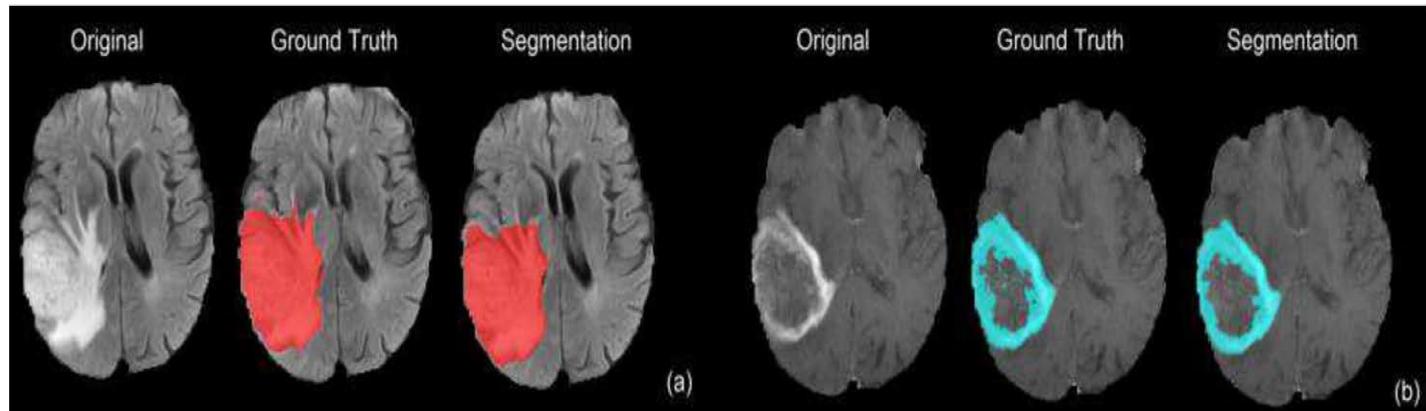


CNN-based system outperformed expert radiologists at detecting breast cancer from mammograms

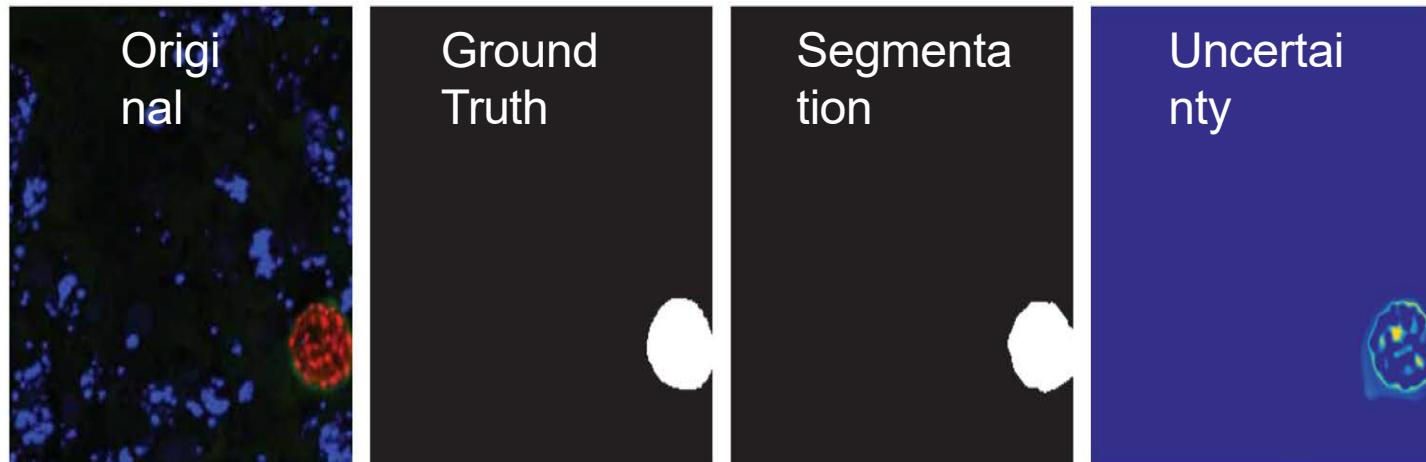
Breast cancer case missed by radiologist but detected by AI

# Semantic Segmentation: Biomedical Image Analysis

Brain Tumors  
Dong+ *MIUA*  
2017.

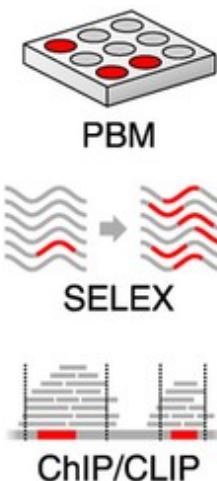


Malaria Infection  
Soleimany+ *arXiv*  
2019.



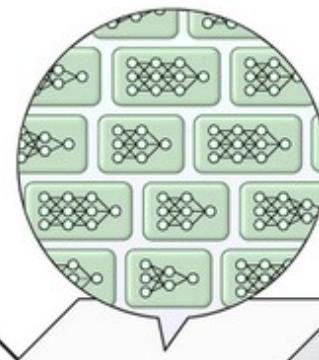
# DeepBind

## 1. High-throughput experiments



## 2. Massively parallel deep learning

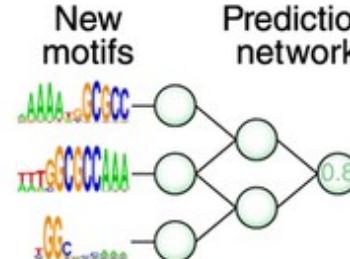
### Automatic model training



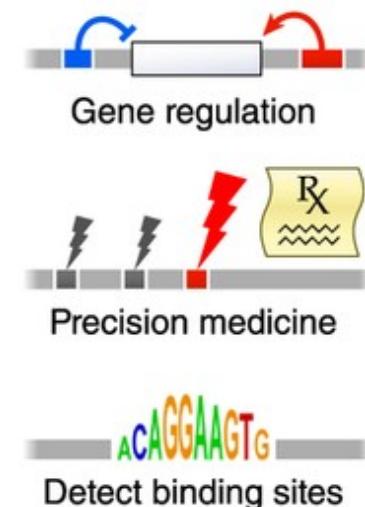
GPU server

### New motifs

### Prediction network



## 3. Community needs



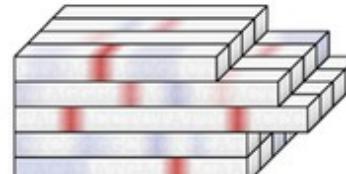
### Current batch of inputs

CTAACGCCACCGTCT  
TTAGGGGCACCACTACT  
TAGCACCTCTATTGCACCC  
CTCGGGGGCCCTGCAAT  
TACAAATGAGCACAA

Convolve

Motif detectors

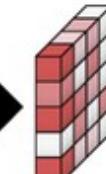
### Motif scans



Rectify

Thresholds

### Features

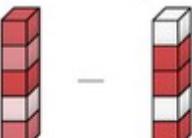


Neural network

Weights

Outputs

Targets



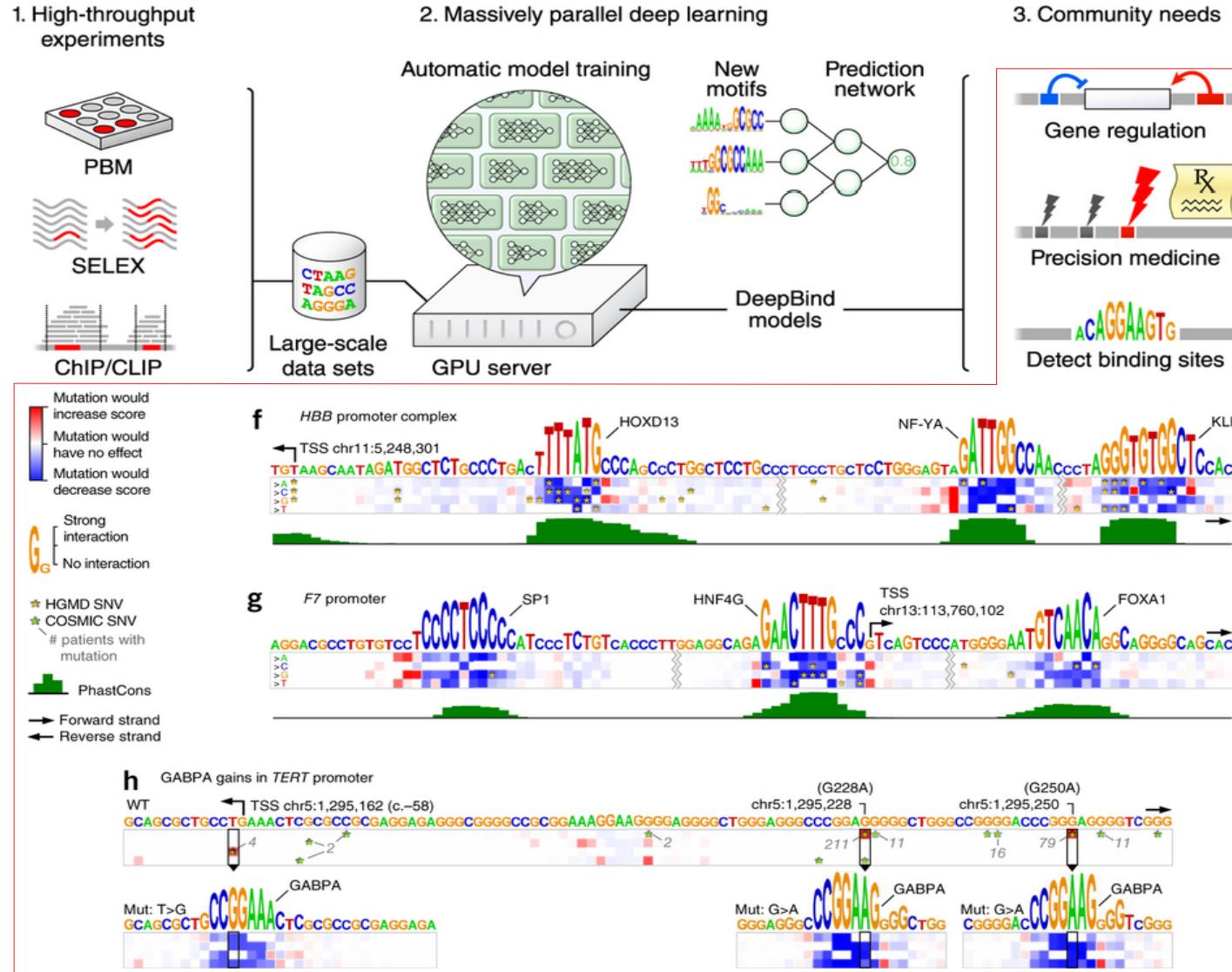
### Current model parameters

Update +

### Parameter updates

Backprop

# Predicting disease mutations



[Alipanahi et al., 2015]

# Deep Learning for Computer Vision: Summary

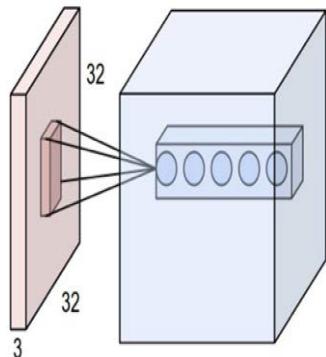
## Foundations

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



## CNNs

- CNN architecture
- Application to classification
- ImageNet



## Applications

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

