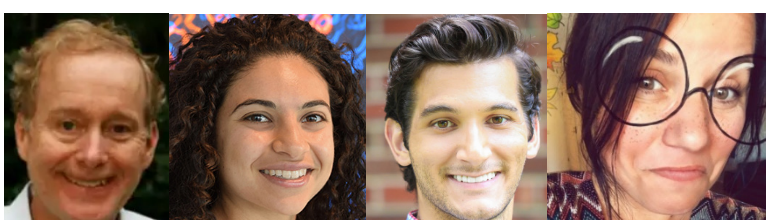
Lecture 3:
Convolutional Neural Networks

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



Massachusetts Institute of Technology

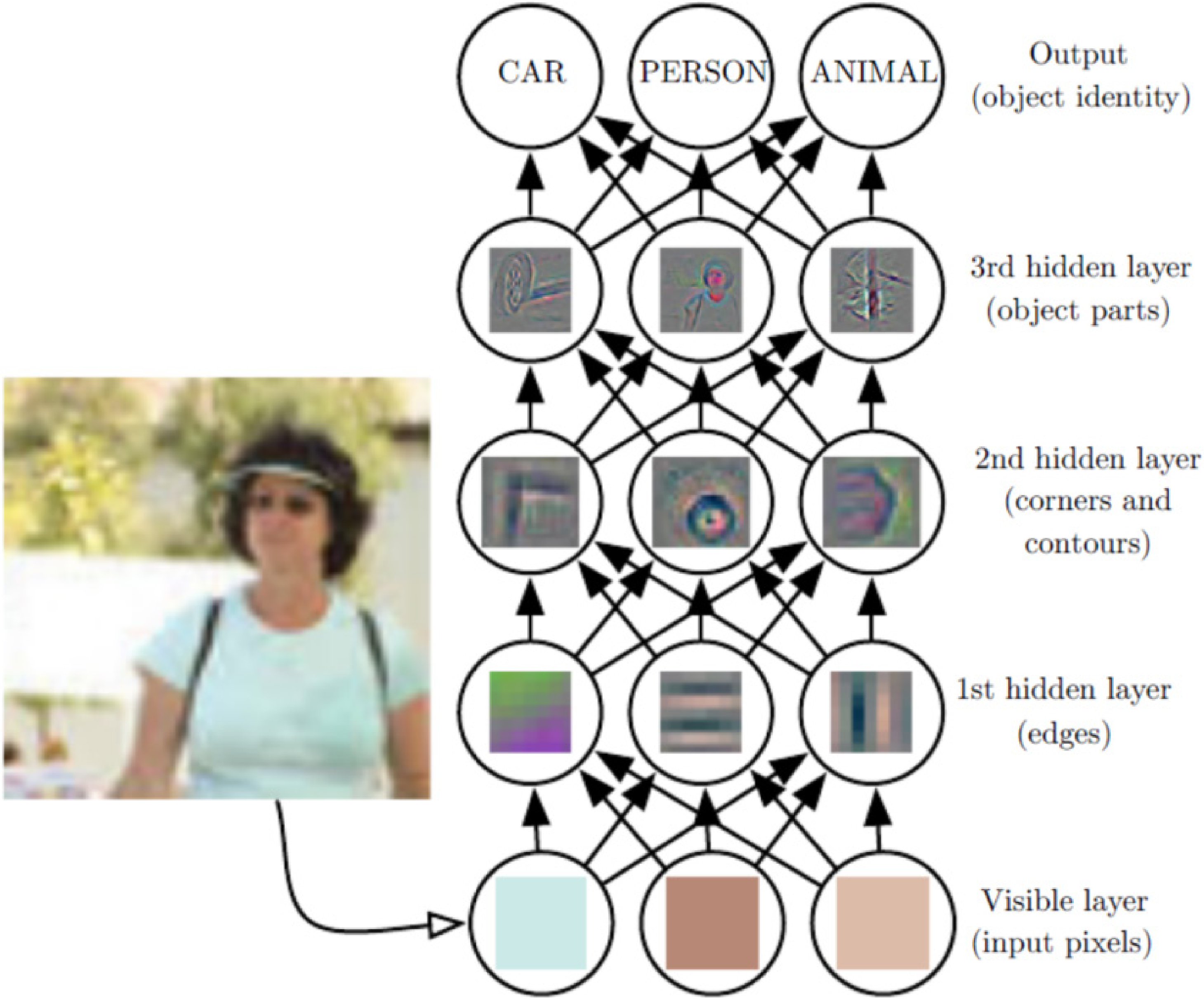
http://mit6874.github.io

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

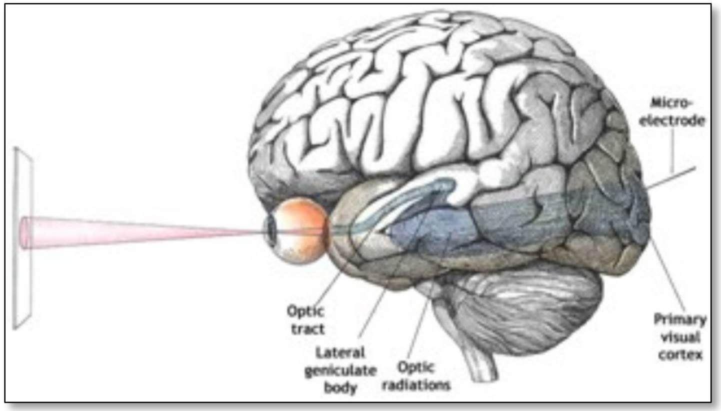
Convolutional neural networks inside our brains

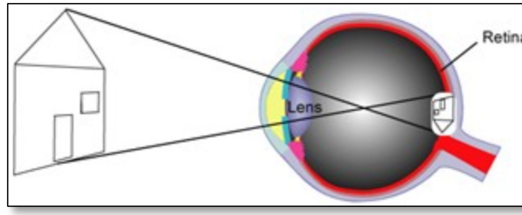
Human Vision ⇔ many **layers** of **abstraction** ⇔ **Deep** learning

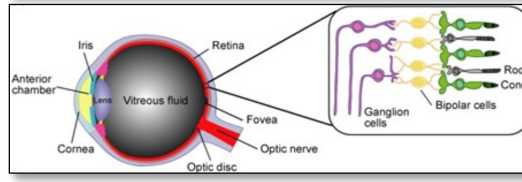


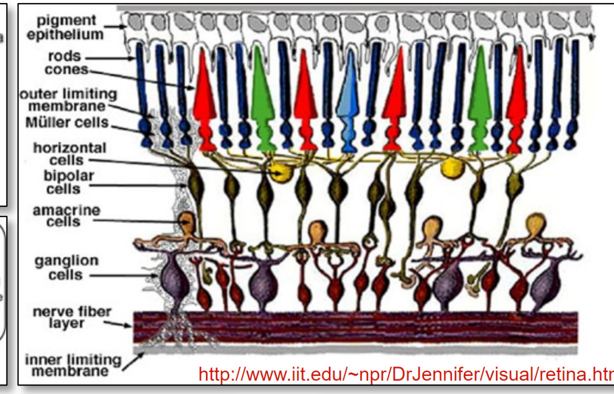
Goodfellow 2016

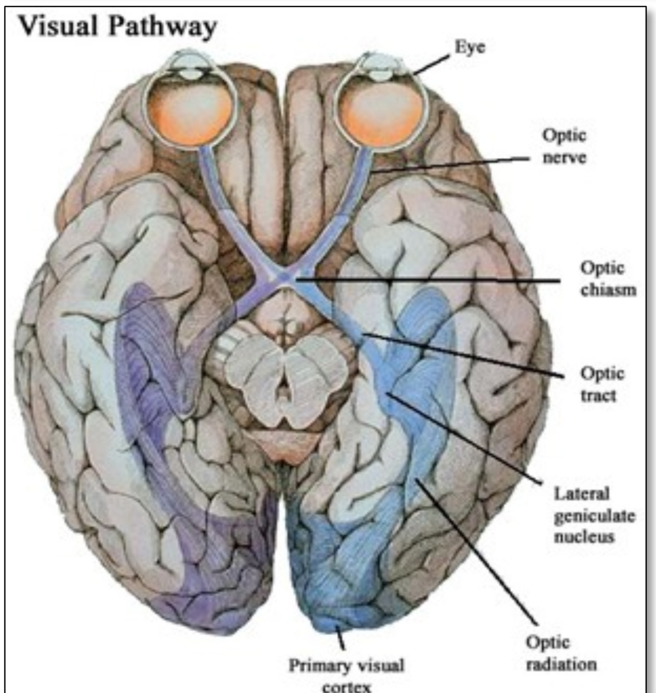
CNN inspiration in the 50s/60s: human/animal visual cortex

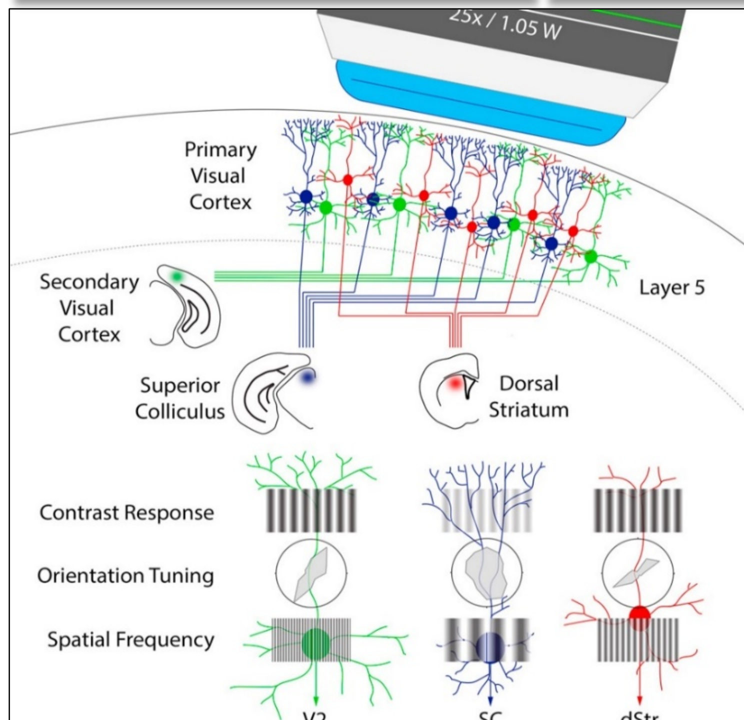


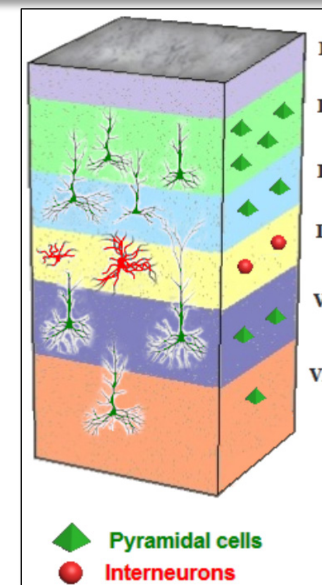






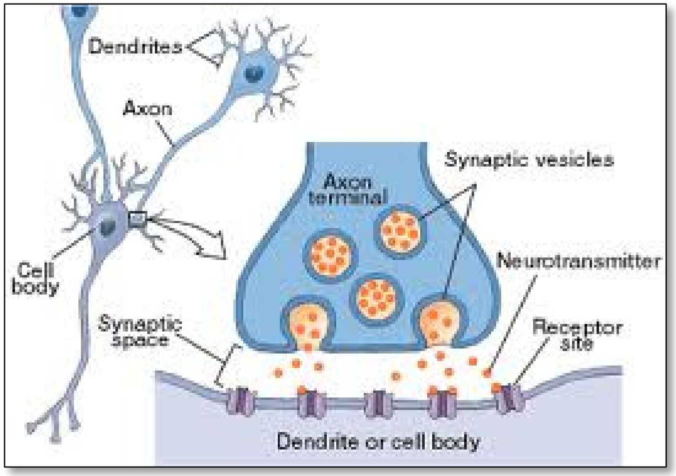


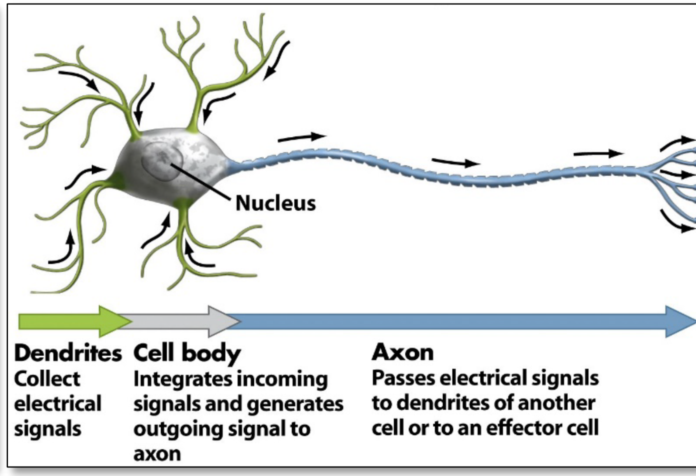


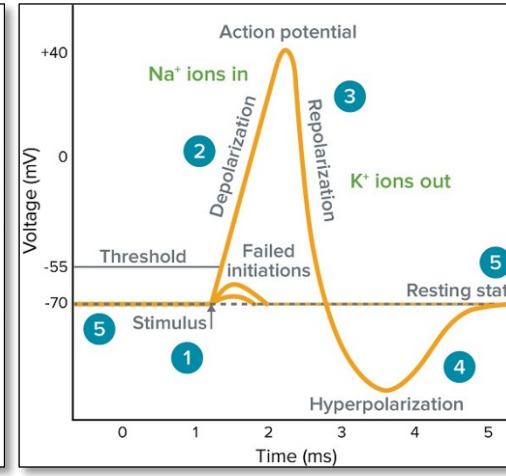


- Hubel/Wisel 1968 cat/monkey: (1) Receptive fields = local computation. (2) Simple cells = edge/orientation detectors. (3) Complex cells = position invariance/pooling
- Layers: pixels, edges (bands given slant, contrast edges), shapes, primitives, scenes
- Hierarchical abstractions, simple building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

Primitives: Neurons, action potentials, networks



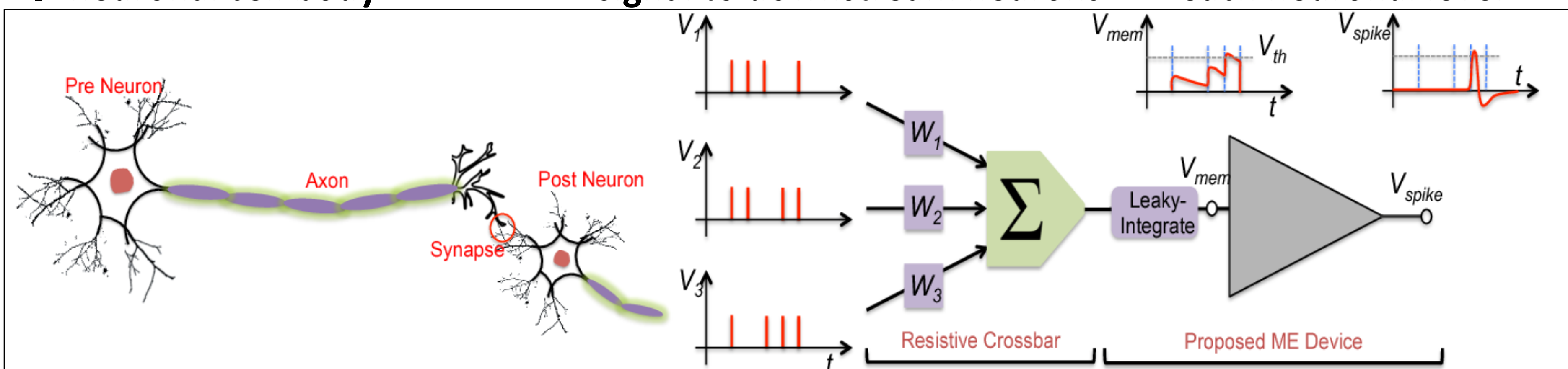




- Chemical accumulation across dendritic connections
- Pre-synaptic axon
  → post-synaptic dendrite
  → neuronal cell body

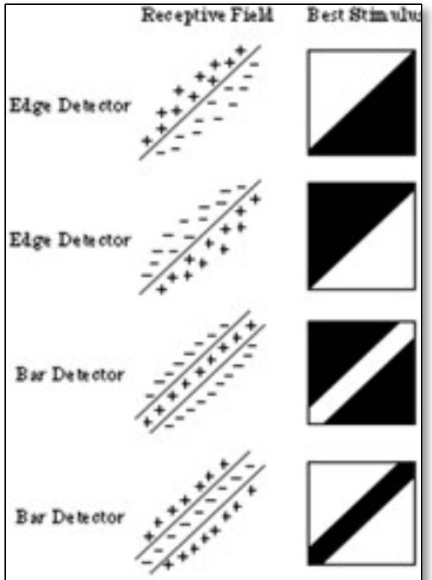
- Each neuron receives multiple signals from its many dendrites
- When **threshold** crossed, it fires
- Its axon then sends outgoing signal to downstream neurons

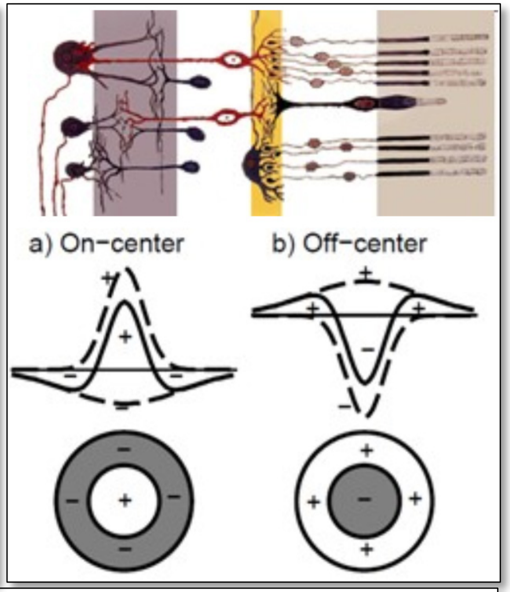
- Weak stimuli ignored
- **Activation function** signal threshold crossed
- **Non-linearity** within each neuronal level

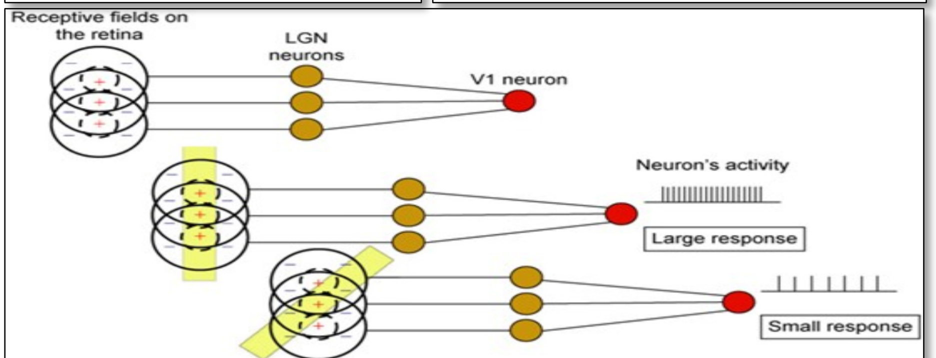


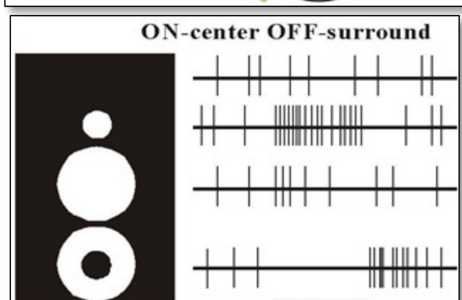
- **Neurons** connected into **circuits** (neural networks): **emergent** properties, **learning**, memory
- Simple **primitives** arranged in simple, repetitive, and extremely **large** and **deep** networks
- 86 billion neurons, each connects to 10k neurons, 1 quadrillion (10^{12}) connections
- Human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

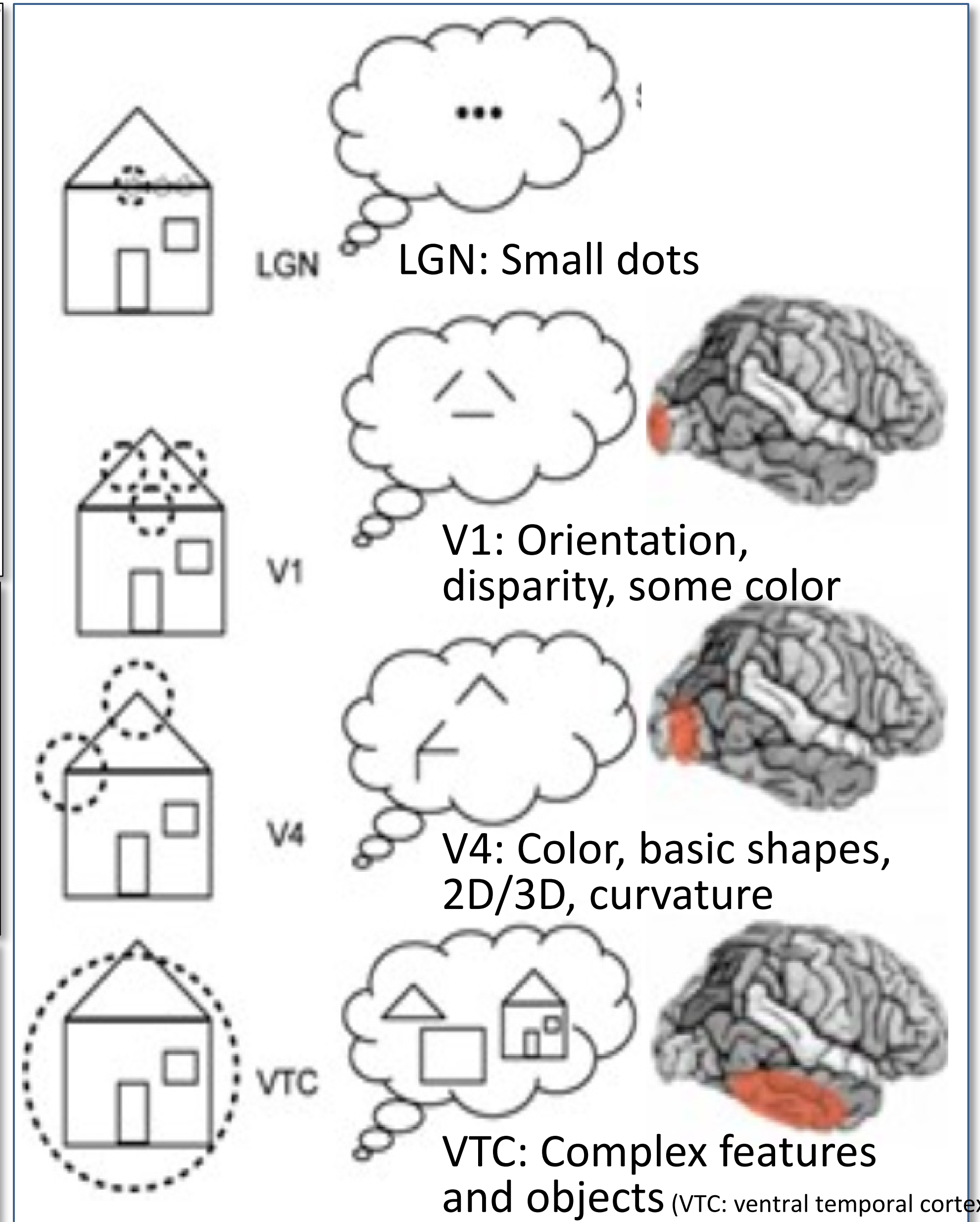
Abstraction layers: edges, bars, dir., shapes, objects, scenes





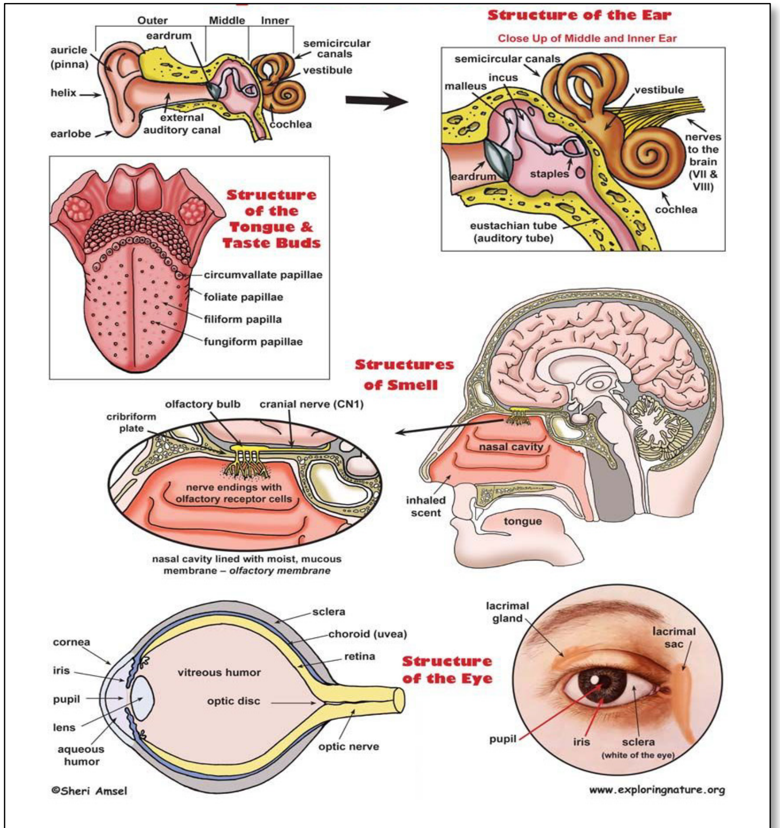




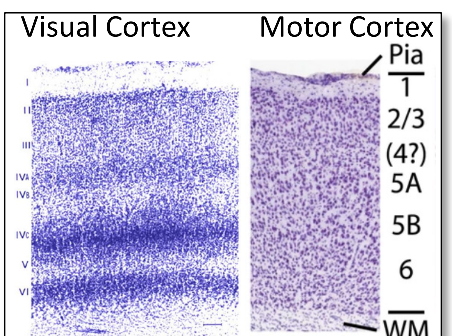


- **Primitives** of visual concepts encoded in neuronal connection in early cortical layers
- **Deep**: Abstraction **layers** ⇔ visual cortex layers
- **Complex concepts** from simple parts, **hierarchy**

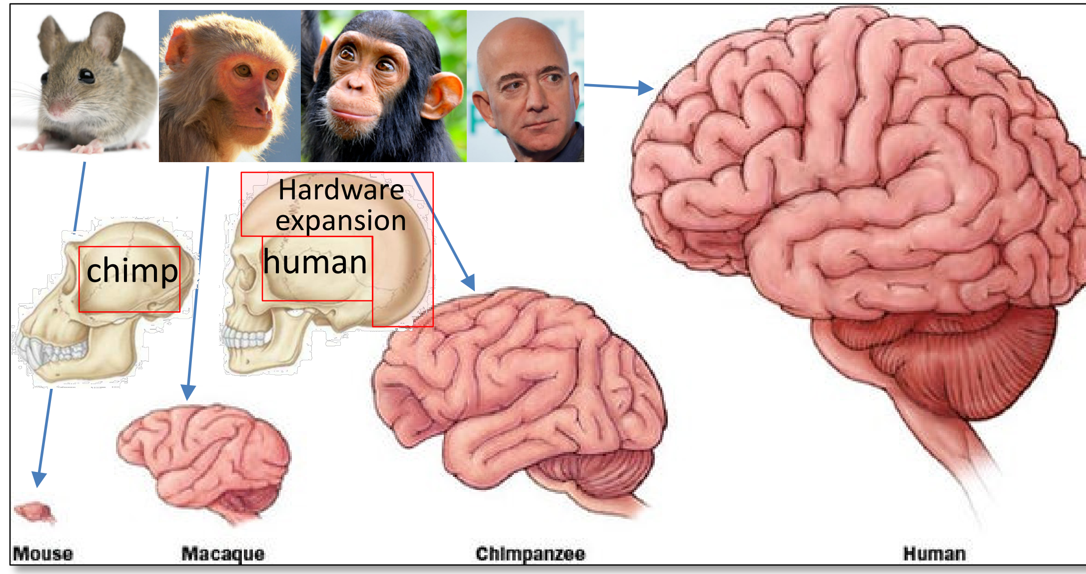
General "learning machine", reused widely



- Hearing, taste, smell, sight, touch all re-use similar learning architecture



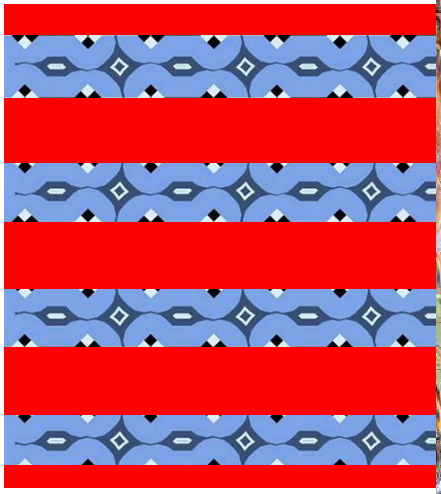
- Interchangeable circuitry
- Auditory cortex learns to 'see' if sent visual signals
- Injury area tasks shift to uninjured areas

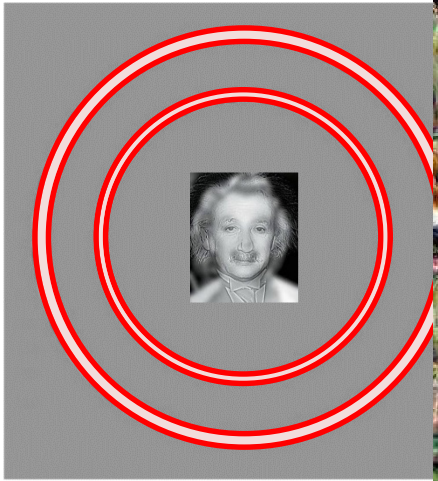


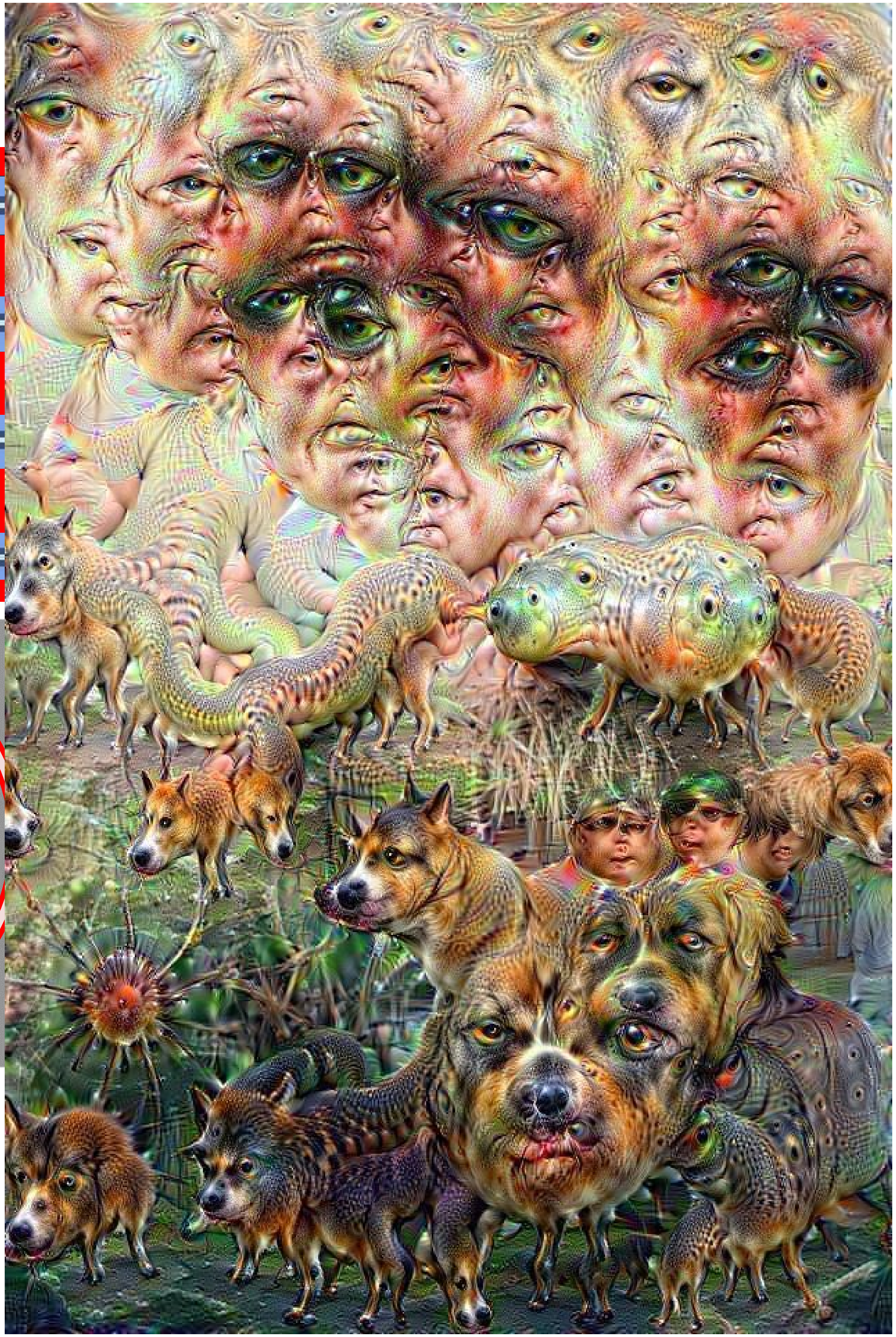
- Massive recent expanse of human brain has re-used a relatively simple but general learning architecture

- 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!

Visual illusions send conflicting signals at different filters/layers







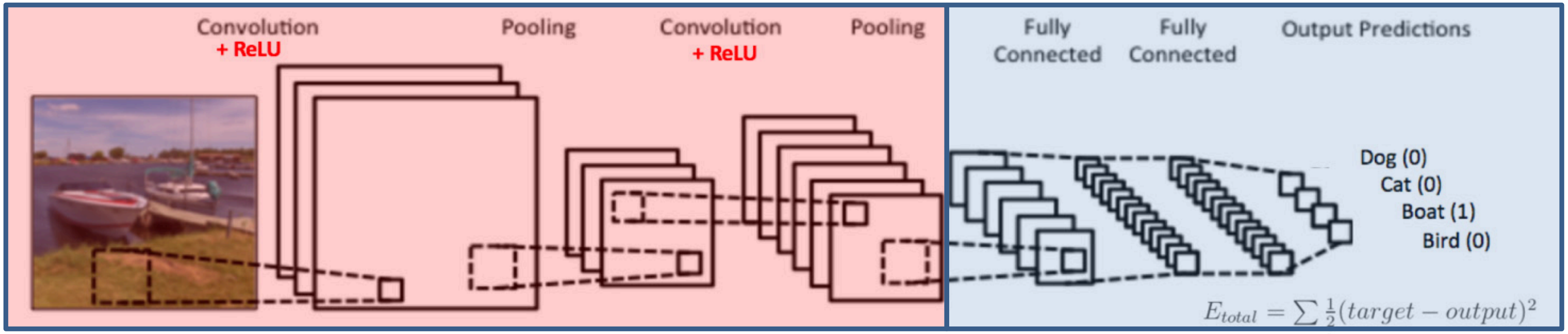
- Visual illusions reveal brain **primitives**, building blocks, computations, architecture
- Deep learning can exploit such **conflicting primitives** to create strong experiences, or for **adversarial** 'confusions' of ML systems

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 invarnce 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-fward 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

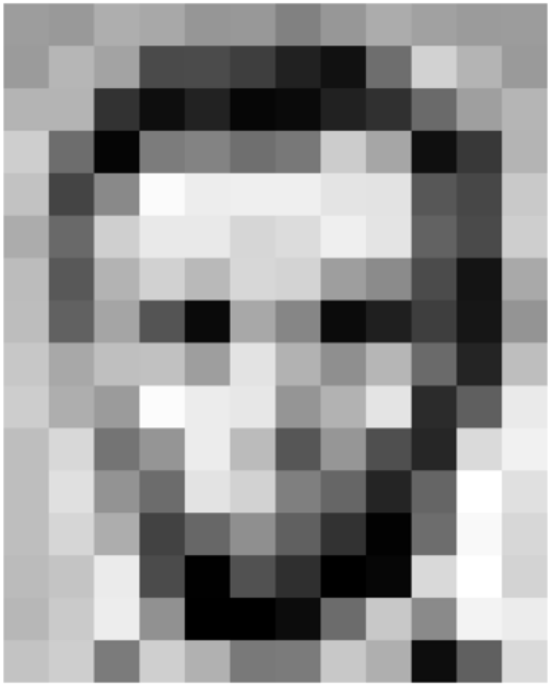
In deep learning, the two tasks are **coupled**:

- 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

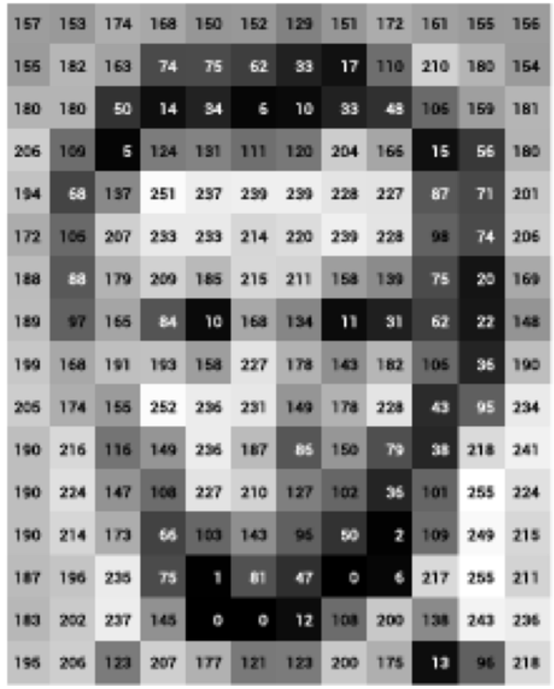
CNNs = Translating pixels to concepts

What you see



Input Image

What you both see



Input Image + values

What the computer "sees"

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	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	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	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	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixel intensity values
(“pix-el”=picture-element)

Levin *Image Processing & Computer Vision*

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?

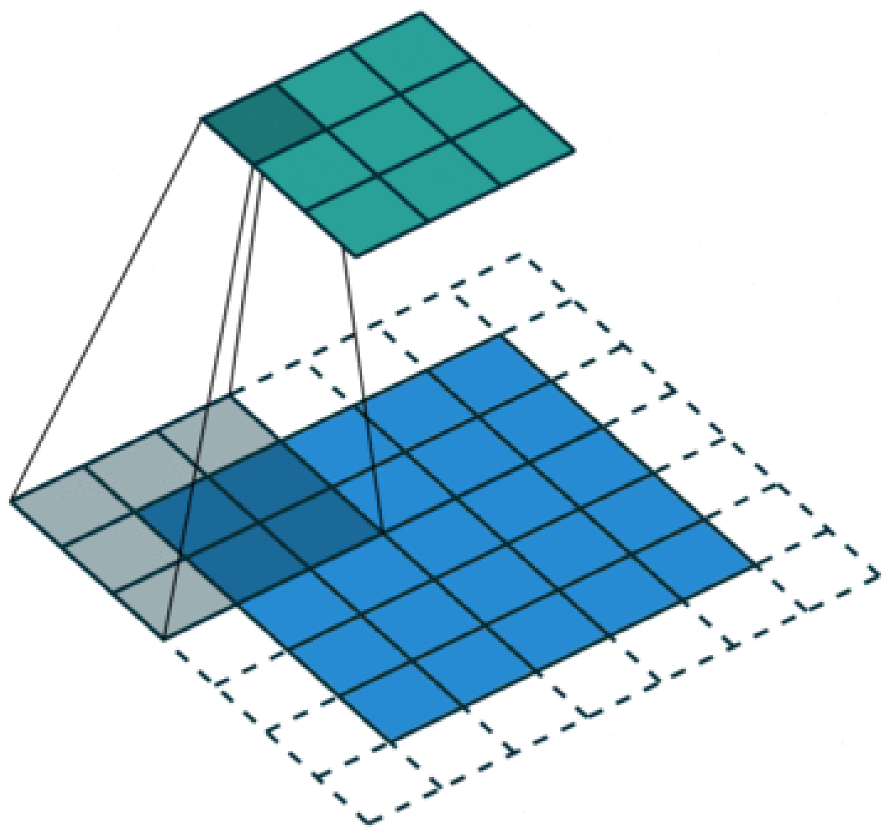
No. Instead: exploit image spatial structure. Learn patches. Build them up

<u>Convolutions:</u>
Spatial structure, local computation, shared parameters

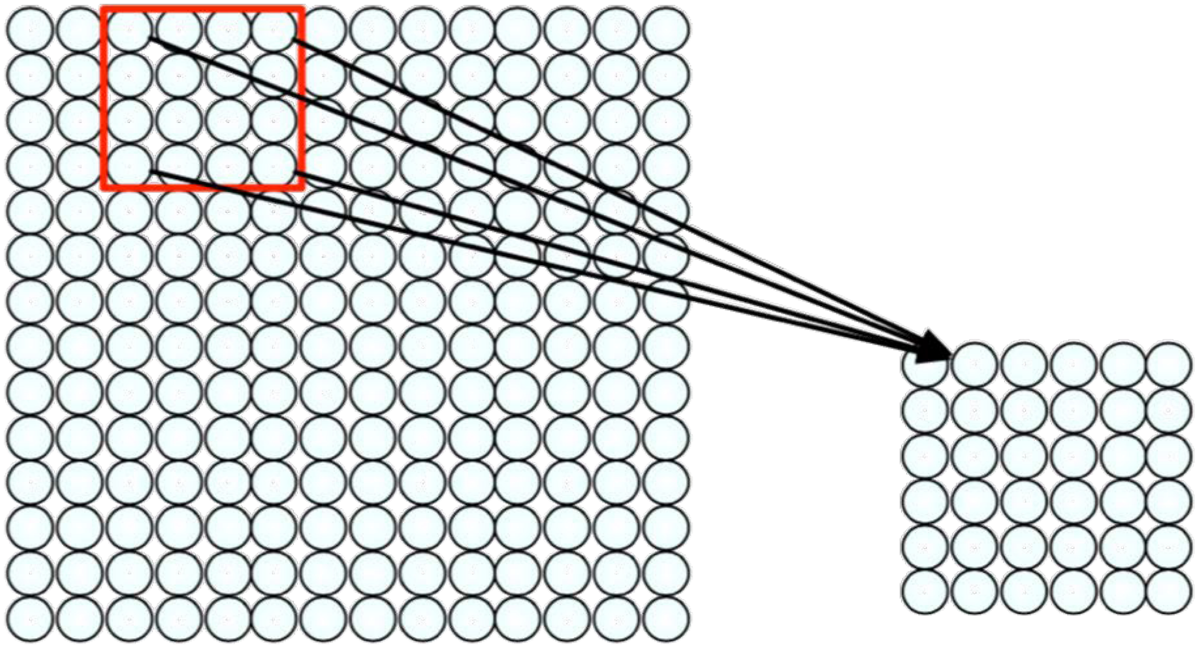
Key idea: re-use parameters

Convolution shares parameters

Example 3x3 convolution on a 5x5 image



Feature Extraction with Convolution

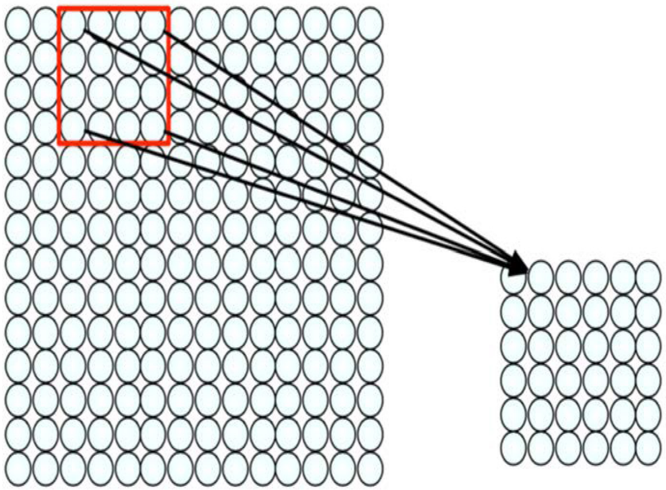


1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) Spatially share parameters of each filter

Feature Extraction with Convolution



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

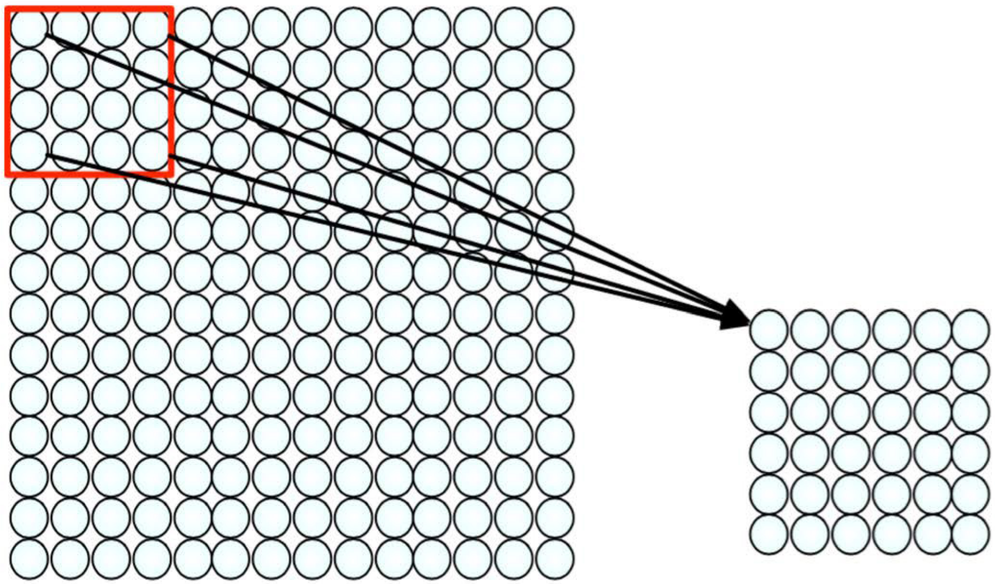
This "patchy" operation is **convolution**

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}

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

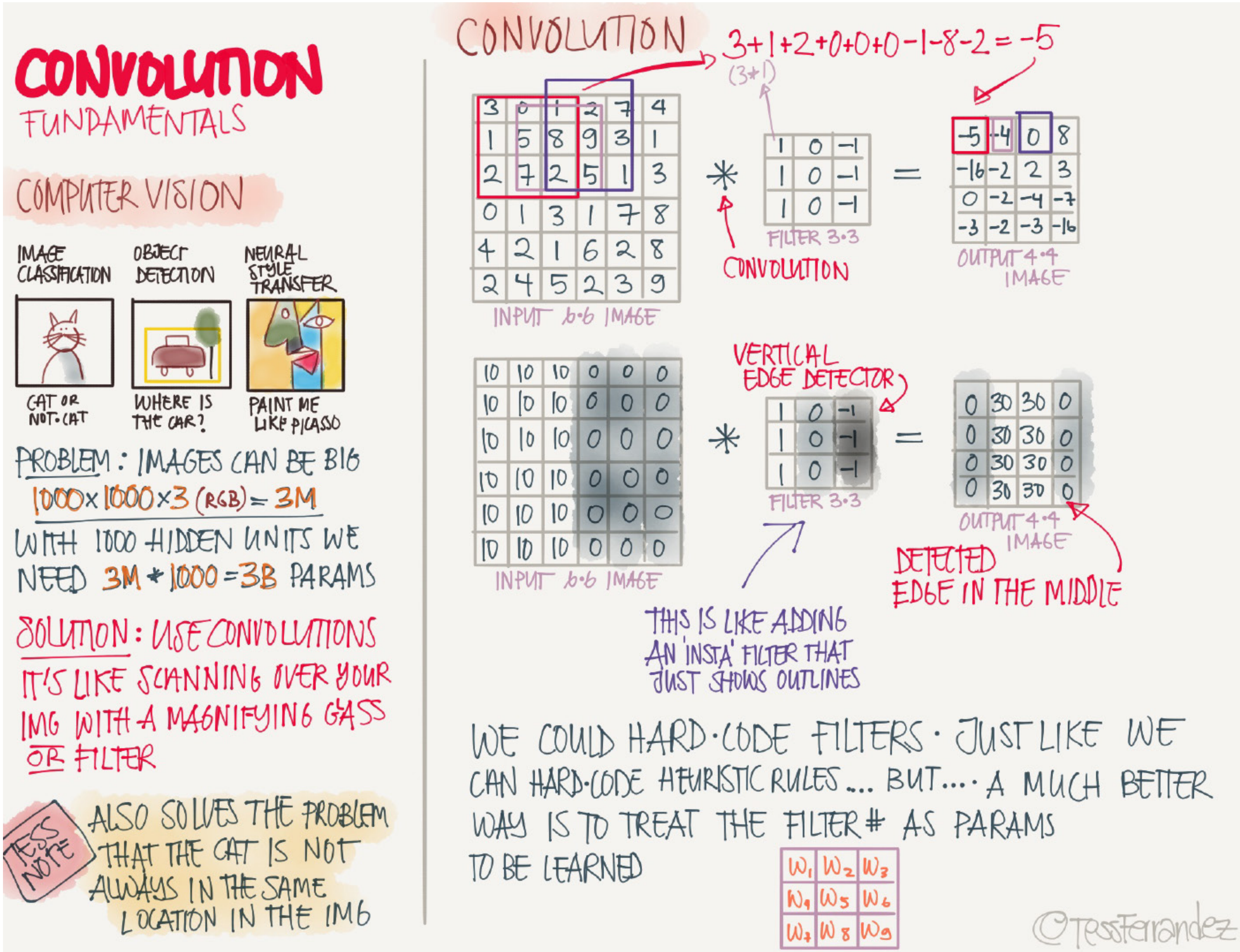
for neuron (p,q) in hidden layer

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

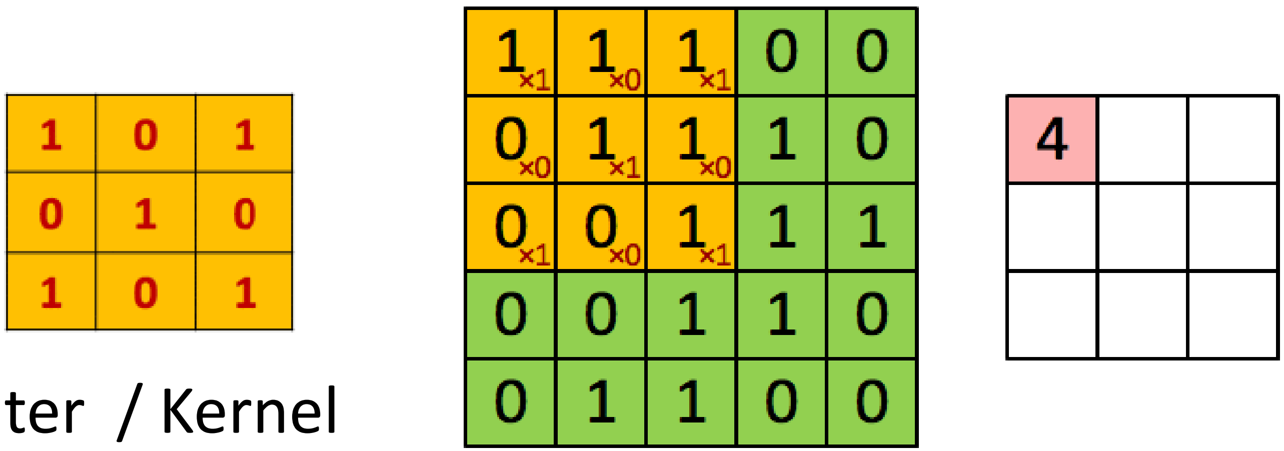
“Representations”

Filters extract Features

Convolution fundamentals



Convolution operation is element wise multiply and add



Filter / Kernel

Image

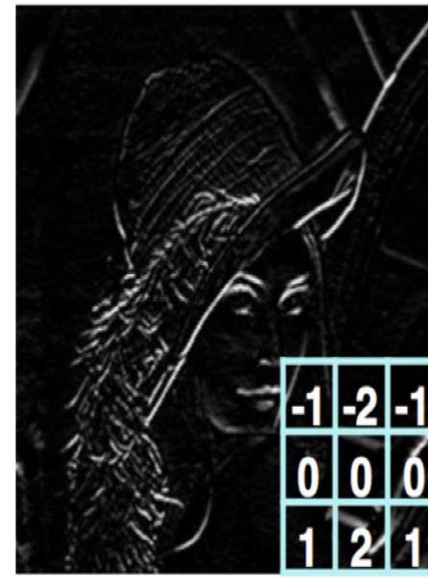
Convolved Feature

Producing Feature Maps

Original

Sharpen

Edge Detect



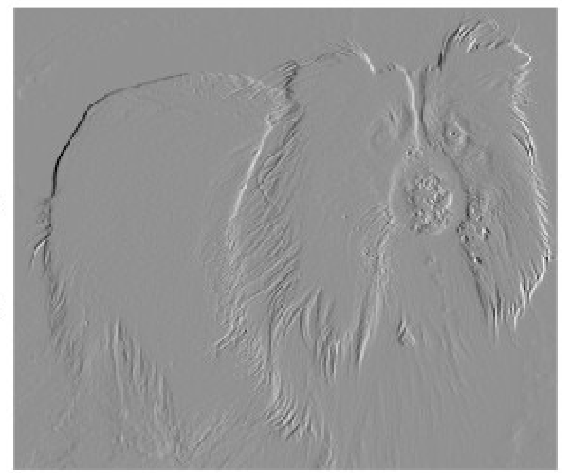
"Strong" Edge Detect

A simple pattern: Edges
How can we detect edges with a kernel?

Input

1	-1

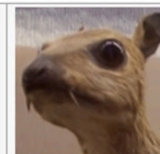
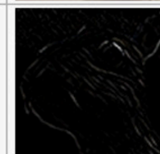
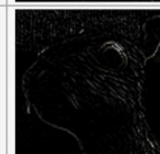
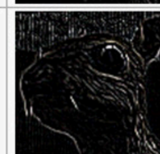
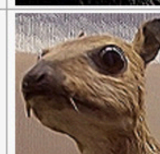
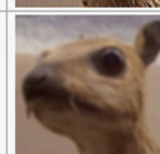
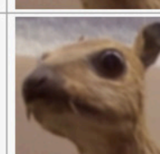
Filter



Output

(Goodfellow 2016)

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

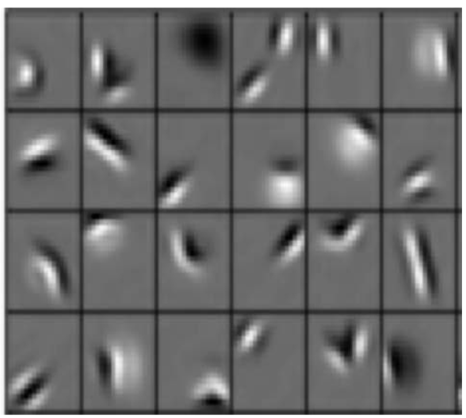
Key idea:

learn hierarchy of features

directly from the data

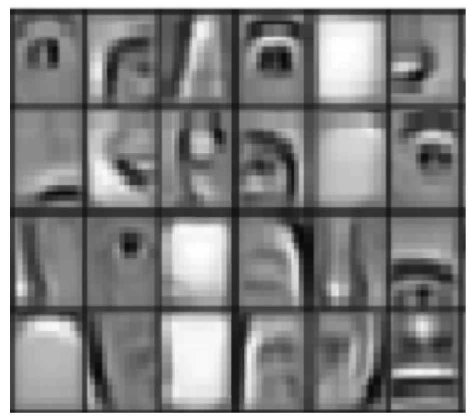
(rather than hand-engineering them)

Low level features



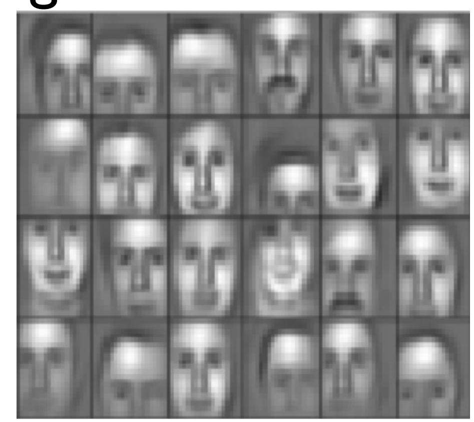
Edges, dark spots

Mid level features



Eyes, ears,nose

High level features



Facial structure

Lee+ *ICML* 2009

Representation Learning in Deep CNNs

Low level features	Mid level features	High level features

Edges, dark spots	Eyes, ears,nose	Facial structure
Conv Layer 1	Conv Layer 2	Conv Layer 3

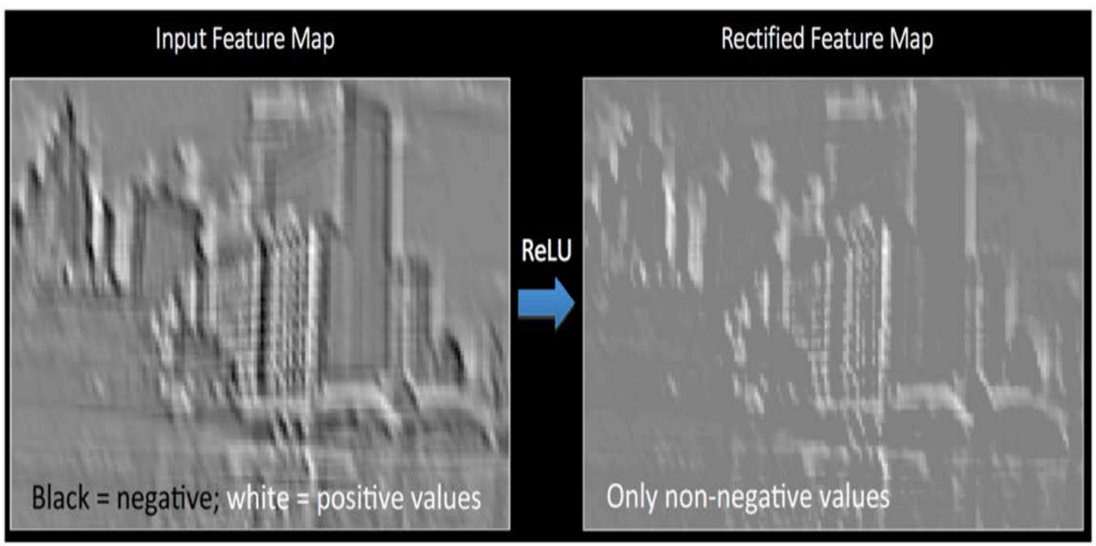
Lee+ *ICML* 2009

Detection:

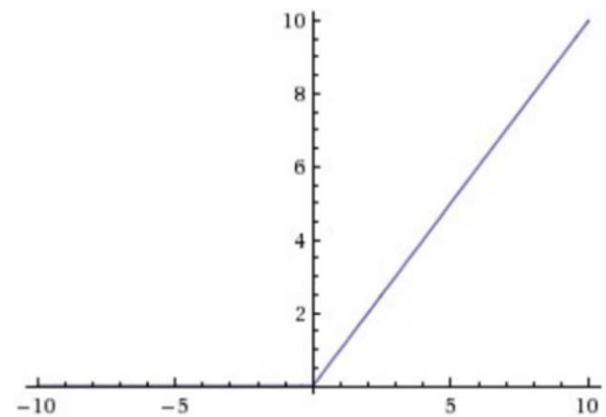
Non-Linearities

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)

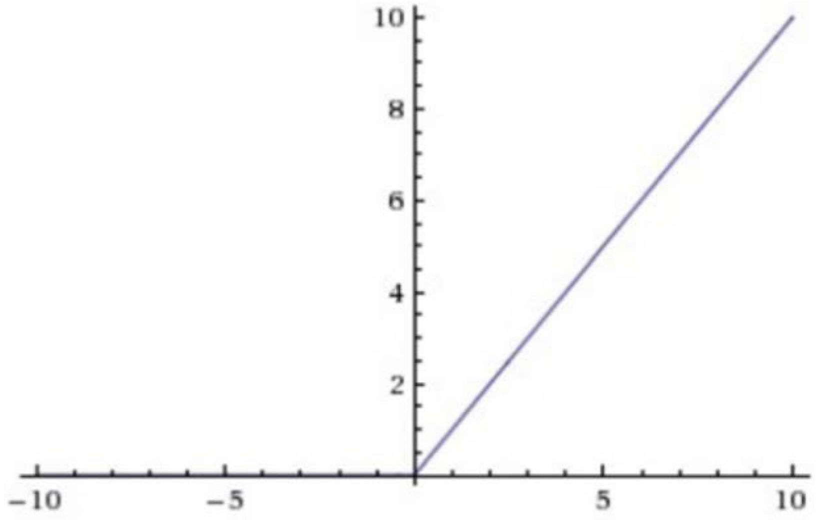


$$g(z) = \max(0, z)$$

```
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)
```



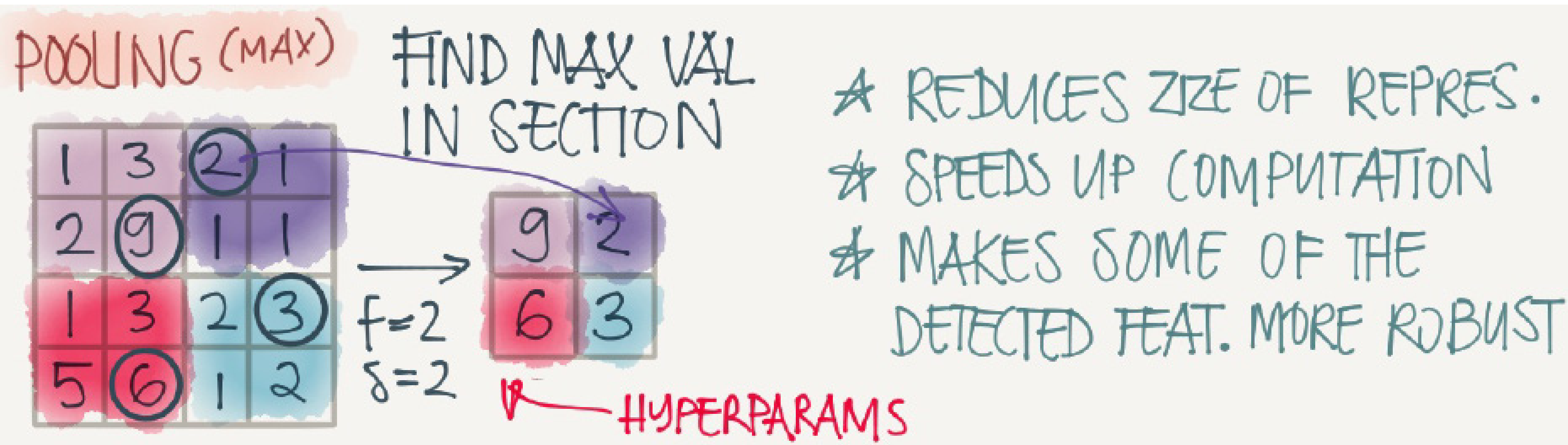
f(x) = max(0, x)

When will we backpropagate through this?

Once it “dies” what happens to it?

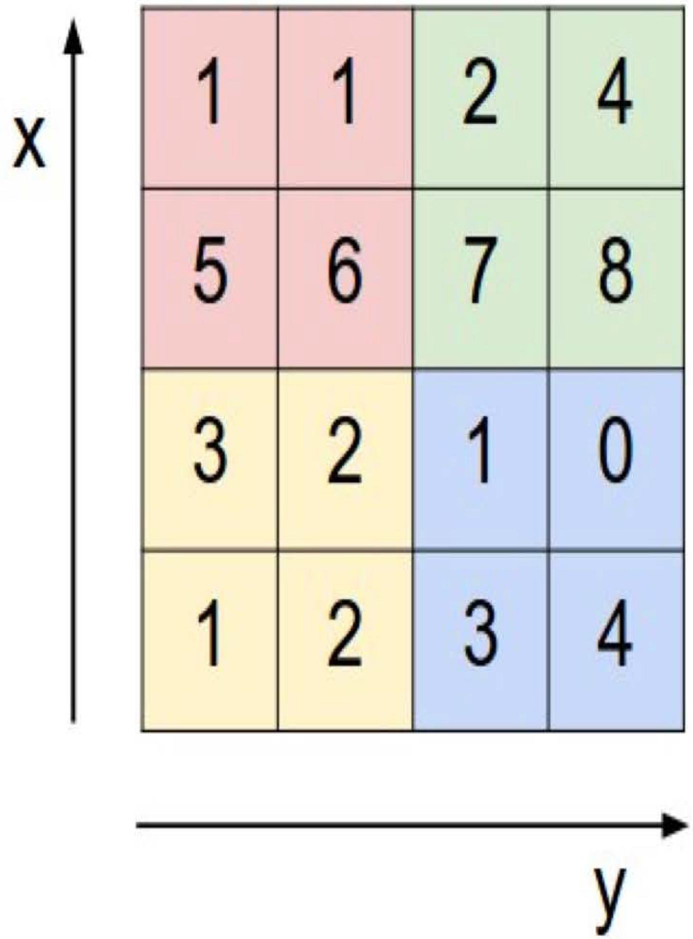
Pooling layers:
Positional invariance

Why Pooling



POOLING (MAX)
FIND MAX VAL IN SECTION
1 3 2 1
2 9 1 1
1 3 2 3
5 6 1 2
f=2
s=2
9 2
6 3
HYPERPARAMS
* REDUCES ZIZE 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.MaxPool2D(
    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

```
x = tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME')
```

Output

Input

Pooling Neighborhood

[batch, height, width, channels]

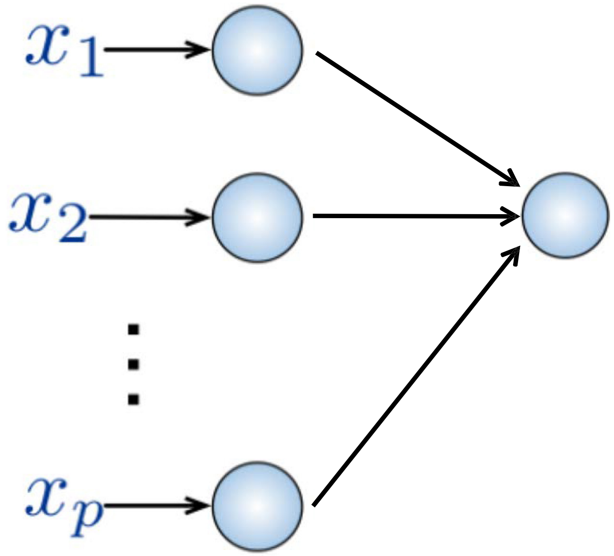
Batch H W

Input channel

<u>Classification</u>:
fully-connected layers

Fully Connected Neural Network

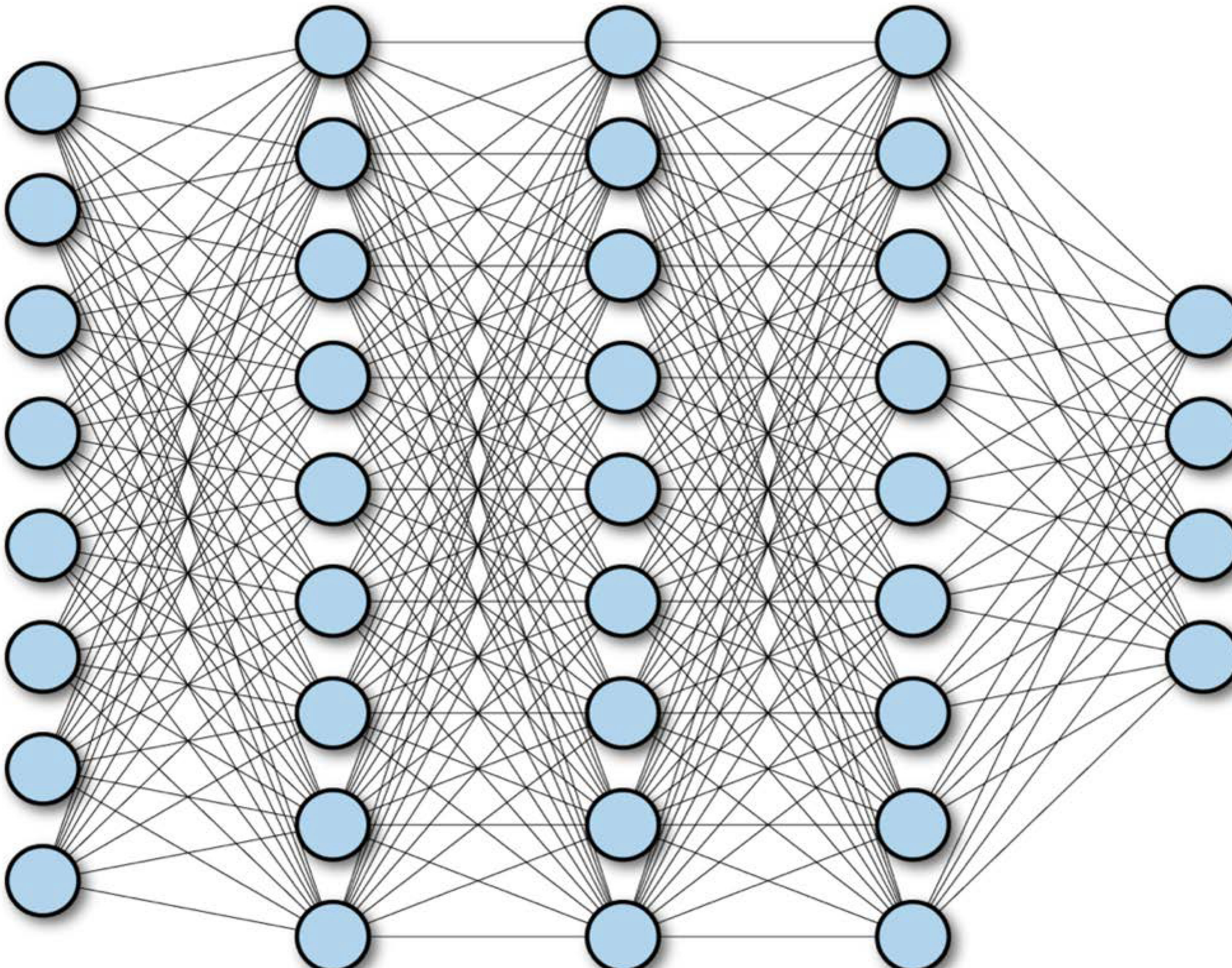
Input:
- 2D image
- Vector of pixel values



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

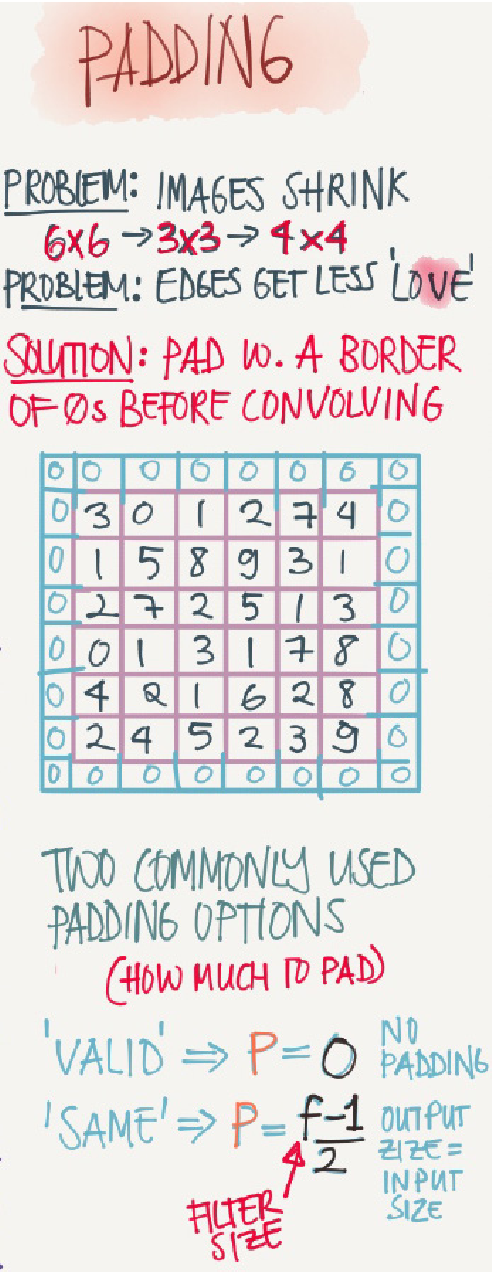
Key idea: Use **<u>spatial structure</u>** in input to inform architecture of the network

Fully Connected Neural Network

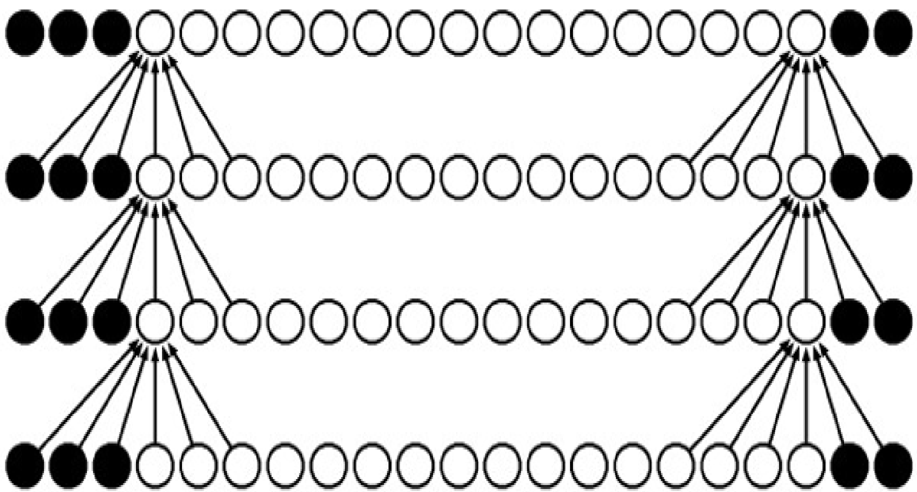


Edge cases (literally):
Practical issues of convolutions

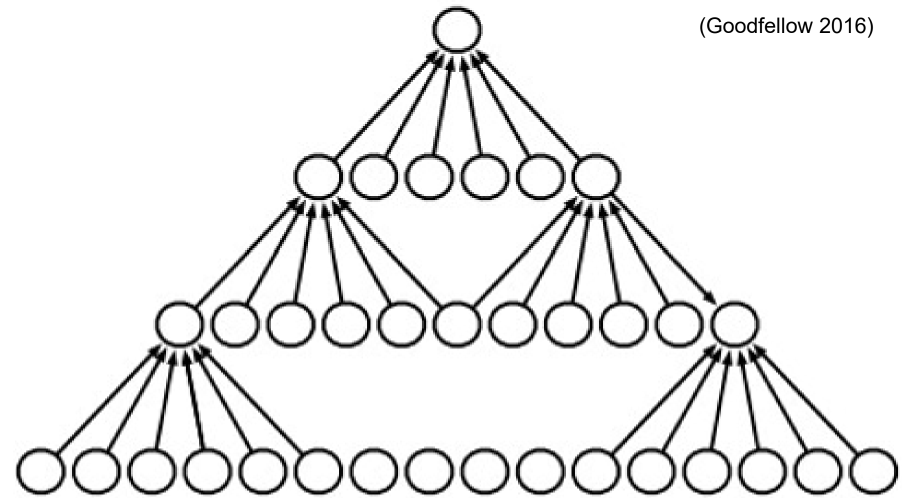
Padding


PADDING
PROBLEM: IMAGES SHRINK
6x6 → 3x3 → 4x4
PROBLEM: EDGES GET LESS 'LOVE'
SOLUTION: PAD W. A BORDER OF ØS BEFORE CONVOLVING
TWO COMMONLY USED PADDING OPTIONS
(HOW MUCH TO PAD)
'VALID' ⇒ P = 0
NO PADDING
'SAME' ⇒ P = (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

• **Valid-only convolution**: output only when entire kernel contained in input (shrinks output)

• **Full convolution**: zero pad input so output is produced whenever an output value contains at least one input value (expands output)

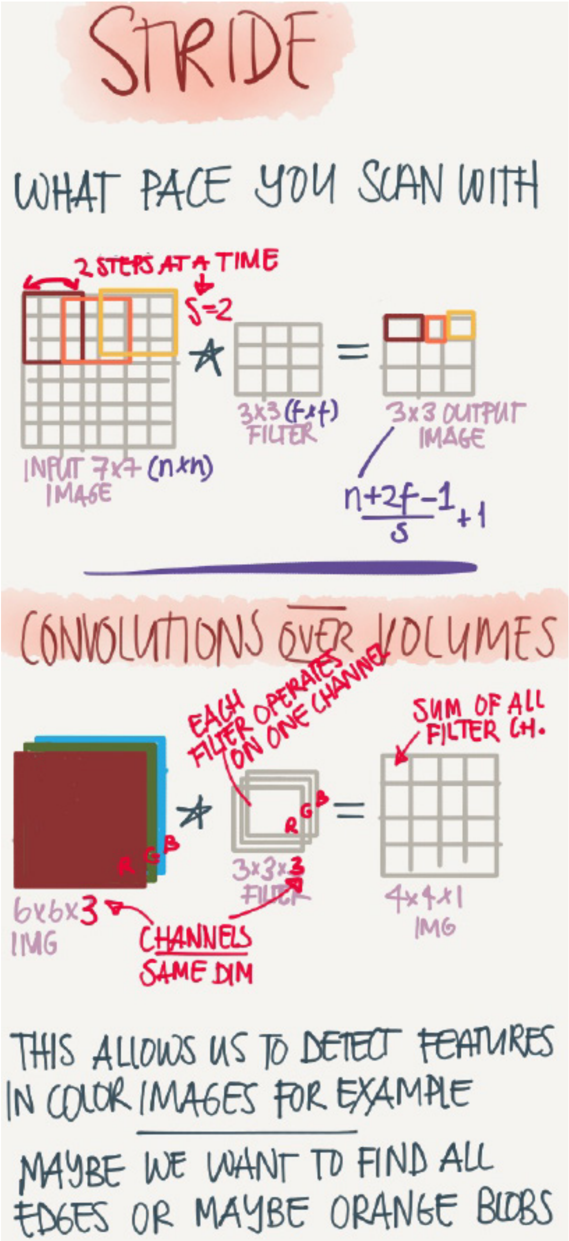
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

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

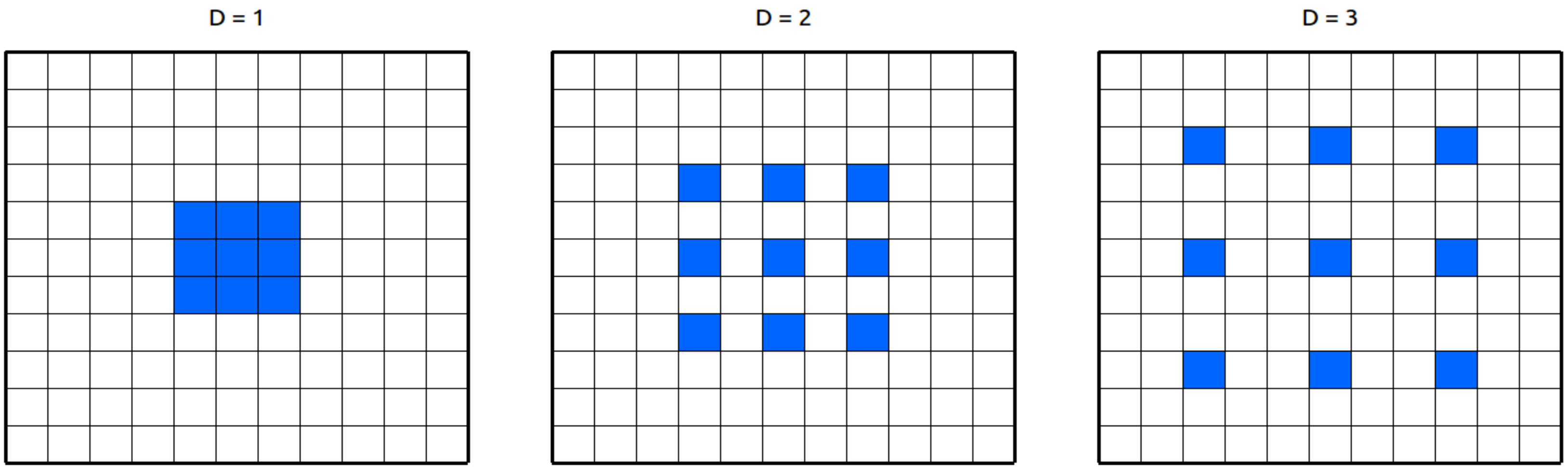
- TF convolution operator takes stride and zero fill option as parameters
- Stride is distance between kernel applications in each dimension
- Padding can be SAME or VALID

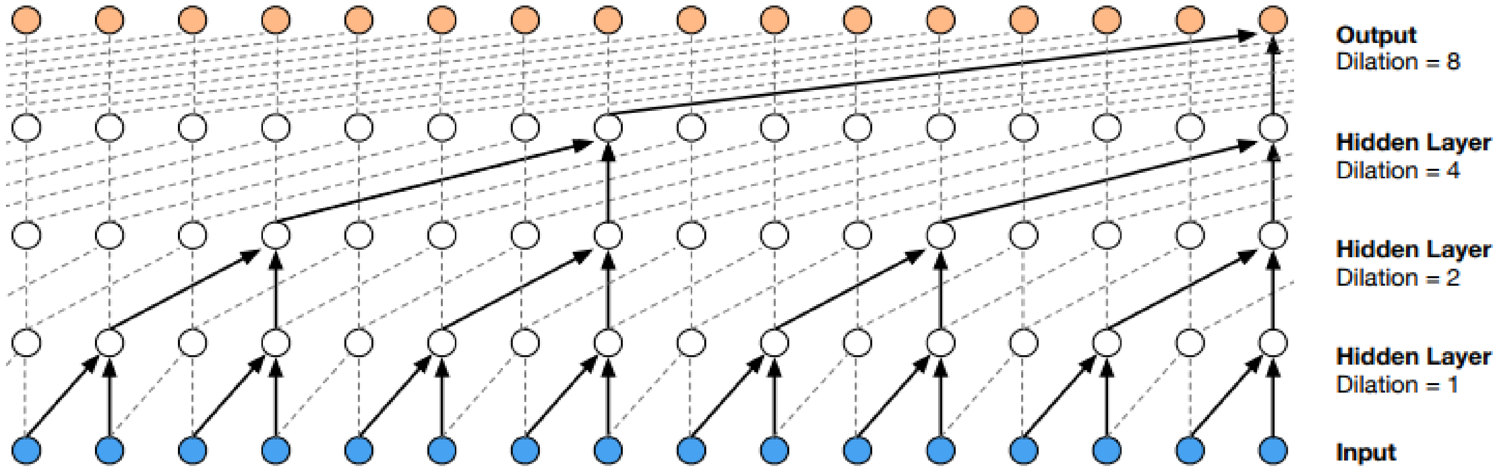
Edge cases (literally):
Practical issues of convolutions

Stride


STRIDE
WHAT PACE YOU SCAN WITH
2 STEPS AT A TIME
S=2
3x3 (f x f) FILTER
3x3 OUTPUT IMAGE
INPUT 7x7 (n x n) IMAGE
n+2f-1 / S + 1
CONVOLUTIONS OVER VOLUMES
EACH FILTER OPERATES ON ONE CHANNEL
SUM OF ALL FILTER CH.
R G B
6x6x3 IMG
3x3x3 FILTER
4x4x1 IMG
CHANNELS SAME DIM
THIS ALLOWS US TO DETECT FEATURES IN COLOR IMAGES FOR EXAMPLE
MAYBE WE WANT TO FIND ALL EDGES OR MAYBE ORANGE BLOBS

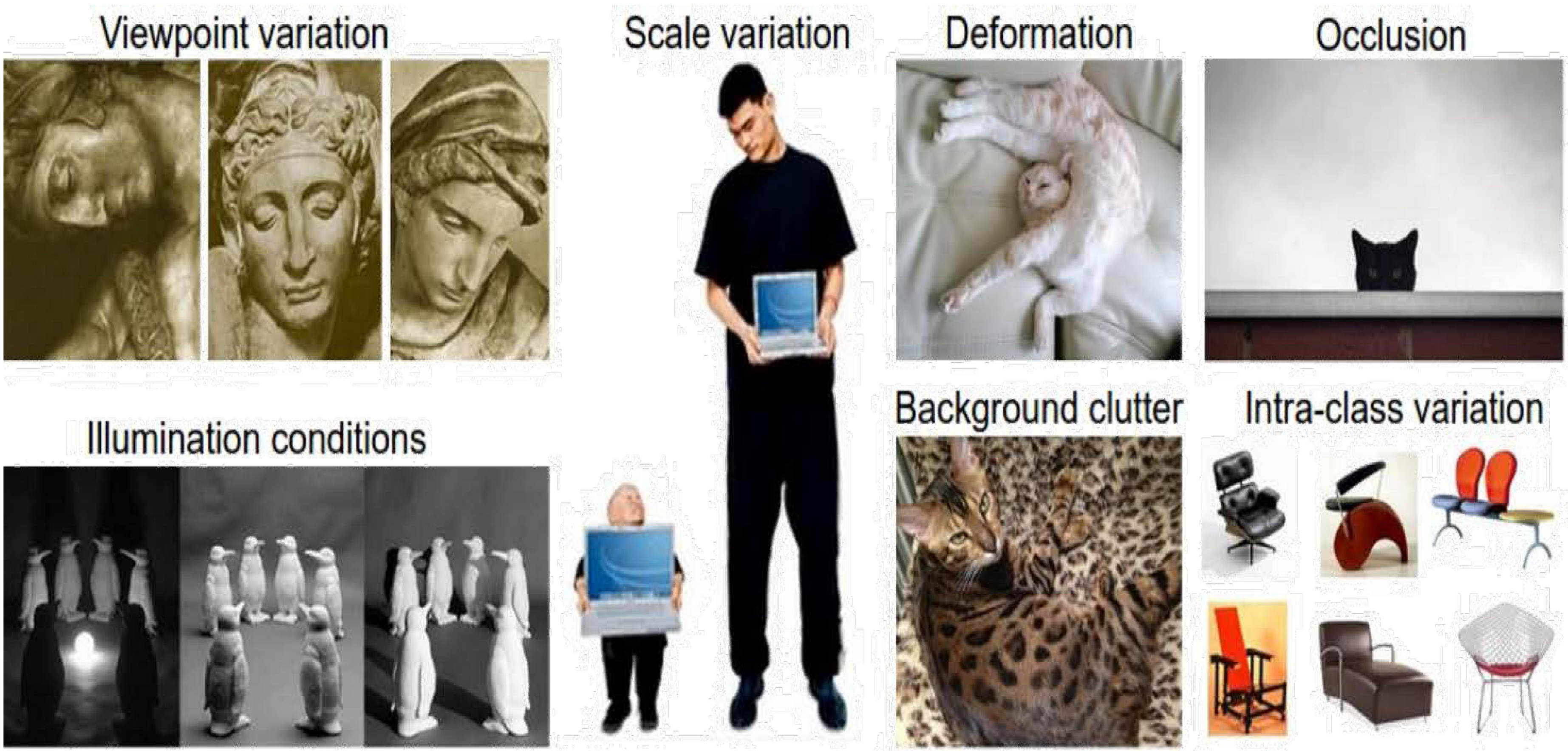
Dilated Convolution





<u>Real-world Feature Invariance:</u>
Data augmentation

Feature invariance to perturbation is hard



X or X?

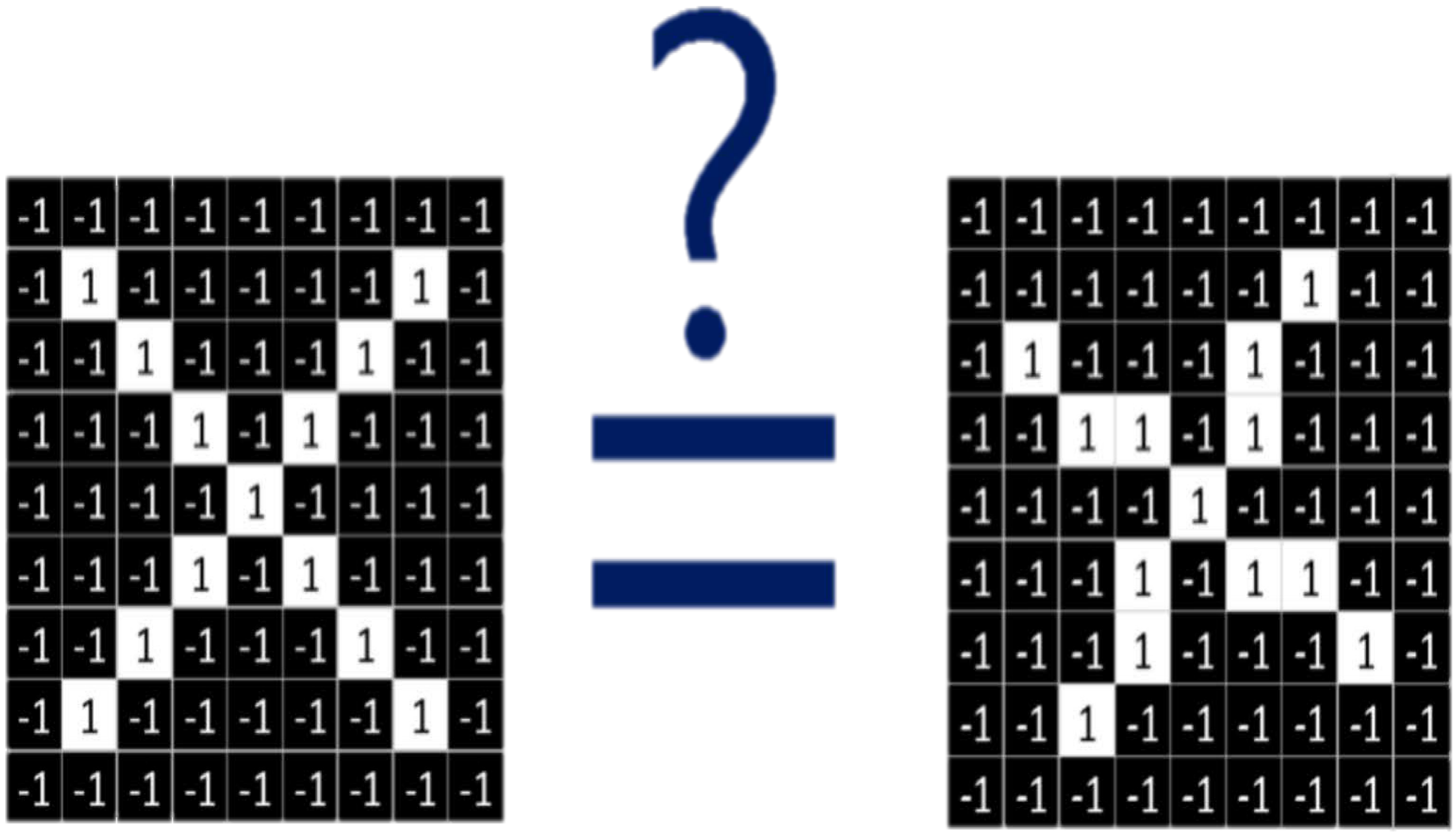
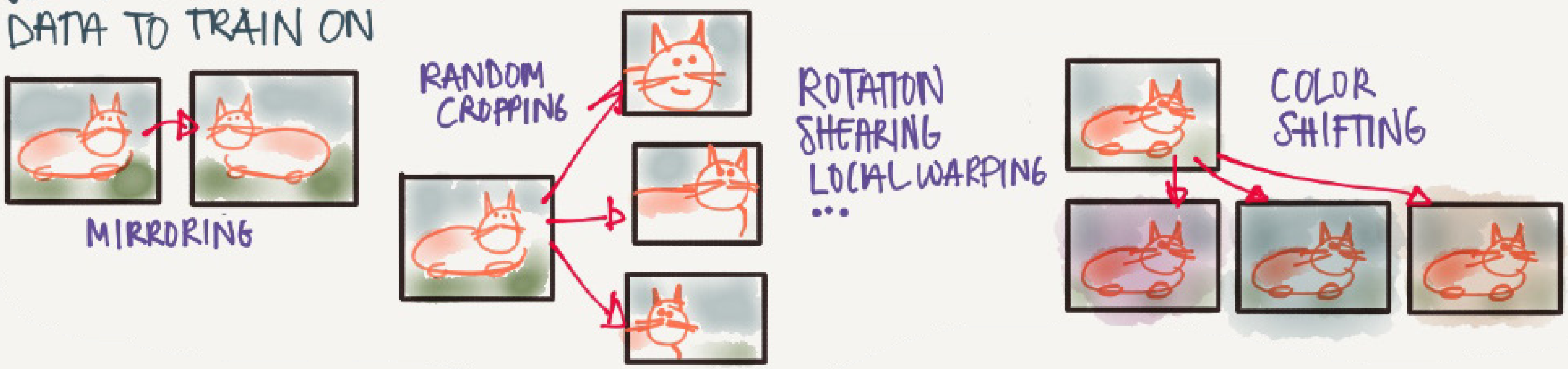


Image is represented as matrix of pixel values… and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

Rohrer *How do CNNs work?*

DATA AUGMENTATION

WE ALMOST ALWAYS NEED MORE DATA TO TRAIN ON



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:

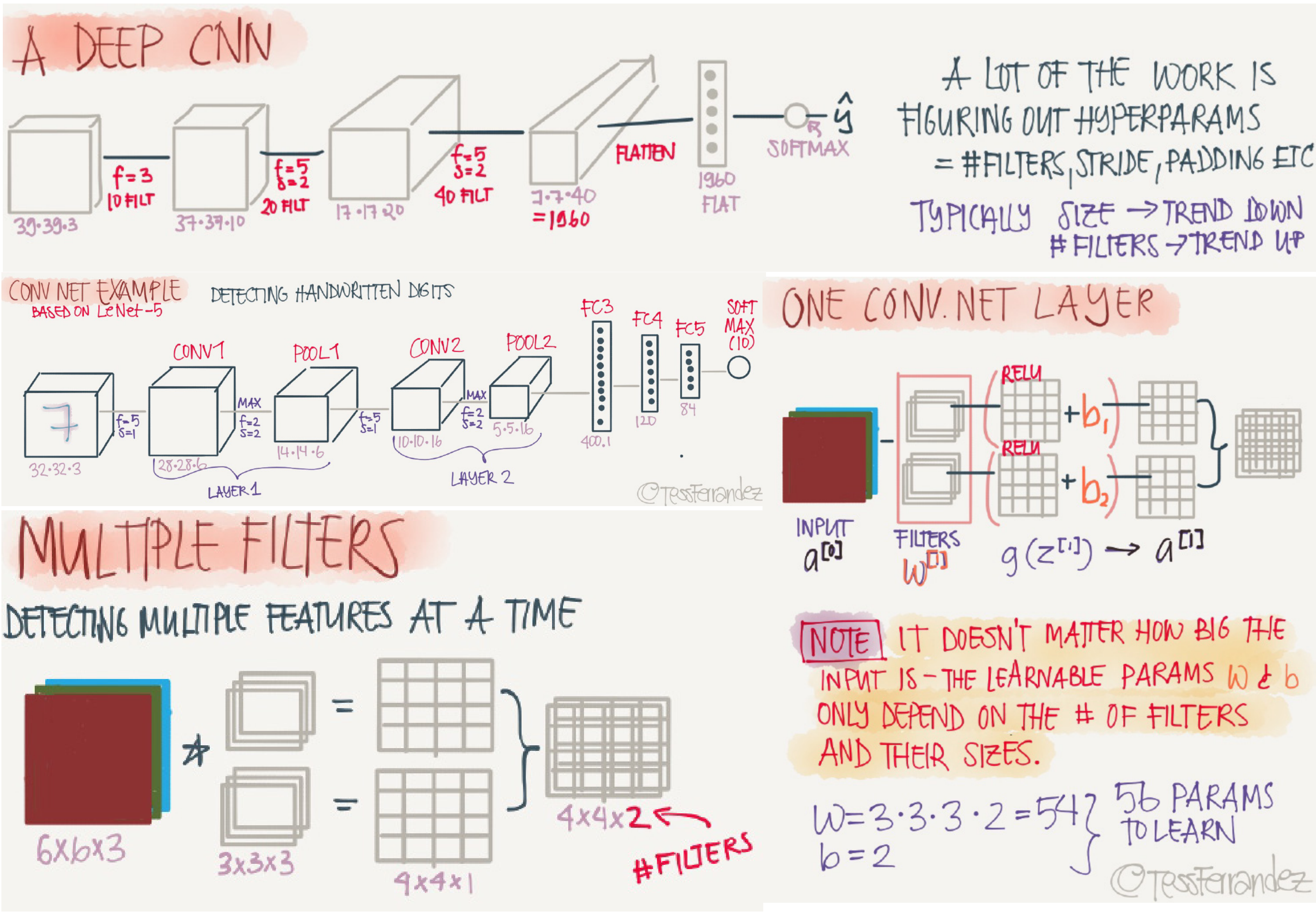
- Learn a ton of features (millions) from the bottom up
- Learn the convolutional filters, rather than pre-computing them

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 invarnce 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-fward 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



Putting it all together

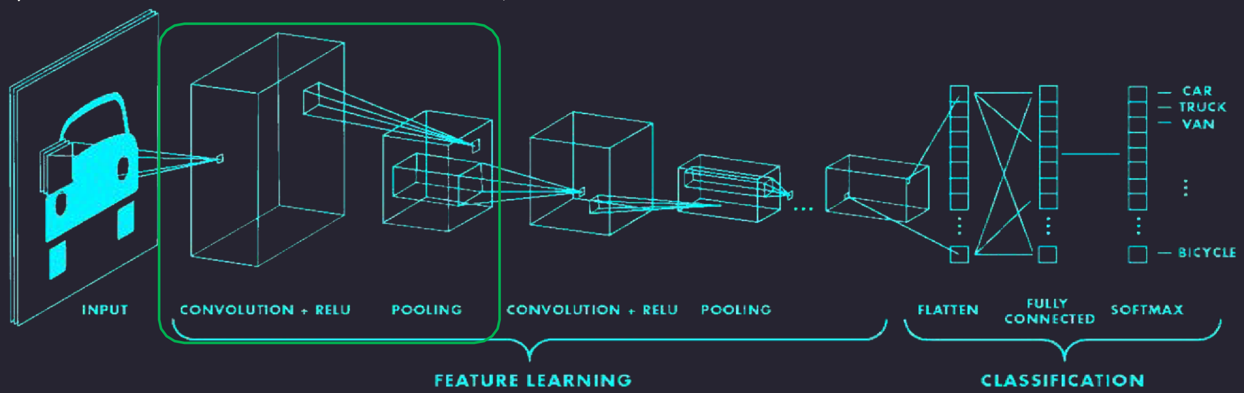
```
import tensorflow as tf

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

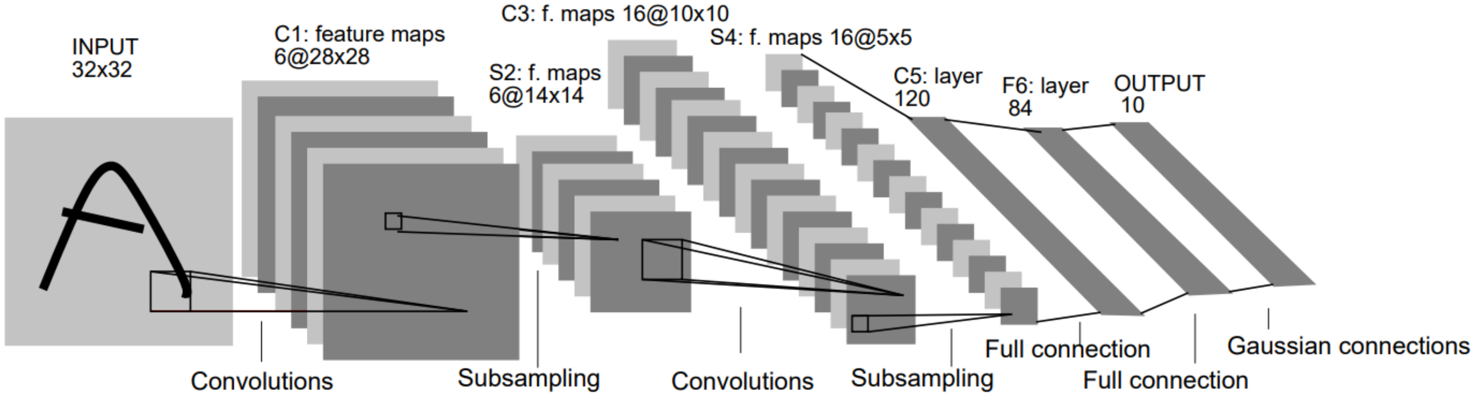
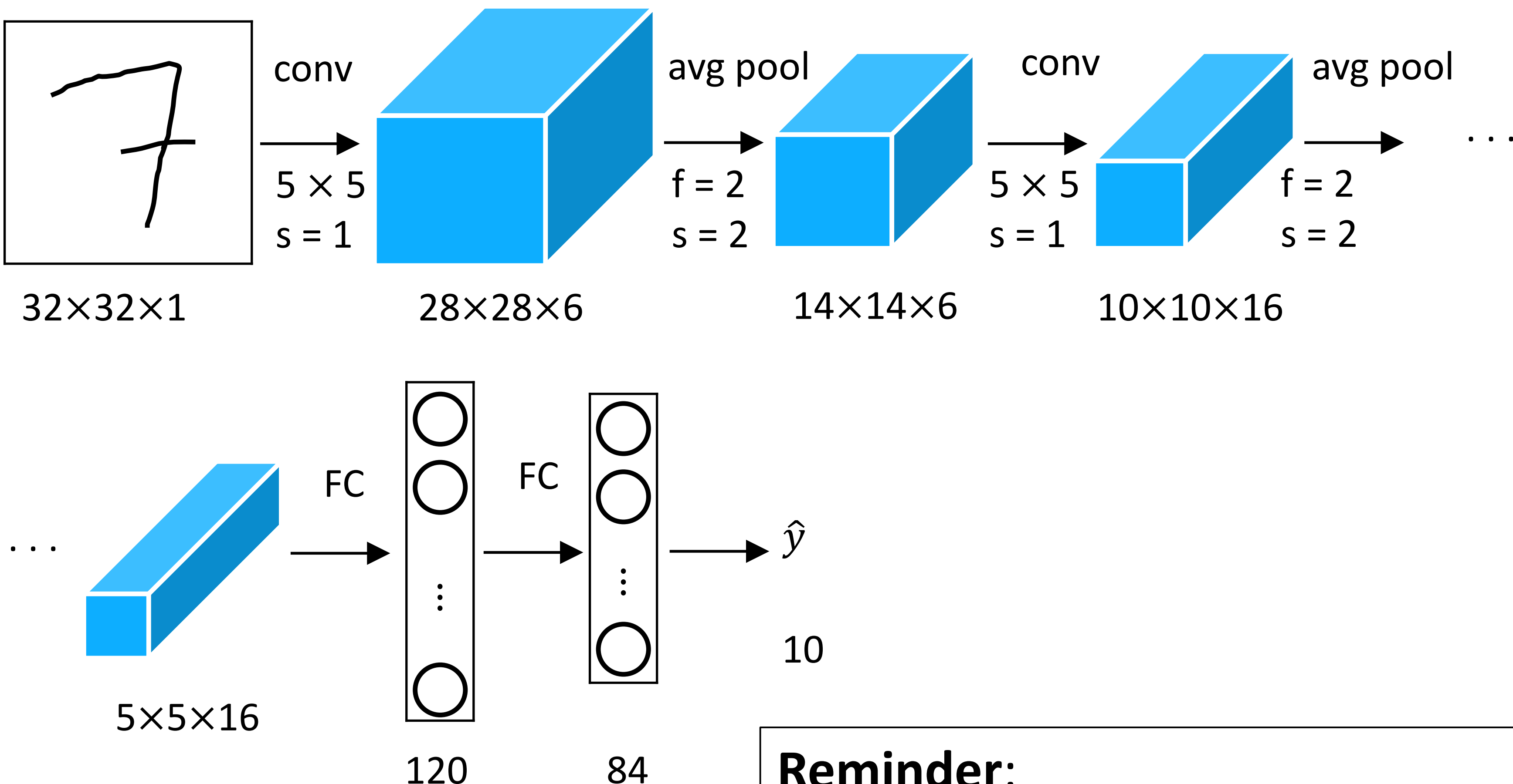


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

[**LeCun et al.**, **1998**]

LeNet-5



Reminder:
Output size = (N+2P-F)/stride + 1

This slide is taken from Andrew Ng

[**LeCun et al.**, **1998**]

LeNet-5

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

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

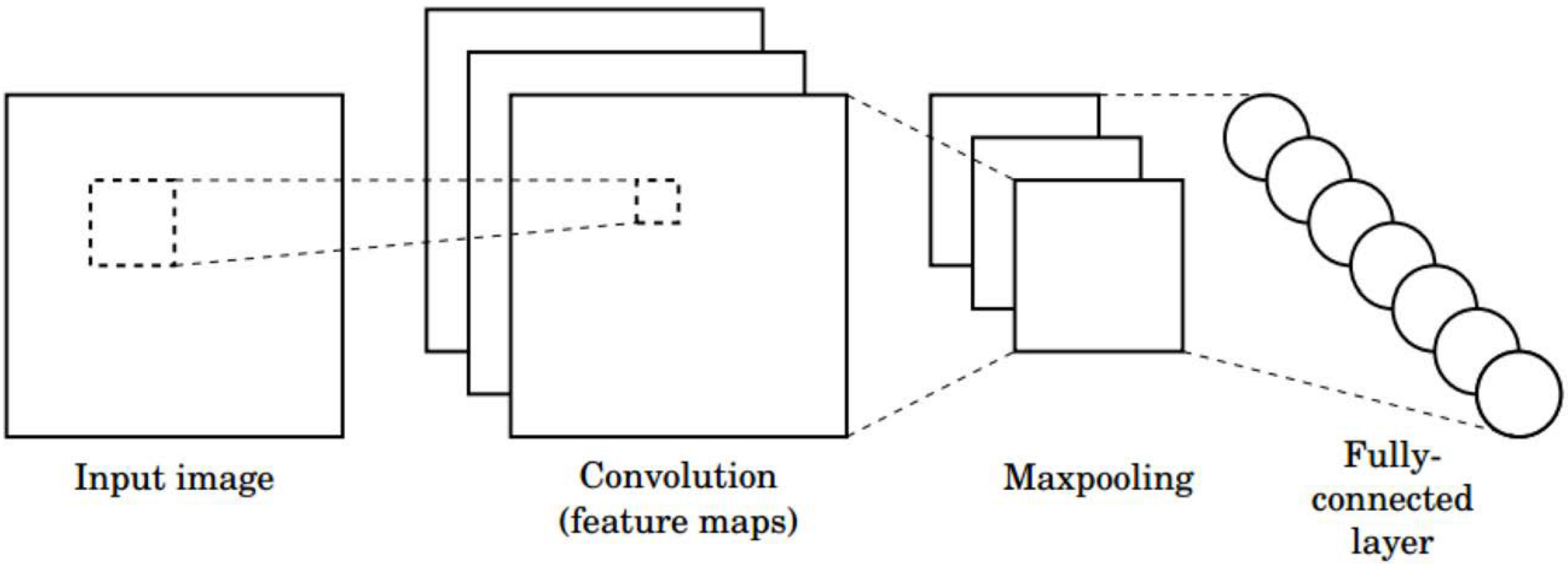
[**LeCun et al.**, **1998**]

Backpropagation of convolution

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

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

CNNs for Classification



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

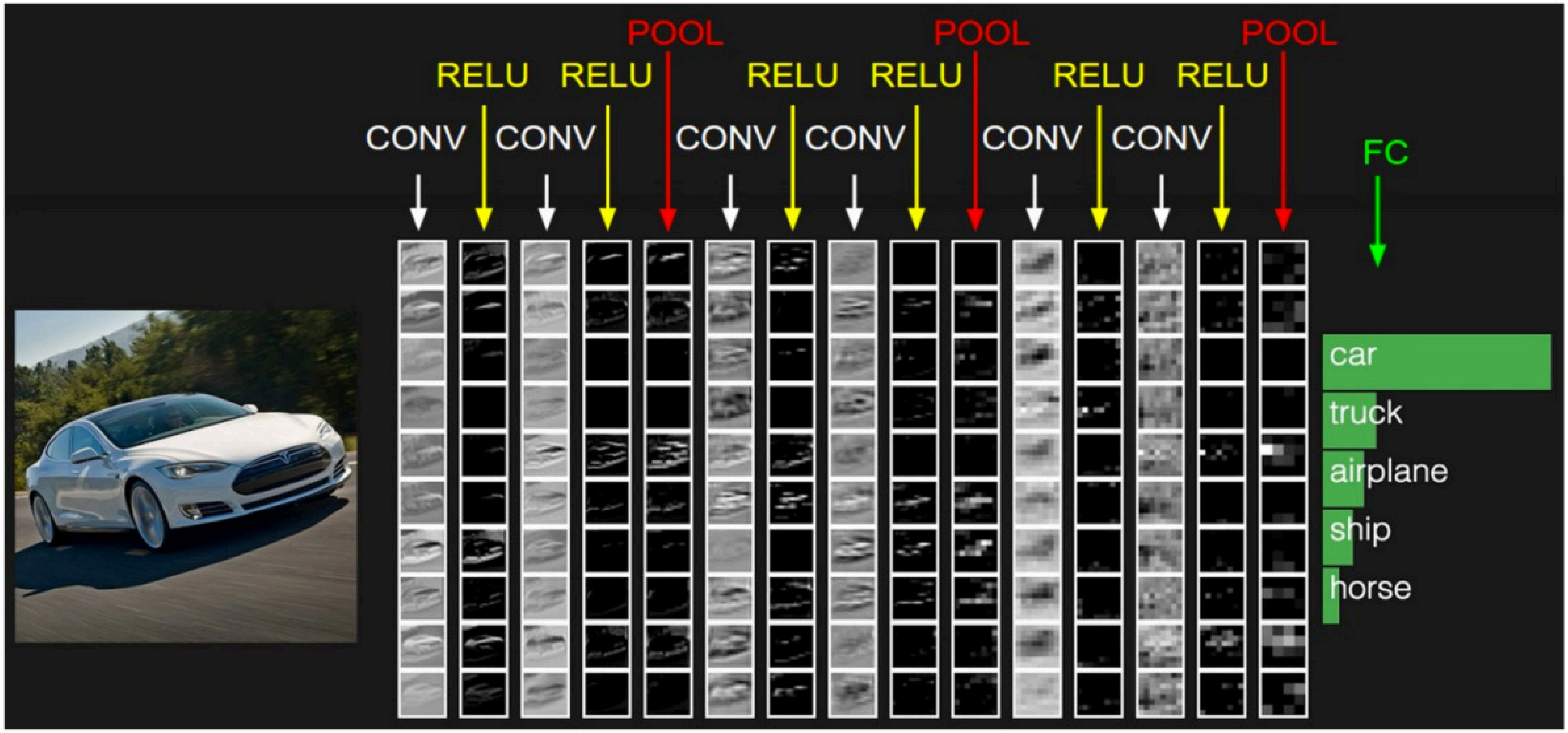
Train model with image data.
Learn weights of filters in convolutional layers.

```
tf.keras.layers.Conv2D
```

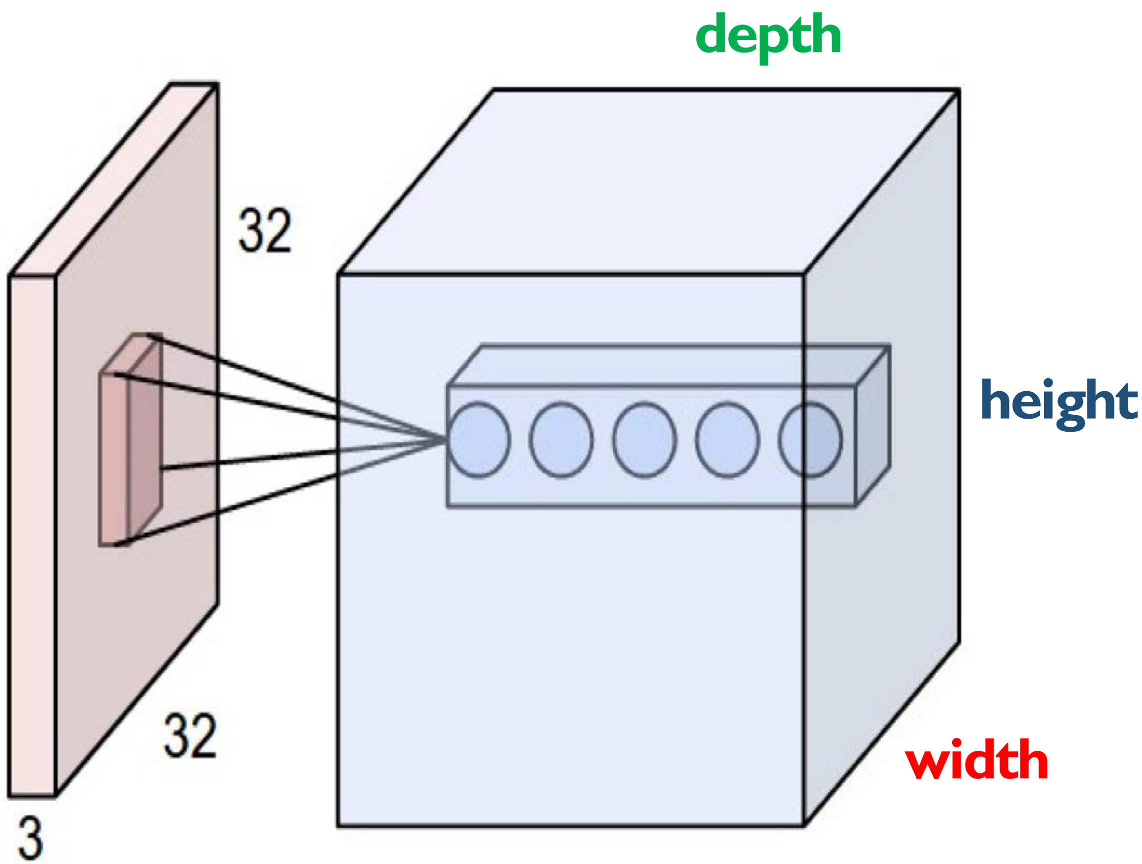
```
tf.keras.activations.*
```

```
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:

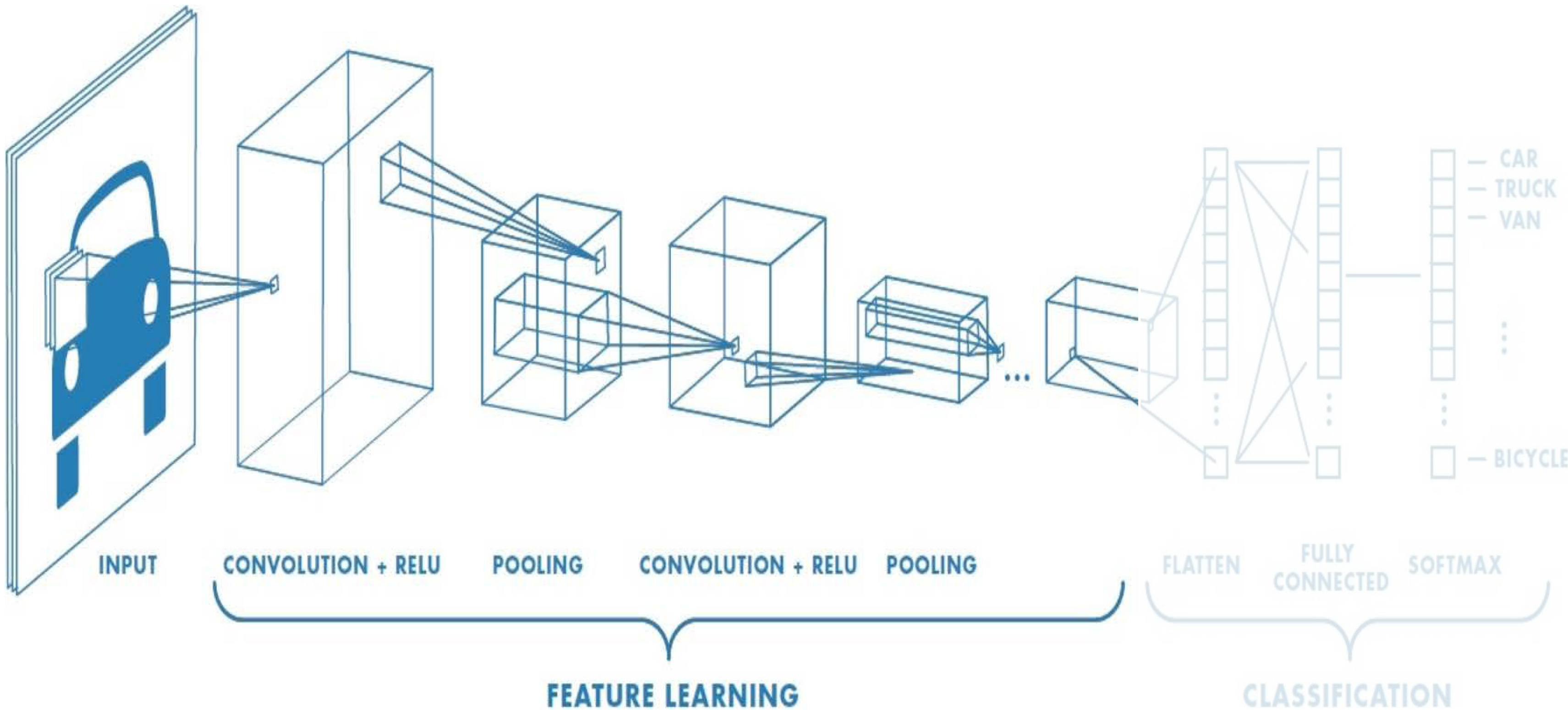
Filter step size

Receptive Field:

Locations in input image that a node is path connected to

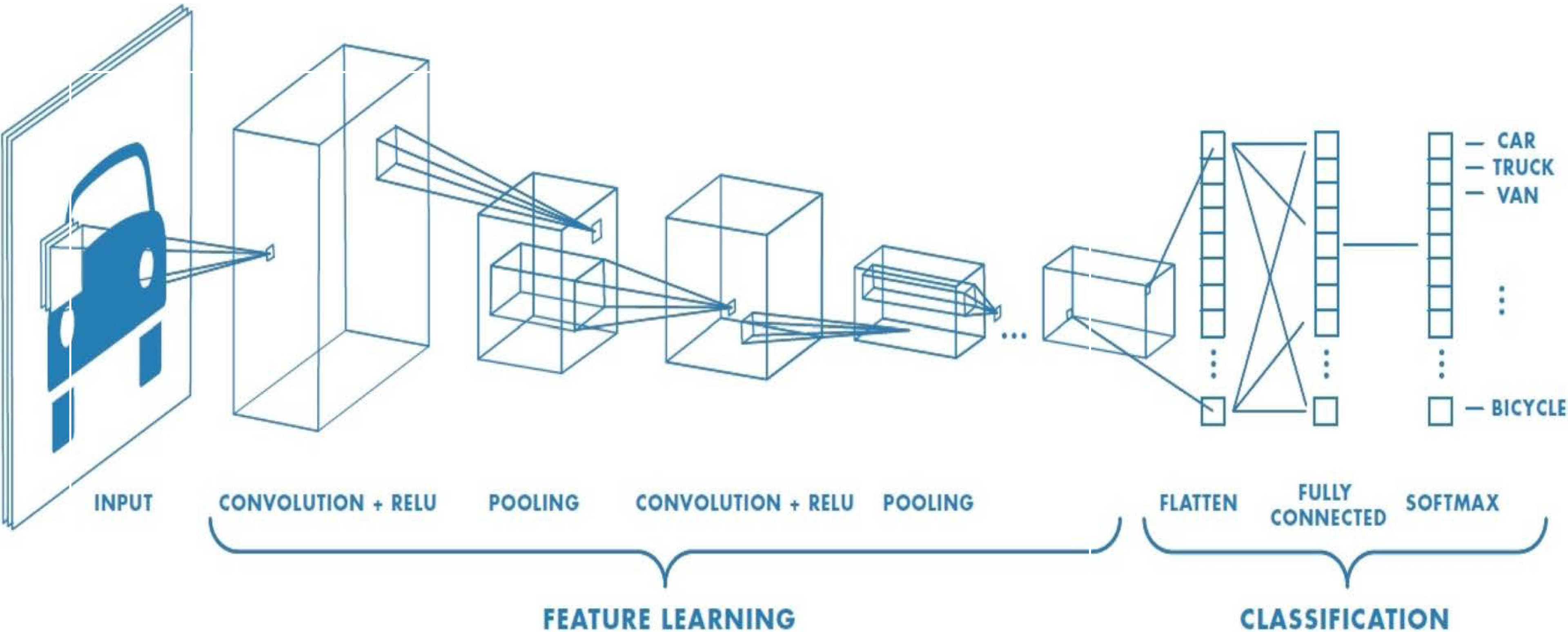
```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

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

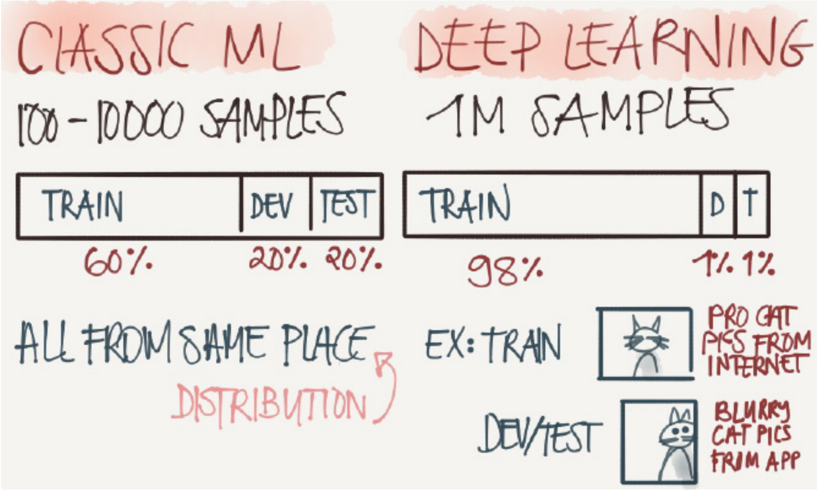


$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

- 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

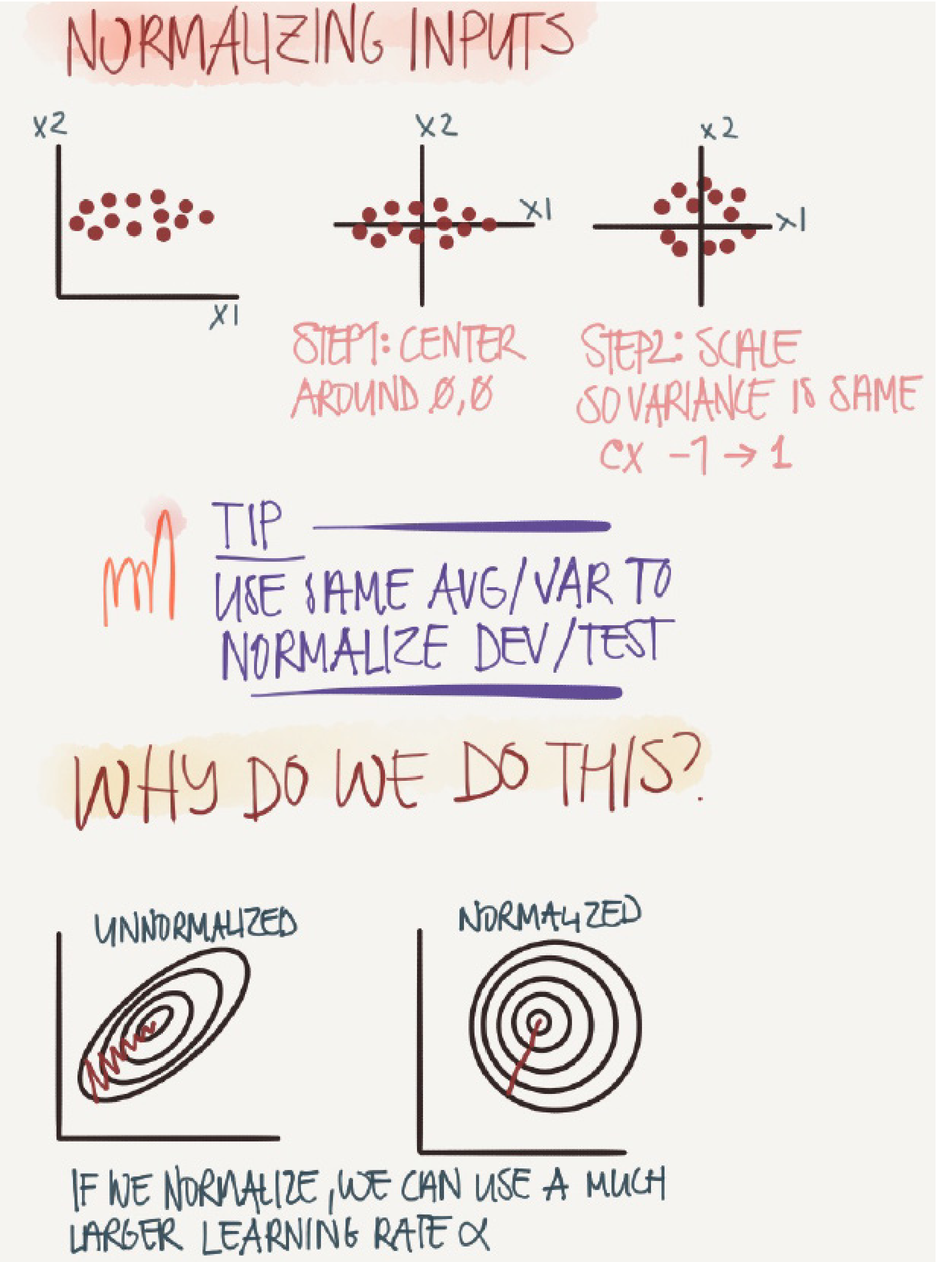
The art of CNN training

Foundations of CNN training



- **Needs lots of data for training**

Normalization matters



Vanishing / exploding gradients

DEALING WITH VANISHING/EXPLODING GRADIENTS

EX: DEEP NW (L LAYERS)

$$\hat{y} = \underbrace{W^{[L-1]} W^{[L-2]} \cdots W^{[1]}}\, x + b$$

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

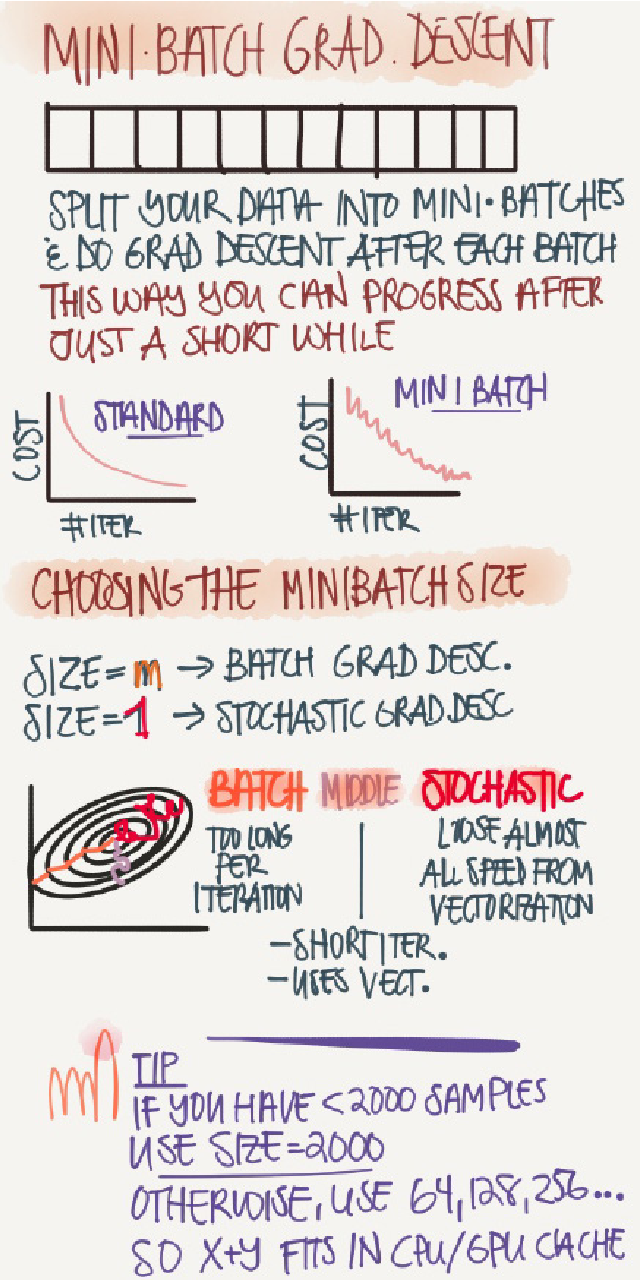
$w^{[l]} = rand * \sqrt{\frac{2}{n^{[l-1]}}}$ (FOR RELU)

#inputs

XAVIER $\sqrt{\frac{1}{n^{[l-1]}}}$ (FOR TANH)

SETS THE VARIANCE

Mini-batch gradient descent



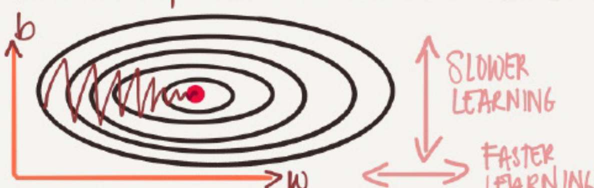
Optimizing training

GRADIENT DESCENT W. MOMENTUM

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



NORMALIZE GRADIENT USING A MOVING AVG.

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

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

$$w = w - \alpha \frac{dw}{\sqrt{S_{dw}}} \qquad b = b - \alpha \frac{db}{\sqrt{S_{db}}}$$

ADAM OPTIMIZATION

COMBO OF GD W MOMENTUM & RMSProp

LEARNING RATE DECAY

IDEA: USE A LARGE α 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: $\alpha = \frac{k}{\sqrt{\text{EPOCH}}} \alpha_0$

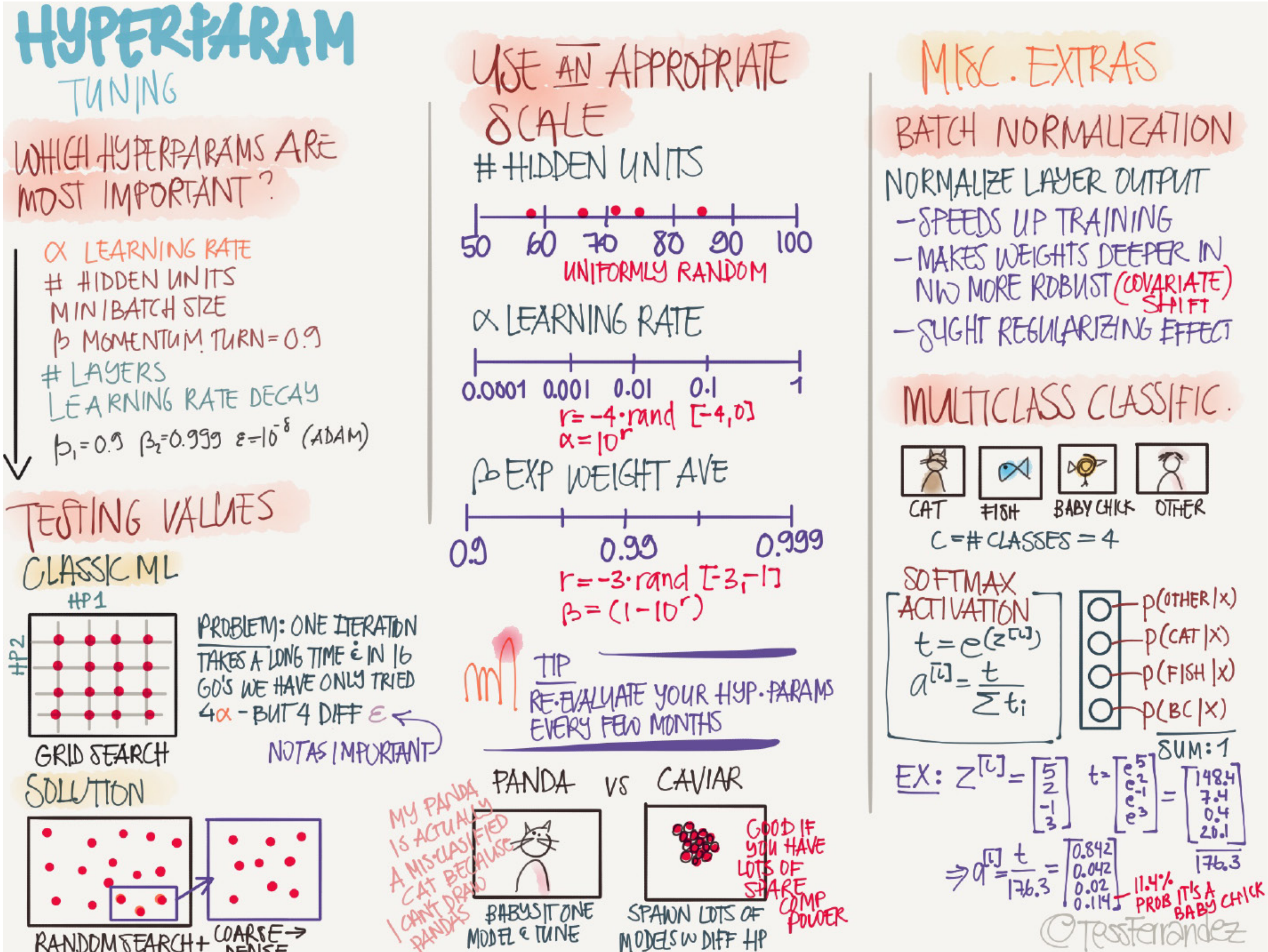
OPTION 4: $\alpha = \frac{k}{\sqrt{t}} \alpha_0$

OPTION 5: DISCRETE STAIRCASE

OPTION 6: MANUAL

EPOCH = 1 PASS THROUGH THE DATA

Hyperparameter Tuning



Train / Dev / Test sets

Importance of train/dev sets

TRAIN vs DEV/TEST MISMATCH

AVAILABLE DATA

200 k PRO CAT PICS FROM INTERNET

10 k BLURRY CAT PICS FROM APP — WHAT WE CARE ABT

HOW DO WE SPLIT → TRAIN/DEV/TEST?

OPTION 1: SHUFFLE ALL

205 k (TRAIN)	D	T

2.5 k

PROBLEM: DEV/TEST IS NOW MOSTLY WEB IMG (NOT REPRES. OF END SCENARIO)

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

205 k	2.5	2.5
WEB+APP	APP	APP

BIAS & VARIANCE w 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

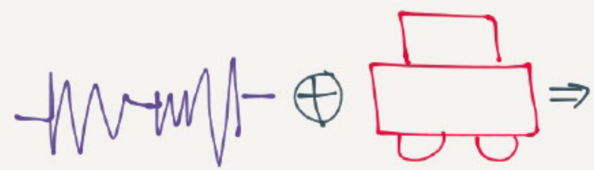
TRAIN	TD	D	T

	A	B	C	D
TRAIN	1%	1%	10%	10%
TRAIN-DEV	9%	1.5%	11%	11%
DEV	10%	10%	12%	20%
	VARIANCE	TRAIN/DEV MISMATCH	BIAS	BIAS + DATA MISMATCH

ADDRESSING DATA MISMATCH

EX. CAR 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

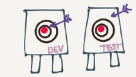


NOTE: BE CAREFUL. IF YOU ONLY HAVE 1 HR OF CAR NOISE & APPLY IT TO 10K HR SPEECH YOU MAY OVERFIT TO THE CAR NOISE

SELECTING YOUR DEV/TEST SETS

DATA: US, UK, EUROPE, S.AM, INDIA, CHINA, AUST.

OPTION 1:
DEV = UK, US, EUR
TEST = REST



IF DEV & TEST ARE DIFF & WE OPTIMIZE FOR DEV WE WILL MISS THE TEST-TARGET

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%

A IS BEST

F1 = HARMONIC MEAN BETW. RECALL & PRECISION

★ DEFINE OPTIMIZING VS SATISFICING METRICS

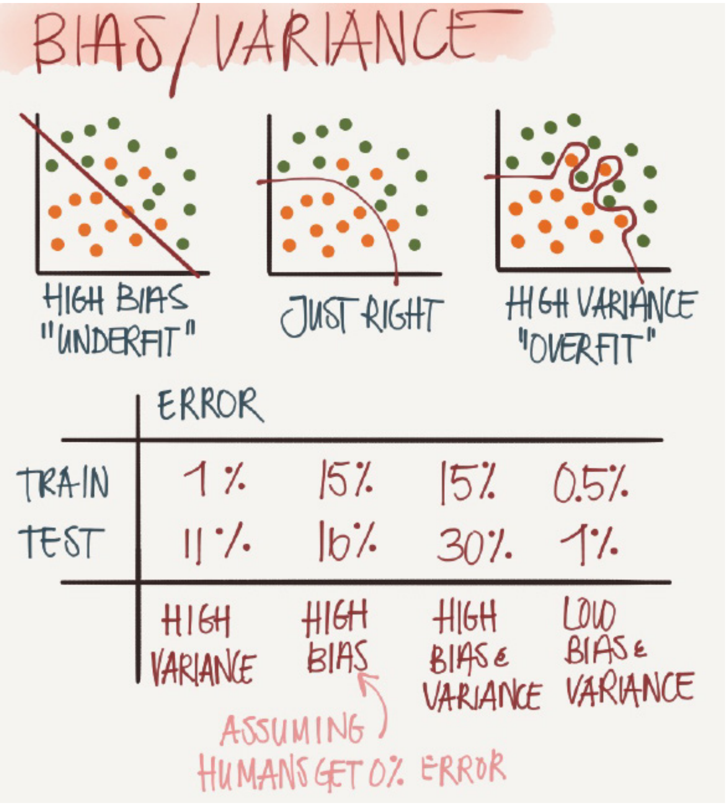
	ACCURACY	RUNTIME
A	90%	80ms
B	92%	95ms
C	~~95%~~	~~1500 ms~~

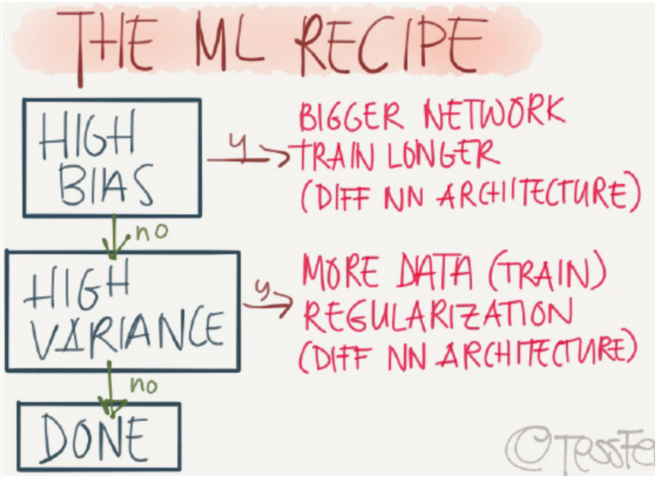
MAXIMIZE ACC. GIVEN TIME ≤ 100ms

ACCURACY = OPTIMIZING

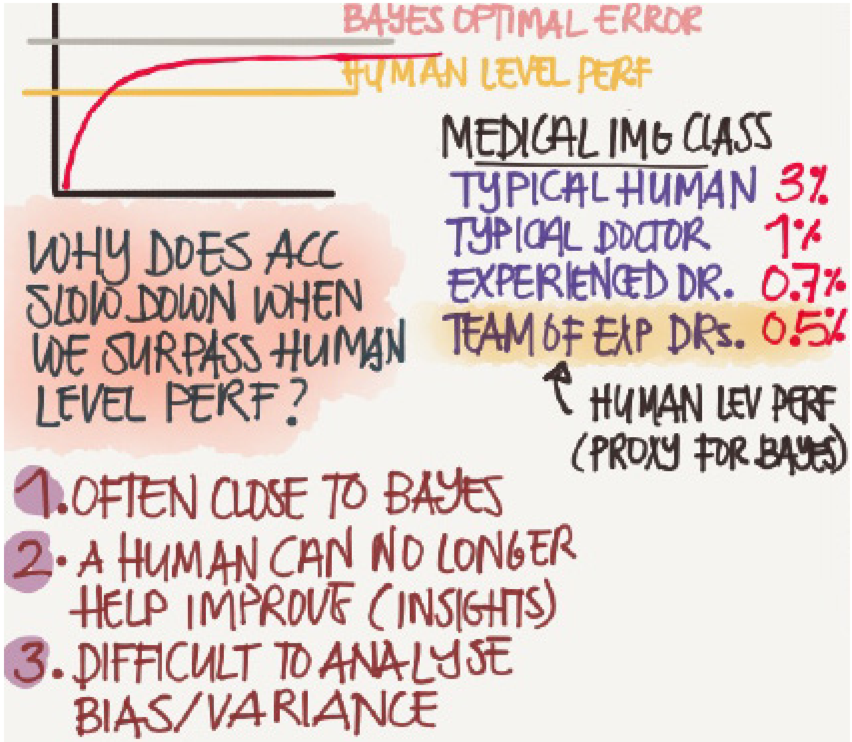
RUNTIME = SATISFICING

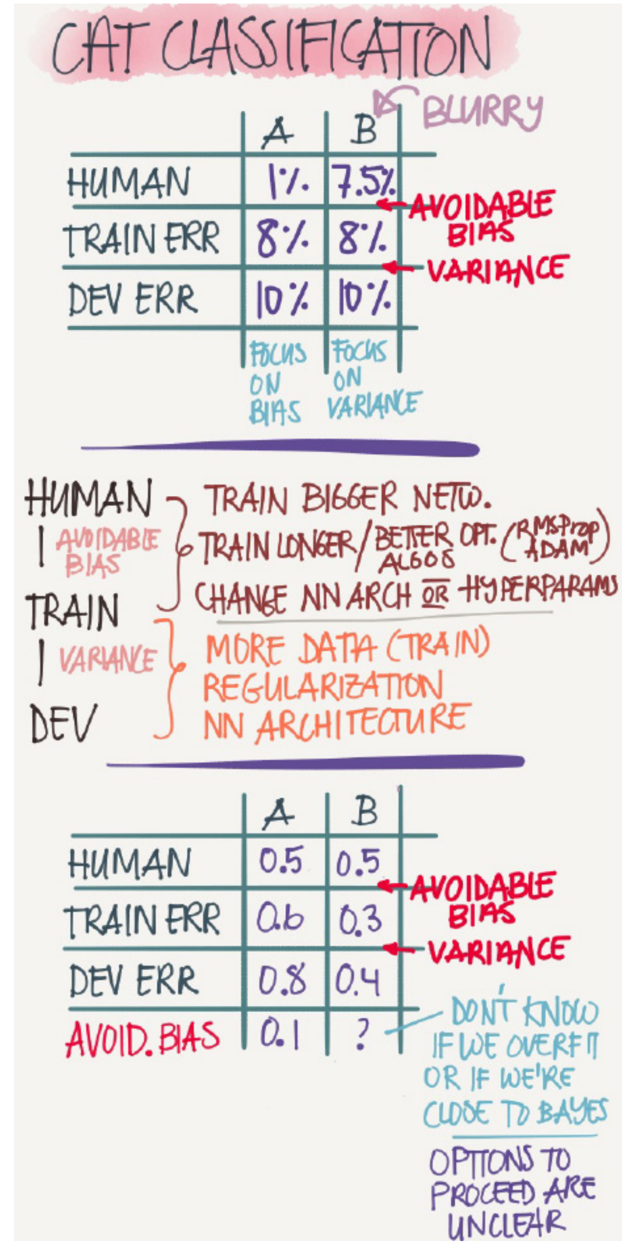
Bias vs. Variance



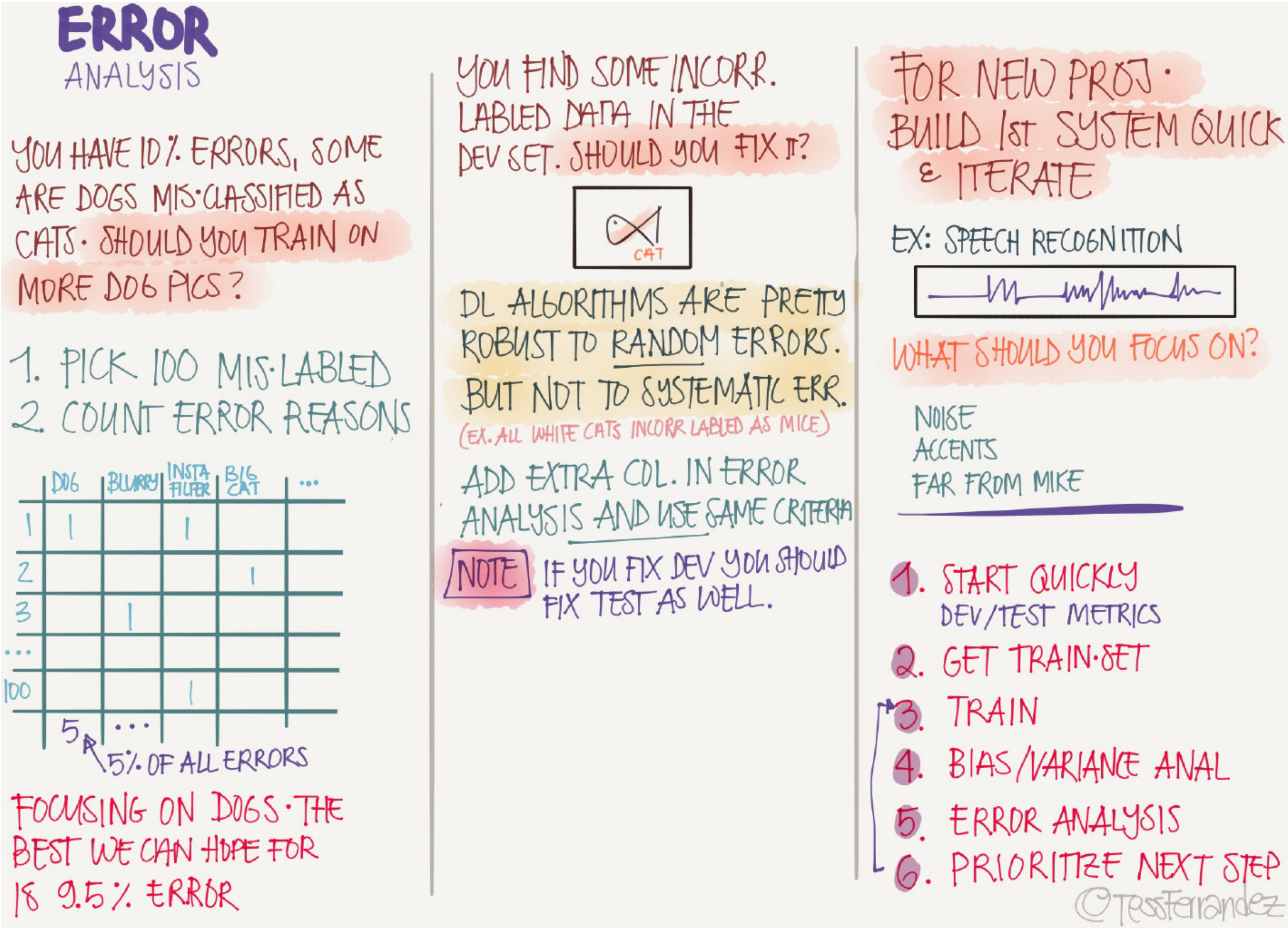


Bayes Optimal Error; Suprassing Human Performance



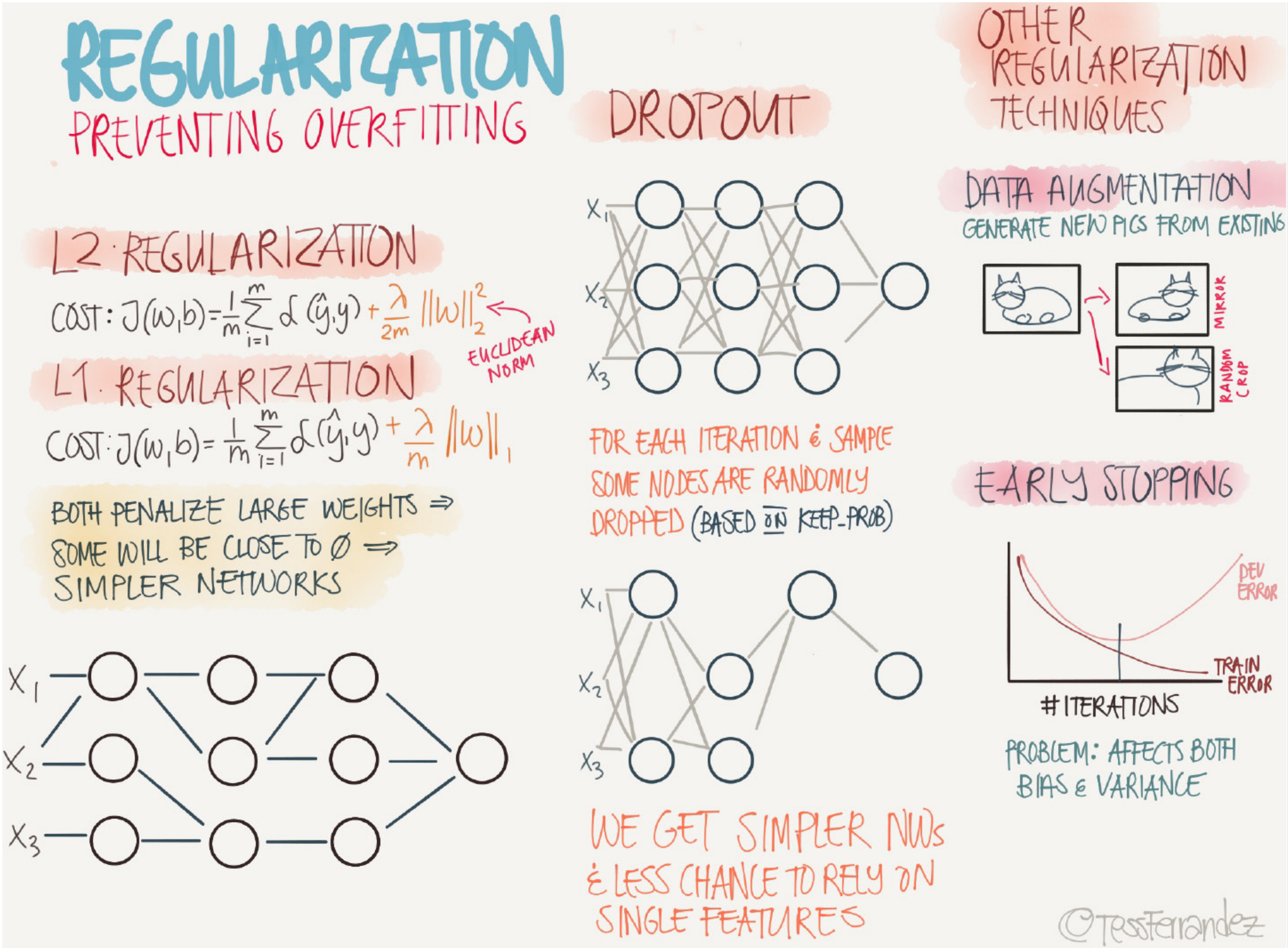


Error Analysis



Regularization

Regularization

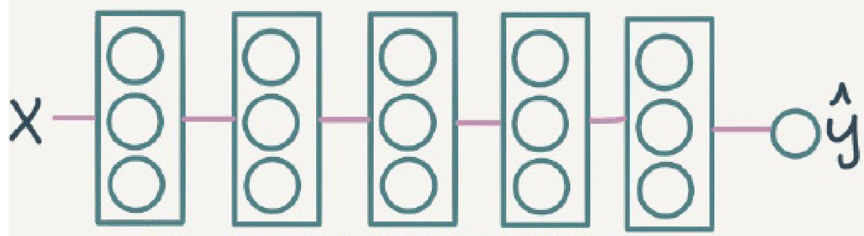


Extended Learning

Transfer learning

TRANSFER LEARNING

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



NN TO CLASSIFY CATS/DOGS/CARS

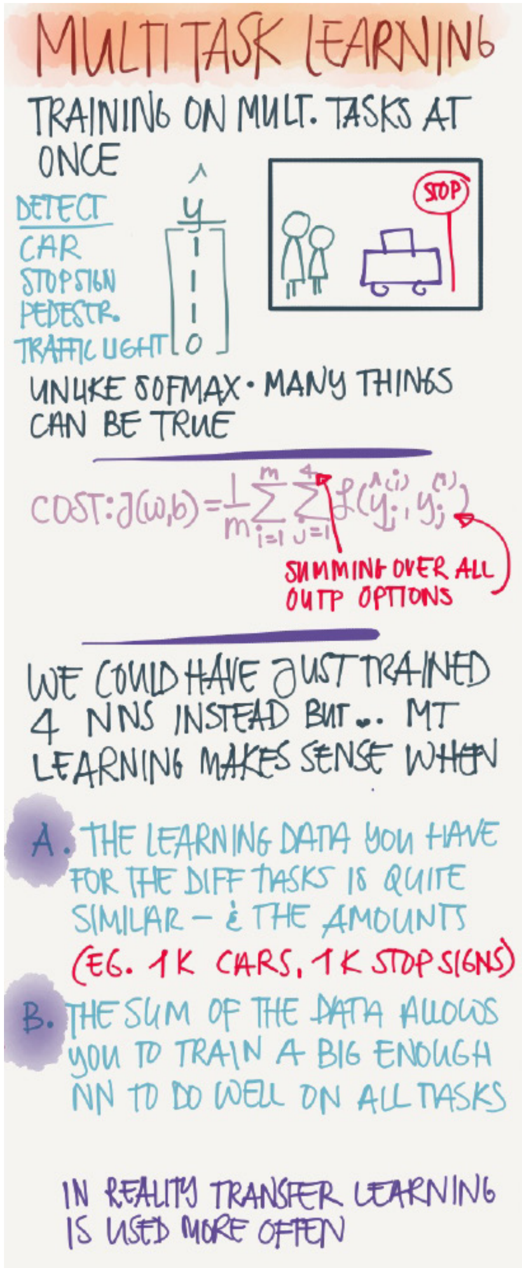
OPTION 1: YOU ONLY HAVE A FEW RADIOLOGY IMAGES

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

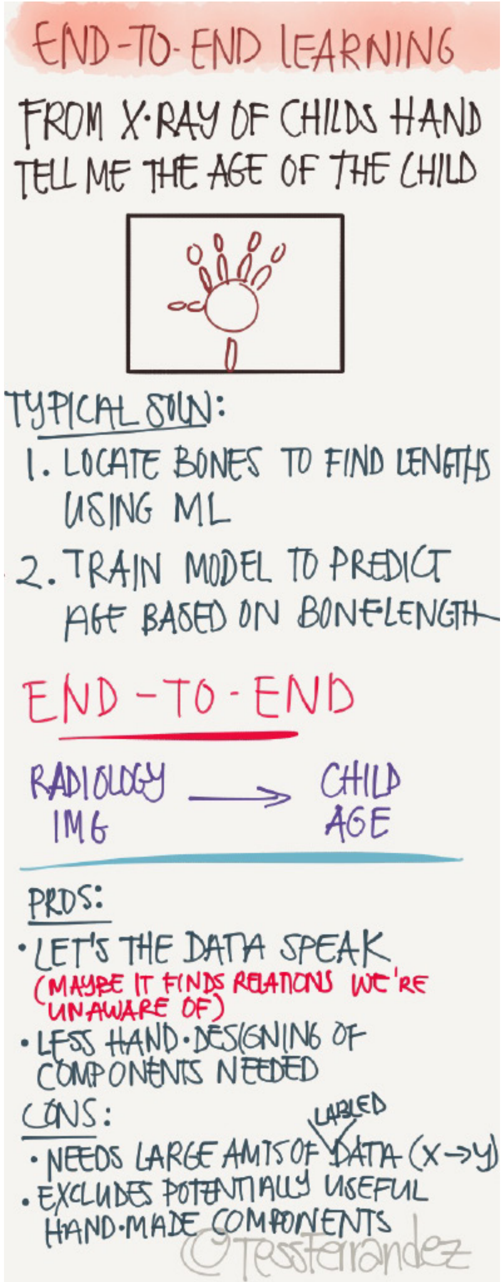
OPTION 2 YOU HAVE LOTS OF RADIOLOGY IMG

SOLUTION: INIT WITH WEIGHTS FROM CAT NN
RETRAIN ALL LAYERS

Multi-task learning

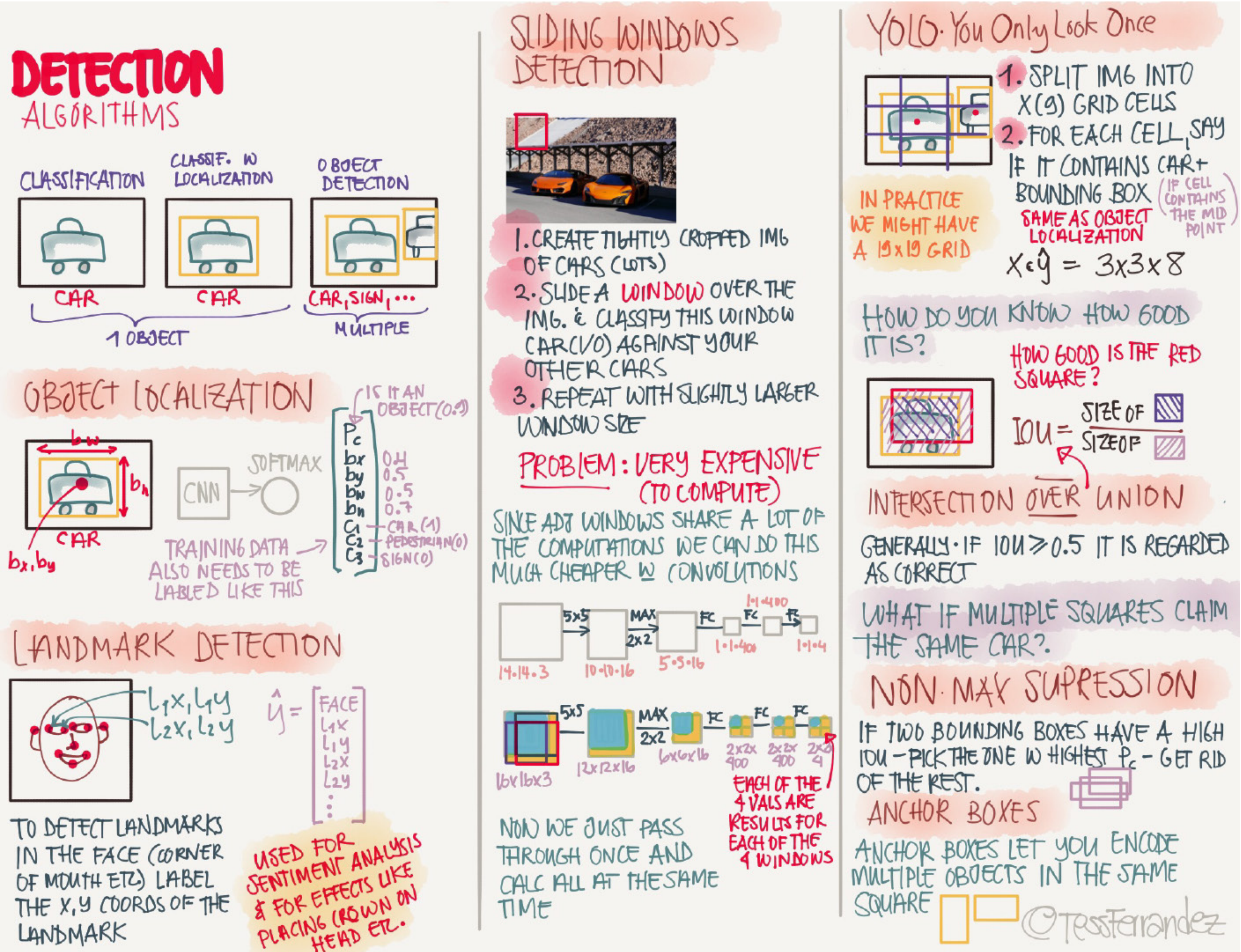


End-to-End Learning

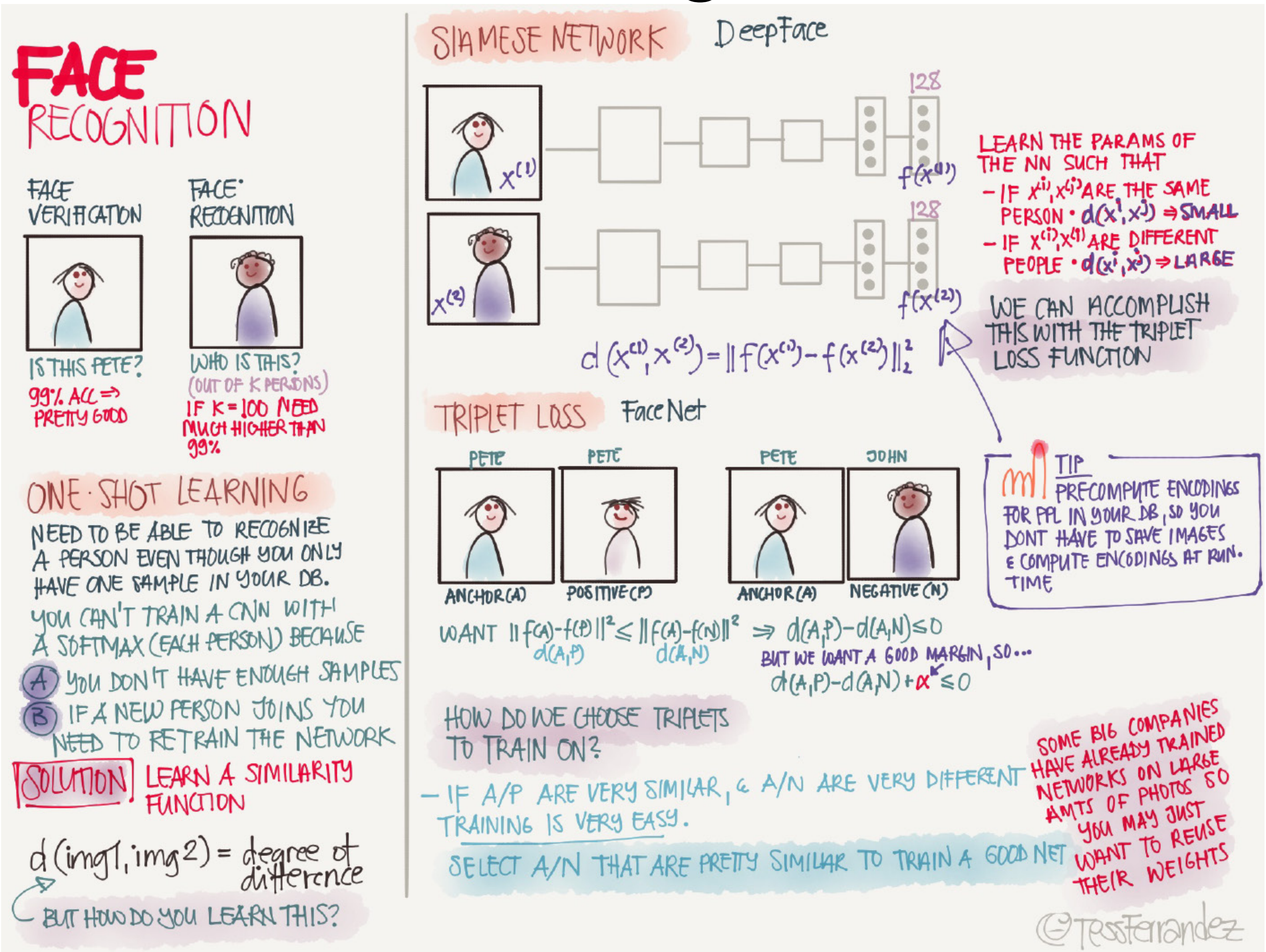


CNN applications

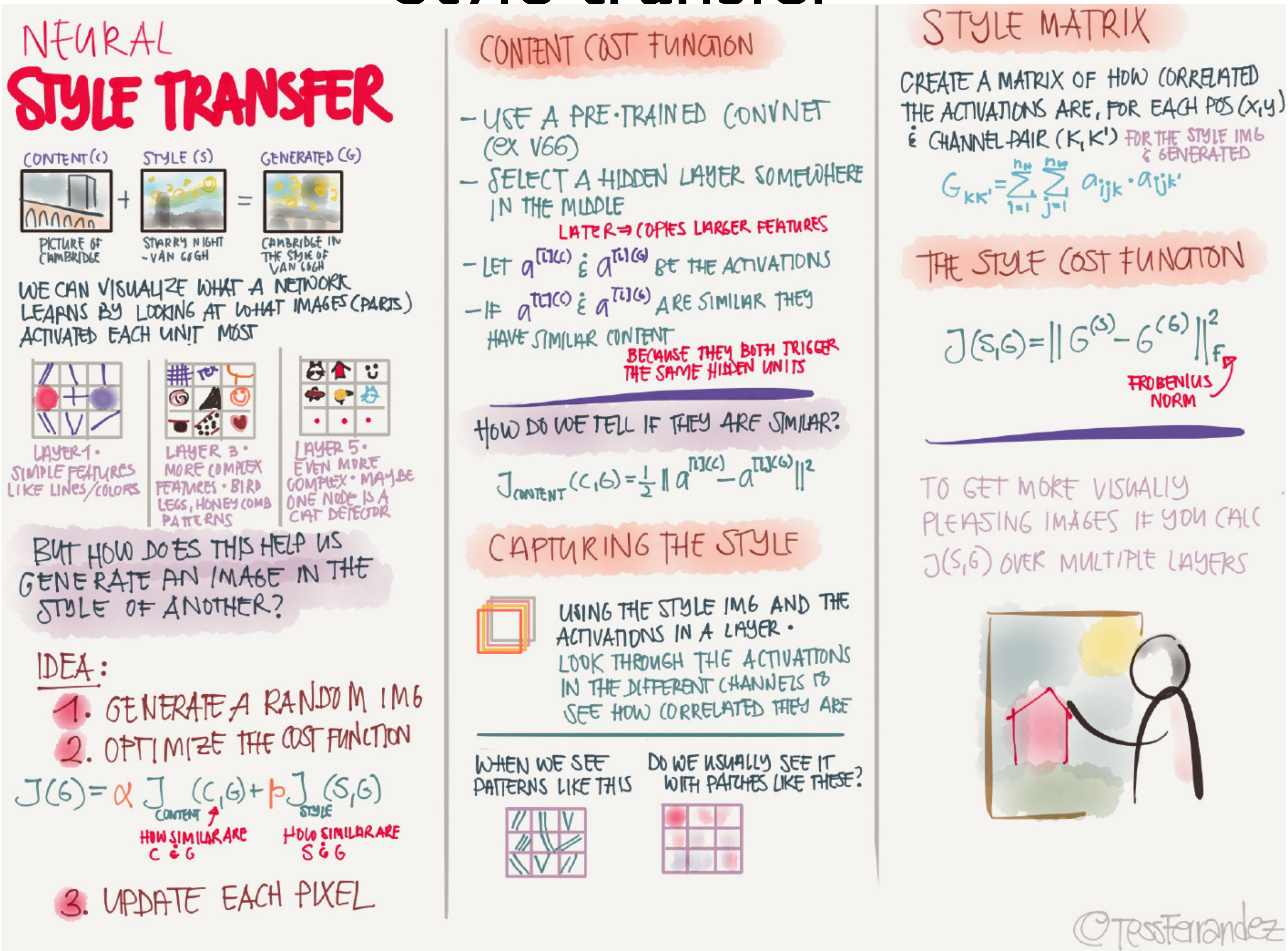
Detection, localization, landmarks



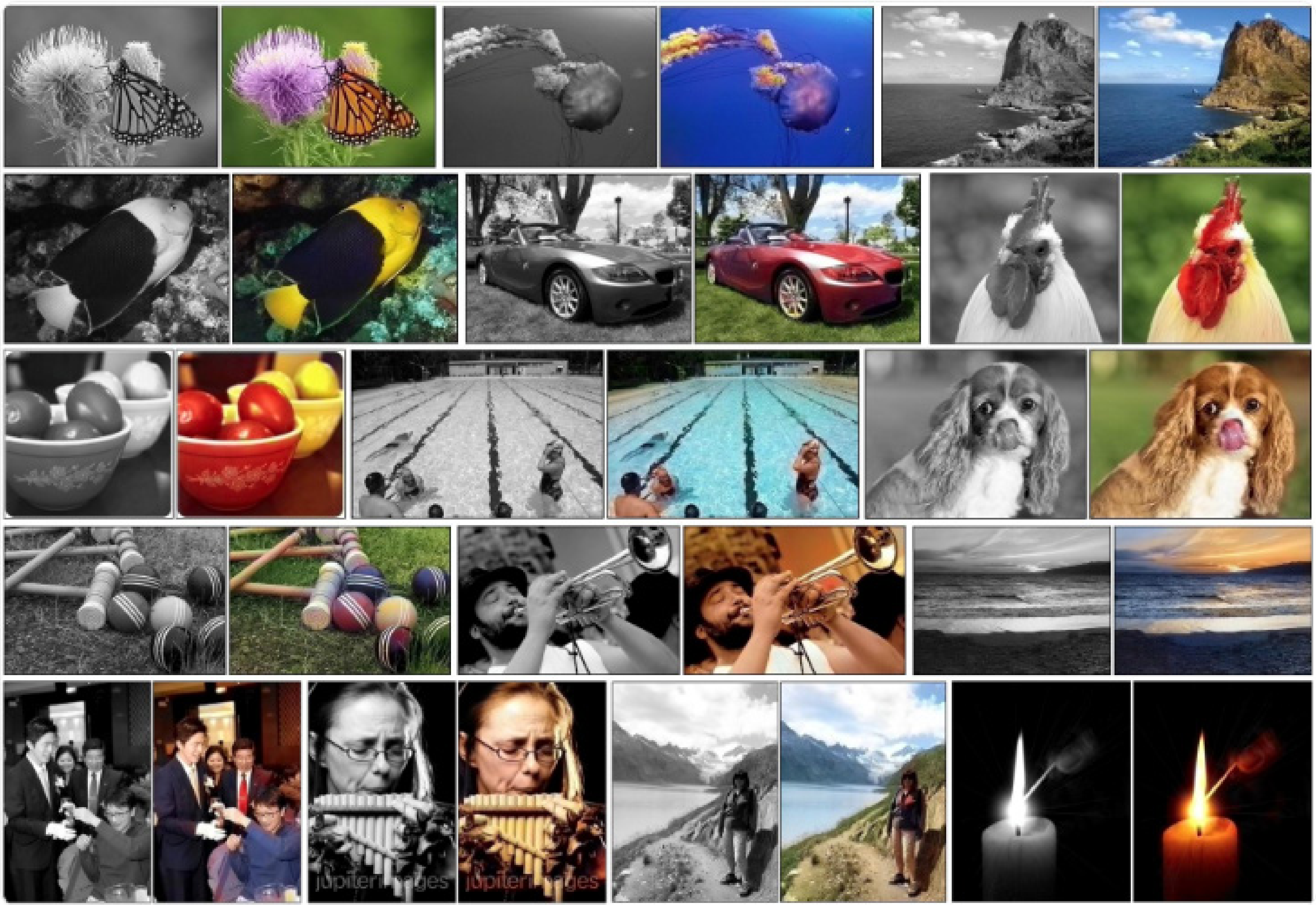
Face Recognition



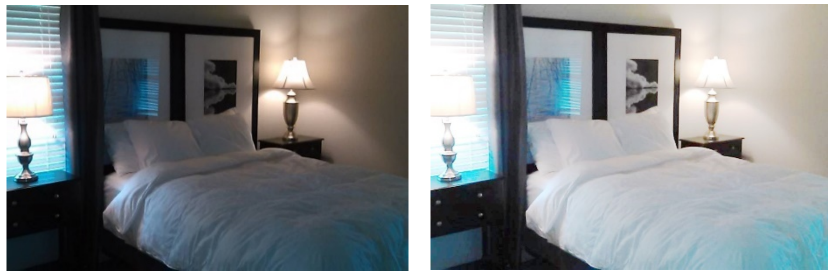
Style transfer



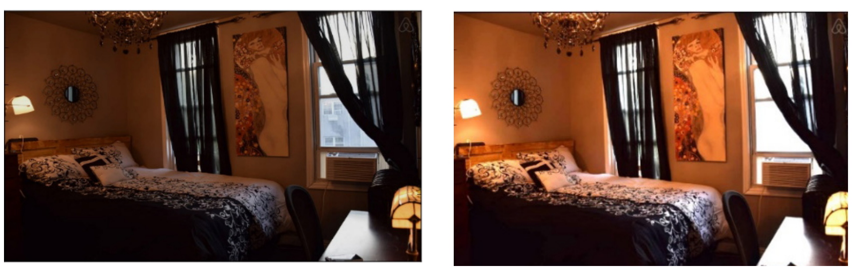
Automatic Colorization of Black and White Images



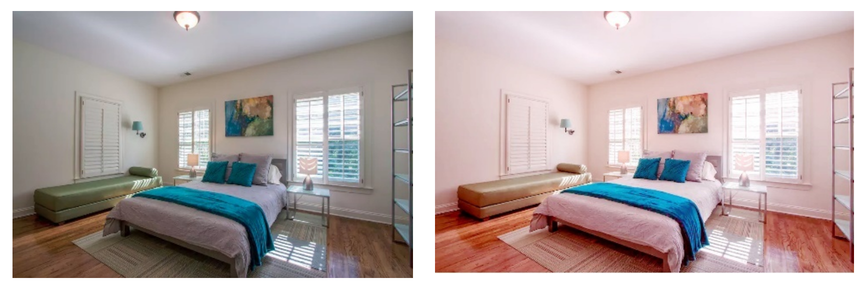
Optimizing Images



Post Processing Feature Optimization
(Color Curves and Details)

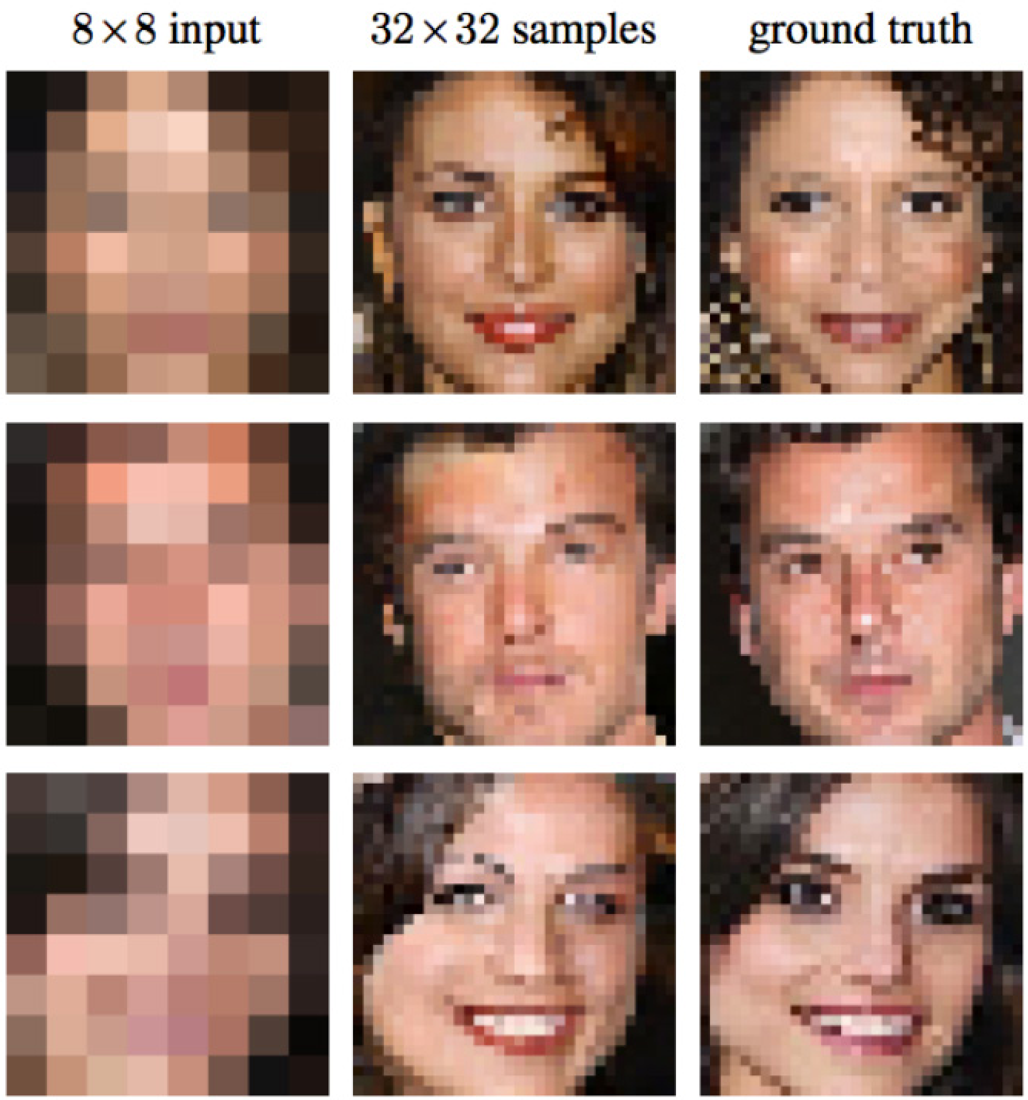


Post Processing Feature Optimization
(Illumination)



Post Processing Feature Optimization
(Color Tone: Warmness)

Up-scaling low-resolution images



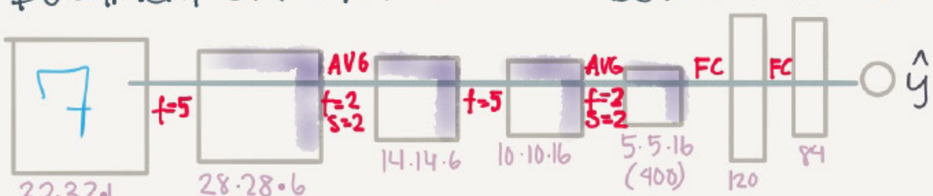
8x8 pixel photos were inputted into a Deep Learning network which tried to guess what the original face looked like. As you can see it was fairly close (the correct answer is under "ground truth").

Next-generation models explode # of parameters

CLASSIC CONV. NETS

LeNet-5

DOCUMENT CLASSIFICATION ≈60K PARAMETERS



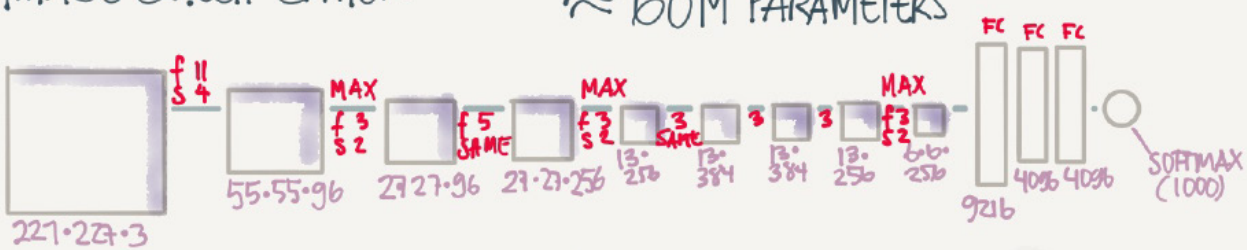
TRENDS: HEIGHT/WIDTH GO DOWN
CHANNELS GO UP

COMMON PATTERN: A COUPLE OF CONV(1+)/POOL LAYERS FOLLOWED BY A FEW FC

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

AlexNet

IMAGE CLASSIFICATION ≈ 60M PARAMETERS



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

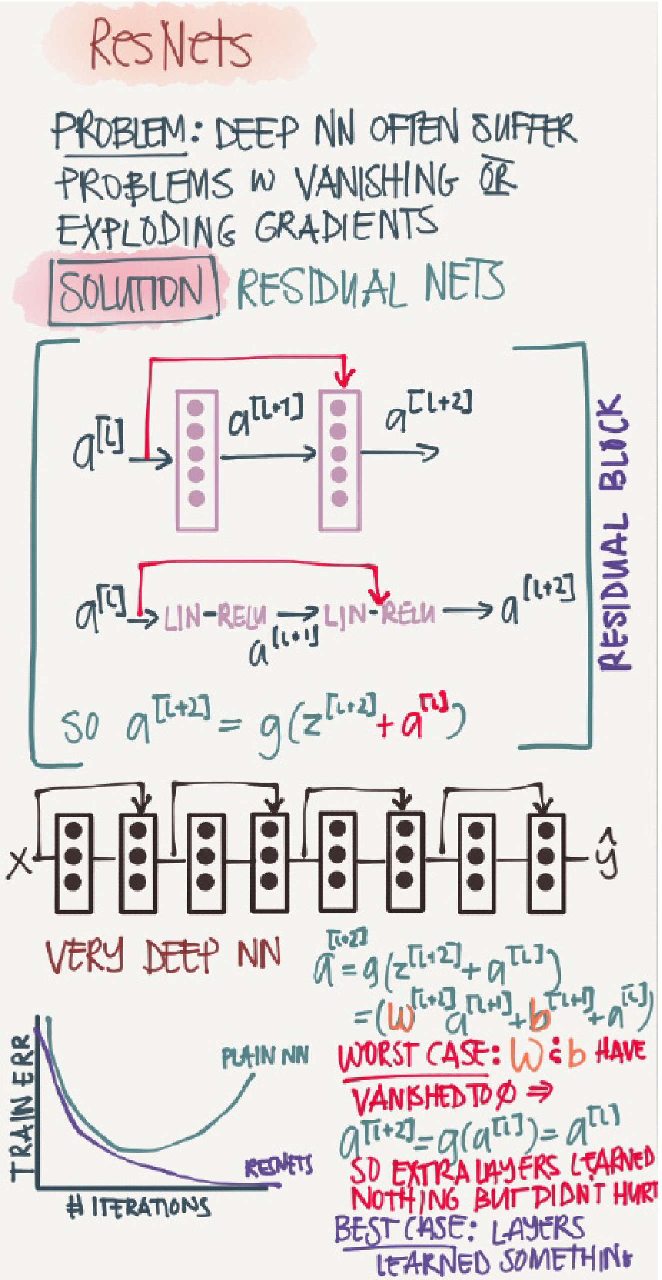
VGG-16

ALL CONV. LAYERS HAVE SAME # PARAMS
f=3x3 S=1 P=SAME
AND POOLING LAYER 2x2 S=2

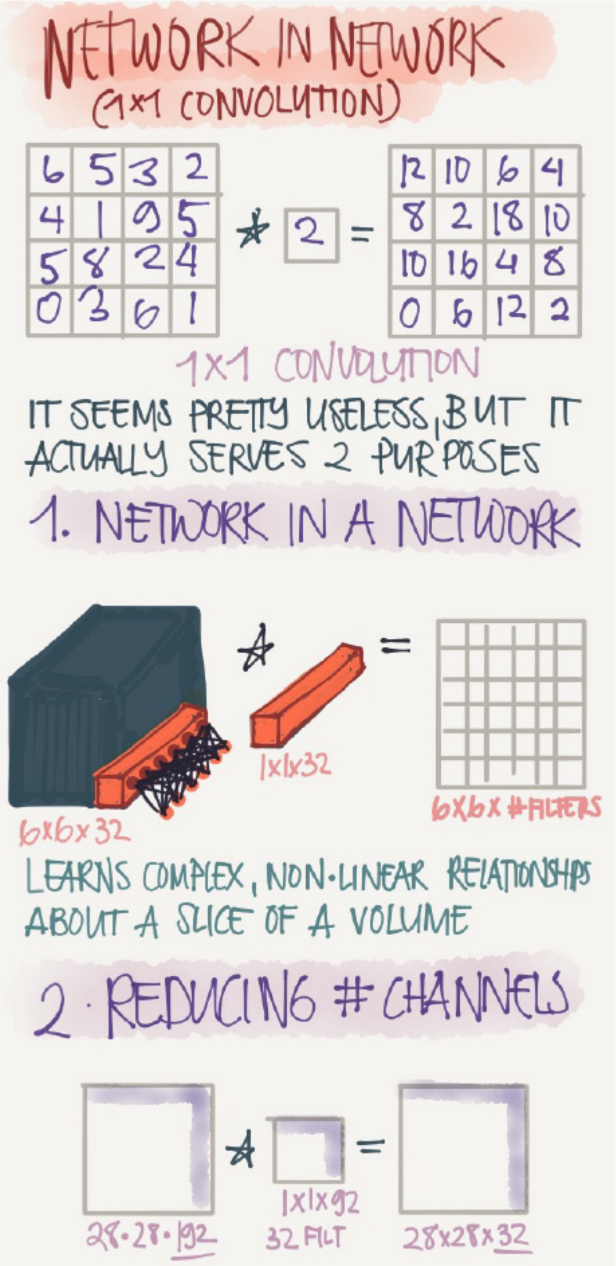
≈138 M PARAMETERS

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

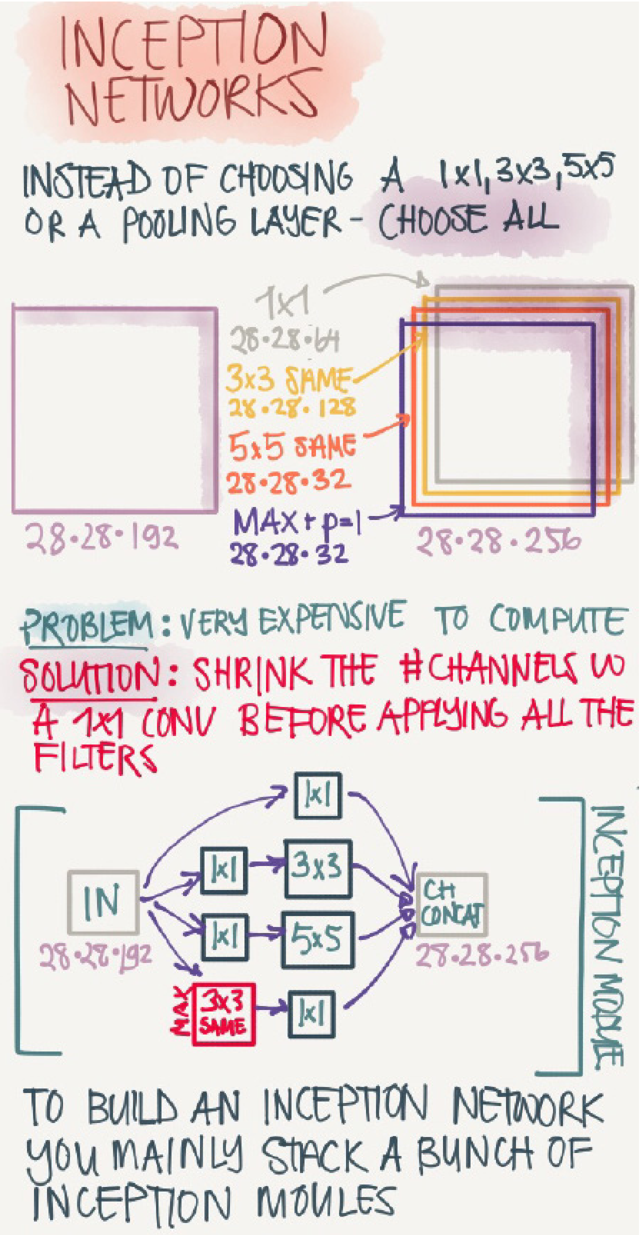
Residual Networks



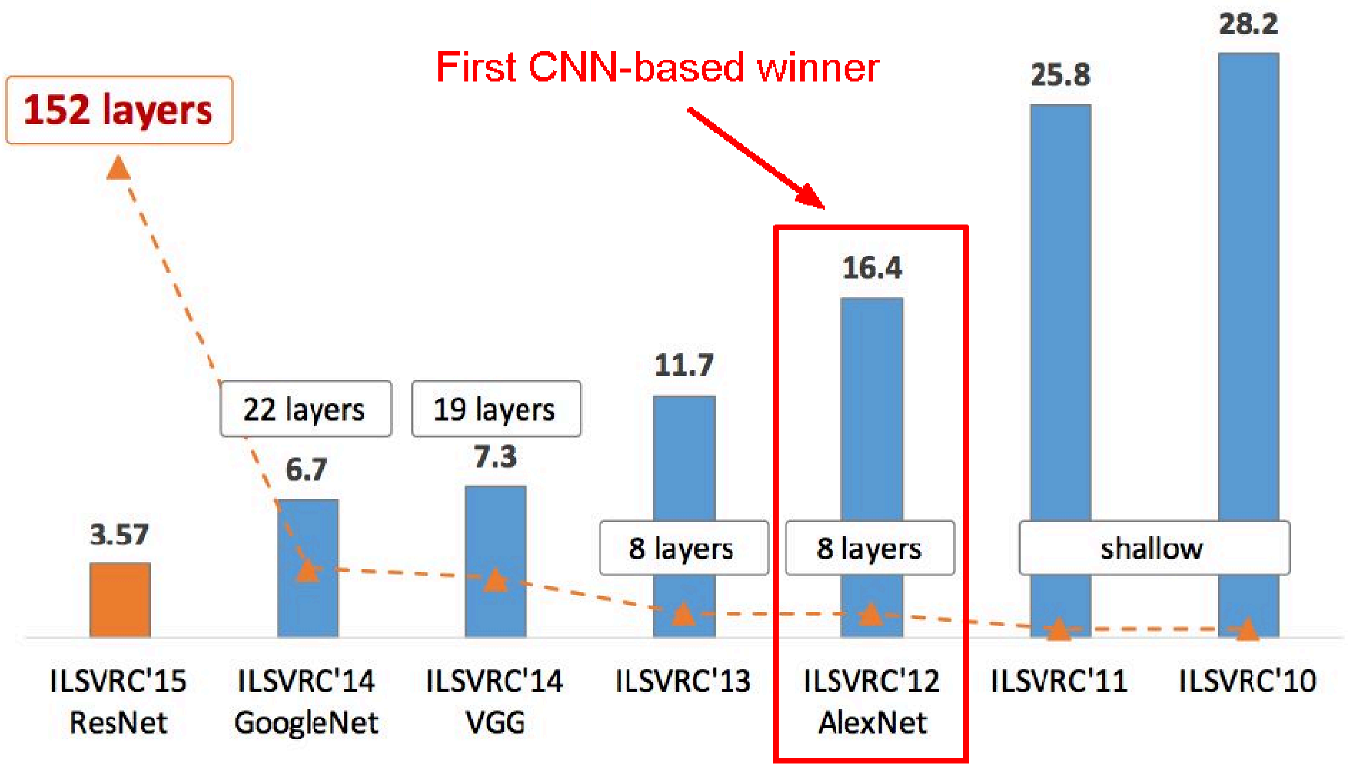
Network-in-Network: 1x1 convolution



Inception networks



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

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

[**Krizhevsky et al**., **2012**]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

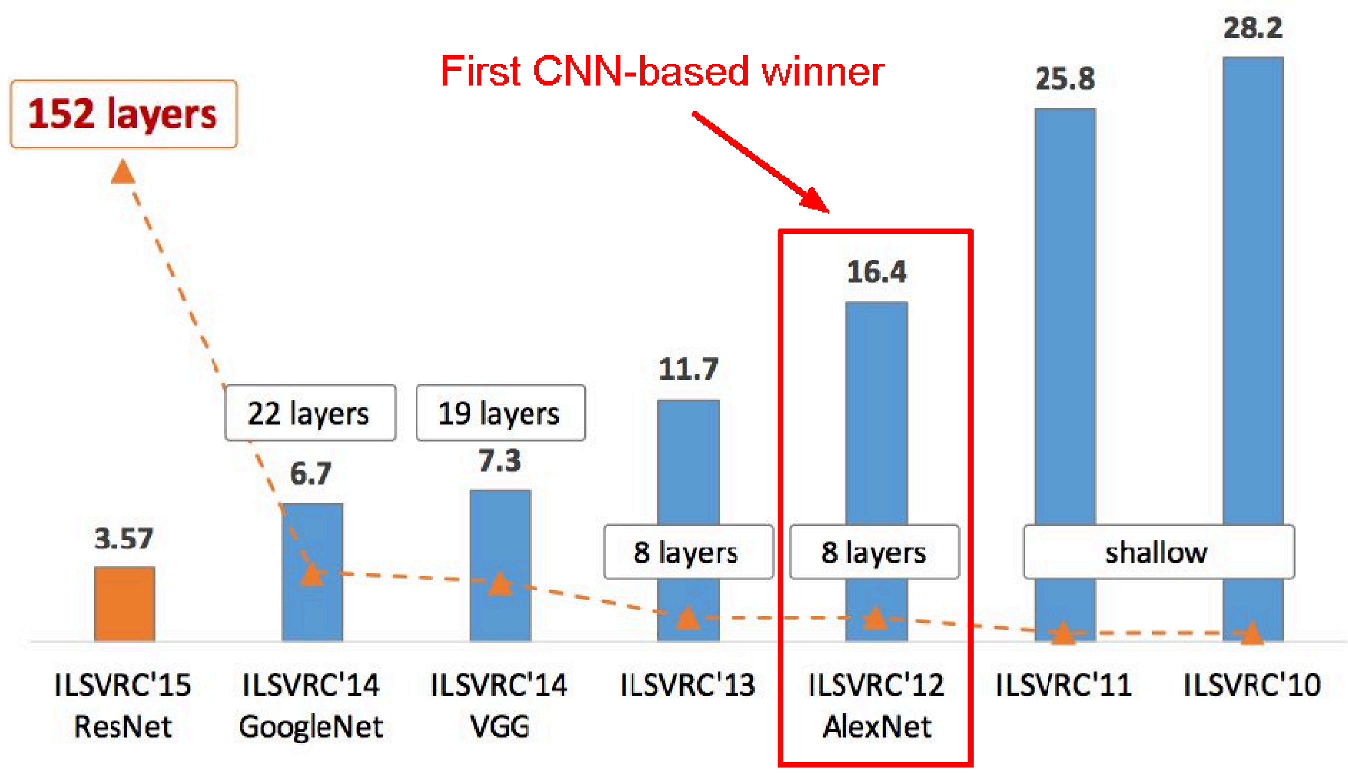
- The annual "Olympics" of computer vision.

- 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



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

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 -> [55x55x96]

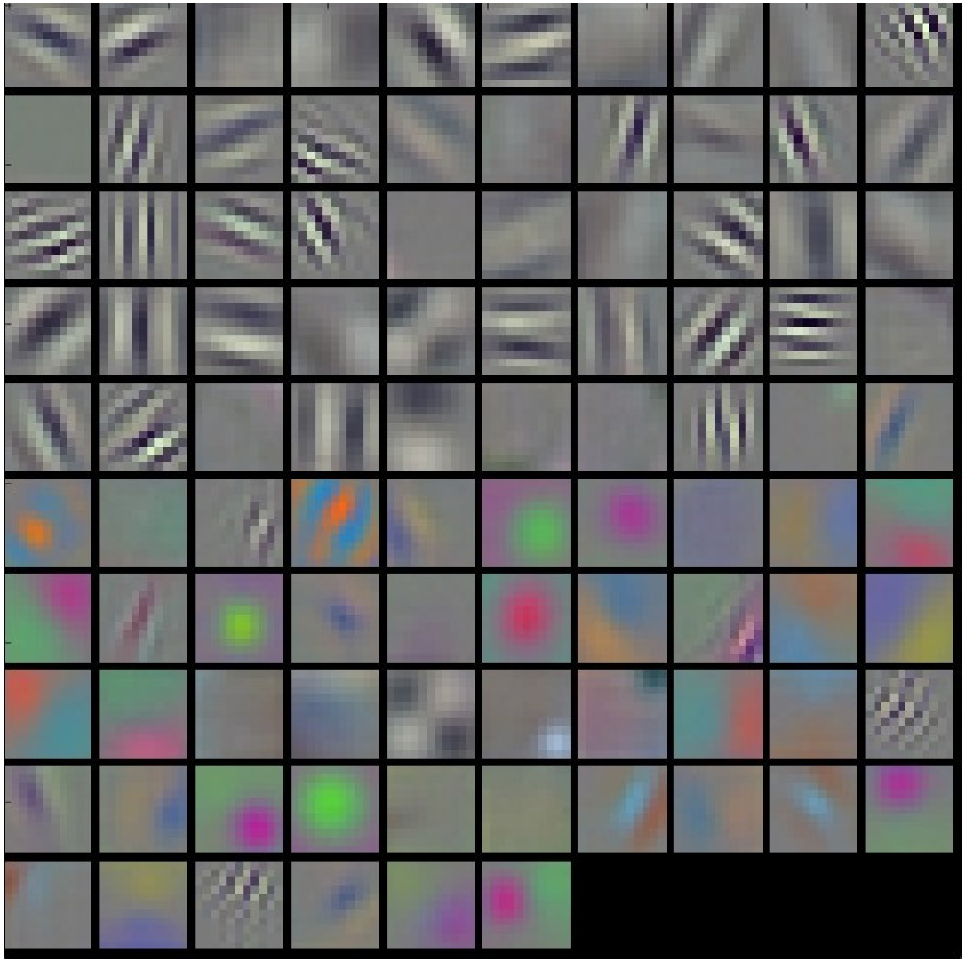
- **Number of parameters in this layer?**

(11*11*3)*96 = 35K

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[Krizhevsky et al., 2012]

AlexNet



[Krizhevsky et al., 2012]

AlexNet

Architecture
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

- 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 -> [27x27x96]

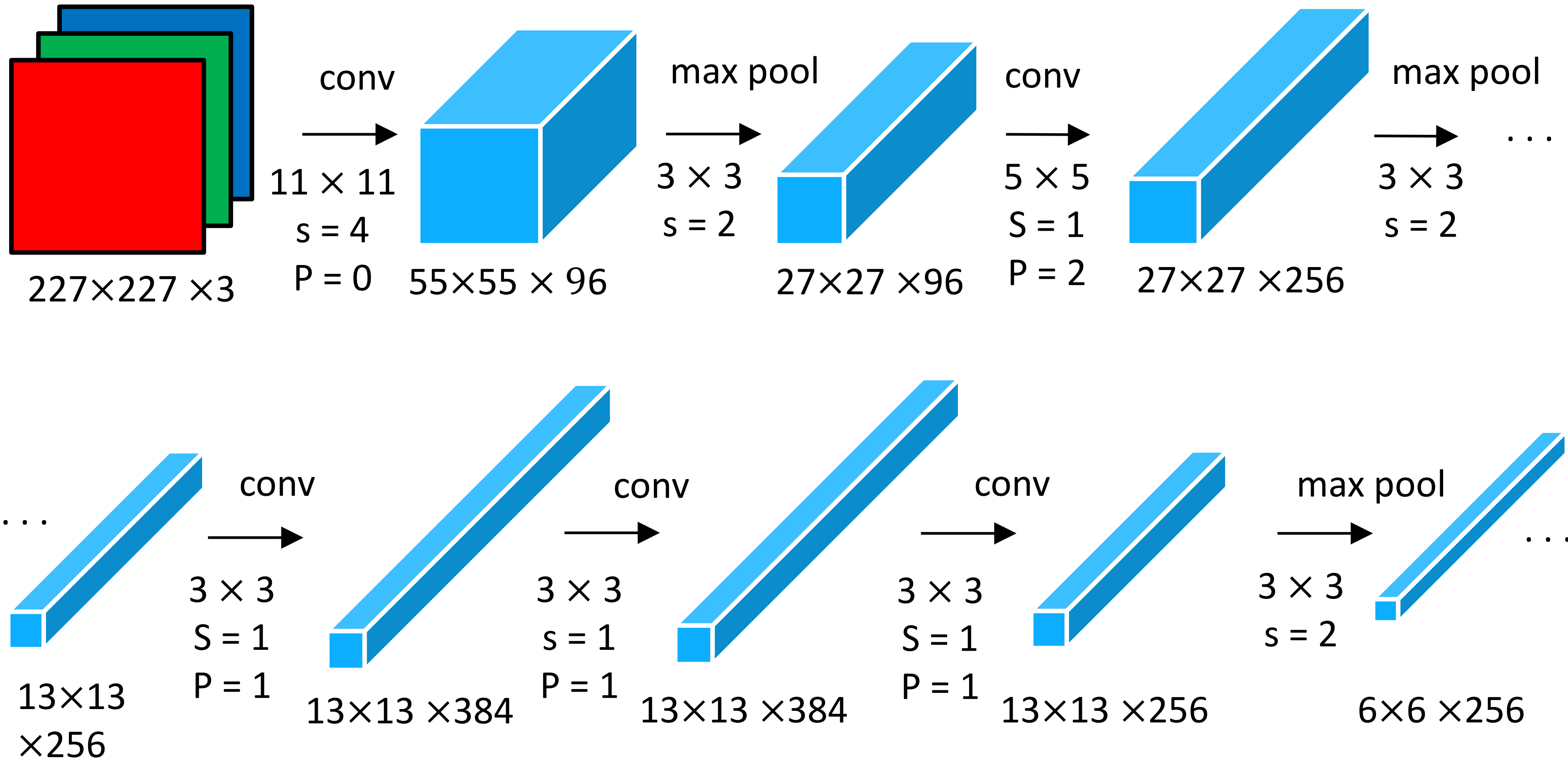
- **Number of parameters in this layer?**

  0!

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[Krizhevsky et al., 2012]

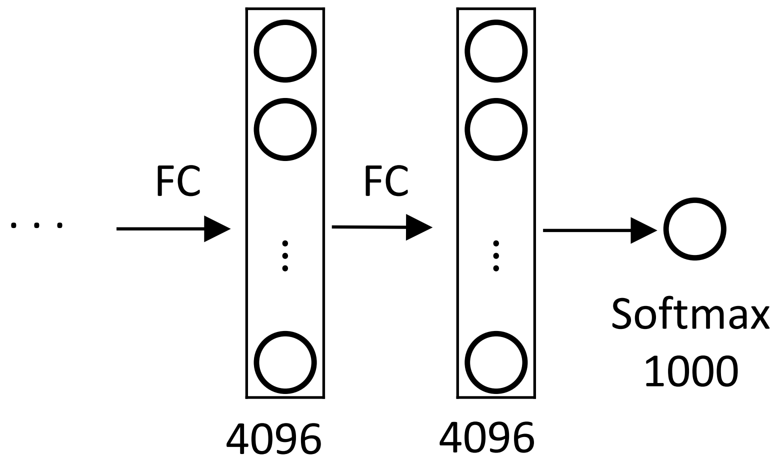
AlexNet



This slide is taken from Andrew Ng

[Krizhevsky et al., 2012]

AlexNet



This slide is taken from Andrew Ng

[**Krizhevsky et al**., **2012**]

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

[**Krizhevsky et al**., **2012**]

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.

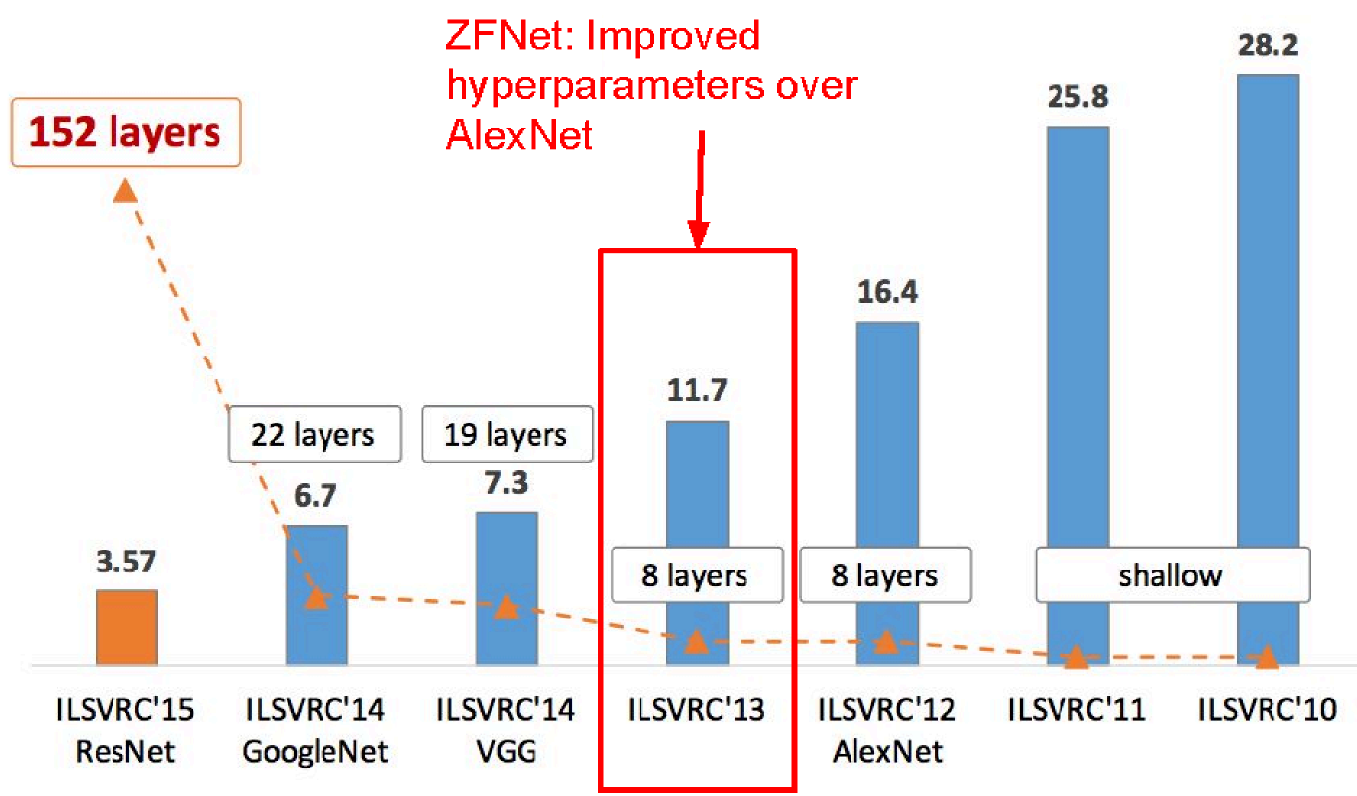
[**Krizhevsky et al**., **2012**]

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.

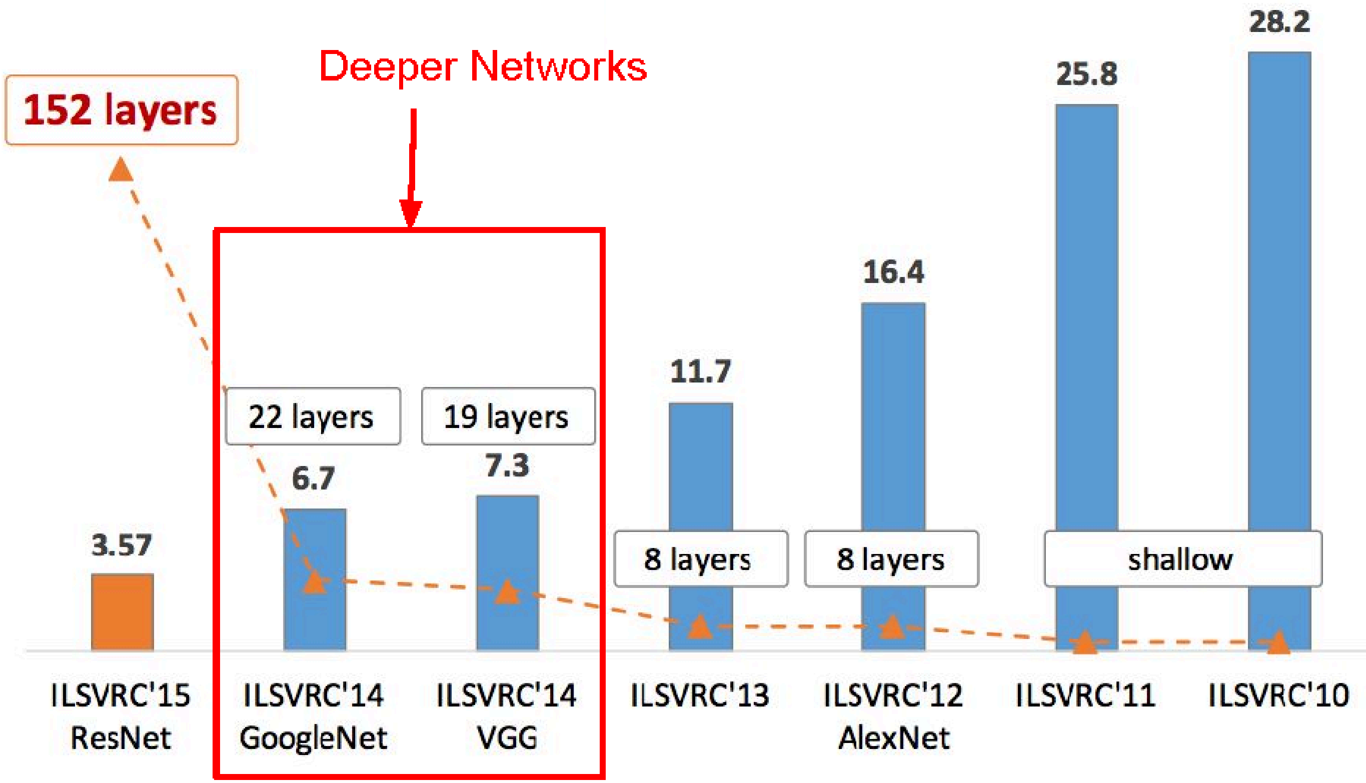
[**Krizhevsky et al**., **2012**]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

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

[Simonyan and Zisserman, 2014]

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

[Simonyan and Zisserman, 2014]

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

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

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[Simonyan and Zisserman, 2014]

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 bytes ~= 96MB / image

TOTAL params: 138M parameters

[Simonyan and Zisserman, 2014]

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

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

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[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**.

[Simonyan and Zisserman, 2014]

VGGNet

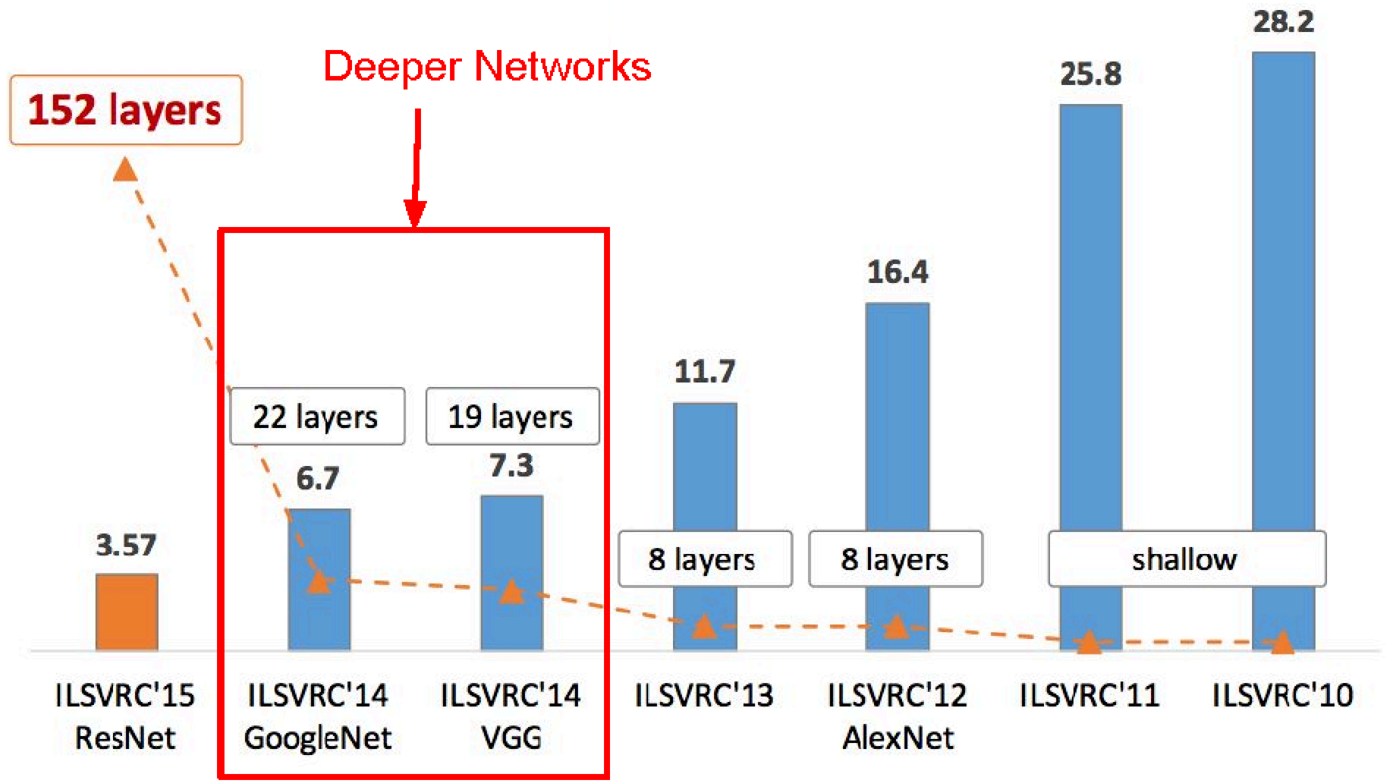
VGG Net reinforced the notion that **convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work**.

Keep it deep.

Keep it simple.

[Simonyan and Zisserman, 2014]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

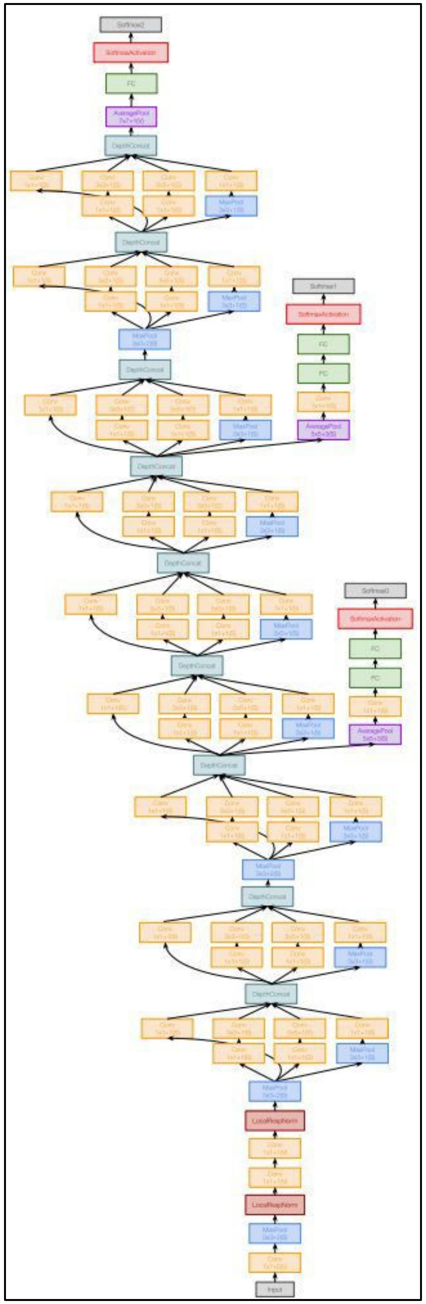


Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

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

[Szegedy et al., 2014]



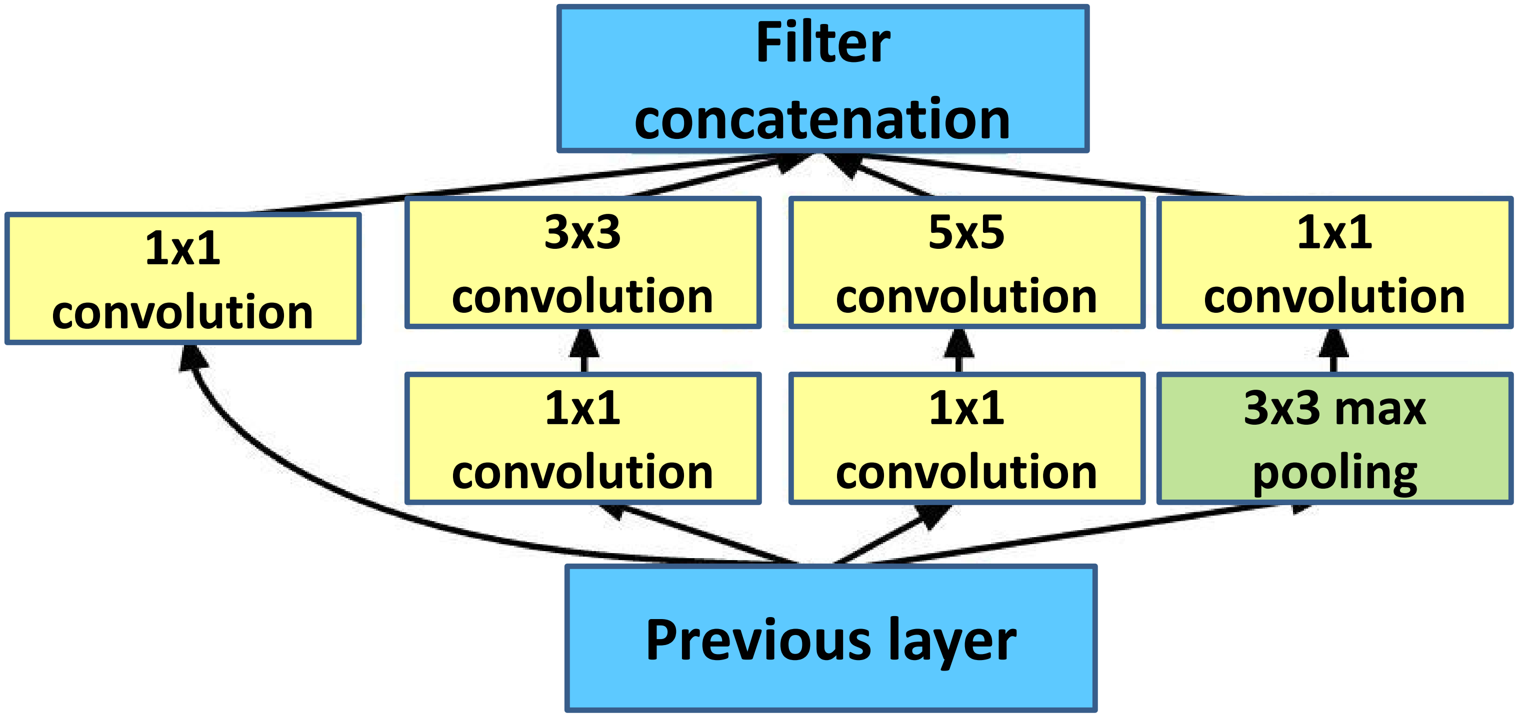
GoogleNet

- 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)

[Szegedy et al., 2014]

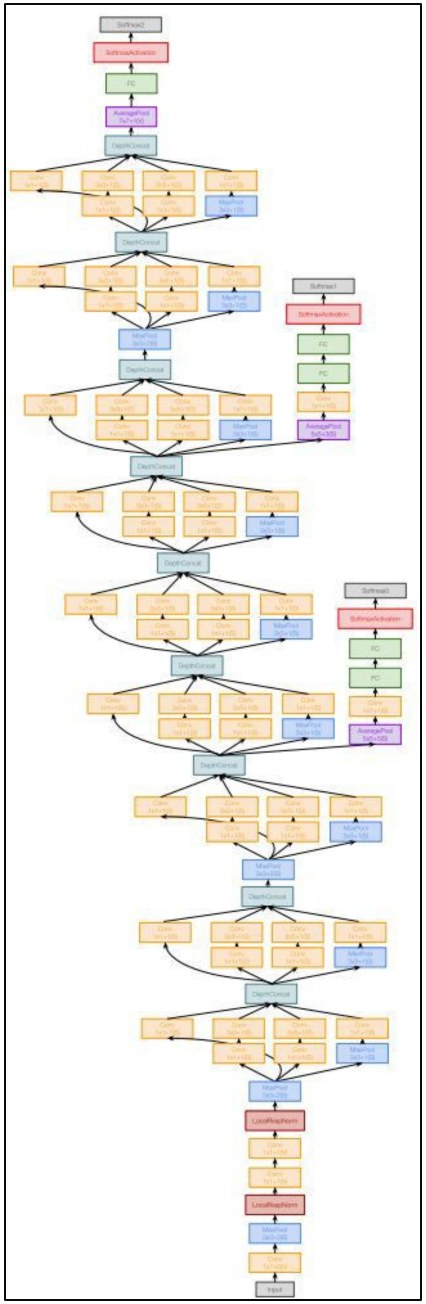
GoogleNet

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



[Szegedy et al., 2014]

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.



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)

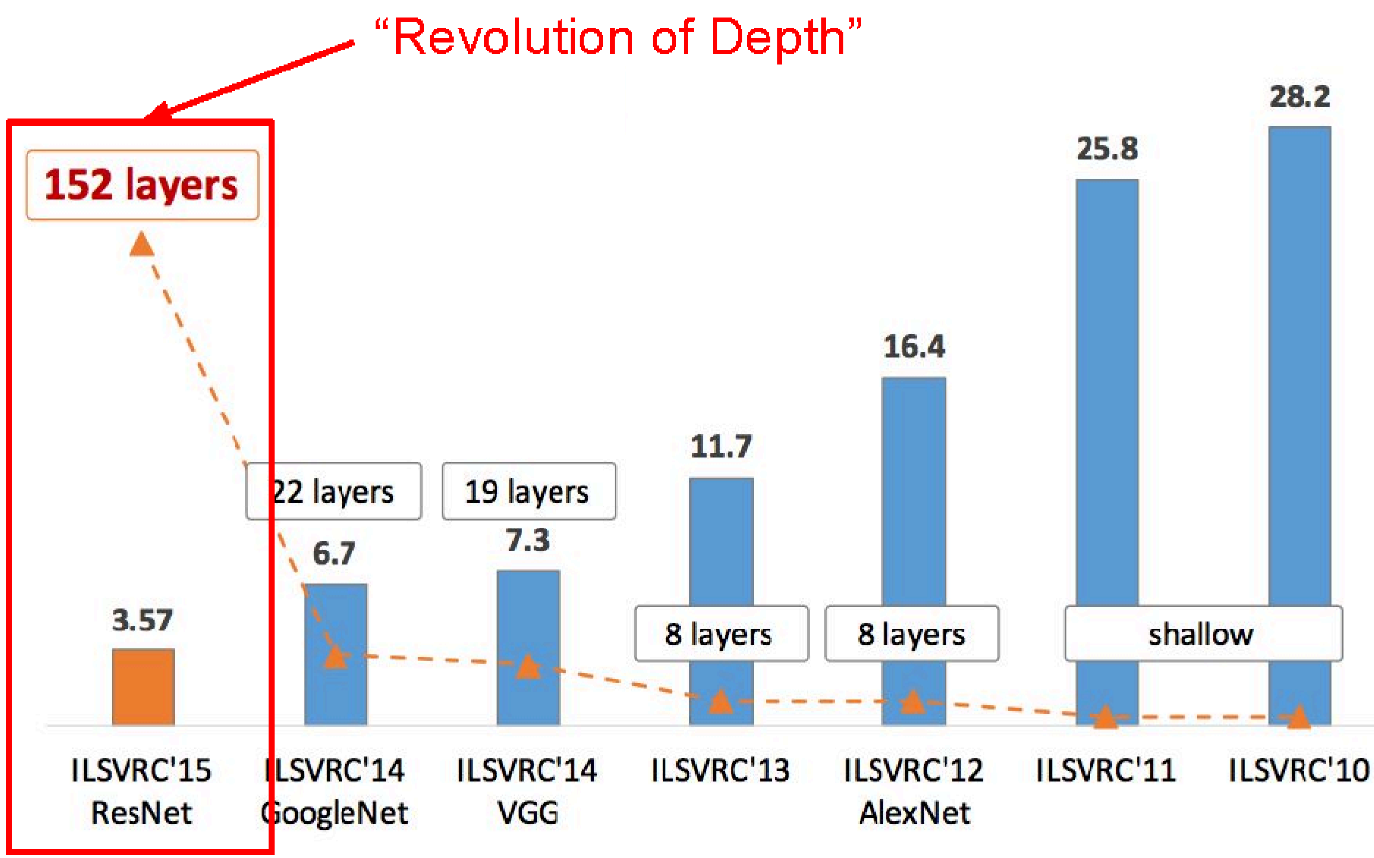
[Szegedy et al., 2014]

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**.

[Szegedy et al., 2014]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

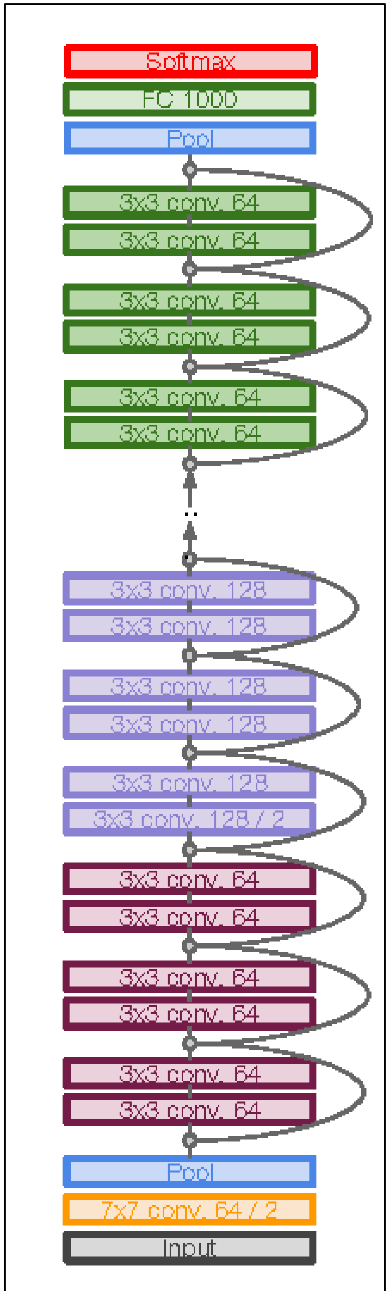


Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

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.

[He et al., 2015]



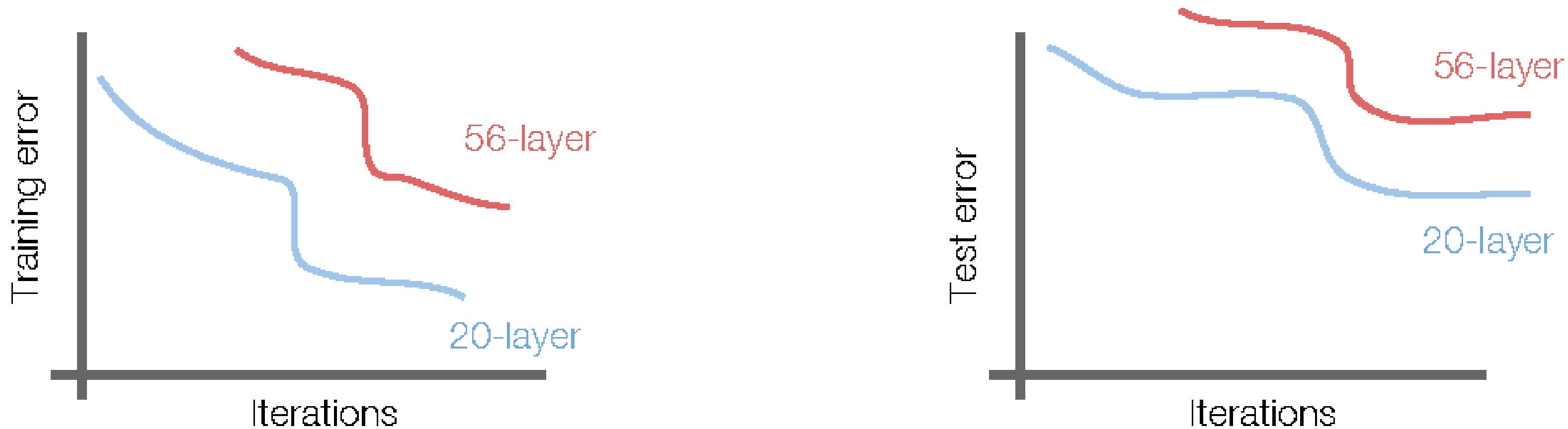
ResNet

- 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!

[He et al., 2015]

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)!

[He et al., 2015]

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

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

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

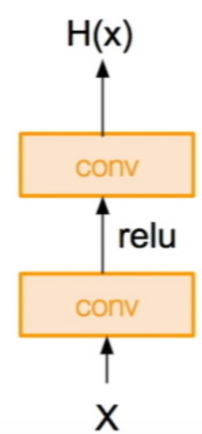
[He et al., 2015]

ResNet

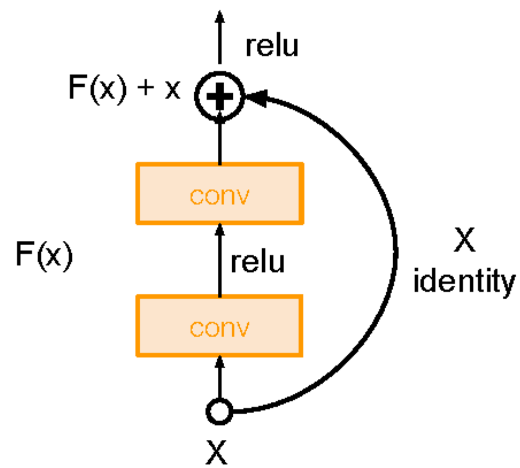
Residual Block

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.



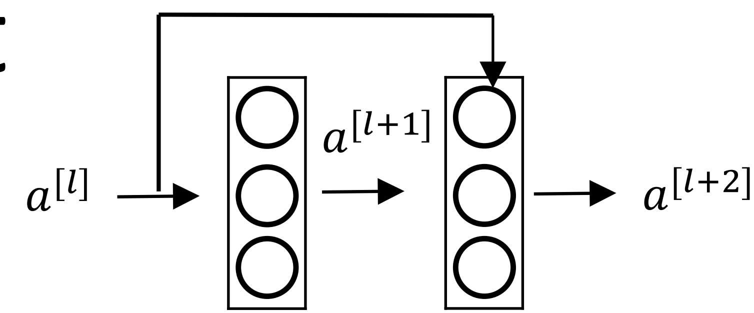
"Plain" layers



Residual block

[He et al., 2015]

ResNet



Short cut/ skip connection

$a^{[l]} \rightarrow$ **Linear** $\rightarrow$ **ReLU** $\rightarrow$ **Linear** $\rightarrow$ **ReLU** $\rightarrow a^{[l+2]}$

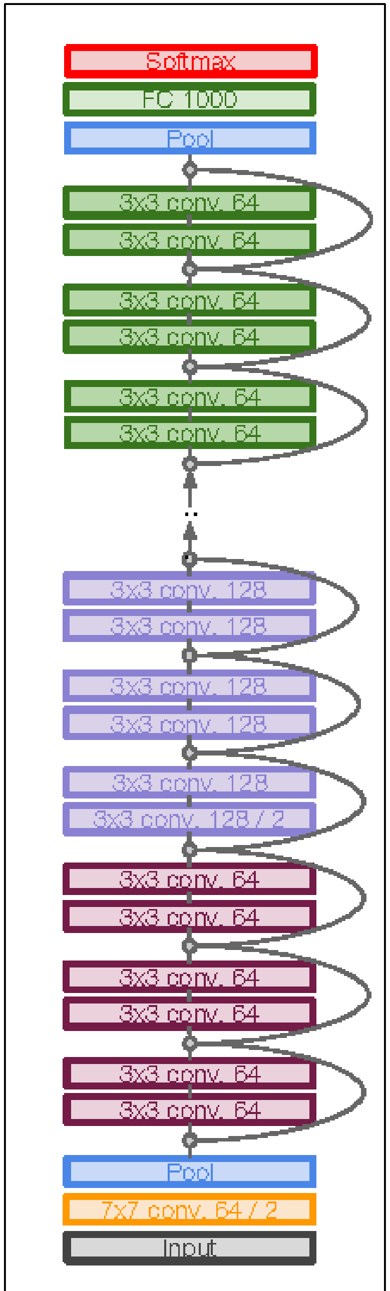
$a^{[l+1]}$

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

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

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

[He et al., 2015]



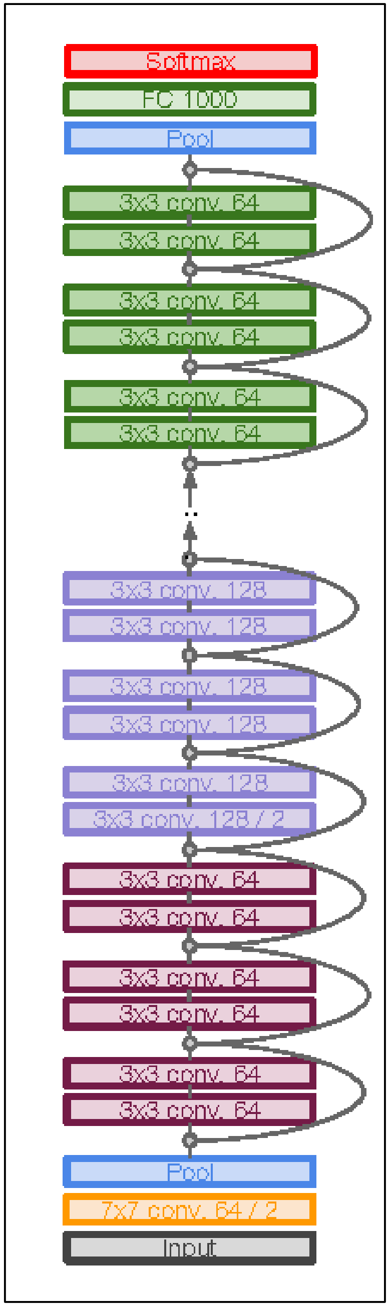
ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 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)

[He et al., 2015]

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.



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)

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[He et al., 2015]

ResNet

Experimental Results:

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

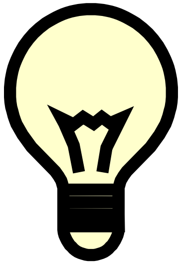
[He et al., 2015]

ResNet

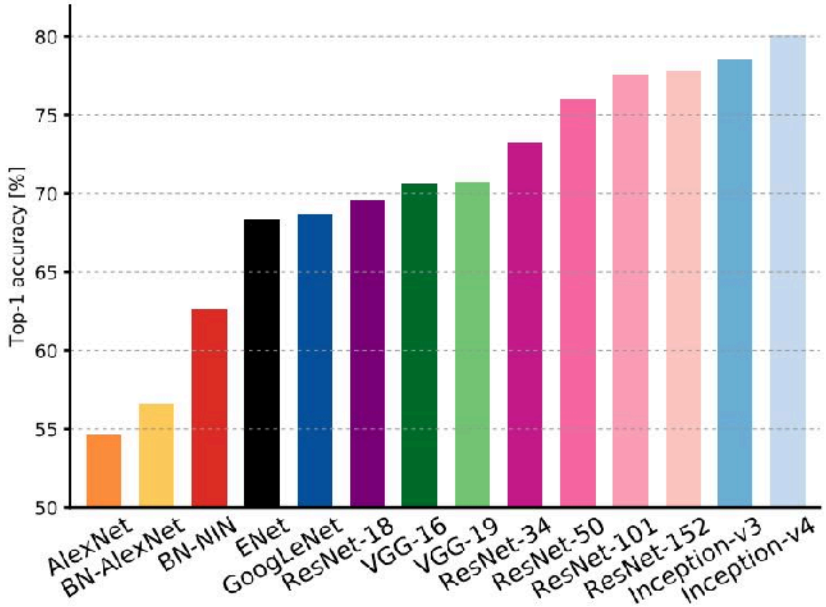
The **best** CNN architecture that we currently have and is a great innovation for the idea of residual learning.

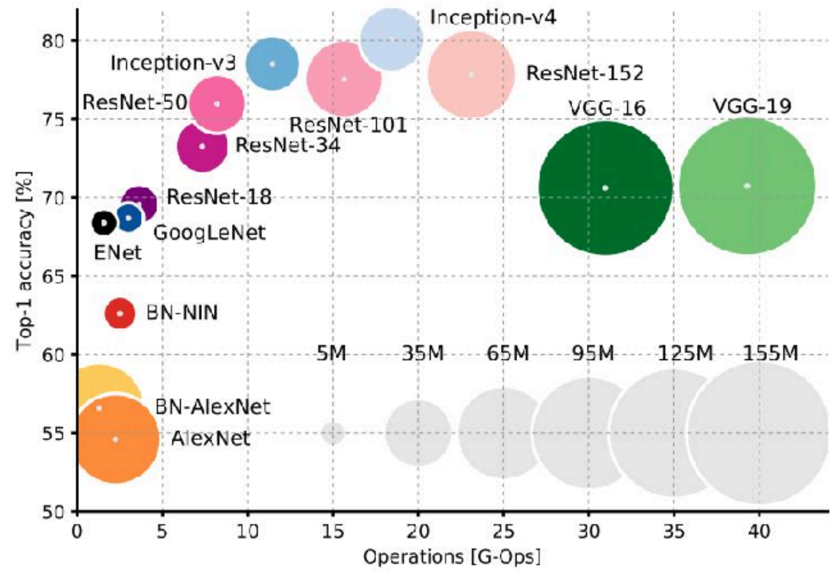
Even better than human performance!

[He et al., 2015]



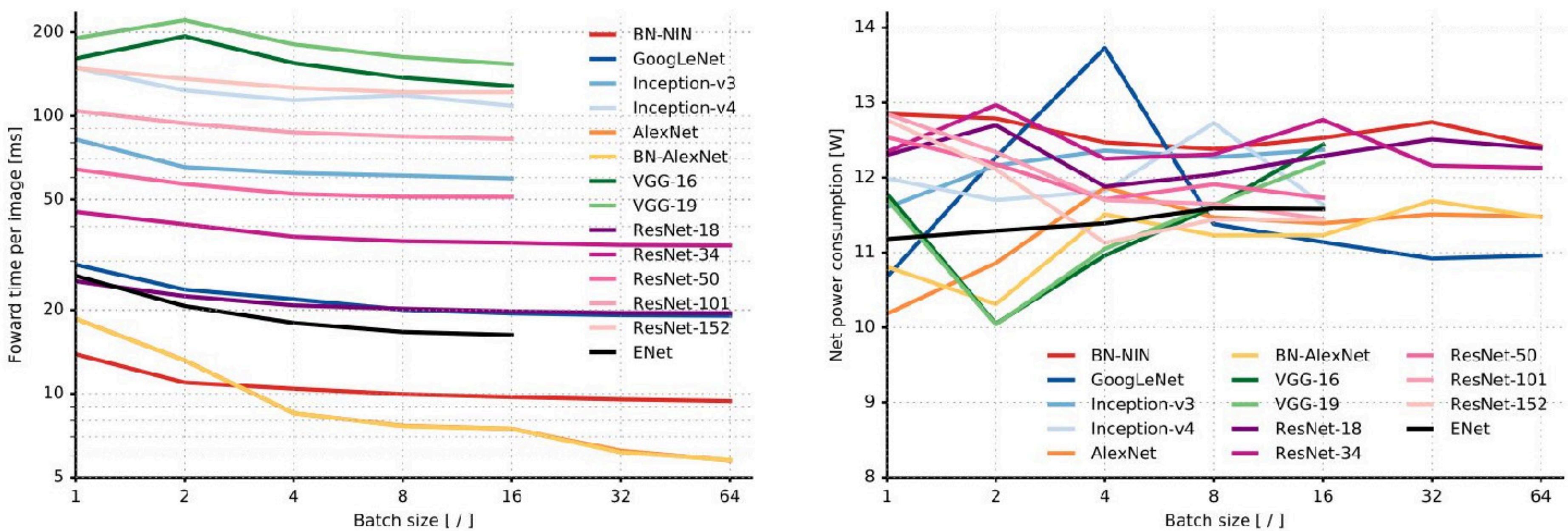
Accuracy comparison





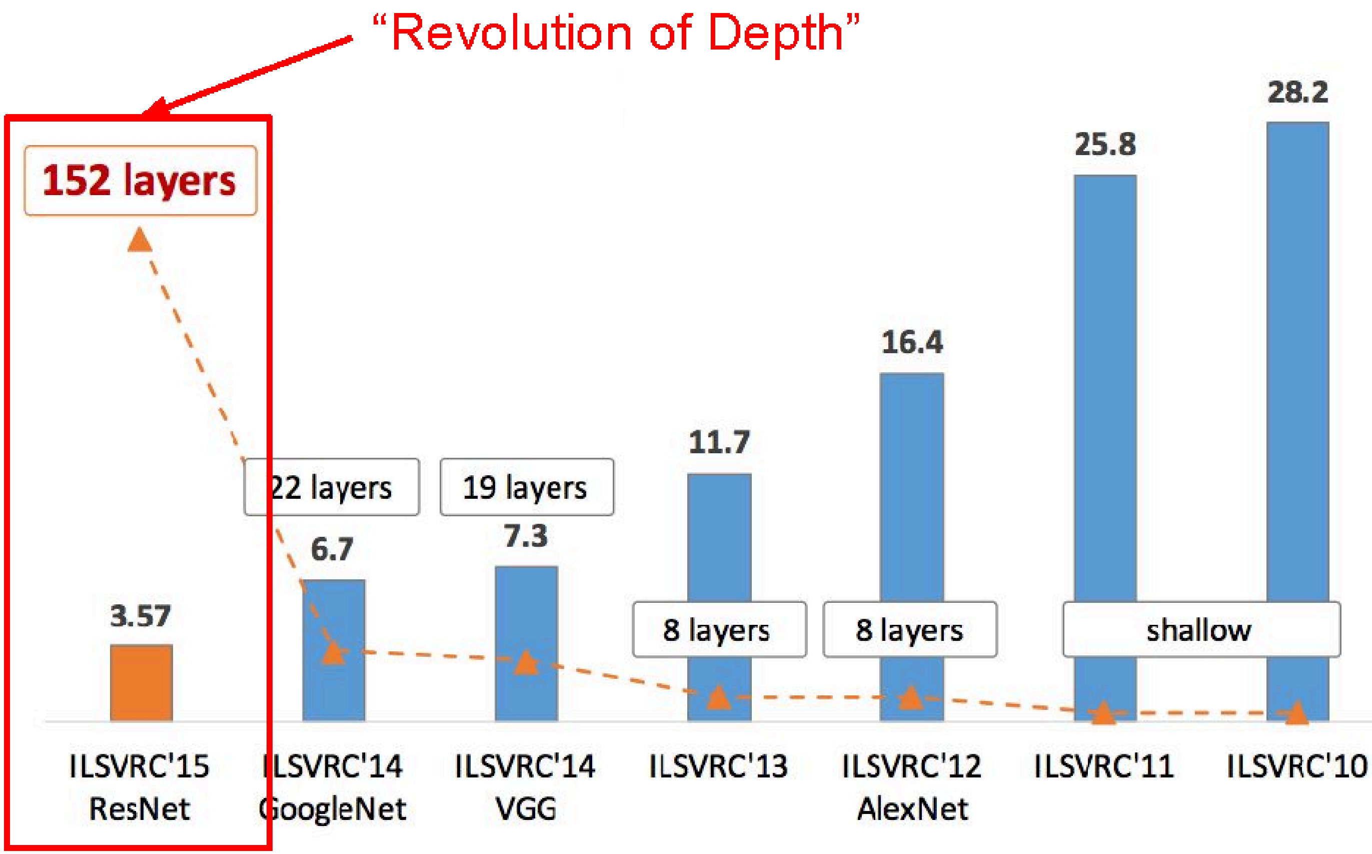
Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

Forward pass time and power consumption



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

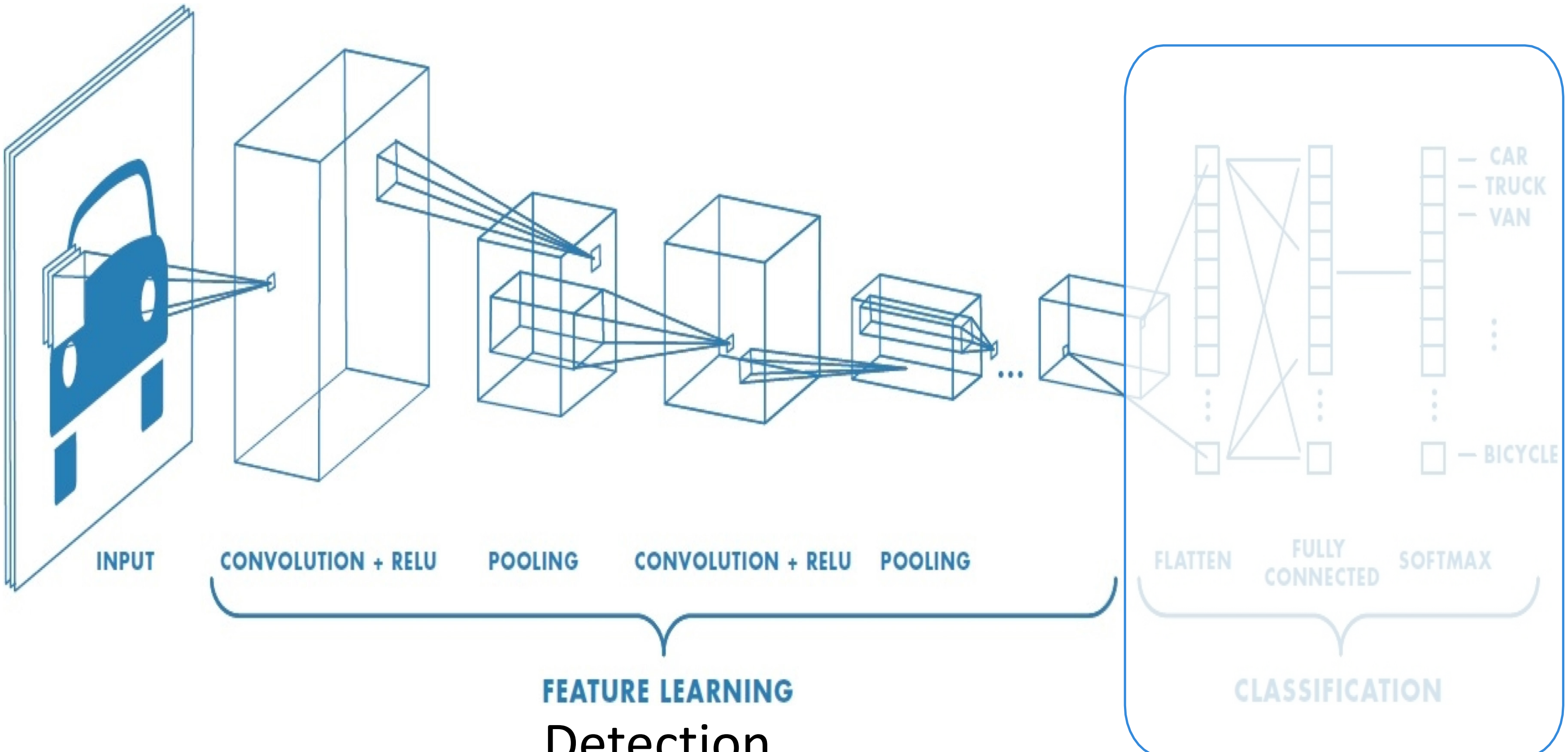
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

Countless applications

An Architecture for Many Applications



Detection

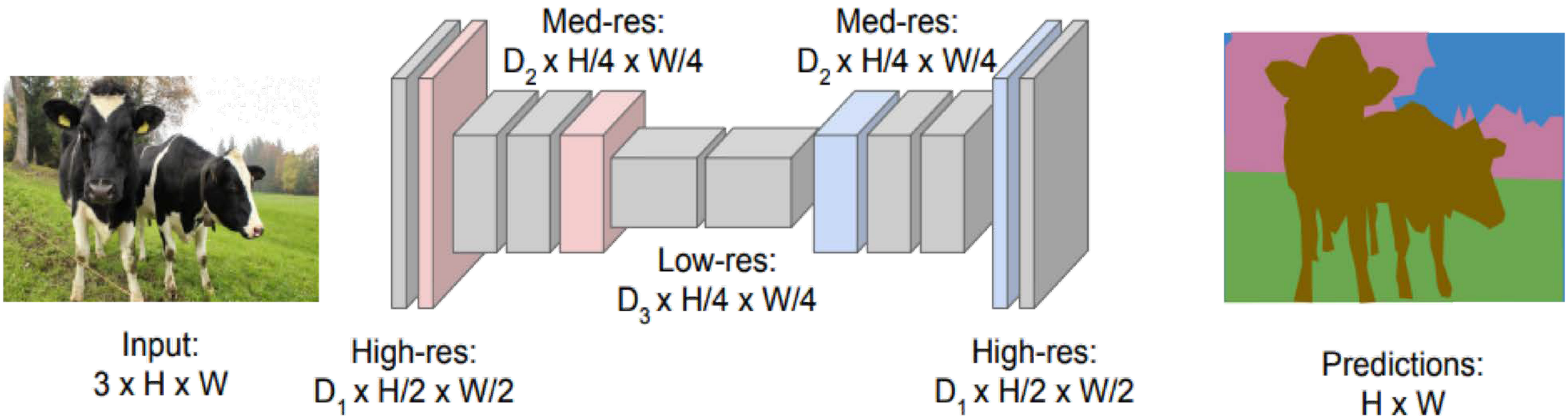
Semantic segmentation

End-to-end robotic control

Semantic Segmentation: Fully Convolutional Networks

FCN: Fully Convolutional Network.

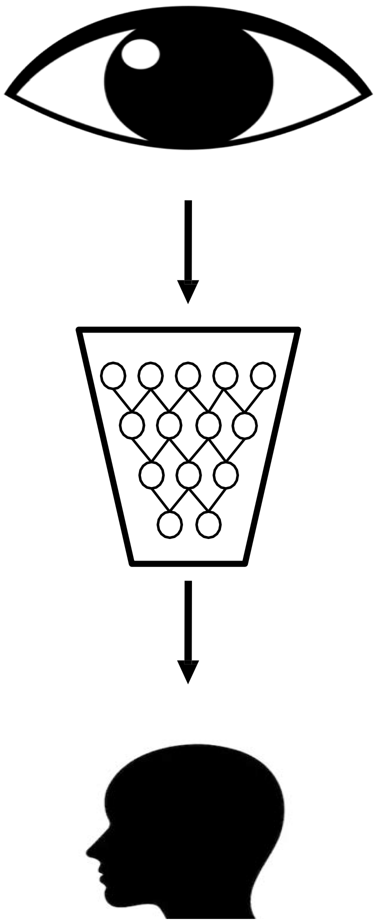
Network designed with all convolutional layers, with **downsampling** and **upsampling** operations

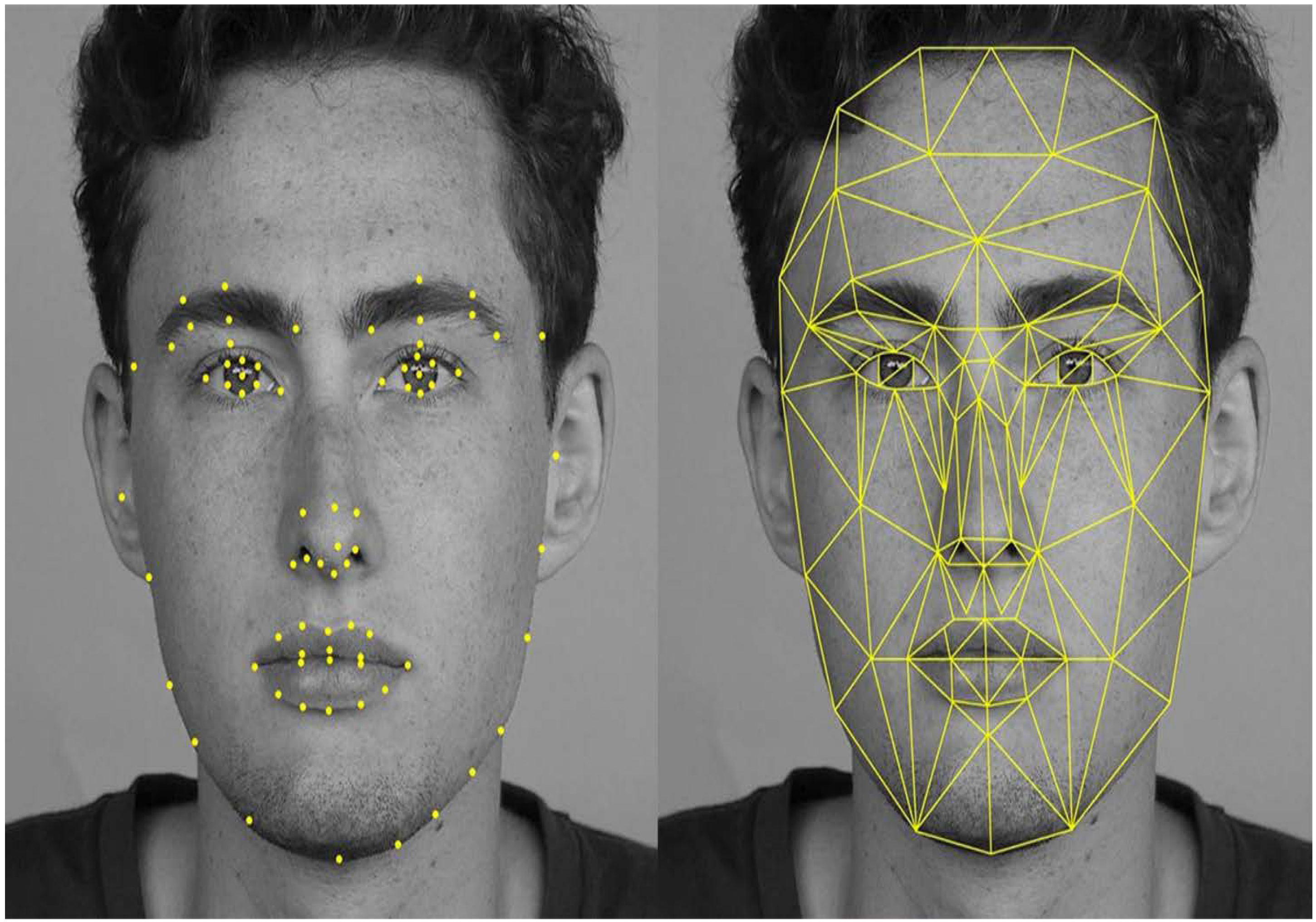



```
tf.keras.layers.Conv2DTranspose
```

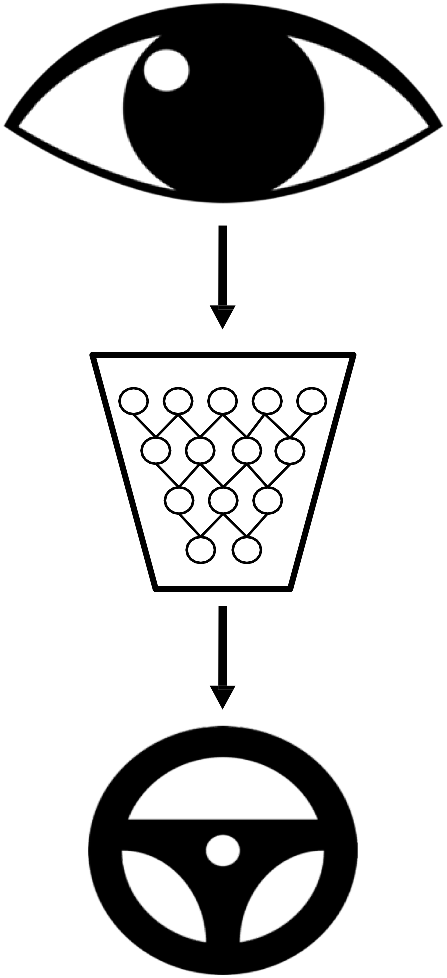
Long+ *CVPR* 2015

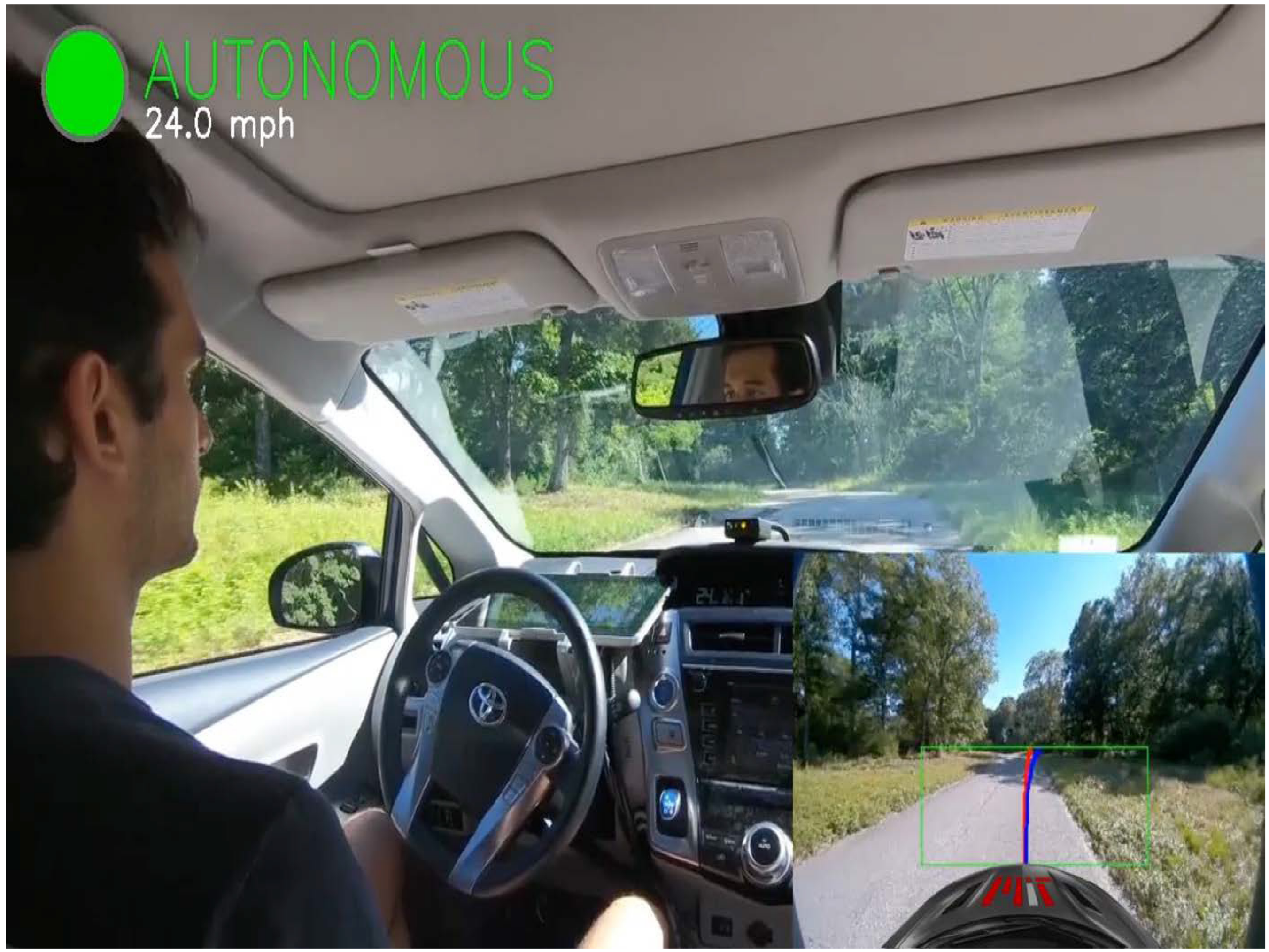
Facial Detection & Recognition





Self-Driving Cars



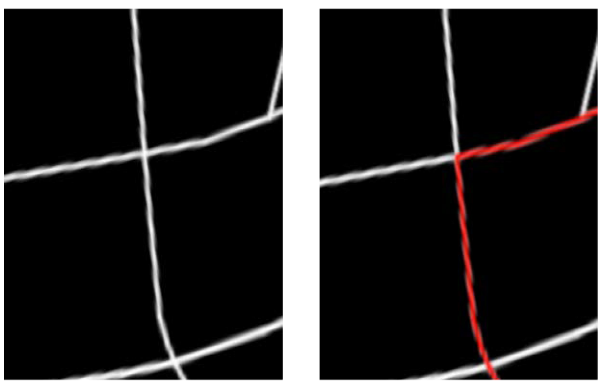


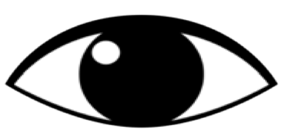
Amini+ *ICRA* 2019.

Self-Driving Cars: Navigation from Visual Perception

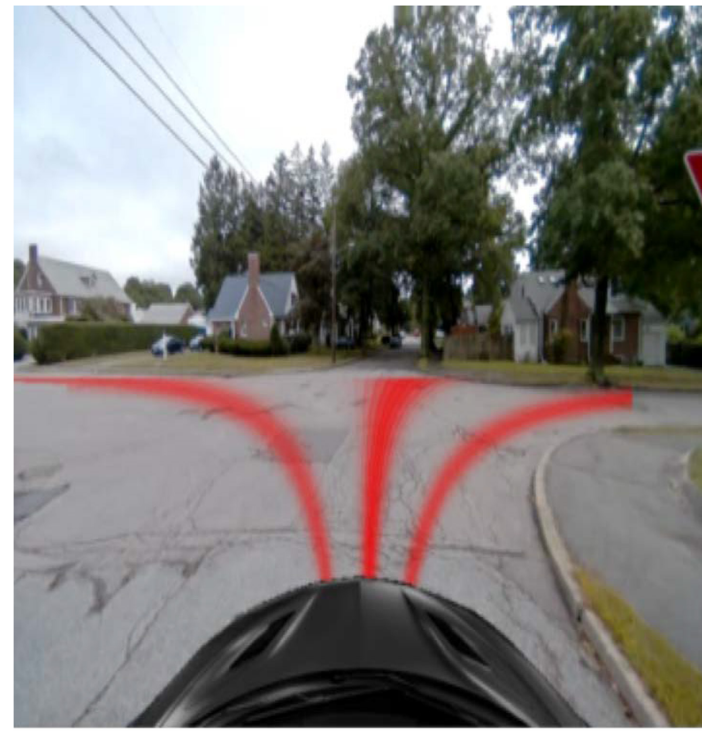
Raw Perception
I
(ex. camera)

Coarse Maps
M
(ex. GPS)



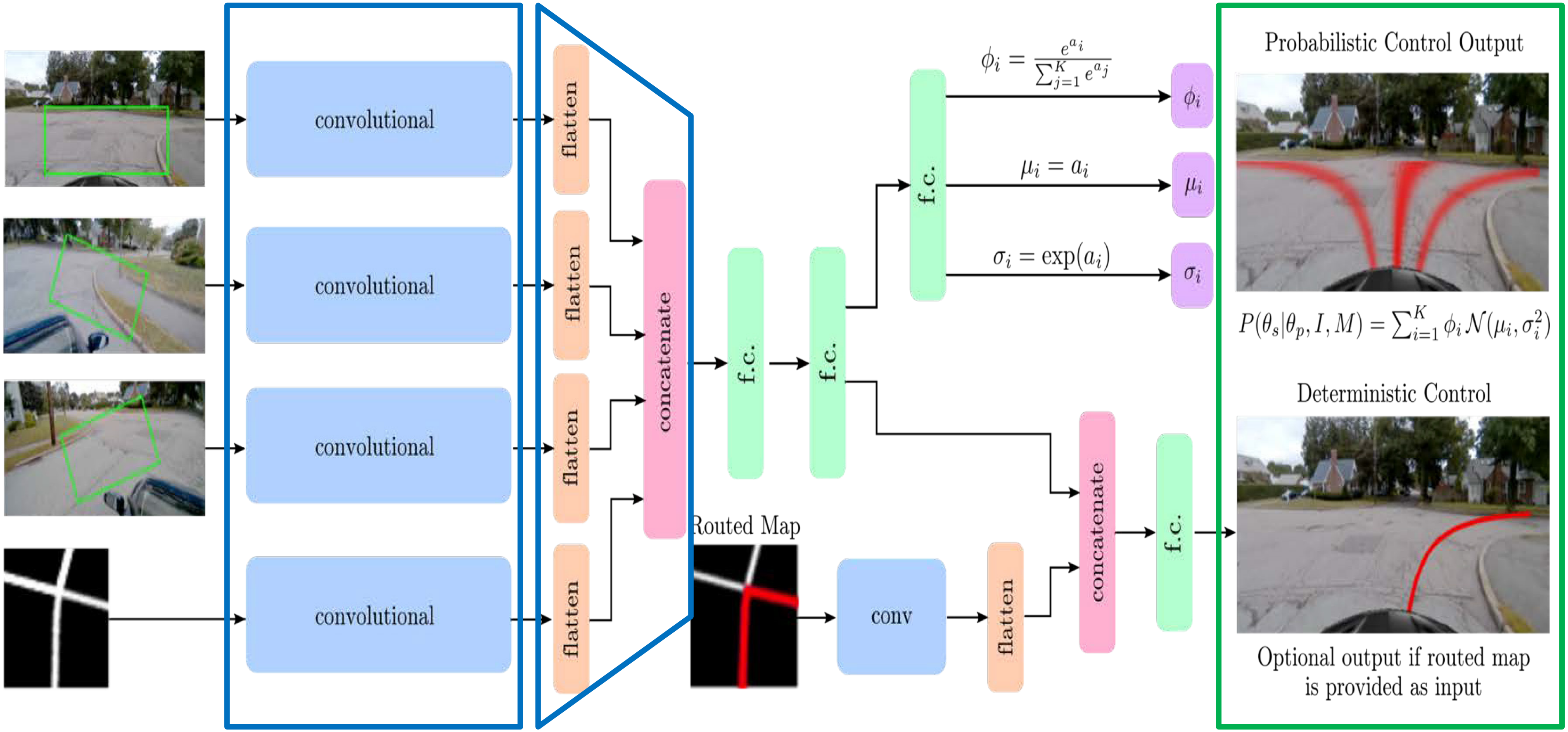


Possible Control Commands



Amini+ *ICRA* 2019

End-to-End Framework for Autonomous Navigation

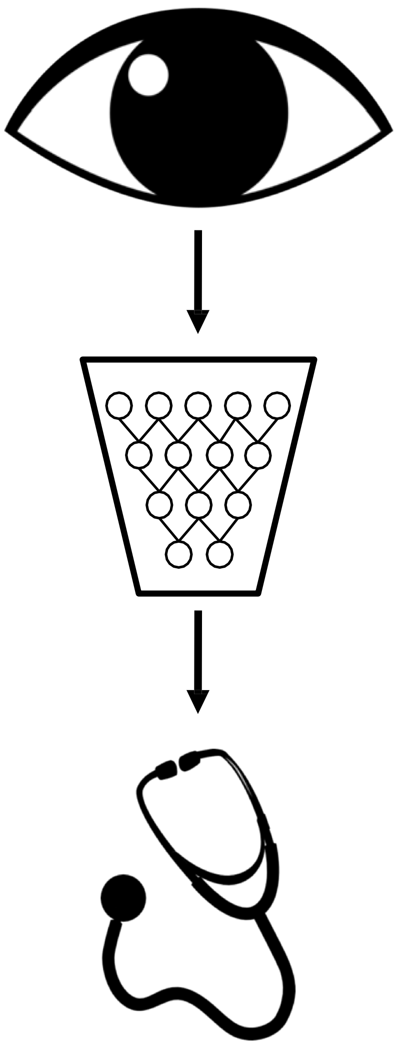


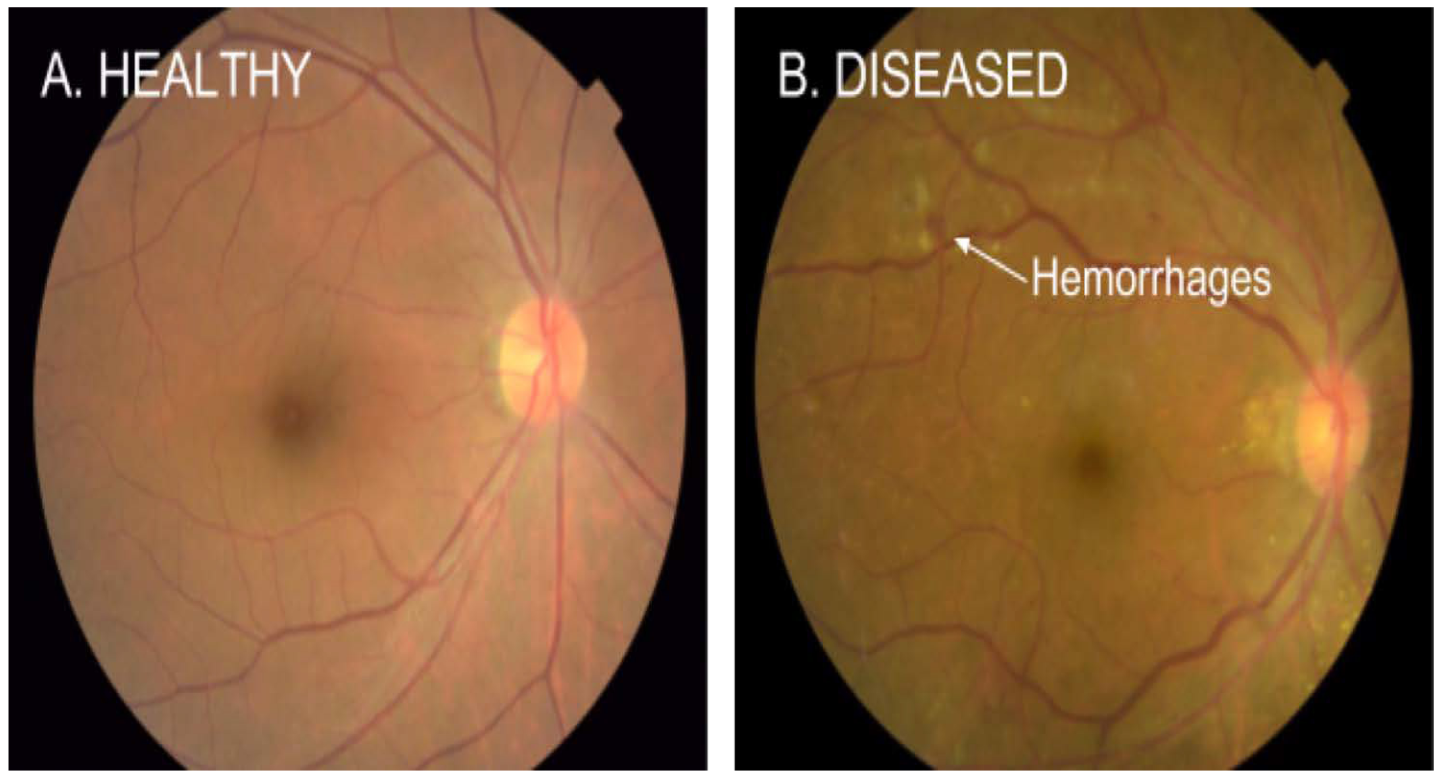
Amini+ *ICRA* 2019

Entire model trained end-to-end

without any human labelling or annotations

Medicine, Biology, Healthcare



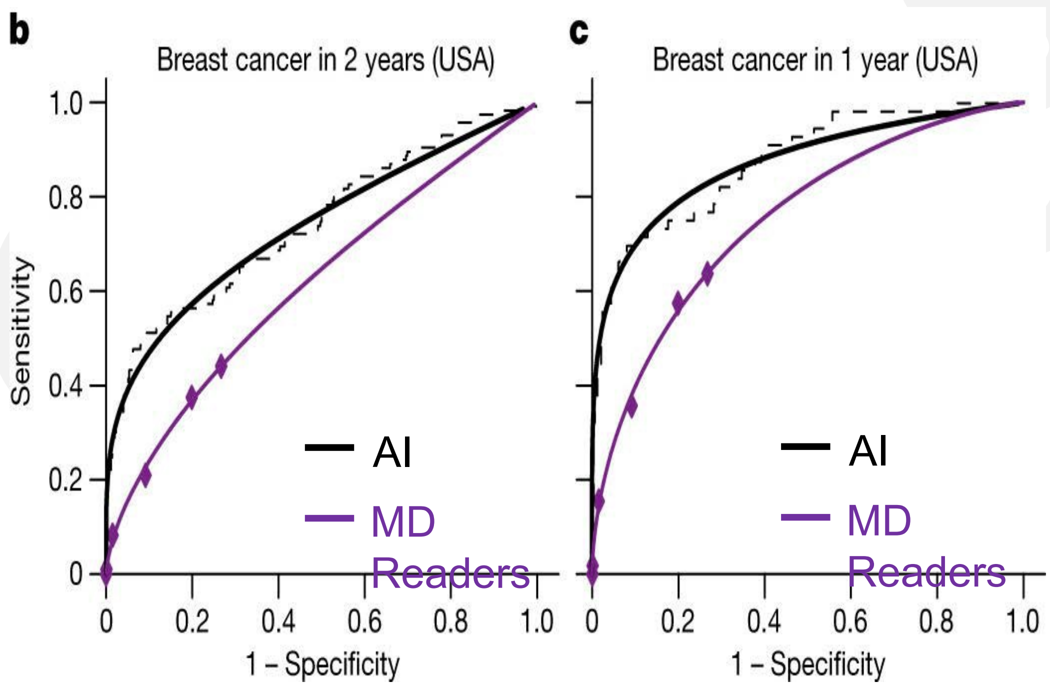


Gulshan+ *JAMA* 2016.

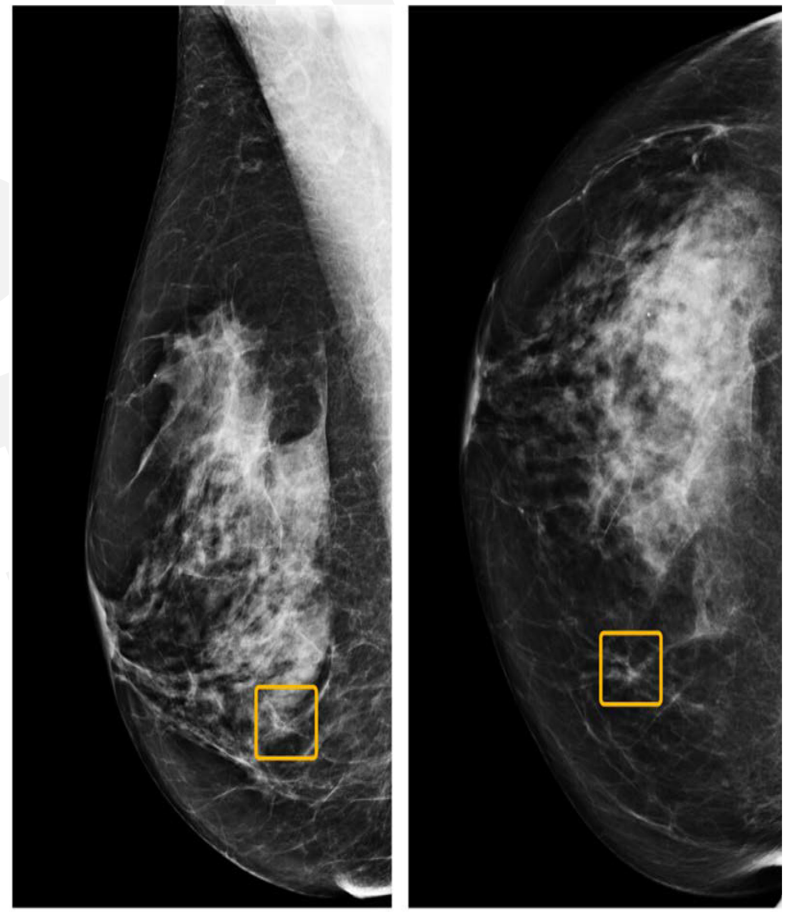
Breast Cancer Screening

International evaluation of an AI system for breast cancer screening

nature



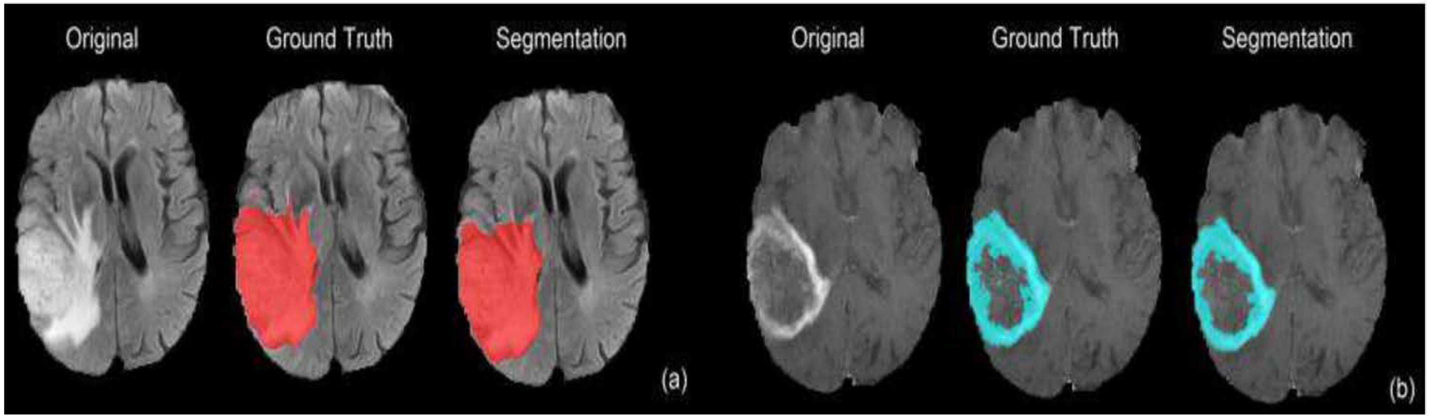
CNN-based system outperformed expert radiologists at detecting breast cancer from mammograms



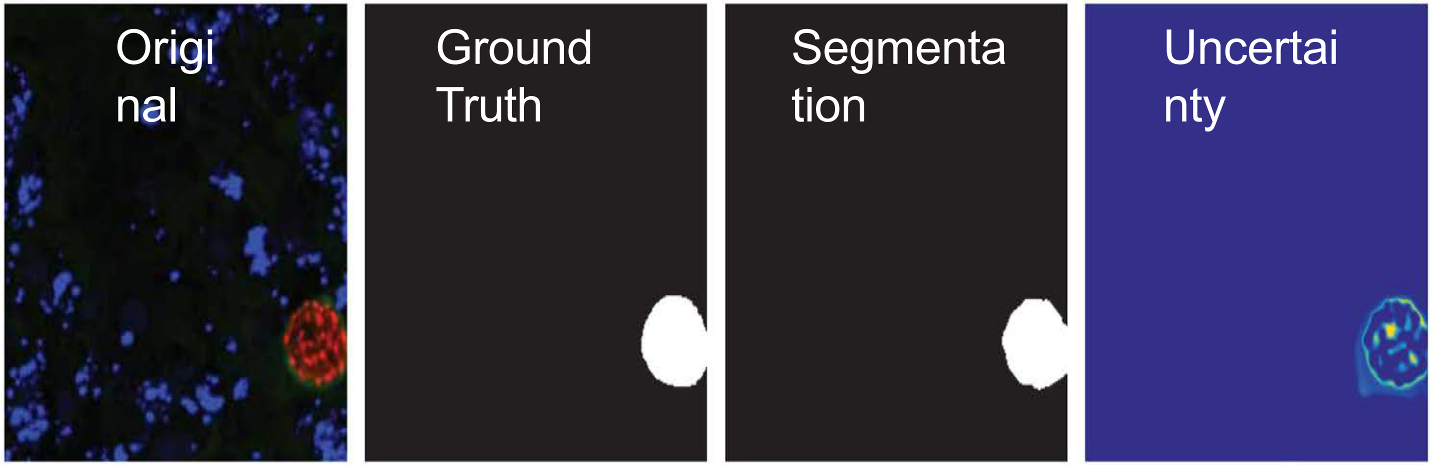
Breast cancer case missed by radiologist but detected by AI

Semantic Segmentation: Biomedical Image Analysis

Brain Tumors
Dong+ *MIUA* 2017.

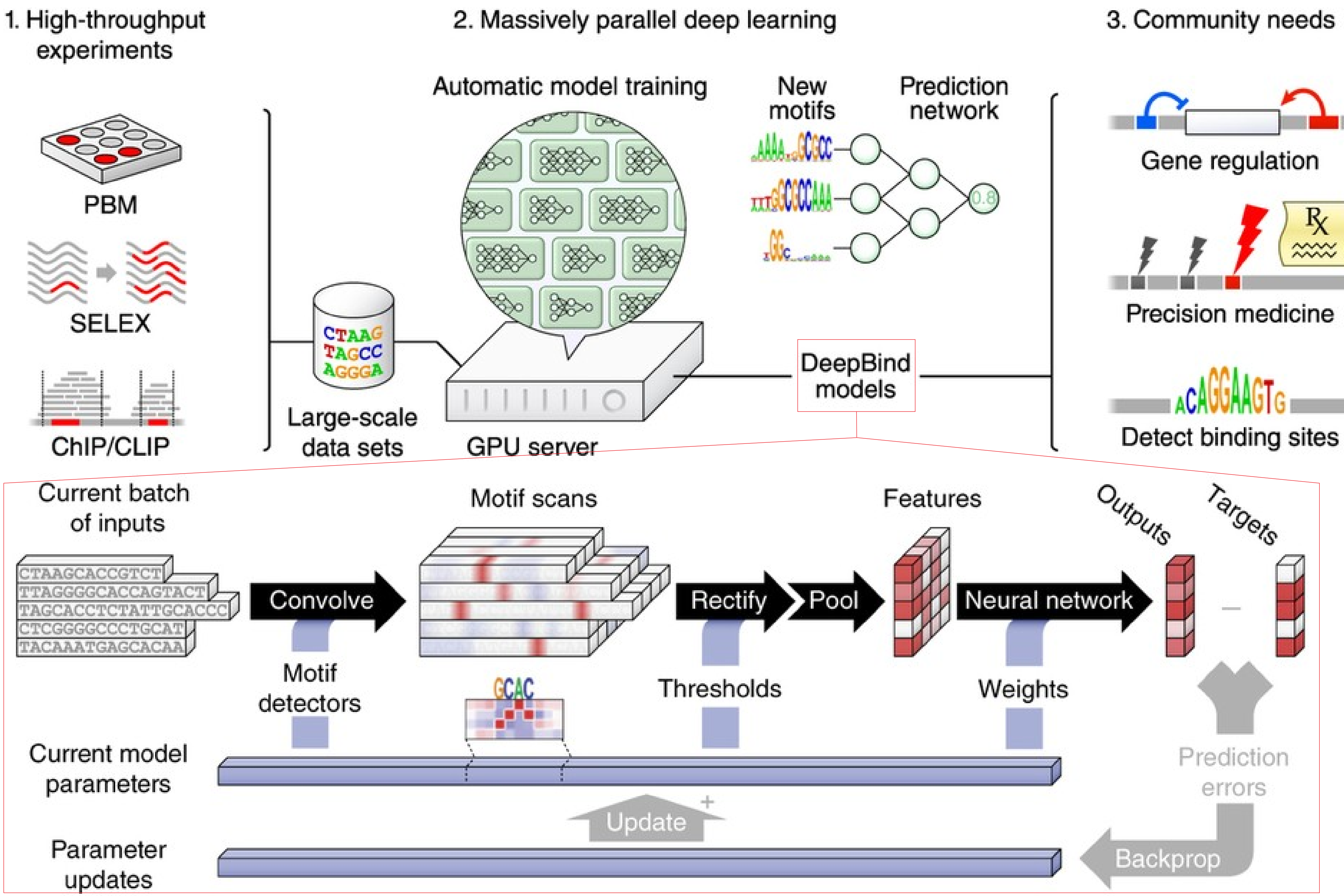


Malaria Infection
Soleimany+ *arXiv* 2019.



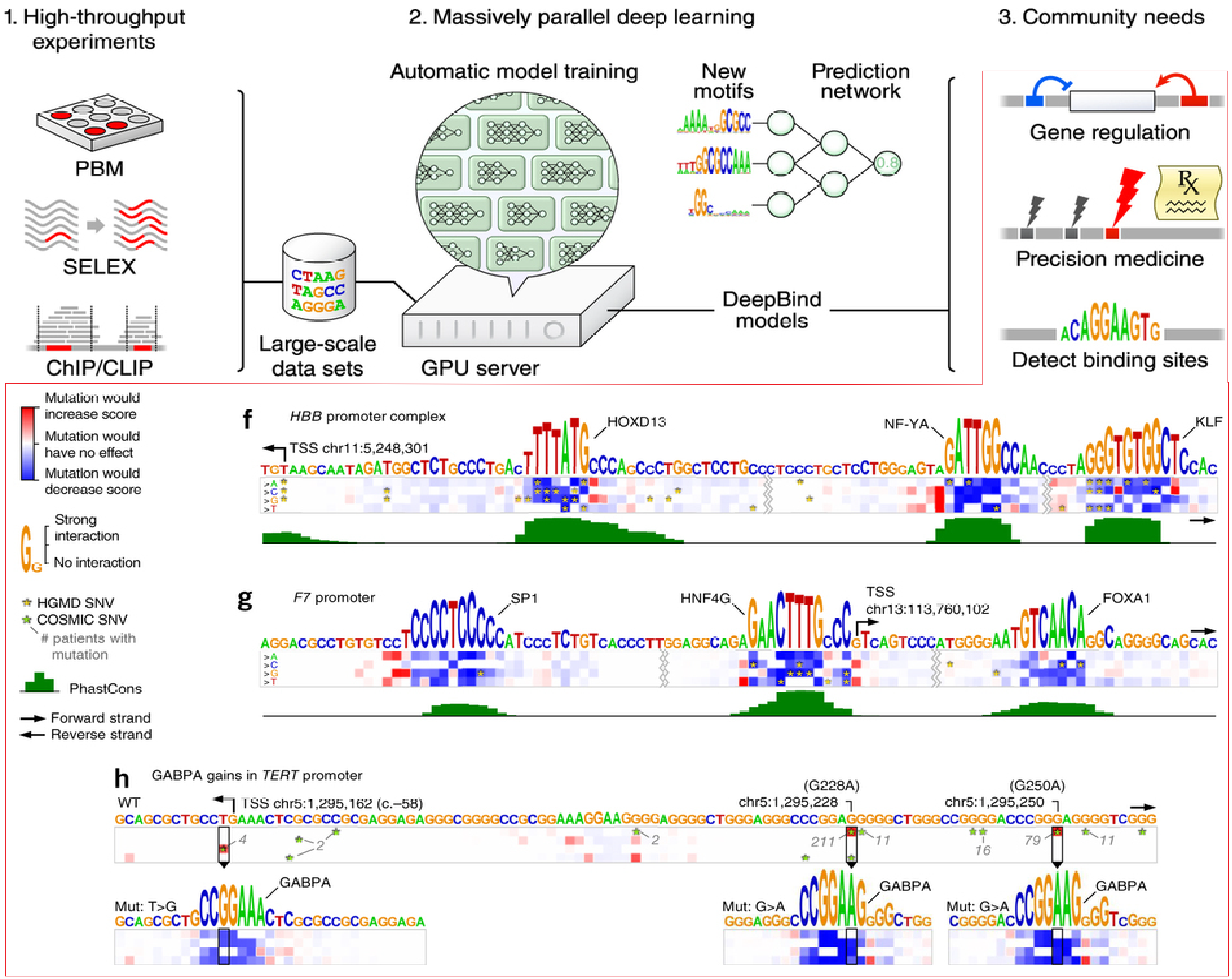
Dong+ *MIUA* 2017; Soleimany+ *arXiv* 2019

DeepBind



[Alipanahi et al., 2015]

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