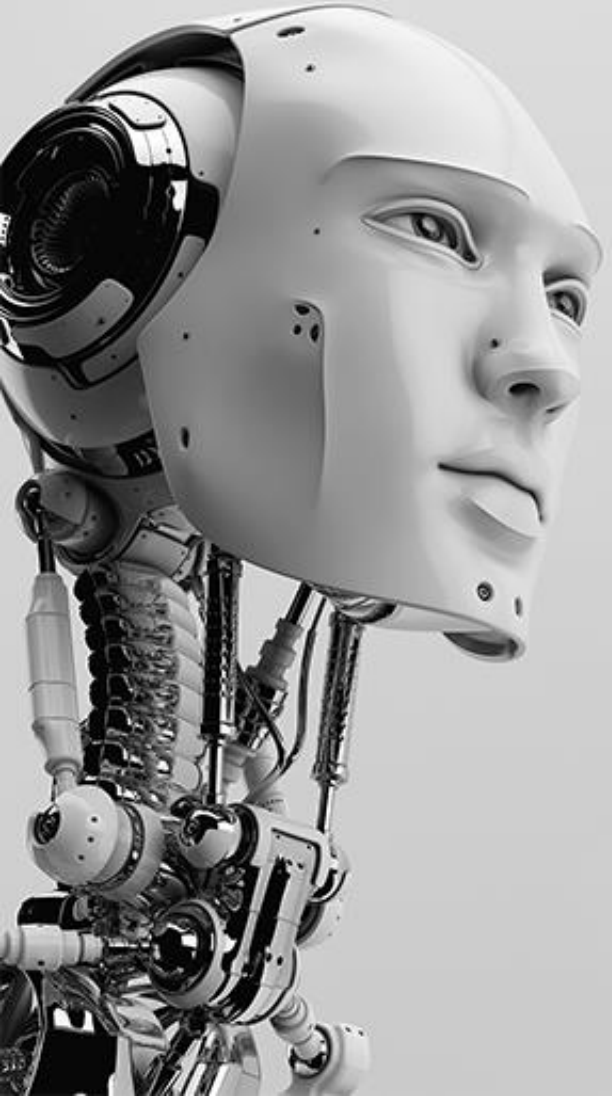


Deep Learning for Autonomous Systems and Robotics

Chris DeBellis
RelationalAI

October 9, 2019

www.linkedin.com/in/chrisdebellis
https://github.com/ctdebellis/GSU_Oct_2019

My Journey to AI and Deep Learning

Georgia Tech College of Engineering

Georgia State University

Honeywell
THE POWER OF CONNECTED

rAI relationalAI

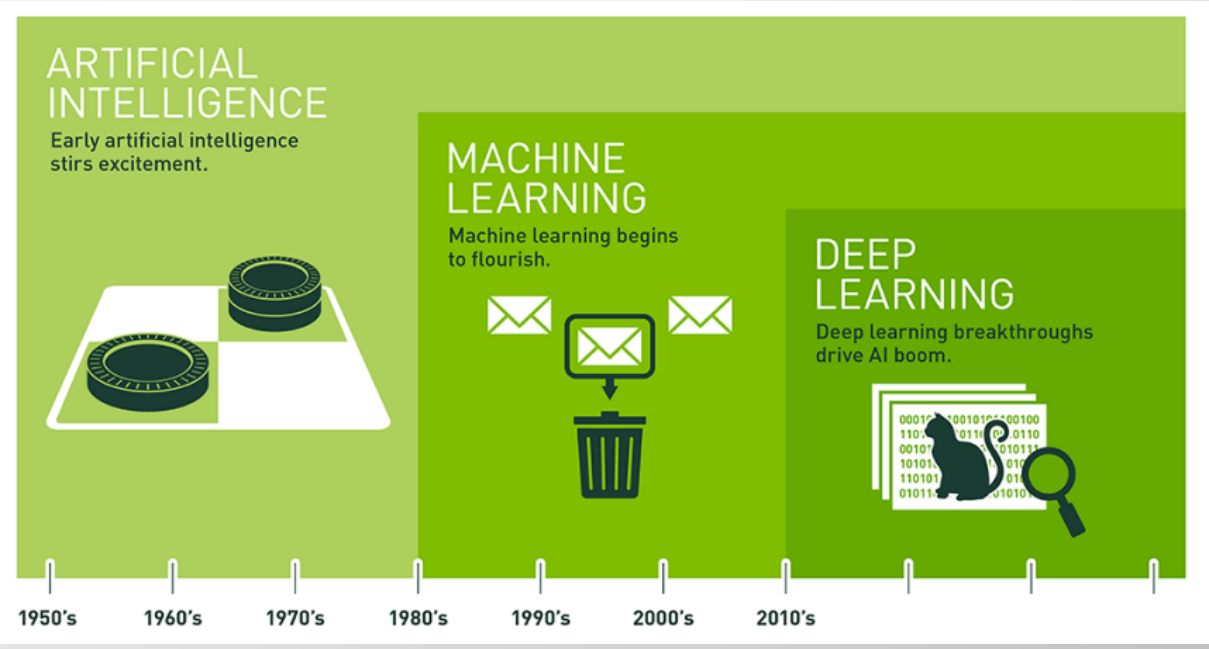
Evolving Definition of AI

- John McCarthy first coined the term in 1956
  - concepts around “thinking machines”

- AI has become synonymous with systems that can perform a specific task as good as a human – playing games, driving, chat bots, decision making, detecting patters / anomalies, computer vision, stock trading, natural language understanding, hacking, military application …

- In 2016, several industry leaders including Amazon, Apple, DeepMind, Google, IBM and Microsoft joined together to create Partnership on AI to Benefit People and Society to develop and share best practices, advance public understanding, provide an open platform for discussion and to identify aspirational effort in AI for socially beneficial purposes

Source: Wikipedia



https://acadgild.com/blog/data-science-deeplearning

Data Science

Gather, clean and analyze data
Data is often structured

Statistical analysis and distributions

R and Python common

Well established methods –
Regression (Ridge, Lasso, Logistic)
Decision Trees, kNN, SVM, Random Forest, Boosting, etc.

May use Deep Learning algorithms.
Becoming more ubiquitous.

Visualizations are important

AI and Deep Learning

Unstructured data – images, voice, text
Collect Data + Synthesize data / Possibly simulate scenarios

New approaches constantly being developed
Read research papers and implement (code)

CNN, RNN, LSTM, GRU, Reinforcement Learning
Python with DL frameworks - Tensorflow, Caffe, PyTorch, etc.

More coding, system integration, deployment trade-offs and Hardware Optimization (especially on mobile devices)

Less emphasis on visualizations / insights / analytics

Solve different problems

Important Roles in AI

Deployment of Deep Learning models

- Edge – models deployed in devices
- ROS – Robot Operating System for robotics
- Cloud engineers – deploy models. Collect and process real time streaming data

Optimization

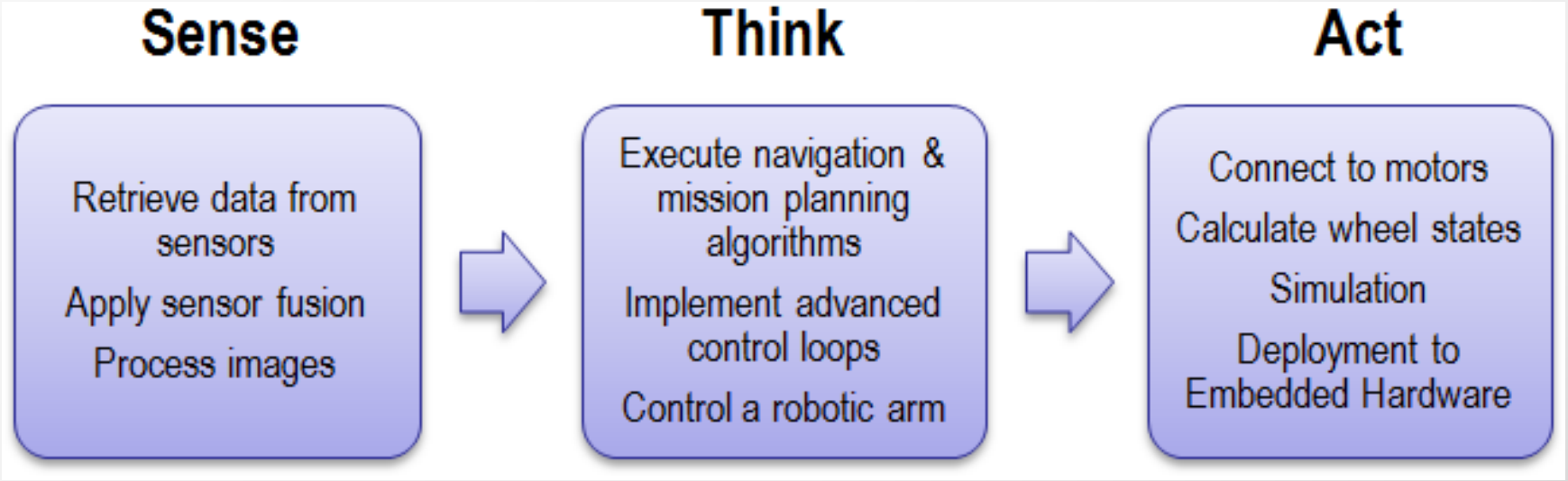
- Hardware accelerated implementations
  - -Nvidia TensorRT, Intel OpenVINO, Qualcomm SnapDragon
- Port to other languages (C++, Java for Android, Objective C for iOS, etc.)

Simulation – generate data or test and validate models

Product Managers, Project Managers, Sales and Marketing, Legal and Policy

And yes, Data Scientists too.

Deep Learning Trends for Robotic Functions



Perception Models (scene understanding and sensor fusion)

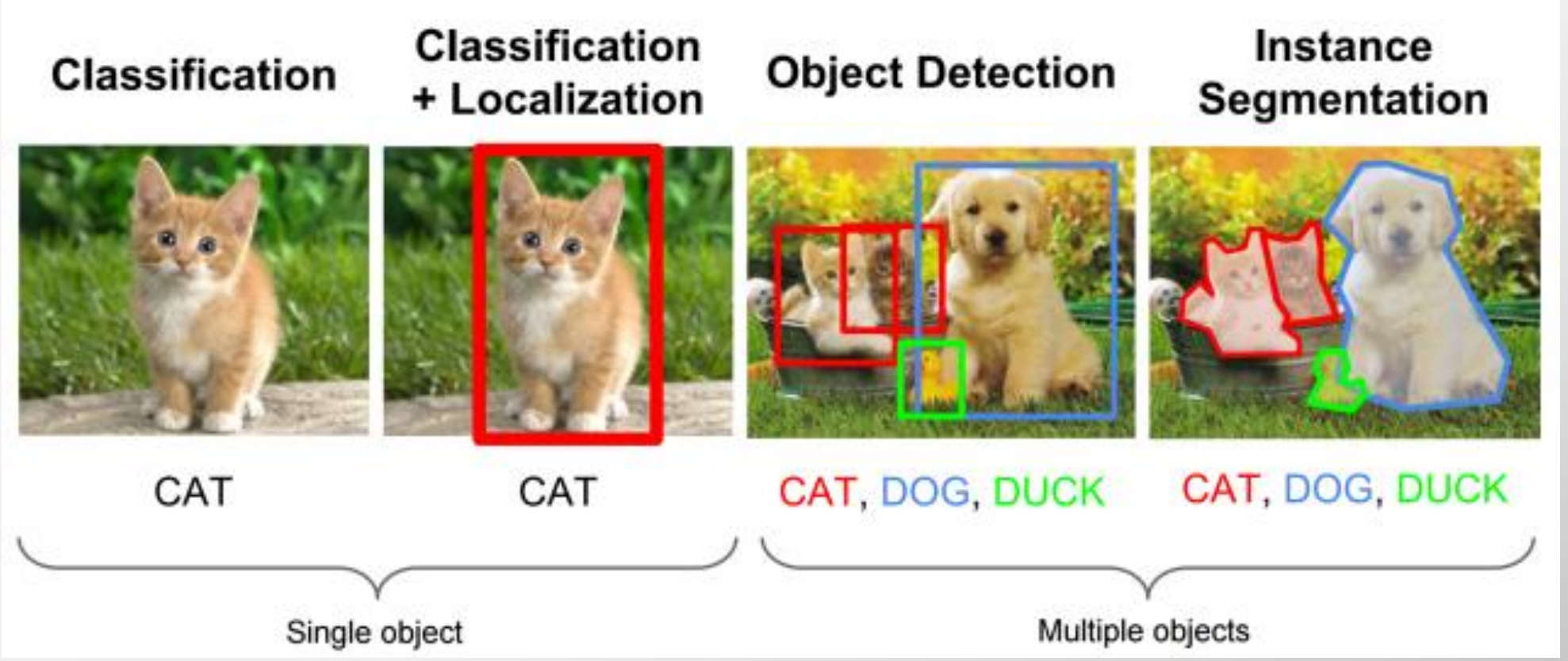
Reinforcement Learning Models (choose best action)

Data Collection for Model Training

2D Perception Examples

Machine Learning for Computer Vision

History of Deep Learning 2D Perception Tasks

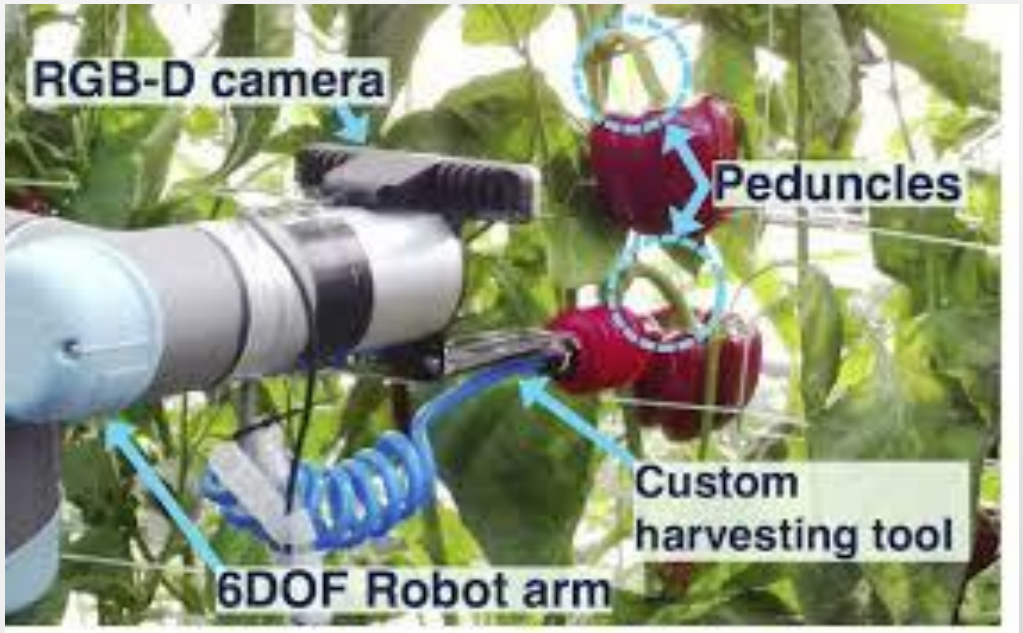


Traditional Computer Vision Techniques	AlexNet (2012) VGG (2014) ResNet (2015) ResNeXt (2016)	Classification + Bounding box regression	RCNN (2014) SSD (2015) YOLO (2016) Faster RCNN (2016)	Mask RCNN (2017) Dense Pose (2018)

AI / Autonomous Systems Need Perception

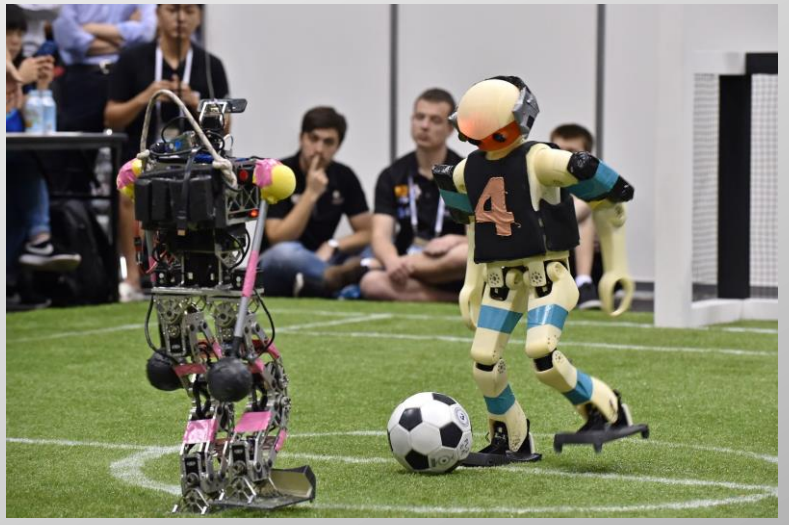
- Autonomous Vehicles
- Robotic Arms
- Sweet Pepper Picking Robot

https://www.youtube.com/watch?v=UlaNDm88yZo



Udacity Self Driving Car Nano-degree

Mask Vs. Bounding Box



Robo Cup 2017

Past Projects

- https://www.intelligrated.com/en/resources/videos/robotics-revolution-distribution-centers

Difficult computer vision problem (edges, color)

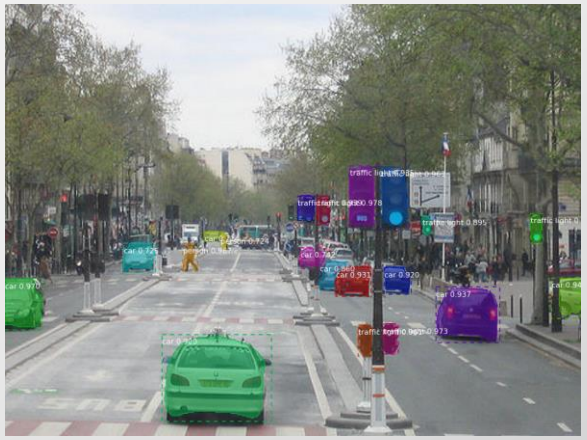
Model output - instance masks and classification (box, envelope, poly bag)

Input image

Model output - instance masks and classification
(Green = shipping label visible)

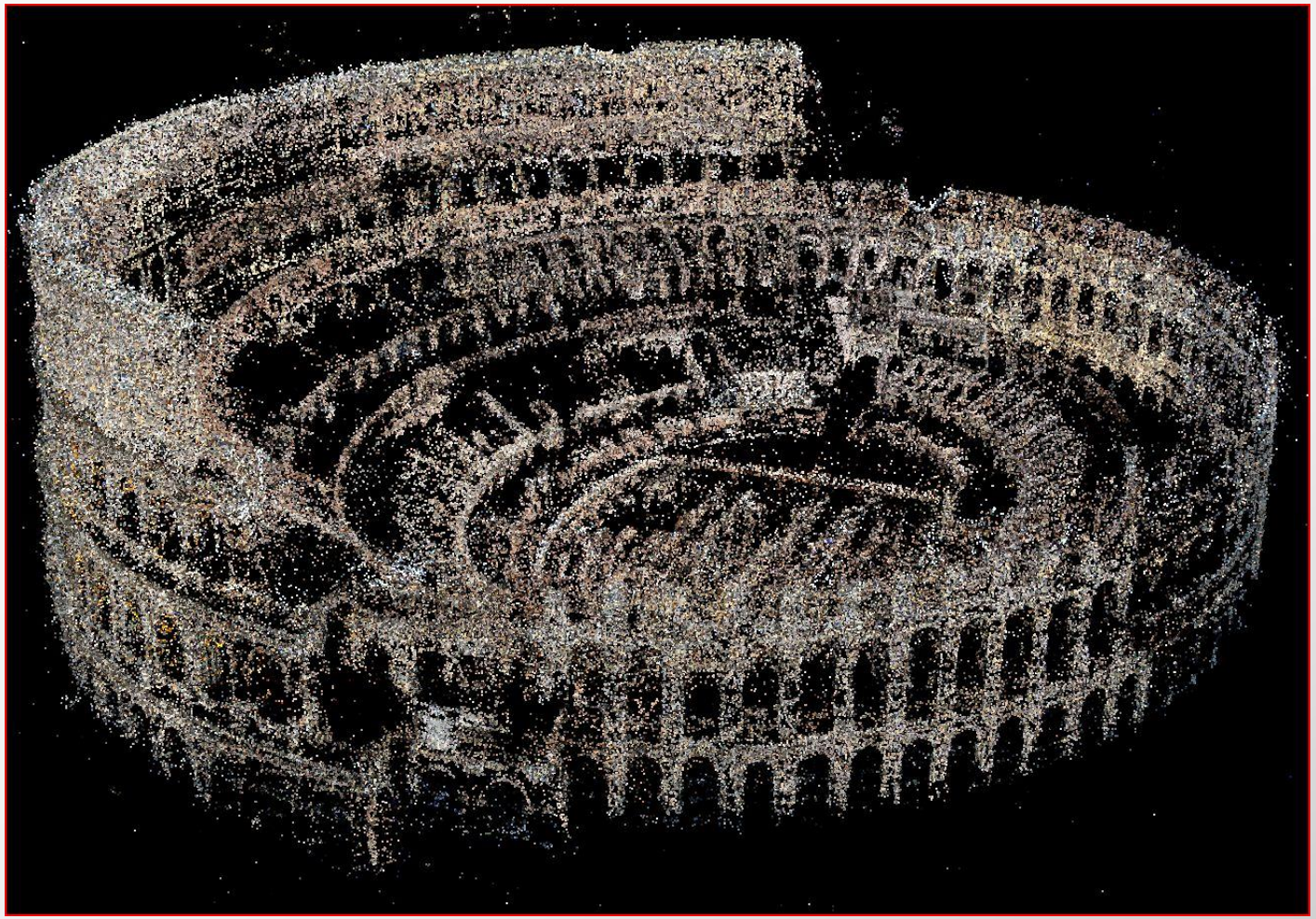
Perception

- Scene understanding
- Using sensors to interpret the environment data
  - 2D sensors
    - RGB color camera
    - Night Vision camera
    - Images
  - 3D sensors
    - RGB-D camera (RGB + Depth)
    - Lidar (uses light waves to measure distance)
    - Point cloud representation
    - Distance/Depth can also be calculated from 2 stereo RGB cameras
  - Other sensors
    - sound, orientation (gyroscope), position(GPS), distance (IR), color, acceleration, pressure, etc.



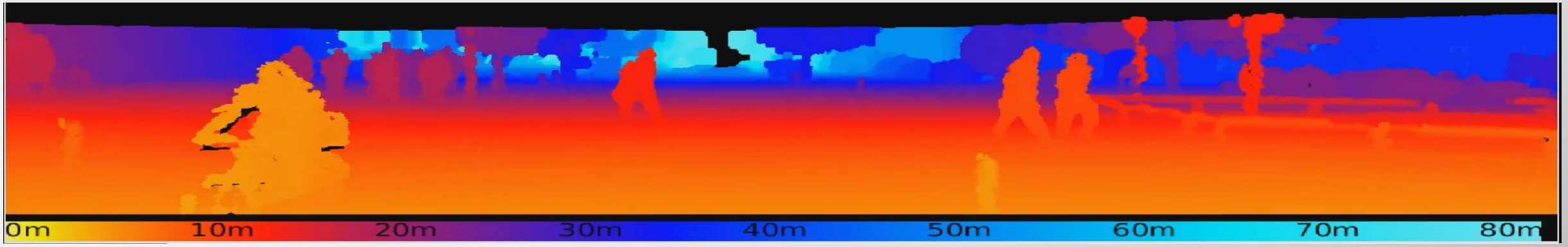
- You need multiple inputs to do robotics. May be a combination of traditional CV and deep learning approaches.

3D Point Cloud



- Data points in 3D space representing the surfaces of an object.
- Good for modeling
- Computationally expensive

3D Depth Maps



Depth maps convey distance, but are not sufficient for complete understanding of the environment

- Another algorithm is needed to classify objects (is the object directly ahead a vehicle or person?)
- Where is the road? (algorithm needed to detect the roads)

Source: http://ipi.ugent.be/ipi/drupal/intel_veh

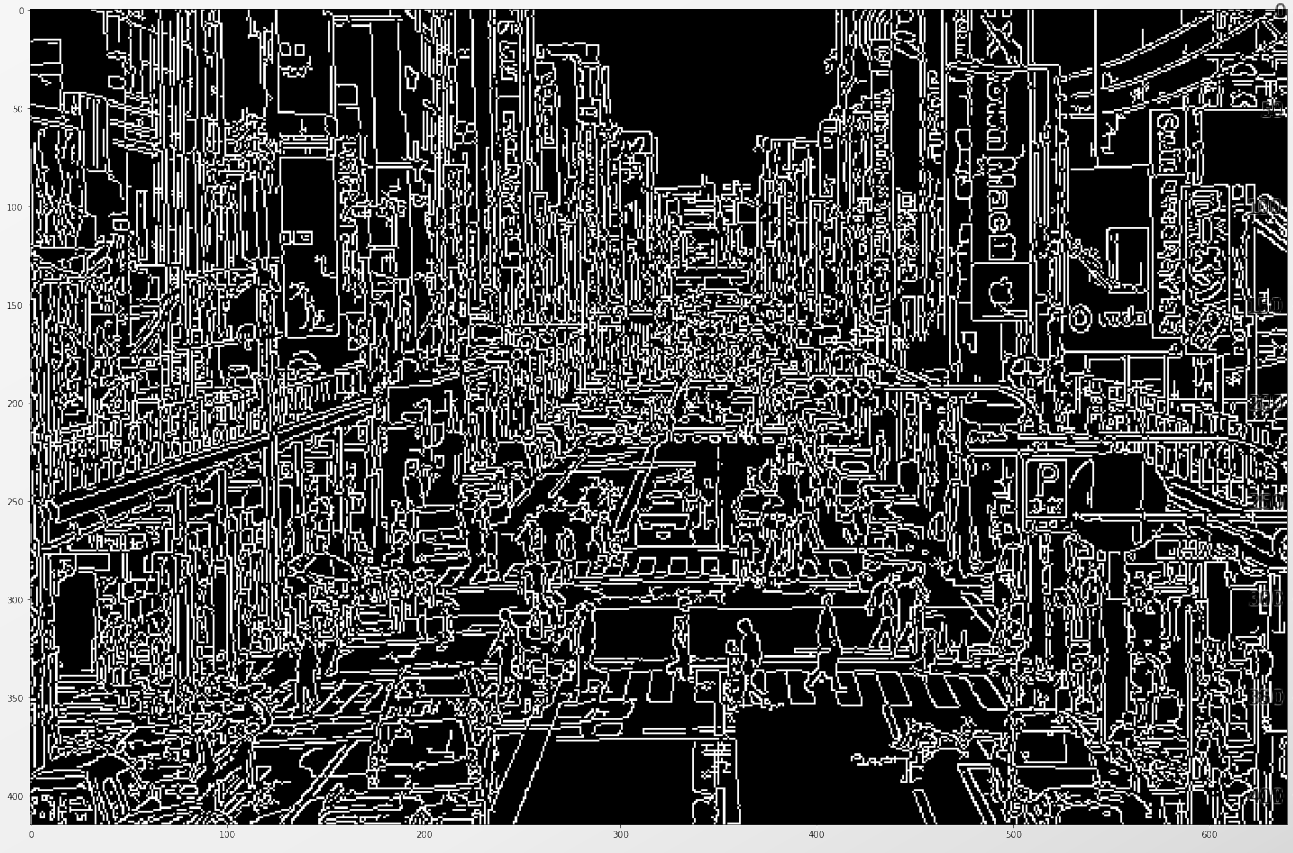
2D Image and 3D Prediction Superimposed

Align Using Transformation Matrices

Source: https://www.businessinsider.com/apple-self-driving-car-crash-2018-8

Multiple Inputs Create a Richer Understanding

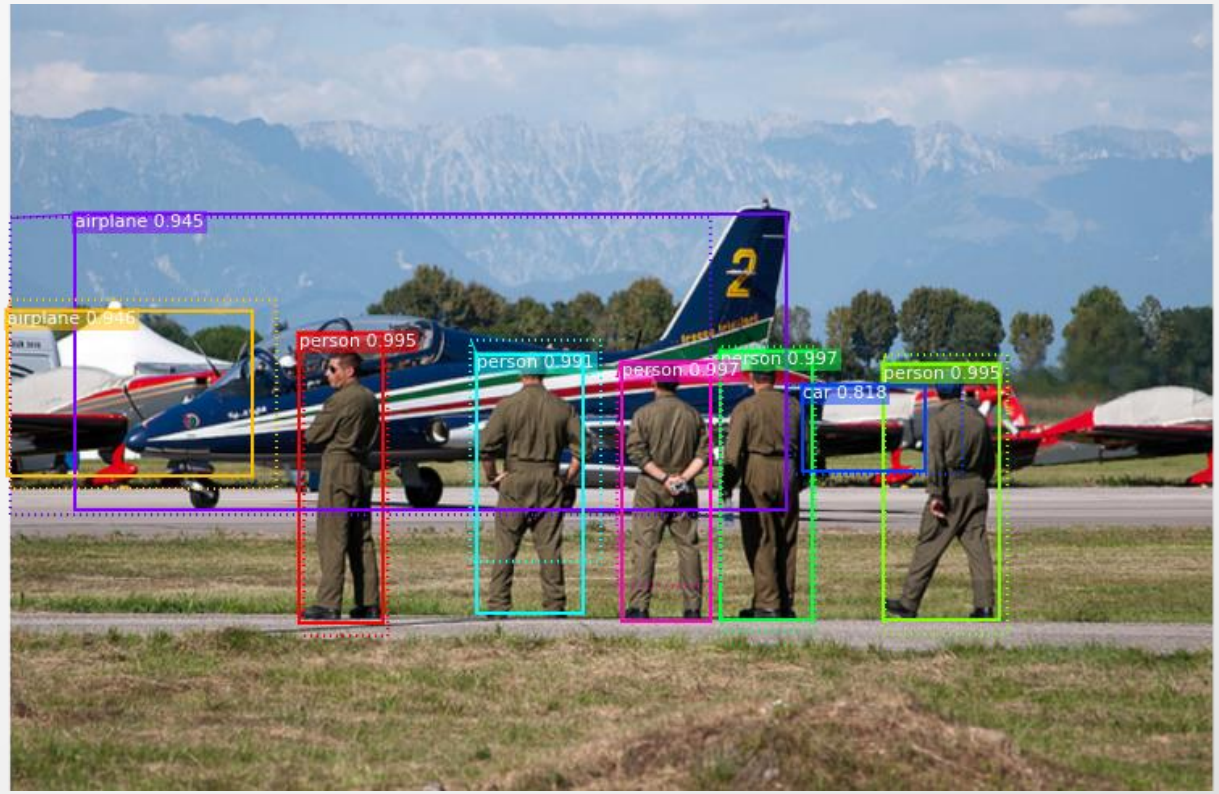
Traditional CV Canny edge detection



Edges plus Mask RCNN model output superimposed on image

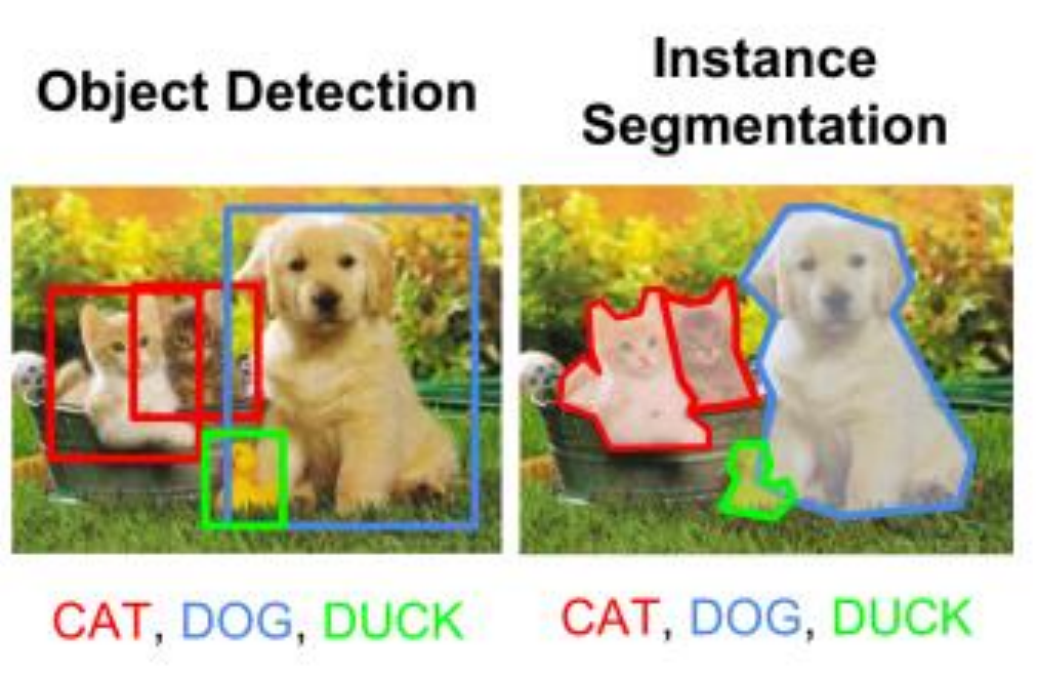
Images source: Chris DeBellis

Bounding Boxes vs. Masks



https://github.com/matterport/Mask_RCNN/

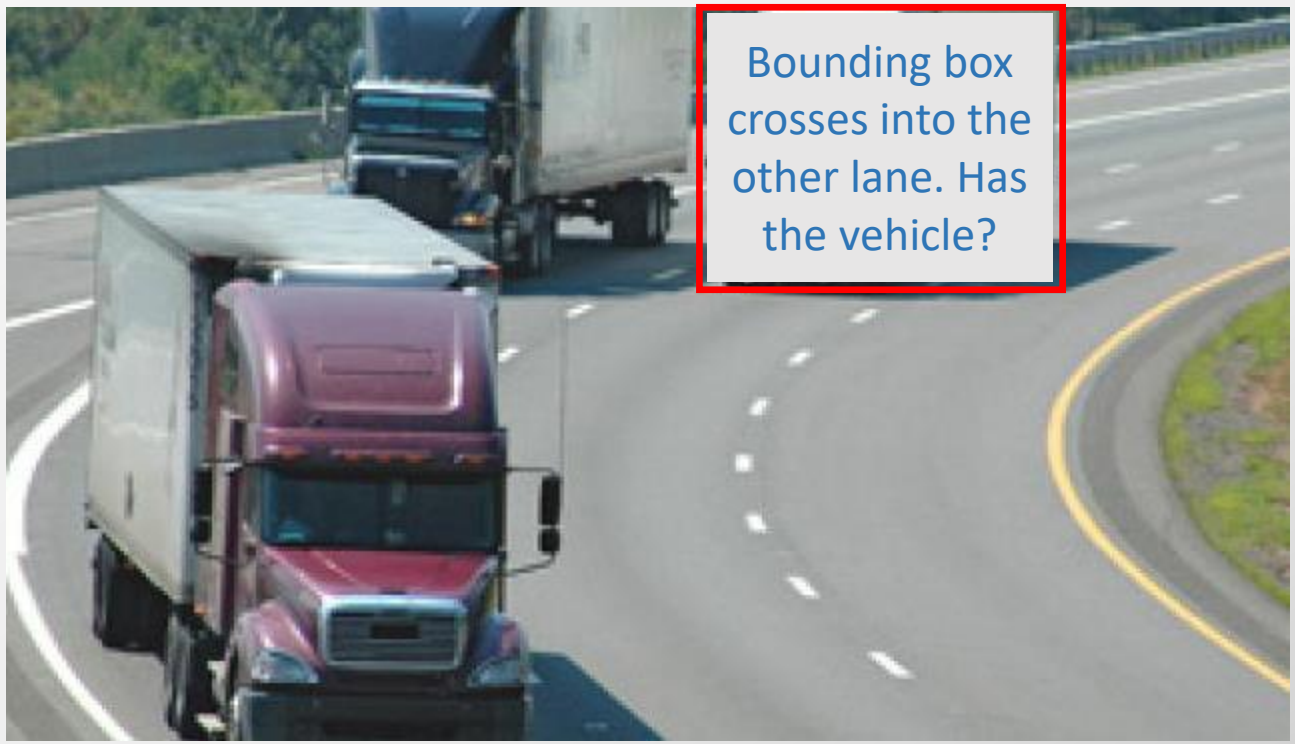
Bounding Boxes and Masks



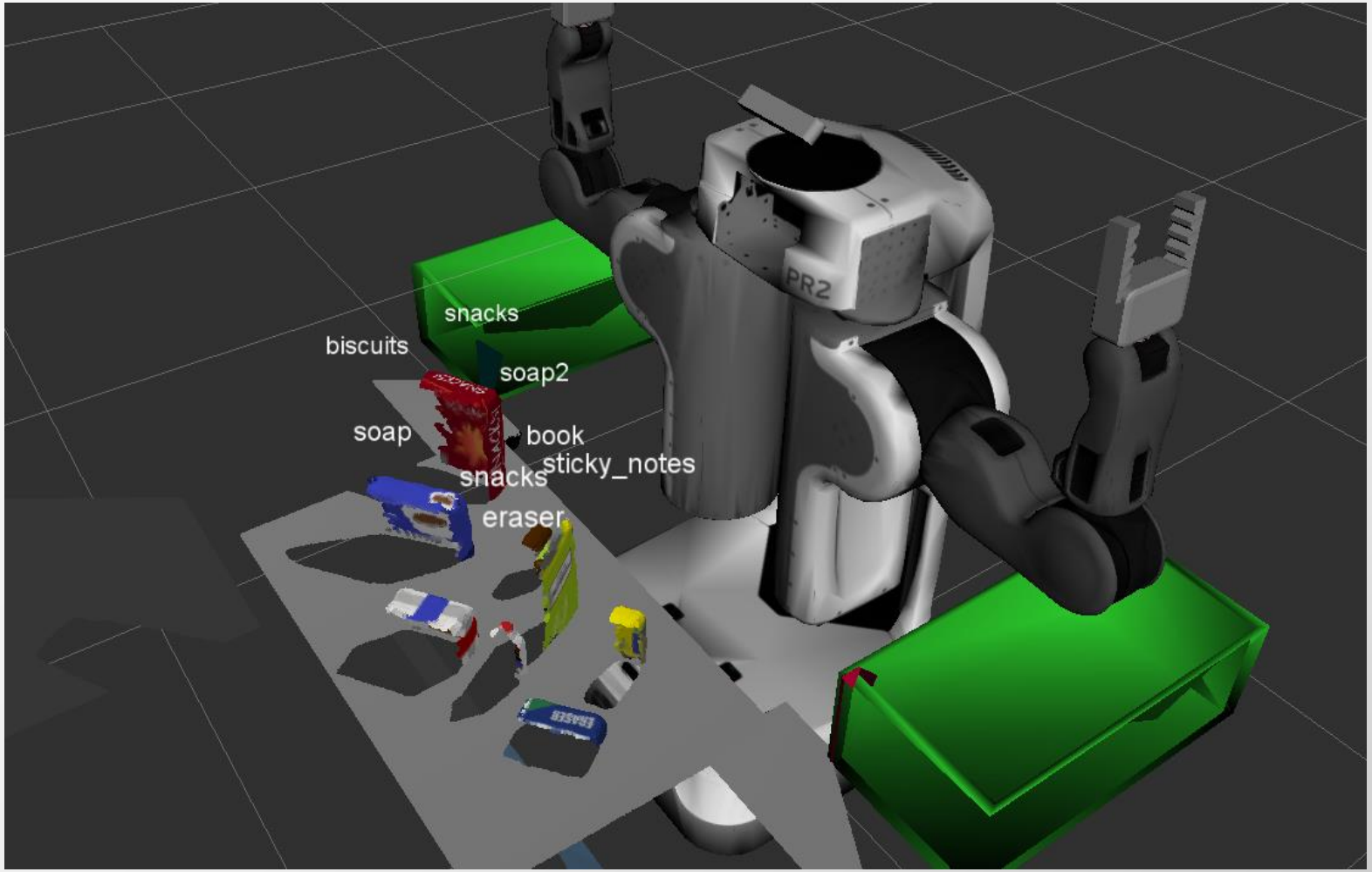
- Bounding boxes encompass the entire object → contains some background
- Masks include only the pixels containing the identified object

Bounding Box Limitations

- Bounding boxes are less accurate and convey less information
  - Objects that are not oriented in a perpendicular frame of reference
  - Objects that are not symmetric



Bounding Boxes Do Not Convey Orientation



Bounding boxes tell us
somewhere within this box
is the object.

Does not
provide the orientation.

How should the arm be positioned?

Source: Udacity Robotics Nanodegree program

Masks have drawbacks too.
Any ideas what those are?

Mask Drawbacks

- Training is more difficult
  - Need more data to train masks
  - Harder to train multiple outputs (loss functions) simultaneously
  - Tedious to label the training data (ground truth) –needed for supervised learning
  - Labeling masks is tedious and time consuming vs. labeling bunding boxes

- Computationally intensive
  - Longer to do inference (predictions) and training
  - May only get a few frames per second on streaming video even with a very good GPU
  - And only at lower resolution

- Not 100% accurate despite the extra complexity
- As with any algorithm the quality of the input affects the predictions
- Often bounding boxes are sufficient (e.g.) detecting other cars vs. detecting roads

Outlining the Masks for Training Can Be Tedious

Source: Mighty AI

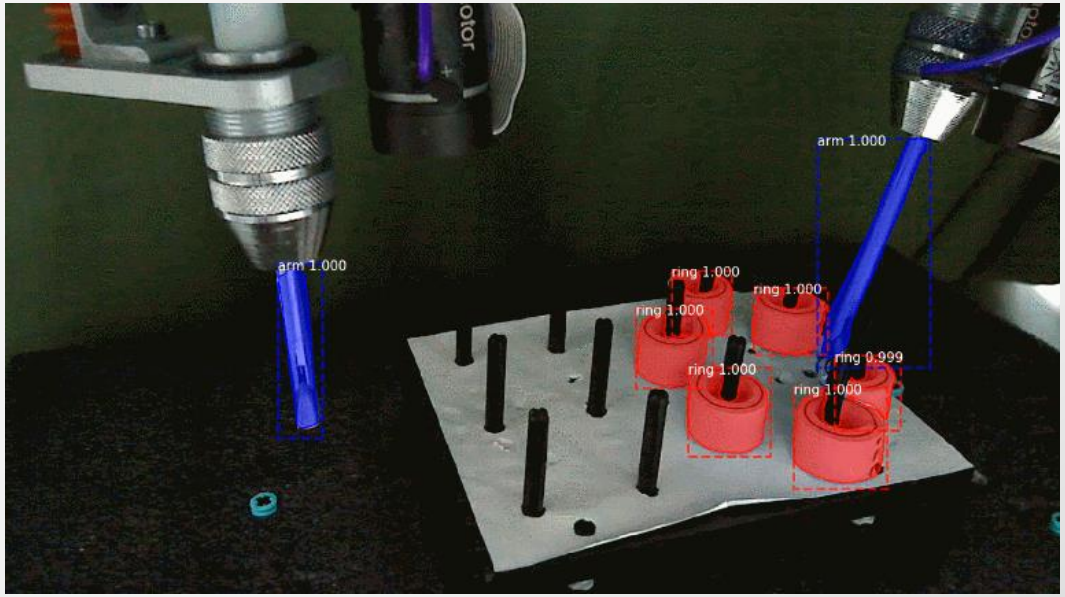
Source: Kitti Vision Benchmark

Supervised Learning requires labeled (annotated) data

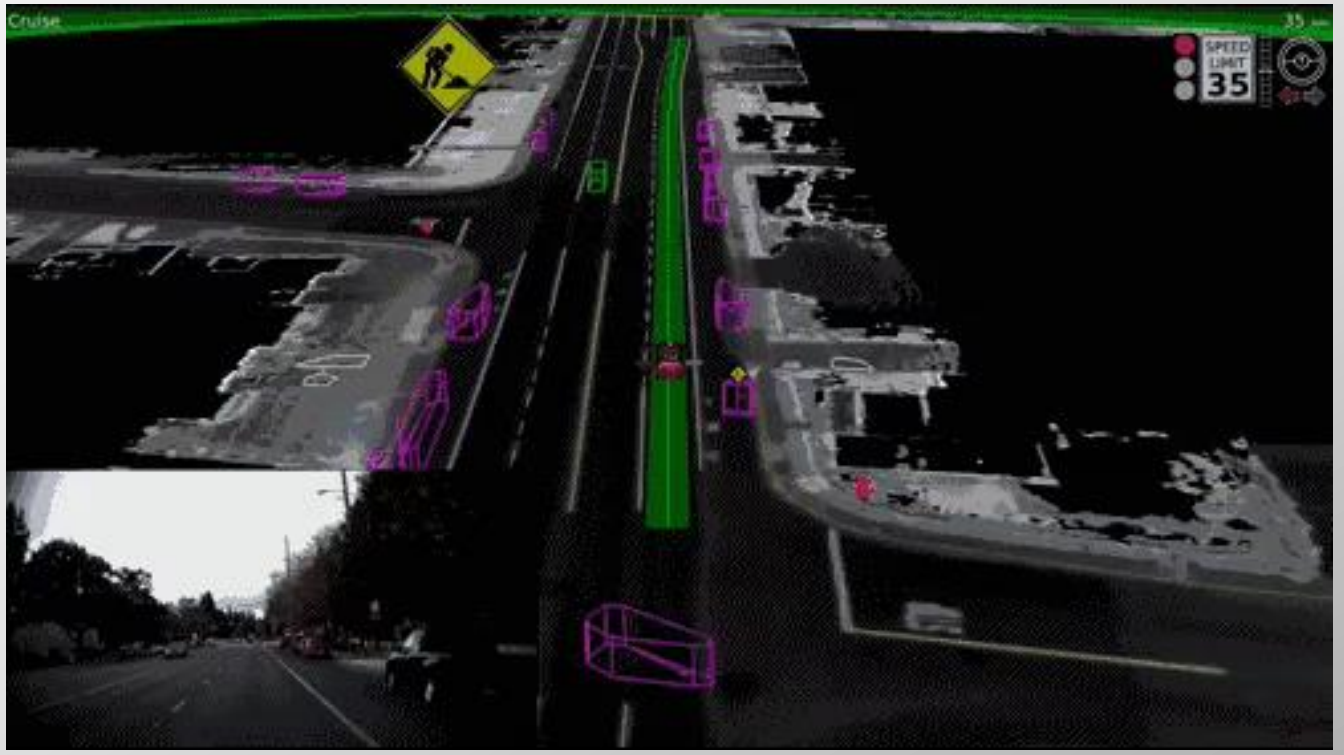
Are you sure you want to be a data scientist?

Mask Use Cases in Autonomous System

- Self driving cars
- UAVs / Drones
- Robotic arms



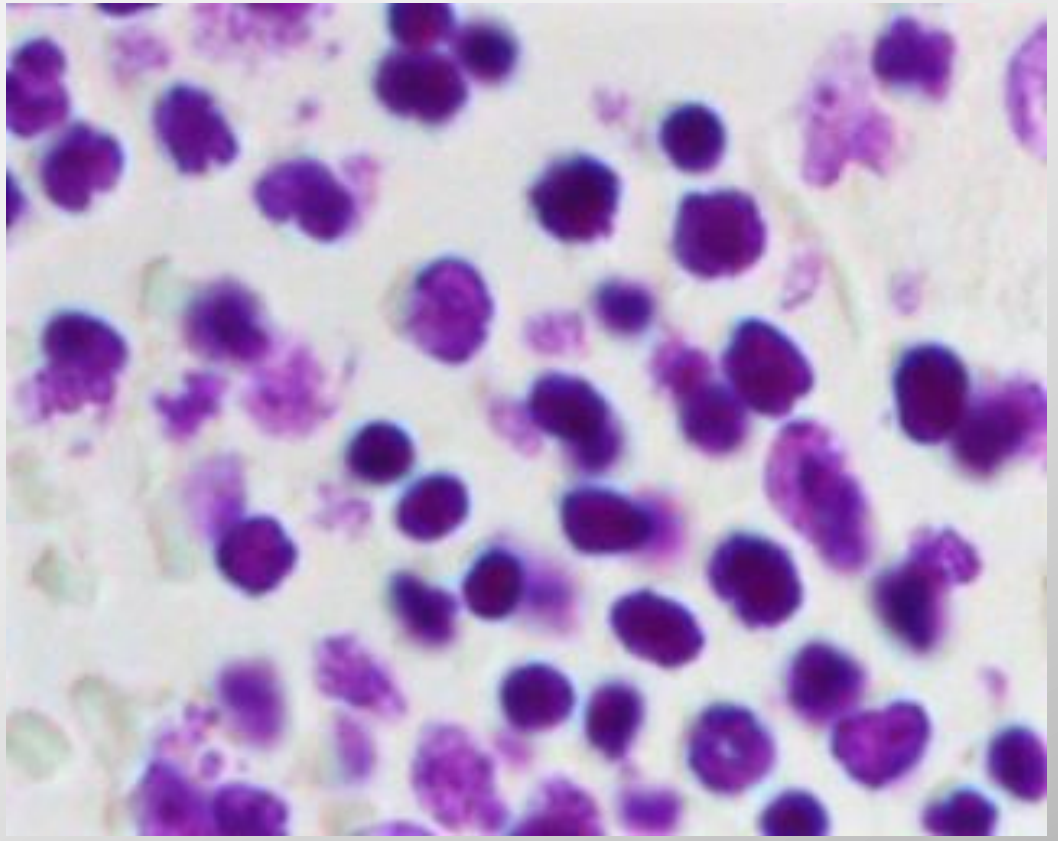
https://github.com/matterport/Mask_RCNN/



https://gifer.com/en/Ubqj

Other Use Cases – Medical Research & Diagnostics

- Kaggle 2018 Data Science Bowl
  - Identifying cell nuclei
  - $50,000 First Prize
  - Many teams used Mask RCNN



Other Use Cases - Zombie Apocalypse Detection

Source: https://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/

Outlining the Masks for Training Can Be Tedious

Source: Mighty AI

Source: Kitti Vision Benchmark

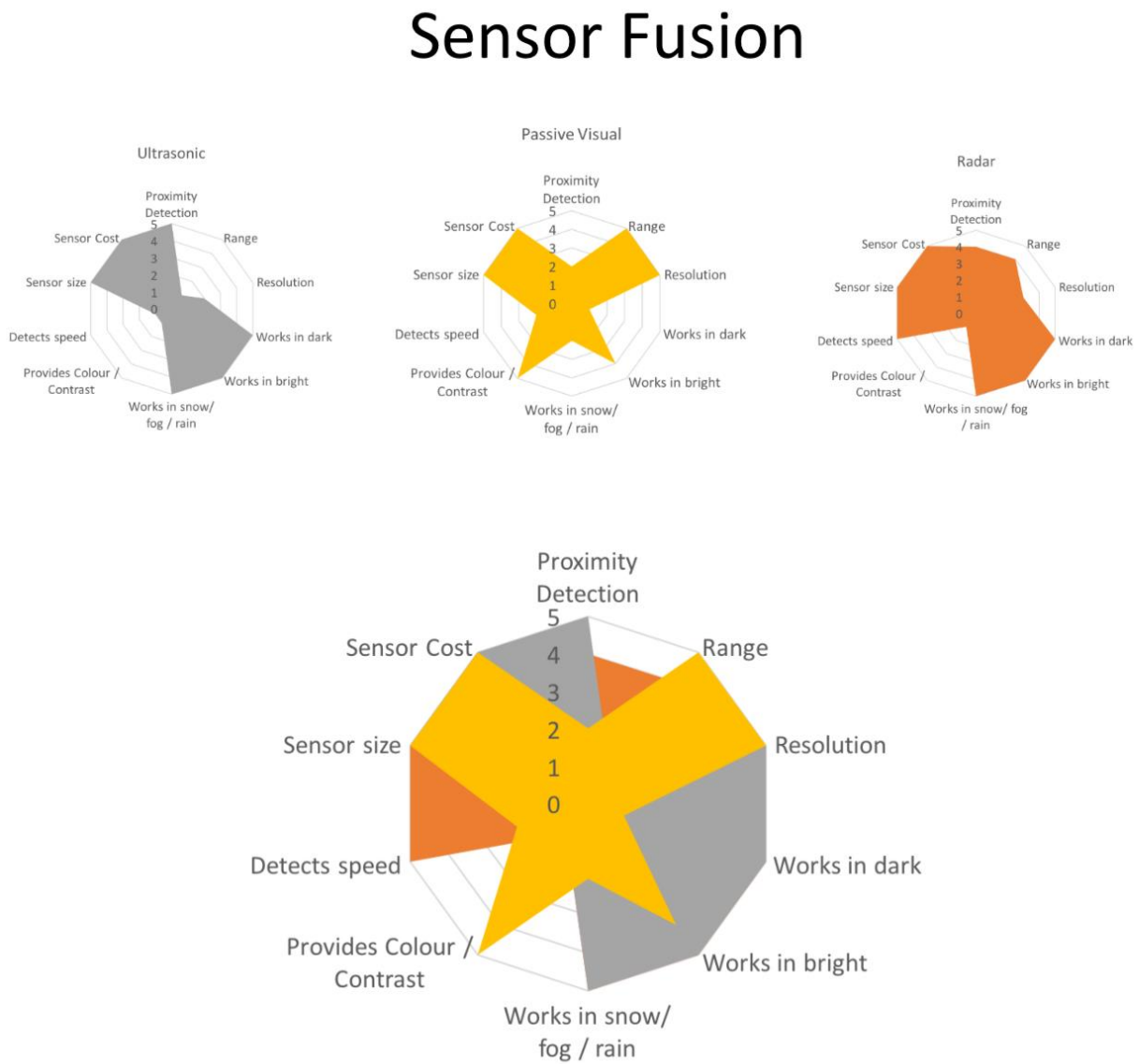
Supervised Learning requires labeled (annotated) data

Image Quality Affects Perception Predictions

What do you see in this night scene?

Source: https://www.bloomberg.com/news/articles/2018-05-29/uber-s-fatal-crash-revealed-a-self-driving-blind-spot-night-vision

Fuse Multiple Complementary Sensors



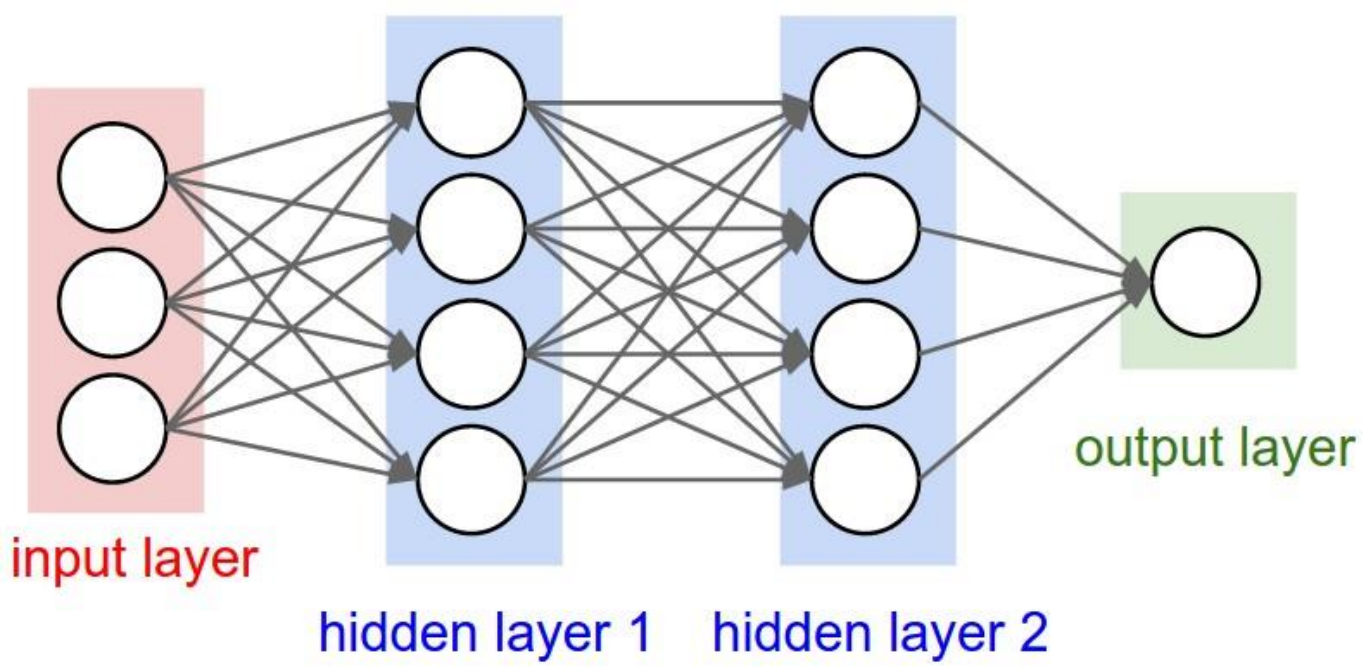
Proximity

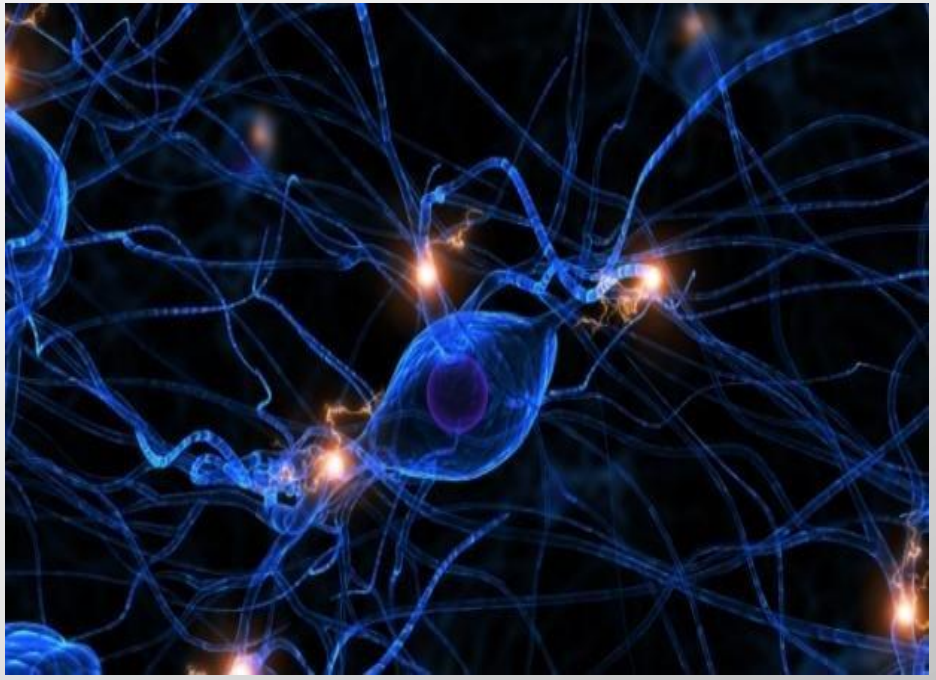
Camera

Radar

Source: Lex Fridman @MIT

Deep Learning - Feed Forward Neural Network (MLP)

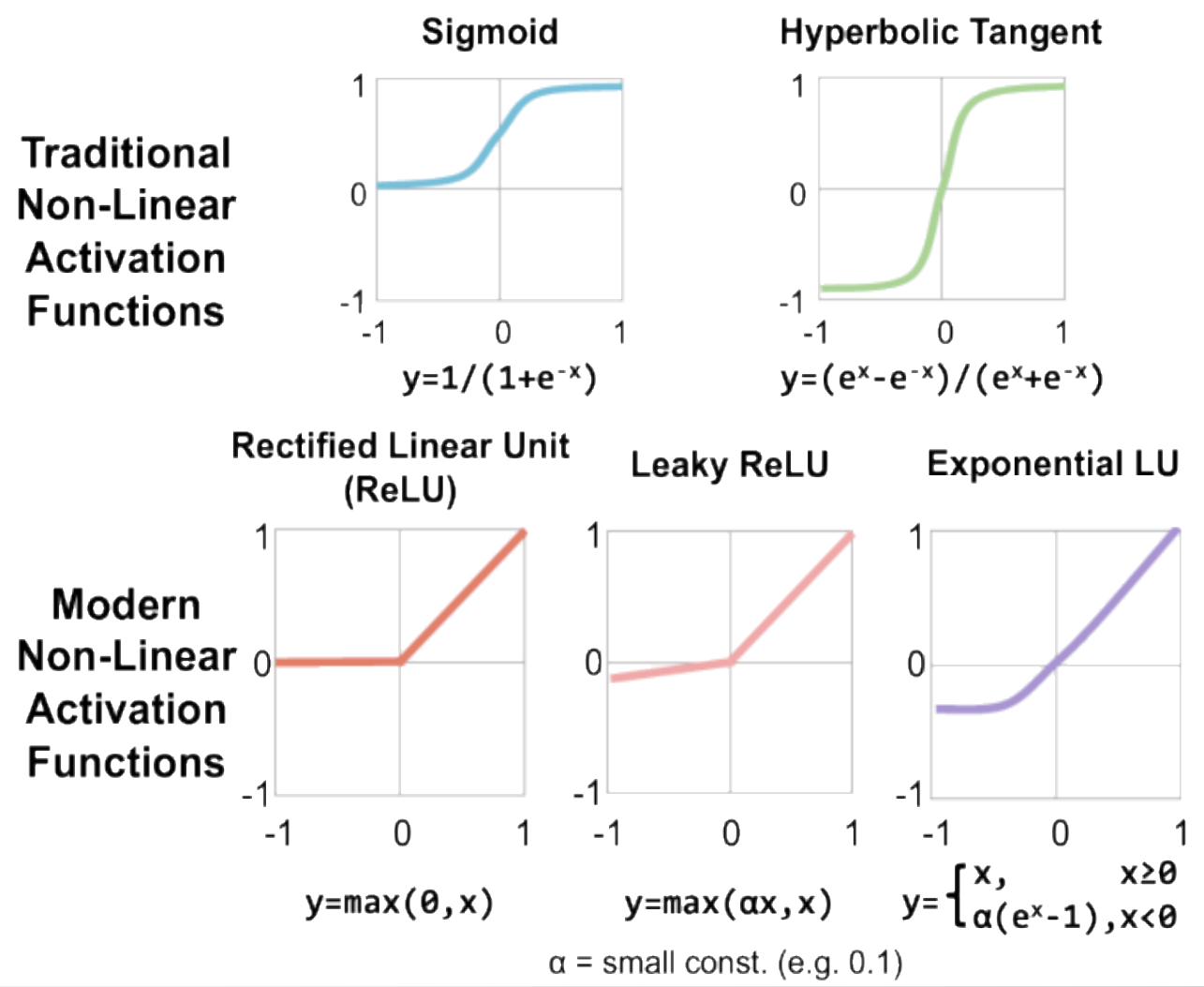




https://nutricaobrasil.wordpress.com/esclerose-neuronios-cerebro-20110608-size-62-2/

- Layers contain nodes that perform mathematical functions on a set of inputs
- Activation functions (represented as circles in the image) bound the output range and add non-linearity
- Outputs of one layer are inputs to the next layer (feed forward)
- Deep networks have more layers
- Motivation derived from our understanding of human brain function

Activation Functions



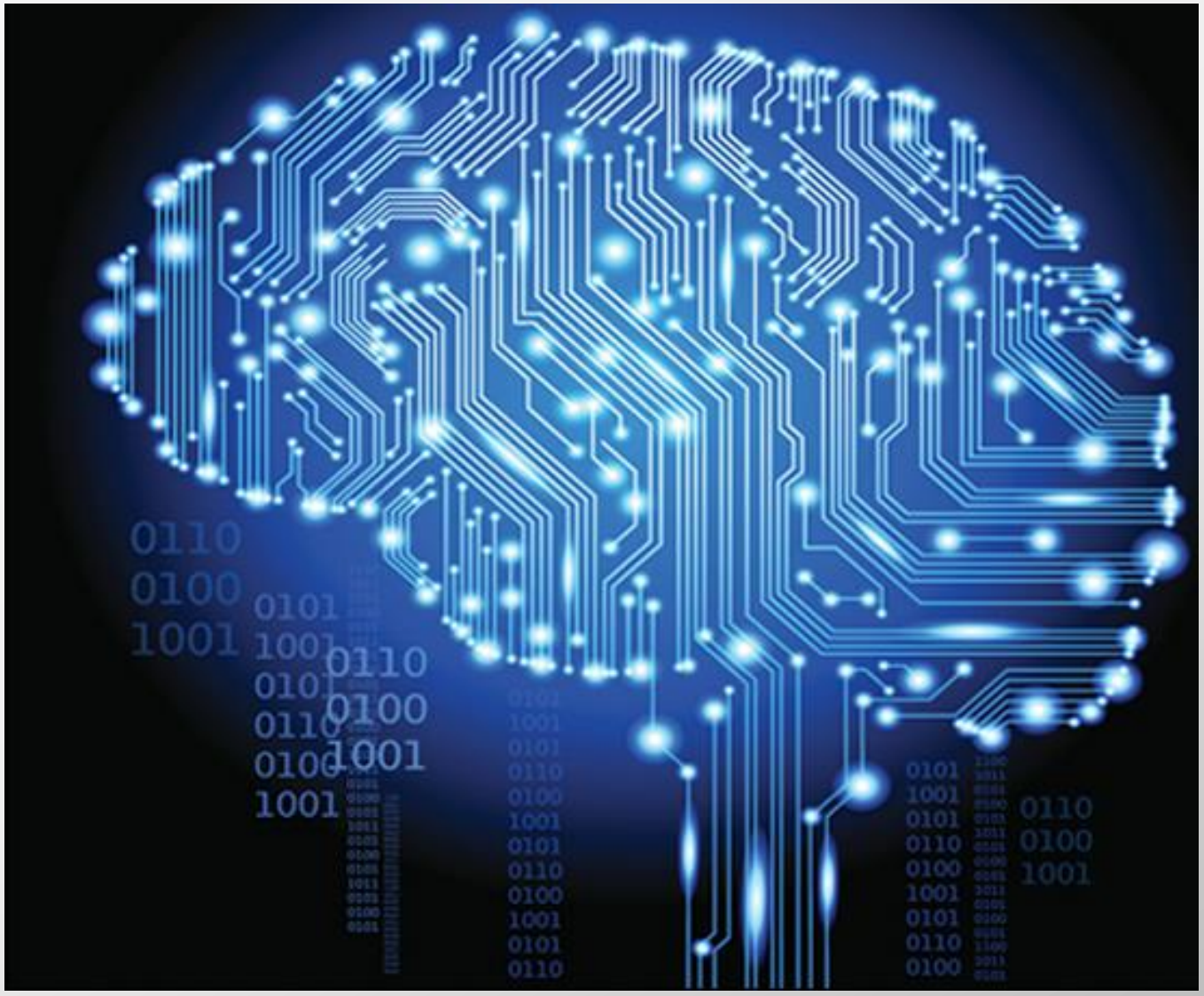
Source: http://gmelli.org/RKB/Rectified_Linear_Unit_(ReLU)_Activation_Function

- Continuous, differentiable, monotonic
- Output is range bound [0, 1] or [-1, 1]
- Normalizes contribution of nodes
- Introduces non-linearity

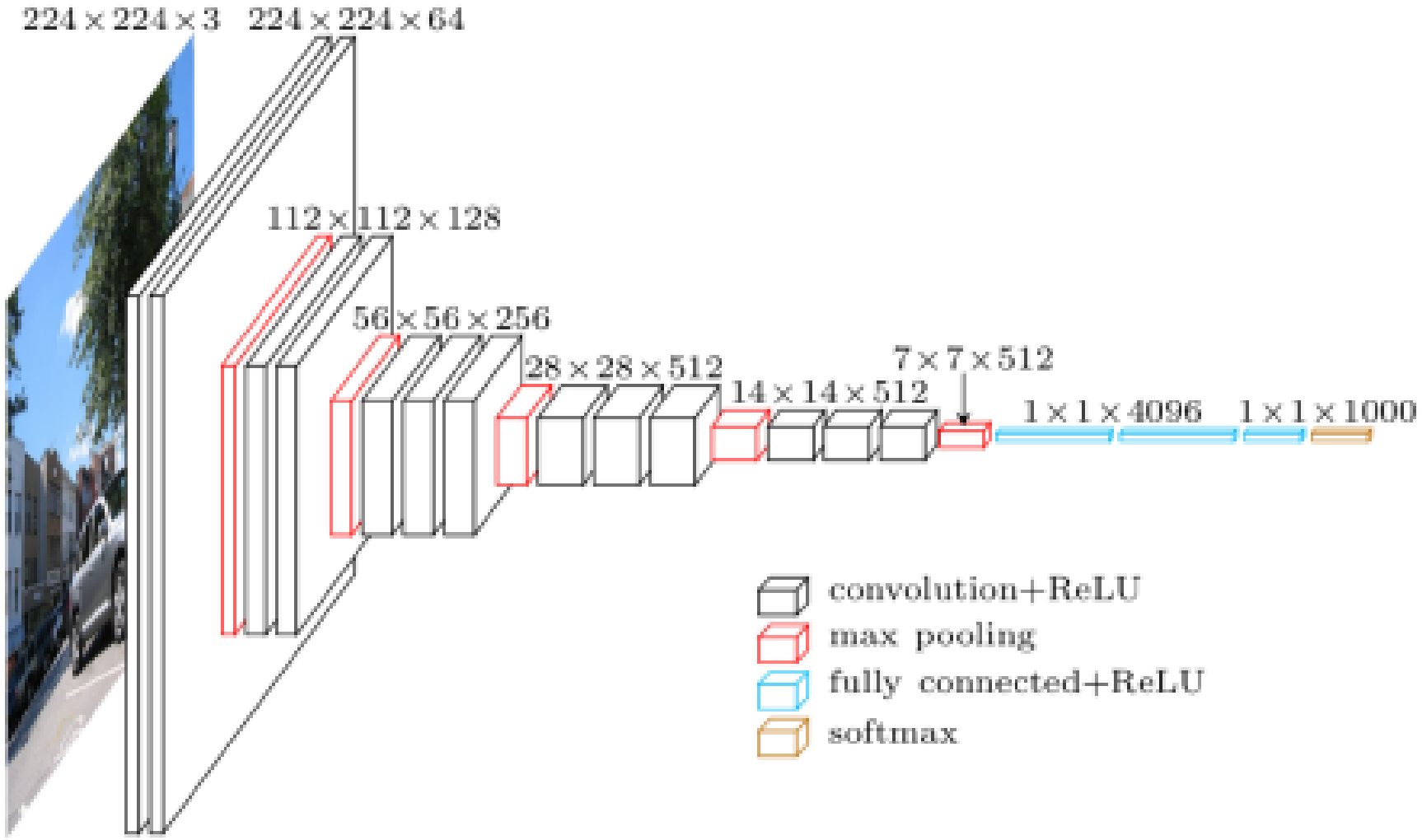
- Relu vs. Leaky Relu
  - Possibility of "Dead Neuron"
  - $\frac{dy}{dx}(0) = 0$

- Why Hyperbolic Tangent?
  - Less chance of vanishing gradient
  - $\frac{\partial}{\partial x}\tanh(x) = 1 - \tanh^2(x)$

Convolutional Neural Networks (ConvNets / CNNs)



VGG16 Network



Source: https://www.researchgate.net/figure/The-architecture-of-a-VGGNet-CNN-after-Wang-et-al-2017_fig1_323796590

Input RGB image 224x224x3

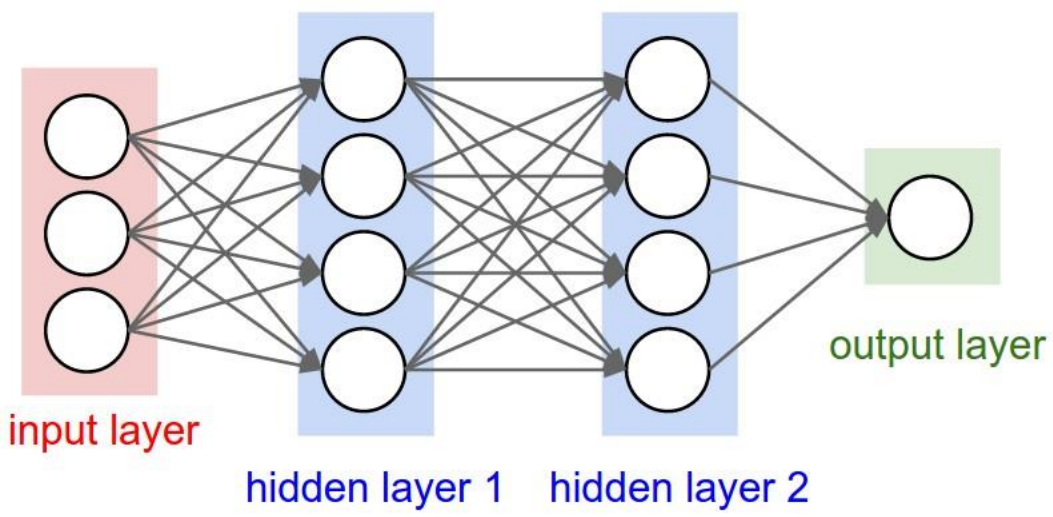
CNNs extract features from small sections at a time. As this is repeated, the height and width of layers become smaller. Deeper networks have more layers.

As more kernels are applied, the depth of each layer increases (H x W x D).

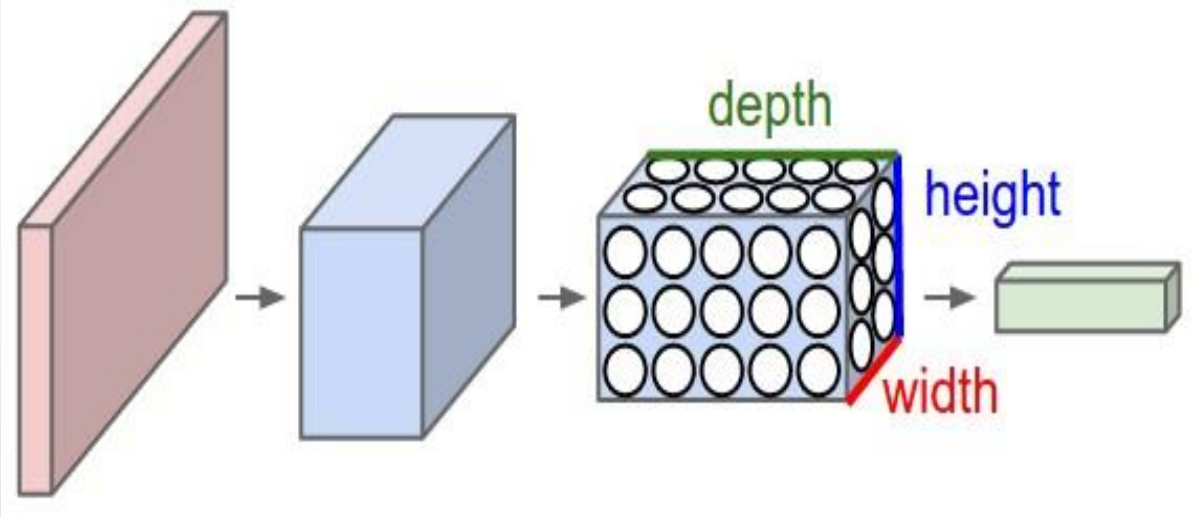
28x28x512 has 512 kernels

Eventually the network is and the 1,000 classes are ranked in order of likelihood.

Fully Connected vs. Convolution Layers



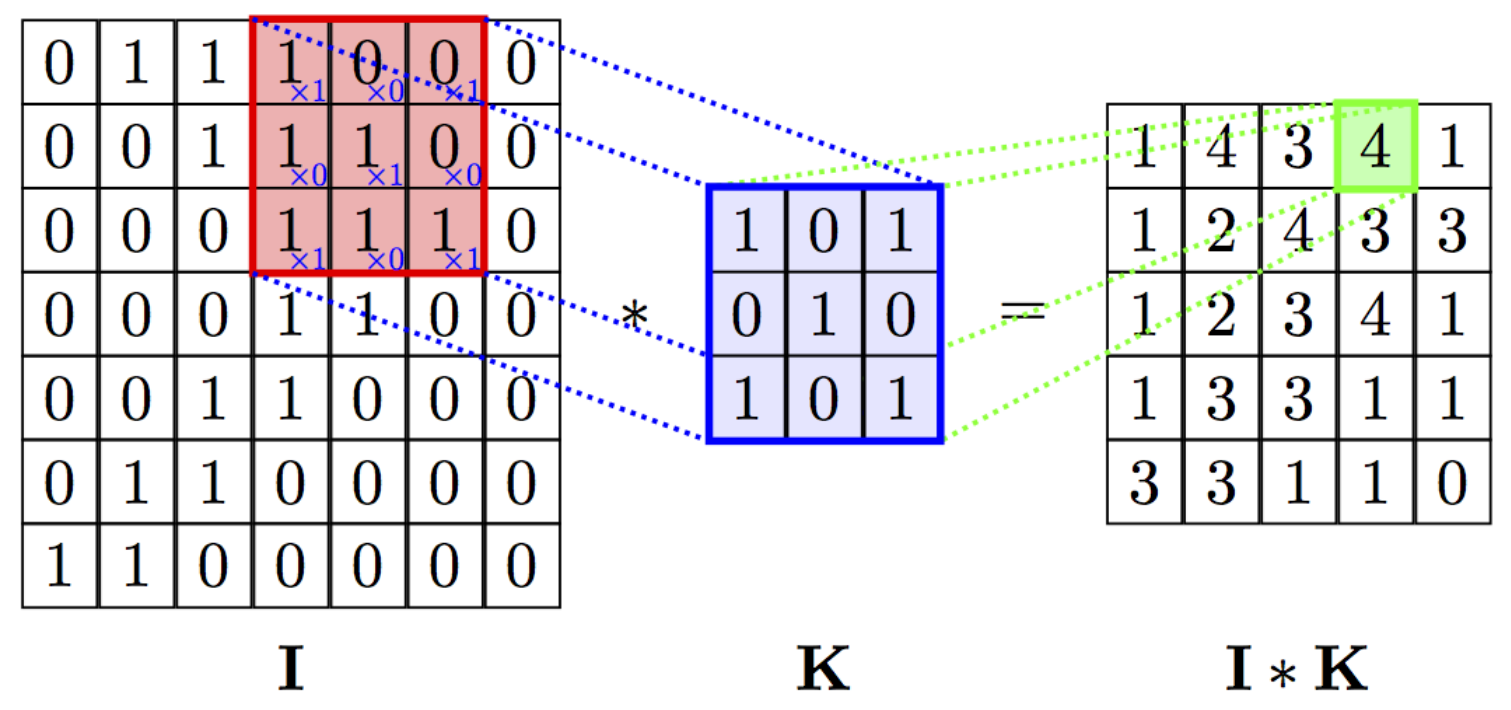
- Fully connected / many relationships
- Spatial relationships not well preserved
- Inputs are typically vectors



- Sparsely connected / fewer relationships
- Preserves spatial relationships → good for images
- Inputs are typically 3D

Image Source: http://cs231n.github.io/convolutional-networks/

Convolution – Weighted Sum



10 parameters
- KxK weights
- 1 bias term

- Dot product of kernel K(typically a square of shape K x K x D) with a part of an input (e.g. an image)
- Kernel values (weights) typically floating point numbers, not integers (unless quantized / binarized)
- Move the kernel across the entire input and perform dot products (fast if done in parallel on GPU)
- Convolution layer output is $\mathcal{F}(\mathrm{I}*\mathrm{K} + \text{bias})$ - activation function introduces non-linearity
- What is D in this example?

Image Source: http://cs231n.github.io/convolutional-networks/

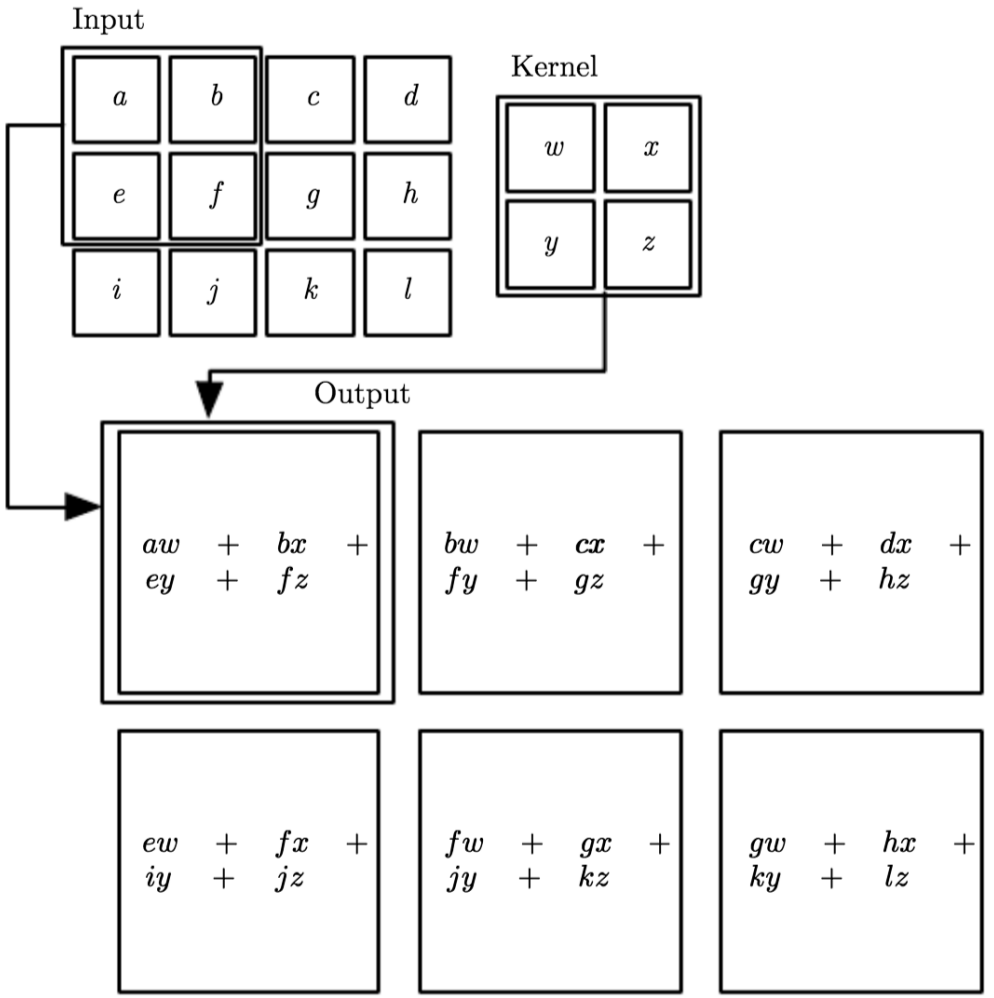


Figure 9.1: An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

Source:
Deep Learning
Book, Goodfellow

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{9.4}$$

Convolution is commutative, meaning we can equivalently write

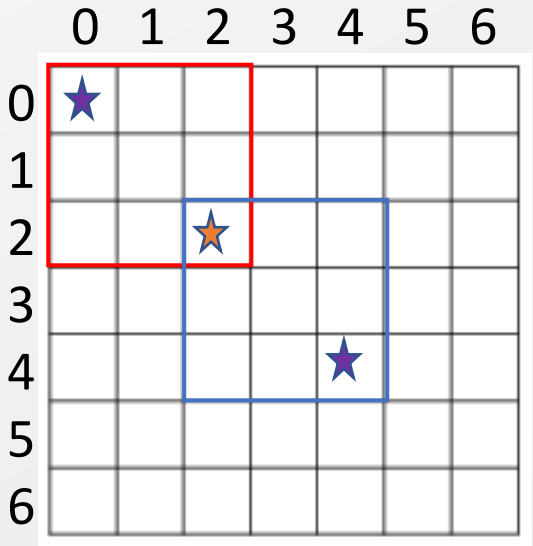
$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \tag{9.5}$$

Usually the latter formula is more straightforward to implement in a machine learning library, because there is less variation in the range of valid values of m and n.

The commutative property of convolution arises because we have **flipped** the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases. The only reason to flip the kernel is to obtain the commutative property. While the commutative property

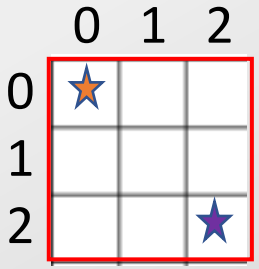
is useful for writing proofs, it is not usually an important property of a neural network implementation. Instead, many neural network libraries implement a related function called the **cross-correlation**, which is the same as convolution but without flipping the kernel:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{9.6}$$



Input I is m x n

Kernel K is 3x3
i=2, j=2 (0 based)

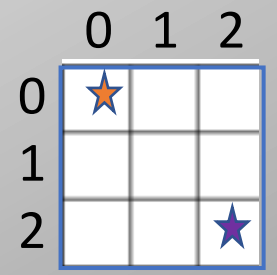


Eq. 9.4 - Flipped / Rotated 180° (Convolution)

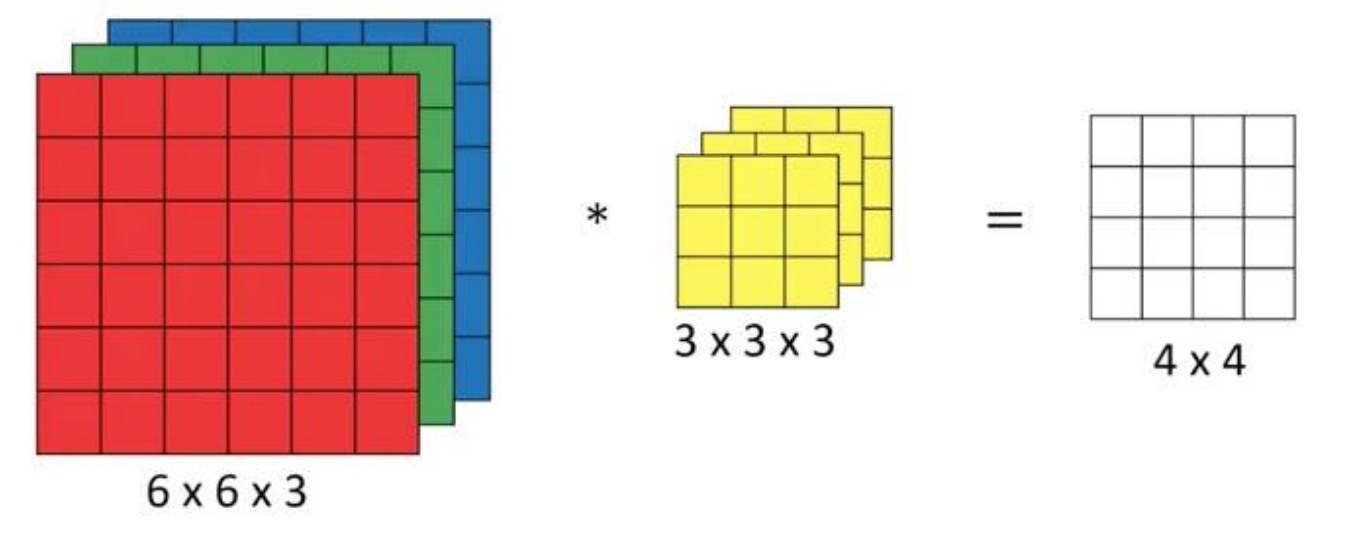
I(m,n)	K(i-m , j-n)
0, 0	2-0, 2-0
1, 1	2-1, 2-1
2, 2	2-2, 2-2
0, 2	2-0, 2-2

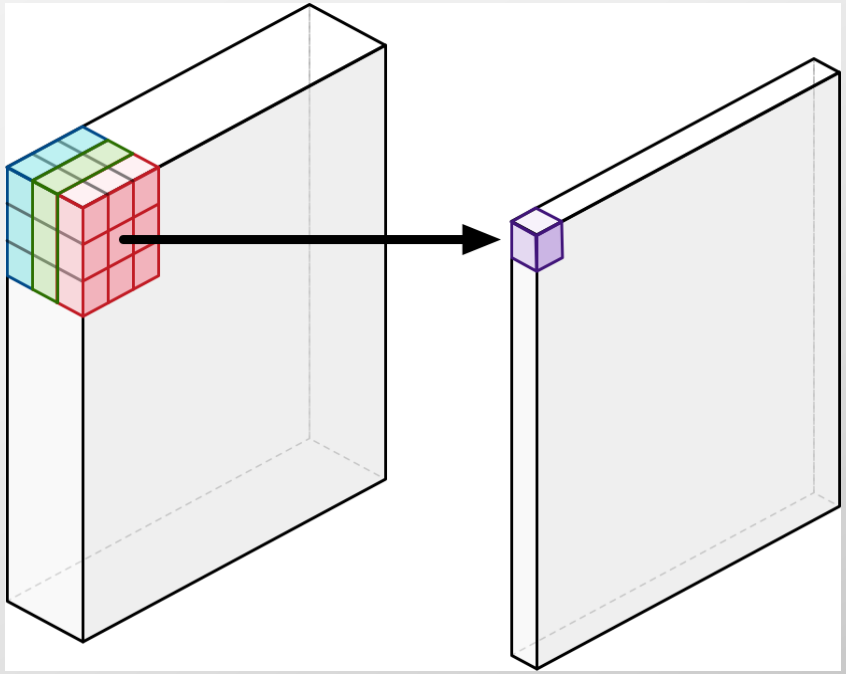
Eq. 9.6 – No Flip / Rotation (Cross-Correlation)

I(i+m, j+n)	K(m, n)
2+0, 2+0.	0, 0
2+1, 2+1.	1, 1
2+2, 2+2.	2, 2
2+0, 2+2.	0, 2



Convolution with Multiple Input Channels



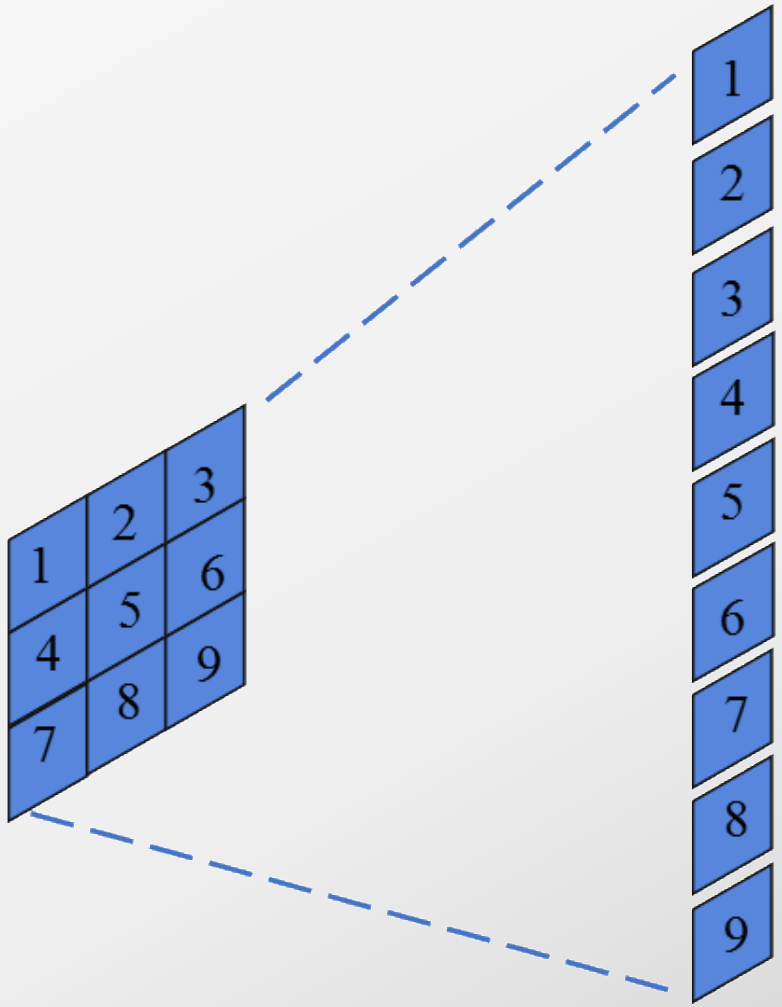


- Channels determine Depth. RGB images have 3 channels.
- Kernels will have the same depth as the number of input channels.
- What is the depth of the output?

How many parameters?
KxKxD +1 bias
(weights + bias)

Image source: https://stackoverflow.com/questions/47616700/why-in-convolutional-neural-network-3-channels-when-con-vol-into-3-channels-filt
http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

Flattening Multi-Dimensional Data to 1D



Example:

keras.layers.Flatten(data_format=**None**)

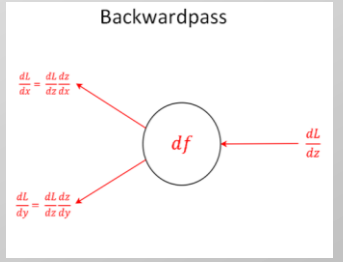
Assume model.output_shape is (None, 3, 32, 32)

model.add(Flatten())

 # now: model.output_shape == (None, 3072)

Why is this important?

→ *Everything you know about backpropagation and training applies to convolutions*



→ *Ian Goodfellow explains it from a 1D perspective*

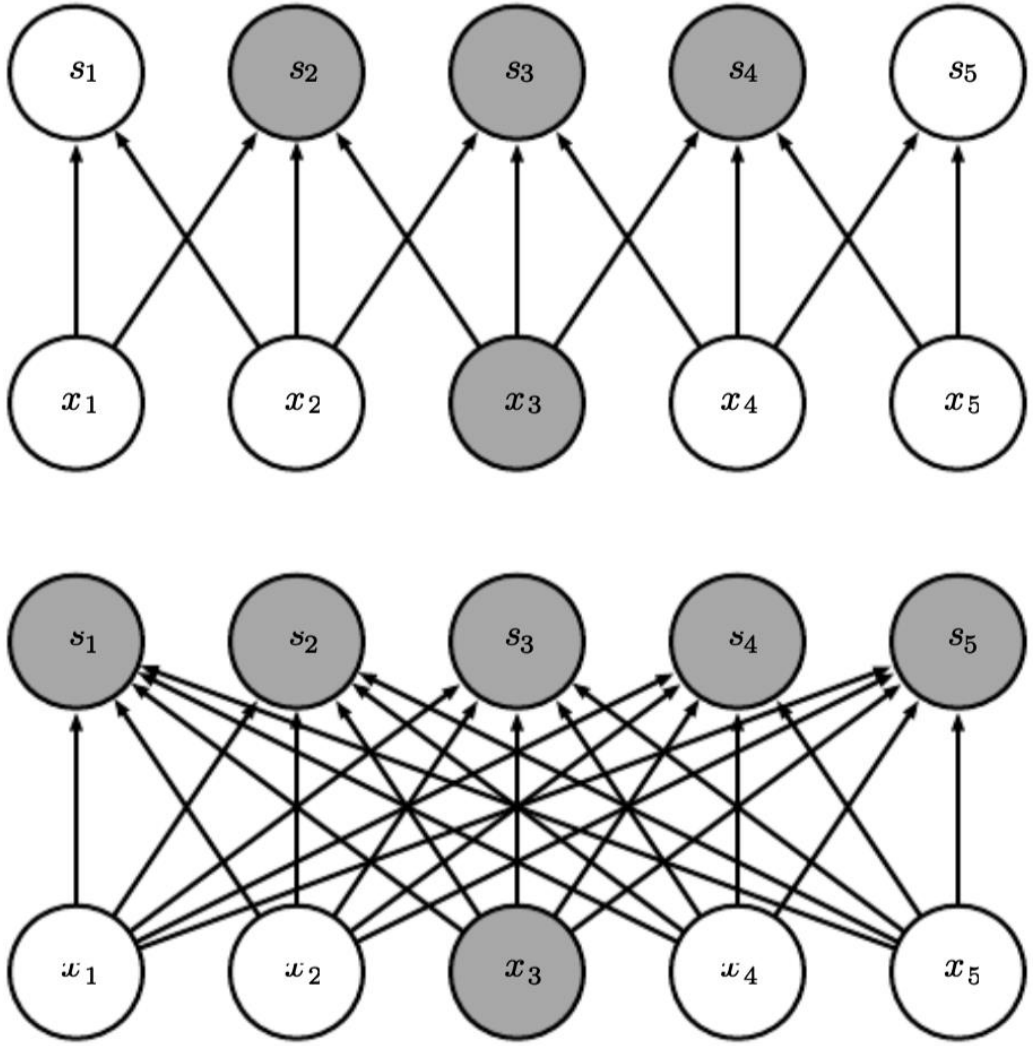


Figure 9.2: Sparse connectivity, viewed from below. We highlight one input unit, x_3, and highlight the output units in $\boldsymbol{s}$ that are affected by this unit. *(Top)*When $\boldsymbol{s}$ is formed by convolution with a kernel of width 3, only three outputs are affected by $\boldsymbol{x}$. *(Bottom)*When $\boldsymbol{s}$ is formed by matrix multiplication, connectivity is no longer sparse, so all the outputs are affected by x_3.

Convolutions have sparse connectivity. Not all inputs affect all outputs.

Fully connected layer has all inputs affecting all outputs.

Which has more computations and parameters?

Parameter Sharing

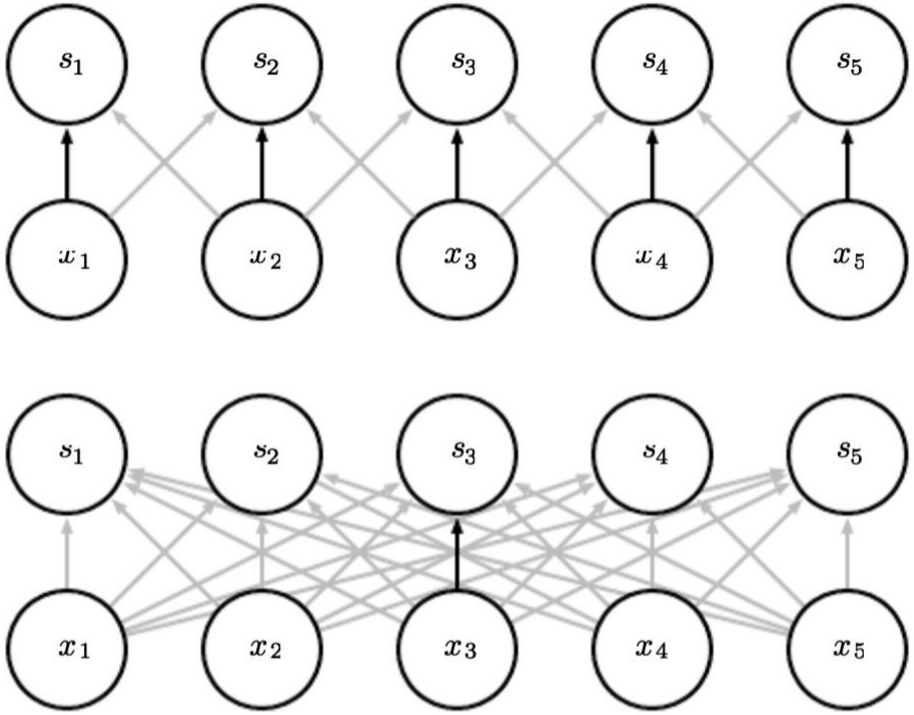
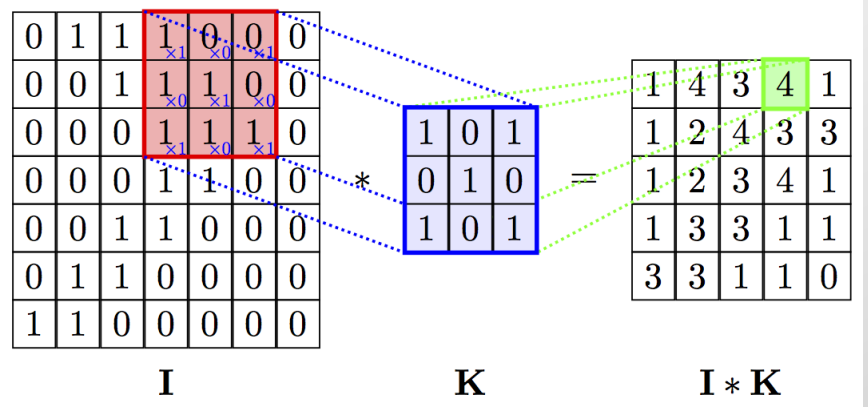


Figure 9.5: Parameter sharing. Black arrows indicate the connections that use a particular parameter in two different models. *(Top)*The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Because of parameter sharing, this single parameter is used at all input locations. *(Bottom)*The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing, so the parameter is used only once.



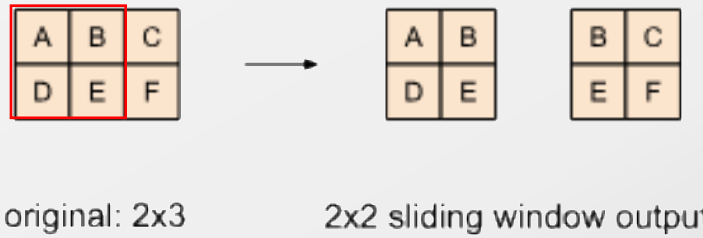
Since the same kernel is applied to many inputs, the parameters (weights) are the same at different inputs (black arrows = central element of the kernel)

An input can be used in multiple convolutions, but it gets paired with a different weight (grey arrows).

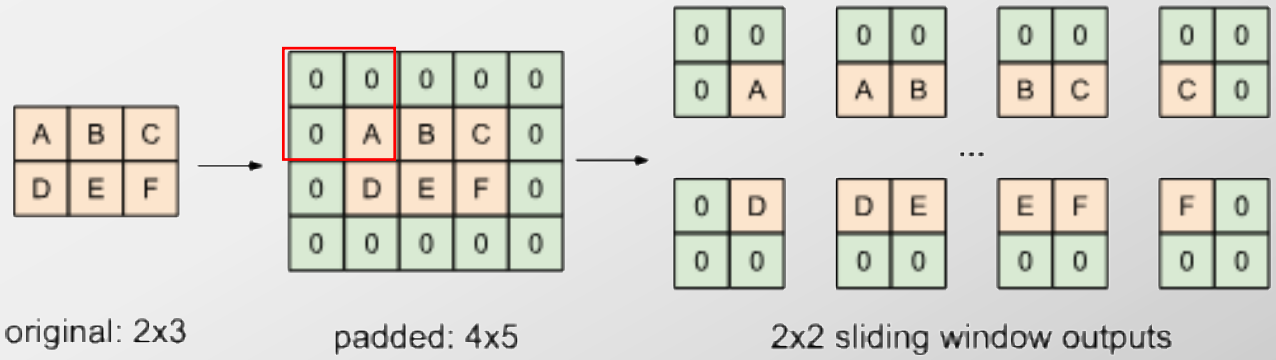
In fully connected layers, all weights can be unique (bottom of Fig 9.5).

CNN Concepts

- Stride – how far the kernel moves before applying it again
  - Larger stride reduces the output size more
- Padding is often added to the input edges



Kernel = 2x2
Stride = 1,1
No Padding

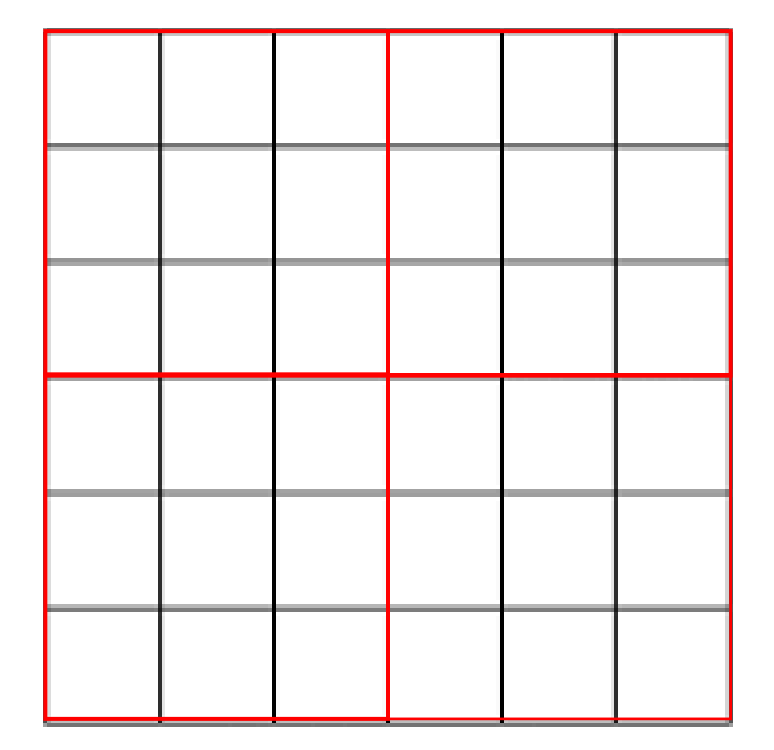


What Stride length is used here?
Stride = 1,2

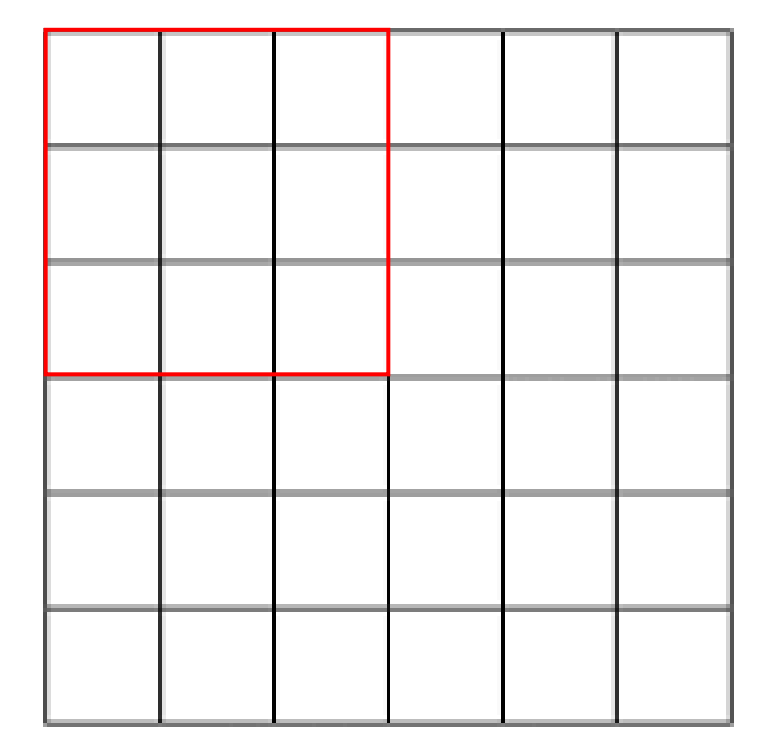
How did the padding affect the size of the output?
1x2 → 2x4

Image Source: https://www.groundai.com/project/scaling-binarized-neural-networks-on-reconfigurable-logic/

Stride



Stride = 3
Output size = 2x2



Stride = 1
Output size = ?
(m-k+1)

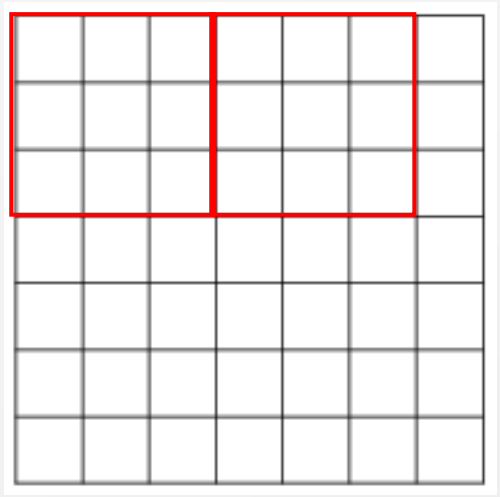
Stride

- How far to move the kernel when convolving
- Sample every s pixels (s=stride)
- Increasing stride reduces the number of computations (relationships)
- Increasing stride decreases output size spatially (Downsampling)
- Can have different strides in each direction, though not common
- How does stride affect the number of computations?

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} \left[V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \right] . \qquad (9.8)$$

Padding

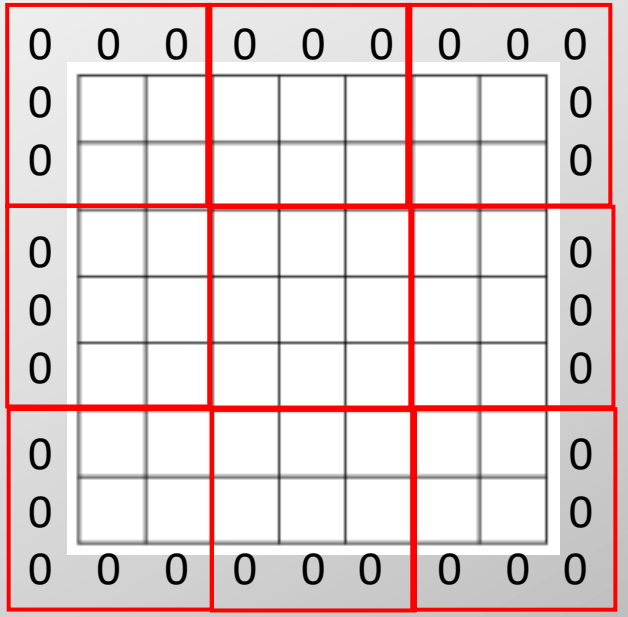
- Zero padding – add zeros around the sides of the image
- Allows us to fit virtually any kernel shape and stride to any input shape



To convolve a 7x7 input with a 3x3 kernel using stride=3, we need 2 more pixels for the kernel to map to the input evenly. Add one pixel in each direction.

Input becomes 9x9.

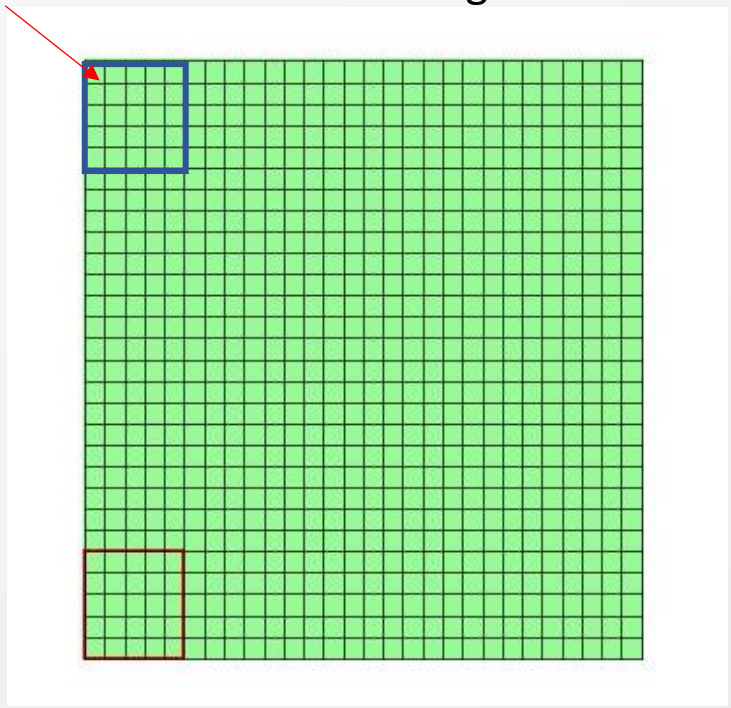
➔ Padding=1 (add one pixel to all sides)



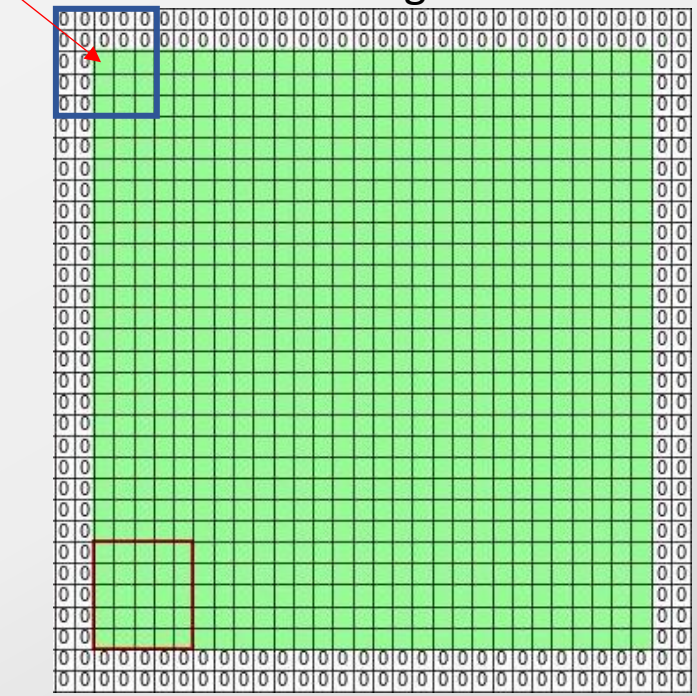
- Padding helps control size of the output feature map. How?

Padding

28x28
No Padding



28x28
Padding=2



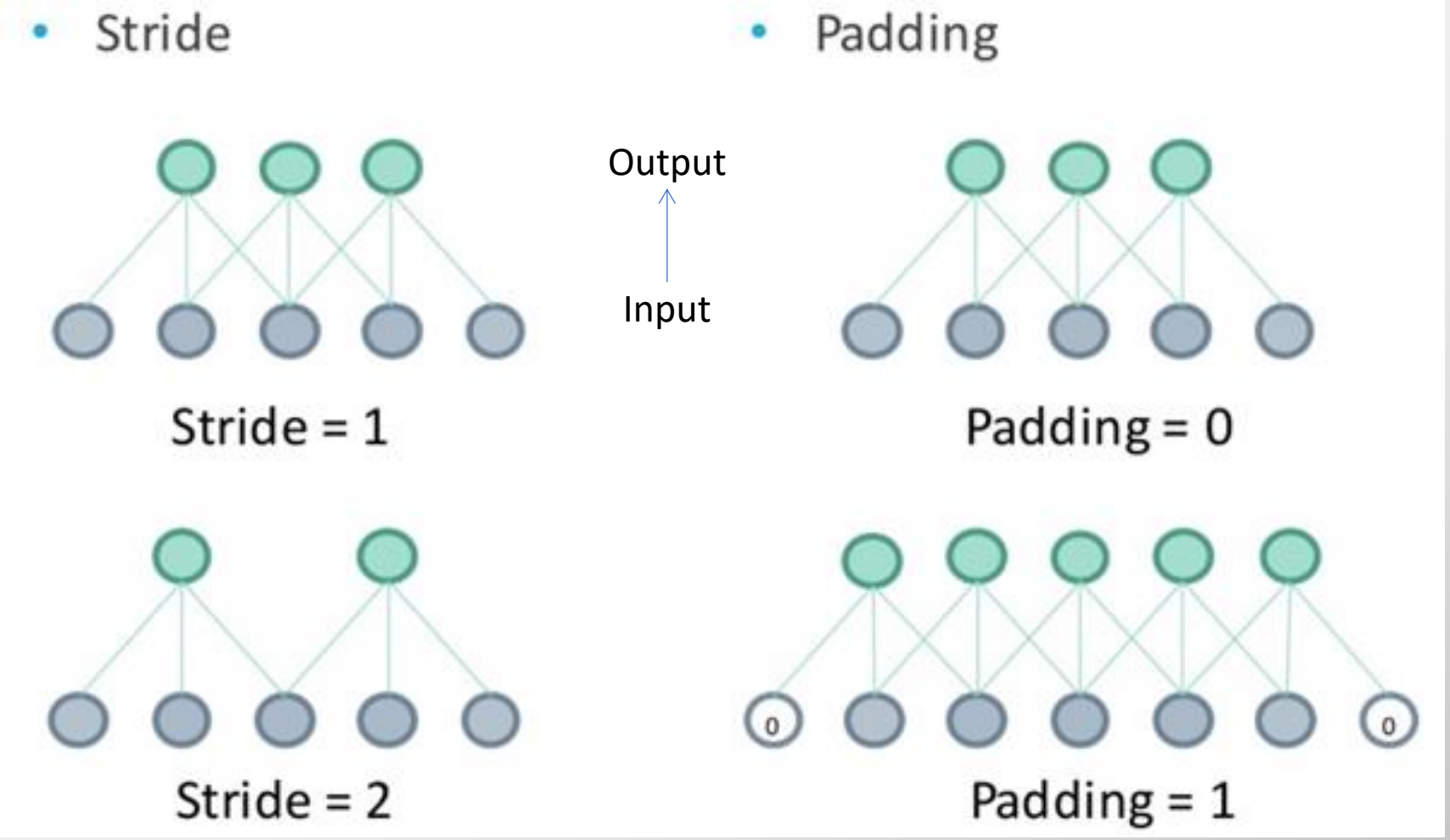
What are 2 reasons why you might want to use Padding=2 and Stride=1 with a 5x5 kernel?

Hint #1: What is the size of the output feature map from each convolution?
Hint #2: Compare how many convolutions sample from the pixel in the upper left corner when P=0 vs. P=2.

Padding

- Types
  - Valid – no padding is used, size of output feature space is reduced
    (m-k+1 when stride=1)
  - Same – add the number of zeros required to keep the output size the same as the input (no downsampling)
- What stride length never needs padding to fit the kernel evenly into the input?
- Why would we pad if the kernel fits evenly?

Strided Convolution and Padding Flattened to 1D



Increasing Stride decreases relationships & computations & output feature maps size

Increasing Padding increases the number of relationships and computations & output feature map size

What is the kernel size here?

Source: https://medium.com/@raycad.seedotech/convolutional-neural-network-cnn-8d1908c010ab

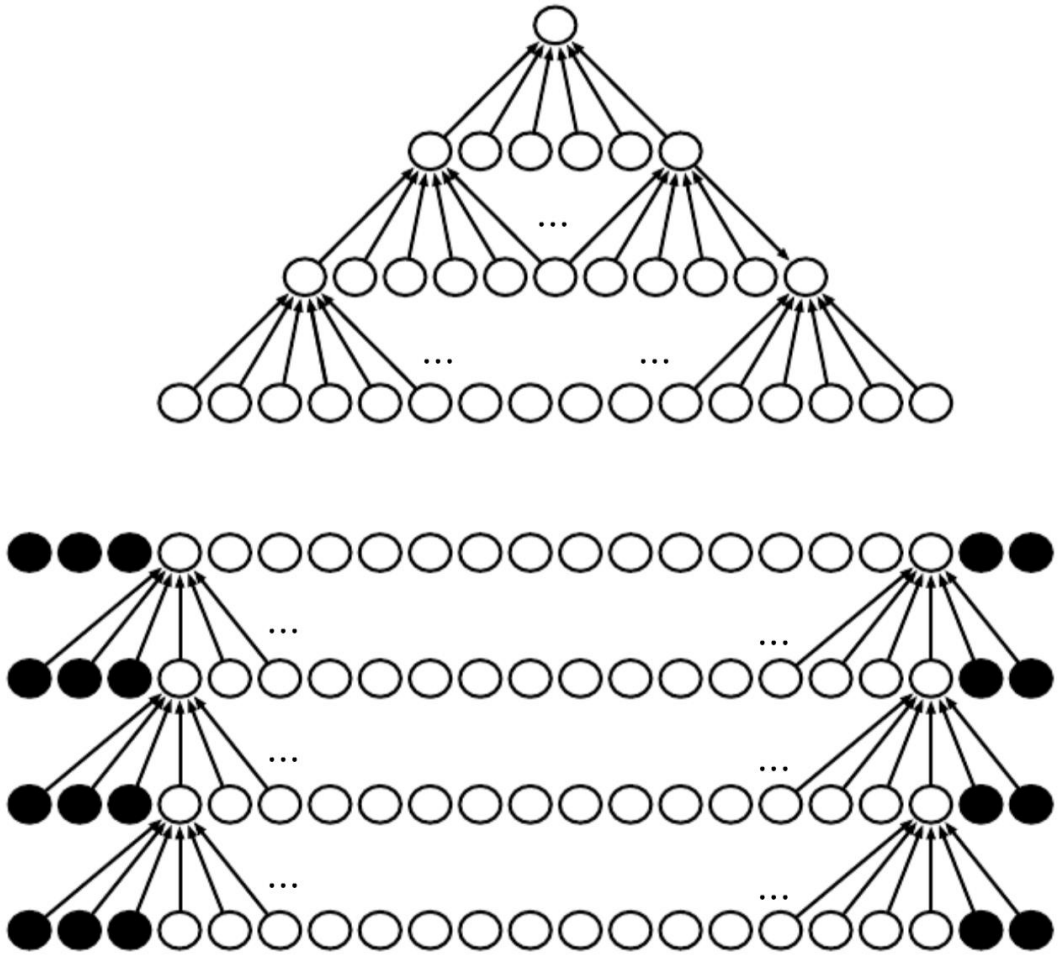


Figure 9.13: The effect of zero padding on network size. Consider a convolutional network with a kernel of width six at every layer. In this example, we do not use any pooling, so only the convolution operation itself shrinks the network size. *(Top)*In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not ever move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive, and some shrinking is inevitable in this kind of architecture. *(Bottom)*By adding five implicit zeros to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

With stride=1, valid padding shrinks output size
by (m-k+1).
Here (16-6+1)=11
And (11-6+1)=6
And last layer =1

Same padding maintains the size.

Padding does not have to be equal on all sides.

Convolutions with Multiple Channels (D>1)

1. What is the size of the output if you convolve an RGB image of (28x28x3) with a kernel of (7x7x3) using Padding=0 and Stride=7?

2. What would you do if you want to preserve the 28x28 output size using the 7x7x3 kernel?
   Hint: Set Stride=1 and choose an appropriate Padding.

3. How does increasing the stride affect the number of computations required?

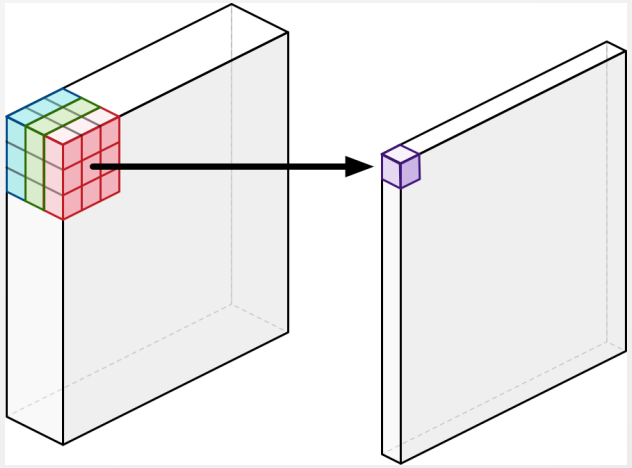
4. a) What is the size of the output if you convolve an RGB image of (28x28x3) with a kernel of (1x1x3)?
      Hint: This causes Dimensionality Reduction … which dimension?

   b) What else does a convolution layer with a 1x1x3 kernel do?
      Hint: Flatten (+ Non-Linearity)

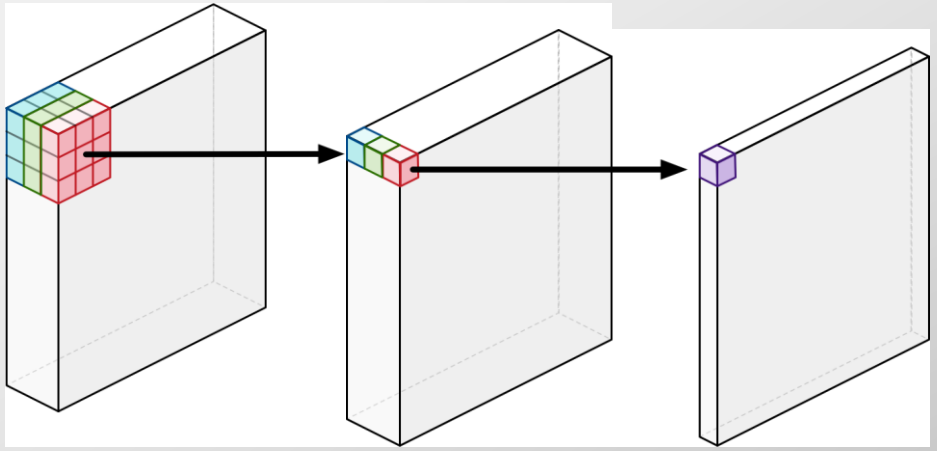
Depthwise Separable Convolution

- Performs the convolution on each channel separately then all channels are convolved with a 1x1xD kernel (pointwise convolution).
- Results in fewer calculations (used in MobileNet versions)
- Extracts channel specific features since channels are not initially combined

Regular Convolution



Depthwise Separable Convolution



Calculations if Input is 112x112x64, Output = 112x112x128 (128 kernels of size 3x3x64), Padding=Same:

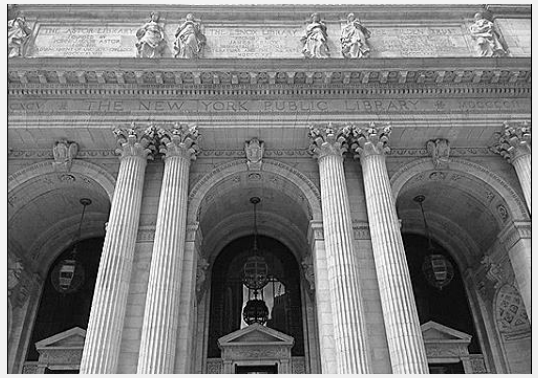
```
(3 × 3 × 64) × (112 × 112) × 128 = 924,844,032 (shape=112x112x128)

(3 × 3 × 1) × 64 × (112 × 112)    =   7,225,344 (shape=112x112x64)
+ (1 × 1 × 64) × (112 × 112) × 128 = 102,760,448 (shape=112x112x128)
```

Image source: http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

Hand Picking Kernels – Traditional Computer Vision

0	0	0
0	1	0
0	0	0

Identity

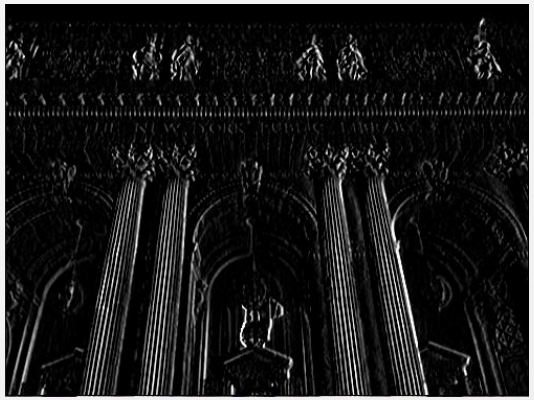


0	-1	0
-1	5	-1
0	-1	0

Sharpen

1	1	1
1	3	1
1	1	1

Custom



1	0	-1
2	0	-2
1	0	-1

Left Sobel

-1	0	1
-2	0	2
-1	0	1

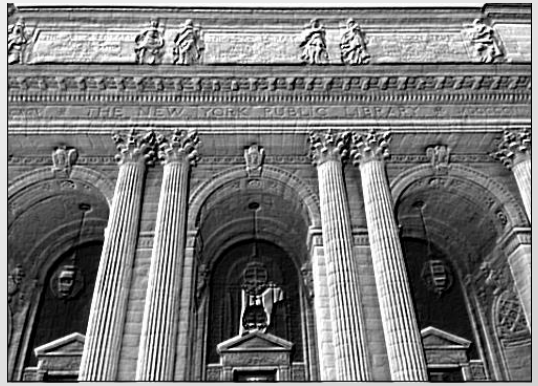
Right Sobel

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

Custom Edge

-1	-1	-1
-1	8	-1
-1	-1	-1

Outline



-2	-1	0
-1	1	1
0	1	2

Emboss

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

Custom Edge

http://setosa.io/ev/image-kernels/

Learned Kernel Weights

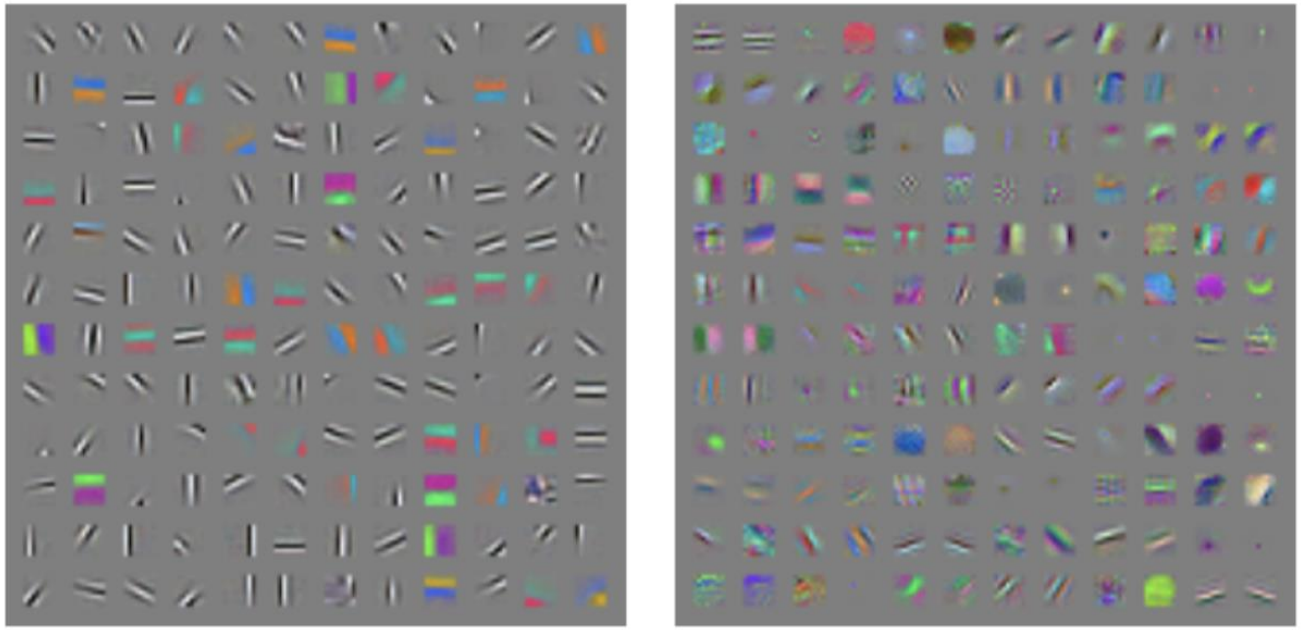


Figure 9.19: Many machine learning algorithms learn features that detect edges or specific colors of edges when applied to natural images. These feature detectors are reminiscent of the Gabor functions known to be present in the primary visual cortex. *(Left)*Weights learned by an unsupervised learning algorithm (spike and slab sparse coding) applied to small image patches. *(Right)*Convolution kernels learned by the first layer of a fully supervised convolutional maxout network. Neighboring pairs of filters drive the same maxout unit.

These kernels are larger than 3x3 and have multiple channels (color).

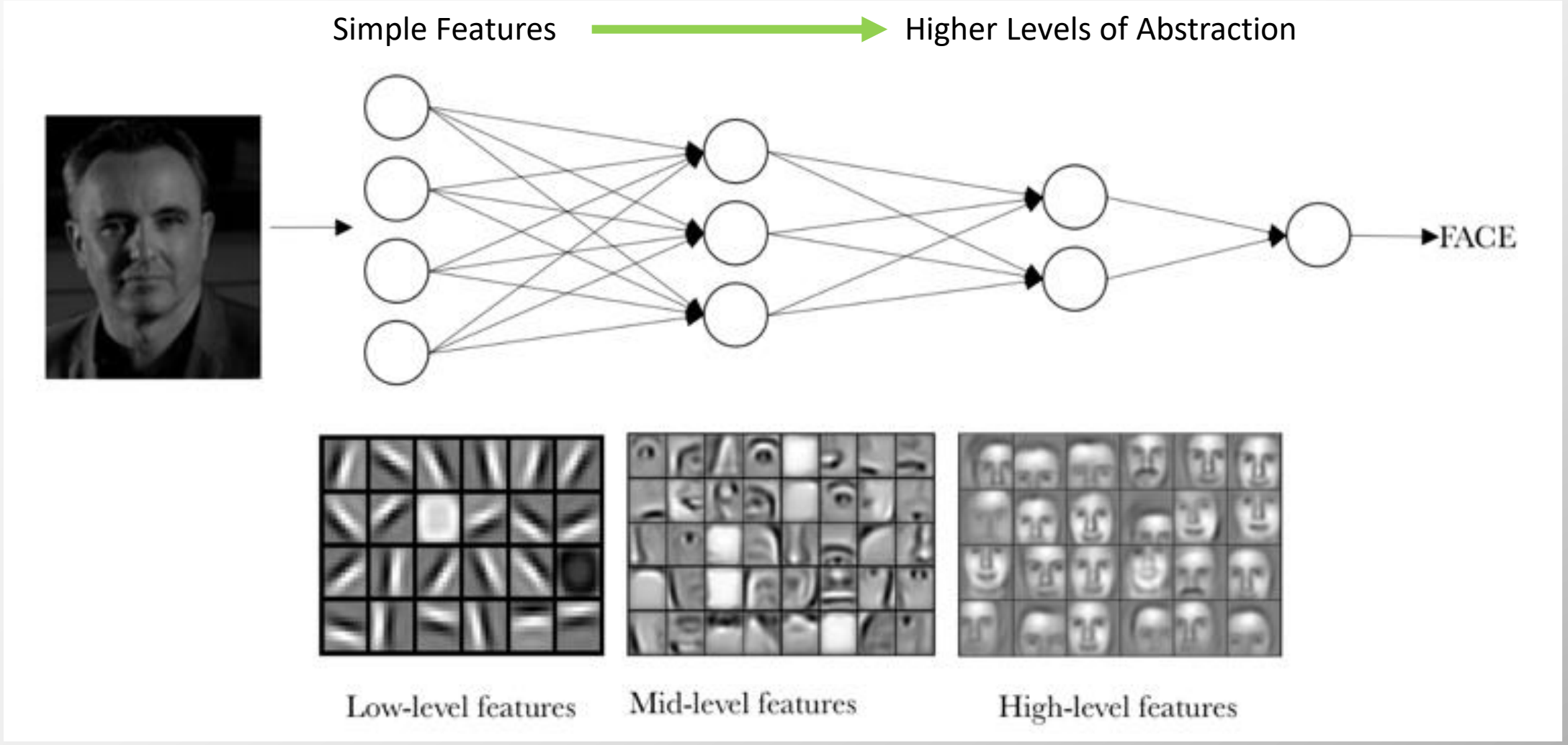
More information is encoded.

More features can be extracted.

From Ian Goodfellow's Deep Learning Book

Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Feature Extraction Across Layers



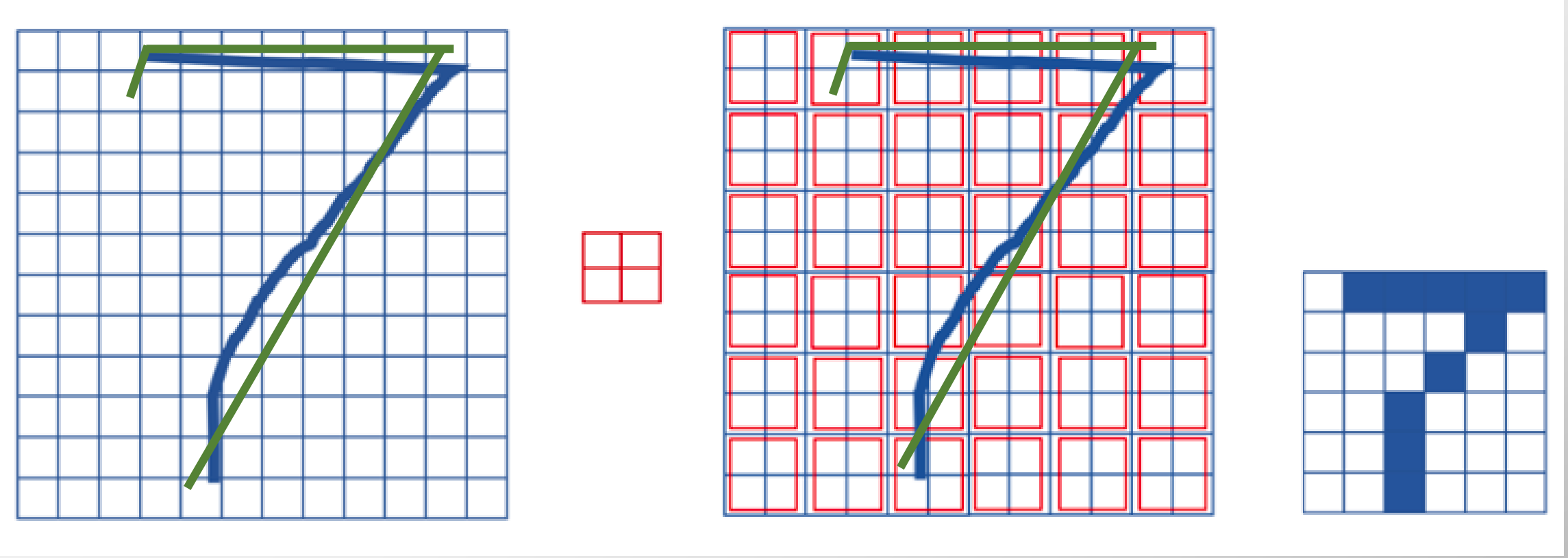
https://towardsdatascience.com/convolutional-neural-networks-for-beginners-practical-guide-with-python-and-keras-dc688ea90dca

Feature Extraction

- Pooling and convolutional layers reduce the size of the data flowing through the network
- Data from farther away gets incorporated into higher layers as layers shrink in size
- This increases the “field of view” of higher layers neurons, thus allowing them to detect higher order features in a larger region of the input image

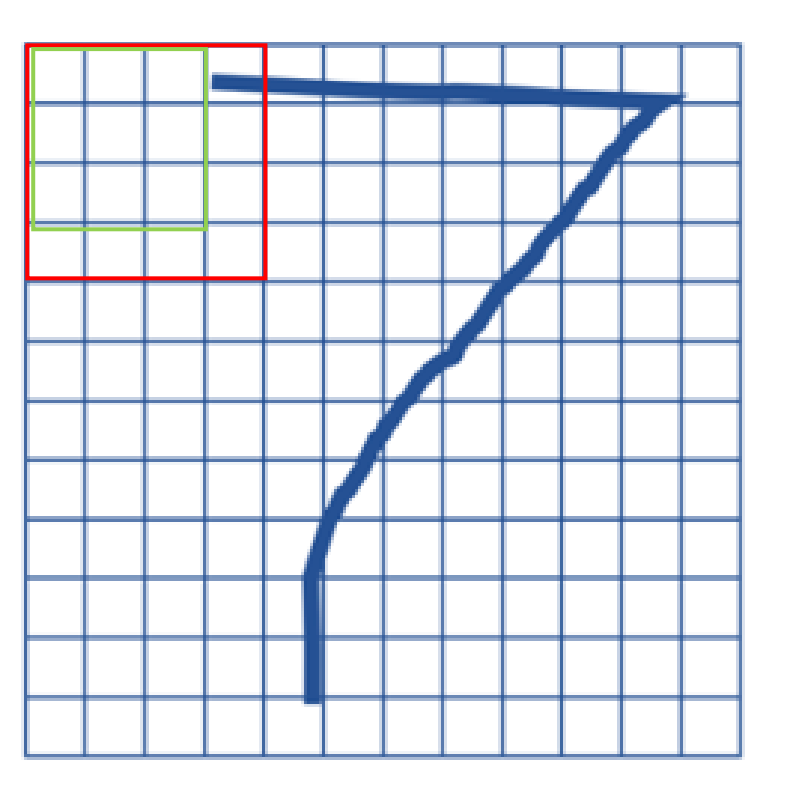
Image Source: http://www.dyxum.com/columns/photoworld/fundamentals/Field_of_view_Angle_of_view.asp

Max Pooling Example (Binary Input Values)



- Allows for different but similar inputs to be represented in the same way (scale / rotational invariance)

Pooling Example



3x3
Pool Size

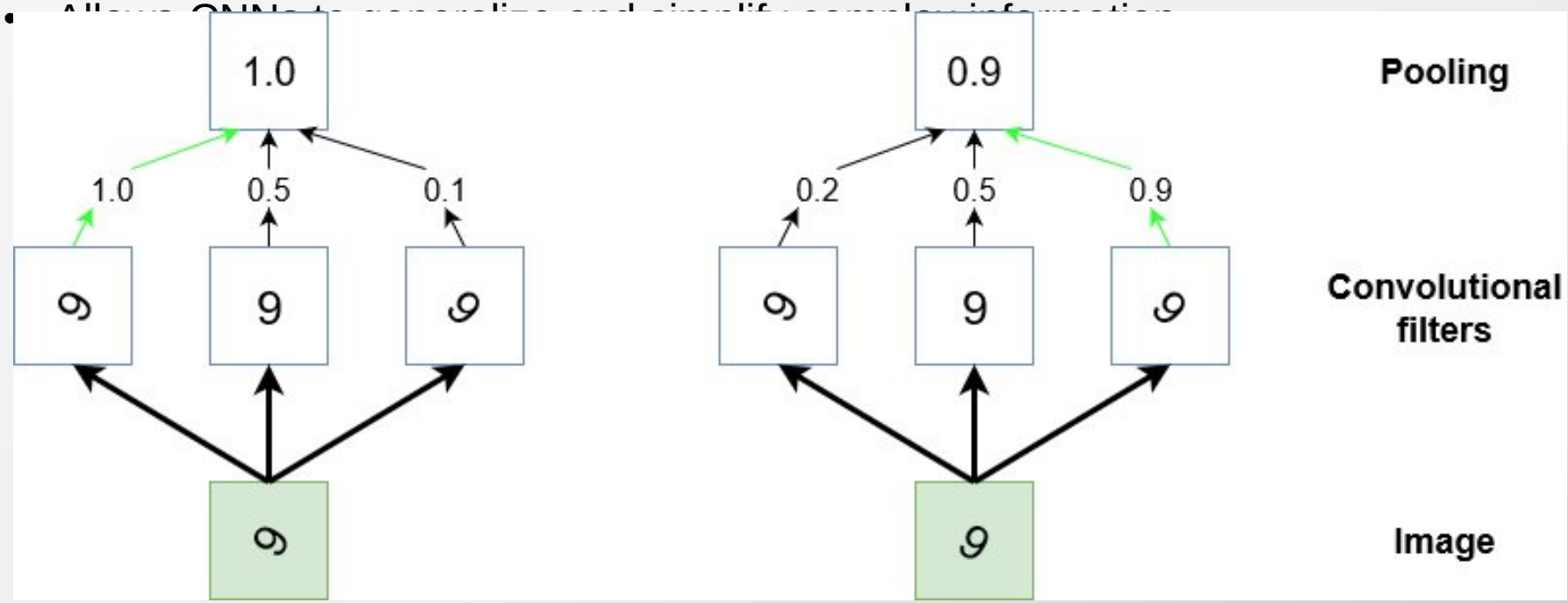
0	1	1	1
0	0	1	1
0	1	1	0
0	1	0	0

4x4
Pool Size

1	1	1
0	1	1
0	1	0

Pooling

- Pooling allows inputs somewhat invariant to be scale and orientation changes (up to a point)
- Allows CNNs to generalize and simplify complex information

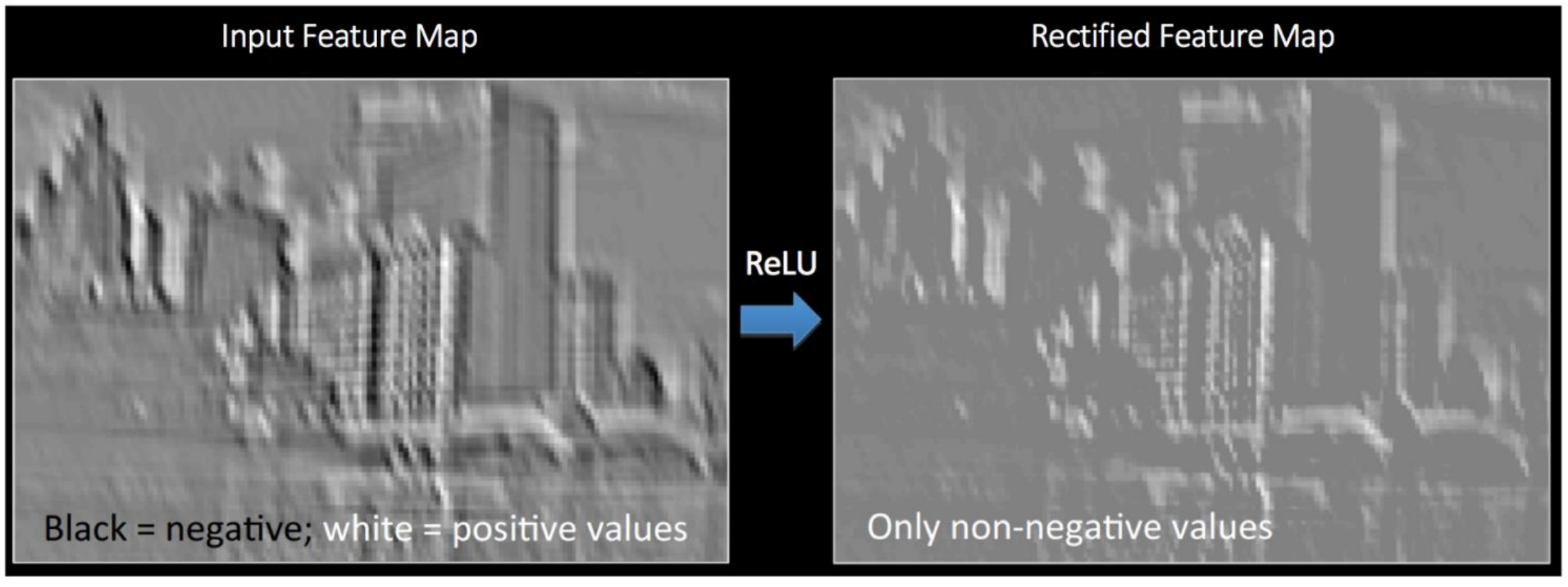


Green arrows indicate which neuron fires most strongly

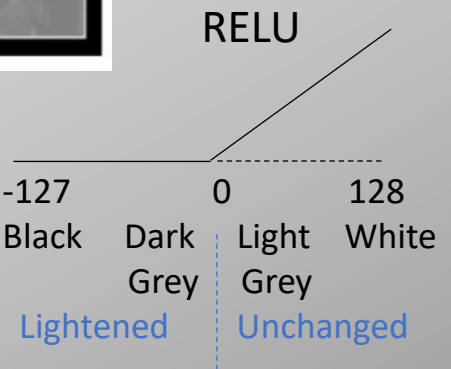
- What are these filters (kernels) detecting?
- What would happen at these nodes (neurons) if the input was "6" ?

Image Source: http://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/

RELU on Greyscale Images



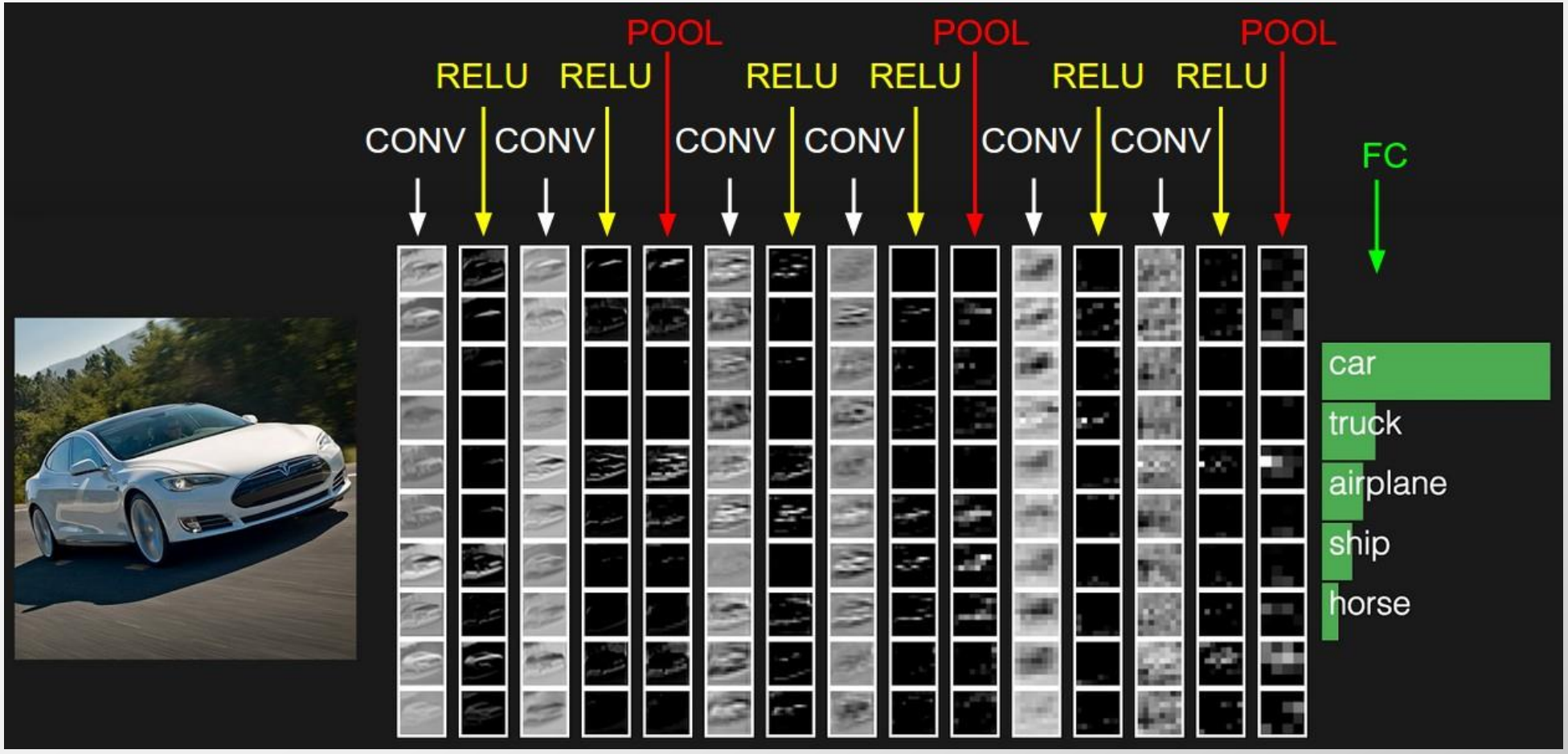
Note: Greyscale values are zero meaned (i.e.) scaled from [0 … 255] to [-127 … 128]



Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

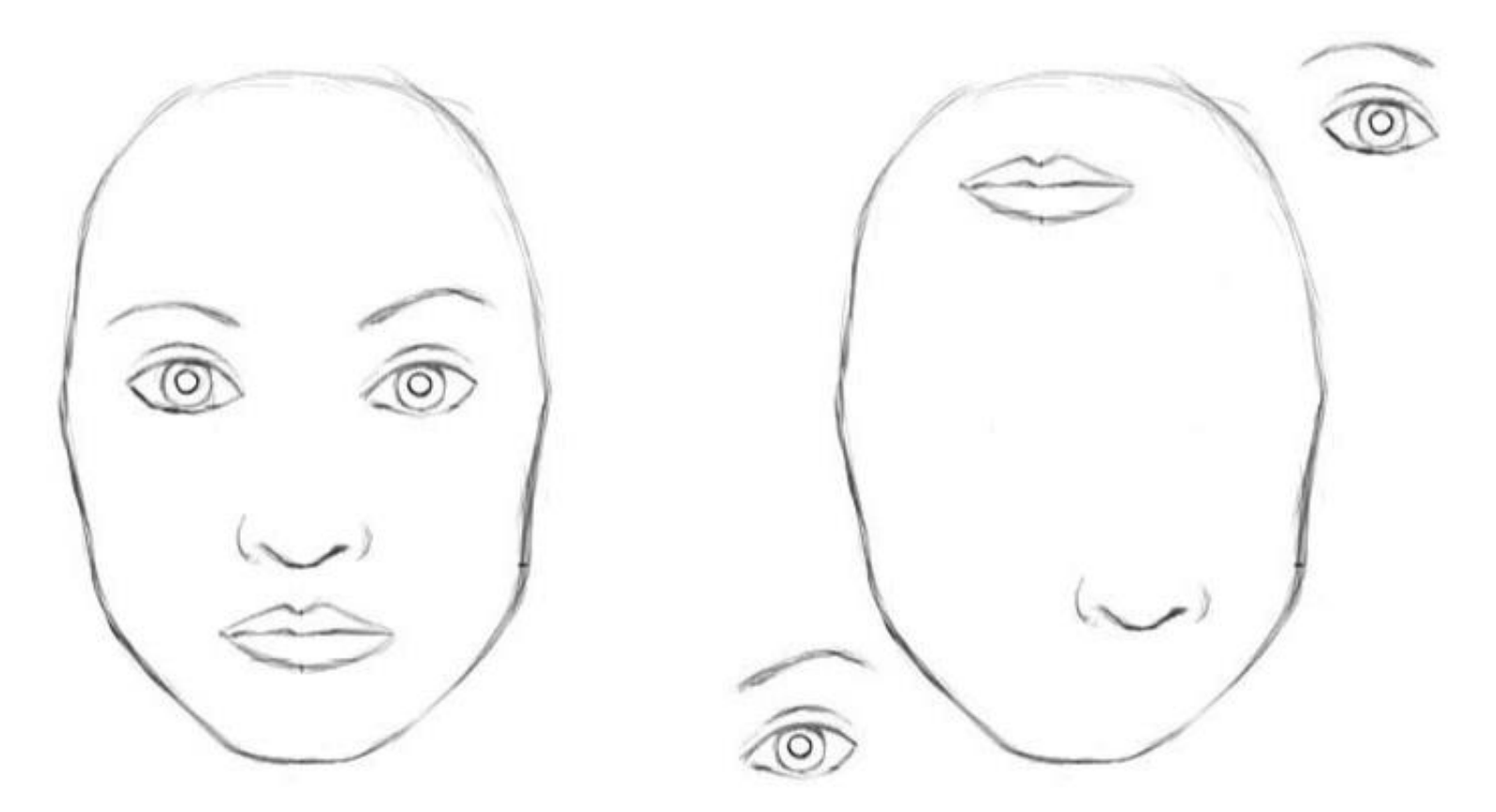
Image Source: http://cs231n.github.io/convolutional-networks/

Activations Across Layers

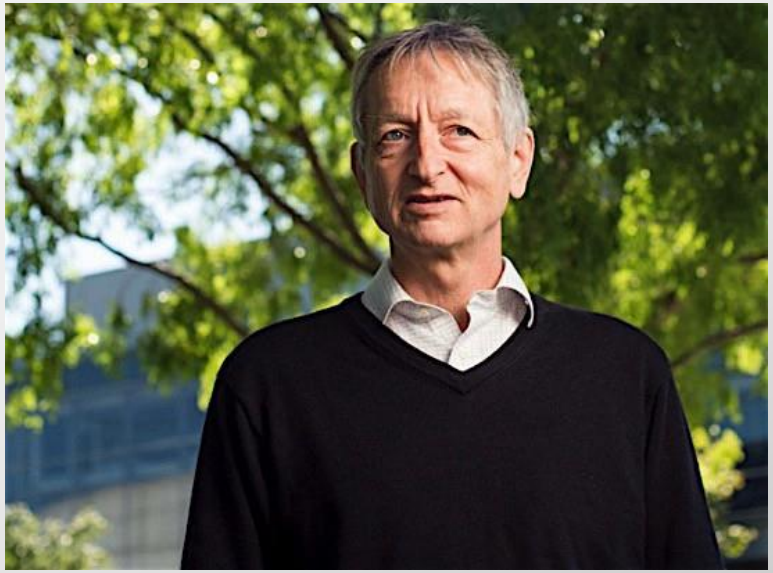


- At each convolution layer 10 different kernels are convolved with feature map input, creating 10 new feature maps
- Scale and rotational invariance – Pooling layers negate the rotation and scale (to a point)
- ReLU highlights dark or bright pixels (negative or positive weights)
- Information is condensed and abstracted as it moves through the layers
- Why does the network yield a small likelihood for classes other than cars?

Criticism of CNNs



Internal data representation of a convolutional neural network does not take into account important spatial hierarchies between simple and complex objects. Why? Consider the information lost in pooling operations.

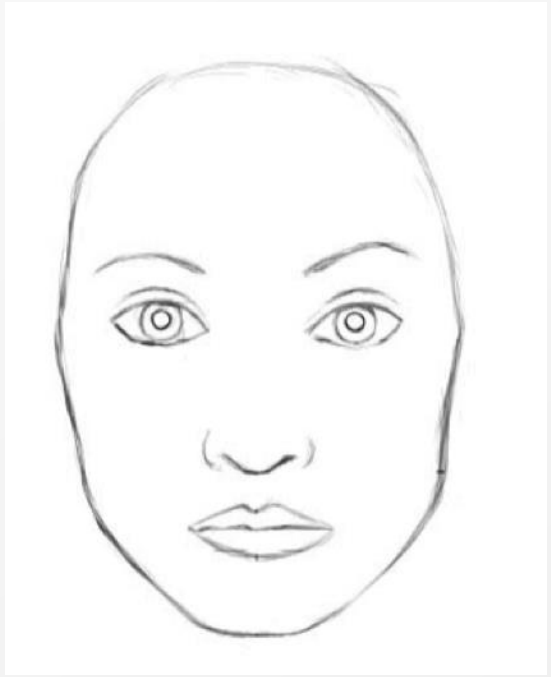


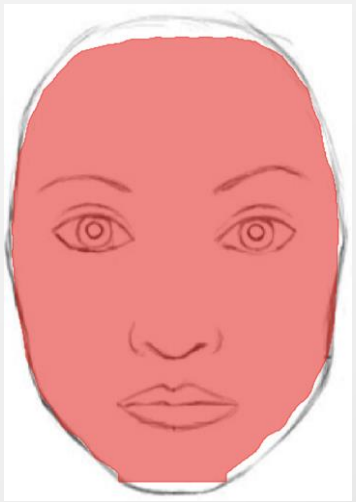
Hinton: "The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster."

Refer to Capsule Networks.

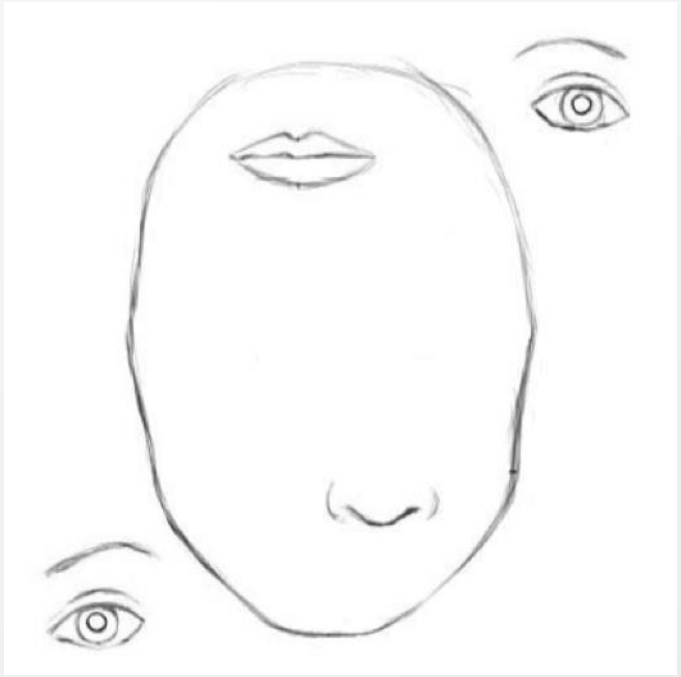
ROI Align – pooling with bilinear interpolation preserves more info

Mask RCNN w/ Faster RCNN Classifier and ROI Align

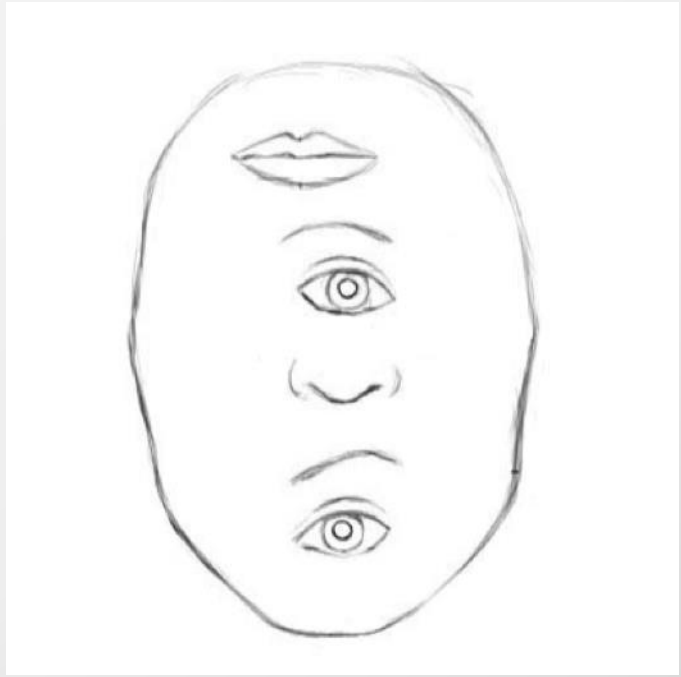


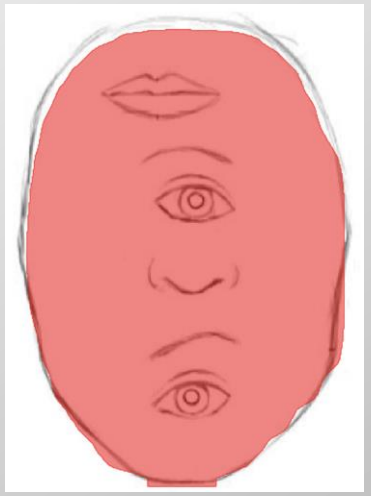


Class: Person



Not Detected





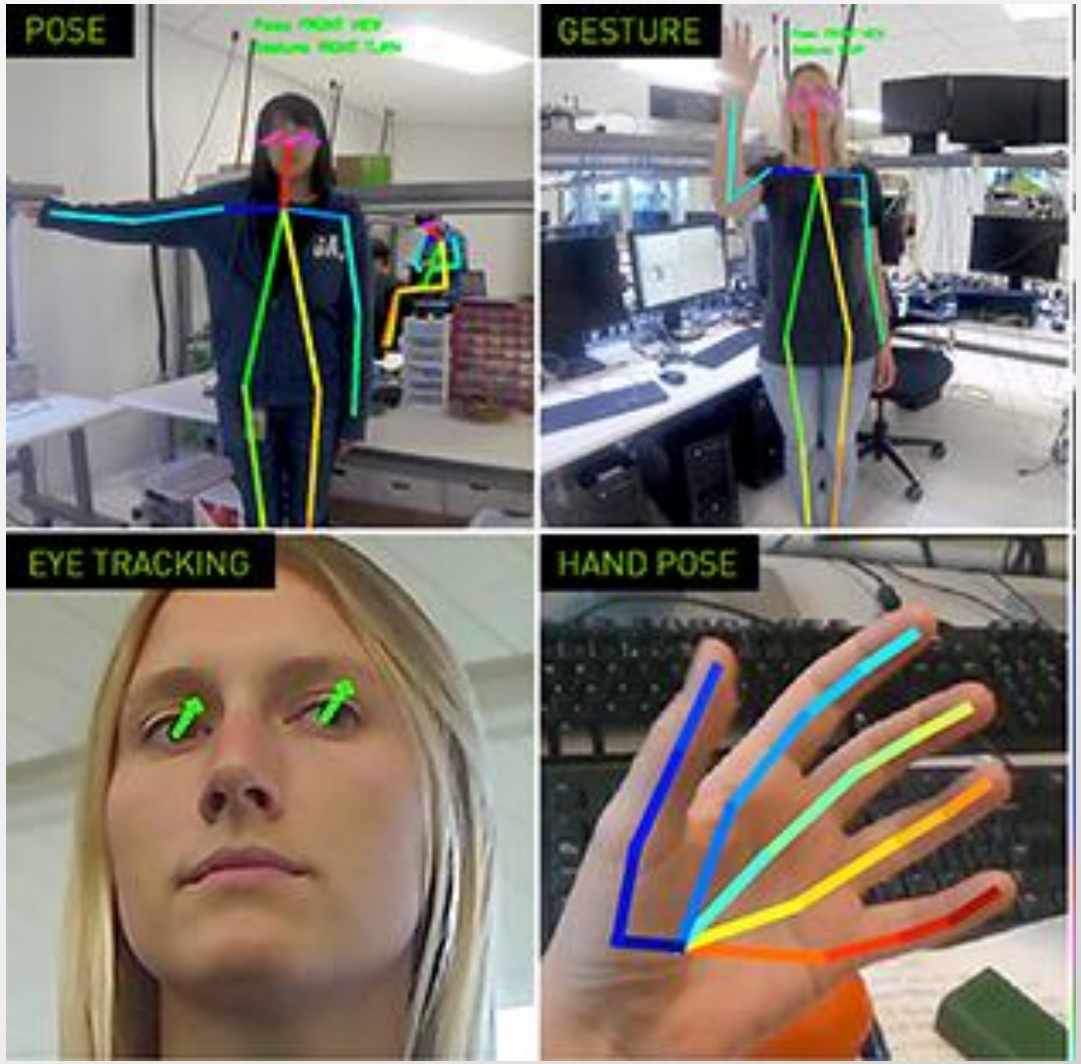
Class: Person

Results Source:
Chris DeBellis

Fooling CNNs and Adversarial Attacks

Source: Ian Goodfellow Lecture 16 | Adversarial Examples and Adversarial Training (YouTube)

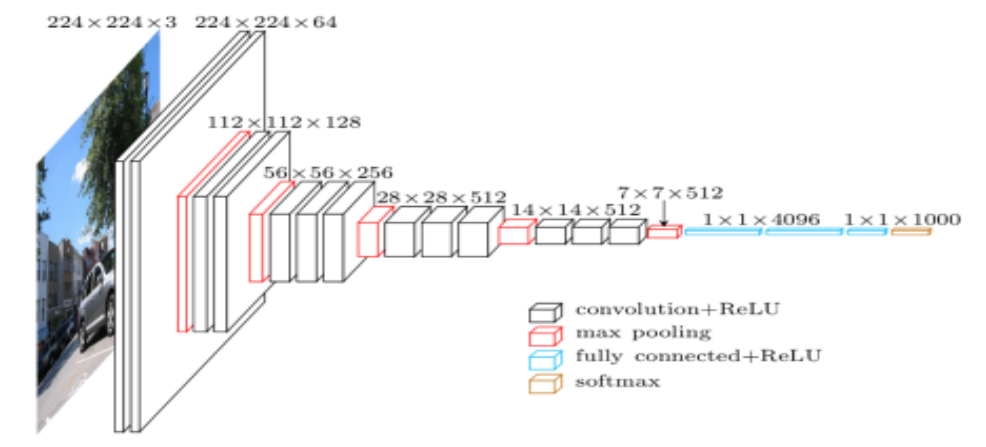
Advances in Deep Learning Perception



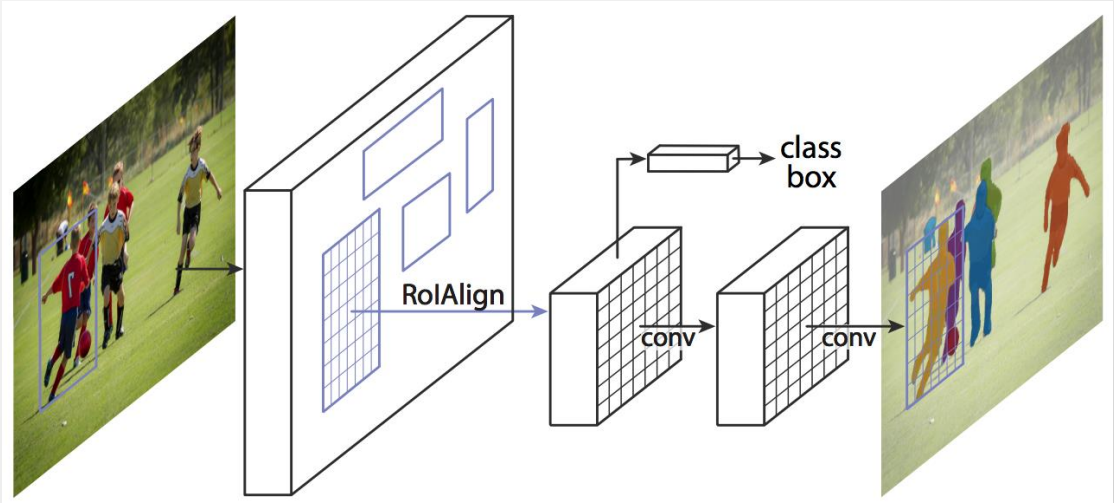
Source: https://www.nvidia.com/en-us/deep-learning-ai/industries/robotics/

Similar Neural Network Architectures

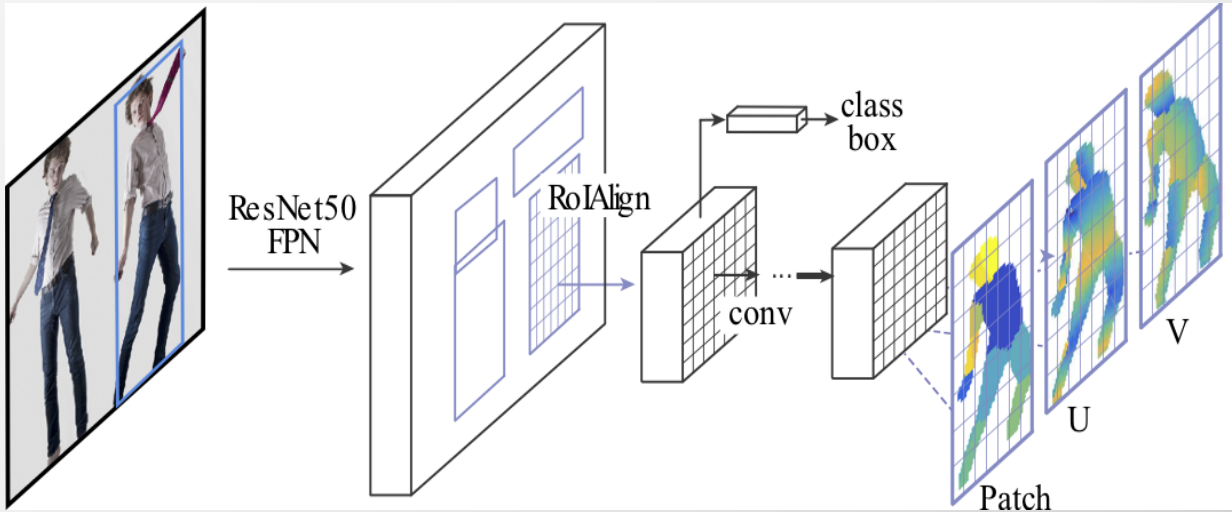
VGG Classifier



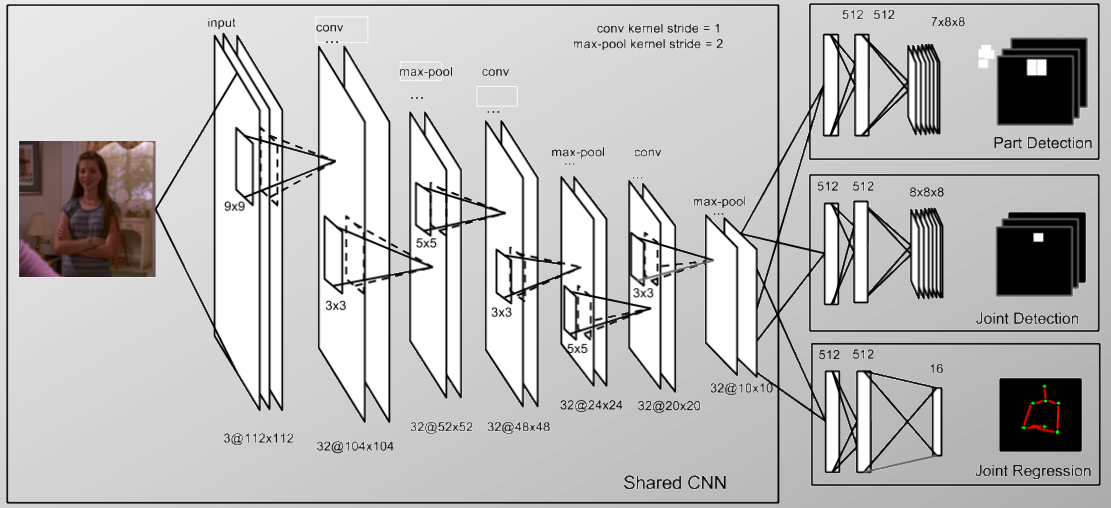
Mask RCNN



Dense Pose



Joint Estimation

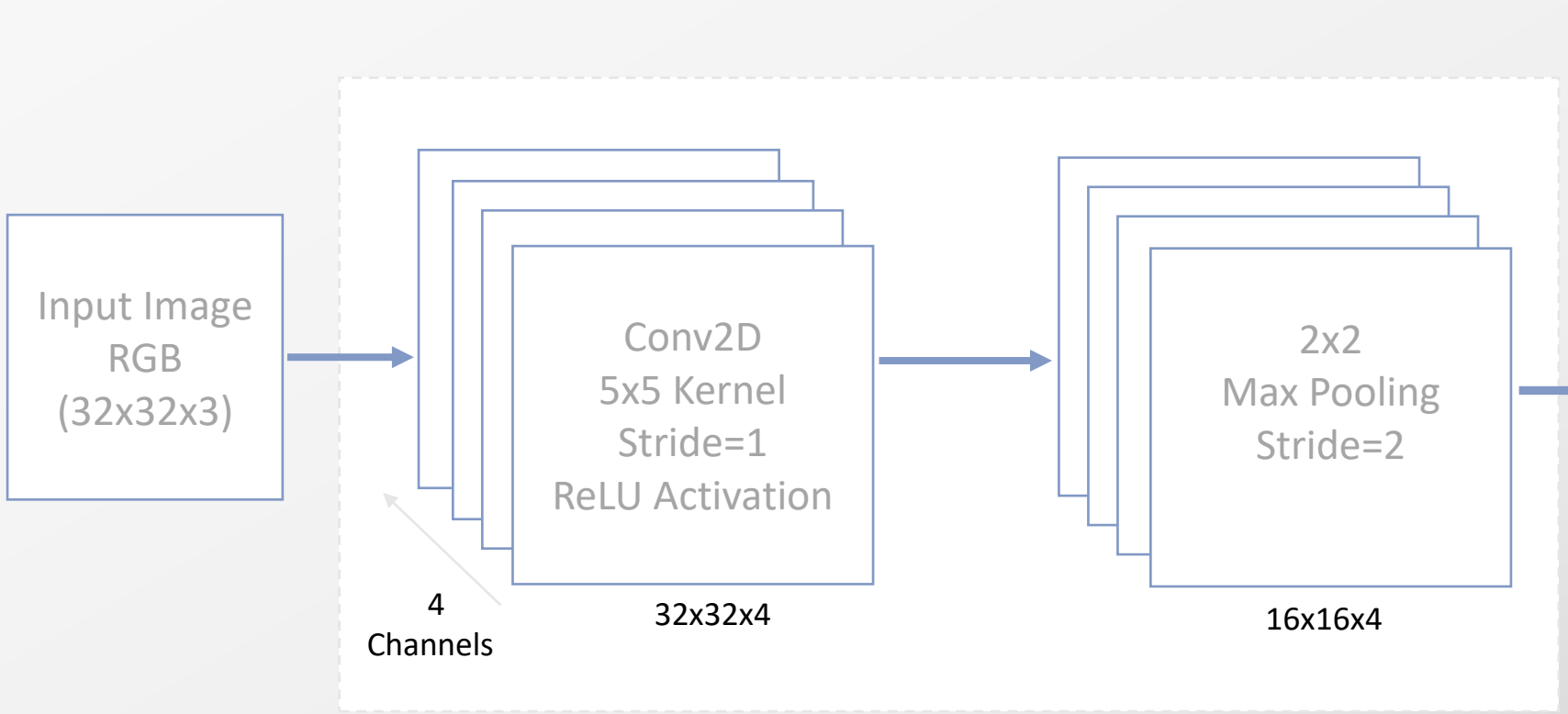


Funny Prediction Fail

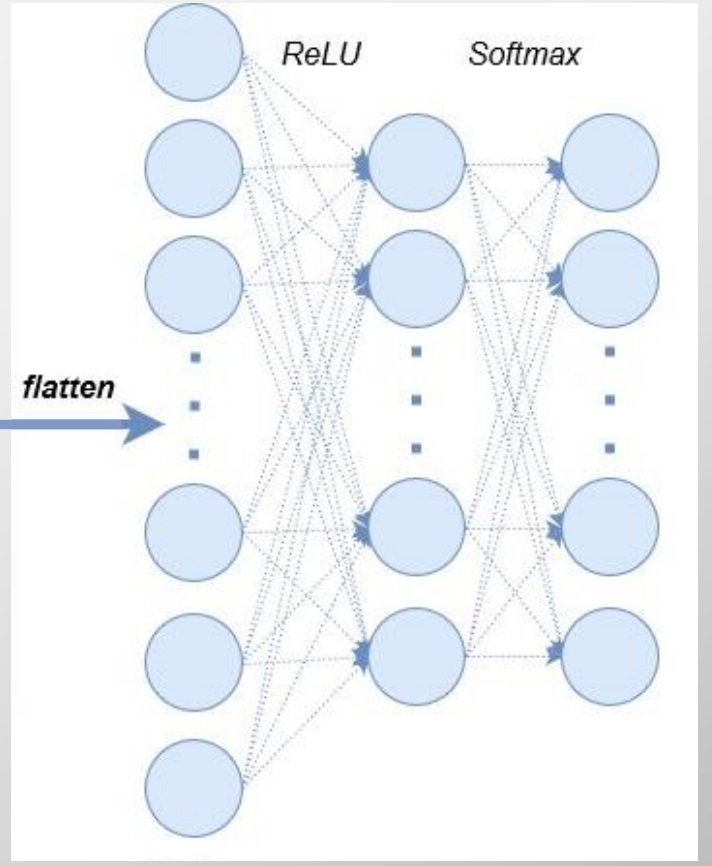
CNN Demo

- MNIST
- Cifar

CNN Model Architecture



Layer 1

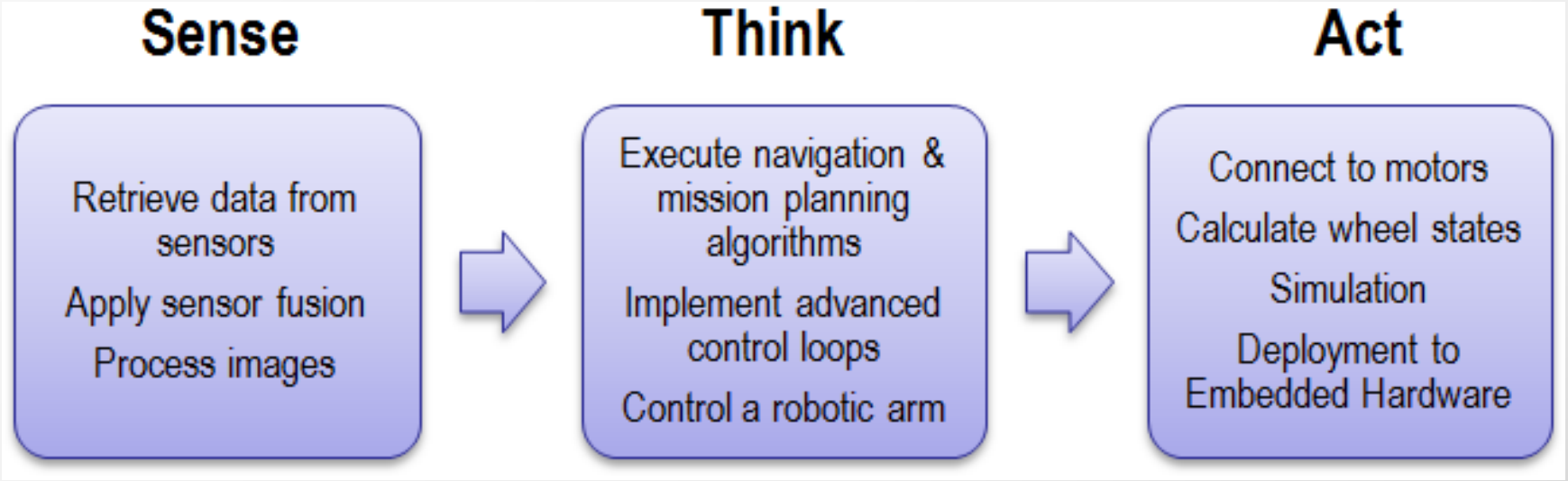


Flattened
1,024 Nodes

Fully
Connected
10 Nodes

SoftMax
Classifier
10 Nodes

Reinforcement Learning

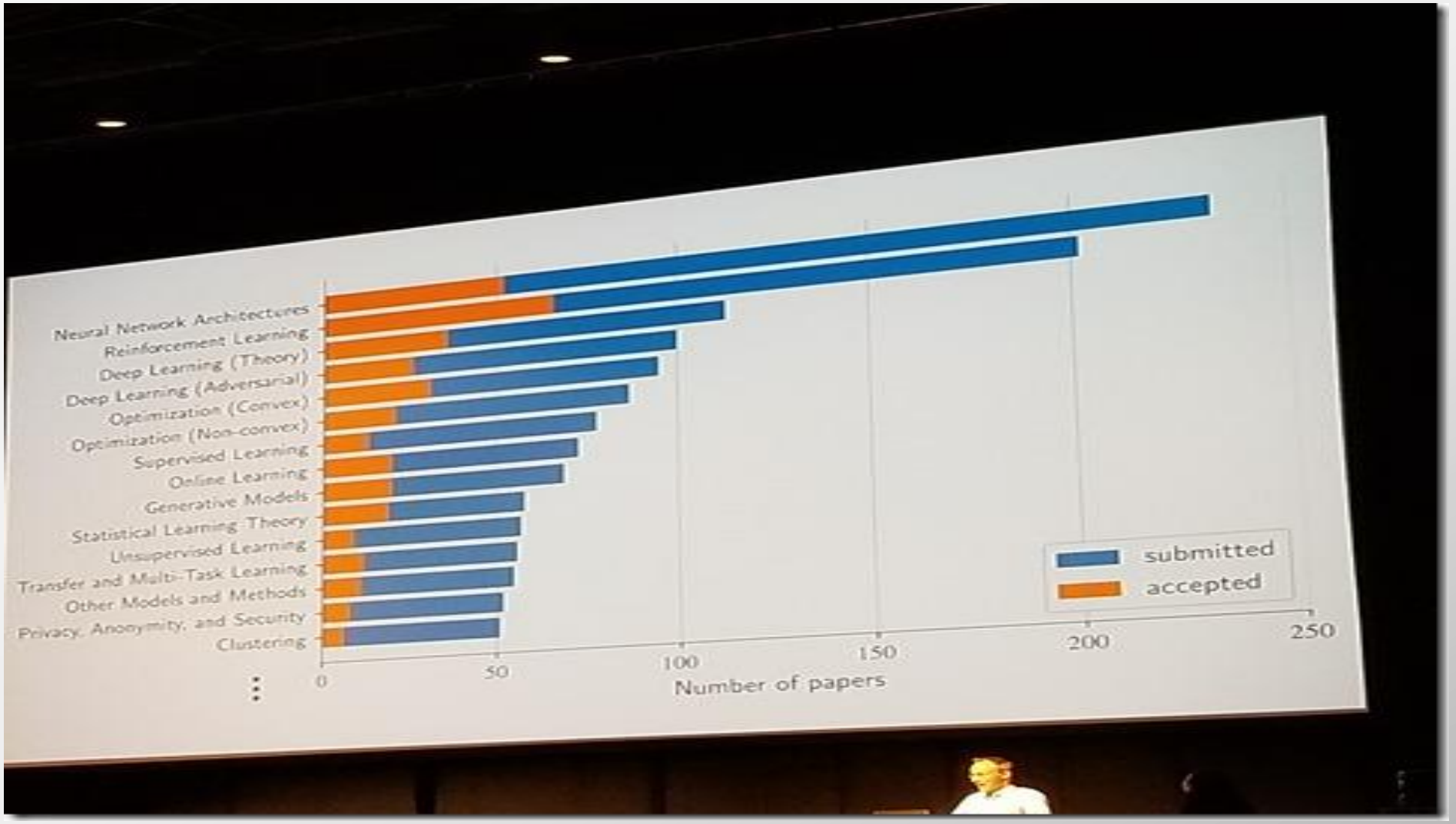


Perception Models (scene understanding and sensor fusion)

Reinforcement Learning Models (choose best action)

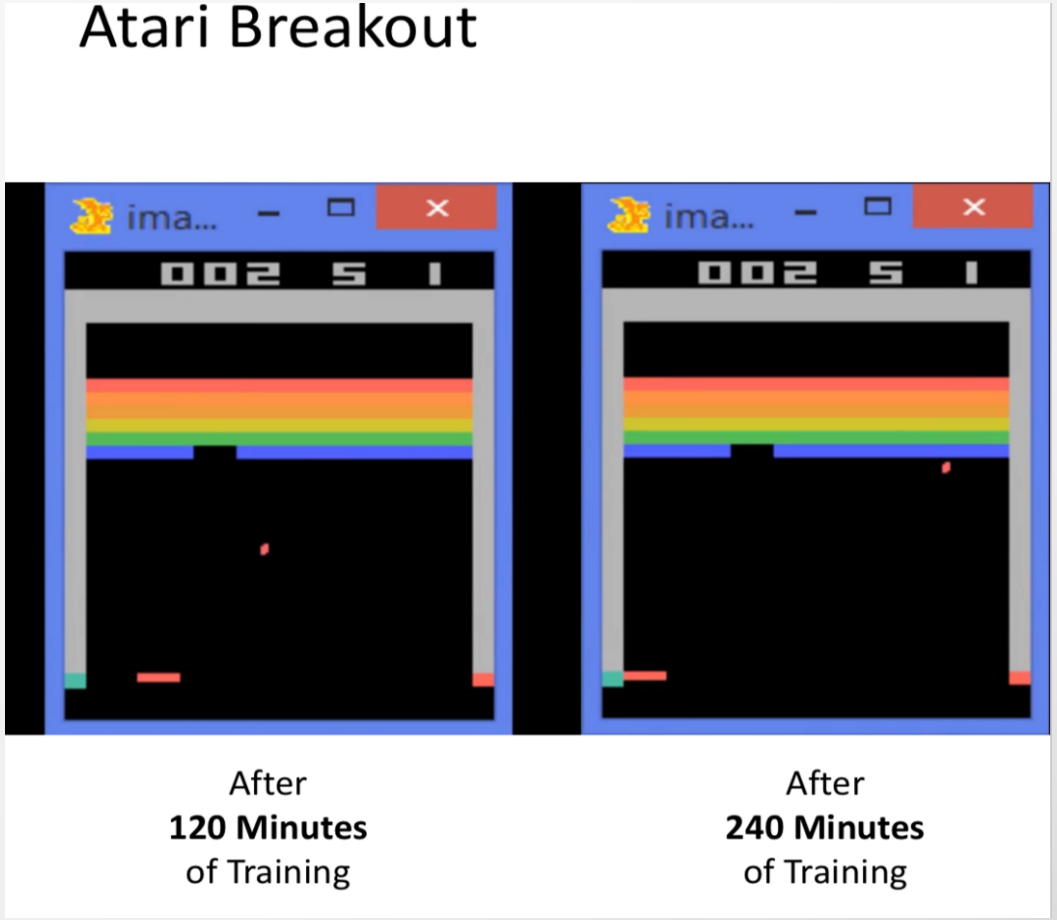
Data Collection for Model Training

Reinforcement Learning Is a Hot Research Area

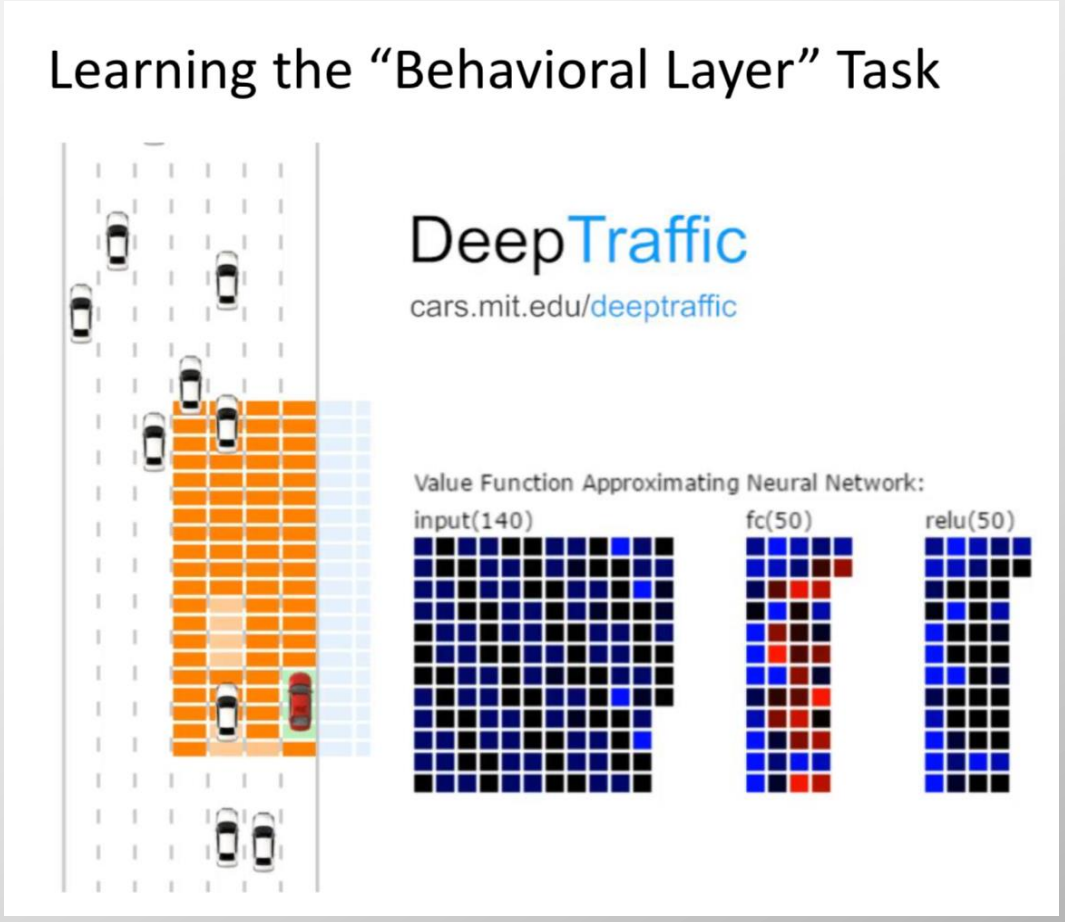


Source: The 35th International Conference on Machine Learning (ICML) - 2018

Reinforcement Learning Use Cases

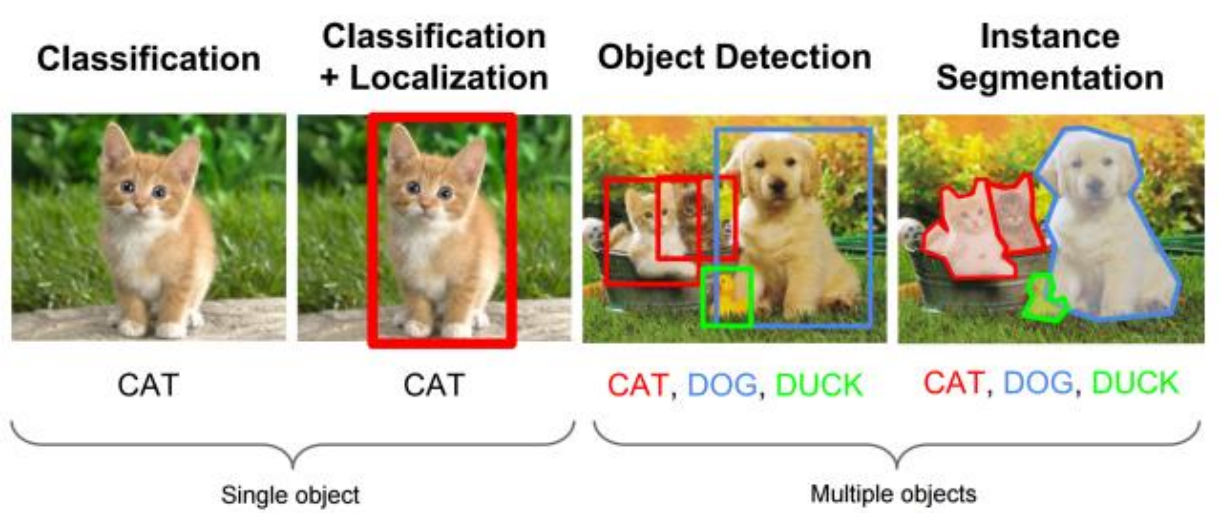


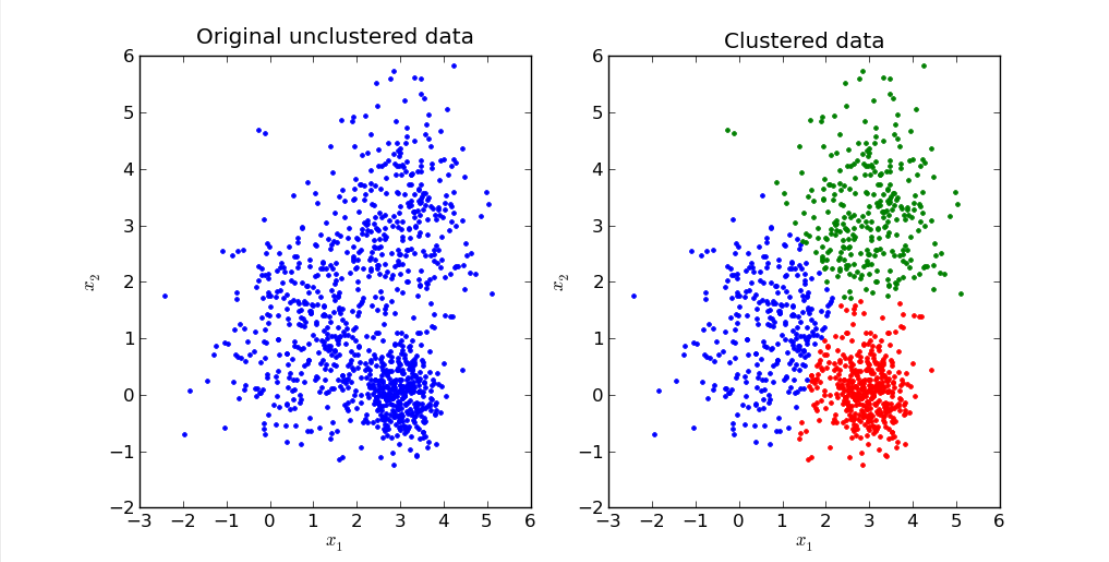
Games – Atari, Doom, Go, etc.



Autonomous Systems – self driving vehicles, robotics, etc.

Machine Learning Approaches





Supervised Learning

- Classification
- Regression
- Deep Learning

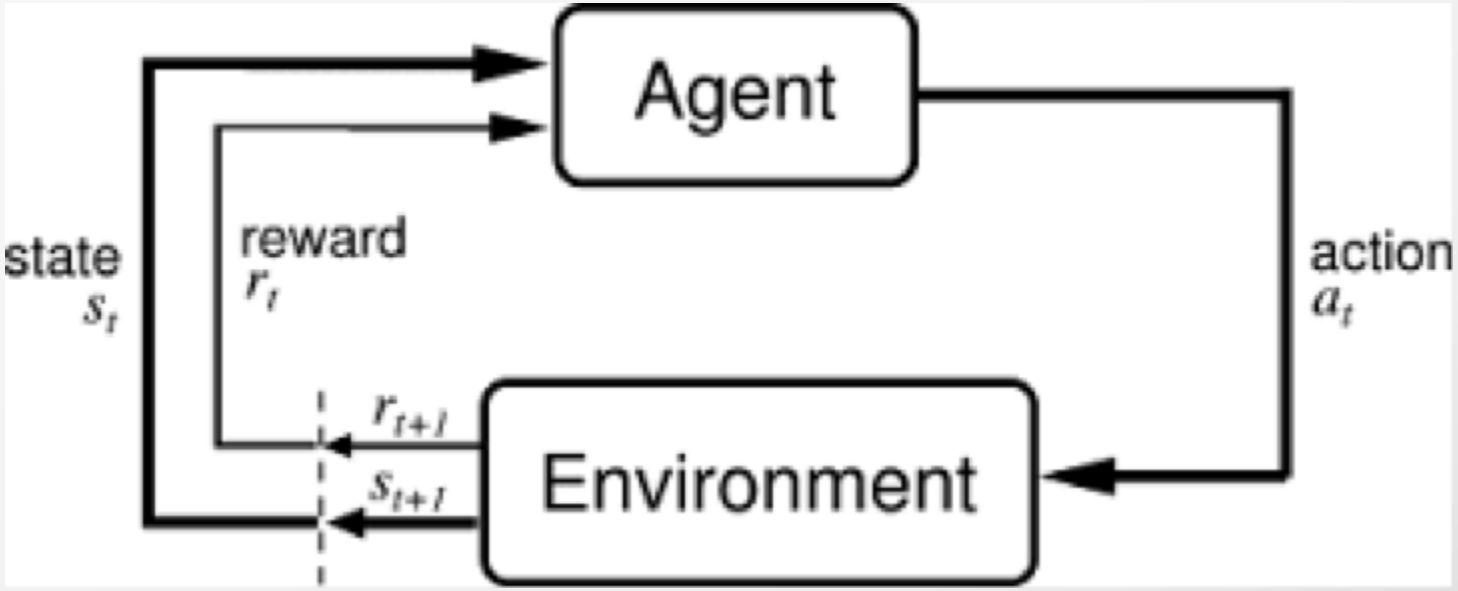
Requires ground truth (i.e. labeled data)

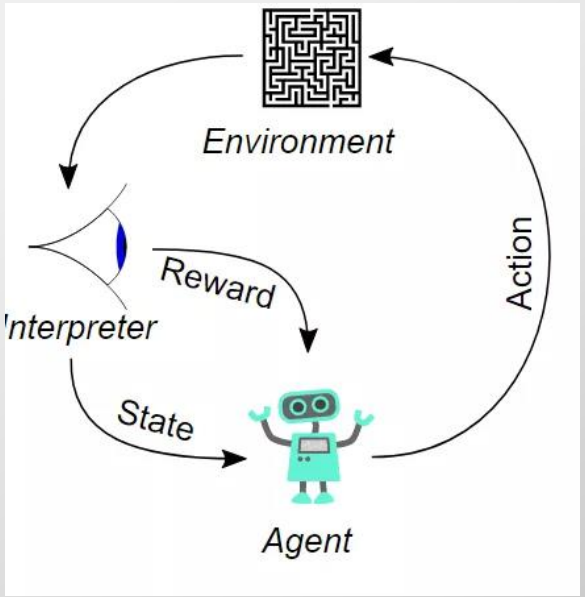
Unsupervised Learning

- (e.g. Clustering)

Group by similarity of features
No ground truth needed, but cannot measure accuracy, however can visualize it

Reinforcement Learning = Experiential Learning





Source: Wikipedia

- Agent (robot, game bot, etc.) operates in an environment
- Agent takes actions, which results in a reward (or penalty) and a new state
- Maximize the rewards (short or long term)
- May or may not use deep learning (probably does except for simple environments)

Reinforcement Learning

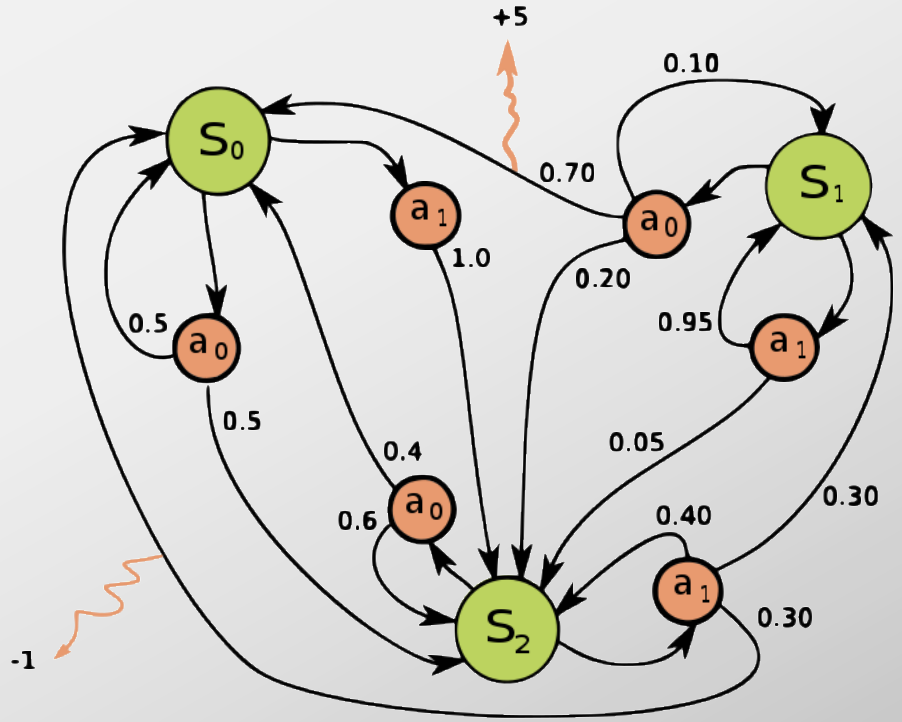
Markov Decision Process (MDP)

S = set of all states
A = set of all actions
R = immediate reward for taking an action
$P_a(s, s')$ = "The Model" = state transition probabilities for the environment

Can be implemented as a table, array, list, etc. of [state][action] pairs (Q table)

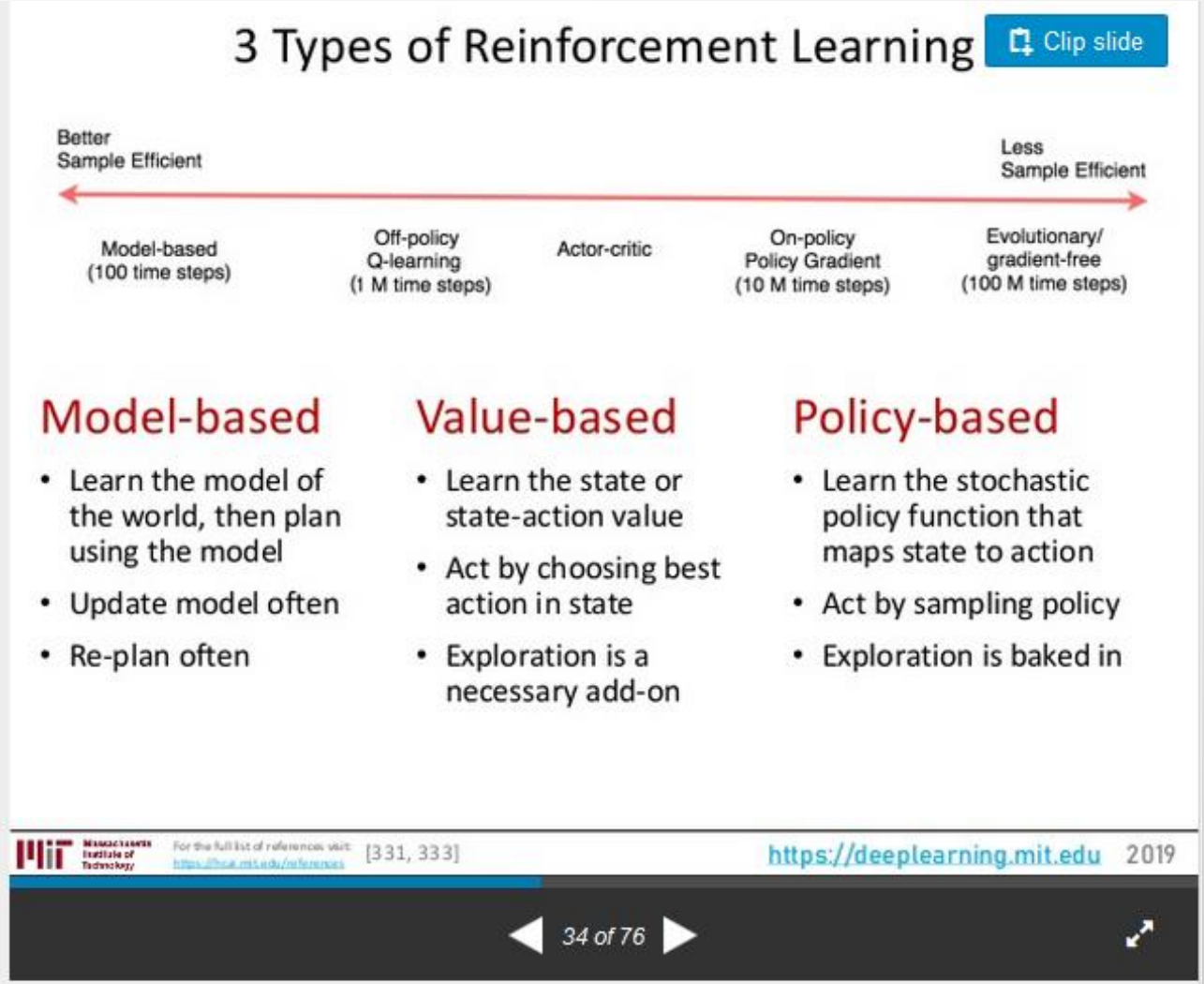
S_0 A_0
S_0 A_1
…
S_0 A_N

S_1 A_0
S_1 A_1
…
S_K A_0
S_K A_1
…
S_K A_N

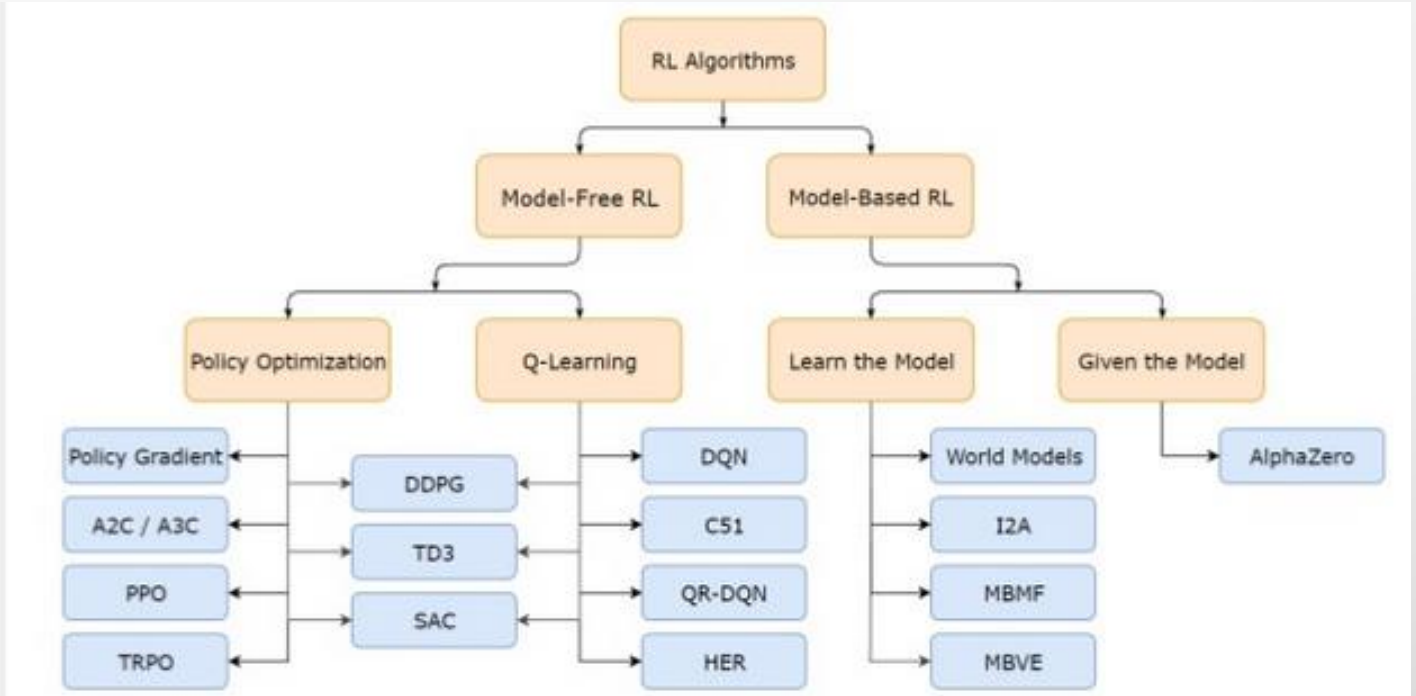


For this model, the Q Table would contain 6 values (3 states * 2 actions)

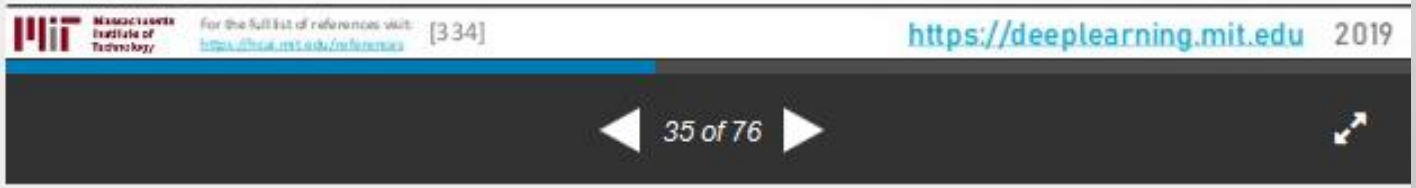
Types of RL


3 Types of Reinforcement Learning
Clip slide
Better
Sample Efficient
Less
Sample Efficient
Model-based
(100 time steps)
Off-policy
Q-learning
(1 M time steps)
Actor-critic
On-policy
Policy Gradient
(10 M time steps)
Evolutionary/
gradient-free
(100 M time steps)
Model-based
• Learn the model of the world, then plan using the model
• Update model often
• Re-plan often
Value-based
• Learn the state or state-action value
• Act by choosing best action in state
• Exploration is a necessary add-on
Policy-based
• Learn the stochastic policy function that maps state to action
• Act by sampling policy
• Exploration is baked in
[331, 333]
https://deeplearning.mit.edu 2019
34 of 76
MIT 6.S091: Introduction to Deep Reinforcement Learning (Deep RL) by Lex
176 views

Types of RL



Link: https://spinningup.openai.com



MIT 6.S091: Introduction to Deep Reinforcement Learning (Deep RL) by Lex

176 views

On-Line vs. Off-Line Learning

- On-line Learning
  - Put the robot / agent in the environment to learn
  - Is the environment safe? Can the robot take a fatal action?
  - May be slow to learn
    - Robots are generally slow moving.
    - Resets are probably also slow.

- Off-line Learning
  - Learn using a simulator to simulate the environment
  - Transfer the learned policy to the agent and let the agent run in the real environment
  - Difficult to build a simulator that accurately reflects the physics of the real world
  - There will likely be a delta between the learned best policy of the simulated environment and the actual best policy for the real environment.

Q-Learning

- V = value (utility / usefulness of being in a given state)
  - Closer to a large reward is more valuable
- Q(*s, a*) = Q-value is the value of taking action *a* at a given state *s*
- Q learning = use Monte Carlo simulation to learn the (*s, a*) values
  Also can infer the transition probabilities
- Policy (∏) maps best action to take for every state
  - By learning Q we infer the policy (don't directly learn the environment / model ➔ model free)
- Learn by iterating (Monte Carlo Simulation)

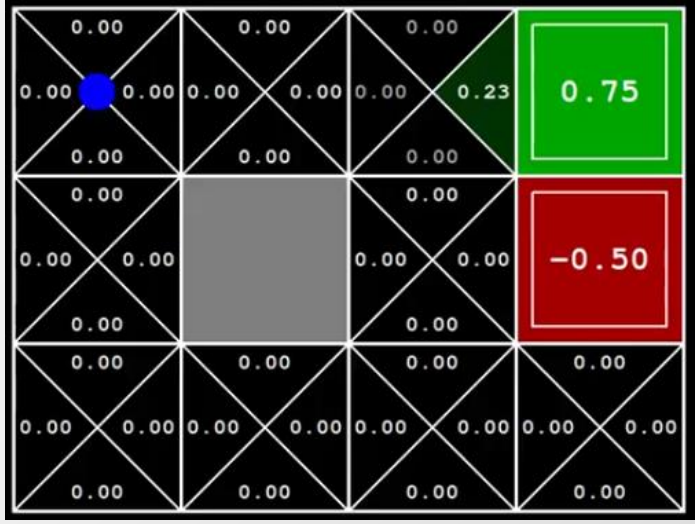
Bellman Equation

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

- Only one state-action pair is updated per action
- Referred to as Bellman Backup

https://skymind.ai/images/wiki/q_learning_equation.png

Q Learning Demo – Grid World



- Agent (blue dot) moves around the grid
  - Only 4 possible actions – move up, down, left, right
- The only states are the position of the robot in 3x4 grid
- 2 possible termination states

- Q table needed is small (< 48 values for the 12 states x 4 actions)

- Only one Q-value updates when you take an action and move to new state
- Q values converge eventually BUT you need sufficient samples
  - Note changes to Q values in bottom left @0:49 for left and up actions
- Nondeterministic model / environment
  - with some probability your agent moves in the wrong direction (robot fails to complete the action and ends up in unexpected state) – refer to video @0:11

Grid World Demo

https://www.youtube.com/watch?v=AMnW-OsOcl8

alpha (learning rate)	0.5	
gamma (discount rate)	0.9	
Reward	0	
Current Q	Next Q	New Q
0	0.5	0.23
0.23	0.75	0.45
0	0.23	0.10
		0.00
Exit Condition		
0	1	0.50
0.5	1	0.75
0.75	1	0.88

Q Learning Concepts

- What is the value of a state?

  See end of video

  $V(s) = Max_a Q(s,a)$

What action do we take?

$a(s) = argmax_a\ Q(s,a)$

What is the optimal policy Π^*?

At any state, take the action that yields greatest Q value (total reward according to Bellman equation)

Rewards Affect Learned Behavior

- How can we encourage the agent to finish as quickly as possible?
- What value should the reward be?

- If the reward is -1, what would the agent do when it was in the bottom right state?

- What if we want to encourage survival in a game?

- But what would the agent do if it is at grid coordiante (1,3) and the reward is +1?

ε-Greedy (Epsilon Greedy) Algorithm

- Exploration vs Exploitation

- During training:
  - With some probability (1-ε) take the optimal action known at that time (exploitation)
  - Else with probability ε, take some other action (exploration)
    - Several methods exist for determining which other action to take

- Why do we need exploration?
- What should we do in production?

Follow the learned policy Π^* → No exploration!

Q-Learning Example – OpenAI Gym Taxi

https://learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

6 Actions

- south
- north
- east
- west
- pickup
- dropoff

500 States

25 grid locations
4 destinations / pickup-up points
5 possible passenger locations
(4 pickup points + 1 inside taxi)

How large does the Q table need to be?

How do we differentiate each state?
(taxi row, taxi col, passenger loc, dest)

Reward=-1 or -10 or +20. Why -1?

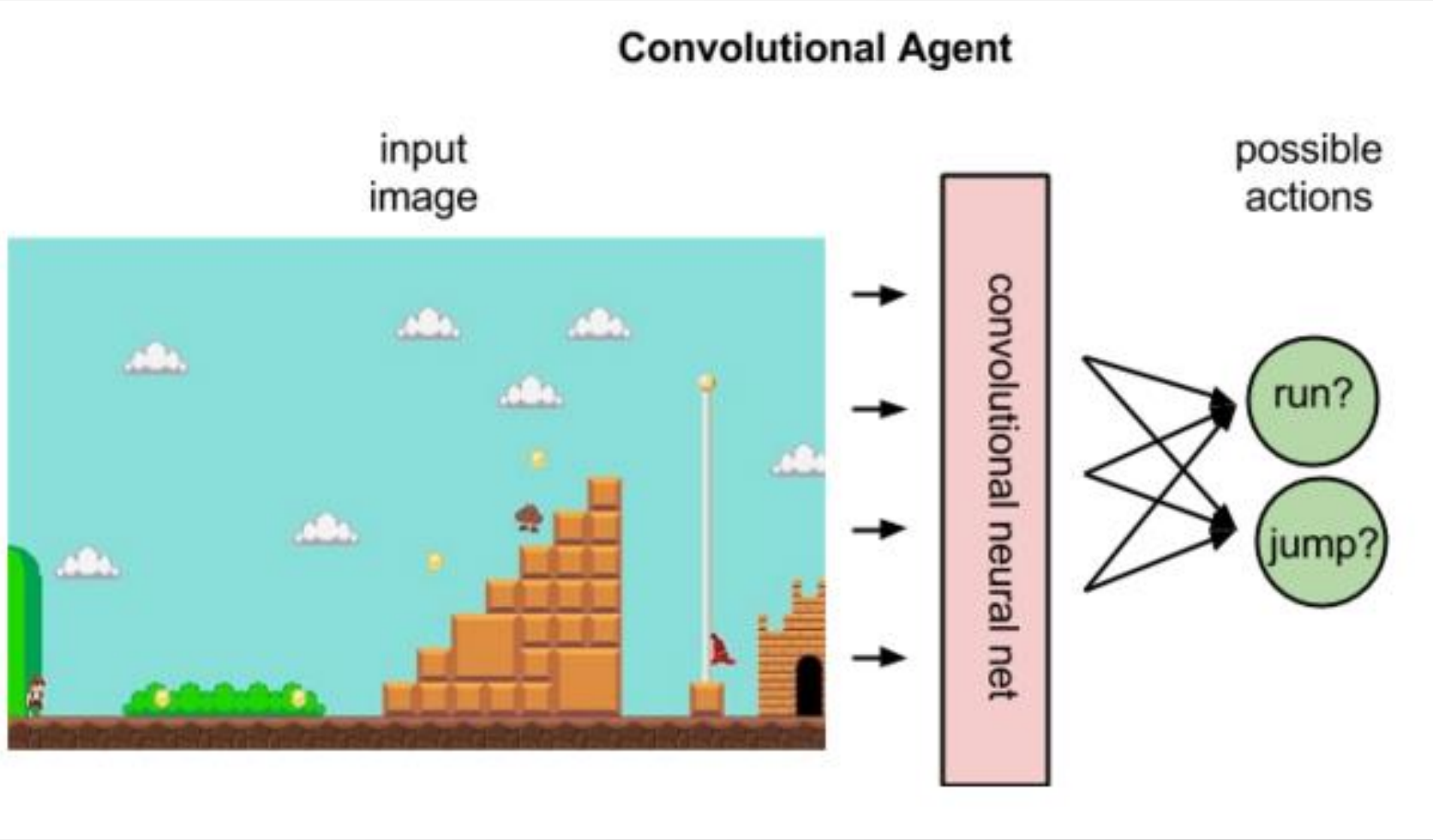
Pickup passenger at Y and drop off at R

Problems with Q-Learning

- Large State Space
  - Needs huge amount of memory for all the (s,a) pairs
  - Need to visit all of the states and take each action multiple times to ensure convergence → takes too long
  - Need to discretize continuous values (velocity, position, rotation, etc.)

- How many states are there in:
  - A tic-tac-toe game? 3^9 = 19,683 including invalid states (upper bound)
  - A game of chess? Estimated to be ~7.7×10^{45}
  - A self driving car? Depends on level of accuracy of velocity, position, etc.
    More states if you round velocity to nearest tenth vs MPH

Deep Q-Learning to the Rescue (Sorta)



Use a neural network to map states to actions rather than build a table for every state.

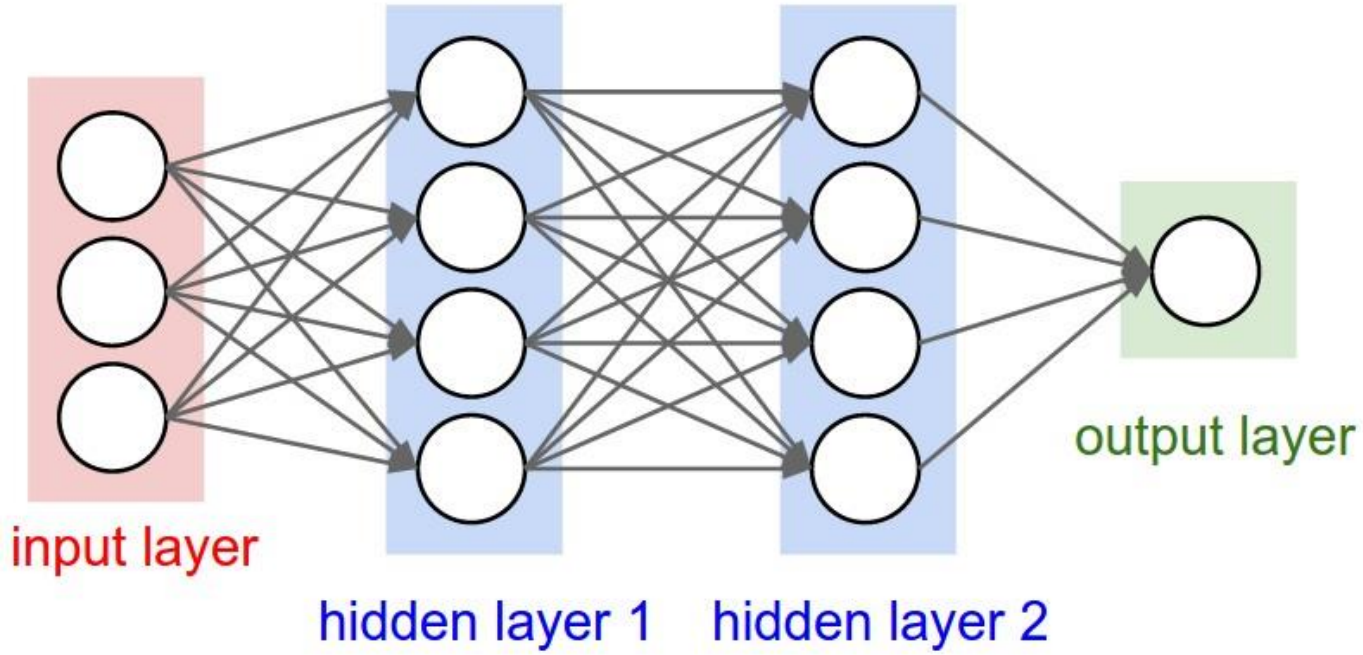
Useful for:

- Very large state space where it is impractical to visit every (s,a) pair
- Continuous state space (avoids the need to discretize states)

Neural network is a universal approximation function

https://skymind.ai/wiki/deep-reinforcement-learning

Feed Forward Neural Network (MLP)



https://nutricaobrasil.wordpress.com/esclerose-neuronios-cerebro-20110608-size-62-2/

- Layers contain nodes that perform mathematical functions on a set of inputs
- Activation functions (represented as circles in the image) bound the output range and add non-linearity
- Outputs of one layer are inputs to the next layer (feed forward)
- Deep networks have more layers
- Motivation derived from our understanding of human brain function

Deep Q-Network (DQN)

What do we need to do supervised learning?

- Ground truth so we can calculate the error of the predictions
- Objective function (loss to minimize)
- How do we implement the Expectation of the Random Variables?

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

New predicted value (target) – original Q value (before taking the action)

Note: this equation used for L_2-norm loss with MSE. Could use L_1-norm loss with MAE instead (Absolute value of the difference).

Problems Implementing the Neural Network

1. Need iid (independent, identically distributed) samples
   Sequential play inherently makes samples correlated
   Network will otherwise forget past learning and optimize for newest samples
2. Ground truth Q changes with each iteration and target depends on Q

   → Chasing (optimizing) a non-stationary target

Experience replay buffer

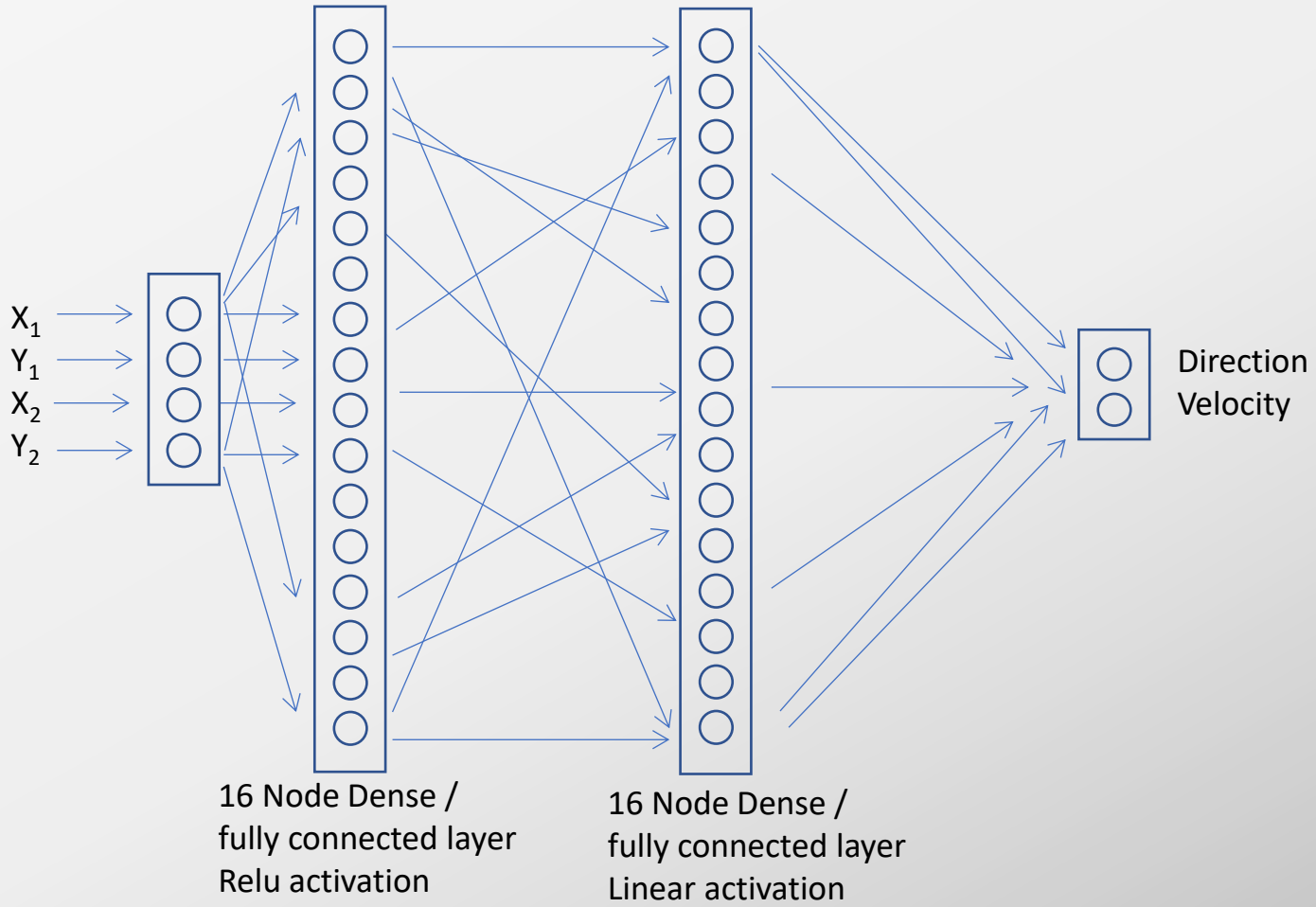
- Save all (s,a,r,s') values in memory
- Train the network on randomly selected samples from the buffer in batches
  - Eliminates sequential correlation
  - Ground truth is static for that batch so neural net can converge

CartPole Demo

- https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/

- Video https://youtu.be/XiigTGKZfks

DQN Architecture for CartPole Example



Questions