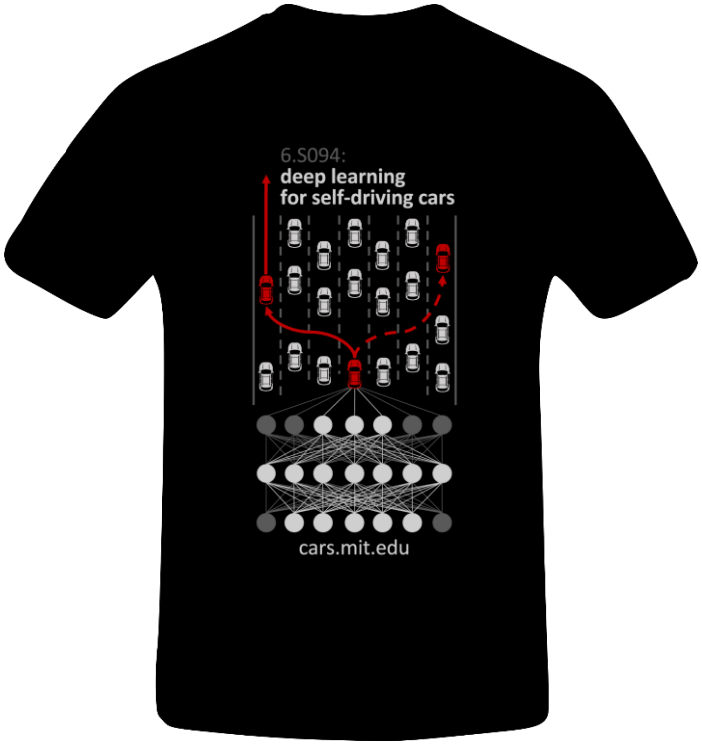
6.S094: Deep Learning for Self-Driving Cars

Learning to Drive: Convolutional Neural Networks and End-to-End Learning of the Full Driving Tasks

cars.mit.edu

Administrative



- **Website:** cars.mit.edu
- **Contact Email:** deepcars@mit.edu
- **Required:**
  - Create an account on the website.
  - Follow the tutorial for each of the 2 projects.
- **Recommended:**
  - Ask questions
  - Win competition!
- **Office hours:** Friday, 5-7pm
  (more info coming soon)

Administrative

- **Website:** cars.mit.edu
- **Contact Email:** deepcars@mit.edu
- **Required:**
  - Create an account on the website.
  - Follow the tutorial for each of the 2 projects.
- **Recommended:**
  - Ask questions
  - Win competition!

Schedule

Mon, Jan 9	Introduction to Deep Learning and Self Driving Cars
Tue, Jan 10	**Learning to Move:** Reinforcement Learning for Motion Planning
	DeepTraffic: Solving Traffic with Deep Reinforcement Learning
Wed, Jan 11	**Learning to Drive:** End-to-End Learning for the Full Driving Task
	DeepTesla: End-to-End Learning from Human and Autopilot Driving
Thu, Jan 12	**Karl Iagnemma:** From Research to Reality: Testing Self-Driving Cars on Boston Public Roads
Fri, Jan 13	**John Leonard:** Mapping, Localization, and the Challenge of Autonomous Driving
Tue, Jan 17	**Chris Gerdes:** TBD
Wed, Jan 18	**Sertac Karaman:** Past, Present, and Future of Motion Planning in a Complex World
Thu, Jan 19	**Learning to Share:** Driver State Sensing and Shared Autonomy
Fri, Jan 20	**Eric Daimler:** The Future of Artificial Intelligence Research and Development
	Learning to Think: The Road Ahead for Human-Centered Artificial Intelligence

DeepTraffic Leaderboard

Rank	User	MPH
1	cmauck10	72.67
2	Jeffrey Li	72.45
3	Ted Grunberg	72.31
4	Xiaoxue Wang	72.31
5	Ethan Weber	71.73
6	Indra den Bakker	71.35
7	Kavya R.	71.24
8	Rakesh	71.21
9	LCMartin	71.02
10	anyati	70.95

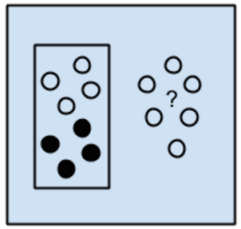
Illustrative Case Study: Traffic Light Detection

DeepTesla: End-to-End Learning from Human and Autopilot Driving

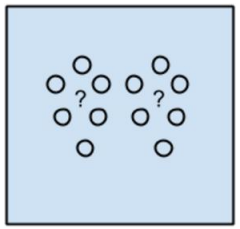
(in ConvnetJS)

DeepTesla: End-to-End Learning from Human and Autopilot Driving
(in TensorFlow)

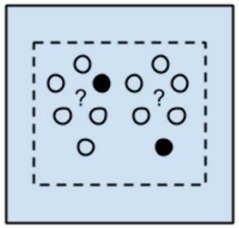
Computer Vision is Machine Learning



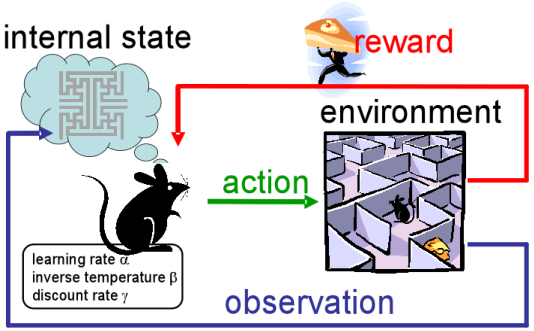
Supervised Learning



Unsupervised Learning



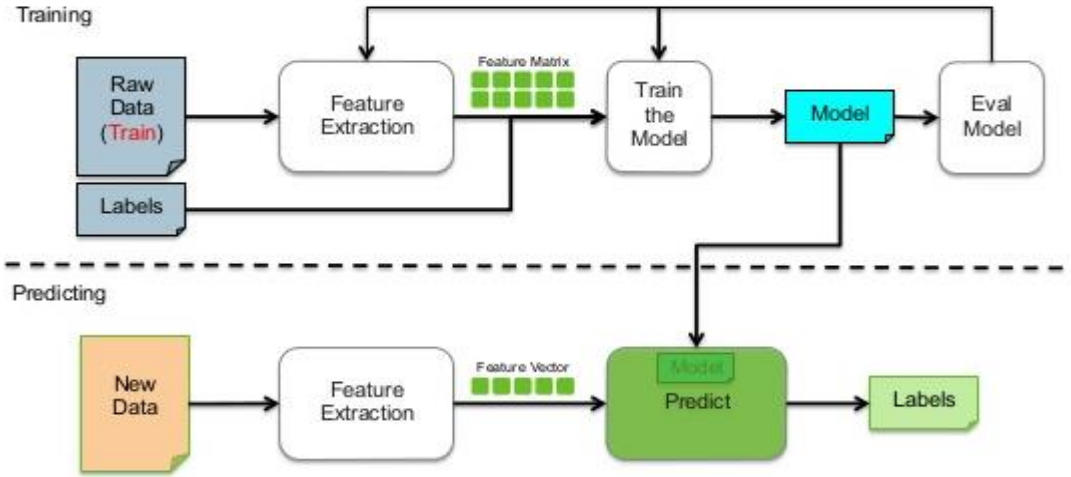
Semi-Supervised Learning



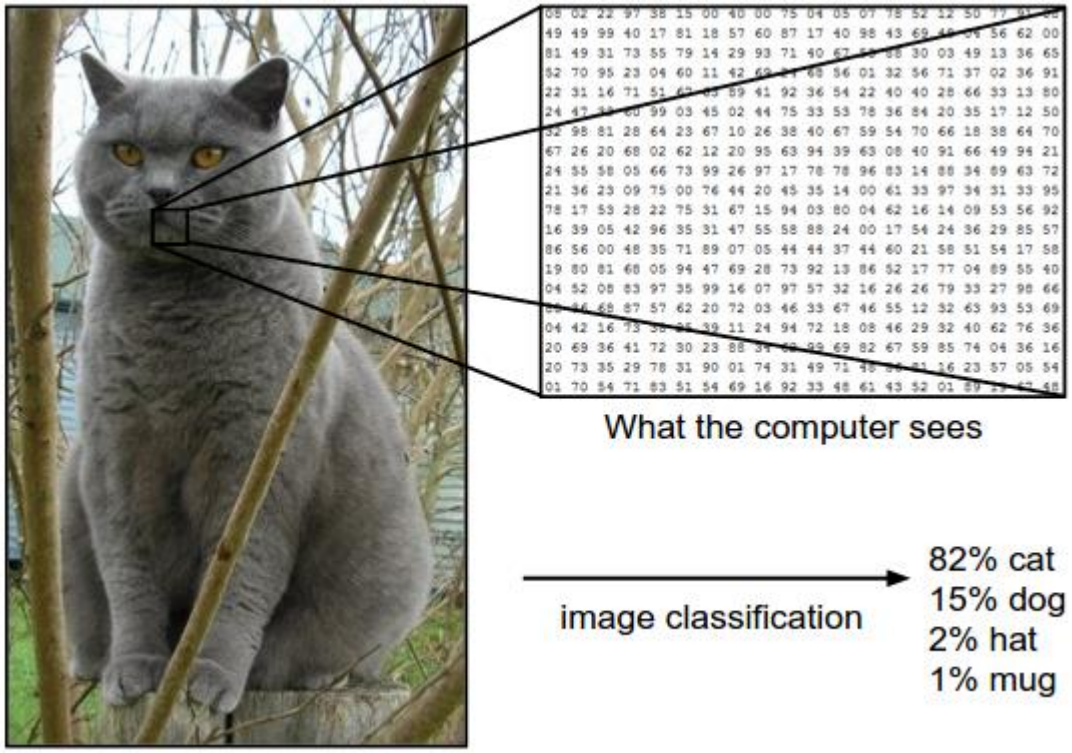
Reinforcement Learning

Computer Vision

Standard supervised learning pipeline:

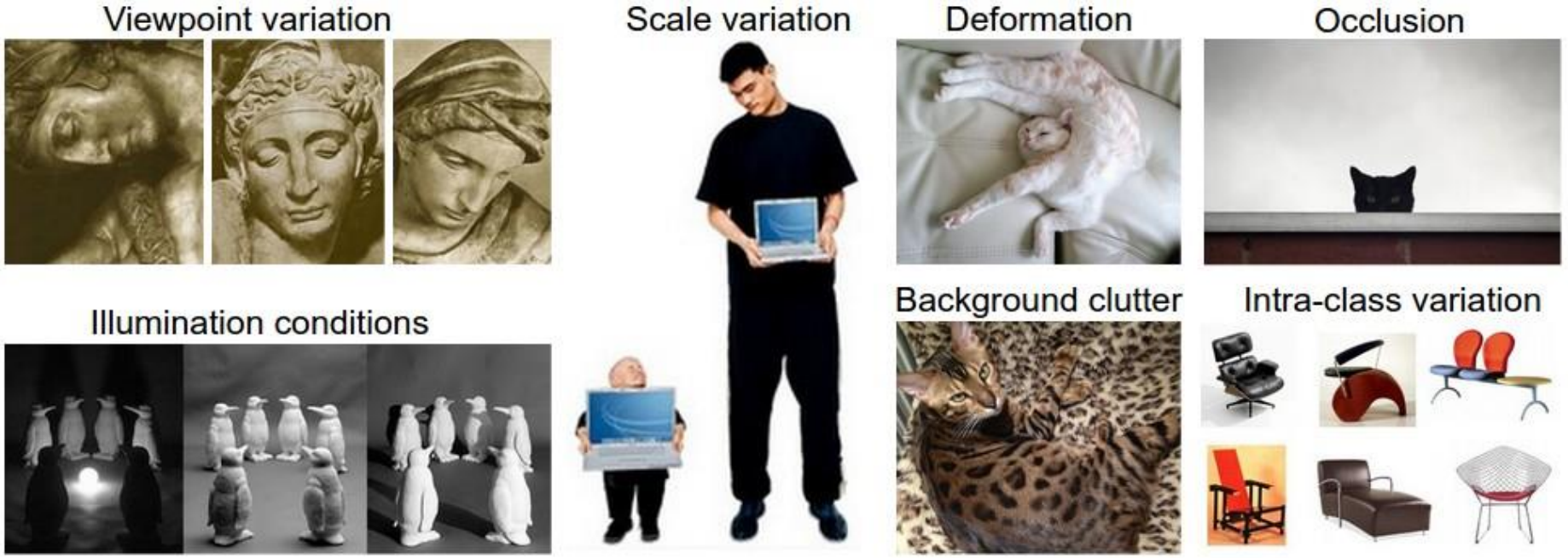


Images are Numbers



- **Regression:** The output variable takes continuous values
- **Classification:** The output variable takes class labels
  - Underneath it may still produce continuous values such as probability of belonging to a particular class.

Computer Vision is Hard



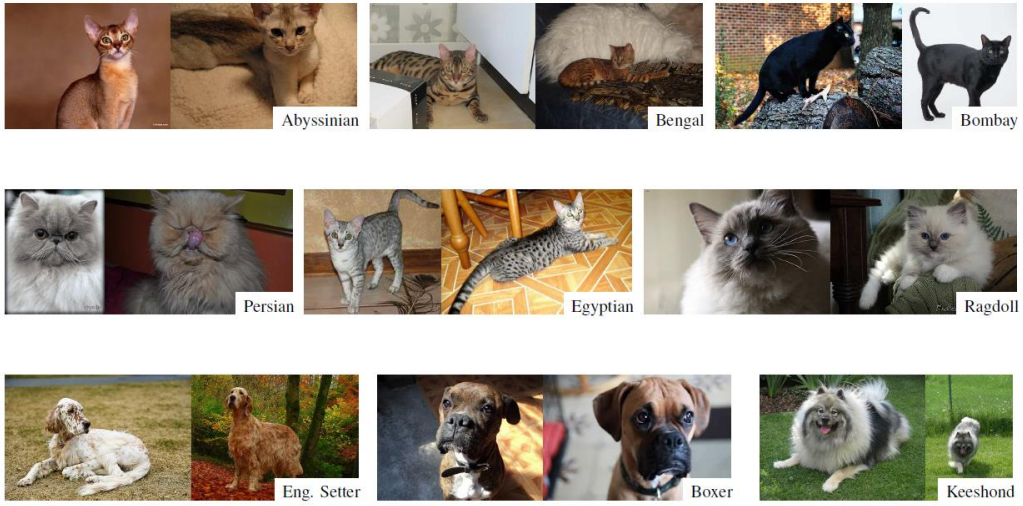
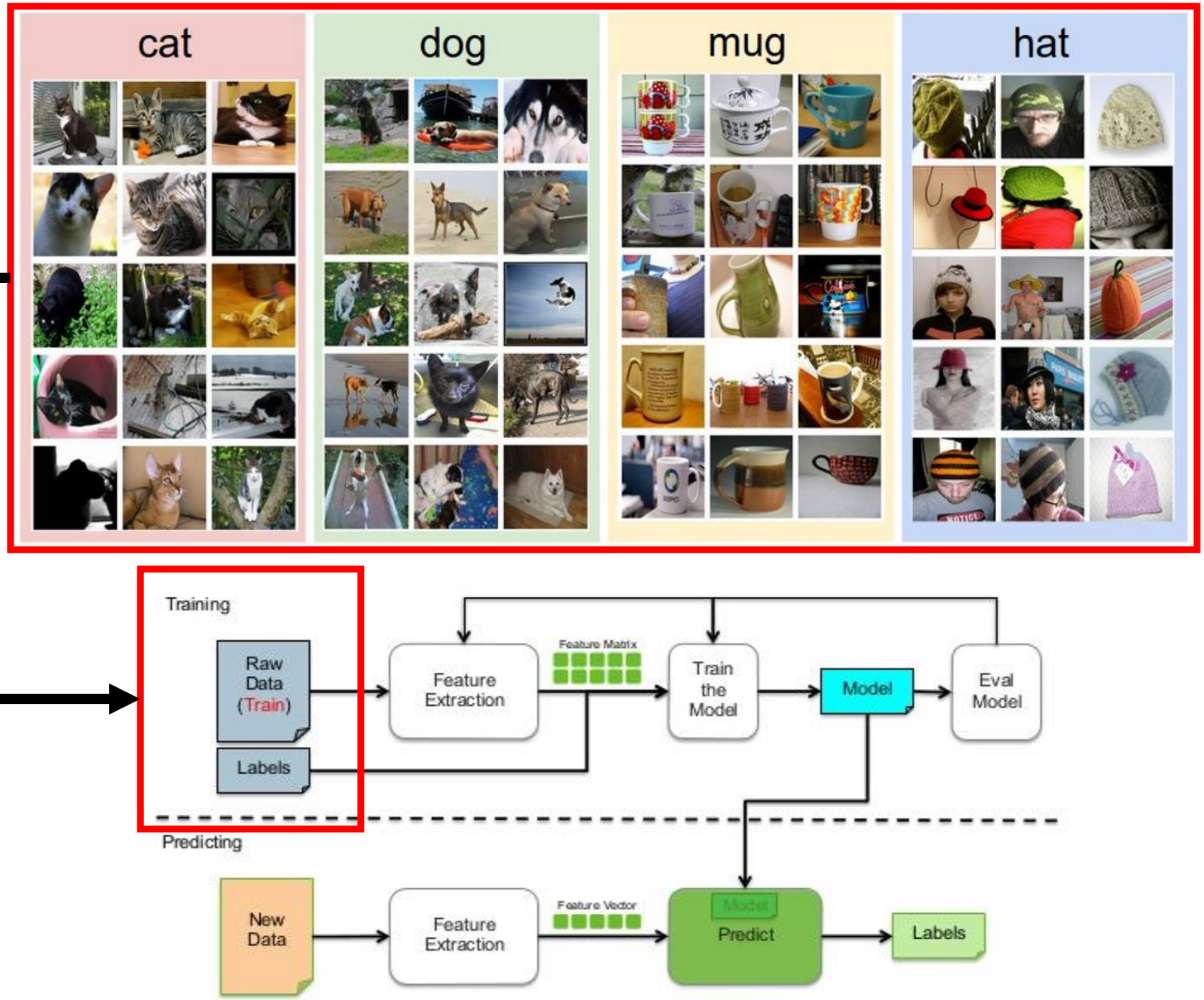
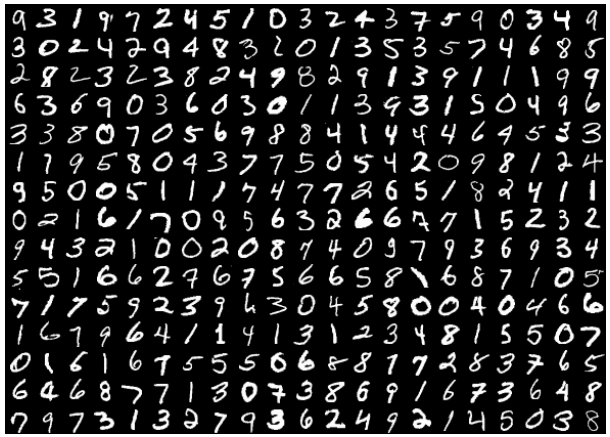


Image Classification Pipeline

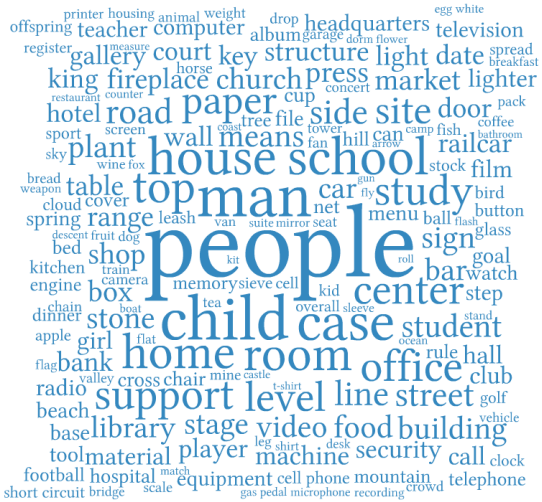


References: [81, 89]

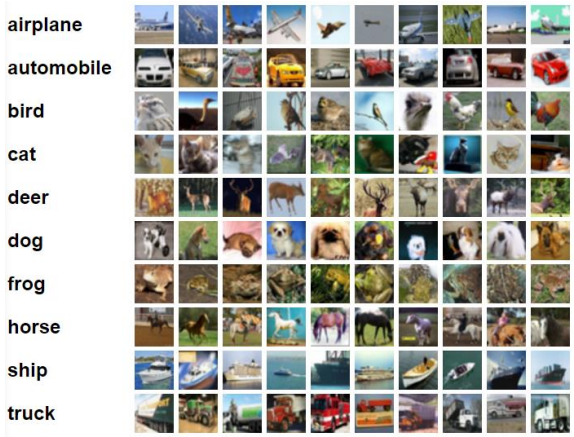
Famous Computer Vision Datasets



MNIST: handwritten digits



ImageNet: WordNet hierarchy

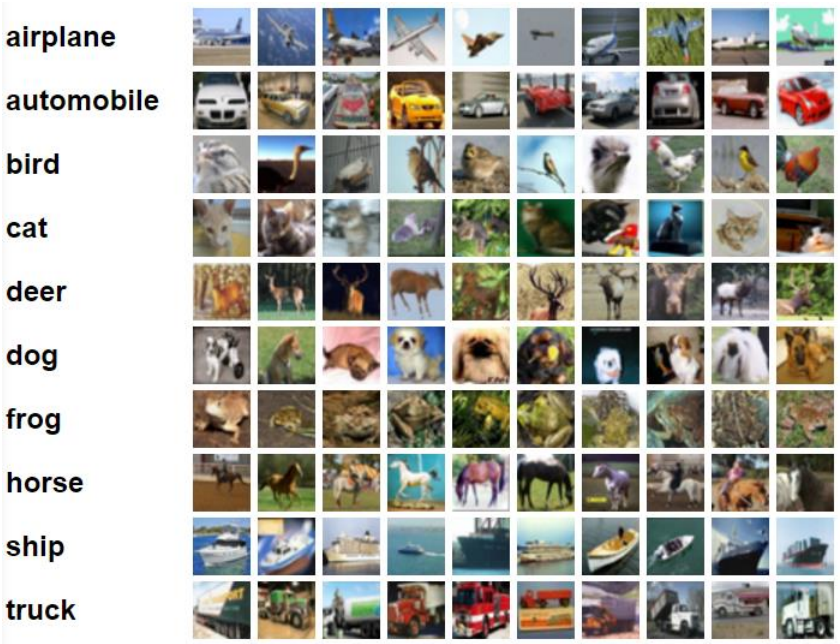


CIFAR-10(0): tiny images

Places: natural scenes

References: [90, 91, 92, 93]

Let's Build an Image Classifier for CIFAR-10



test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

→ 456

References: [89, 91]

Let's Build an Image Classifier for CIFAR-10

test image					training image					pixel-wise absolute value differences				
56	32	10	18	-	10	20	24	17	=	46	12	14	1	→ 456
90	23	128	133		8	10	89	100		82	13	39	33	
24	26	178	200		12	16	178	170		12	10	0	30	
2	0	255	220		4	32	233	112		2	32	22	108	

Accuracy

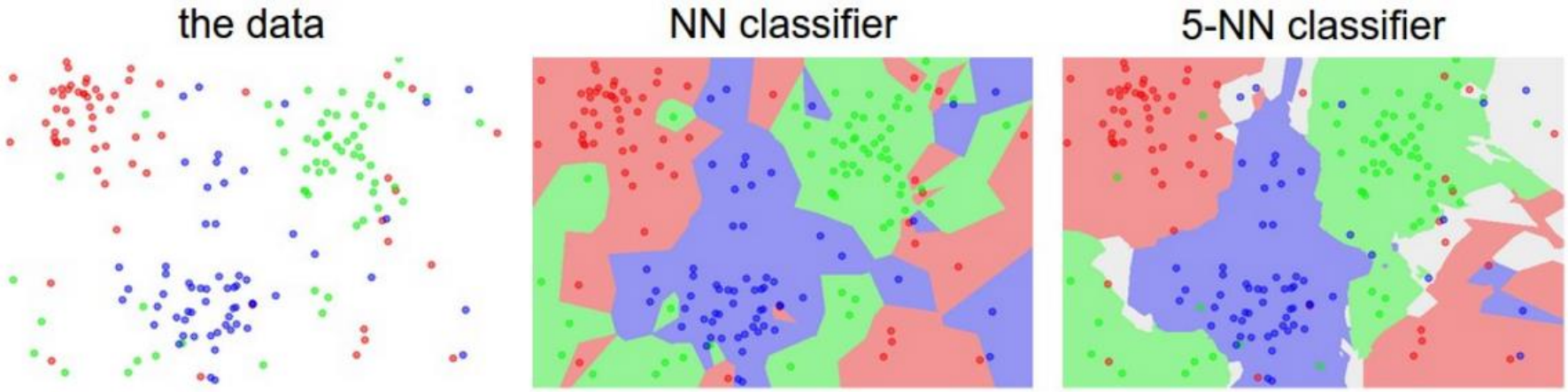
Random: **10%**

Our image-diff (with L1): **38.6%**

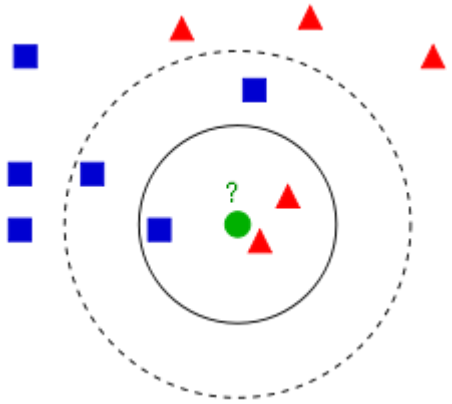
Our image-diff (with L2): **35.4%**

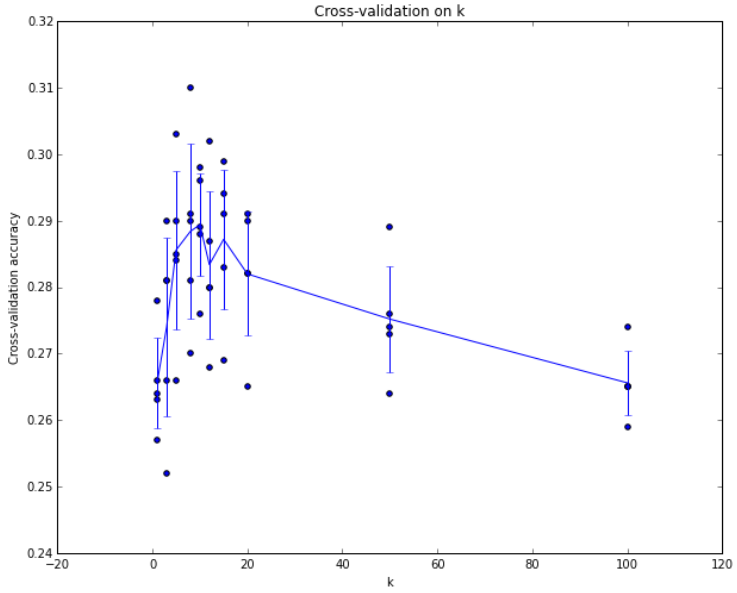
References: [89, 91]

K-Nearest Neighbors: Generalizing the Image-Diff Classifier

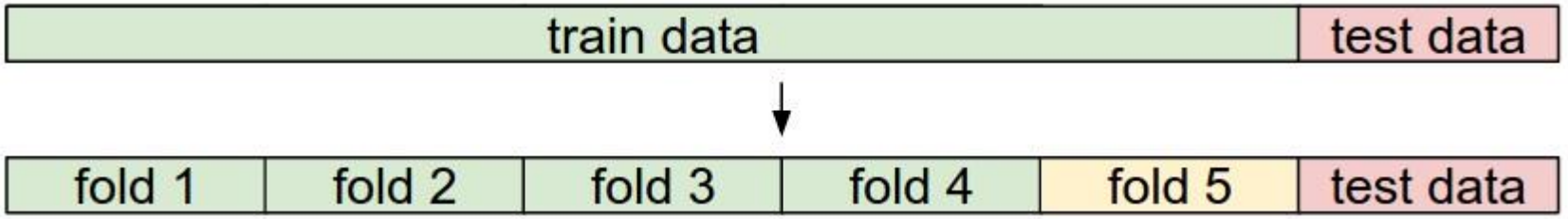


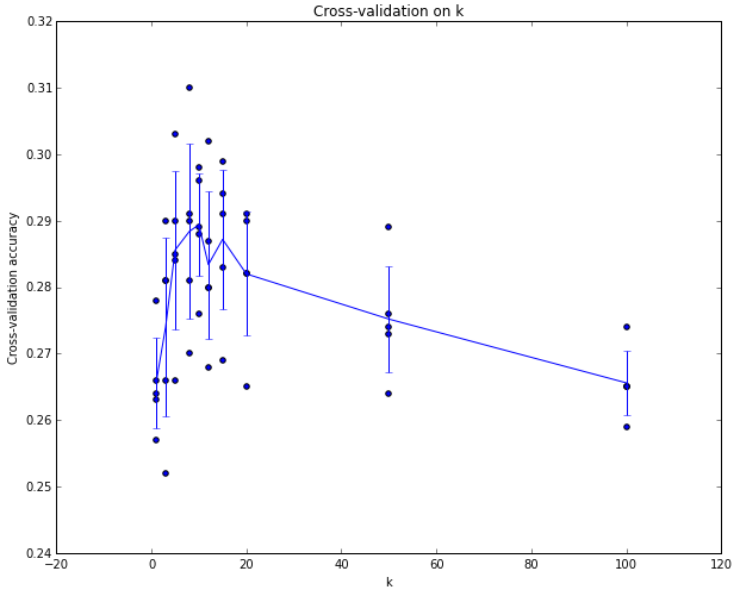
Tuning (hyper)parameters:





K-Nearest Neighbors: Generalizing the Image-Diff Classifier





Accuracy

Random: **10%**

Training and testing on the same data: **35.4%**

7-Nearest Neighbors: **~30%**

Human: **~94%**

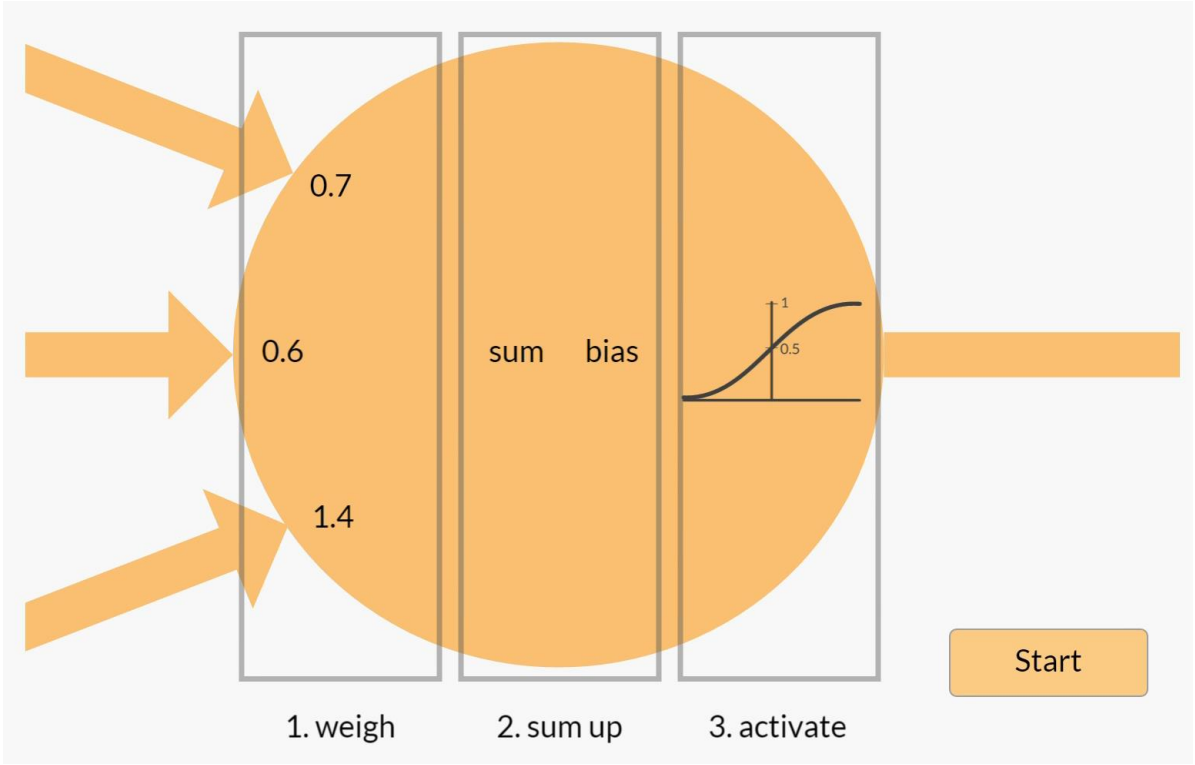
…

Convolutional Neural Networks: **~95%**

References: [89, 94]

Reminder: Weighing the Evidence

Evidence

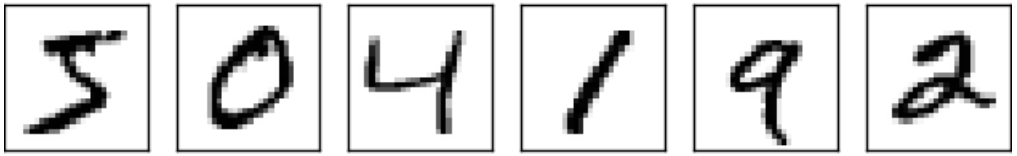


Decisions

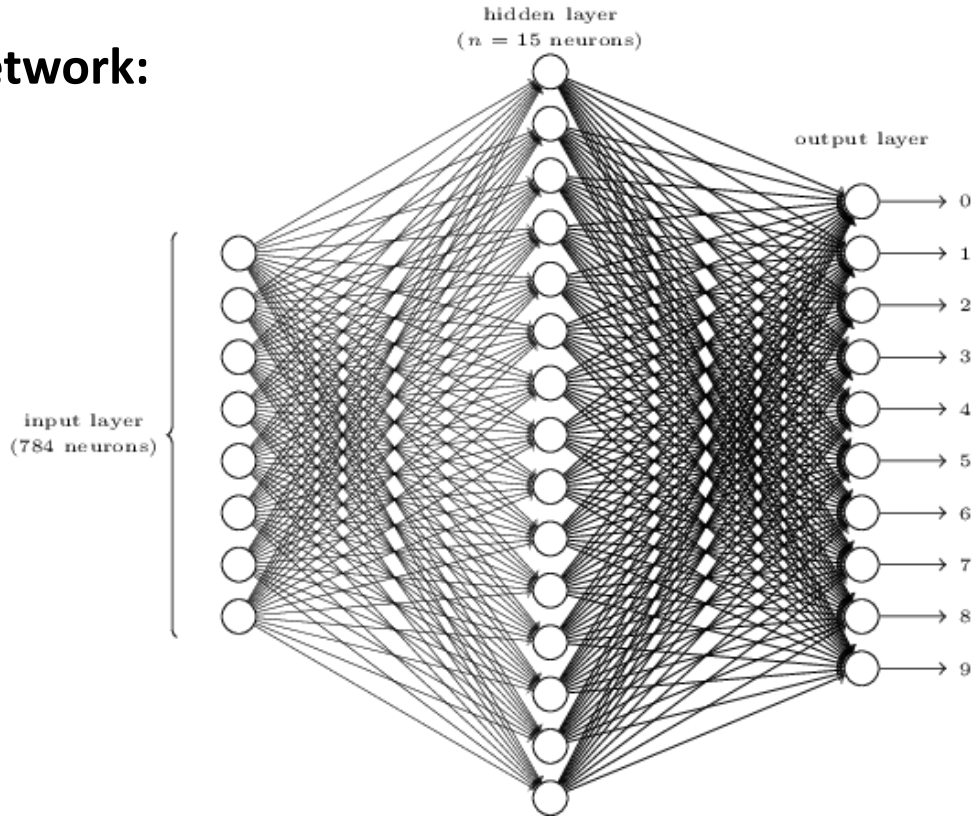
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Reminder: Classify and Image of a Number

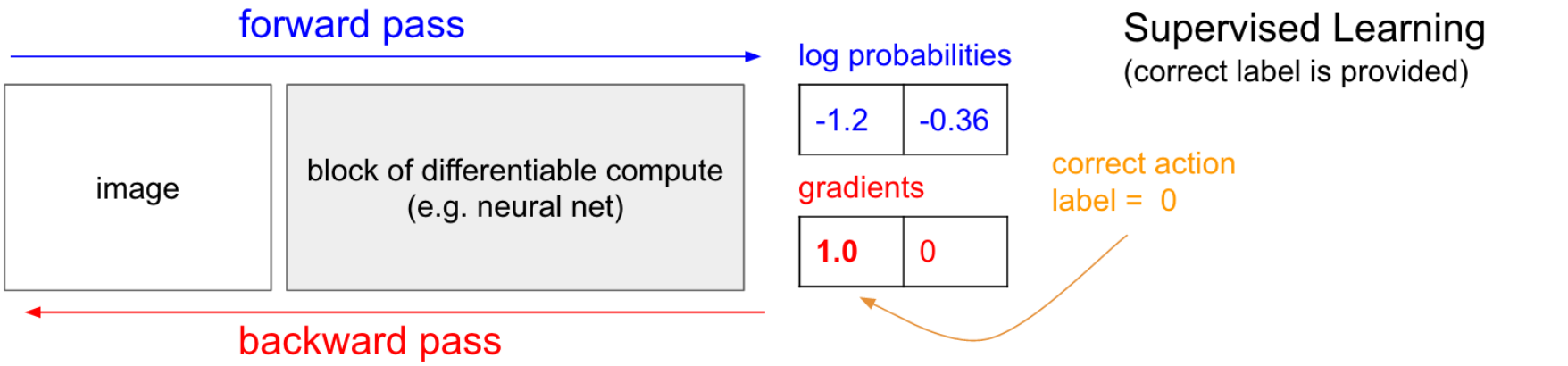
Input:
(28x28)



Network:



Reminder: "Learning" is Optimization of a Function

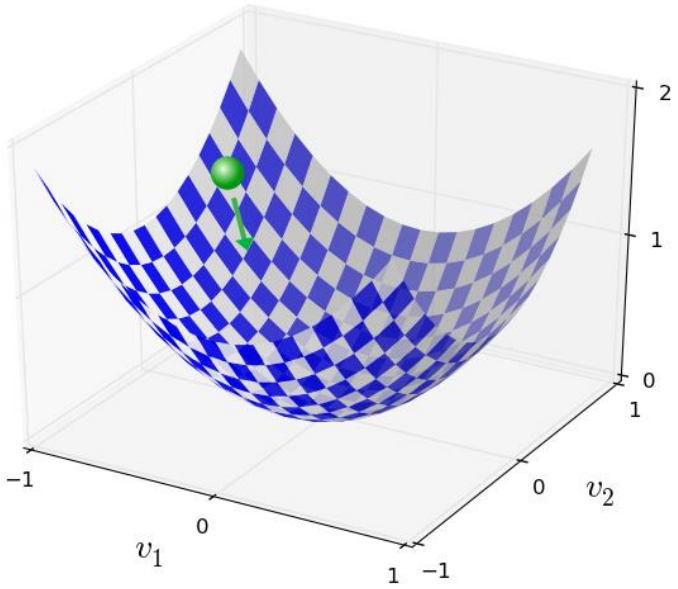


Ground truth for "6":

$$y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T$$

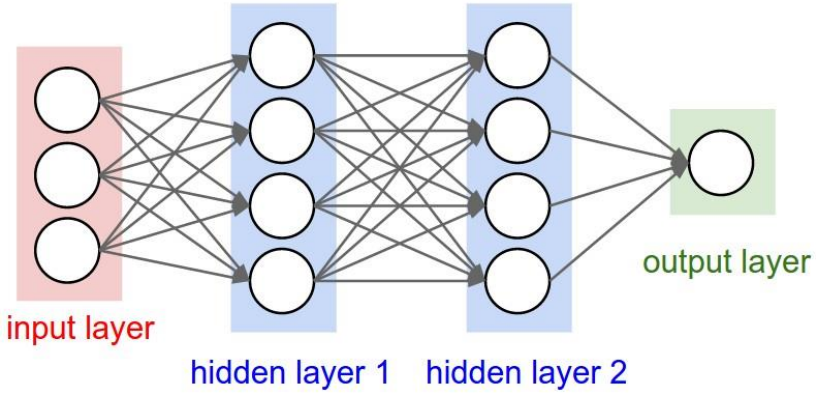
"Loss" function:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

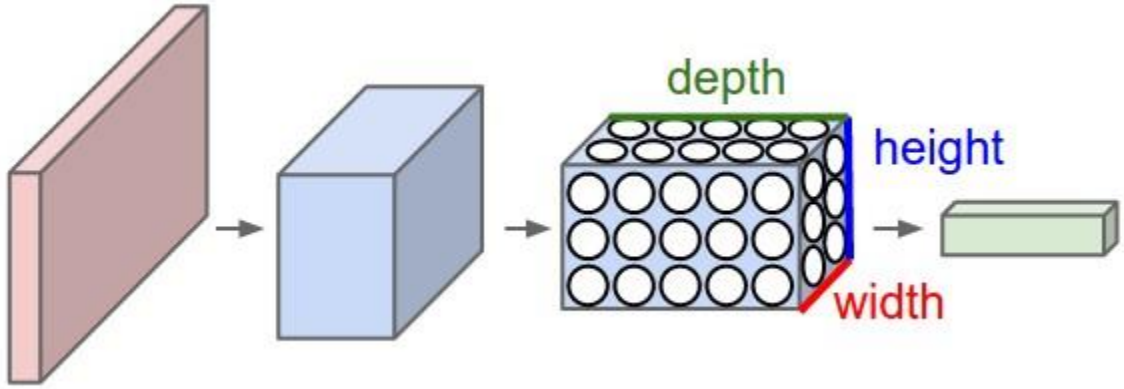


Convolutional Neural Networks

Regular neural network (fully connected):



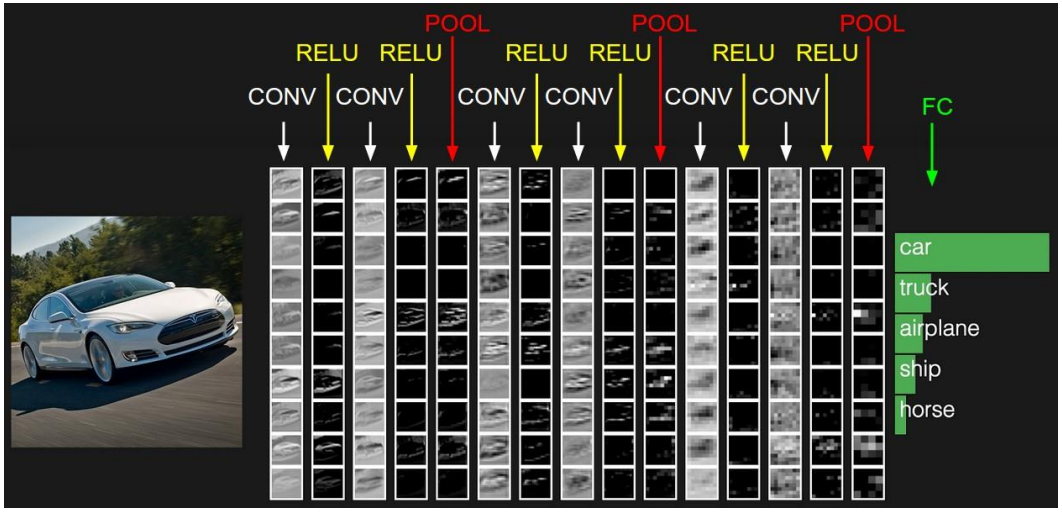
Convolutional neural network:



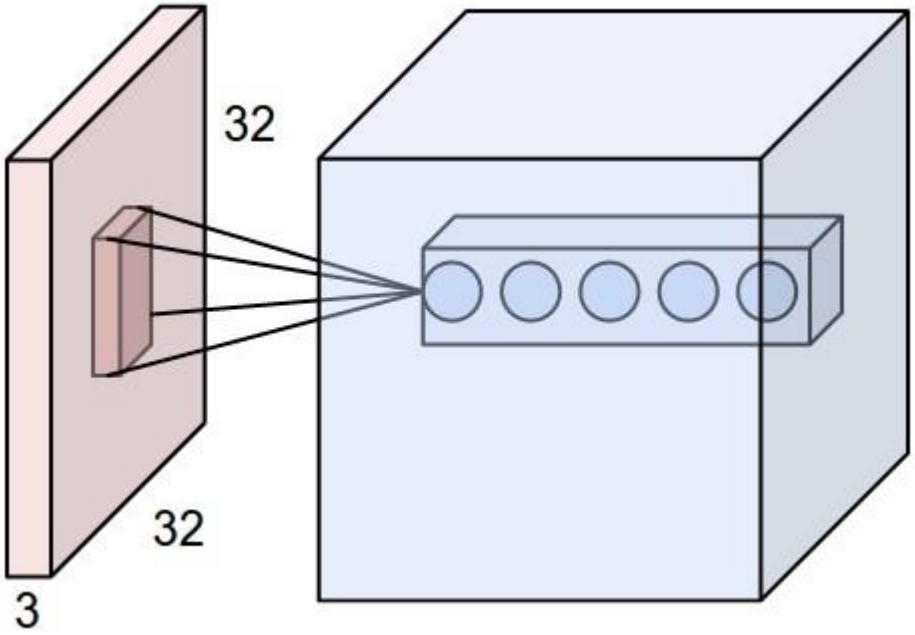
Each layer takes a 3d volume, produces 3d volume with some smooth function that may or may not have parameters.

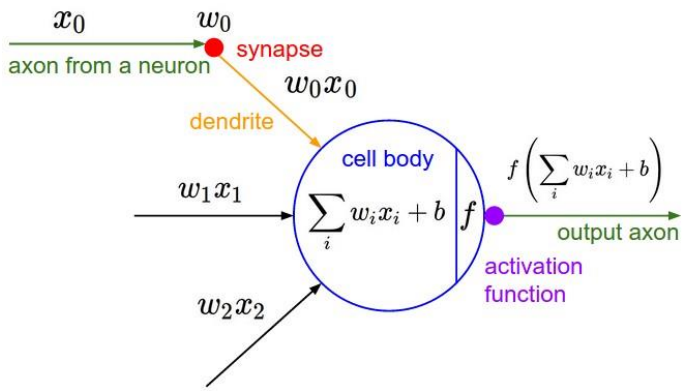
Convolutional Neural Networks: Layers

- **INPUT** [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- **CONV** layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- **RELU** layer will apply an elementwise activation function, such as the *max(0,x)* thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- **POOL** layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- **FC** (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.



Dealing with Images: Local Connectivity



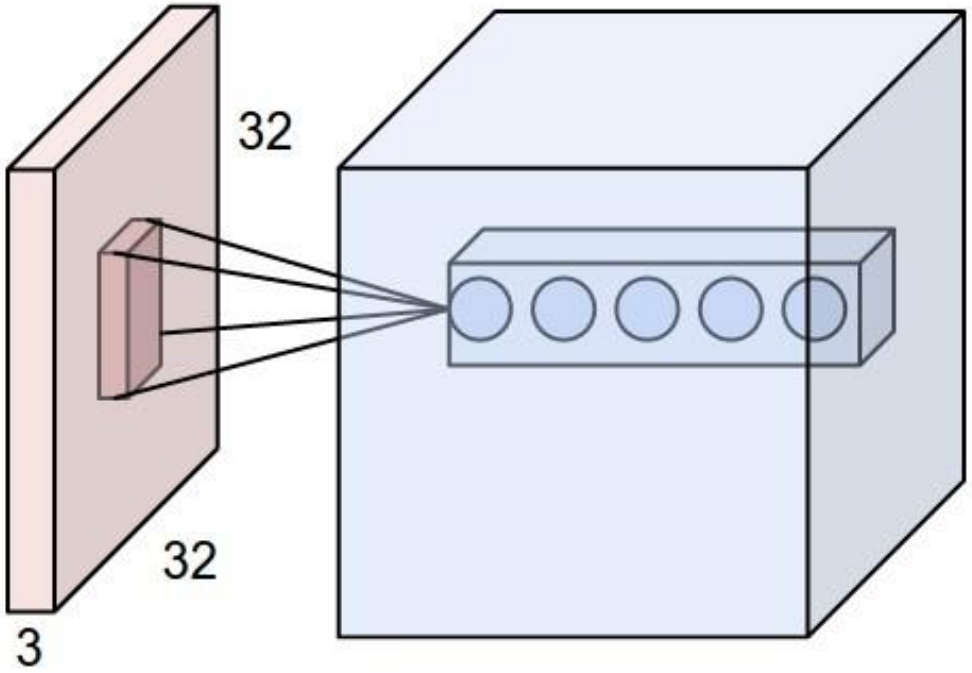


Same neuron. Just more focused (narrow “receptive field”).

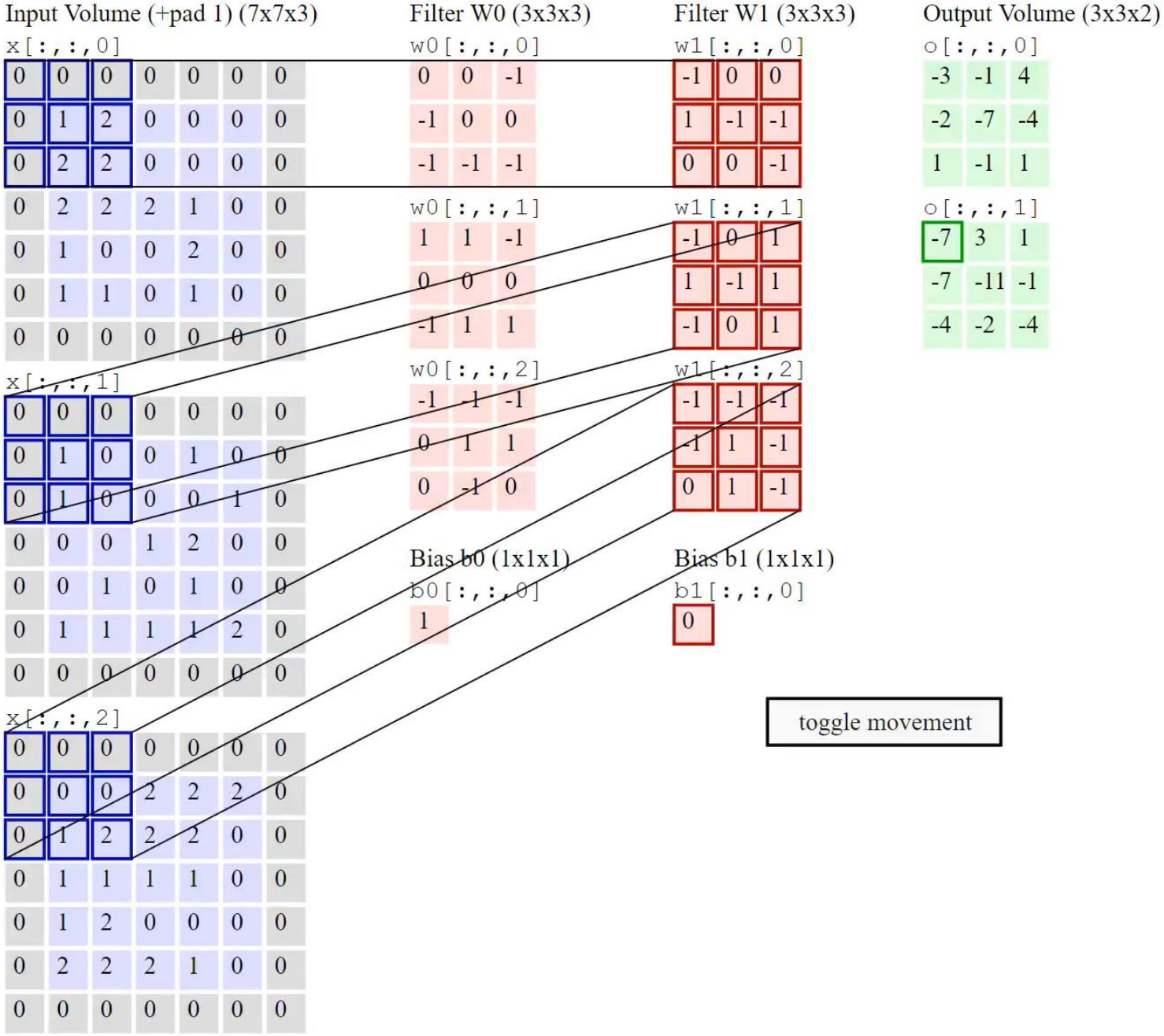
The parameters on a each filter are spatially “shared”

(if a feature is useful in one place, it’s useful elsewhere)

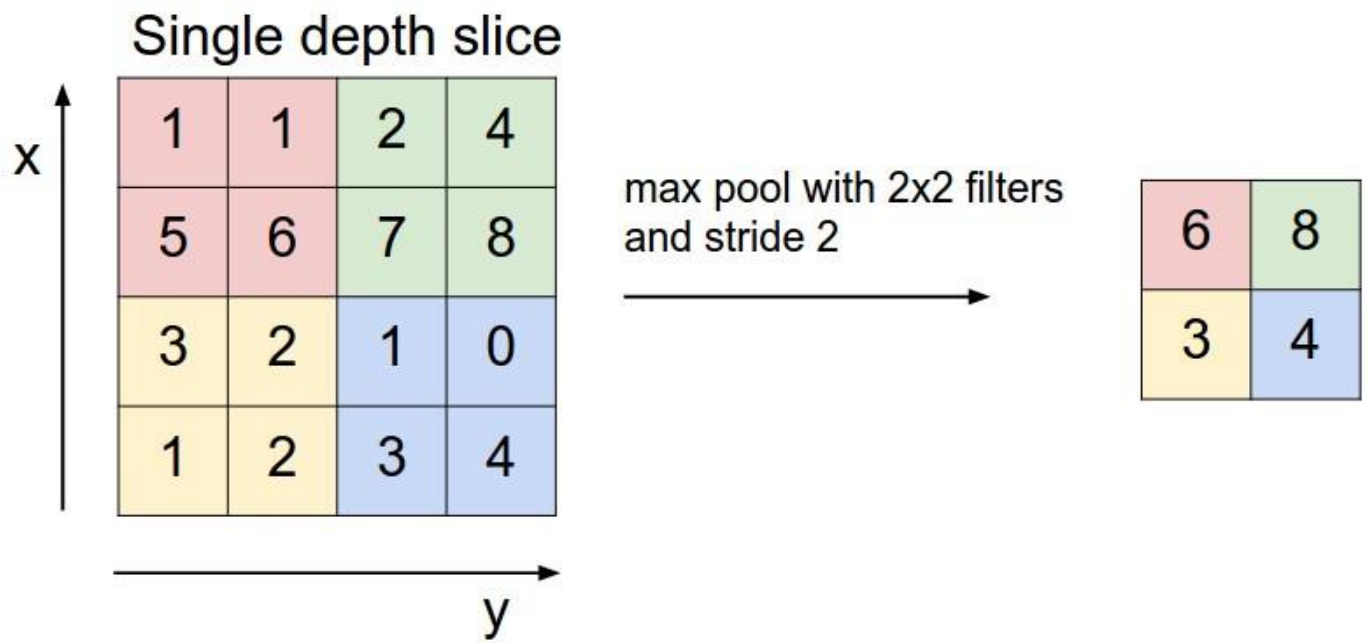
ConvNets: Spatial Arrangement of Output Volume

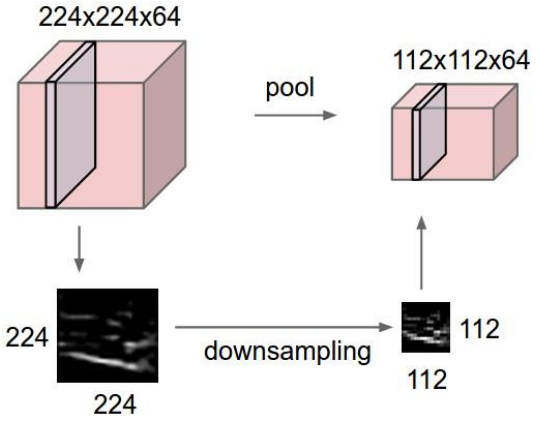


- **Depth:** number of filters
- **Stride:** filter step size (when we “slide” it)
- **Padding:** zero-pad the input

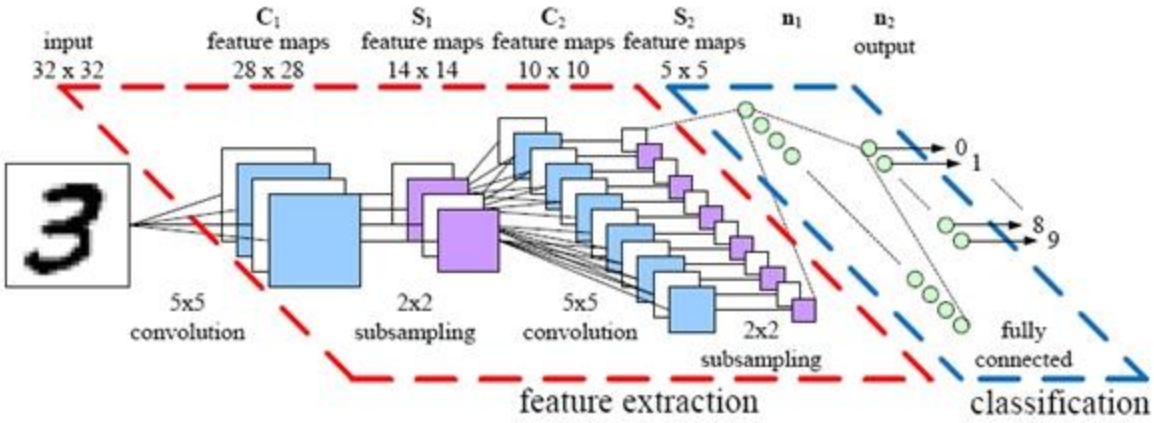

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4
Bias b0 (1x1x1)
b0[:,:,0]
1
Bias b1 (1x1x1)
b1[:,:,0]
0
toggle movement

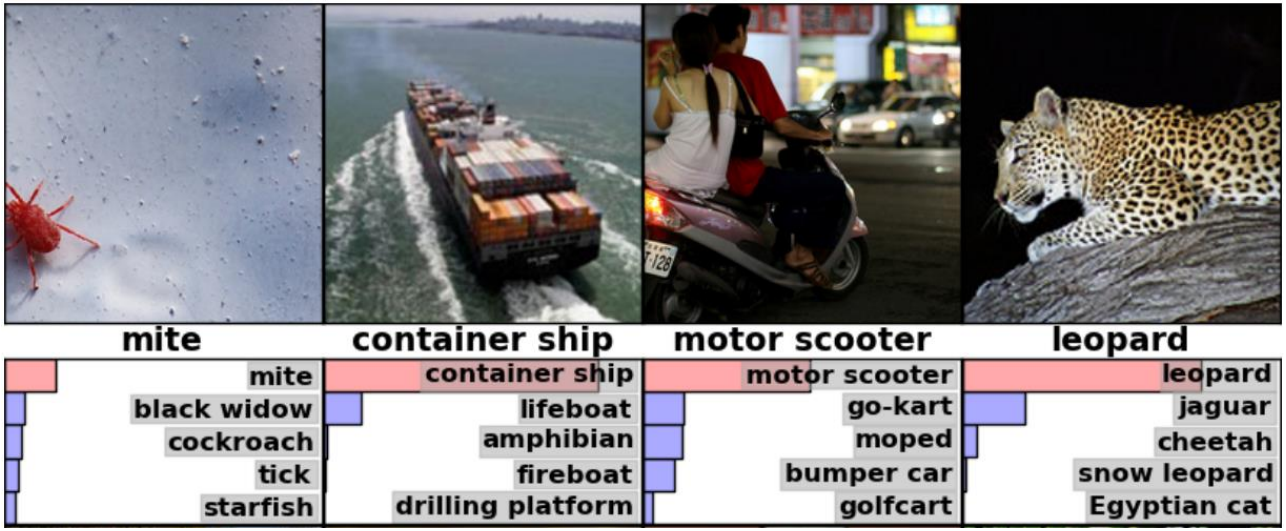
ConvNets: Pooling





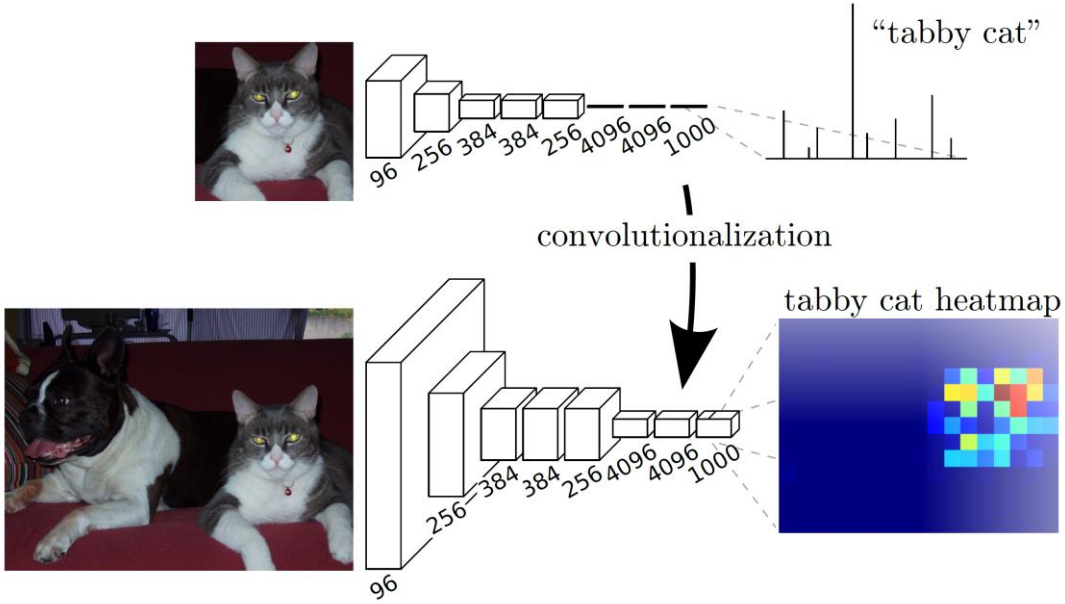
Computer Vision:
Object Recognition / Classification





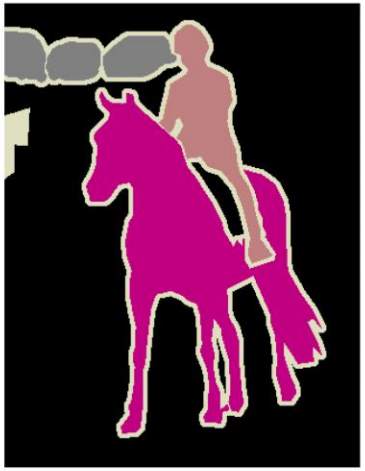
References: [4]

Computer Vision: Segmentation

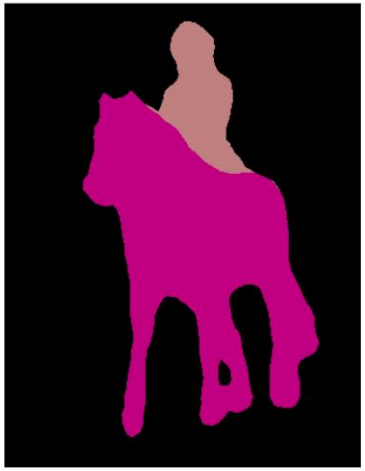


Original

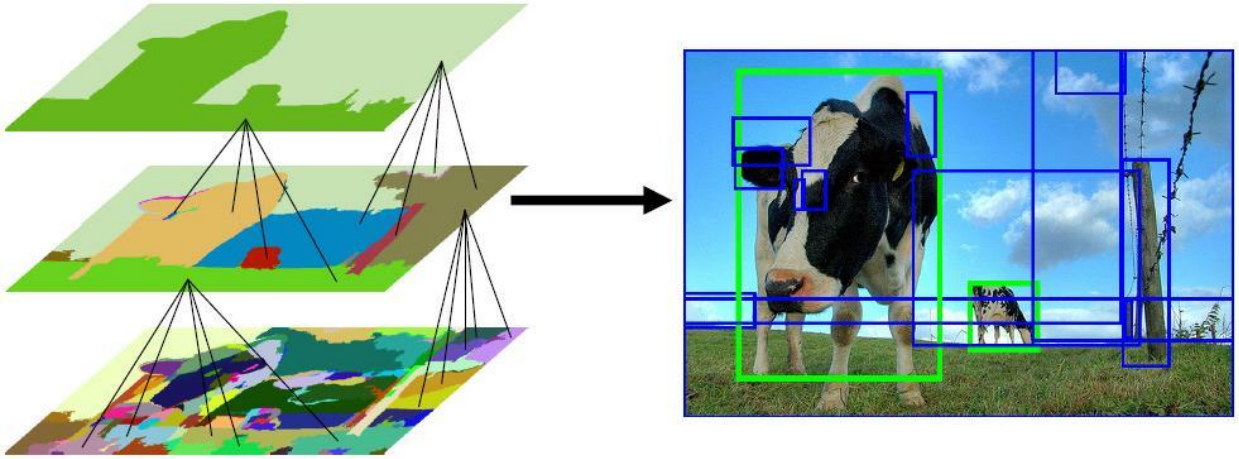
Ground Truth

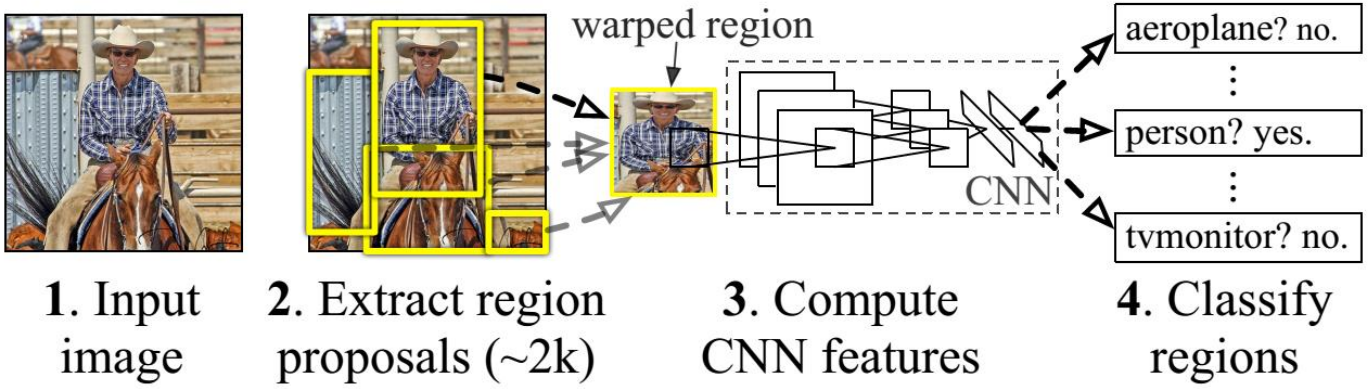


FCN-8



Computer Vision: Object Detection





How Can Convolutional Neural Networks Help Us Drive?

Driving: The Numbers

(in United States, in 2014)

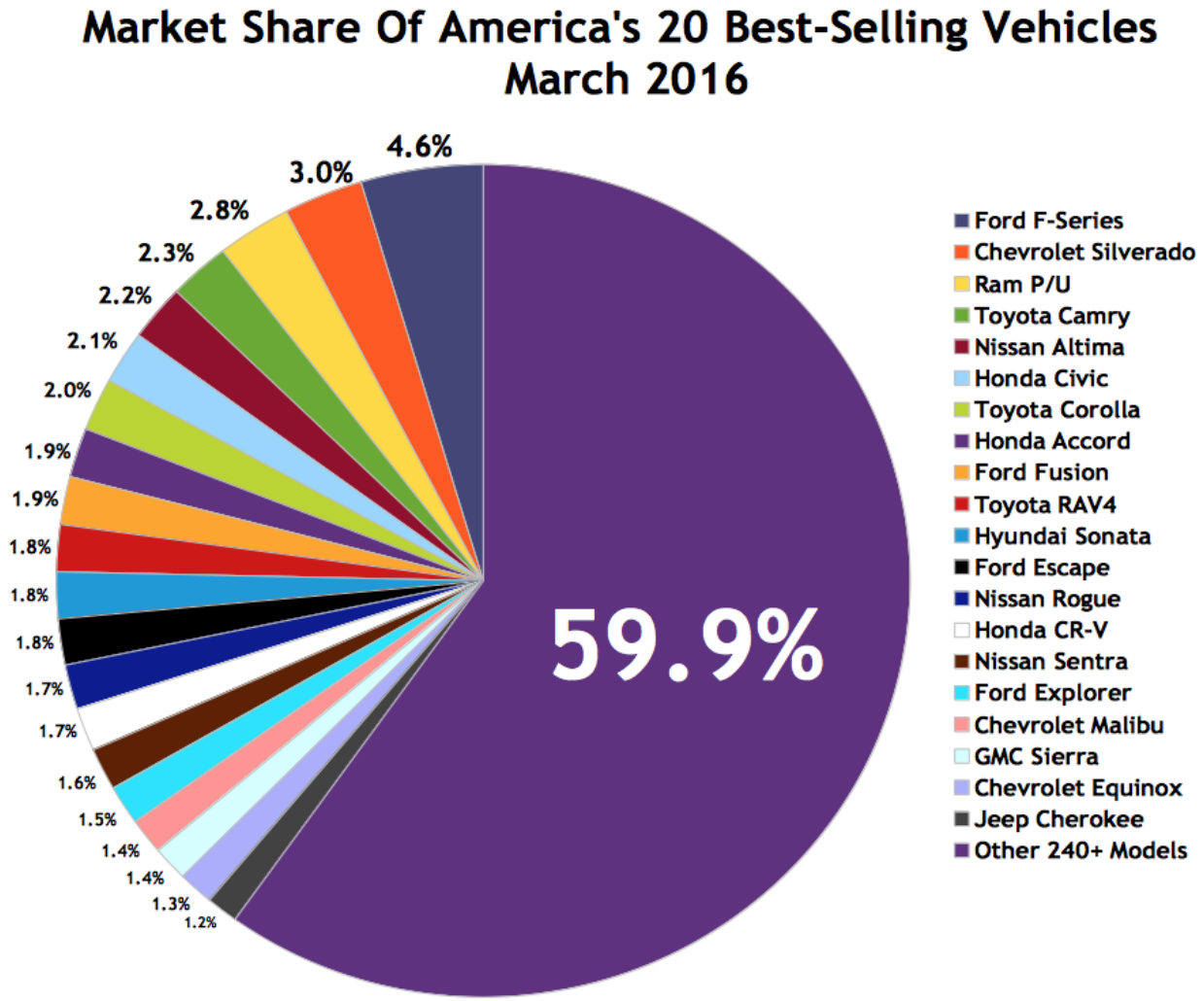
Miles

- **All drivers:** 10,658 miles (29.2 miles per day)
- **Rural drivers:** 12,264 miles
- **Urban drivers:** 9,709 miles

Fatalities

- **Fatal crashes:** 29,989
- **All fatalities:** 32,675
- **Car occupants:** 12,507
- **SUV occupants:** 8,320
- **Pedestrians:** 4,884
- **Motorcycle:** 4,295
- **Bicyclists:** 720
- **Large trucks:** 587

Cars We Drive



Human at the Center of Automation:
The Way to Full Autonomy Includes the Human

Fully Human Controlled

Fully Machine Controlled

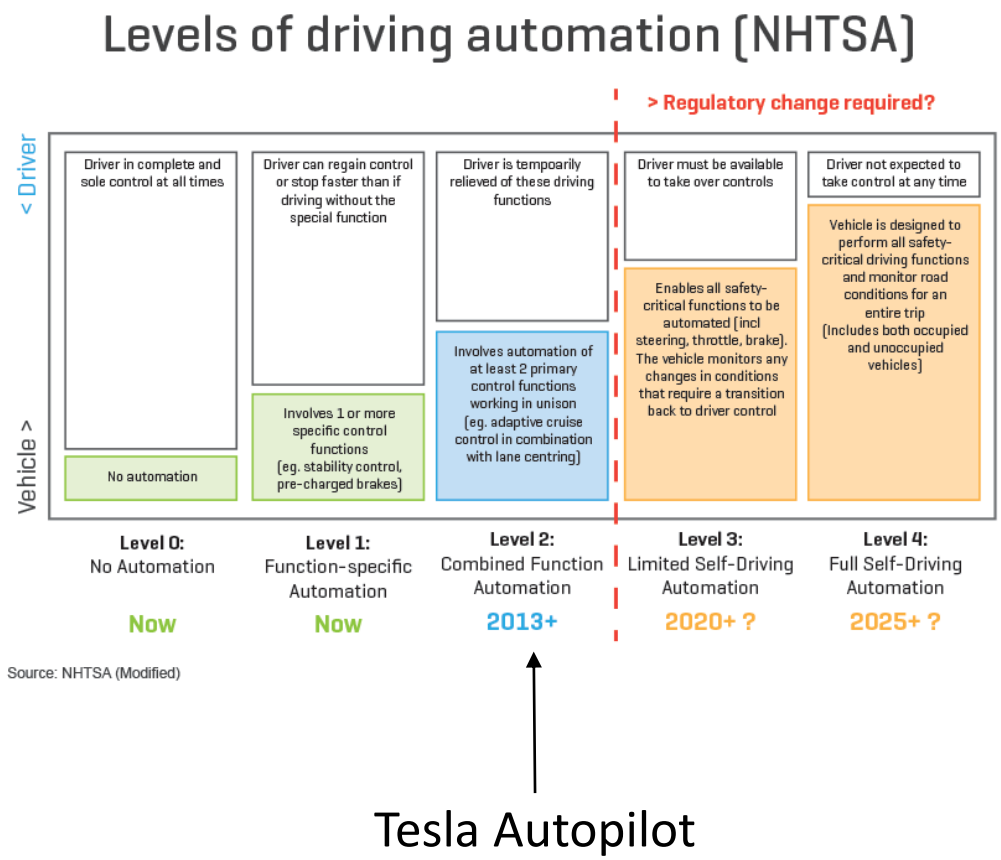
Ford F150

Tesla Model S

Google Self-Driving Car

Human at the Center of Automation:
The Way to Full Autonomy Includes the Human

- **Emergency**
  - Automatic emergency breaking (AEB)
- **Warnings**
  - Lane departure warning (LDW)
  - Forward collision warning (FCW)
  - Blind spot detection
- **Longitudinal**
  - Adaptive cruise control (ACC)
- **Lateral**
  - Lane keep assist (LKA)
  - Automatic steering
- **Control and Planning**
  - Automatic lane change
  - Automatic parking



Tesla Autopilot

Distracted Humans

What is distracted driving?

- Texting
- Using a smartphone
- Eating and drinking
- Talking to passengers
- Grooming
- Reading, including maps
- Using a navigation system
- Watching a video
- Adjusting a radio

- **Injuries and fatalities:**
  3,179 people were killed and 431,000 were injured in motor vehicle crashes involving distracted drivers
  (in 2014)
- **Texts:**
  169.3 billion text messages were sent in the US every month.
  (as of December 2014)
- **Eye off road:**
  5 seconds is the average time your eyes are off the road while texting. When traveling at 55mph, that's enough time to cover the length of a football field blindfolded.

4 D's of Being Human:
Drunk, Drugged, Distracted, Drowsy

- **Drunk Driving:** In 2014, 31 percent of traffic fatalities involved a drunk driver.
- **Drugged Driving:** 23% of night-time drivers tested positive for illegal, prescription or over-the-counter medications.
- **Distracted Driving:** In 2014, 3,179 people (10 percent of overall traffic fatalities) were killed in crashes involving distracted drivers.
- **Drowsy Driving:** In 2014, nearly three percent of all traffic fatalities involved a drowsy driver, and at least 846 people were killed in crashes involving a drowsy driver.

In Context: Traffic Fatalities

Total miles driven in U.S. in 2014:

3,000,000,000,000 (3 million million)

Tesla Autopilot miles driven since October 2015:

300,000,000 (300 million)

(as of December 2016)

In Context: Traffic Fatalities

Total miles driven in U.S. in 2014:

3,000,000,000,000 (3 million million)

Fatalities: **32,675**

(1 in 90 million)

Tesla Autopilot miles driven since October 2015:

300,000,000 (300 million)

Fatalities: **1**

In Context: Traffic Fatalities

Total miles driven in U.S. in 2014:

3,000,000,000,000 (3 million million)

We (increasingly) understand this → Fatalities: **32,675**

(1 in 90 million)

Tesla Autopilot miles driven since October 2015:

300,000,000 (300 million)

We do not understand this (yet) → Fatalities: **1**

We need A LOT of real-world semi-autonomous driving data!

Computer Vision + Machine Learning + Big Data = Understanding

The Data

Teslas instrumented: 17

Hours of data: 5,000+ hours

Distance traveled: 70,000+ miles

The Data


Total Time Driving: 0 mins
Autopilot Available: 0 mins
Autopilot Engaged: 0 mins

Camera and Lens Selection

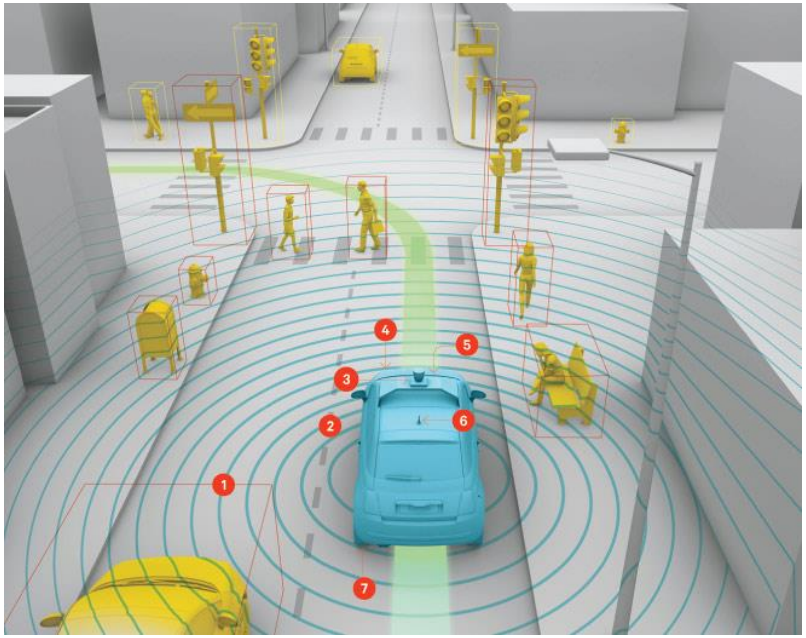
Logitech C920:
On-board H264 Compression

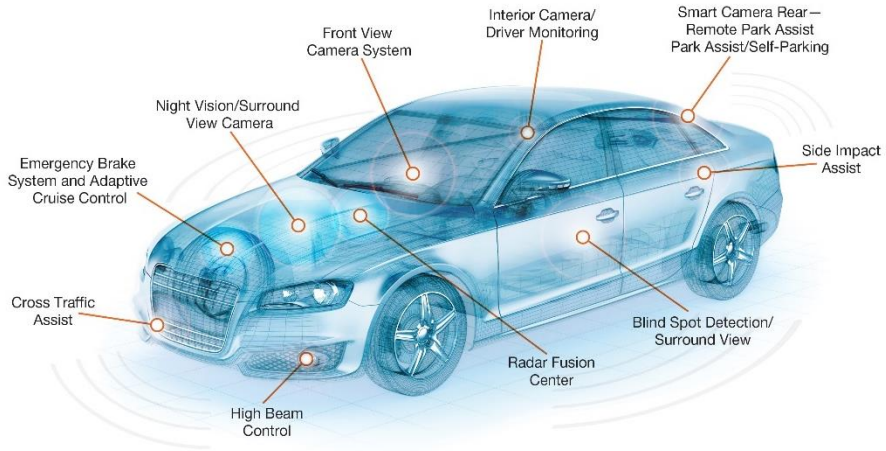
Fisheye: Capture full range of head, body movement inside vehicle.

Case for C-Mount Lens:
Flexibility in lens selection

2.8-12mm Focal Length: "Zoom" on the face without obstructing the driver's view.

Semi-Autonomous Vehicle Components





External

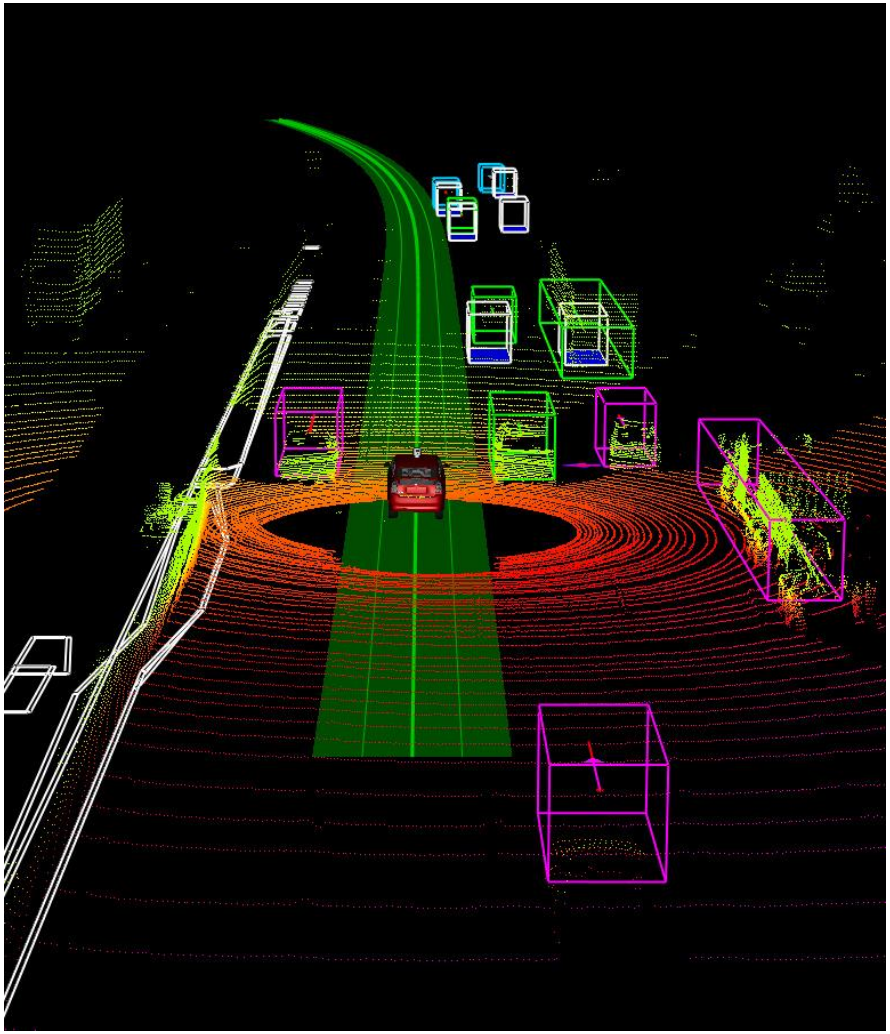
1. Radar
2. Visible-light camera
3. LIDAR
4. Infrared camera
5. Stereo vision
6. GPS/IMU
7. CAN
8. Audio

Internal

1. Visible-light camera
2. Infrared camera
3. Audio

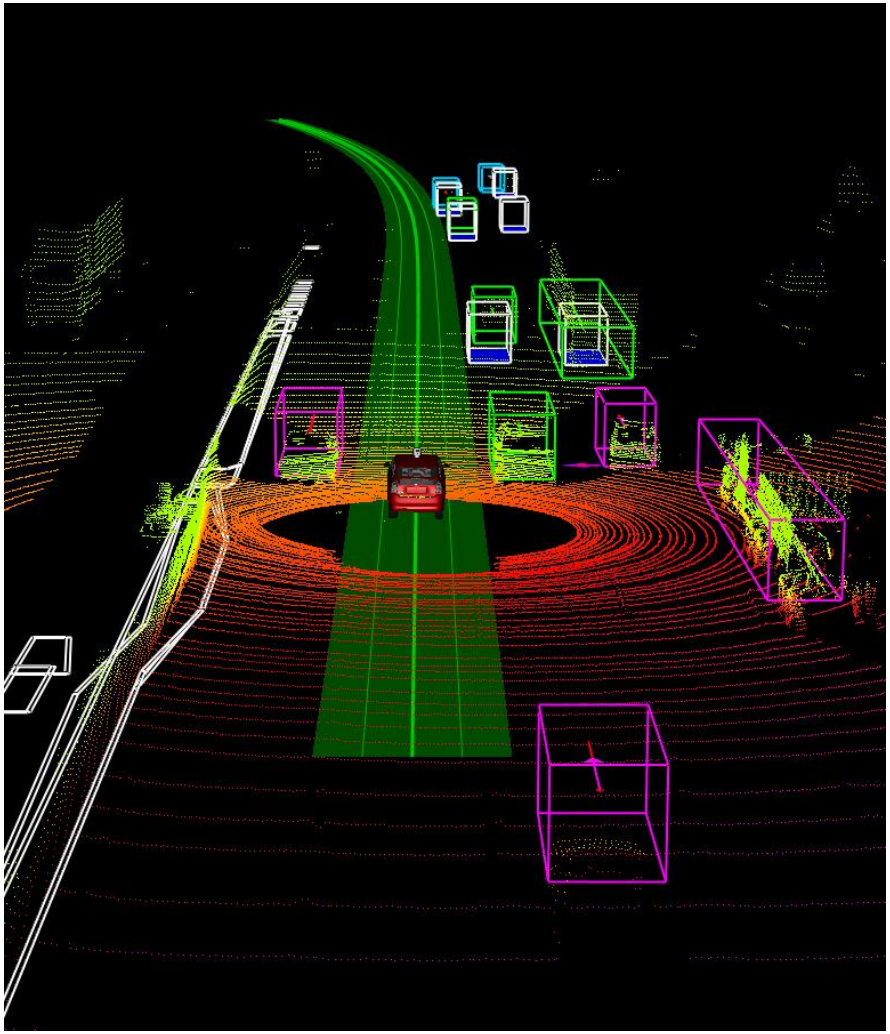
Self-Driving Car Tasks

- **Localization and Mapping:**
  Where am I?
- **Scene Understanding:**
  Where is everyone else?
- **Movement Planning:**
  How do I get from A to B?
- **Driver State:**
  What's the driver up to?



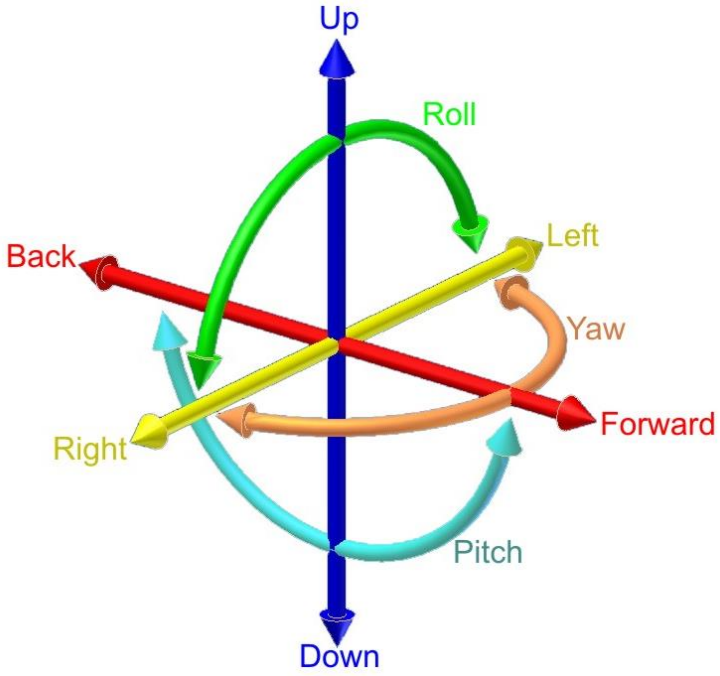
Self-Driving Car Tasks

- **Localization and Mapping:**
  Where am I?
- **Scene Understanding:**
  Where is everyone else?
- **Movement Planning:**
  How do I get from A to B?
- **Driver State:**
  What's the driver up to?



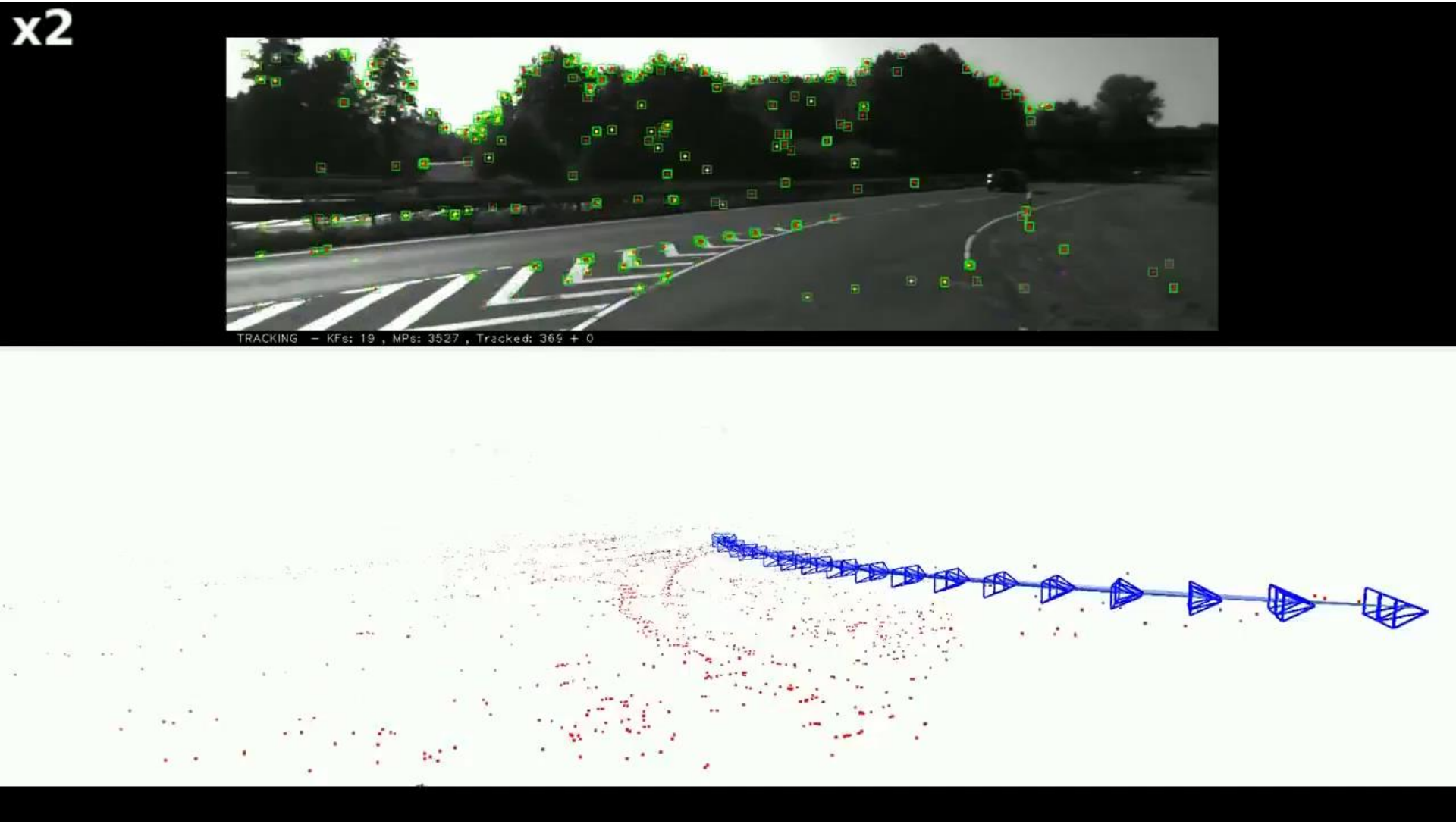
Visual Odometry

- 6-DOF: freed of movement
  - Changes in position:
    - Forward/backward: surge
    - Left/right: sway
    - Up/down: heave
  - Orientation:
    - Pitch, Yaw, Roll
- Source:
  - **Monocular:** I moved 1 unit
  - **Stereo:** I moved 1 meter
  - Mono = Stereo for far away objects
    - PS: For tiny robots everything is “far away” relative to inter-camera distance



SLAM: Simultaneous Localization and Mapping

What works: SIFT and optical flow



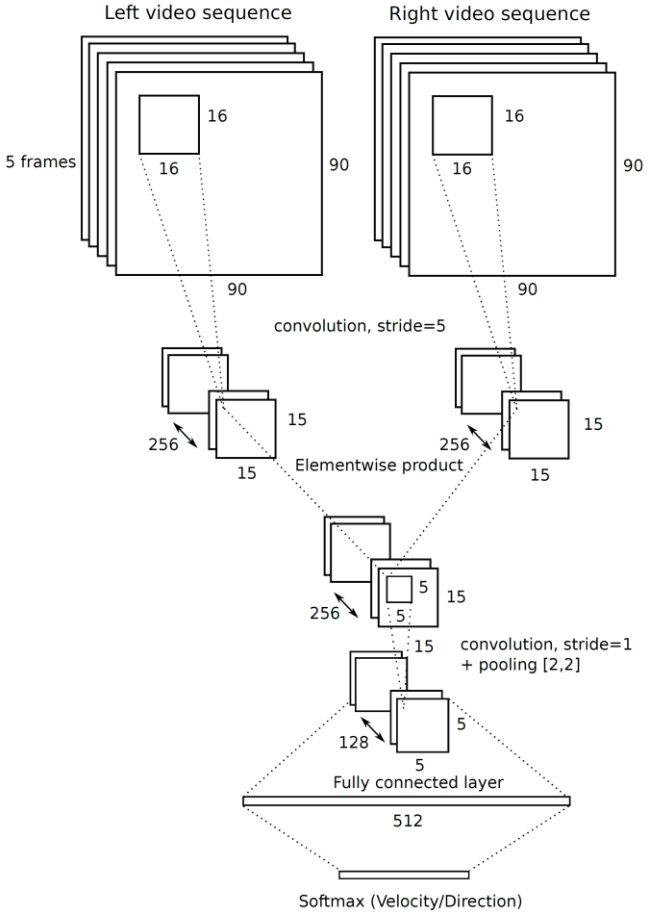
References: [98, 99]

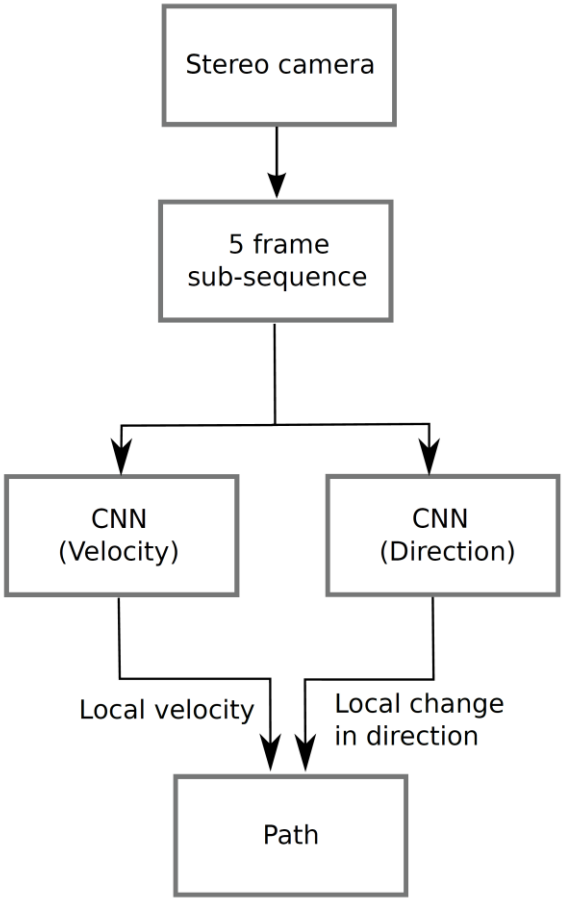
Visual Odometry in Parts

- (Stereo) Undistortion, Rectification
- (Stereo) Disparity Map Computation
- Feature Detection (e.g., SIFT, FAST)
- Feature Tracking (e.g., KLT: Kanade-Lucas-Tomasi)
- Trajectory Estimation
  - Use rigid parts of the scene (requires outlier/inlier detection)
  - For mono, need more info* like camera orientation and height of off the ground

* Kitt, Bernd Manfred, et al. "Monocular visual odometry using a planar road model to solve scale ambiguity." (2011).

End-to-End Visual Odometry

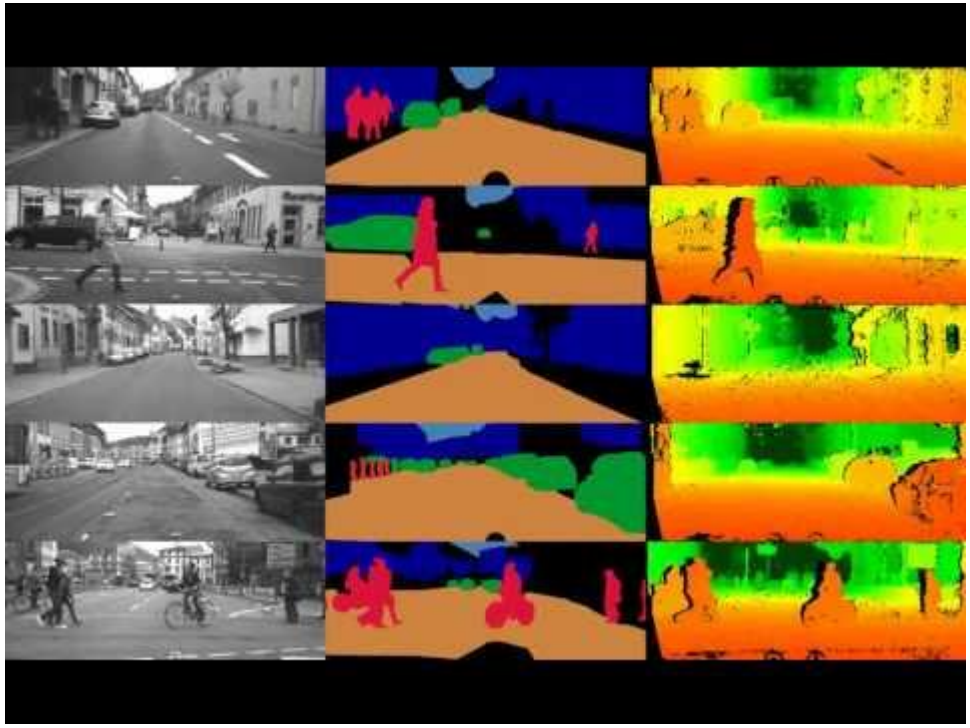




Konda, Kishore, and Roland Memisevic. "Learning visual odometry with a convolutional network." *International Conference on Computer Vision Theory and Applications*. 2015.

Self-Driving Car Tasks

- **Localization and Mapping:**
  Where am I?
- **Scene Understanding:**
  Where is everyone else?
- **Movement Planning:**
  How do I get from A to B?
- **Driver State:**
  What's the driver up to?



Object Detection

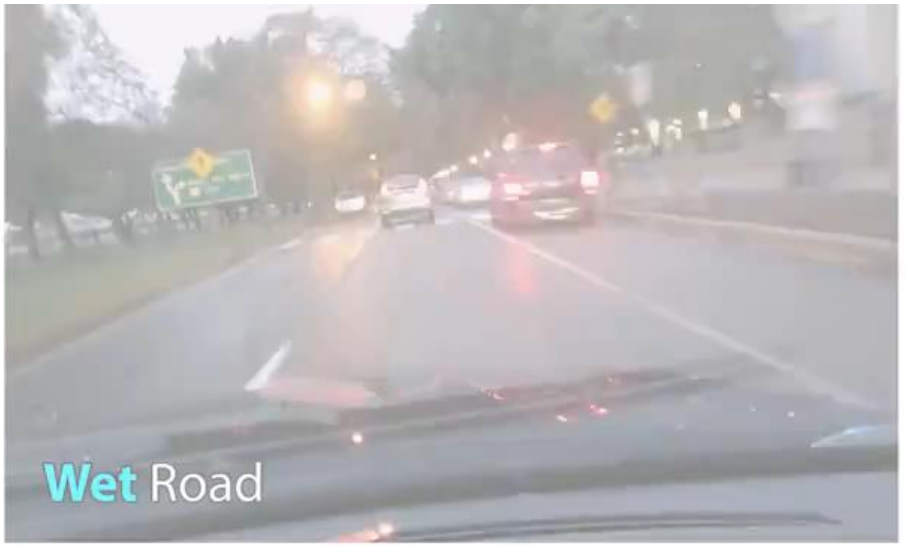
- Past approaches: cascades classifiers (Haar-like features)
- Where deep learning can help:
  recognition, classification, detection

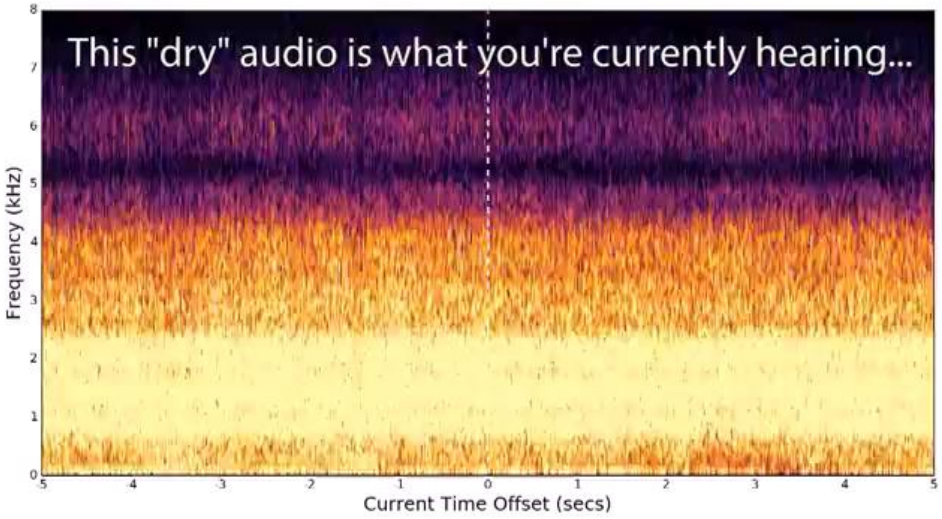
Full Driving Scene Segmentation

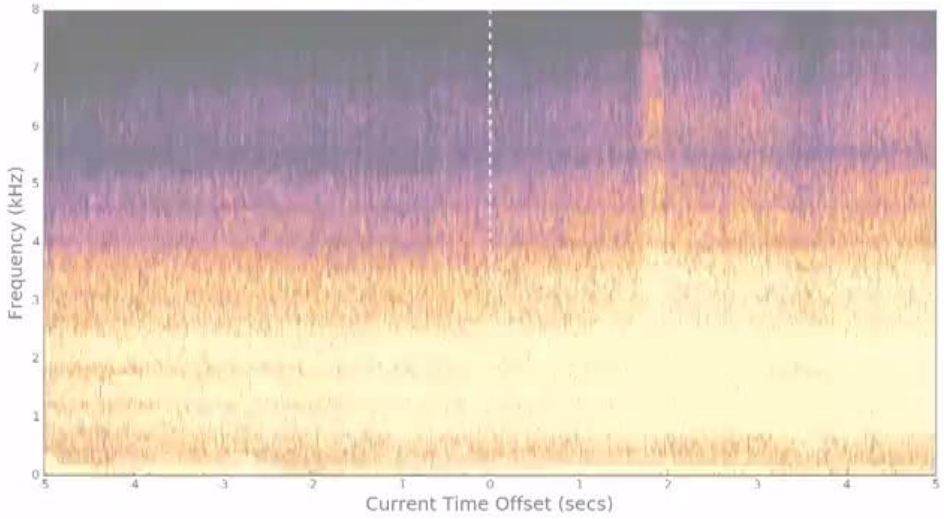
Fully Convolutional Network implementation:
https://github.com/tkuanlun350/Tensorflow-SegNet

Road Texture and Condition from Audio

(with Recurrent Neural Networks)

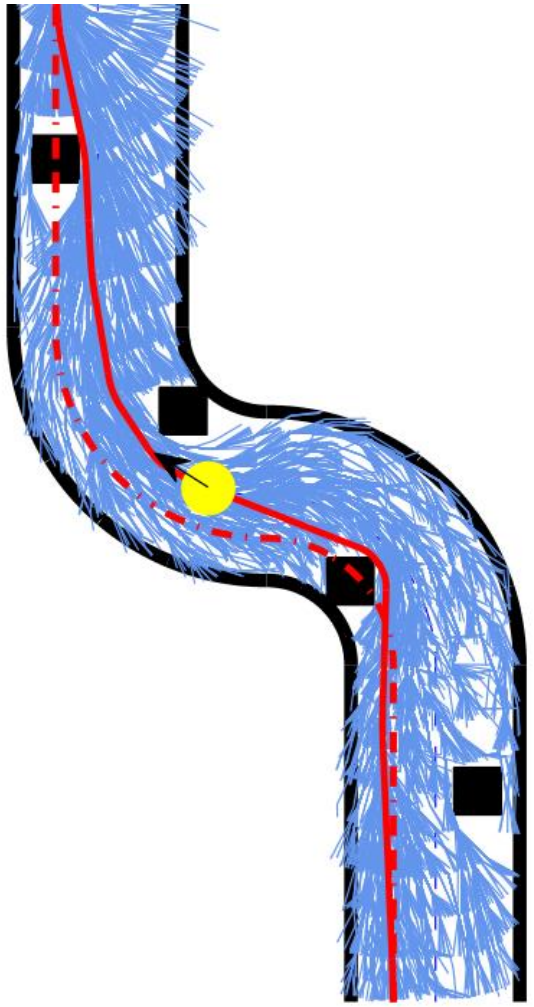




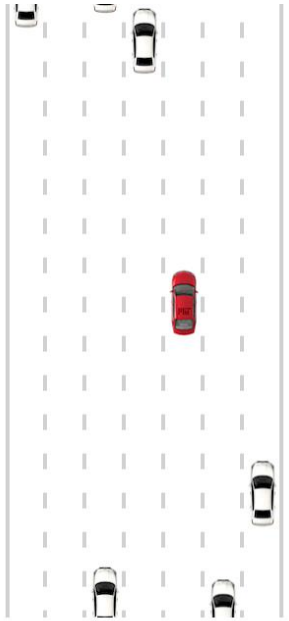


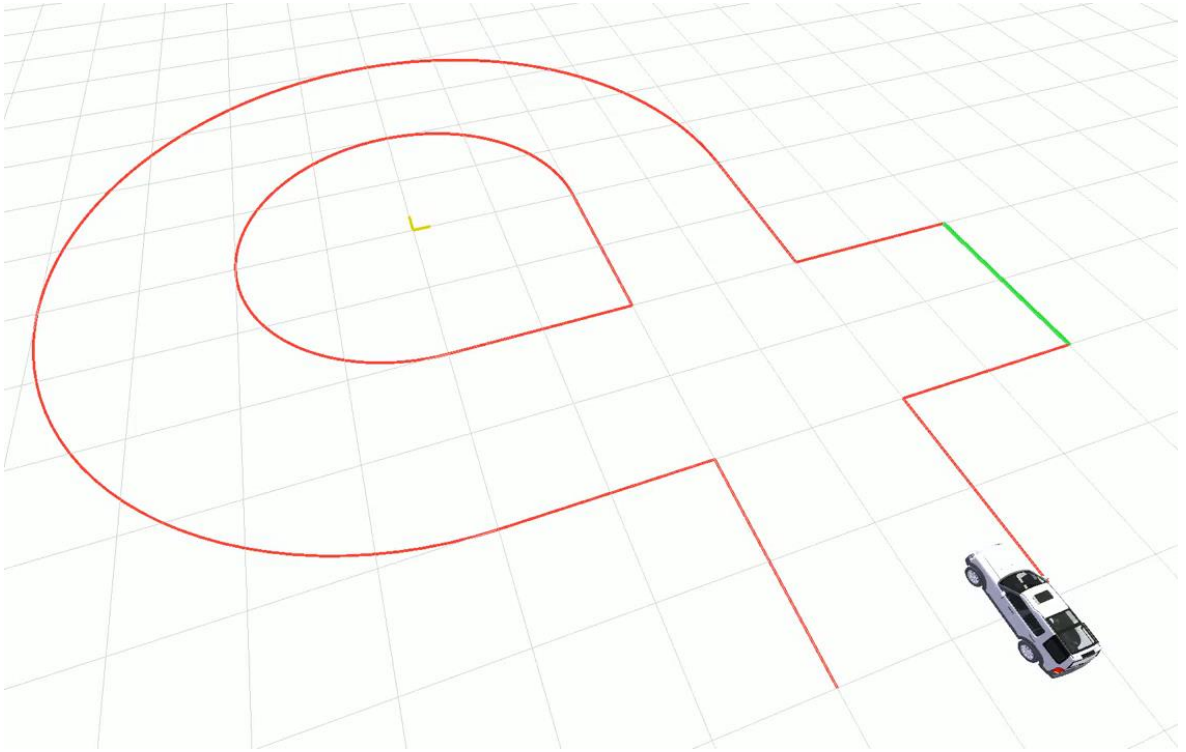
Self-Driving Car Tasks

- **Localization and Mapping:**
  Where am I?
- **Scene Understanding:**
  Where is everyone else?
- **Movement Planning:**
  How do I get from A to B?
- **Driver State:**
  What's the driver up to?



- Previous approaches: optimization-based control
- Where deep learning can help: reinforcement learning





Deep Reinforcement Learning implementation:
https://github.com/nivwusquorum/tensorflow-deepq

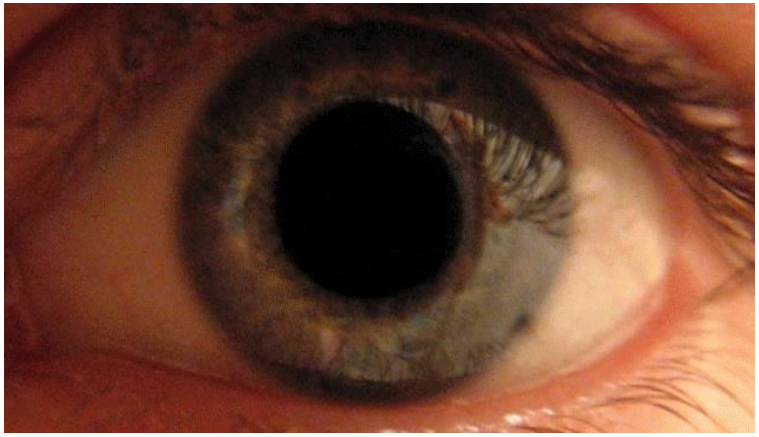
Self-Driving Car Tasks

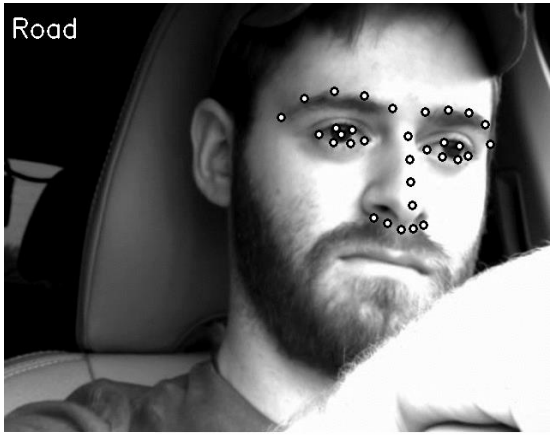
- **Localization:**
  Where am I?

- **Object detection:**
  Where is everyone else?

- **Movement planning:**
  How do I get from A to B?

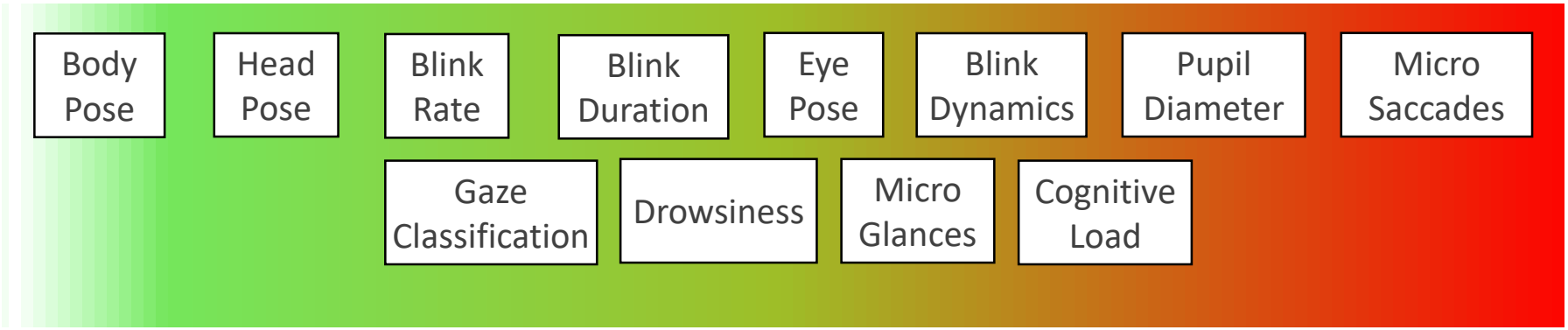
- **Driver state:**
  What's the driver up to?

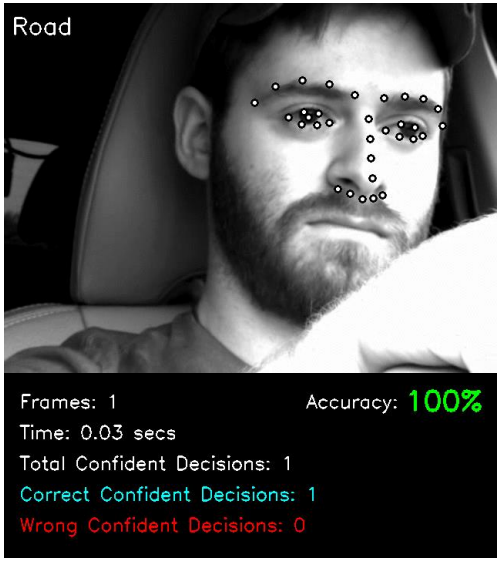


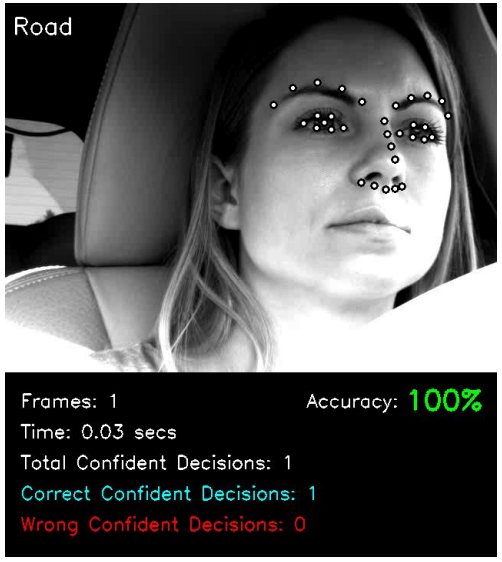


Drive State Detection:
A Multi-Resolutional View

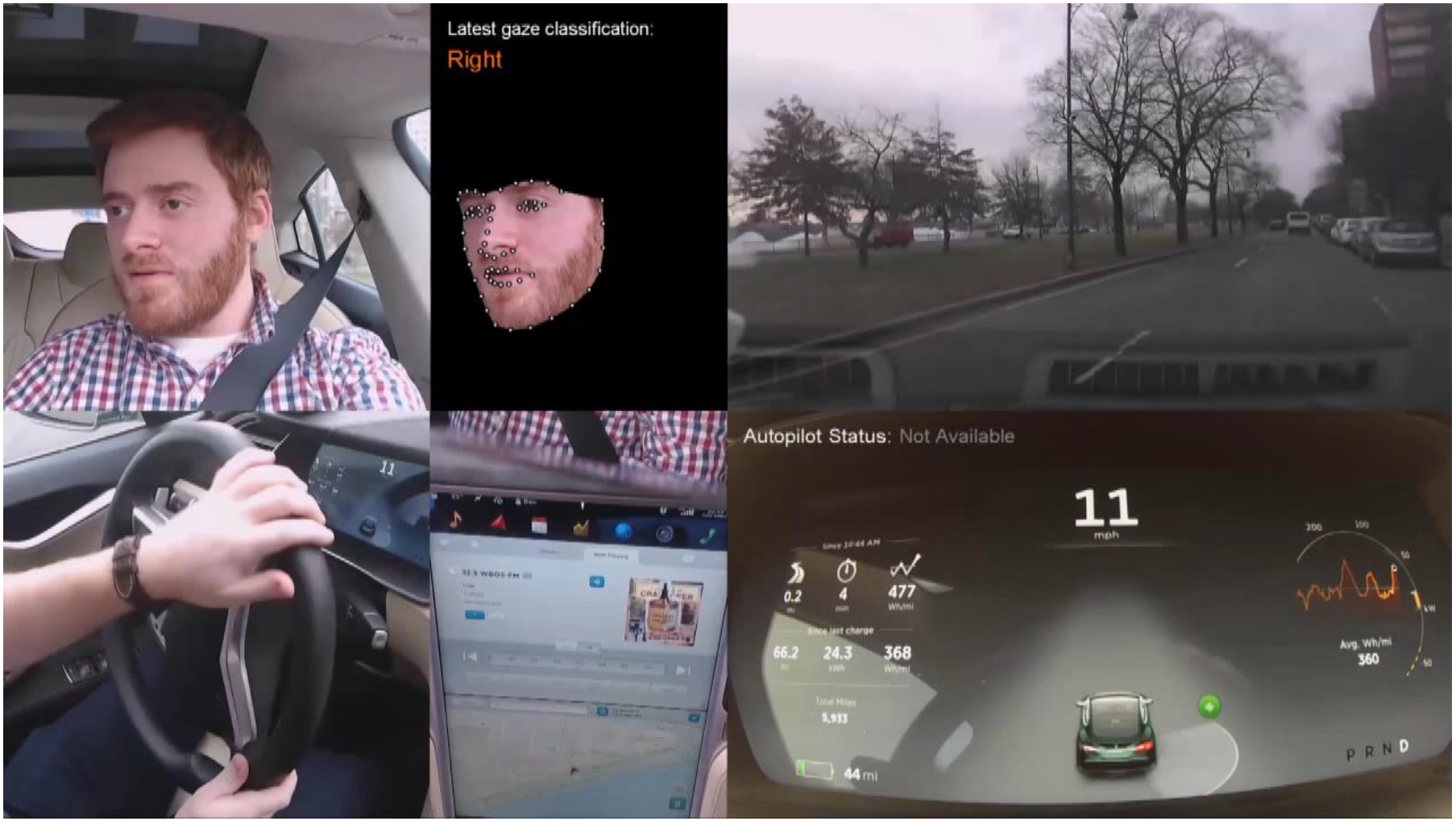
Increasing level of detection resolution and difficulty







Gaze Region and Autopilot State



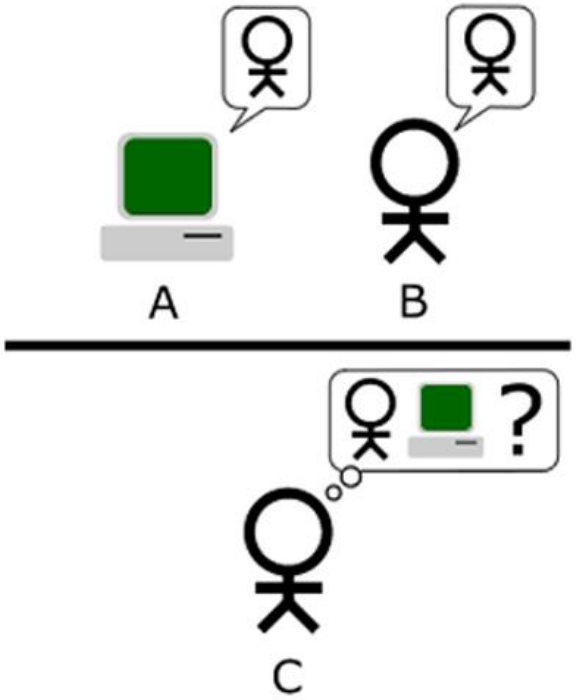
Driver Emotion

Class 1: Satisfied with Voice-Based Interaction

Class 2: Frustrated with Voice-Based Interaction

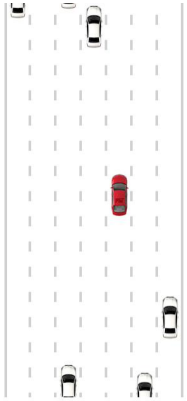
If Driving is a Conversation, this is an

End-to-End Natural Language Generation



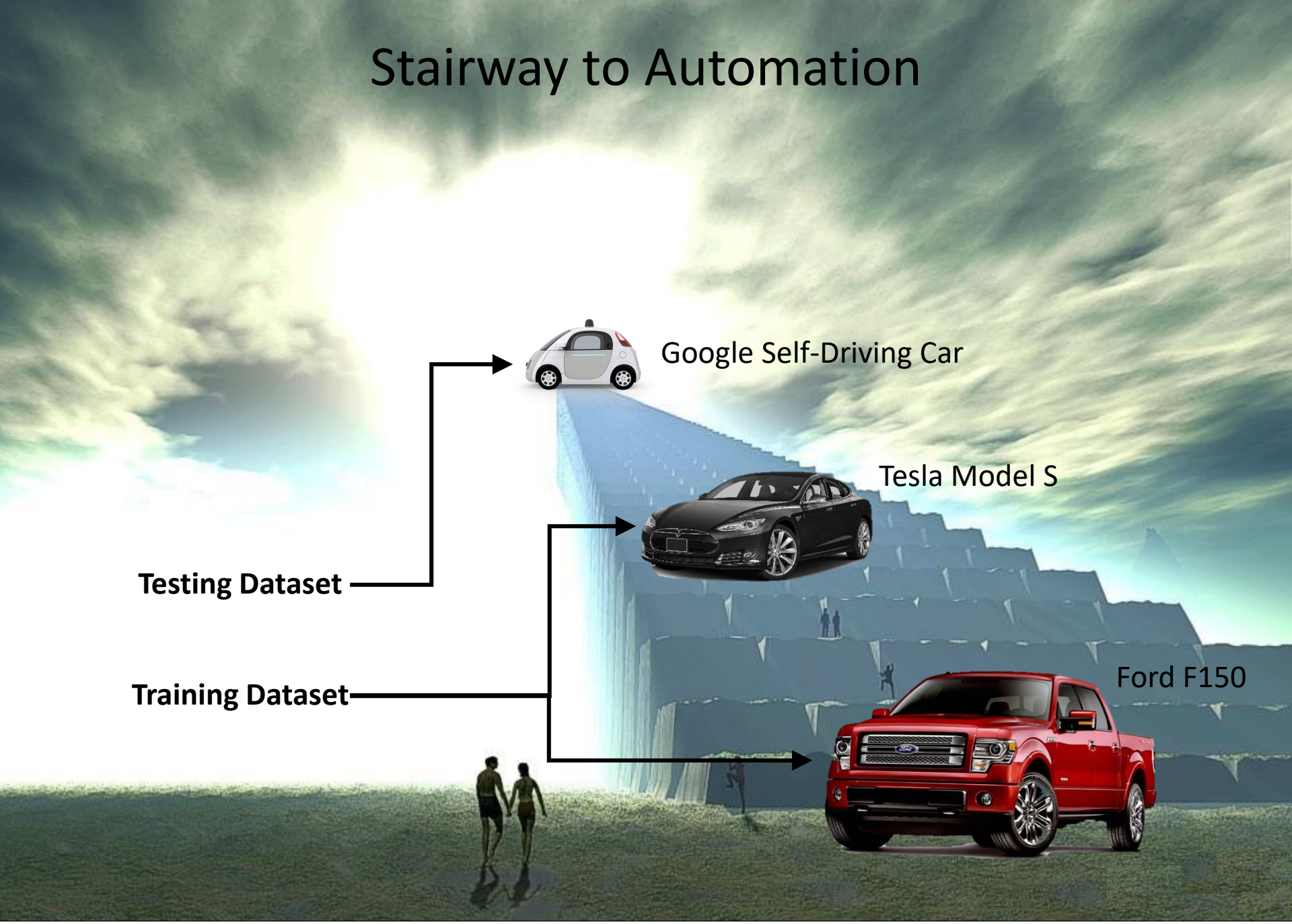
1. **Natural language processing** to enable it to communicate successfully
2. **Knowledge representation** to store information provided before or during the interrogation
3. **Automated reasoning** to use the stored information to answer questions and to draw new conclusions

Turing Test:
Can a computer be mistaken for a human more than 30% of the time?



Autonomous Driving: End-to-End

Stairway to Automation



Autonomous Driving: End-to-End

End to End Learning for Self-Driving Cars

Mariusz Bojarski
NVIDIA Corporation
Holmdel, NJ 07735

Davide Del Testa
NVIDIA Corporation
Holmdel, NJ 07735

Daniel Dworakowski
NVIDIA Corporation
Holmdel, NJ 07735

Bernhard Firner
NVIDIA Corporation
Holmdel, NJ 07735

Beat Flepp
NVIDIA Corporation
Holmdel, NJ 07735

Prasoon Goyal
NVIDIA Corporation
Holmdel, NJ 07735

Lawrence D. Jackel
NVIDIA Corporation
Holmdel, NJ 07735

Mathew Monfort
NVIDIA Corporation
Holmdel, NJ 07735

Urs Muller
NVIDIA Corporation
Holmdel, NJ 07735

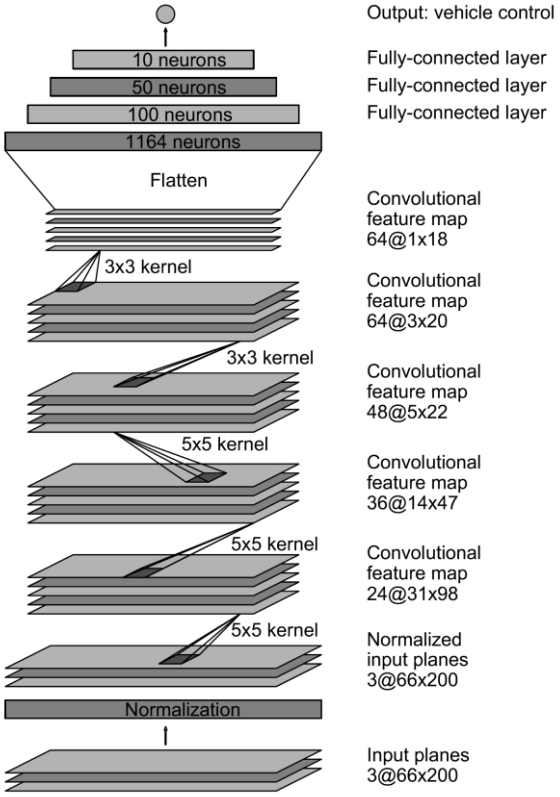
Jiakai Zhang
NVIDIA Corporation
Holmdel, NJ 07735

Xin Zhang
NVIDIA Corporation
Holmdel, NJ 07735

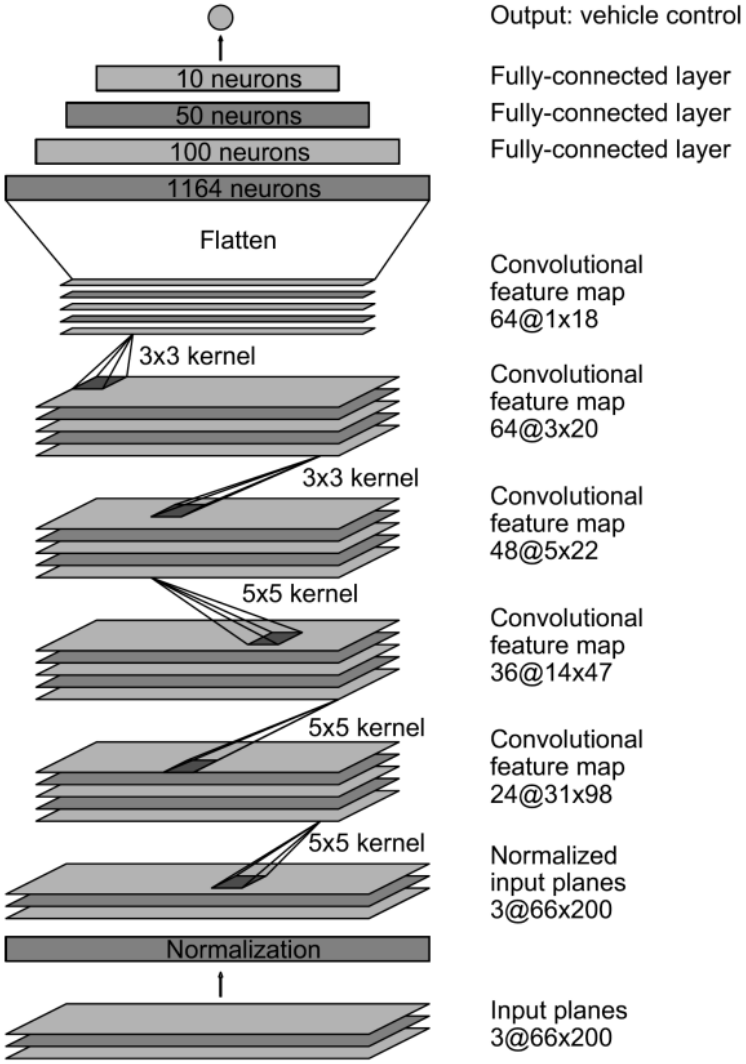
Jake Zhao
NVIDIA Corporation
Holmdel, NJ 07735

Karol Zieba
NVIDIA Corporation
Holmdel, NJ 07735

Autonomous Driving: End-to-End



Autonomous Driving: End-to-End



- 9 layers
  - 1 normalization layer
  - 5 convolutional layers
  - 3 fully connected layers
- 27 million connections
- 250 thousand parameters

End-to-End Driving with ConvnetJS

Tutorial on http://cars.mit.edu/deeptesla

End-to-end Steering

- By the end of this lecture, you'll be able to train a model that can steer a vehicle
- Our input to our network will be a single image of the forward roadway from a Tesla
- Our output will be a steering wheel value between -20 and 20

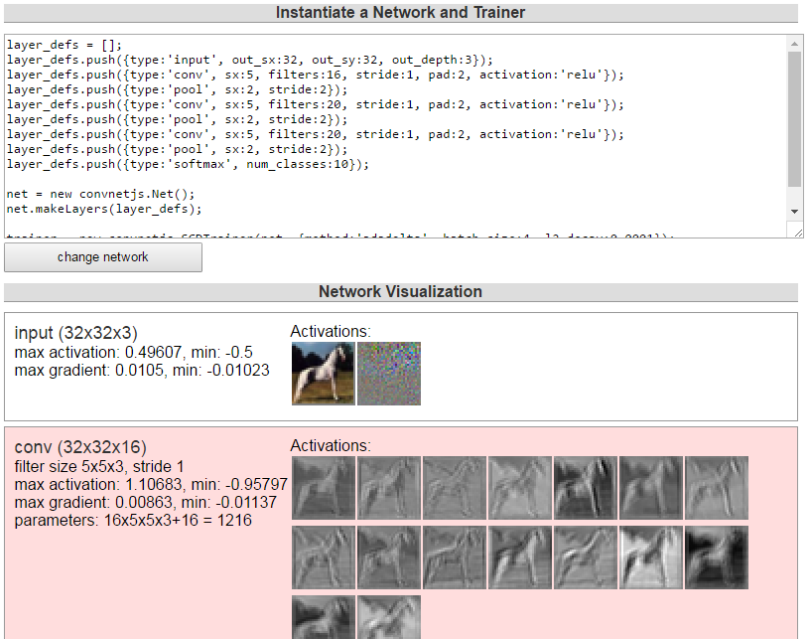
Creating the Dataset

- We recorded and extracted 10 video clips of highway driving from a Tesla
- The wheel value was extracted from the in-vehicle CAN
- We cropped/extracted a window from each video frame and provide a CSV linking the window to a wheel value

Lighting and Road Conditions

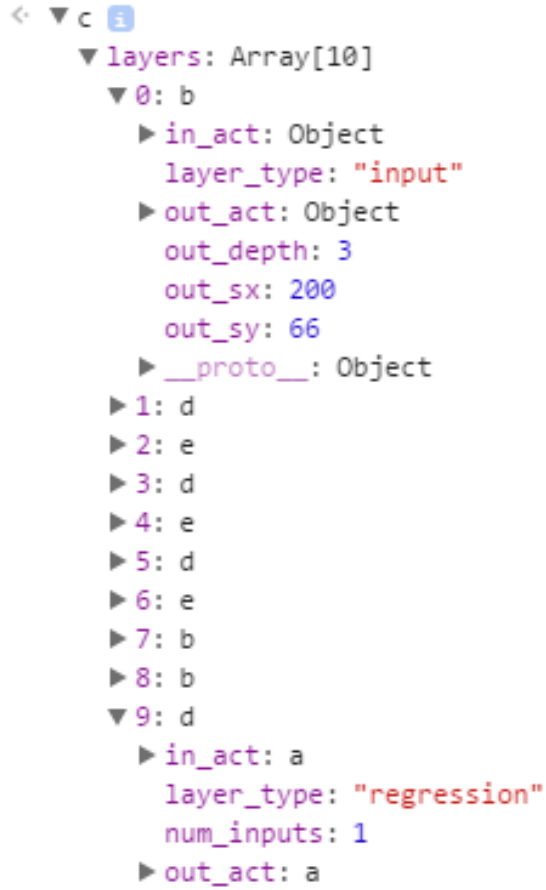
ConvNetJS Overview

- ConvNetJS is a Javascript implementation for using and training neural networks within the browser
- It supports simple networks with several different layer types and training algorithms
- Constructing and training a network can be performed in very few lines of code, great for demonstrations



ConvNetJS – Neural Network Representation

- The network is represented by a single Javascript object which contains a list of layers
- Each layer contains a plain array of weights (w), the activation/activation gradients of the last forward pass, as well as the shape and layer type



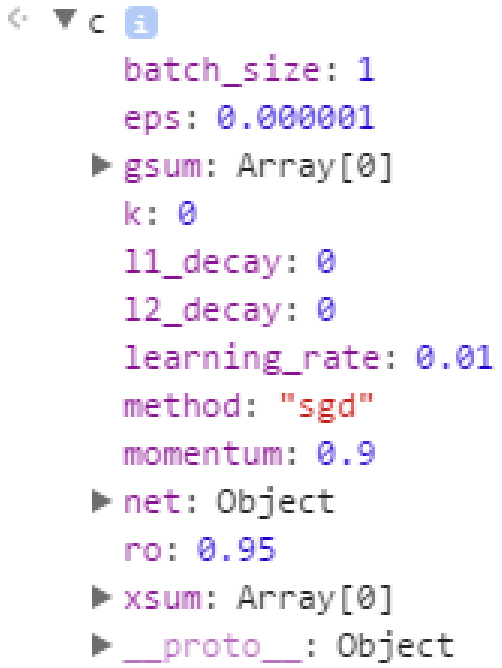
Layer Types

- ConvNetJS implements several different layer types: convolutional, pooling, fully-connected, local contrast normalization, and loss layers
- There are three available output types: regression, softmax, and SVM

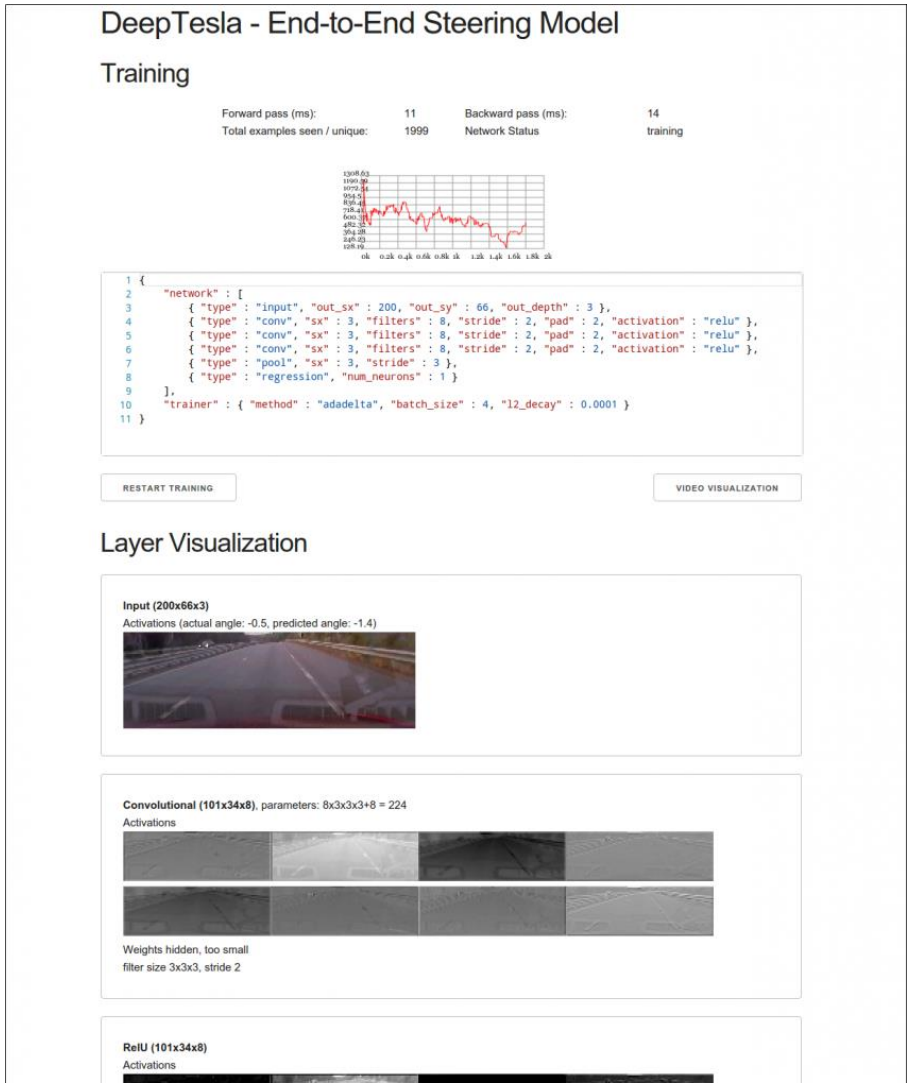
```
{
    "network" : [
        { "type" : "input", "out_sx" : 200, "out_sy" : 66, "out_depth" : 3 },
        { "type" : "conv", "sx" : 3, "filters" : 8, "stride" : 1, "pad" : 2, "activation" : "relu" },
        { "type" : "pool", "sx" : 2, "stride" : 2 },
        { "type" : "conv", "sx" : 3, "filters" : 8, "stride" : 1, "pad" : 2, "activation" : "relu" },
        { "type" : "pool", "sx" : 2, "stride" : 2 },
        { "type" : "conv", "sx" : 3, "filters" : 8, "stride" : 1, "pad" : 2, "activation" : "relu" },
        { "type" : "pool", "sx" : 3, "stride" : 3 },
        { "type" : "regression", "num_neurons" : 1 }
    ],
    "trainer" : { "method" : "adadelta", "batch_size" : 4, "l2_decay" : 0.0001 }
}
```

ConvNetJS – Training Overview

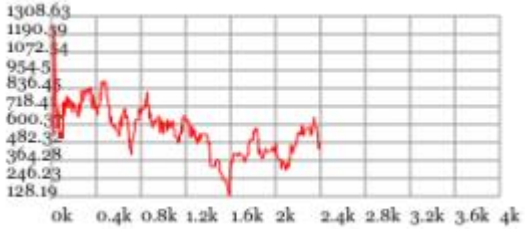
- To train a network, you first must initialize a “Trainer” object
  - var trainer = new convnetjs.SGDTrainer(net, { method: ‘adadelta’, batch_size: 1, l2_decay: 0.0001});
- There are three training algorithms available: SGD, Adadelta, and Adagrad.
- Training is performed by manually calling trainer.train(input_volume, expected_output)
- Returns an object containing timing and loss function information



DeepTesla Overview



Model Metrics



Forward pass (ms):	7	Backward pass (ms):	9
Total examples seen / unique:	2827	Network Status	training

Network Designer

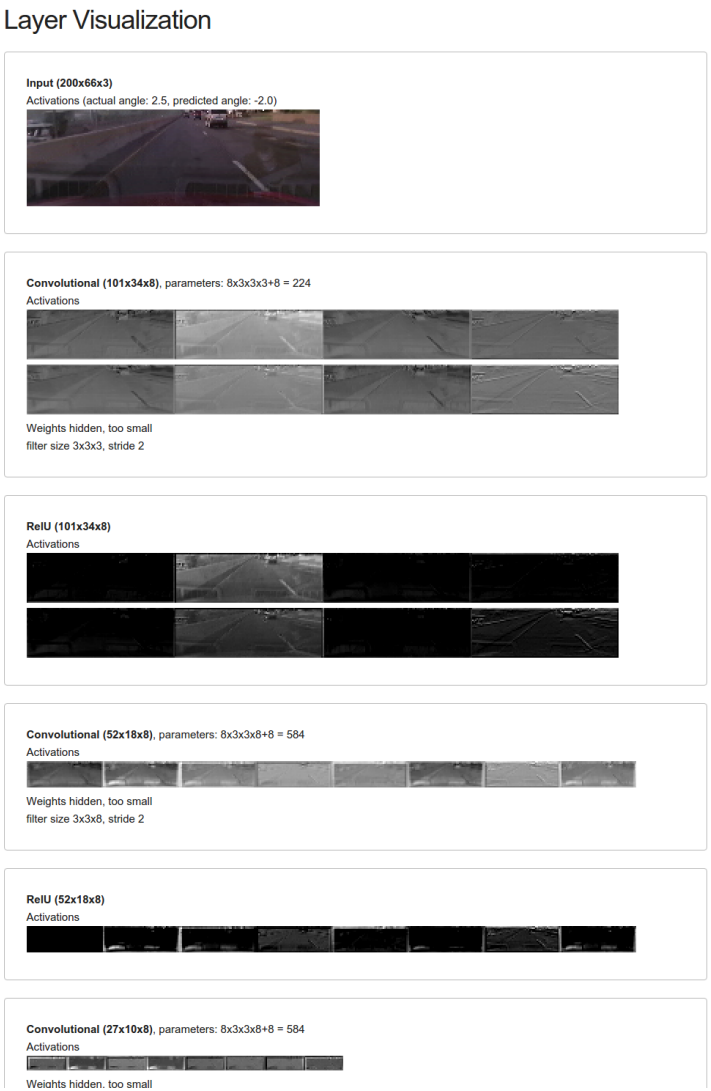
```
{
    "network" : [
        { "type" : "input", "out_sx" : 200, "out_sy" : 66, "out_depth" : 3 },
        { "type" : "conv", "sx" : 3, "filters" : 8, "stride" : 2, "pad" : 2, "activation" : "relu" },
        { "type" : "conv", "sx" : 3, "filters" : 8, "stride" : 2, "pad" : 2, "activation" : "relu" },
        { "type" : "conv", "sx" : 3, "filters" : 8, "stride" : 2, "pad" : 2, "activation" : "relu" },
        { "type" : "pool", "sx" : 3, "stride" : 3 },
        { "type" : "regression", "num_neurons" : 1 }
    ],
    "trainer" : { "method" : "adadelta", "batch_size" : 4, "l2_decay" : 0.0001 }
}
```

RESTART TRAINING

VIDEO VISUALIZATION

Training Interaction

Layer Visualization



Input Layer

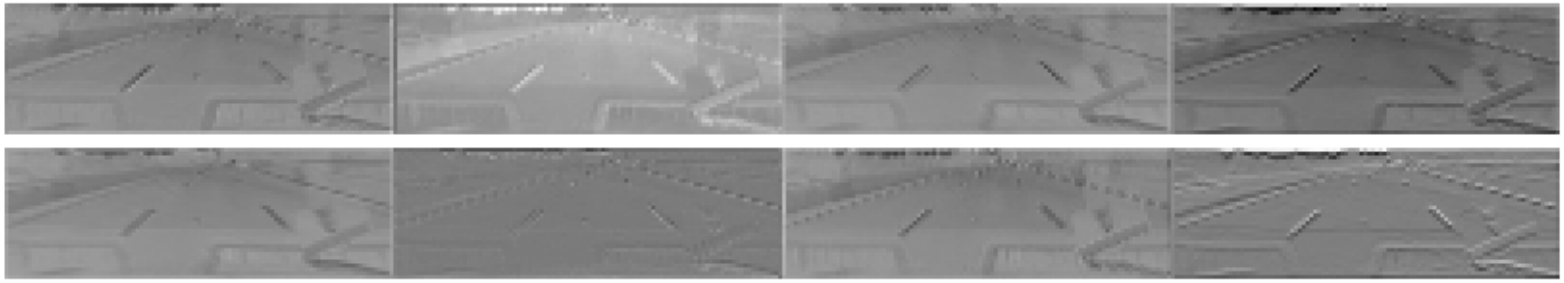
Input (200x66x3)

Activations (actual angle: 1.0, predicted angle: 1.5)

Convolutional Layer Visualization

Convolutional (101x34x8), parameters: 8x3x3x3+8 = 224

Activations



Weights hidden, too small

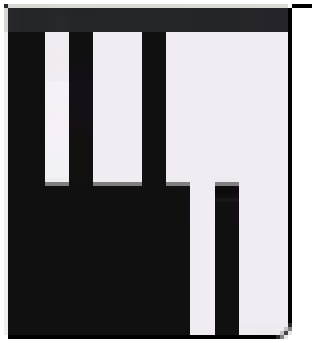
filter size 3x3x3, stride 2

Video Visualization

Information Bar

Input Box

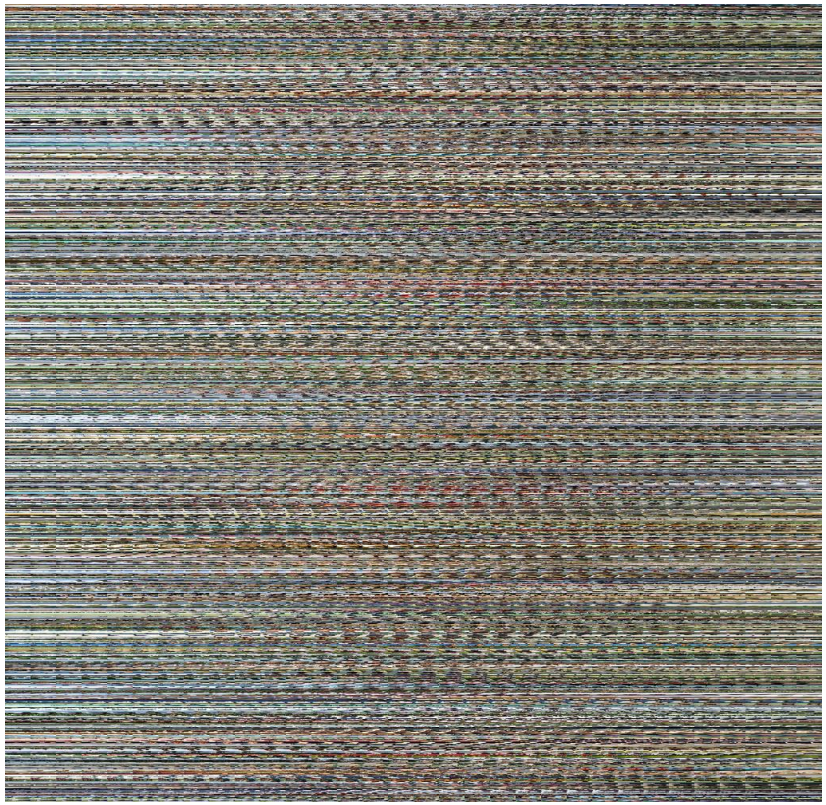
Barcodes



- 17 bit, sign-magnitude
- Encoded into actual video
- 0 = black, 1 = white
- Frame on top, wheel on bottom (divided by two)

Image Batches

- Each image loaded of the network contains an entire batch
- There is one image per row, and 250 rows in total
- These images are reassembled into volumes upon download

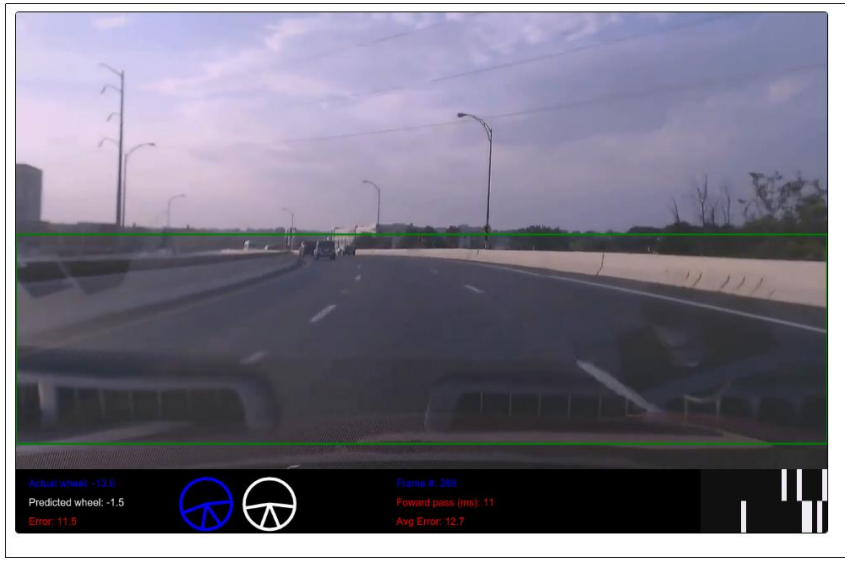


Training Explanation

- One web worker used for loading examples
  - Each batch of training images is one large image with each row as a single training example
  - After an image finishes loading asynchronously, sends the training examples to another worker
- One web worker used for training network
  - Train on each image and push the network/outputs to visualization worker
- One web worker used for visualization
  - For a specified training example interval, blit the activation/gradient output of each training example onto a canvas
- Each web worker behaves as a single thread, and we use message passing to communicate state between the workers

ConvNetJS Evaluation - Video Explanation

- The videos are encoded as 1280x820 in H264/MKV with 17 bit sign-magnitude barcodes
- The video main video frame is stored in the box (0, 1280, 0, 720)
- The frame barcode is in box (1144, 720, 1280, 770)
- The wheel value barcode is in box (1144, 770, 1280, 820)



ConvNetJS Evaluation - Creating the Video

- Each epoch is synchronized to 30 fps
- We extract the wheel value from the CAN data and synchronize each message to a frame (both the frame and the CAN message are timestamped)
- Using OpenCV, we process the data
  - Generate a bar code for the frame containing the wheel data
  - Crop the image portion used for training
  - Create single images containing batches of training data
- The epochs and associated data are copied to our web server which serves

ConvNetJS Evaluation - Playing the Video

- To be able to use the video in the neural network, we need to do some preprocessing
- First, we have a hidden video element and rely on modern HTML5 video implementations
- When the user requests the video to play, we begin tracking each redraw of the page
- With each redraw, we grab the currently rendered video frame, extract the RGBA values and blit them to two different canvases: one canvas, which the user sees, and another canvas which is hidden and only contains a cropped portion of the frame (the part we will use for the neural network)

ConvNetJS Evaluation - Playing the Video

- Next, we read the image data from the hidden canvas and shape it into a ConvNetJS volume
- For each image we first create a volume:
  - var image_vol = new convnetjs.Vol(x_size, y_size, depth, default_value)
- Next, we extract each pixel from the canvas and set the equivalent voxel (volume pixel) to the value (skipping the alpha value)
- We can also extract the expected steer value by parsing the barcode (a 17 bit, signed-magnitude barcode, where white = 1, black = 0)

ConvNetJS Evaluation - Forward Pass

- Now we can use our extracted volume in the forward pass by calling net.forward(our_volume)
- The predicted value is stored in the output neuron:
  - var prediction = net.forward(vol);
  - var raw_regression_value = prediction.w[0];

- Because we min-max normalized our inputs while training the network, we need to transform our outputs – this is just the reverse transformation we performed on input:
  - Wheel value = (raw_regression_value * total_wheel_range) – wheel_min

- We visualize the predicted and actual steering wheel values and calculate the error

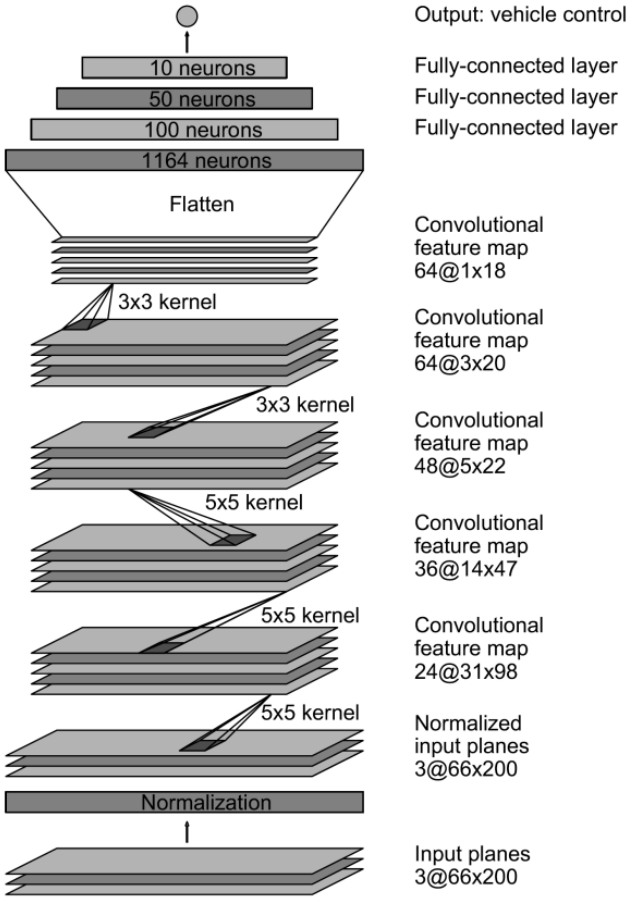
End-to-End Driving with ConvnetJS

Tutorial on http://cars.mit.edu/deeptesla

End-to-End Driving with TensorFlow

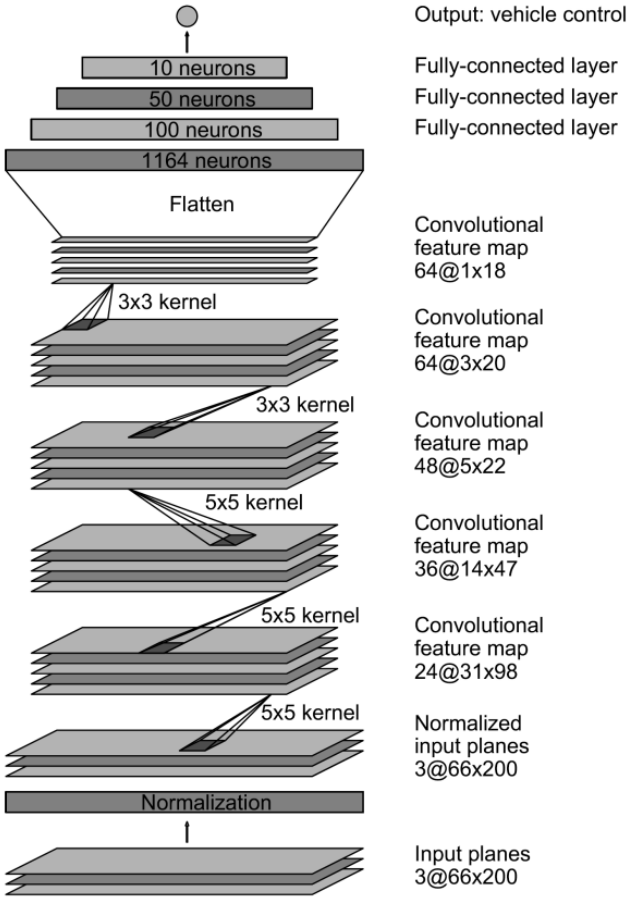
Available on http://github.com/lexfridman/deeptesla

Build the Model: Input and Output



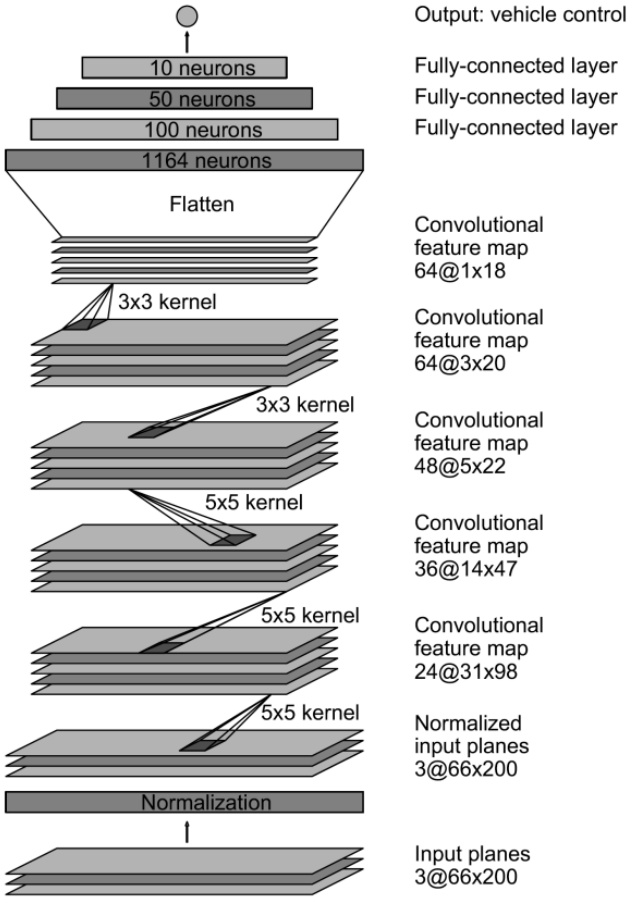

```
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

def conv2d(x, W, stride):
  return tf.nn.conv2d(x, W, strides=[1, stride, stride, 1],
padding='VALID')

x = tf.placeholder(tf.float32, shape=[None, 66, 200, 3])
y_ = tf.placeholder(tf.float32, shape=[None, 1])

x_image = x
```

Build the Model: Convolutional Layers




```
#first convolutional layer
W_conv1 = weight_variable([5, 5, 3, 24])
b_conv1 = bias_variable([24])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1, 2) + b_conv1)

#second convolutional layer
W_conv2 = weight_variable([5, 5, 24, 36])
b_conv2 = bias_variable([36])

h_conv2 = tf.nn.relu(conv2d(h_conv1, W_conv2, 2) + b_conv2)

#third convolutional layer
W_conv3 = weight_variable([5, 5, 36, 48])
b_conv3 = bias_variable([48])

h_conv3 = tf.nn.relu(conv2d(h_conv2, W_conv3, 2) + b_conv3)

#fourth convolutional layer
W_conv4 = weight_variable([3, 3, 48, 64])
b_conv4 = bias_variable([64])

h_conv4 = tf.nn.relu(conv2d(h_conv3, W_conv4, 1) + b_conv4)

#fifth convolutional layer
W_conv5 = weight_variable([3, 3, 64, 64])
b_conv5 = bias_variable([64])

h_conv5 = tf.nn.relu(conv2d(h_conv4, W_conv5, 1) + b_conv5)
```

Build the Model: Fully Connected Layers




```
# fully connected layer 1
W_fc1 = weight_variable([1152, 1164])
b_fc1 = bias_variable([1164])

h_conv5_flat = tf.reshape(h_conv5, [-1, 1152])
h_fc1 = tf.nn.relu(tf.matmul(h_conv5_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# fully connected layer 2
W_fc2 = weight_variable([1164, 100])
b_fc2 = bias_variable([100])

h_fc2 = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)

# fully connected layer 3
W_fc3 = weight_variable([100, 50])
b_fc3 = bias_variable([50])

h_fc3 = tf.nn.relu(tf.matmul(h_fc2_drop, W_fc3) + b_fc3)

h_fc3_drop = tf.nn.dropout(h_fc3, keep_prob)

# fully connected layer 4
W_fc4 = weight_variable([50, 10])
b_fc4 = bias_variable([10])

h_fc4 = tf.nn.relu(tf.matmul(h_fc3_drop, W_fc4) + b_fc4)

h_fc4_drop = tf.nn.dropout(h_fc4, keep_prob)

#Output
W_fc5 = weight_variable([10, 1])
b_fc5 = bias_variable([1])

y = tf.mul(tf.atan(tf.matmul(h_fc4_drop, W_fc5) + b_fc5), 2)
```

Train the Model

```
sess = tf.InteractiveSession()

loss = tf.reduce_mean(tf.square(tf.sub(model.y_, model.y)))
train_step = tf.train.AdamOptimizer(1e-4).minimize(loss)
sess.run(tf.initialize_all_variables())

saver = tf.train.Saver()

for i in range(int(driving_data.num_images * 0.3)):
    xs, ys = driving_data.LoadTrainBatch(100)
    train_step.run(feed_dict={model.x: xs, model.y_: ys, model.keep_prob: 0.8})
    if i % 10 == 0:
        xs, ys = driving_data.LoadValBatch(100)
        print("step %d, val loss %g"%(i, loss.eval(feed_dict={
            model.x:xs, model.y_: ys, model.keep_prob: 1.0})))
    if i % 100 == 0:
        if not os.path.exists(LOGDIR):
            os.makedirs(LOGDIR)
        checkpoint_path = os.path.join(LOGDIR, "model.ckpt")
        filename = saver.save(sess, checkpoint_path)
        print("Model saved in file: %s" % filename)
```

Run the Model

```
import tensorflow as tf
import scipy.misc
import model
import cv2

sess = tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, "save/model.ckpt")

img = cv2.imread('steering_wheel_image.jpg',0)
rows,cols = img.shape

smoothed_angle = 0

cap = cv2.VideoCapture(0)
while(cv2.waitKey(10) != ord('q')):
    ret, frame = cap.read()
    image = scipy.misc.imresize(frame, [66, 200]) / 255.0
    degrees = model.y.eval(feed_dict={model.x: [image], model.keep_prob:
1.0})[0][0] \
              * 180 / scipy.pi
    cv2.imshow('frame', frame)
    smoothed_angle += 0.2 * pow(abs((degrees - smoothed_angle)), 2.0 / 3.0) * \
                      (degrees - smoothed_angle) / abs(degrees - smoothed_angle)
    M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)
    dst = cv2.warpAffine(img,M,(cols,rows))
    cv2.imshow("steering wheel", dst)

cap.release()
cv2.destroyAllWindows()
```

End-to-End Driving with TensorFlow

Available on http://github.com/lexfridman/deeptesla

TrafficLight Classification with TensorFlow



We will be implementing a simple traffic light classifier, with 3 classes (red, green, yellow)

Parameters

```
# Basic parameters

max_epochs = 25
base_image_path = "images/"
image_types = ["red", "green", "yellow"]
input_img_x = 32
input_img_y = 32
train_test_split_ratio = 0.9
batch_size = 32
checkpoint_name = "model.ckpt"
```

- Max epochs: the number of times the neural network will see all training examples
- Input_img_x/y: the size we will use for inputs into the the network
- Batch size: # of examples the neural network will see before making a gradient step

Helper Functions

```python
# Helper layer functions
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def conv2d(x, W, stride):
    return tf.nn.conv2d(x, W, strides=[1, stride, stride, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
```

We use some helper functions to make adding layers easier/more consistent

Model Input/Output

```
x = tf.placeholder(tf.float32, shape=[None, input_img_x, input_img_y, 3])
y_ = tf.placeholder(tf.float32, shape=[None, len(image_types)])
```

- We specify our input and output types in the same lines to make sure they agree with our idea of the network
- Our input is an image of sized 32x32x3 (RGB channels)
- Our output consists of 3 neurons, representing the probability of each class

Convolutional Layer

```
# Our first three convolutional layers, of 16 3x3 filters
W_conv1 = weight_variable([3, 3, 3, 16])
b_conv1 = bias_variable([16])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1, 1) + b_conv1)
```

- Here we specify our first convolutional layer using our helper function
- W_conv1 – a 4D tensor representing the weights [filter_x, filter_y, previous layer neurons, # of filters]
- b_conv1 – our simple addition variable
- h_conv1 – our actual layer/activation

Pooling Layer

```
h_pool4 = max_pool_2x2(h_conv3)
```

Flattening Pool Layer

```
n1, n2, n3, n4 = h_pool4.get_shape().as_list()

W_fc1 = weight_variable([n2*n3*n4, 3])
b_fc1 = bias_variable([3])

# We flatten our pool layer into a fully connected layer

h_pool4_flat = tf.reshape(h_pool4, [-1, n2*n3*n4])
```

We calculate the total number of neurons needed in our first fully-connected layer by multiplying all the dimensions of the pool layer shape

Output Layer

```
y = tf.matmul(h_pool4_flat, W_fc1) + b_fc1
```

Loss and Optimizer

```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
train_step = tf.train.AdamOptimizer(1e-4).minimize(loss)
sess.run(tf.initialize_all_variables())
```

- Our loss function performs softmax and then computes cross-entropy
- We use the AdamOptimizer and specify a learning rate

Saver Object

```
saver = tf.train.Saver()
time_start = time.time()
```

Loading Images

```
full_set = []

for im_type in image_types:
    for ex in glob.glob(os.path.join(base_image_path, im_type, "*")):
        im = cv2.imread(ex)
        if not im is None:
            im = cv2.resize(im, (32, 32))

            # Create an array representing our classes and set it
            one_hot_array = [0] * len(image_types)
            one_hot_array[image_types.index(im_type)] = 1
            assert(im.shape == (32, 32, 3))

            full_set.append((im, one_hot_array))

random.shuffle(full_set)
```

- Iterate over each image, resize to 32x32
- Create a one hot encoding of our class
- Shuffle the entire dataset

Splitting the Dataset

```
# We split our data into a training and test set here

split_index = int(math.floor(len(full_set) * train_test_split_ratio))
train_set = full_set[:split_index]
test_set = full_set[split_index:]
```

```
train_x, train_y = zip(*train_set)
test_x, test_y = zip(*test_set)
```

- Split our data set into train and test
- We truncate our sets to a multiple of batch size (all batches have to be the same size)

Training Loop

```
for tt in range(0, (len(train_x) / batch_size)):
    start_batch = batch_size * tt
    end_batch = batch_size * (tt + 1)
    train_step.run(feed_dict={x: train_x[start_batch:end_batch], y_: train_y[start_batch:end_batch]})
    ex_seen = "Current epoch, examples seen: {:20} / {} \r".format(tt * batch_size, len(train_x))
    sys.stdout.write(ex_seen.format(tt * batch_size))
    sys.stdout.flush()
```

- Iterate over each batch and train on it
- (we assume training examples are a multiple of the batch size)

Best Model

```
t_loss = loss.eval(feed_dict={x: train_x, y_: train_y})
v_loss = loss.eval(feed_dict={x: test_x, y_: test_y})

sys.stdout.write("Epoch {:5}: loss: {:15.10f}, val. loss: {:15.10f}".format(i + 1, t_loss, v_loss))

if v_loss < least_loss:
    sys.stdout.write(", saving new best model to {}".format(checkpoint_name))
    least_loss = v_loss
    filename = saver.save(sess, checkpoint_name)
```

- We evaluate the loss on all of our training examples and test examples
- If the validation loss is lower than the lowest loss, we save our model

Expected Output

```
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcublas.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcudnn.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcufft.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcurand.so locally
I tensorflow/core/common_runtime/gpu/gpu_init.cc:102] Found device 0 with properties:
name: GeForce GTX 980 Ti
major: 5 minor: 2 memoryClockRate (GHz) 1.076
pciBusID 0000:02:00.0
Total memory: 6.00GiB
Free memory: 5.52GiB
I tensorflow/core/common_runtime/gpu/gpu_init.cc:126] DMA: 0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 0:   Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:838] Creating TensorFlow device (/gpu:0) -> (device: 0, n
orce GTX 980 Ti, pci bus id: 0000:02:00.0)
Starting training... [1312 training examples]
Epoch     1: loss:    3.4614486694, val. loss:    4.1707162857, saving new best model to model.ckpt
Epoch     2: loss:    1.8578453064, val. loss:    1.8420848846, saving new best model to model.ckpt
Epoch     3: loss:    1.2075960636, val. loss:    1.1884875298, saving new best model to model.ckpt
Epoch     4: loss:    0.7959808707, val. loss:    0.8523907661, saving new best model to model.ckpt
Epoch     5: loss:    0.5186702609, val. loss:    0.8251042366, saving new best model to model.ckpt
Epoch     6: loss:    0.3895130754, val. loss:    0.7883402705, saving new best model to model.ckpt
Epoch     7: loss:    0.2840102911, val. loss:    0.8282299042
Epoch     8: loss:    0.2281106114, val. loss:    0.7970542312
Epoch     9: loss:    0.1889465898, val. loss:    0.8308163881
Epoch    10: loss:    0.1507862508, val. loss:    0.8214046359
Epoch    11: loss:    0.1126181334, val. loss:    0.7745668888, saving new best model to model.ckpt
Epoch    12: loss:    0.0972050354, val. loss:    0.7550495267, saving new best model to model.ckpt
Epoch    13: loss:    0.0831822604, val. loss:    0.7207581401, saving new best model to model.ckpt
Epoch    14: loss:    0.0760012567, val. loss:    0.7217628956
Epoch    15: loss:    0.0637019351, val. loss:    0.7158752084, saving new best model to model.ckpt
Epoch    16: loss:    0.0536490902, val. loss:    0.7022477984, saving new best model to model.ckpt
Epoch    17: loss:    0.0512478650, val. loss:    0.6906370521, saving new best model to model.ckpt
Epoch    18: loss:    0.0333782099, val. loss:    0.6657178402, saving new best model to model.ckpt
Epoch    19: loss:    0.0367136374, val. loss:    0.6645810008, saving new best model to model.ckpt
Epoch    20: loss:    0.0175283104, val. loss:    0.6335256100, saving new best model to model.ckpt
Epoch    21: loss:    0.0203501396, val. loss:    0.6400516629
Epoch    22: loss:    0.0372636504, val. loss:    0.6770884395
Epoch    23: loss:    0.0096475044, val. loss:    0.5926443338, saving new best model to model.ckpt
Epoch    24: loss:    0.0095629999, val. loss:    0.5750324726, saving new best model to model.ckpt
Epoch    25: loss:    0.0072135003, val. loss:    0.5805844069
```

TrafficLight Classification with TensorFlow



We will be implementing a simple traffic light classifier, with 3 classes (red, green, yellow)

References

All references cited in this presentation are listed in the following Google Sheets file:

https://goo.gl/9Xhp2t