Lecture 5: Recognition

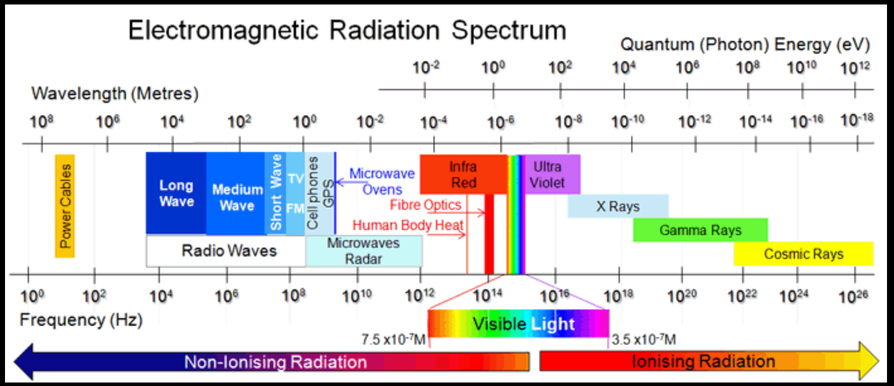
Professor Katie Driggs-Campbell

February 11, 2021

ECE484: Principles of Safe Autonomy

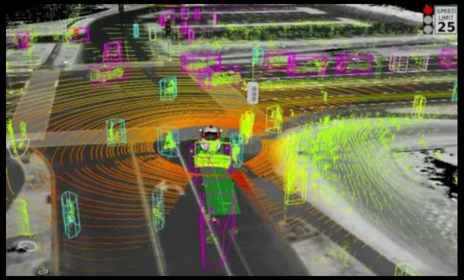
Administrivia

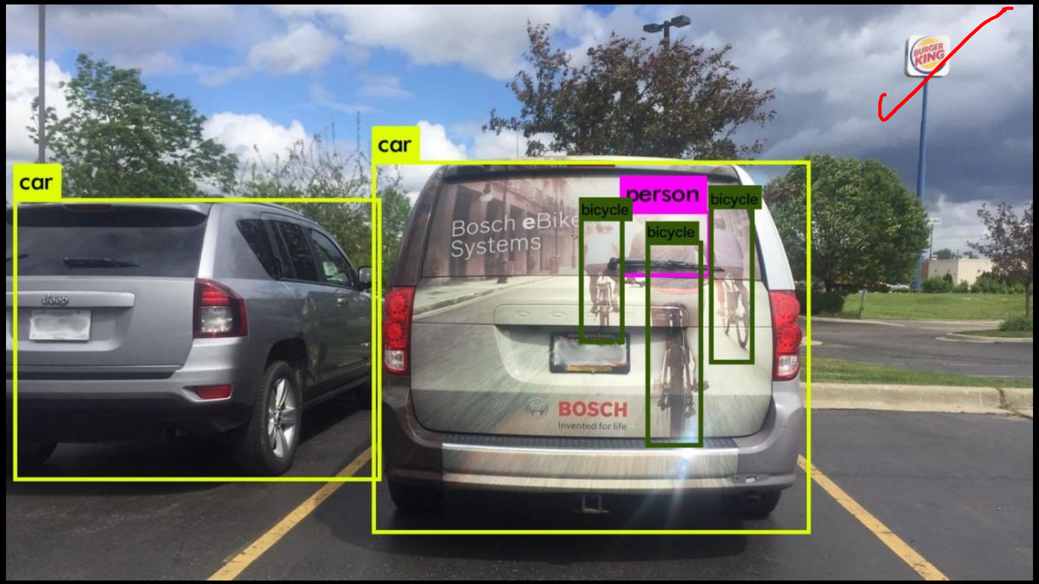
- MP0 is due tomorrow
  - Make sure you give a demo by the end of today!
  - Report due tomorrow (Friday) at 5pm



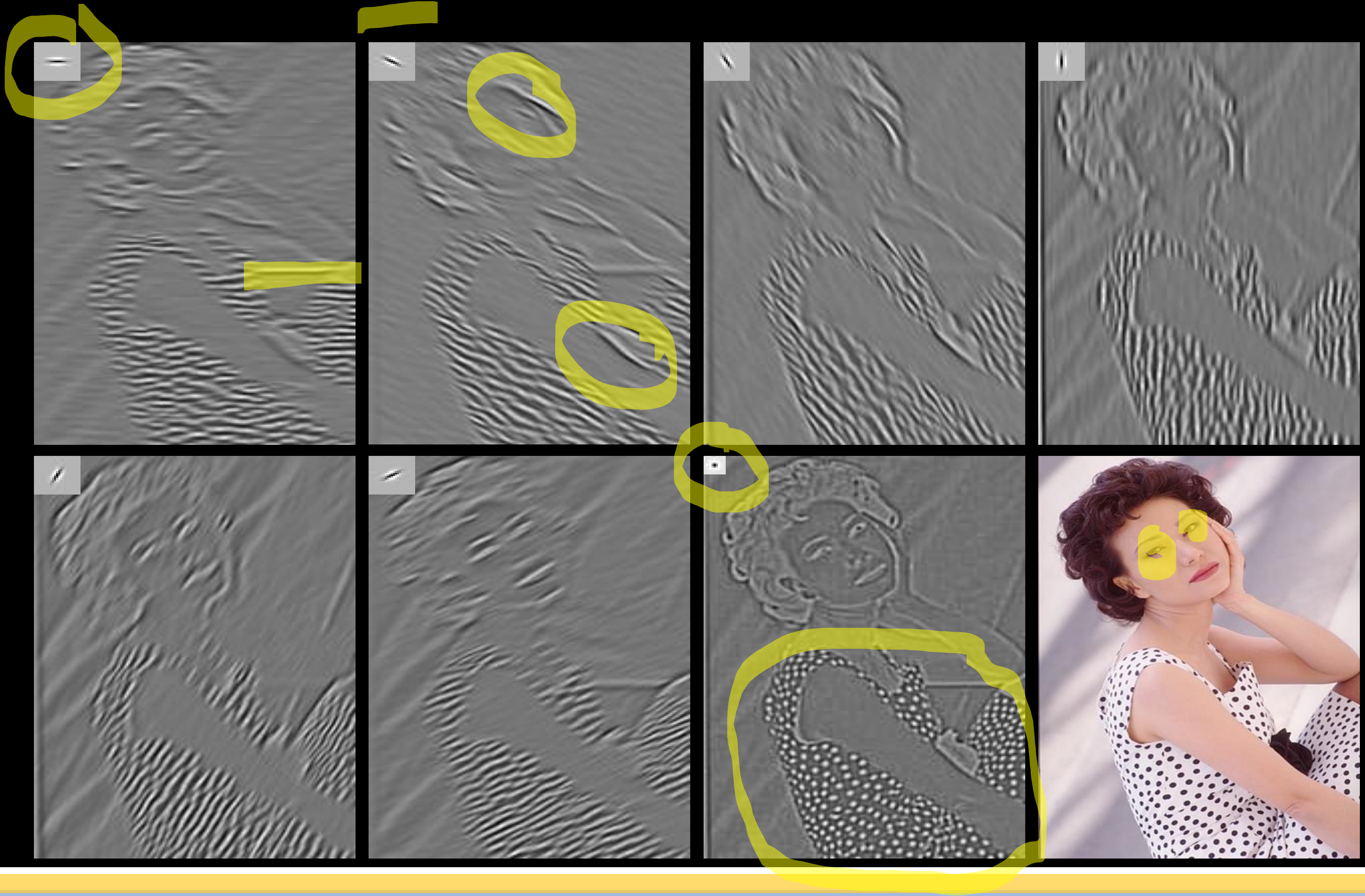
The Challenge of Perception

Sensor Goal: Process electromagnetic radiation from the environment to construct a *model* of the world, so that the constructed model is close to the real world and that the output is *actionable*





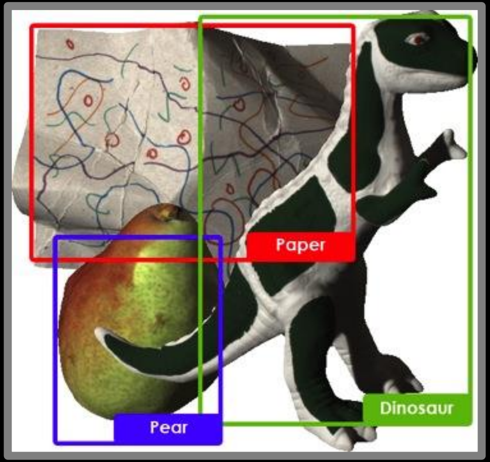
Filter Outputs



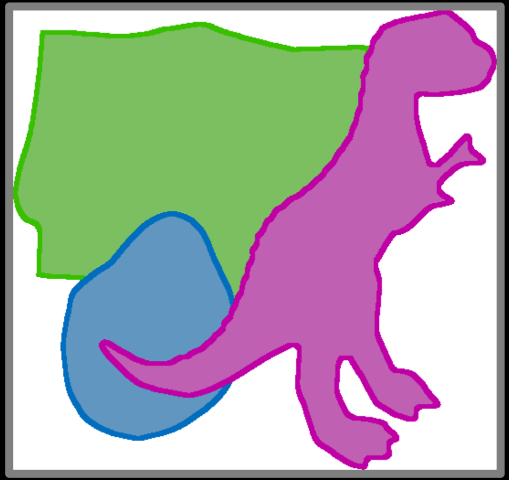
Today’s Plan

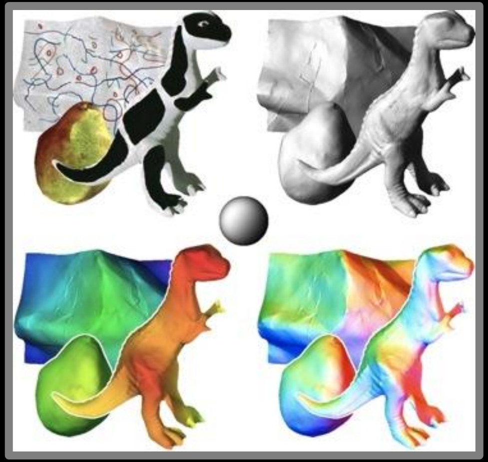
- Computer vision overview
- Object recognition
  - Feature representations
  - Classification
- (Convolutional) Neural Networks

The Three R's of Computer Vision



Recognition





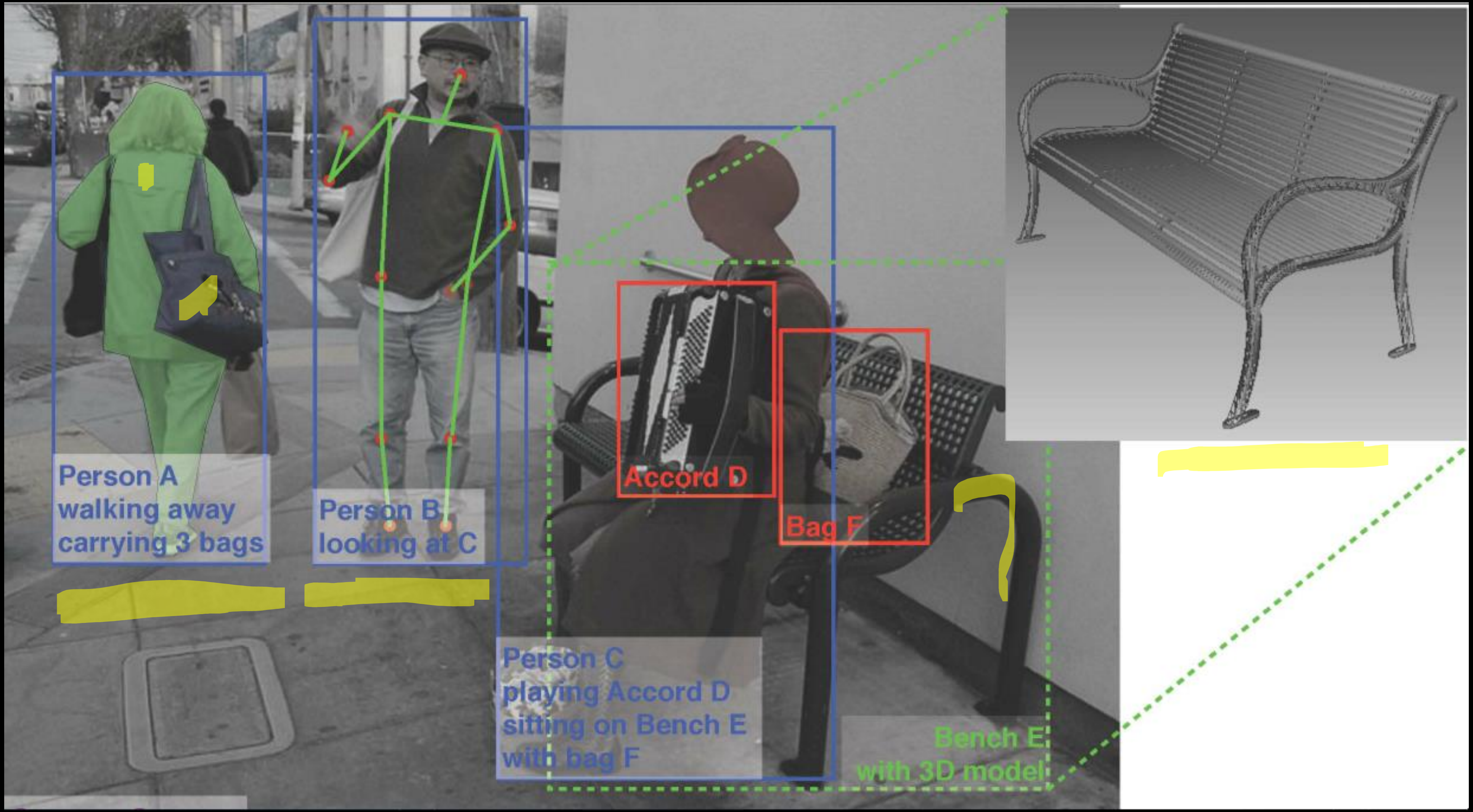
Reconstruction

Reorganization

Slide Credit: J. Malik

Slide Credit: J. Malik

What we would like to infer…

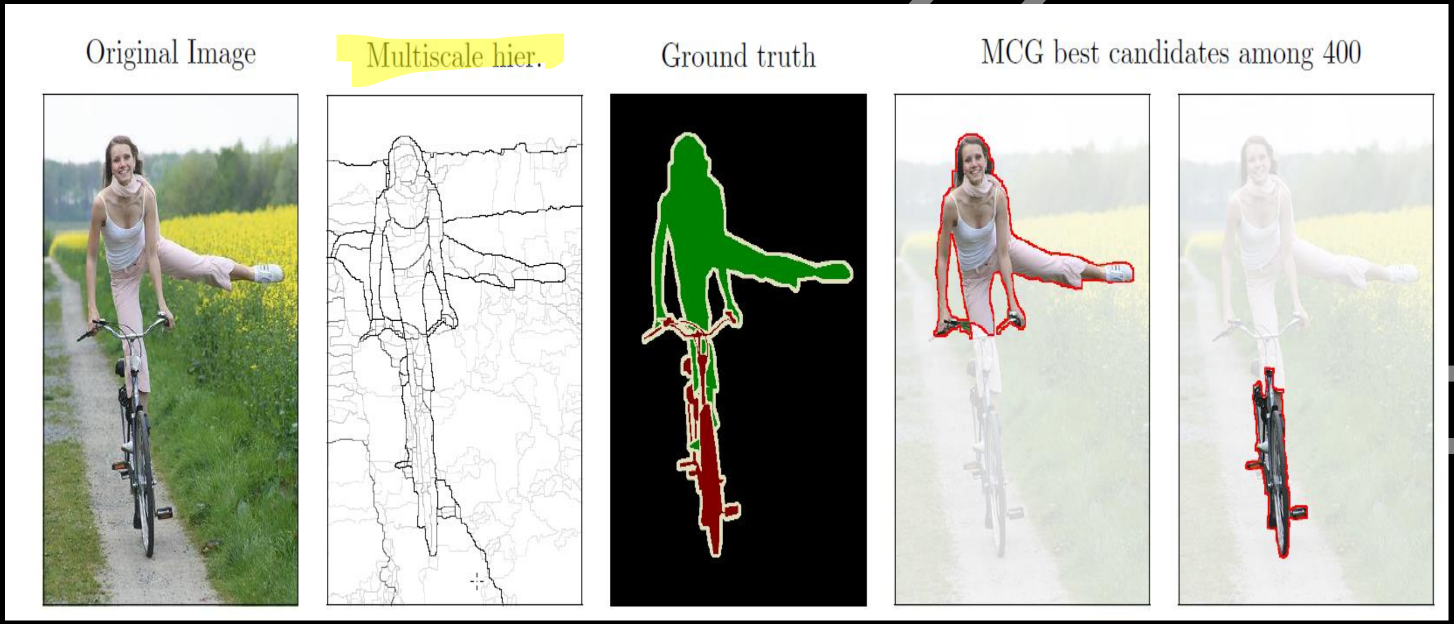


Will person B put some money into Person C's tip bag?

Slide Credit: J. Malik

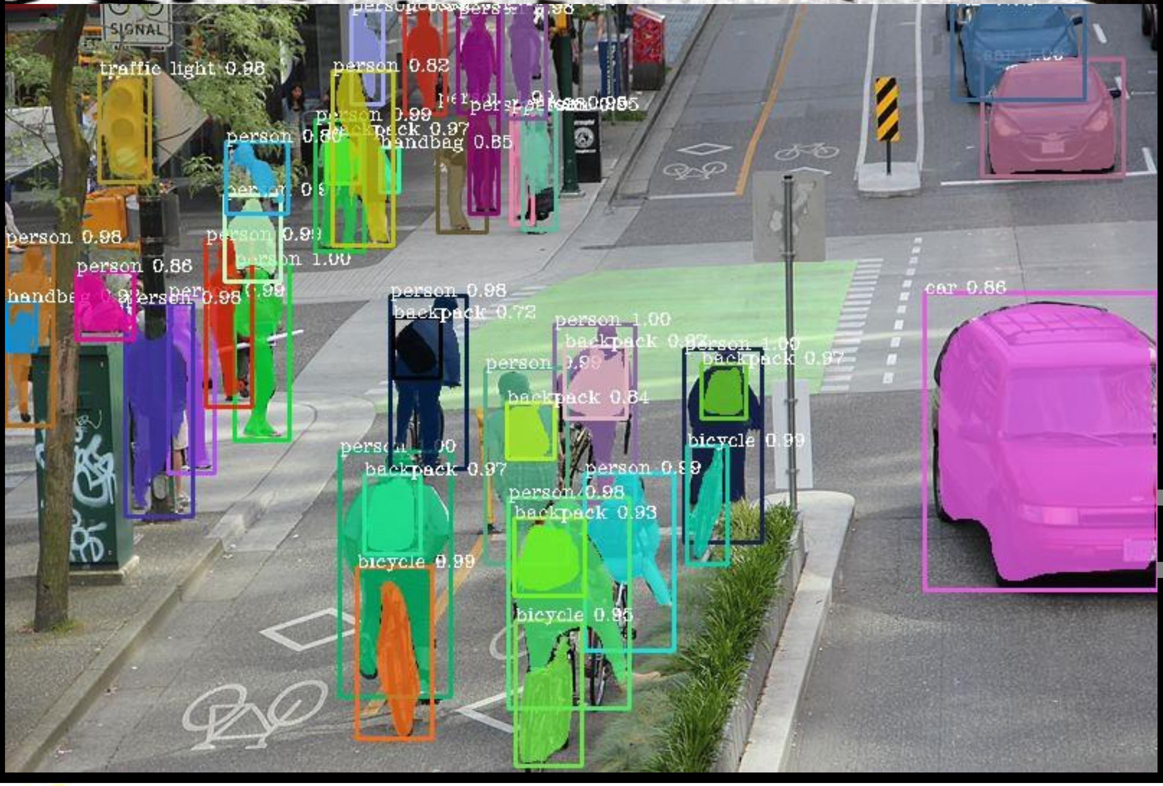
The Three R's of Vision

Recognition



Reorganization

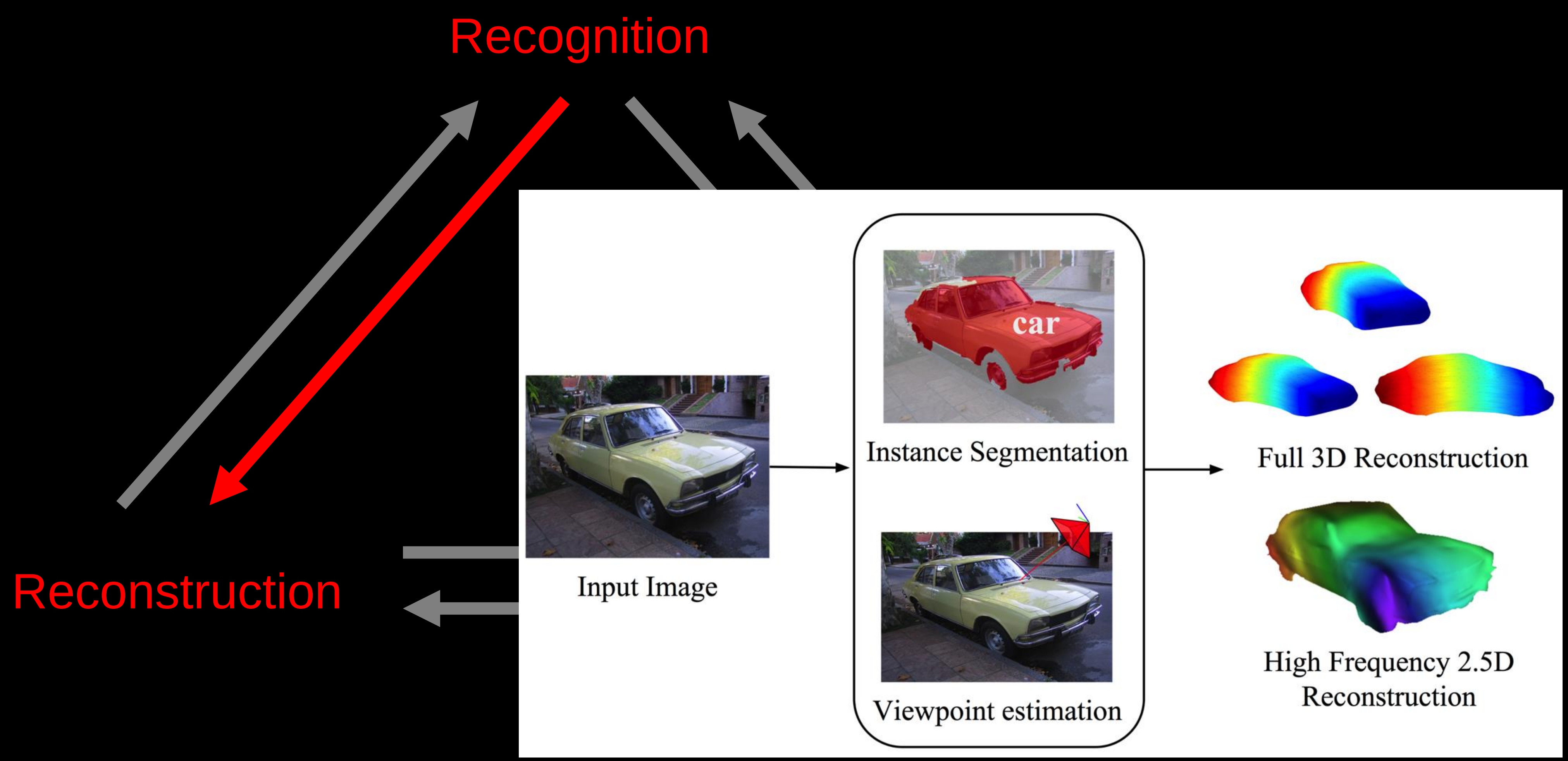
other direction…



cognition

Reorganization

The Three R's of Vision



What is image recognition?

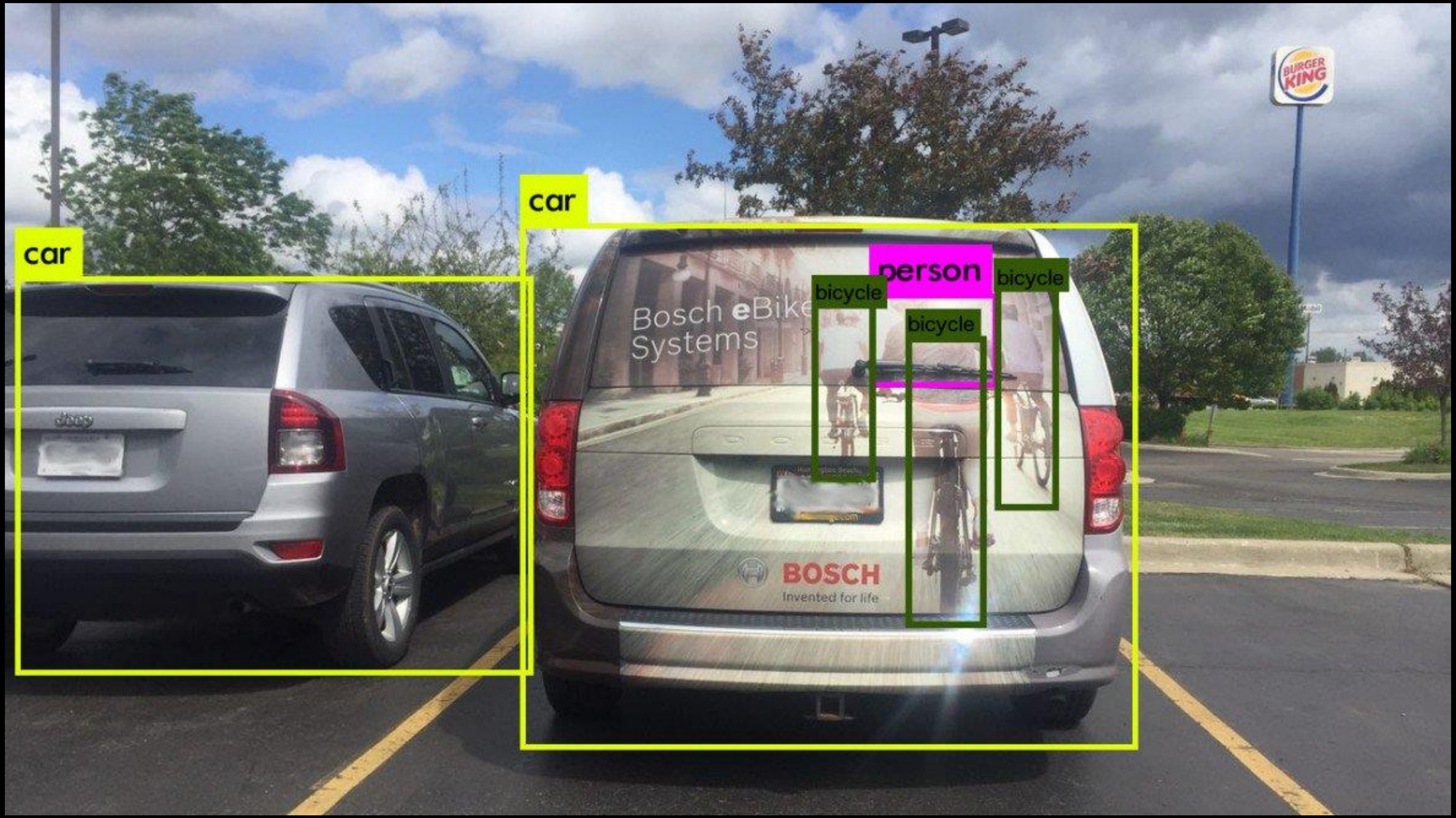


Image Recognition

samples →

classes ↓

f() = "apple" ✓

f() = "tomato"

f() = "cow"

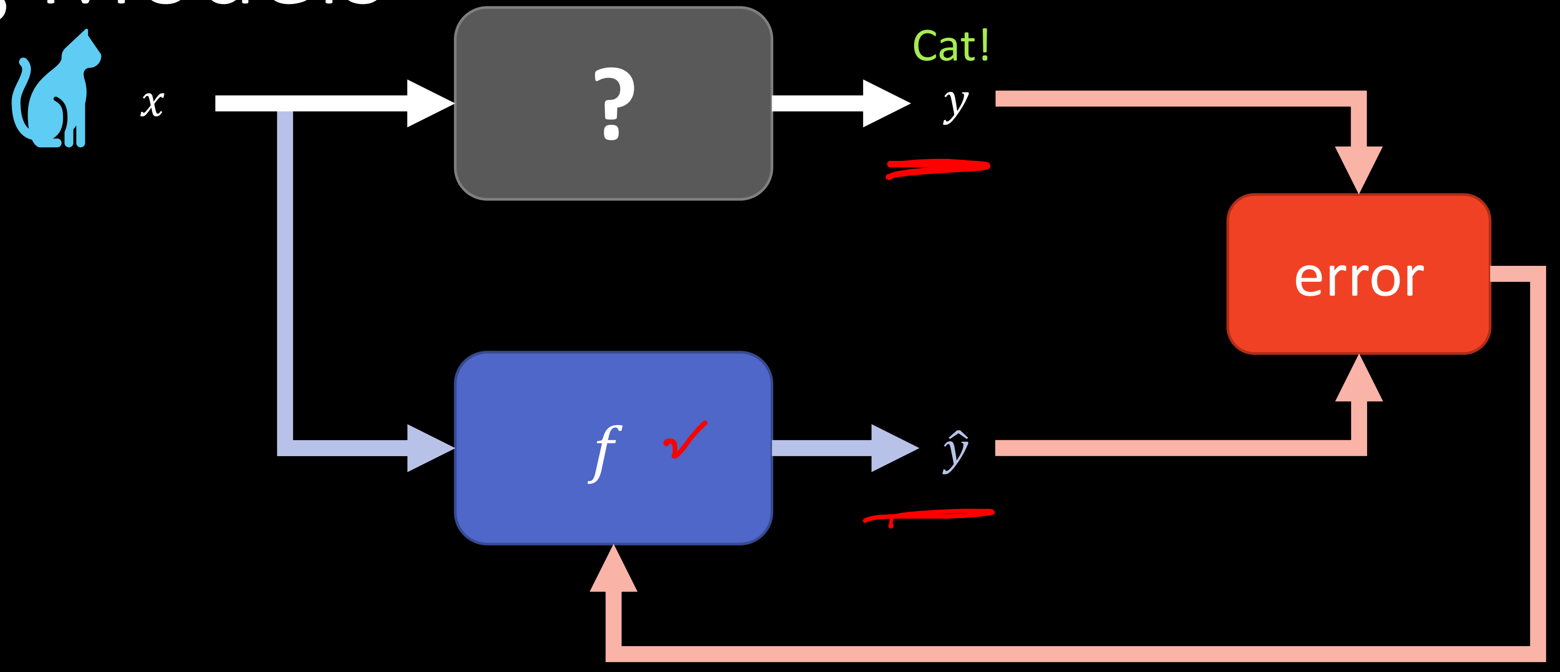
output label, input features

$y = f(x)$

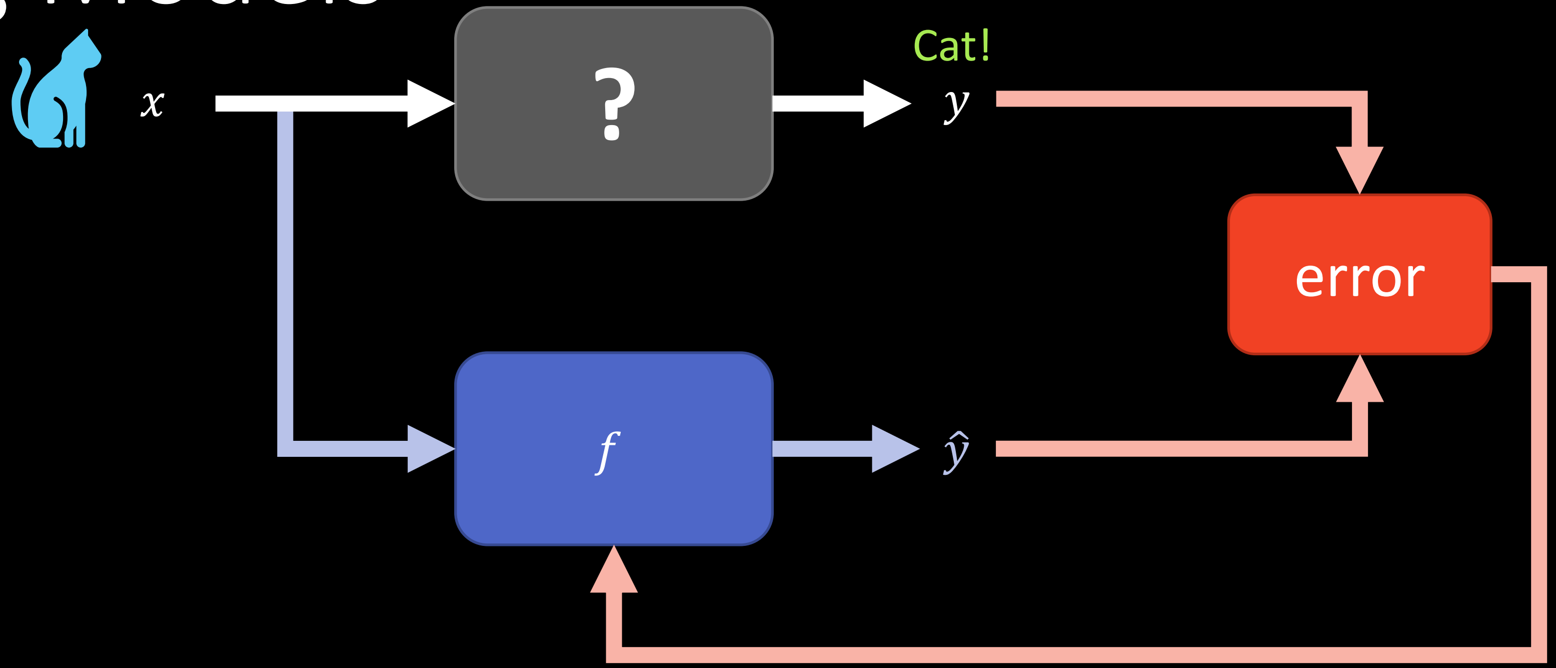
given samples: $\{(x_1, y_1), \ldots, (x_n, y_n)\}$

estimate f by min error in

Learning Models

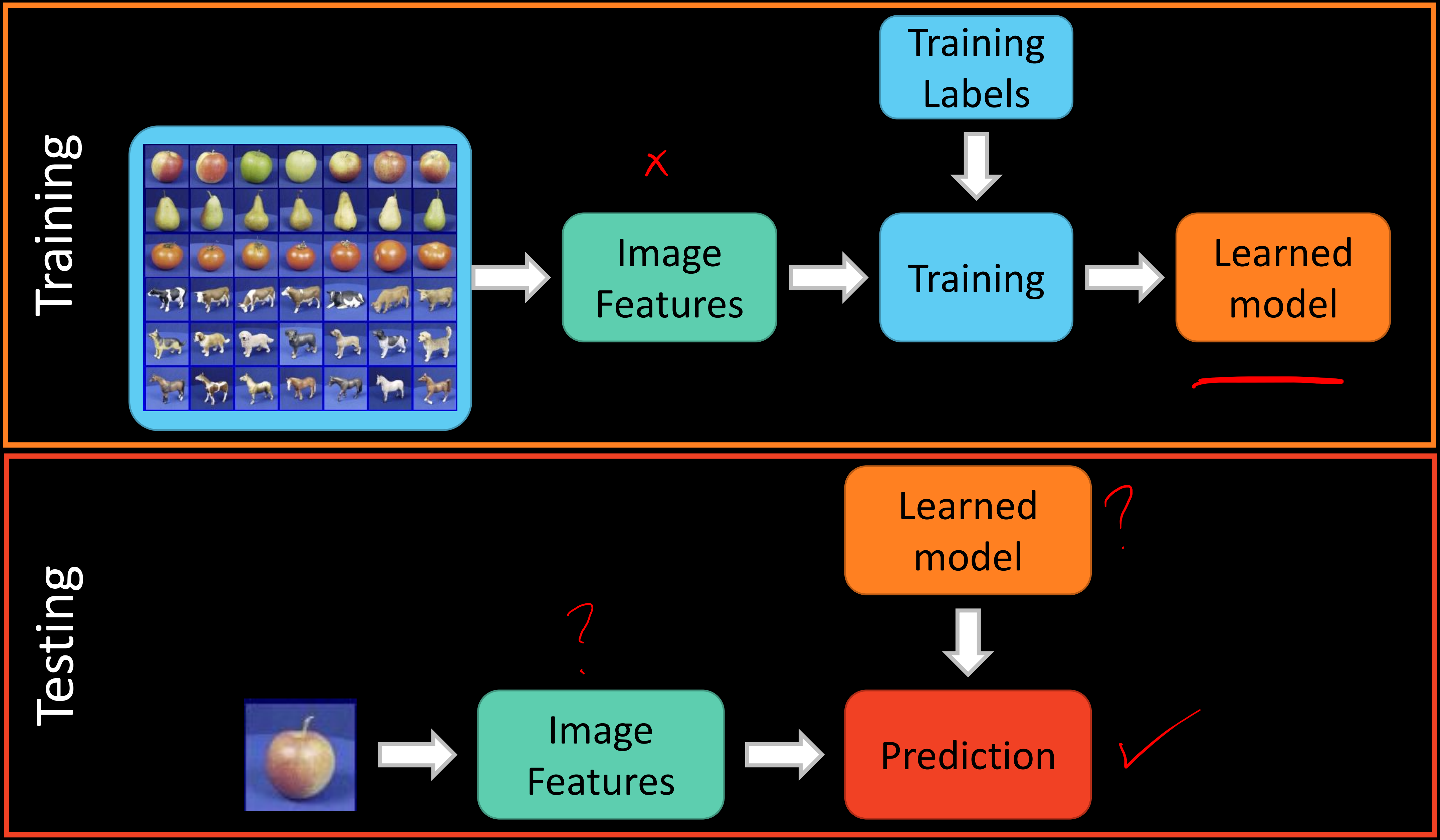


Learning Models



Learned models (like neural networks) are good when:

- Your system needs to learn and adapt
- Original is highly nonlinear / multi-variable
- Physics / model based approaches are not available or are too computationally expensive


Training
Training Labels
Image Features
Training
Learned model
Testing
Learned model
Image Features
Prediction

Naïve classification

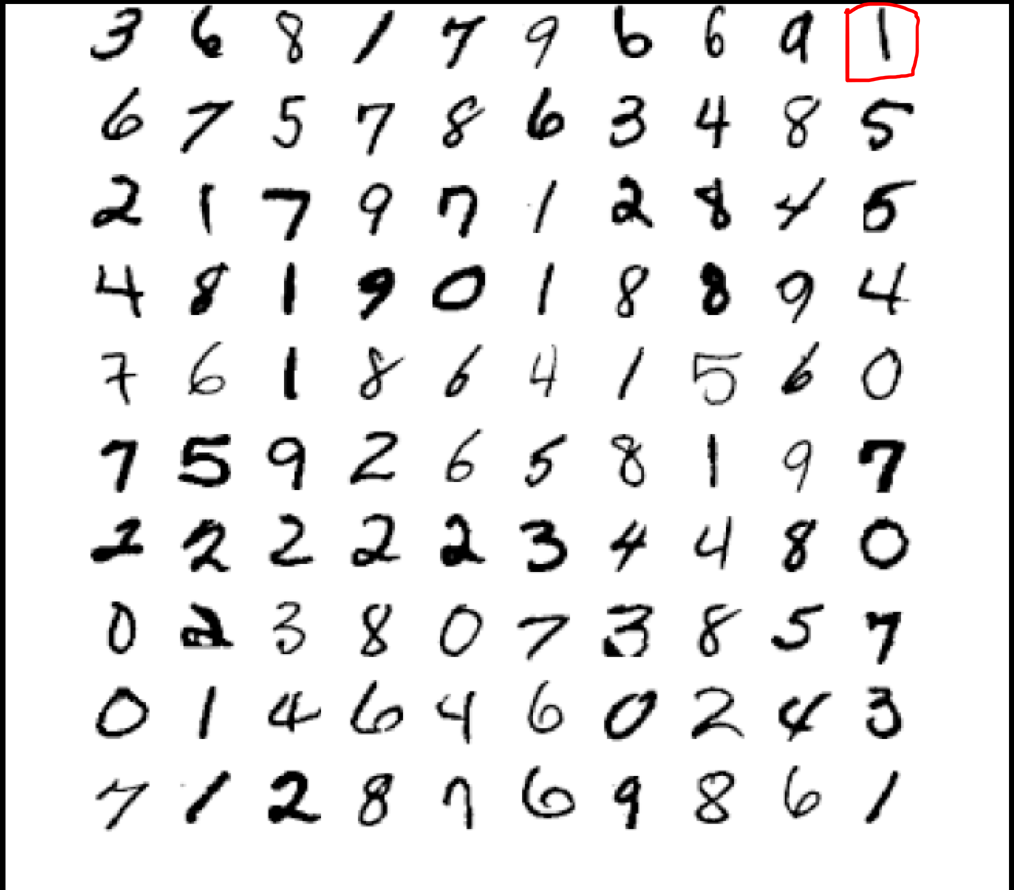
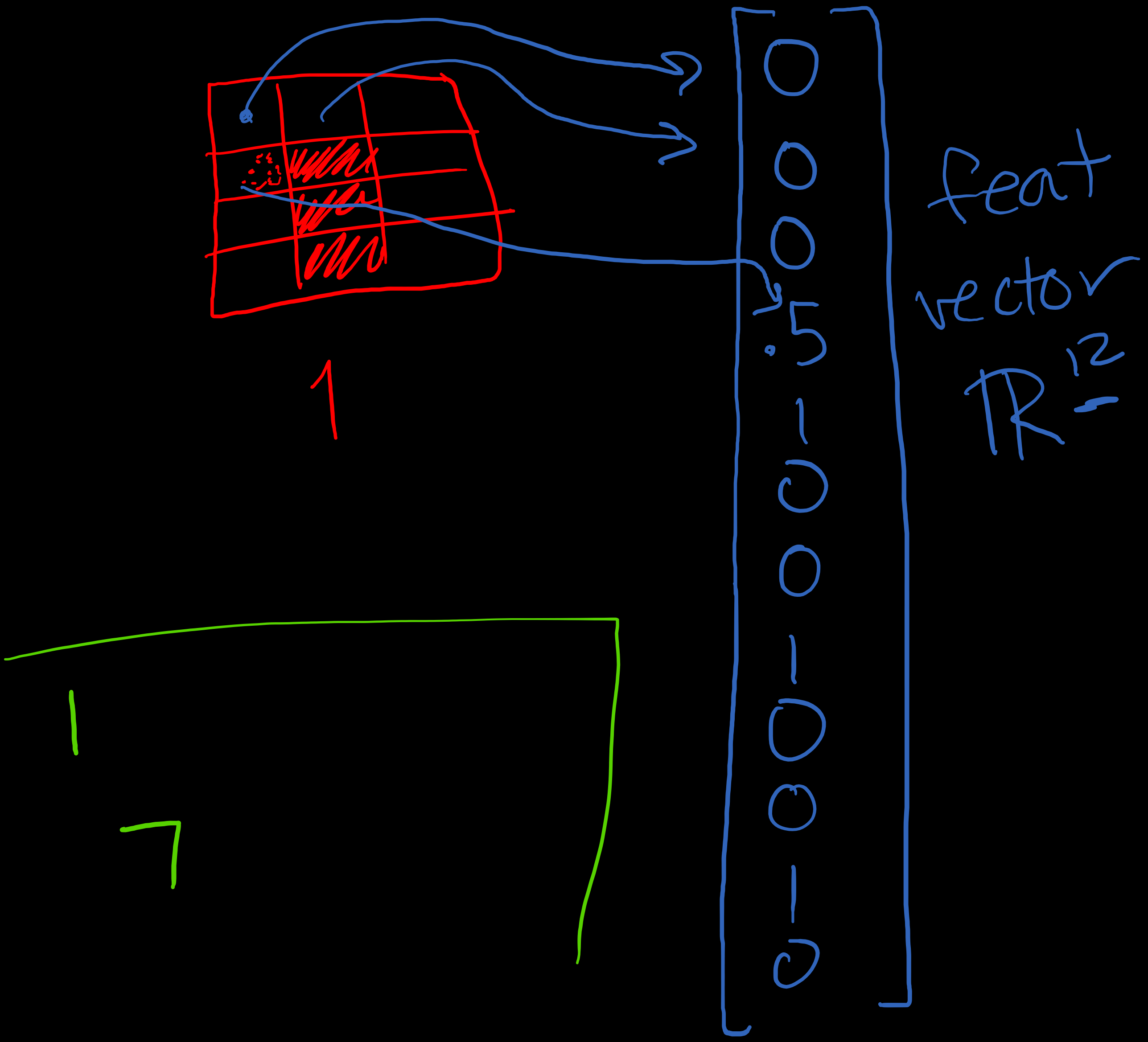
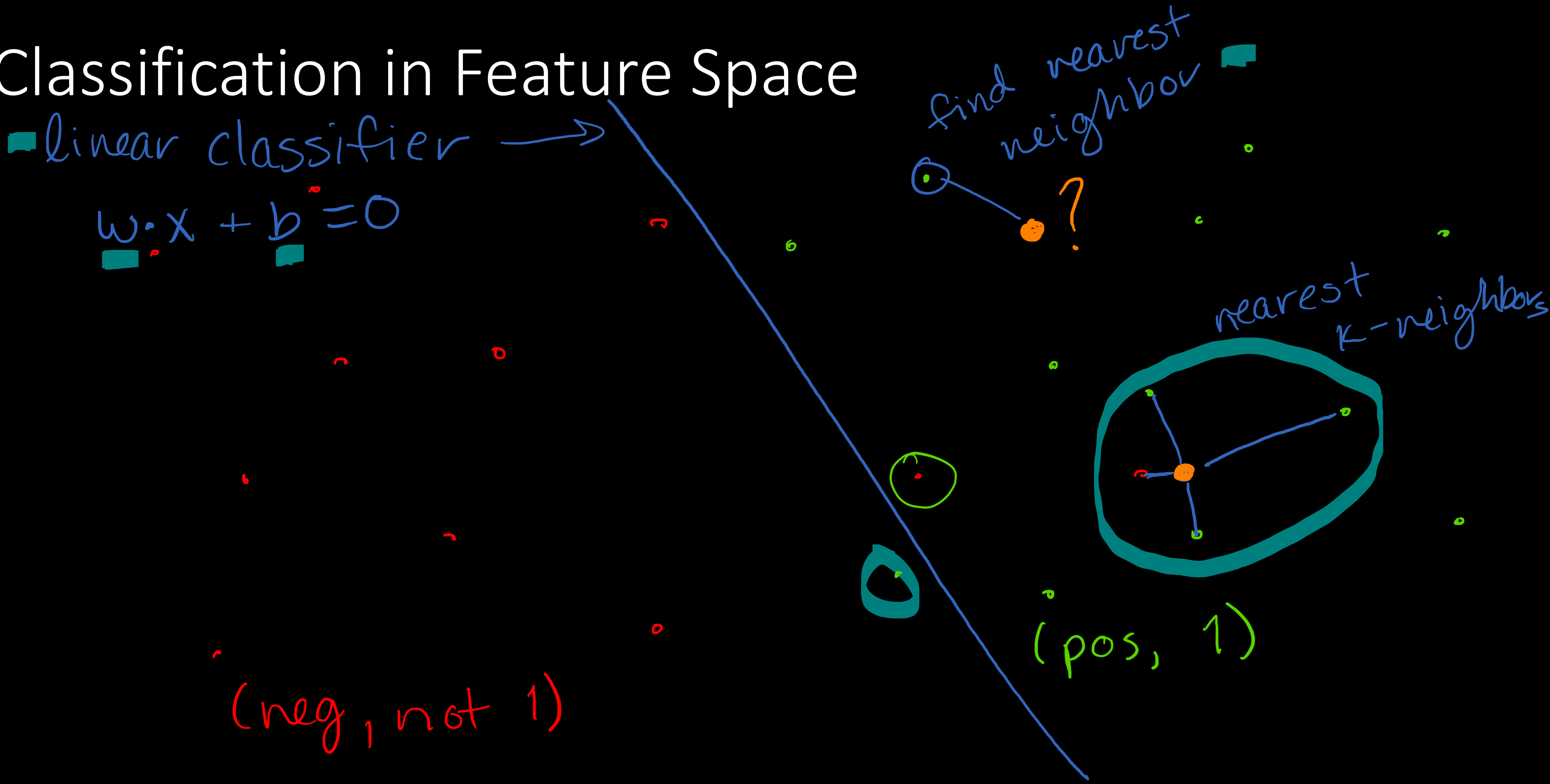


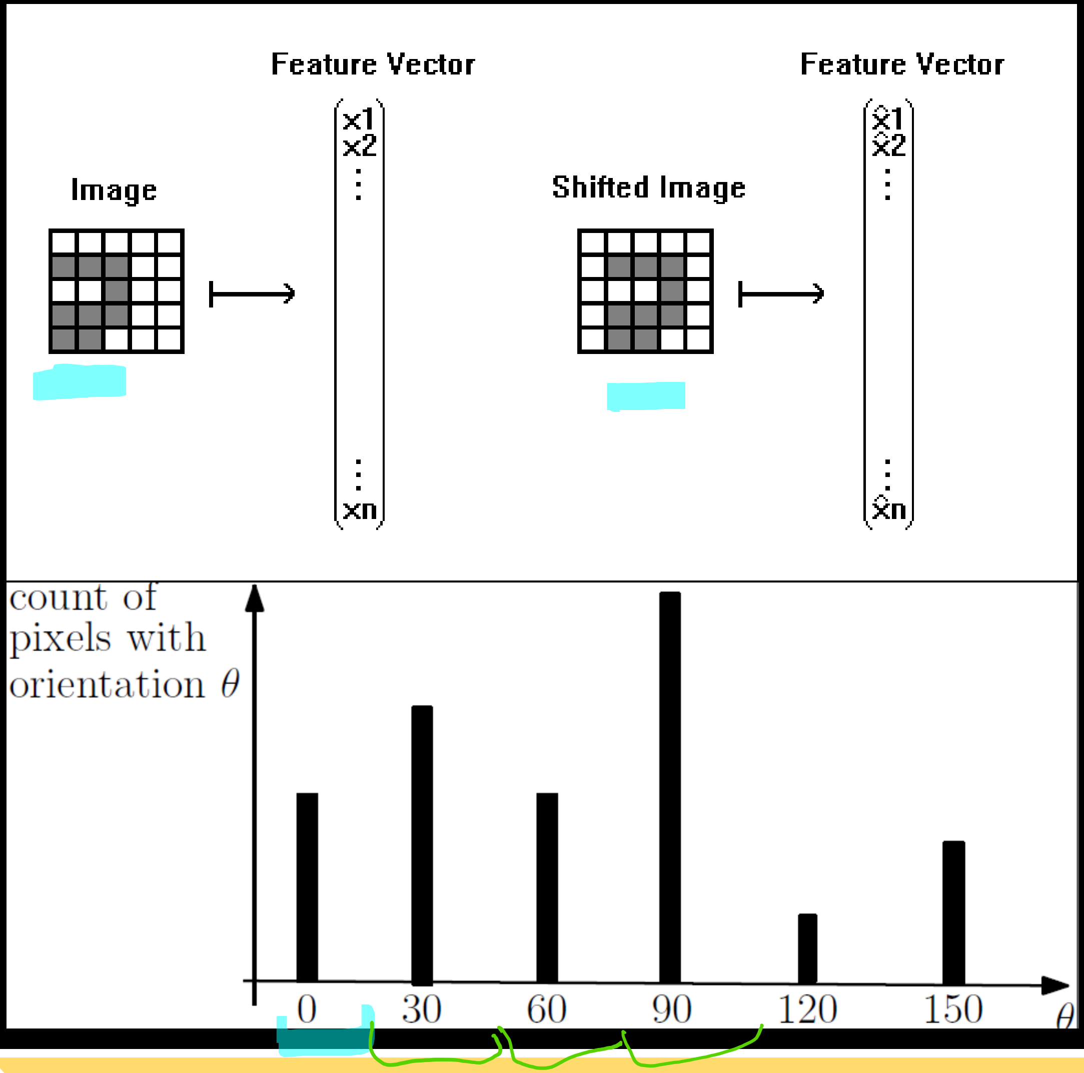
Fig. 4. Size-normalized examples from the MNIST database.



Classification in Feature Space



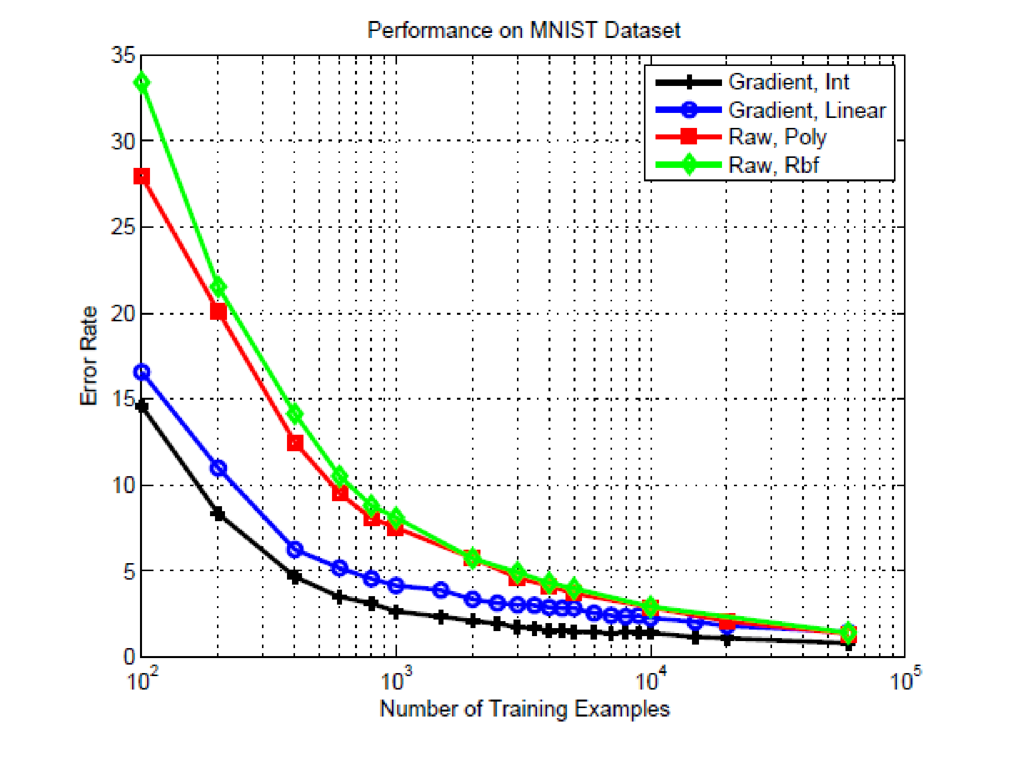
Naïve Features → Orientation Histograms



- Orientation histograms can be computed on blocks of pixels, so we can obtain tolerance to small shifts of a part of the object.
- For gray-scale images of 3d objects, the process of computing orientations, gives partial invariance to illumination changes.
- Small deformations when the orientation of a part changes only by a little causes no change in the histogram, because we bin orientations

Histogram of Oriented Gradients

Error rates vs. training examples

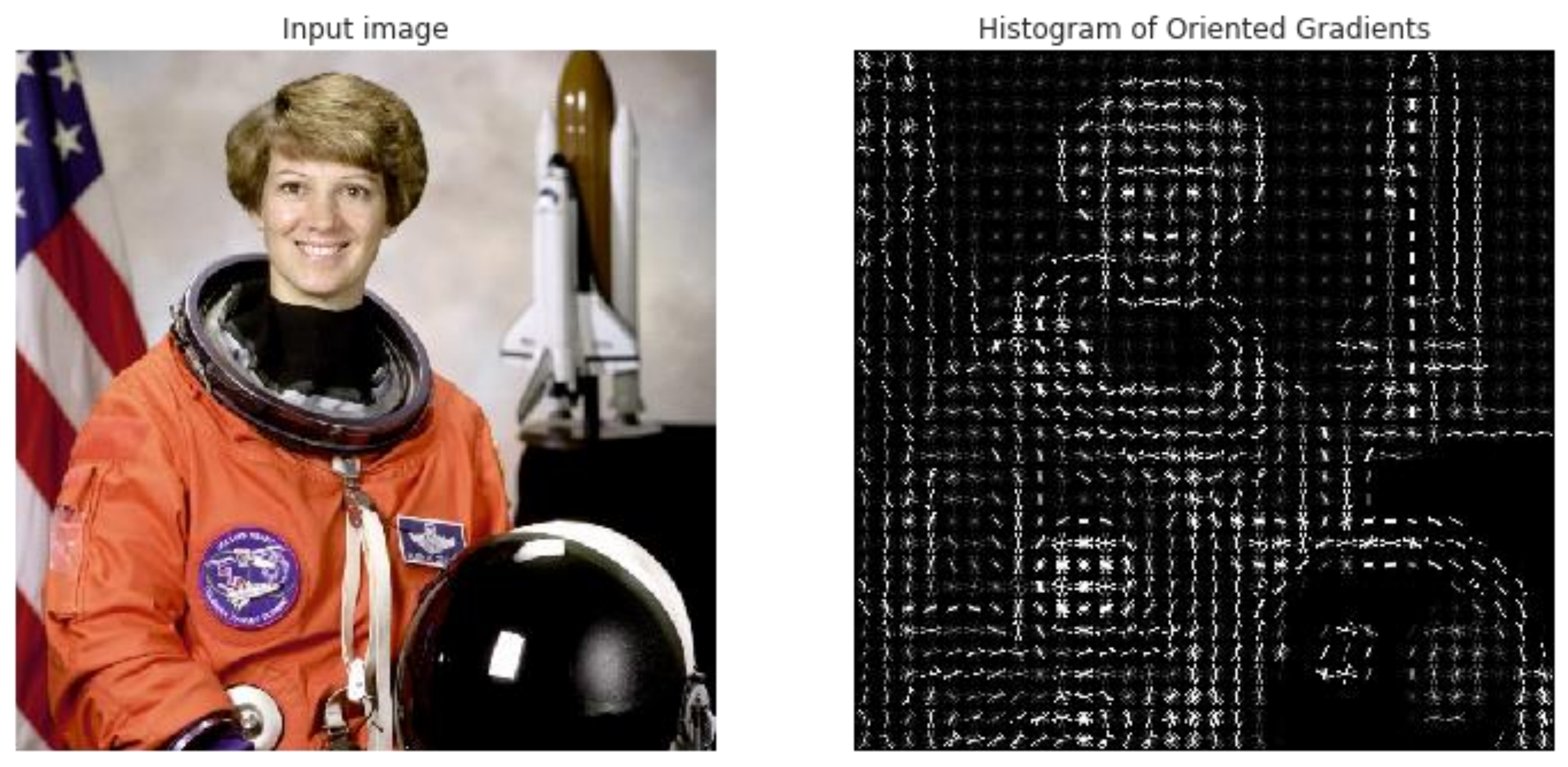


Misclassifications

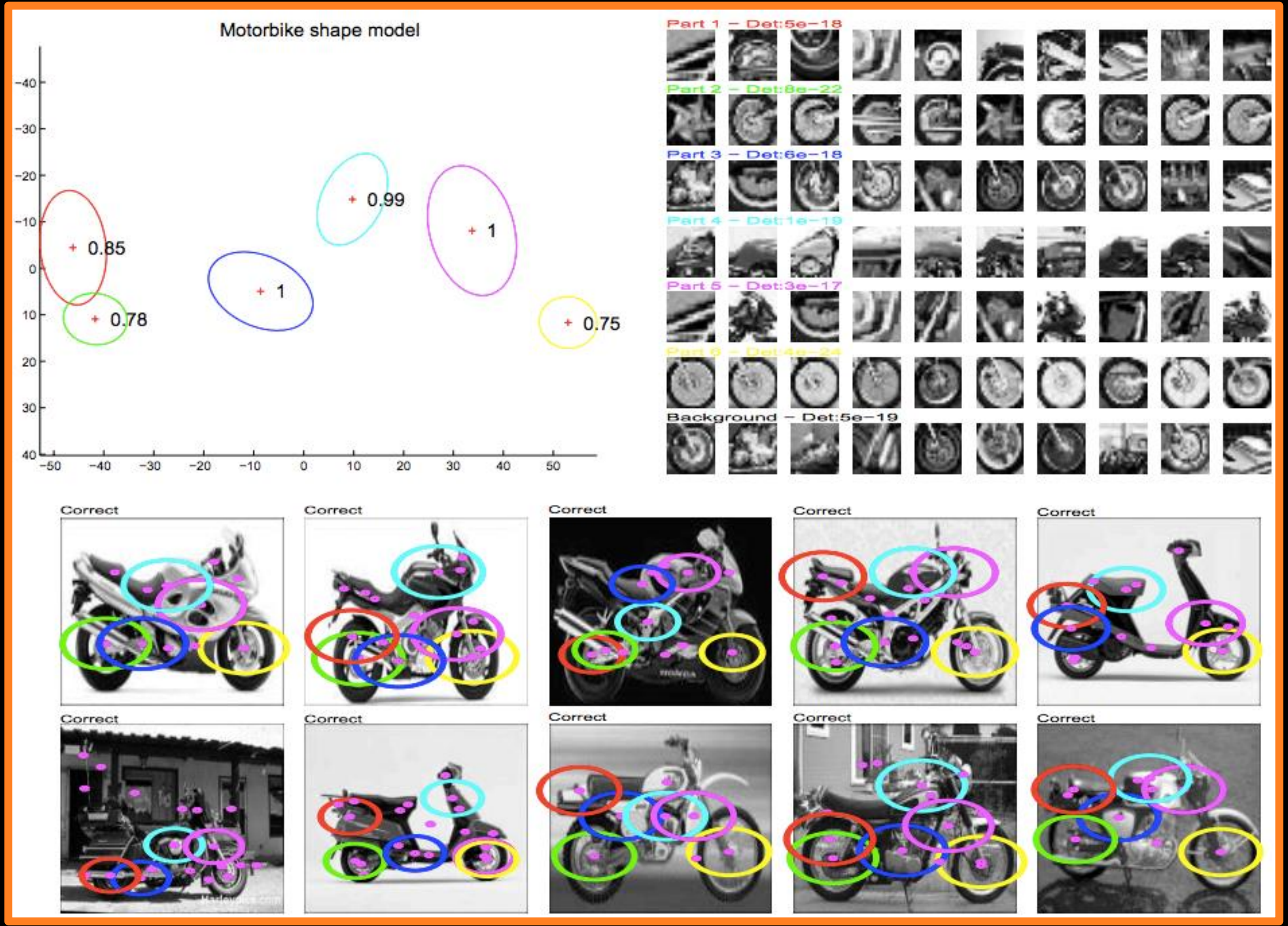
Slide Credit: J. Malik

$$\theta = \arctan\left(\nabla f_x / \nabla f_y\right)$$

Histogram of Oriented Gradients

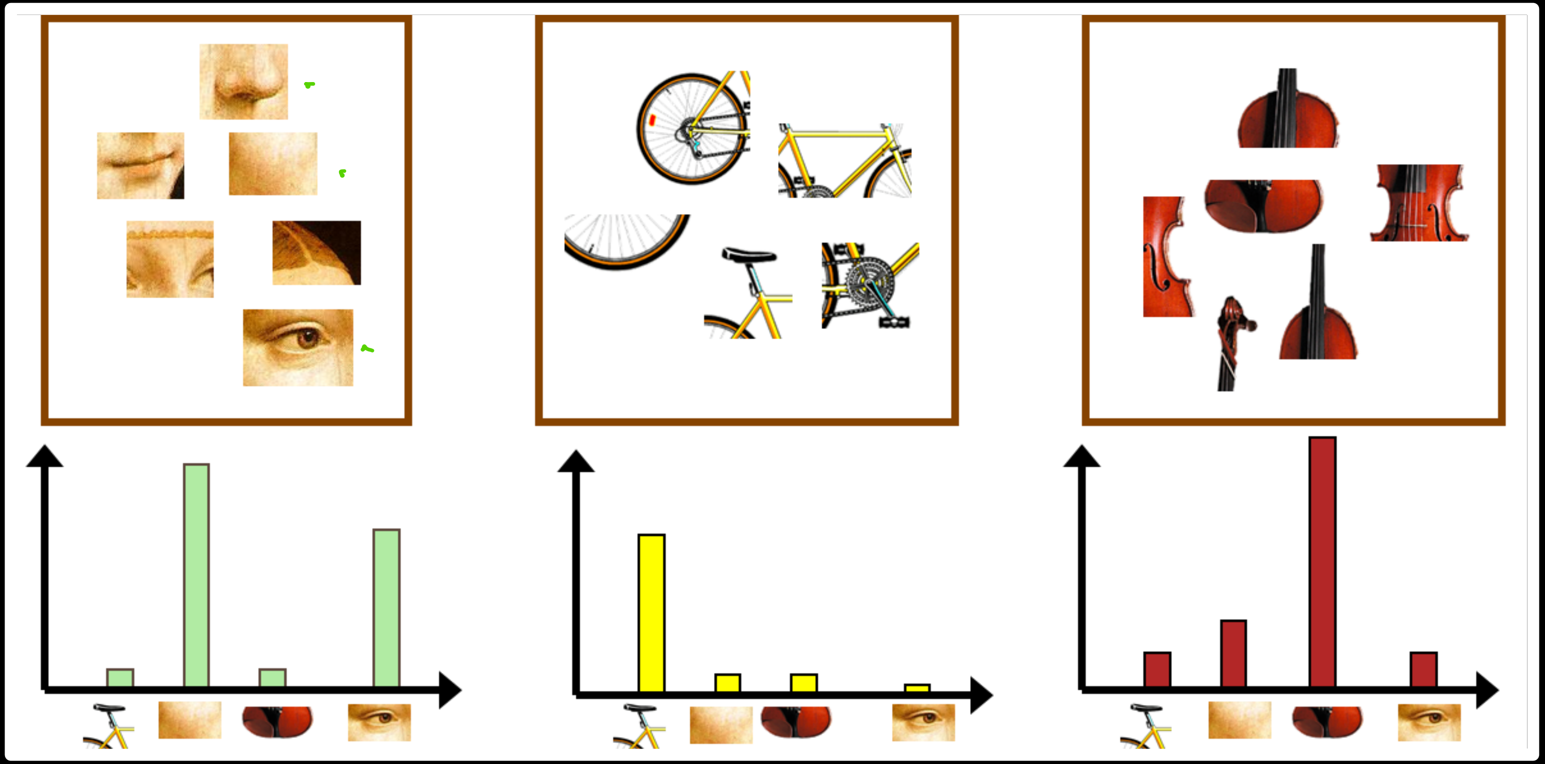


Part-based models

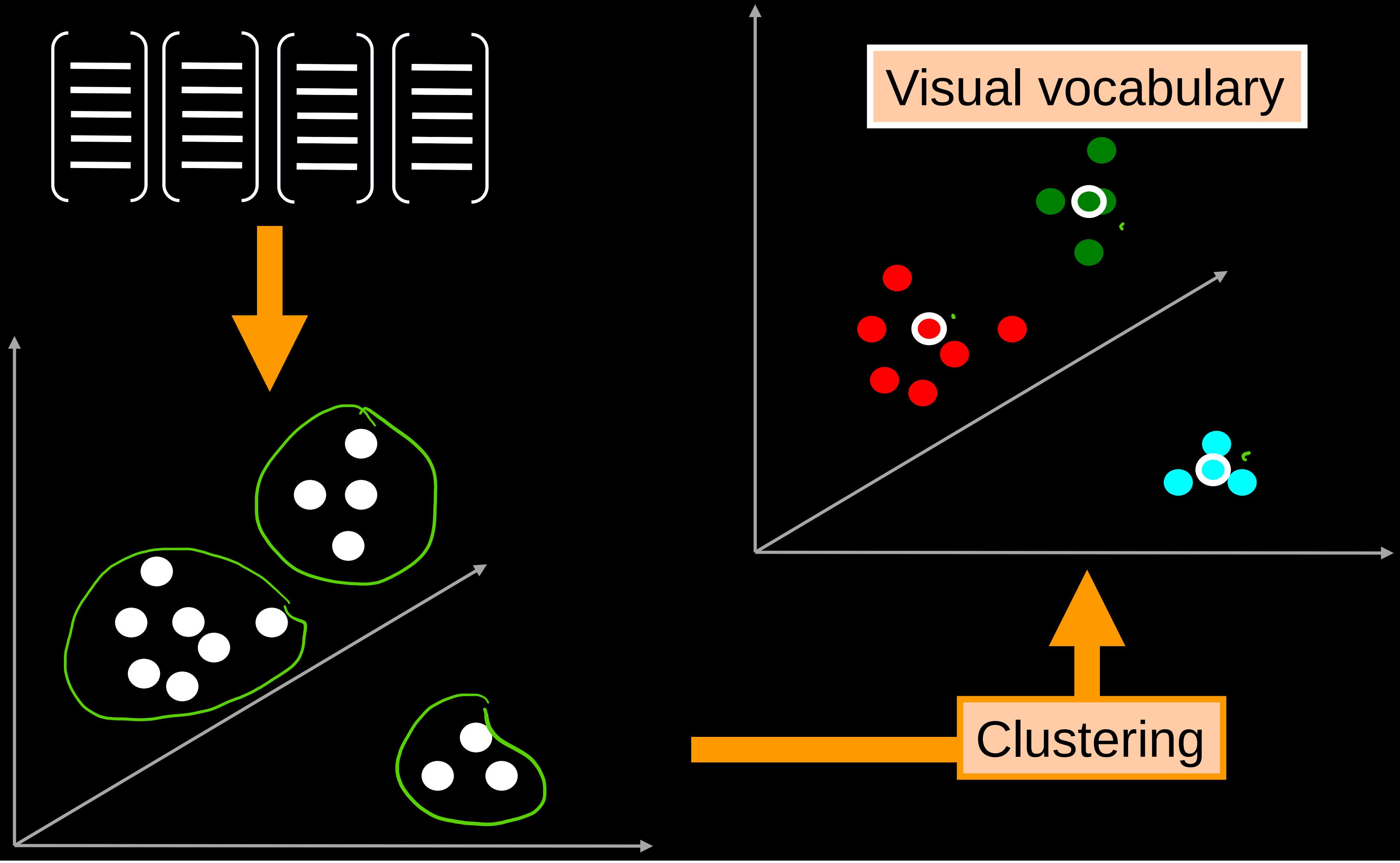


Bag of features

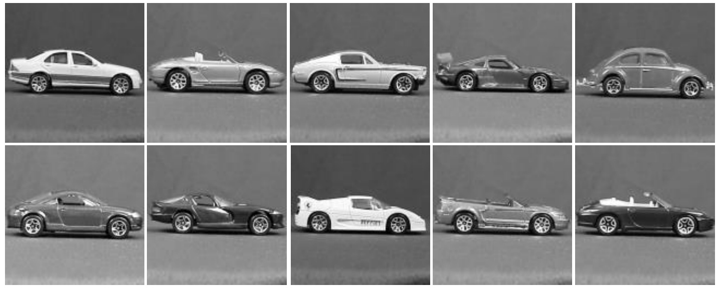
1. Extract local features
2. Learn “visual vocabulary”
3. Quantize local features using visual vocabulary
4. Represent images by frequencies of “visual words”

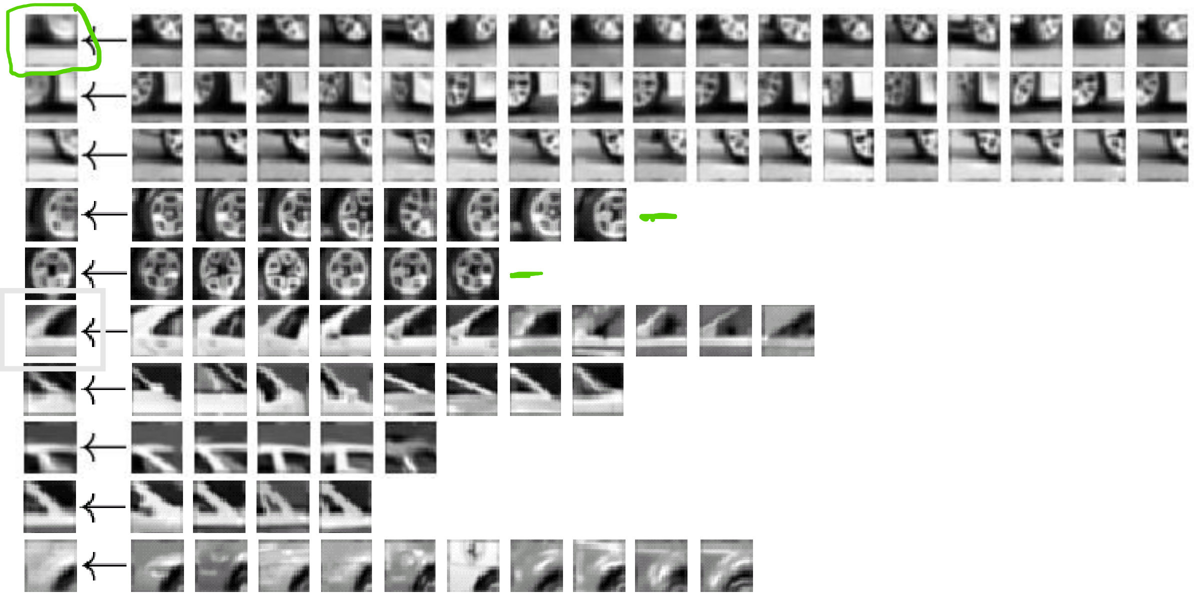


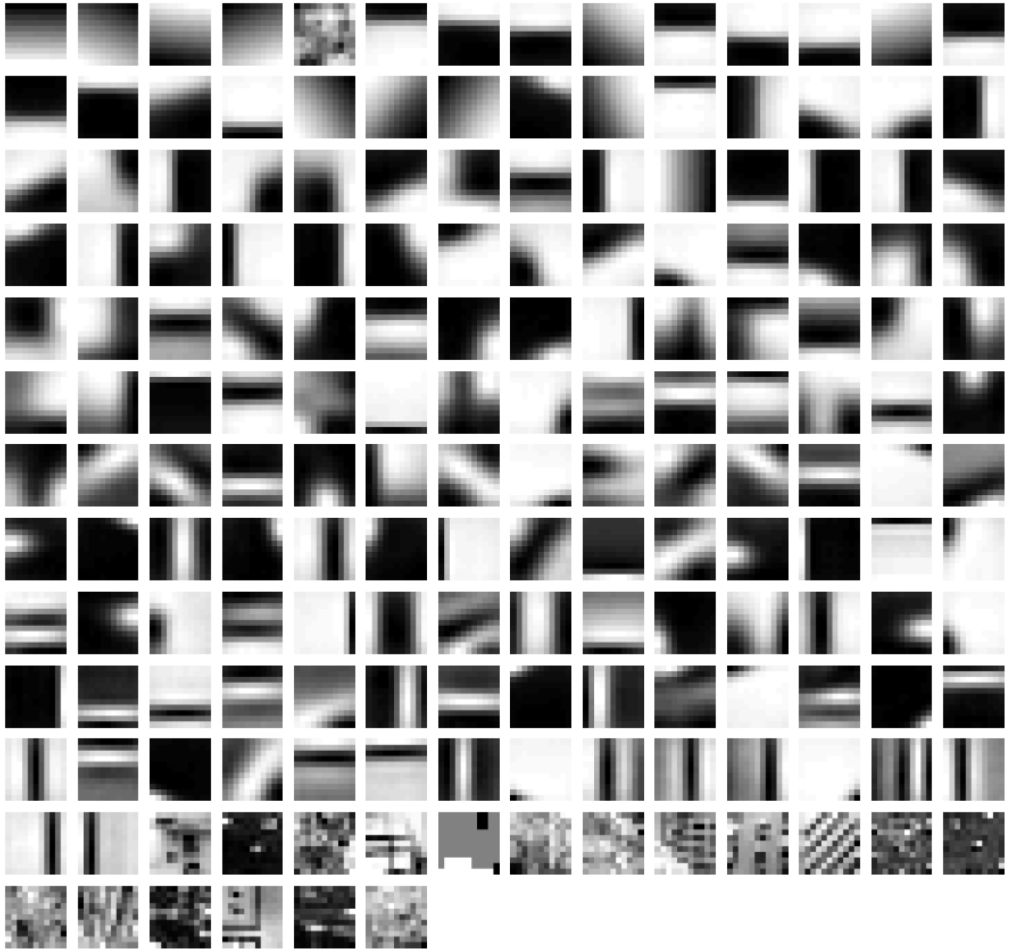
Learning a Visual Vocabulary



Example Visual Codebooks

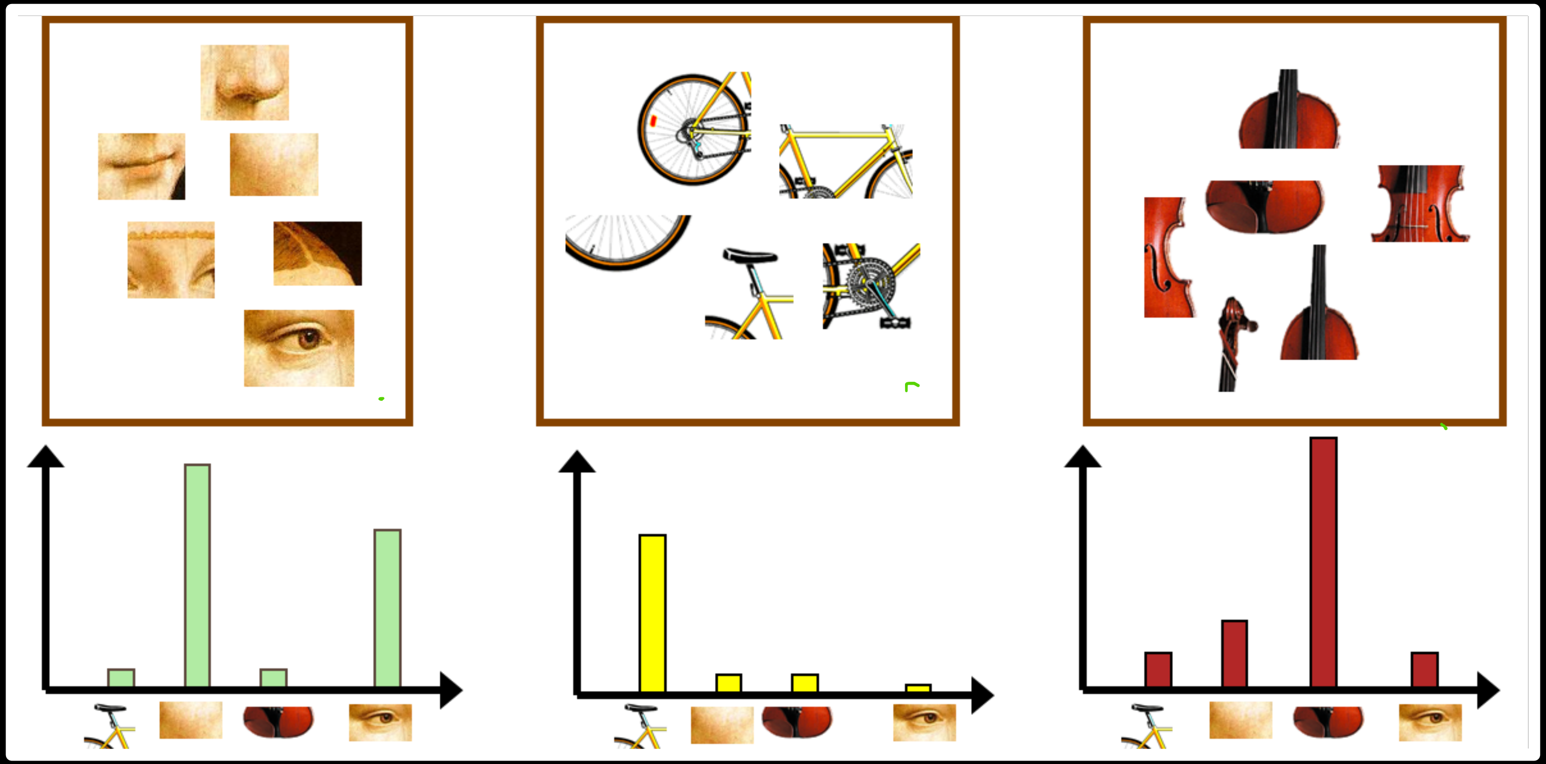






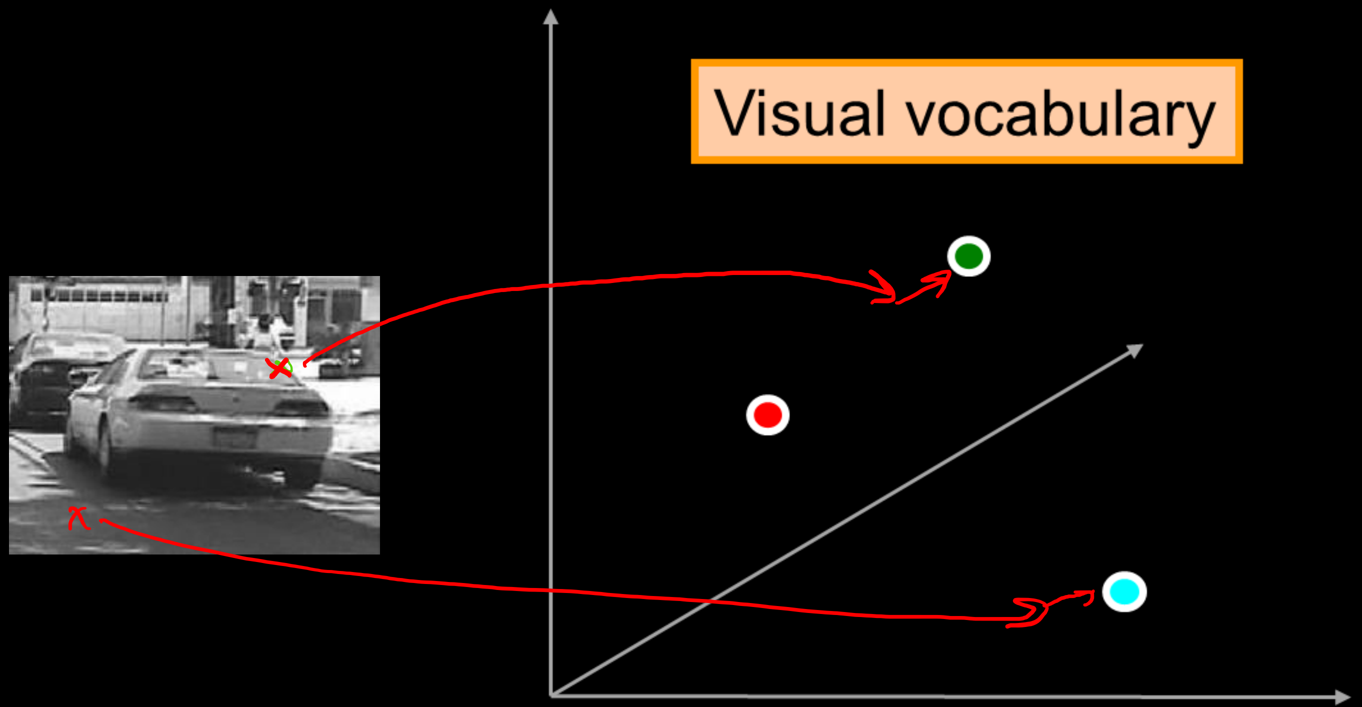
Bag of features

1. Extract local features
2. Learn "visual vocabulary"
3. Quantize local features using visual vocabulary
4. Represent images by frequencies of "visual words"

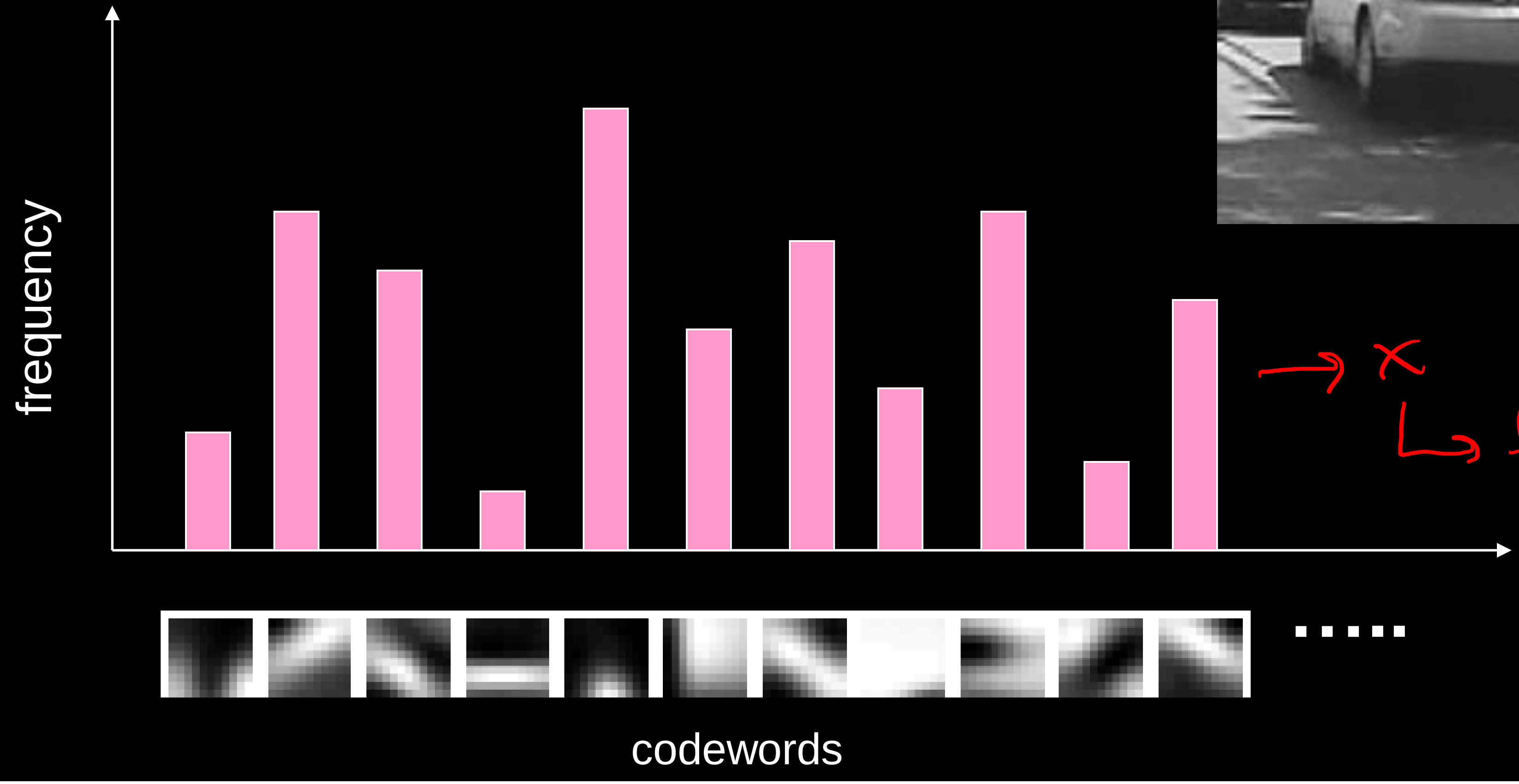


Bag of features

1. Extract local features
2. Learn “visual vocabulary”
3. Quantize local features using visual vocabulary
4. Represent images by frequencies of “visual words”



Images as Histogram of Patches



source: Svetlana Lazebnik

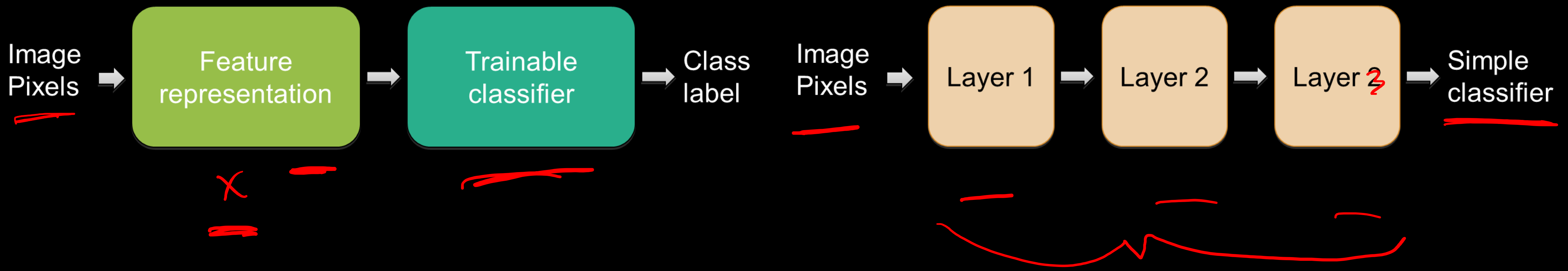
Today’s Plan

- Computer vision overview
- Object recognition
  - Feature representations
  - Classification
- (Convolutional) Neural Networks

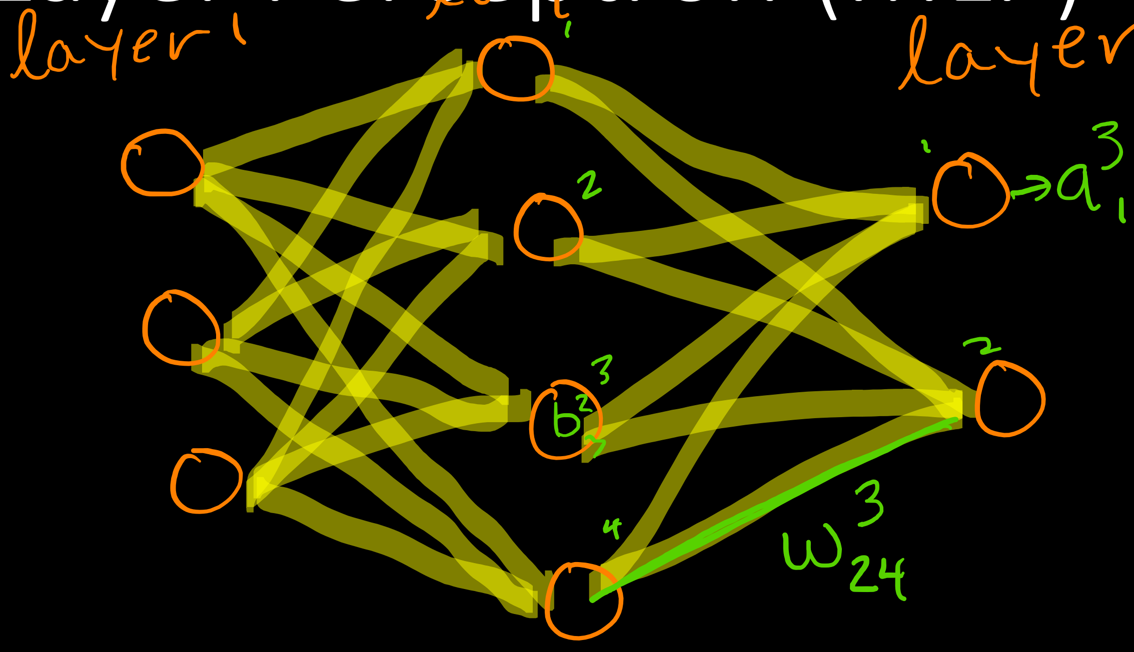
From Shallow to Deep Learning

Traditional “Shallow” Pipeline

“Deep” Recognition Pipeline



Multi-Layer Perceptron (MLP)



activation:

$$a_j^l = f\Big(\underbrace{\sum_k w_{jk}^l \cdot a_k^{l-1} + b_j^l}_{z_j^l}\Big)$$

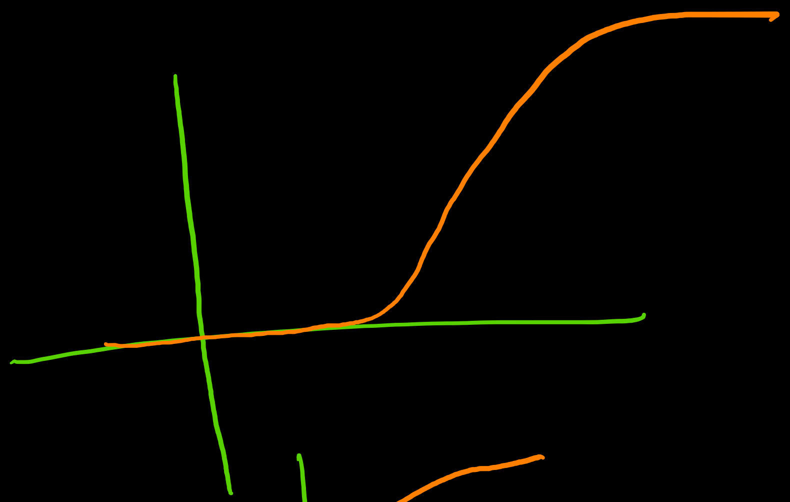
$$= f(w^l a^{l-1} + b^l)$$

weights: w_{jk}^l ← from $l-1$ (k); from l (j)

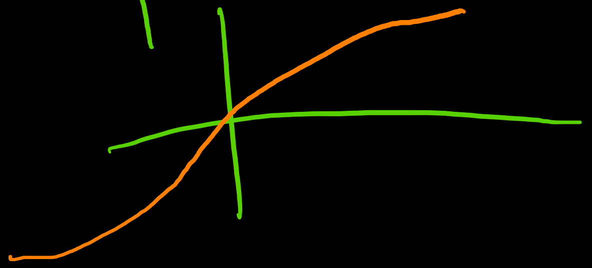
bias: b_j^l

Activation Functions

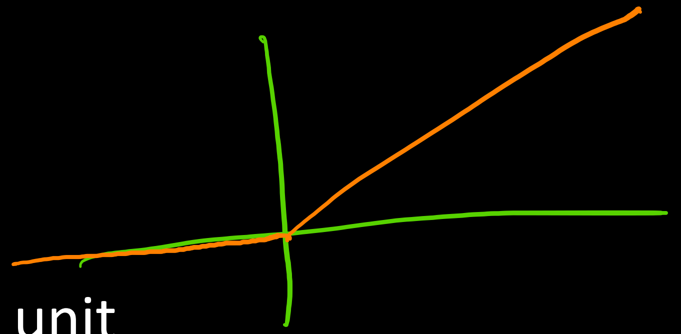
- Sigmoid
  - Homage to the original formulation
  - Not very popular nowadays as they tend to saturate and kills gradients



- Tanh
  - This is a scaled sigmoid and is almost always preferred



- ReLU – Rectified Linear Unit
  - Fast computation, doesn't saturate, might lead to better convergence rates
  - Tends to be fragile in training

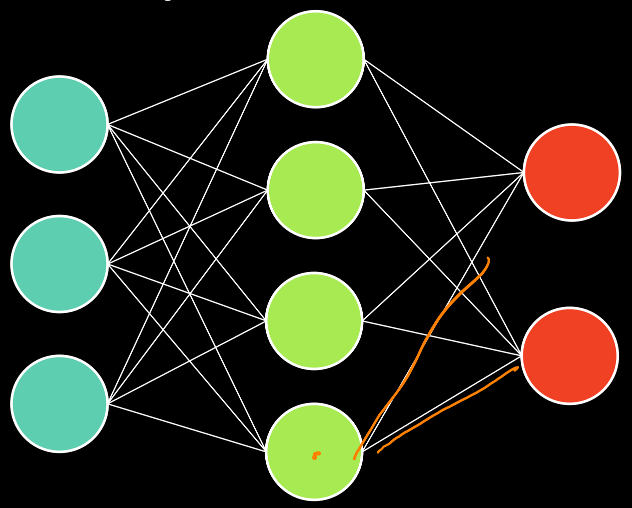


- Maxout: $\max(w_1^\top x + b_1, w_2^\top x + b_2)$
  - Extension of ReLU that does not die
  - Doubles the number of parameters for every unit

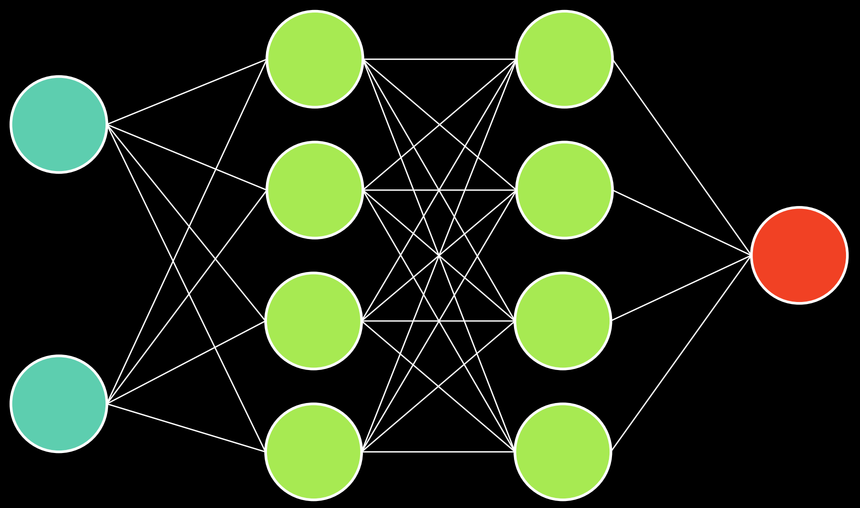
Network Architectures and Sizes

For regular networks, most commonly use fully connected network

Sizing of networks determined by layers, number of units, and/or number of parameters



6 units, 6 biases
[3*4]+[4*2]=20 weights
26 learnable parameters



9 units, 9 biases
[2*4]+[4*4]+[4*1]=28 weights
37 learnable parameters

For context, convolutional networks typically have on the order of 100 million parameters

Universal Function Approximators

Let $\varphi : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded, and continuous function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. The space of real-valued continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, \ldots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

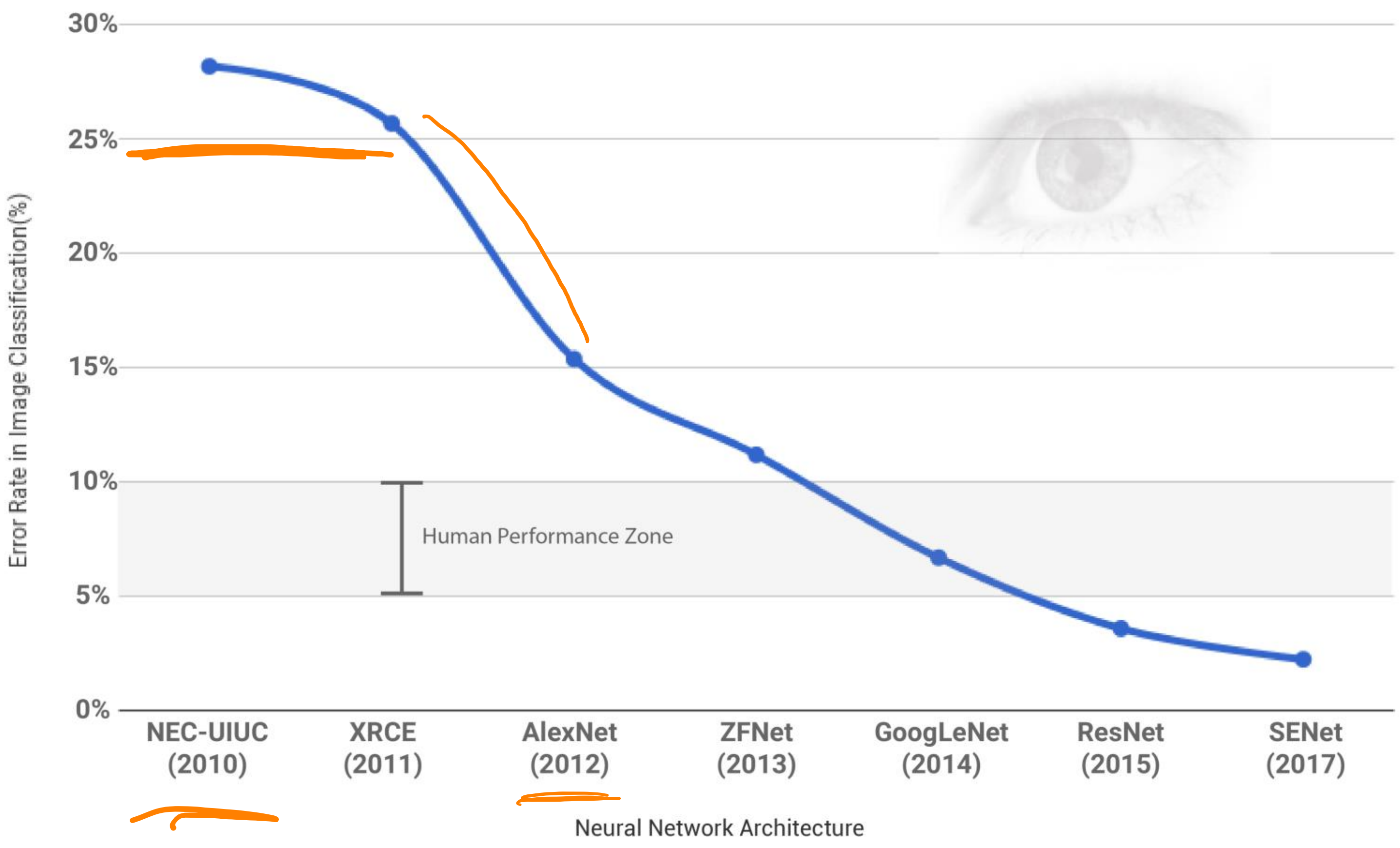
as an approximate realization of the function f; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

A feedforward network with a single hidden layer containing a finite number of units can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function.

Classification Improvements

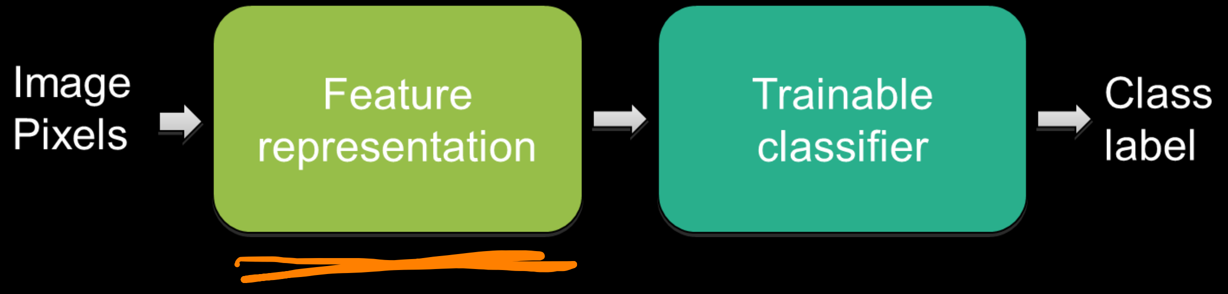


Neural Networks

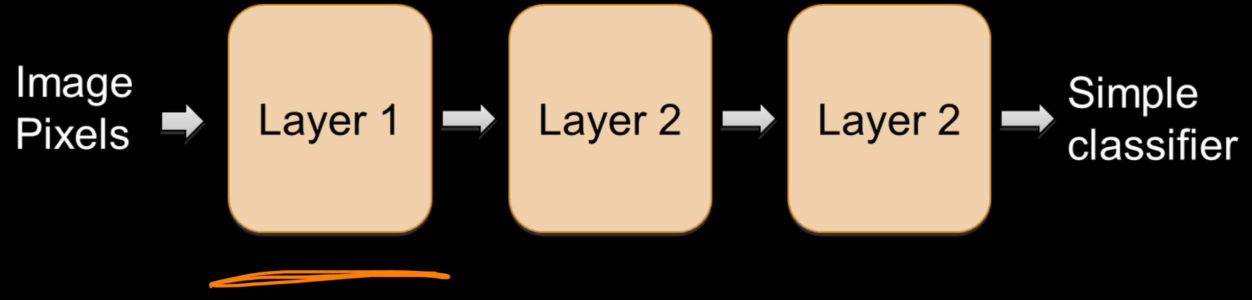
- Pros:
  - + Flexible and general function approximation framework
  - + Generally successful is high dimensional and model free problems
- Cons
  - Very few theoretical guarantees
  - Training is prone to local optima and unstable
  - Large amount of training data and computing power are required
  - Huge variety of implementation choices need to be hand tuned (network architectures, parameters, etc.)

From Shallow to Deep Learning

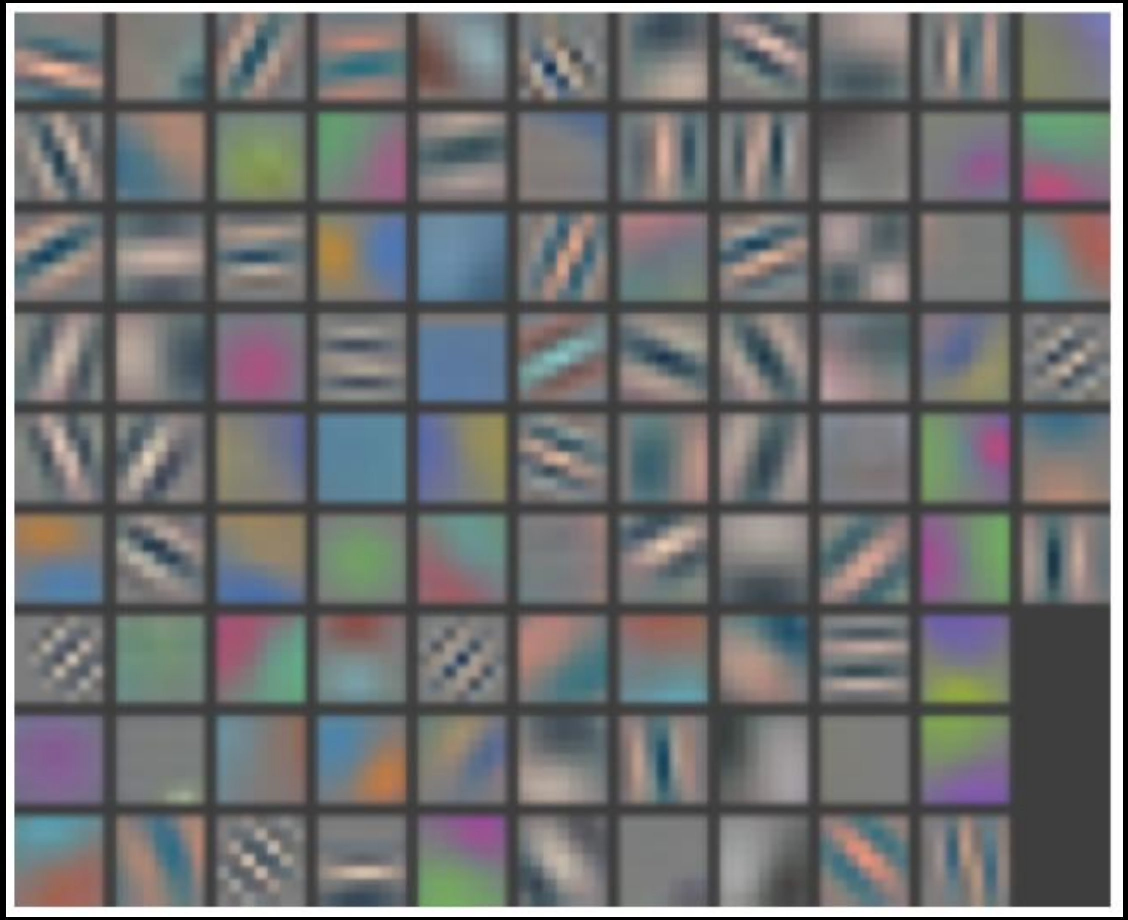
Traditional "Shallow" Pipeline

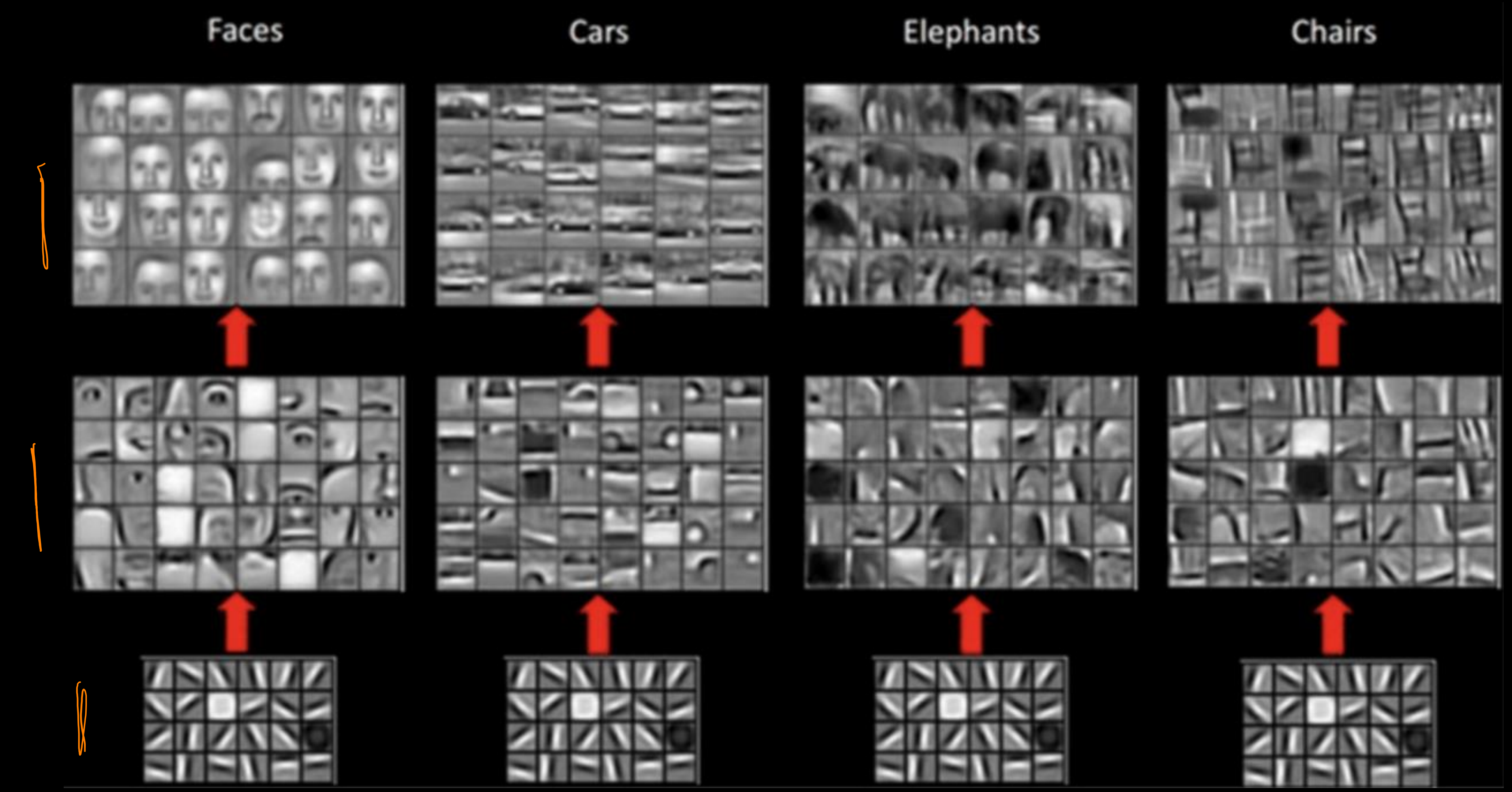


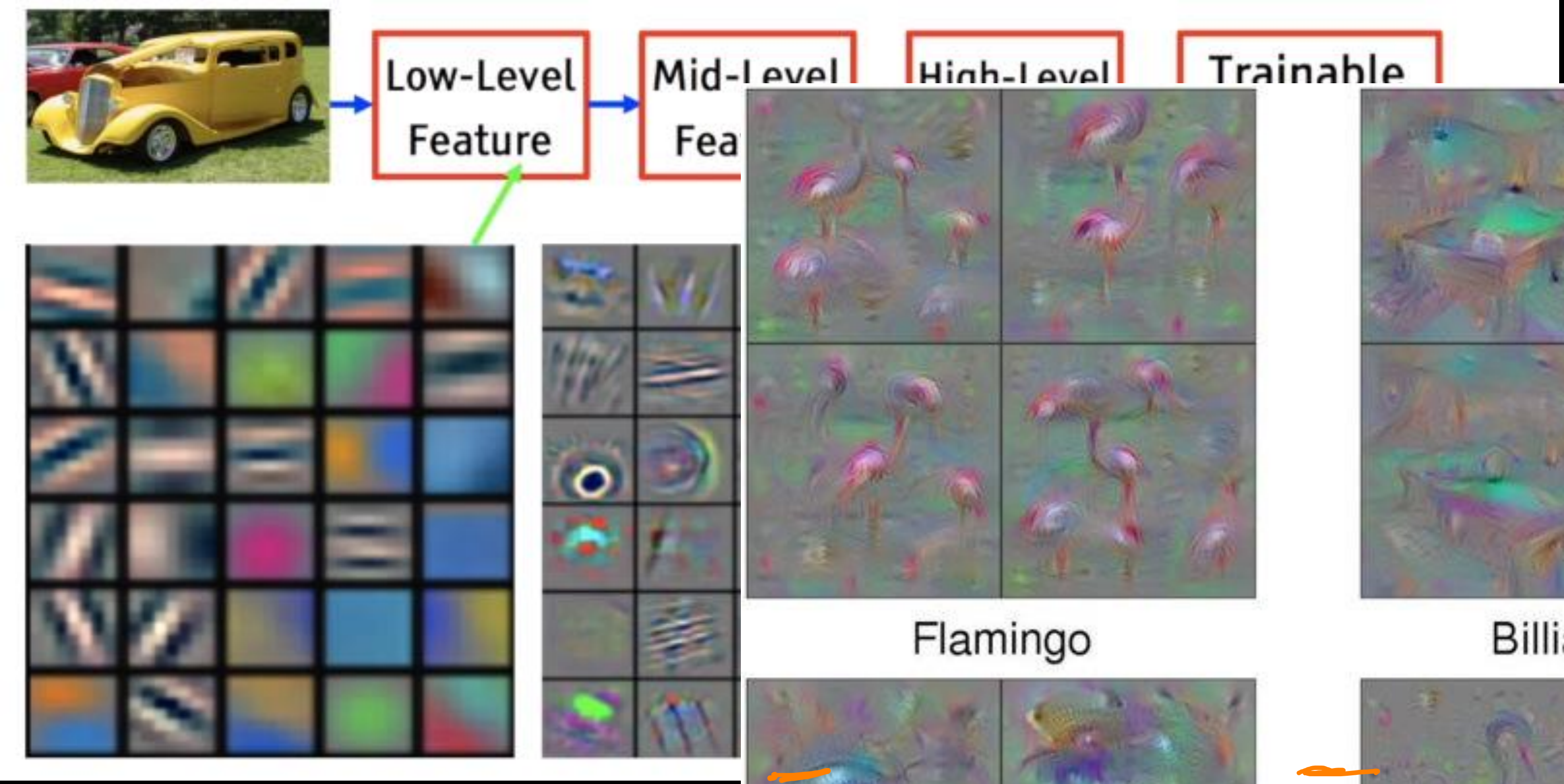
"Deep" Recognition Pipeline

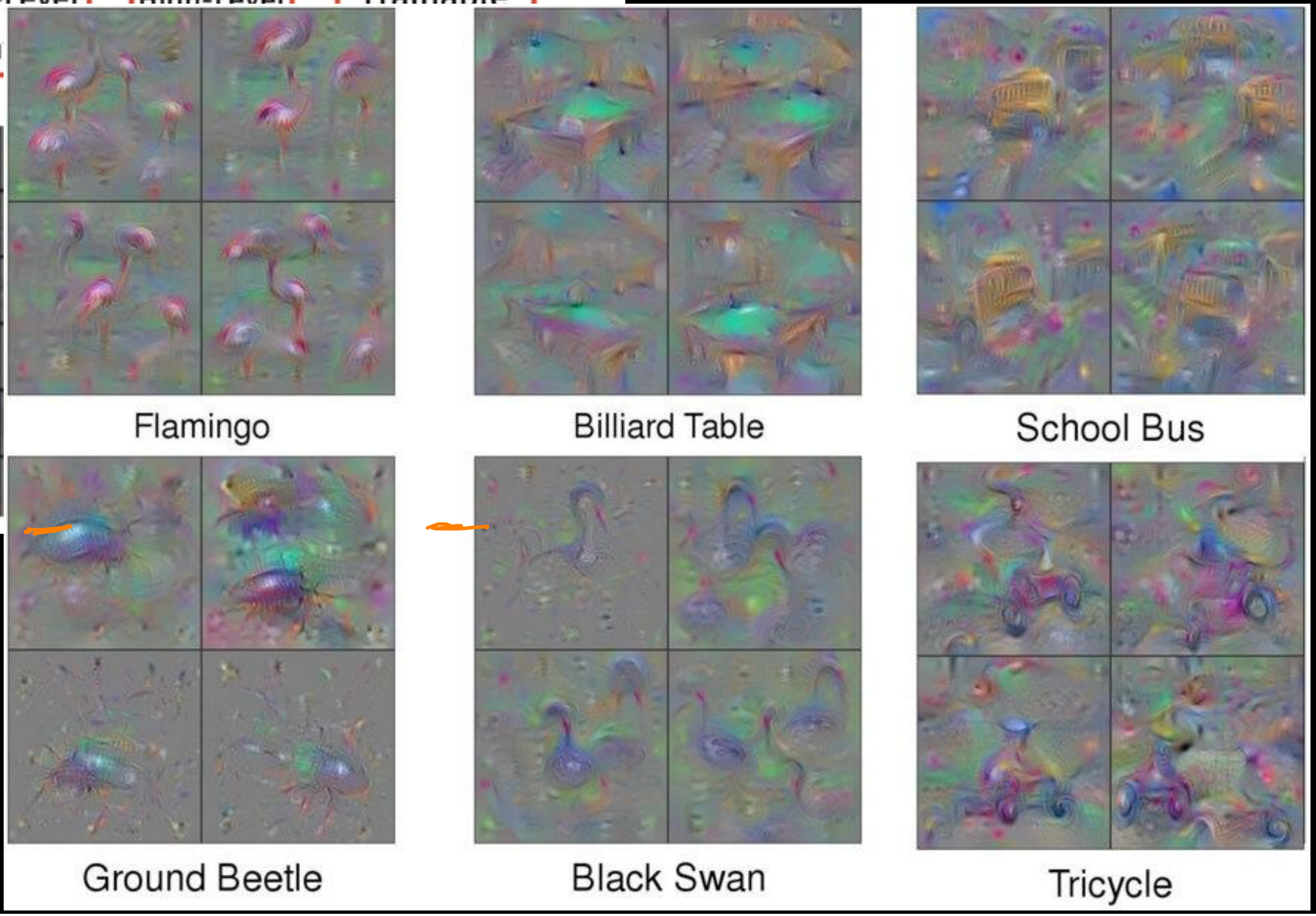


Layers as filters



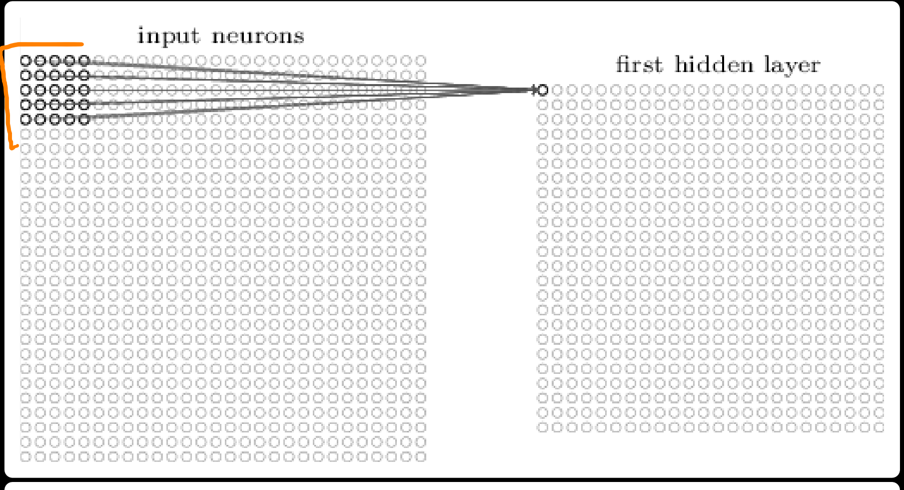

Faces
Cars
Elephants
Chairs

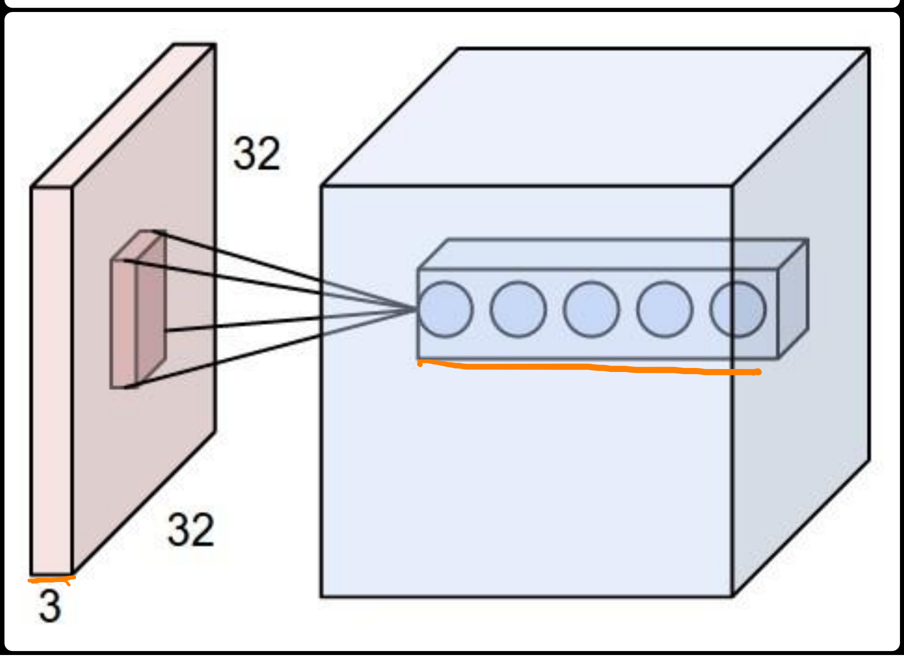

Low-Level Feature
Mid-Level Fea
High-Level
Trainable


Flamingo
Billiard Table
School Bus
Ground Beetle
Black Swan
Tricycle

Convolutional Layers

- Each unit has a receptive field that connects it to a small local region of the input
  - If all units in a depth slice use identical weights, then the forward pass of this layer can be computed as a *convolution* of the weights with the input volume
- Each conv layer acts like a learnable filter that activates for some type of visual feature (e.g., edge, corner, eye, cat)
- Recall: large ConvNets have *a ton of* parameters
  - Parameter sharing restricts the weights along one *slice* of the depth, reducing the parameters down to ~35,000 (see first point)





Convolutional Network Components

ConvNets transform the original image layer by layer from the original pixel values to the final class scores

This is done via convolutional layers, pooling, ReLUs, and fully connected (FC) layers

- Conv/FC layers perform transformations that are a function of trainable parameters
  - Ex] CIFAR-10 images are size 32x32x3, so one fully-connected unit in a first hidden layer of a regular NN would have 32*32*3 = 3072 weights
- ReLU/Pool layers are fixed and not trained

Pooling Layers

The goal of pooling is to progressively reduce the spatial size of the representation and the amount of parameters and computation

- operates independently on depth slice and resizes it spatially, often using the max operation

Generally speaking:

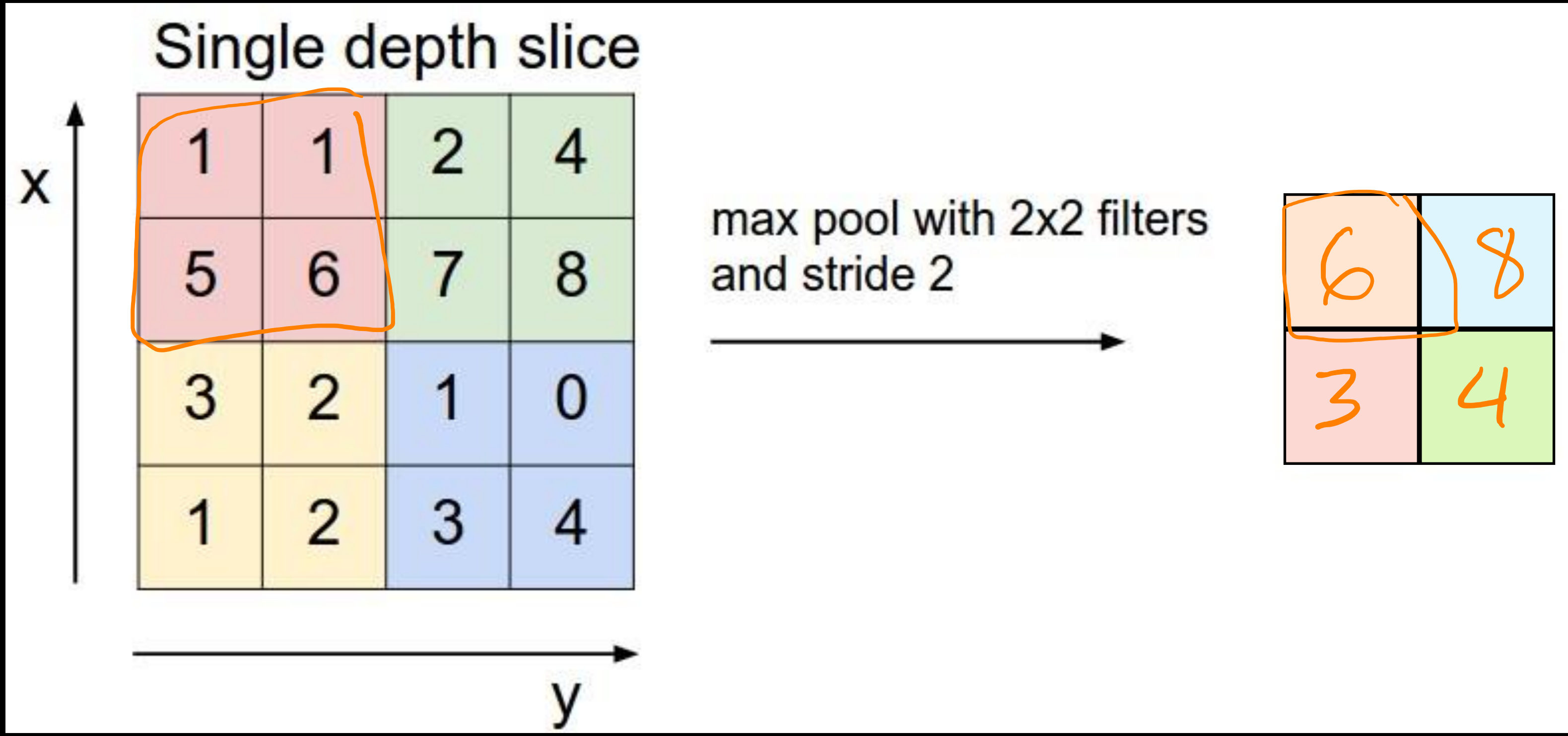
- Accepts a volume of size W1×H1×D1
- Requires two hyperparameters: their spatial extent F, the stride S,
- Produces a volume of size W2×H2×D2 where:

  W2=(W1−F)/S+1

  H2=(H1−F)/S+1

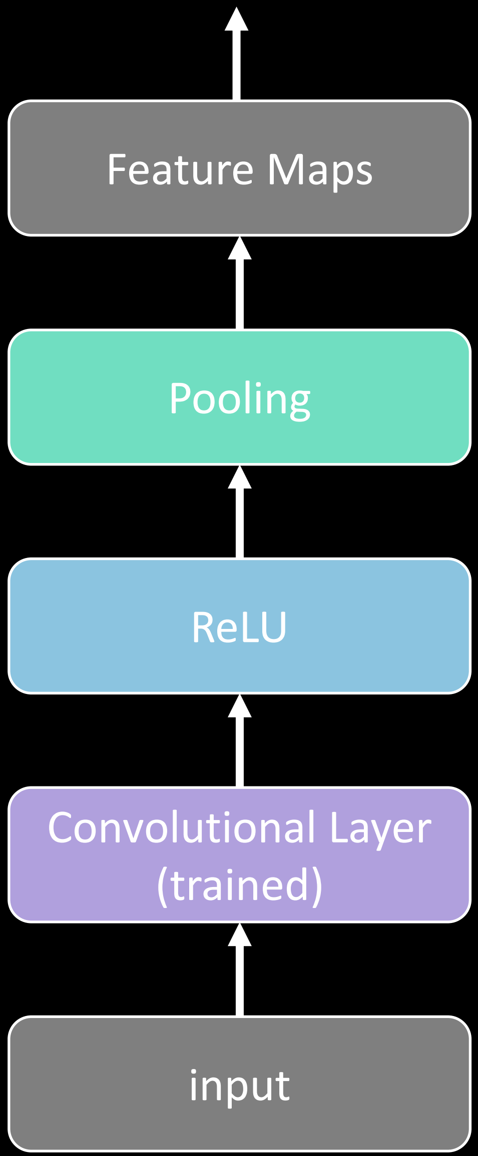
  D2=D1

Pooling



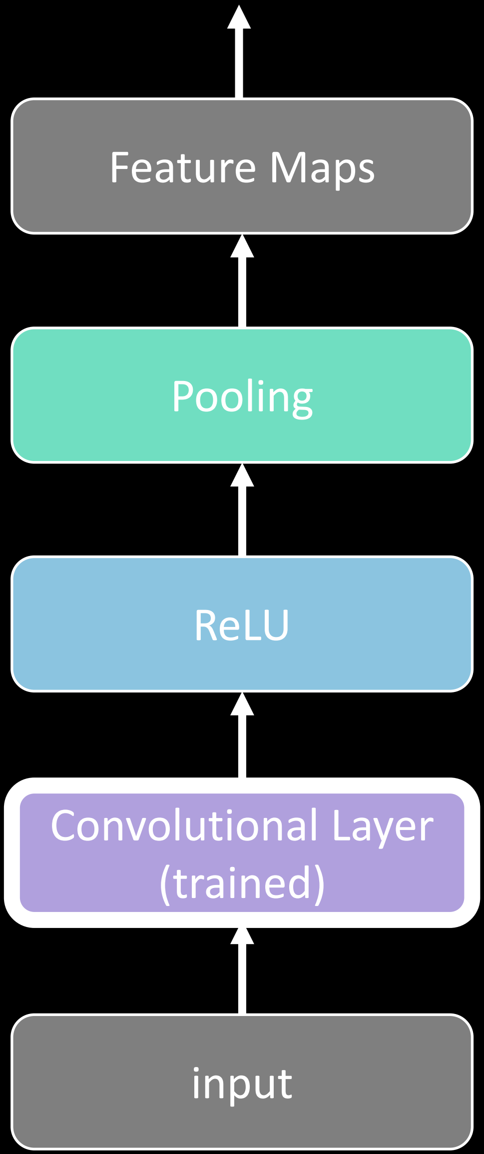
http://cs231n.github.io/convolutional-networks/

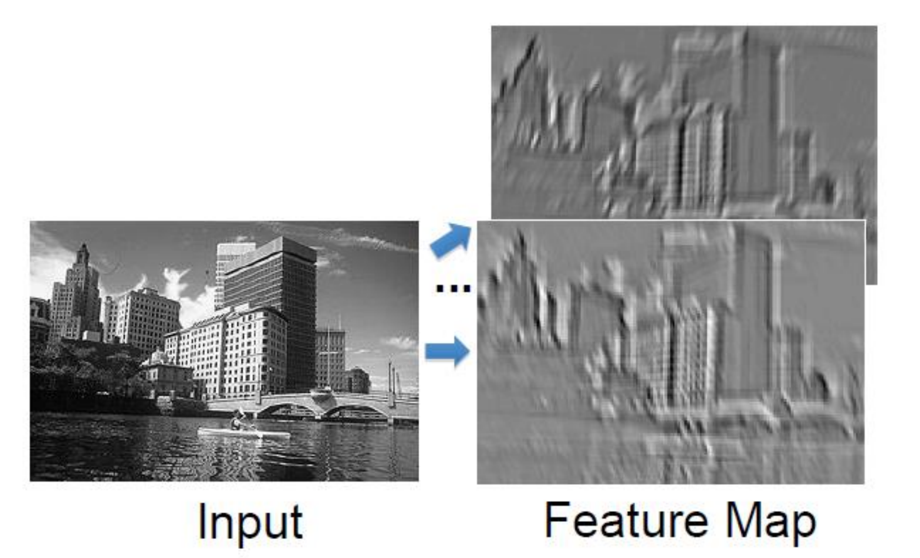
ConvNet Recap



Source: R. Fergus, Y. LeCun

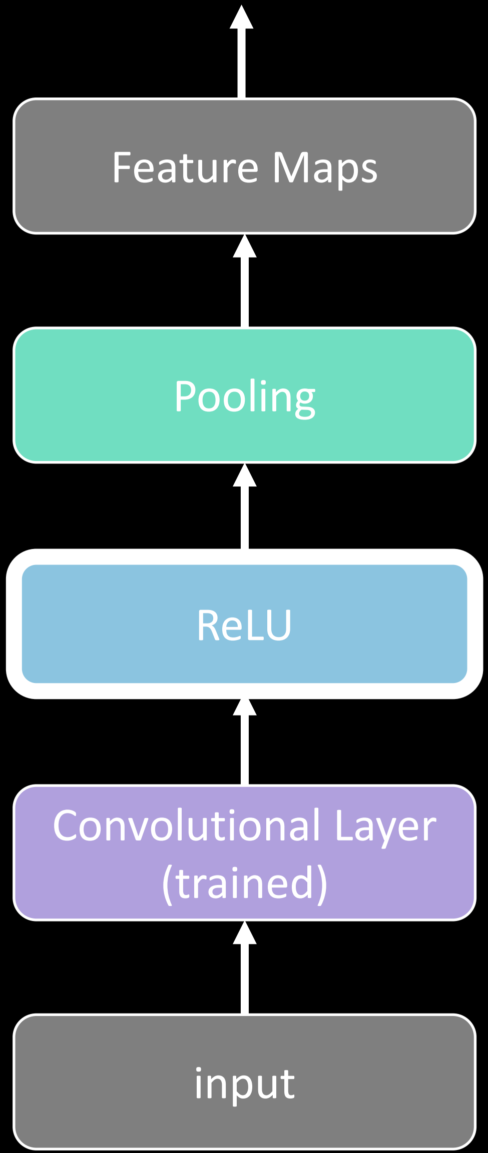
ConvNet Recap

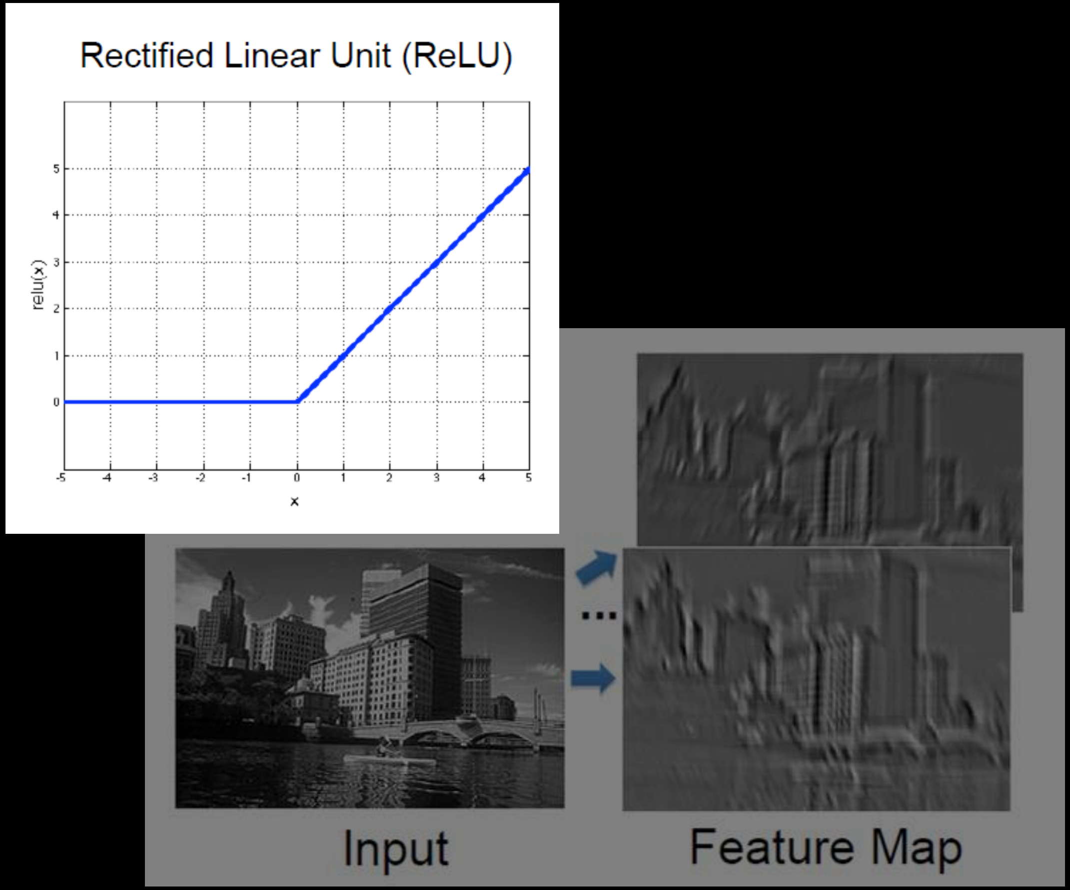




Source: R. Fergus, Y. LeCun

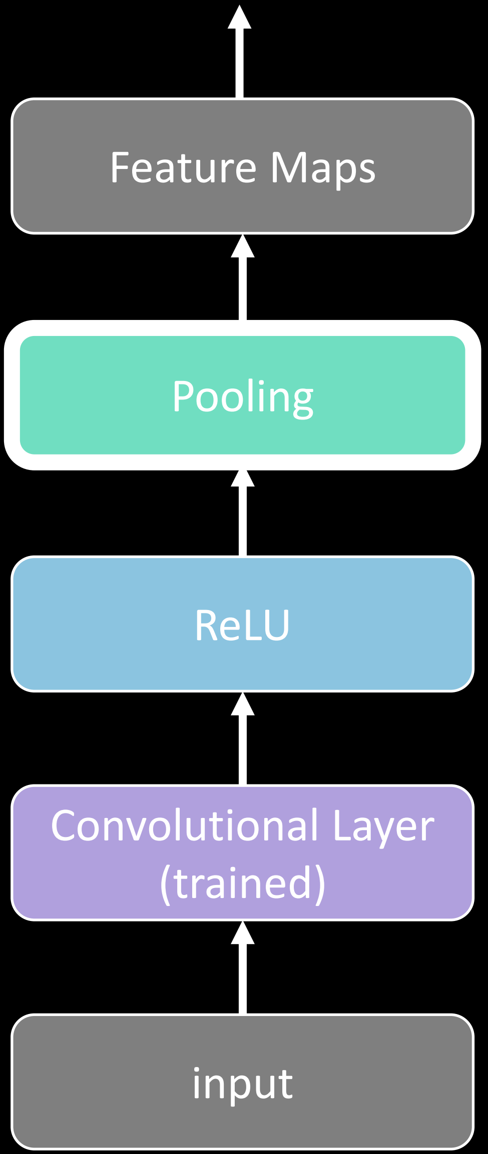
ConvNet Recap

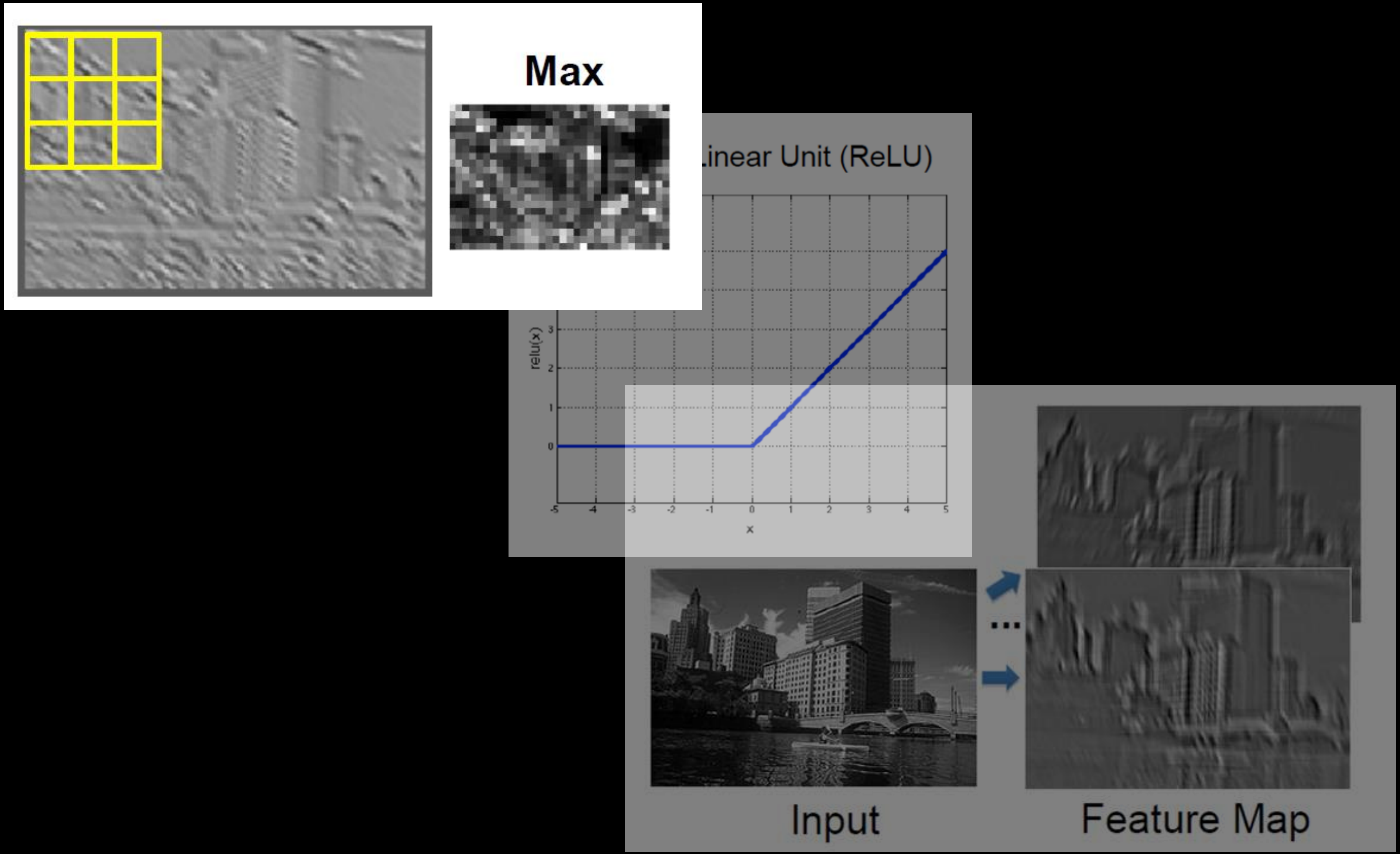


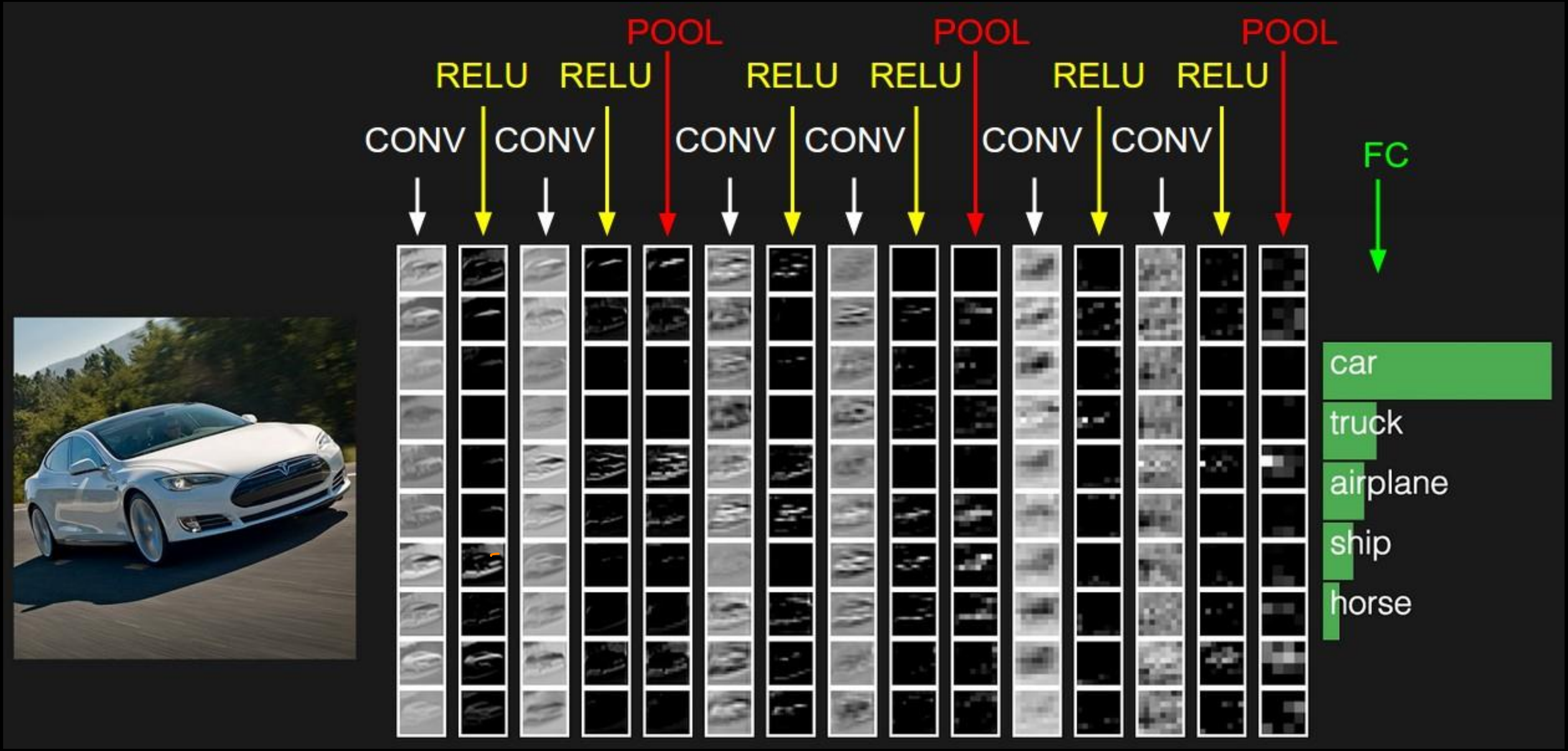


Source: R. Fergus, Y. LeCun

ConvNet Recap







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

Summary

- Crash course in computer vision
  - Recognition, reconfiguration, and reconstruction
  - Traditional features vs. learned features
- Introduced the basics of neural networks
  - Did not discuss: backpropagation or training methods
  - Did not discuss: state-of-the-art object detection architectures
    - Take a look at yolo tutorials – will post a few on discord!
- Next time: we'll look at modeling and control of vehicles!

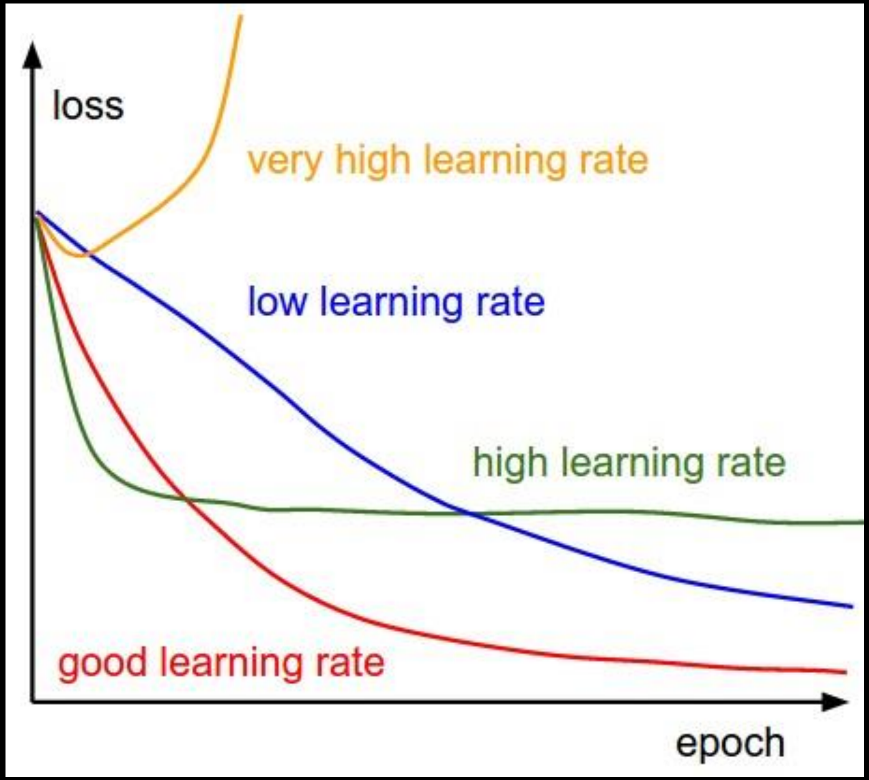
Extra Slides

So you want to train a neural net.

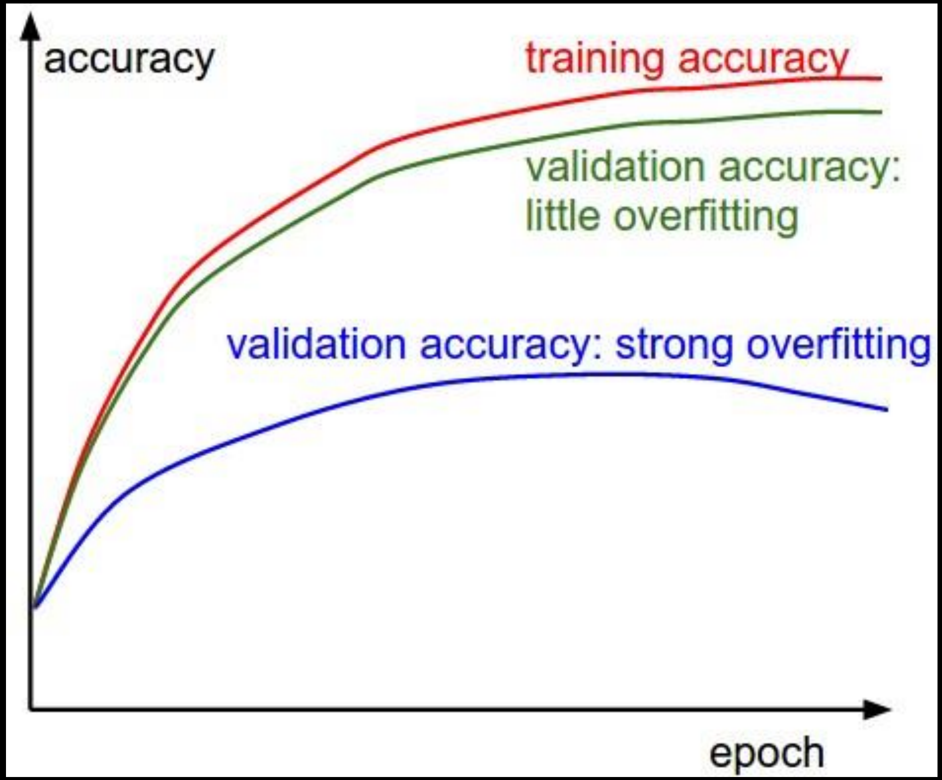
1. Pre-process data
   - Zero-center and scale by standard deviation
2. Initialize network
   - Initializing weights can be difficult due to instabilities
   - Small random numbers from normal distribution
3. Set up your regularization (penalty term, dropout)
4. Pick a loss function
   - Depends on problem, but try to shoot for softmax whenever possible
5. You are ready to train your network!
   - Initially try to overfit on a tiny subset of your data ~20 samples. Make sure you get zero loss.
6. Sweep over hyperparameters
   - Initial learning rate and decay schedule, regularization strength
   - Use cross-validation techniques and be prepared to wait. This can take weeks for large networks.

What to watch during training

Loss Rates



Training vs. Validation

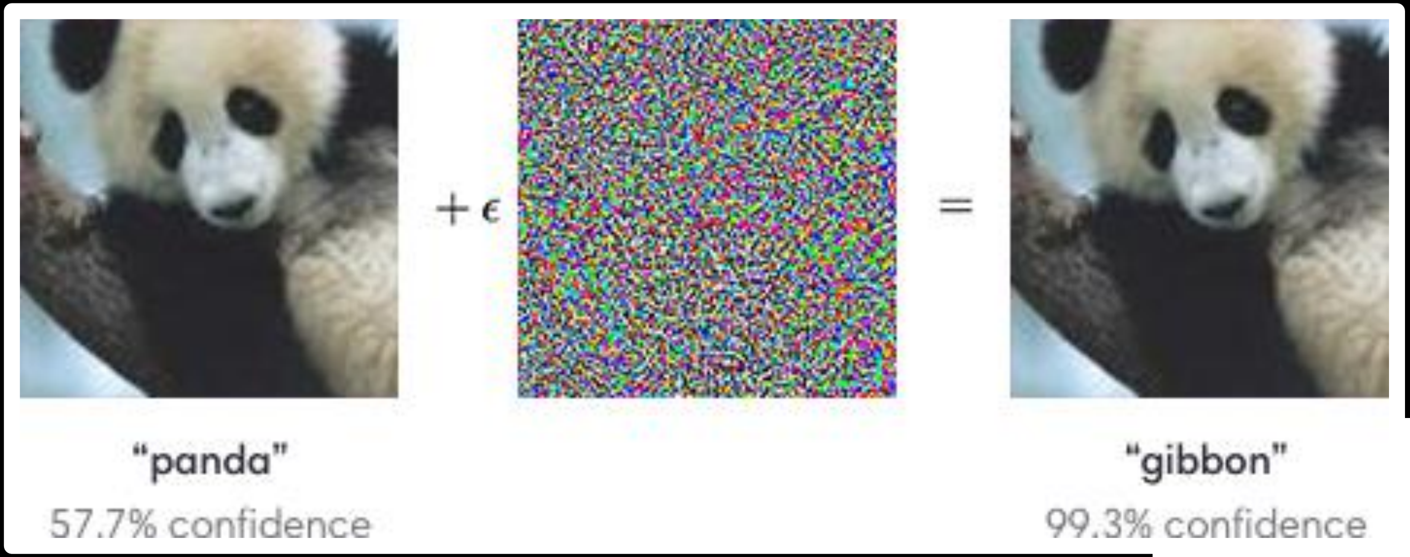


http://cs231n.github.io/neural-networks-3/

Risks for Autonomy

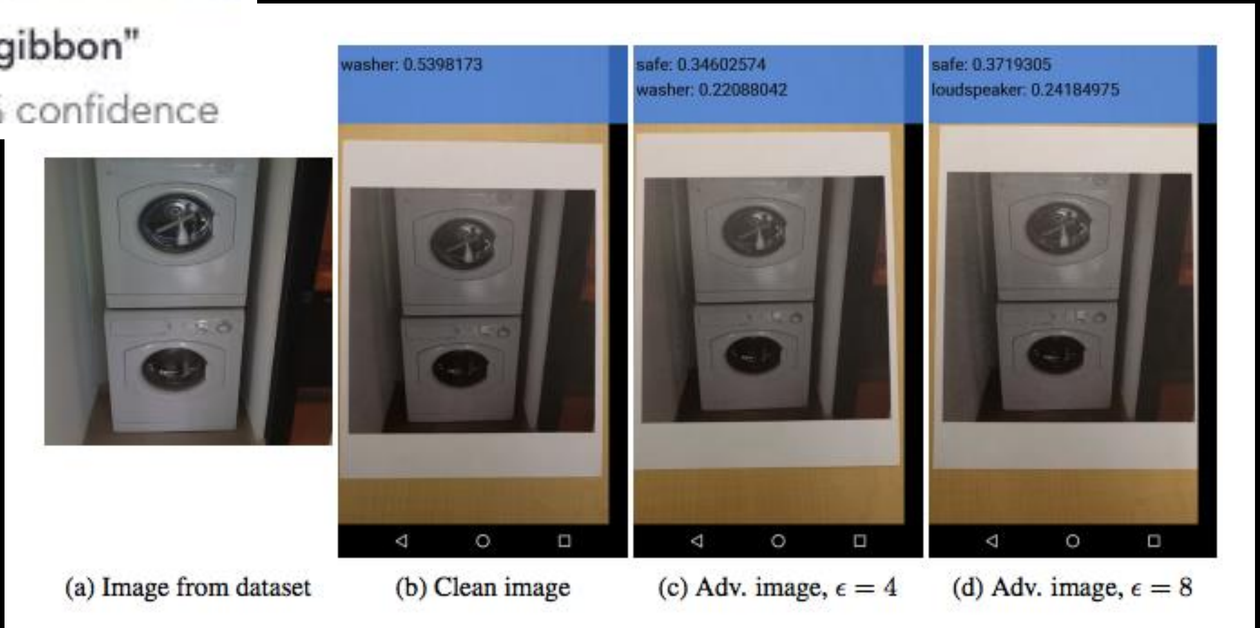
- What is it about neural networks that are particularly difficult?
  - Training stability is a problem
  - Large amounts of data are required
  - Huge amounts of computation are required

Adversarial Examples



Find the minimum perturbation that will result in a misclassification.
Resulting noise is often imperceptible.

Even transferring to different cameras or into the physical world can be quite difficult.



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

Adversarial examples from Ian Goodfellow.

A few things to keep in mind

1. Machine Learning is not always the answer – try simple methods first.
   - However, use ML over a complex heuristic. A simple heuristic can only get you so far, while a complex heuristic is unmaintainable.
2. When picking features, make sure they are generalizable!
3. Watching out for data imbalances or other quirks with your data.
4. You may skew your data by causing a discrepancy between how you handle data in the training and testing.
5. Cross validation is key – never peek at your testing data. You may create a feedback loop between your model and your algorithm.