

CAP5415

Computer Vision

Yogesh S Rawat

yogesh@ucf.edu

HEC-241

Course project

Questions?

Introduction to Neural Networks

Lecture 5

Agenda

- Feature engineering vs feature learning
- Neural network basics
- Non-linearity

Introduction to Neural Networks

Lecture 5

Feature engineering vs feature learning

Where to go from edges?

Image



Gradients, edges, etc.

Features

Recognition

Scenes, places, objects, etc.

Detection

Object, action, etc.

Segmentation

Object, semantic, etc.

Our goal in object classification

Our goal in object classification

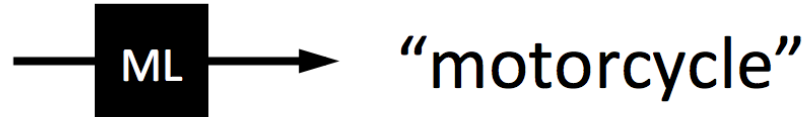


Our goal in object classification



— ML

Our goal in object classification



Why is this hard?

You see this:



Why is this hard?

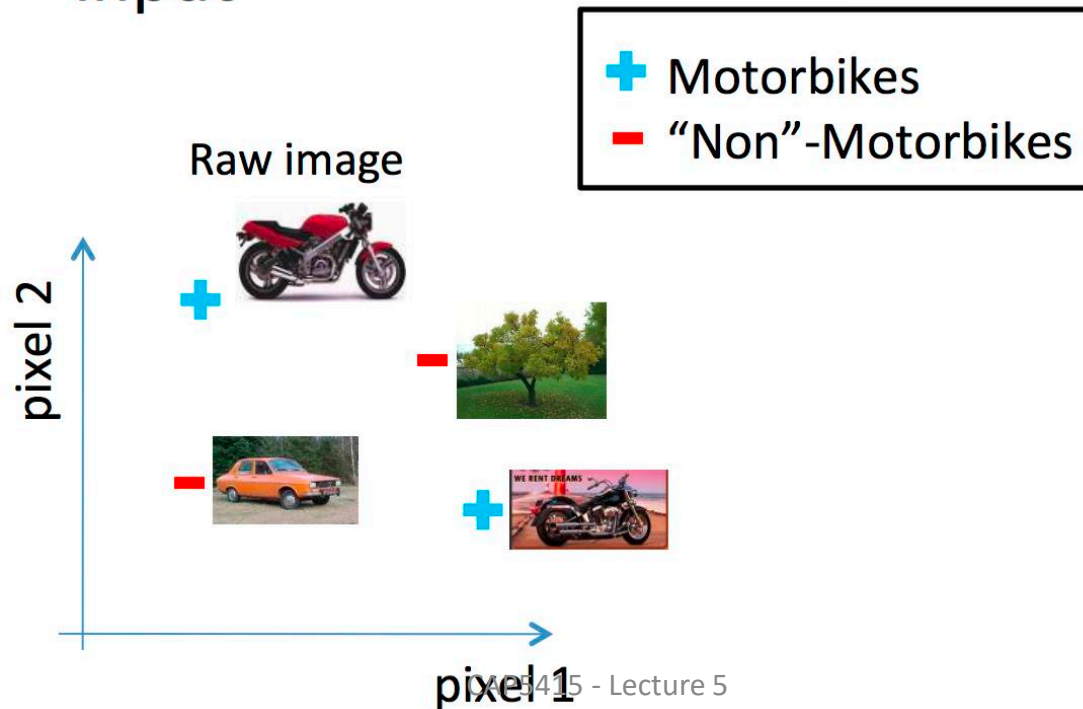
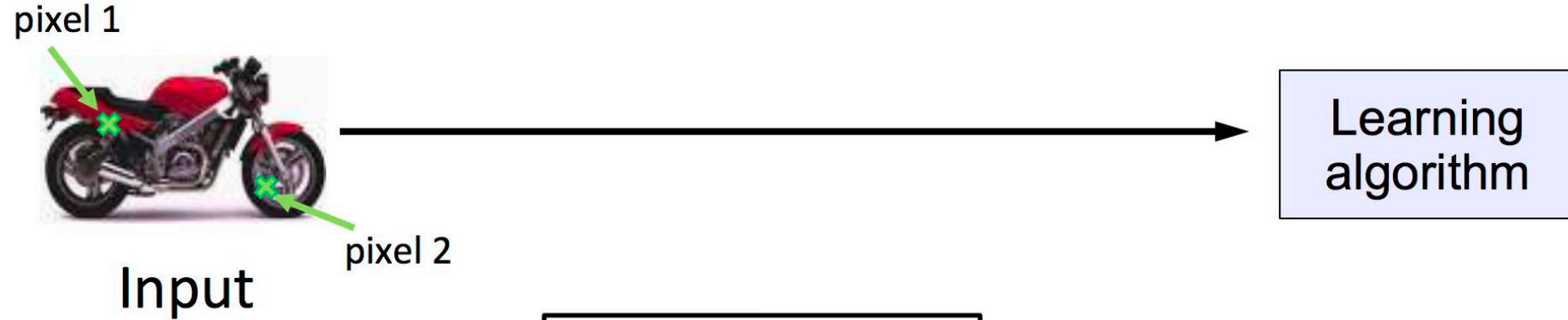
You see this:



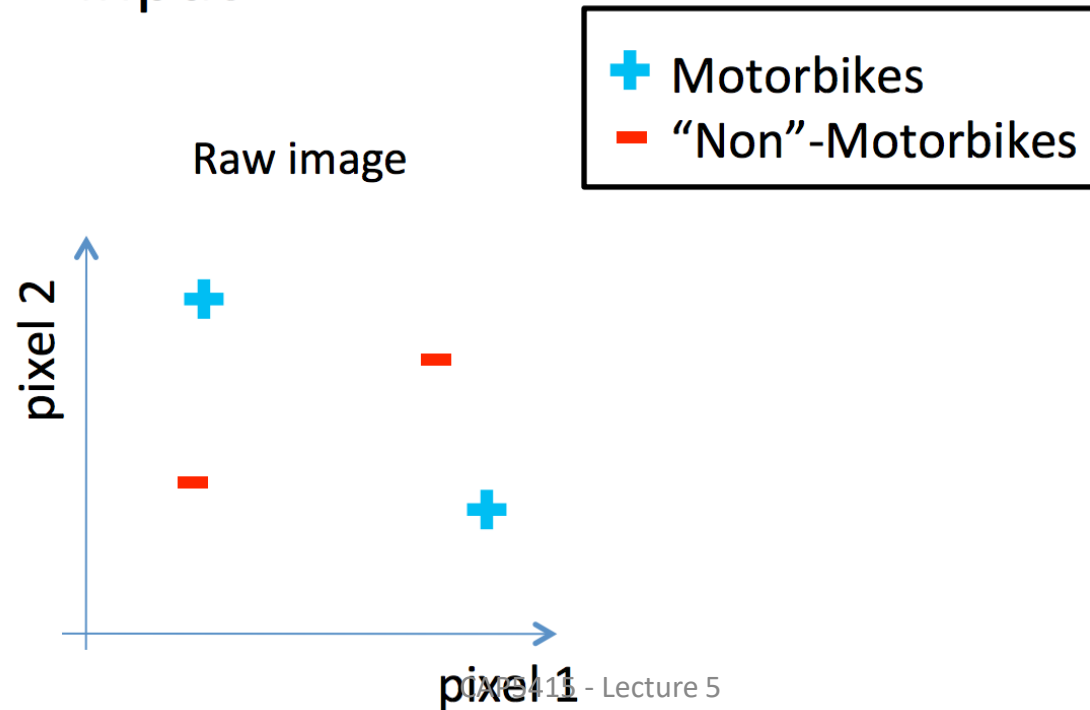
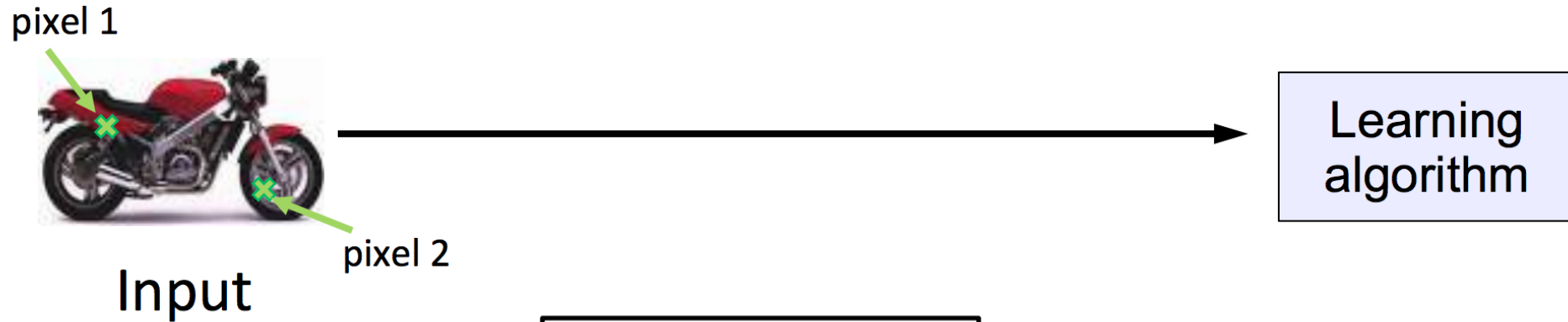
But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

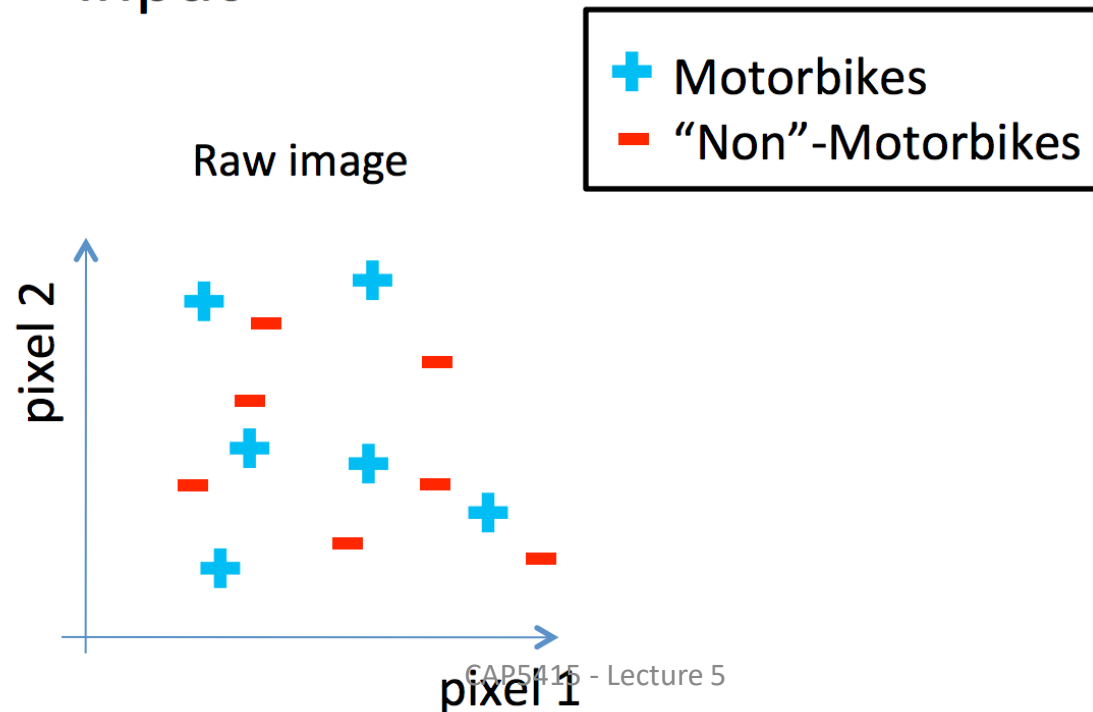
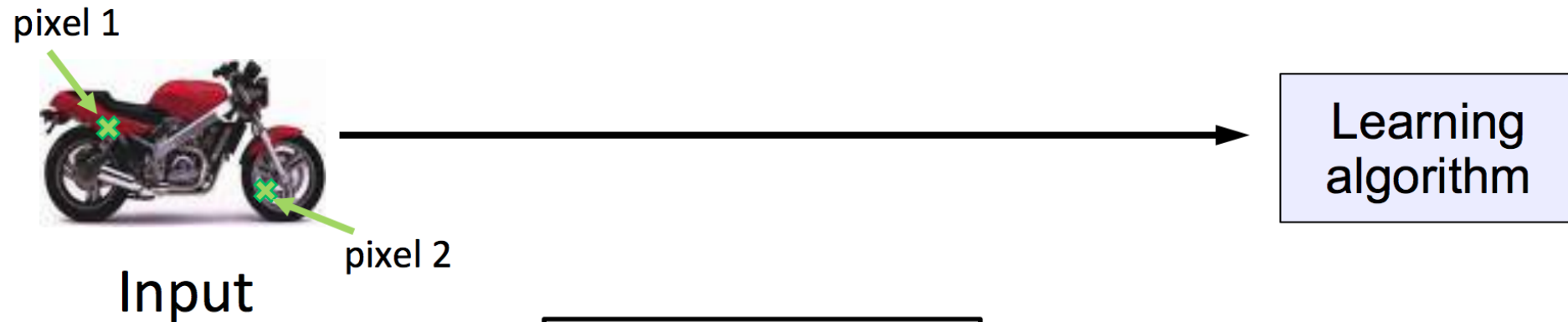
Pixel-based representation



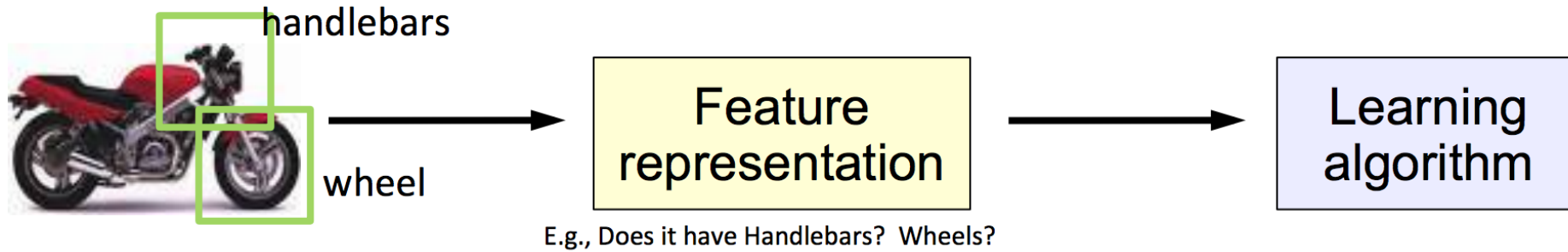
Pixel-based representation



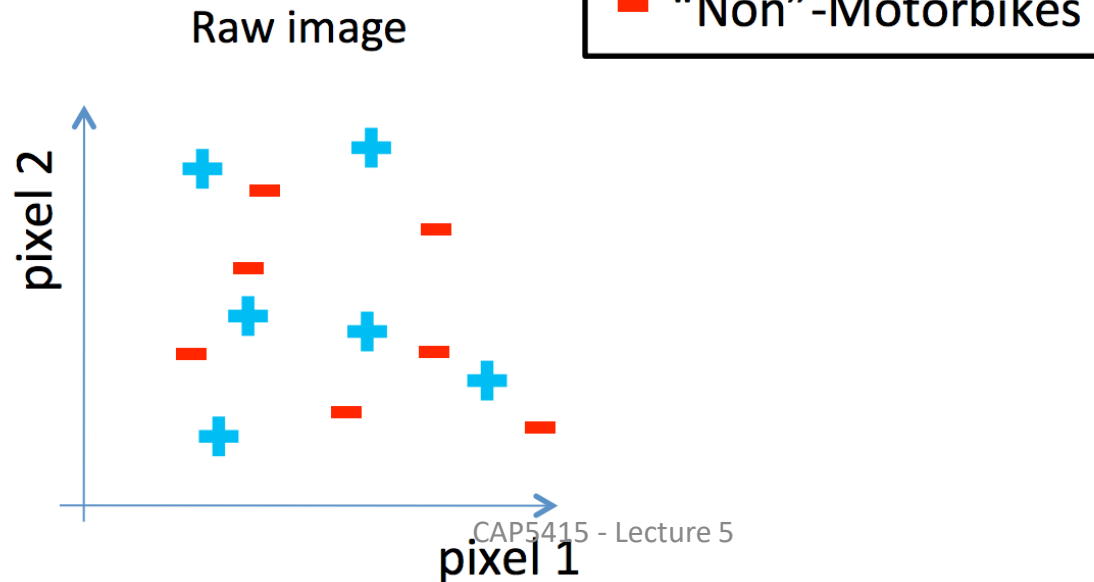
Pixel-based representation



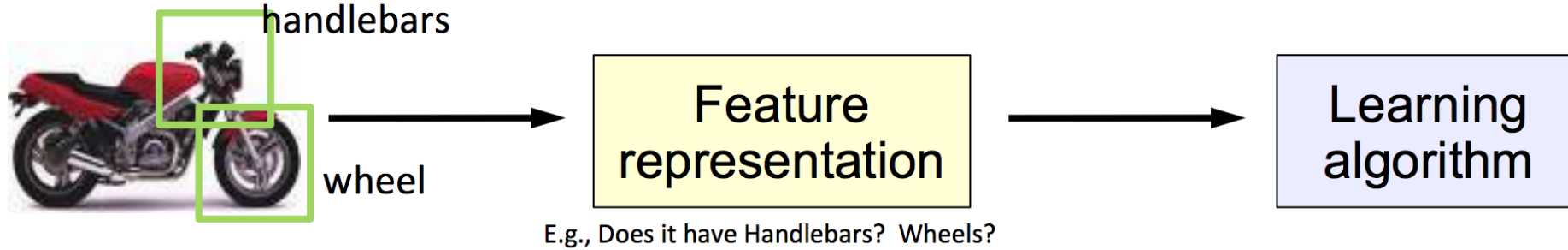
What we want



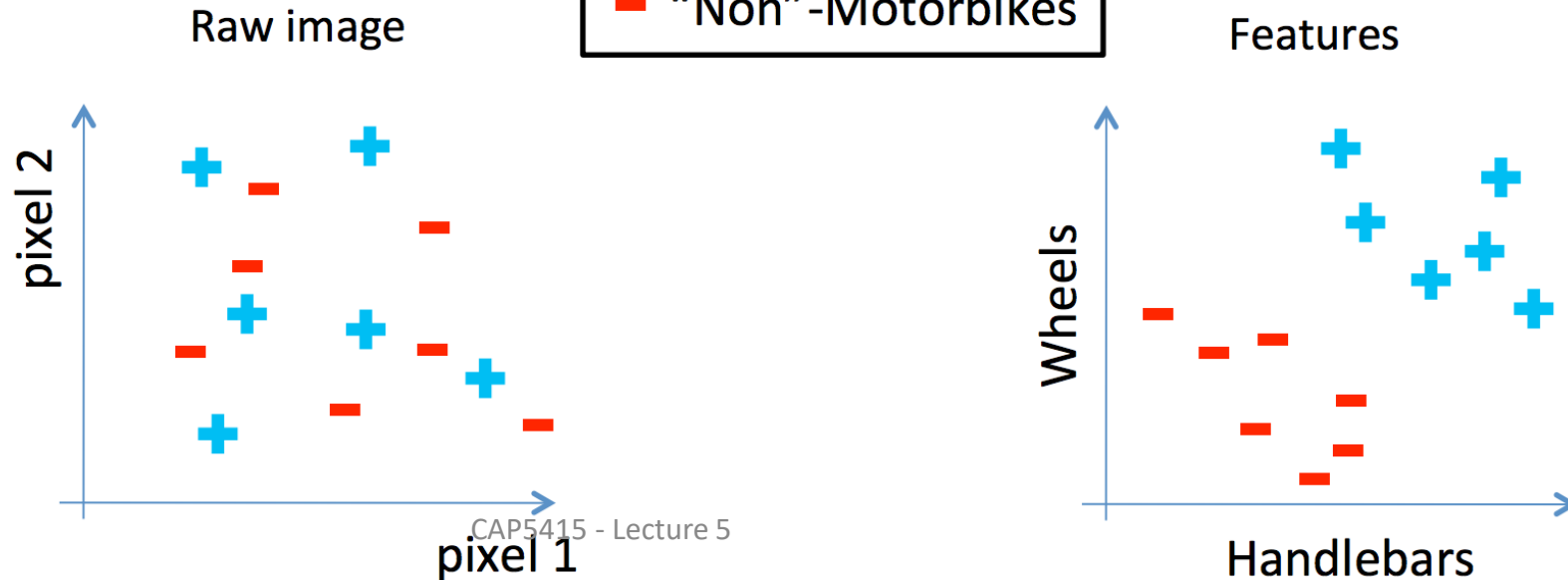
Input



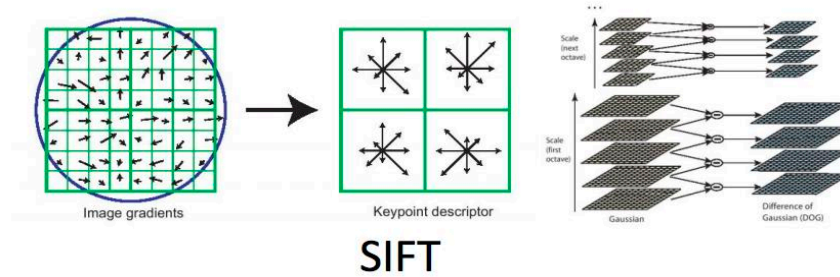
What we want



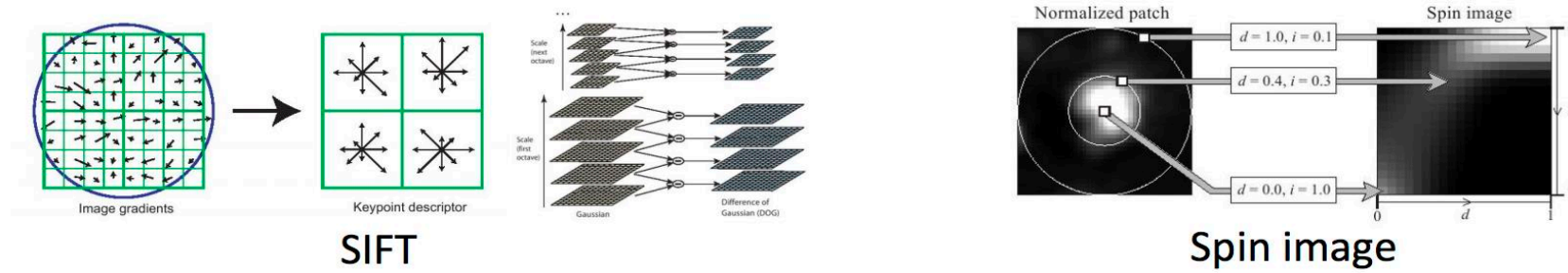
Input



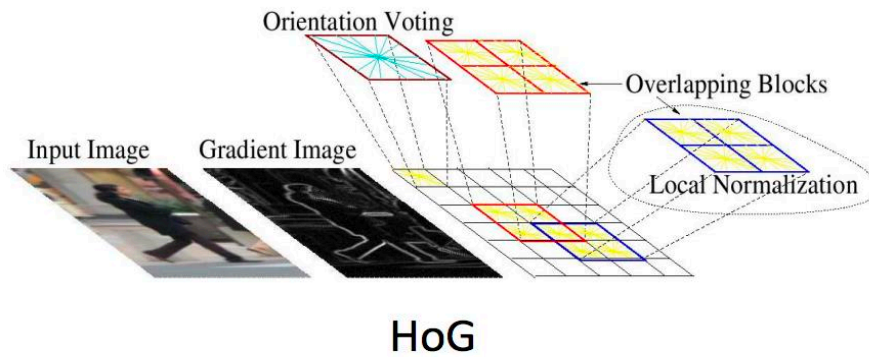
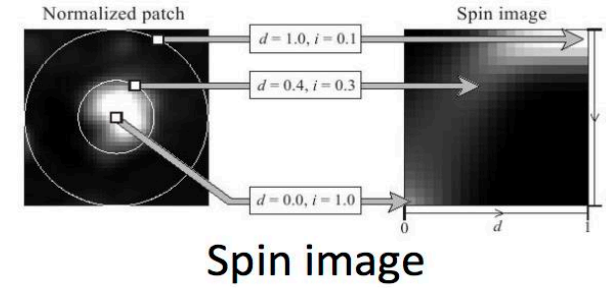
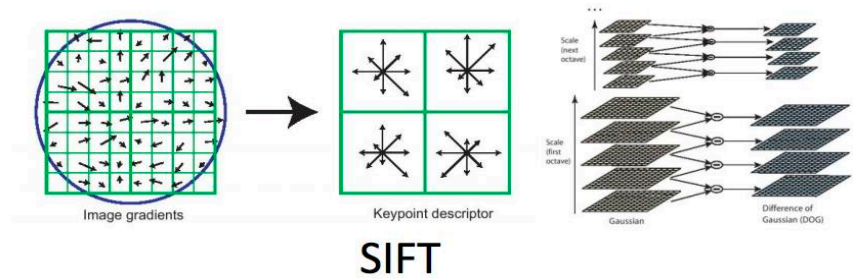
Some feature representations



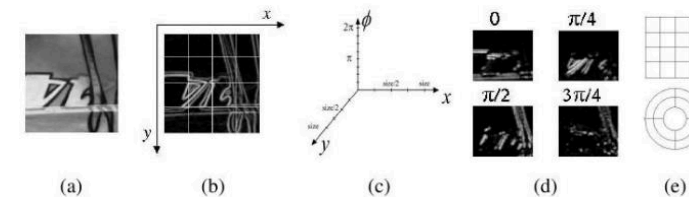
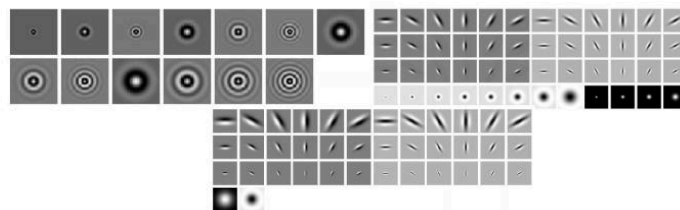
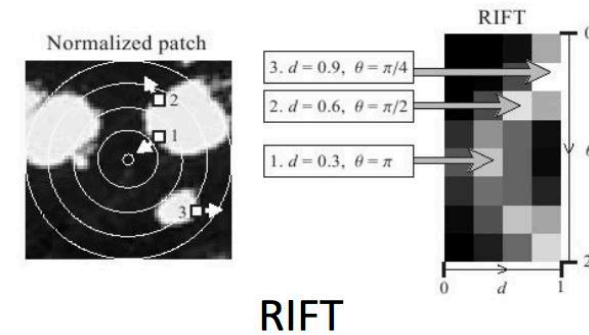
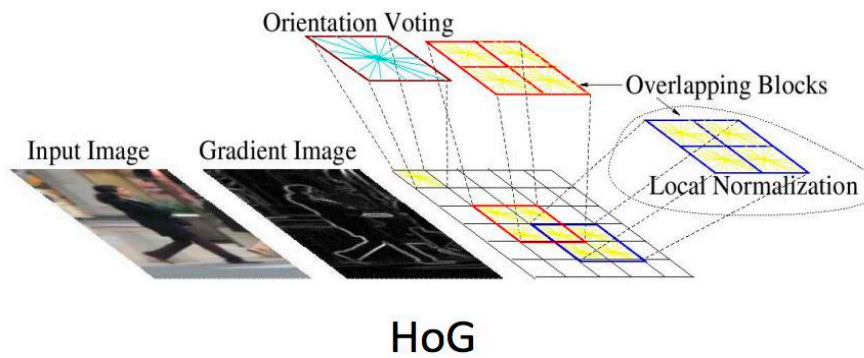
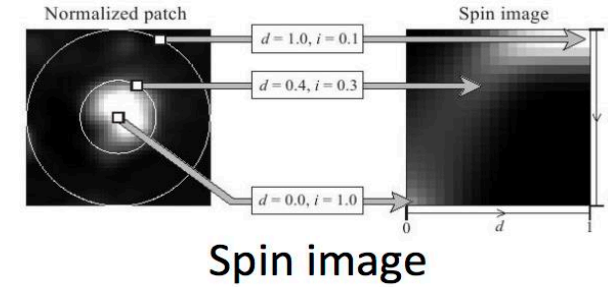
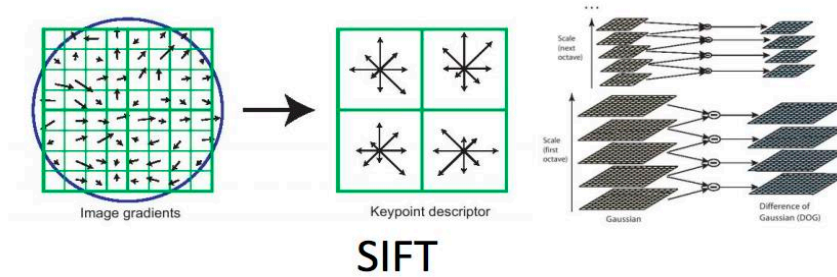
Some feature representations



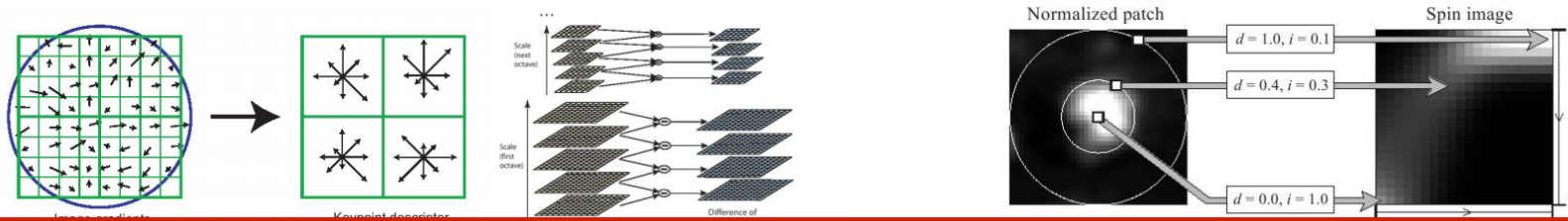
Some feature representations



Some feature representations

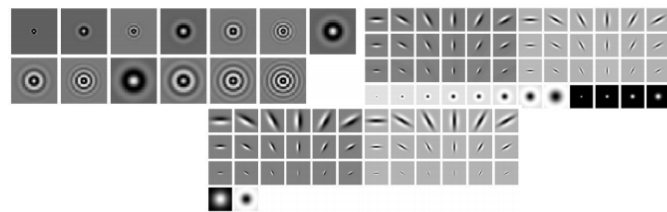


Some feature representations



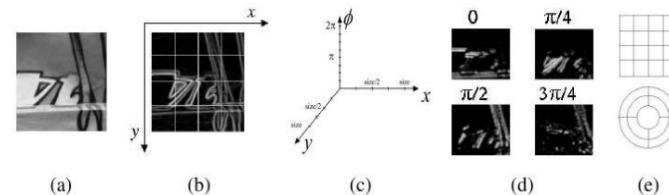
Coming up with features is often difficult, time-consuming, and requires expert knowledge.

HoG



Textons

RIFT



GLOH

Today

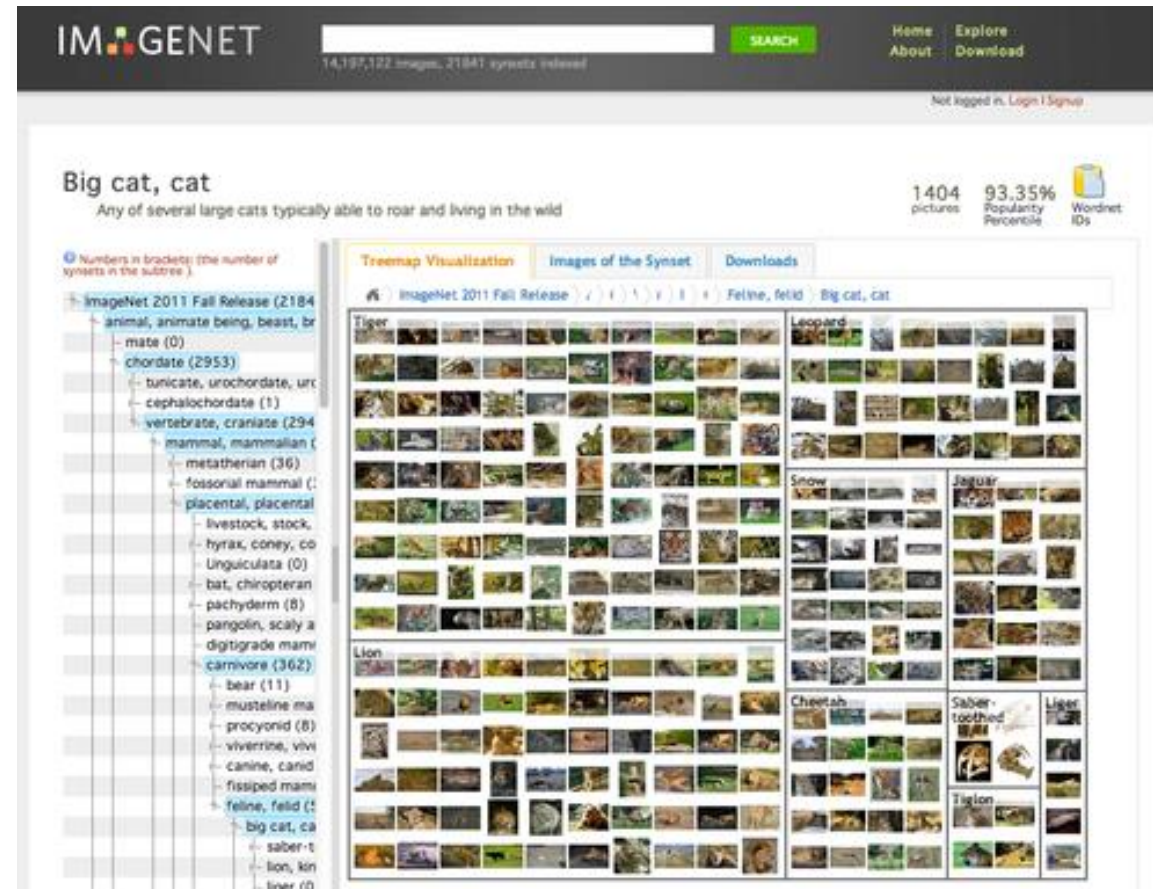
Feature engineering
Expert knowledge

->

Feature learning
Data

Image classification - ImageNet

- Images for each category of WordNet
- 1000 classes
- 14M images
- 100K test



Dataset split

Training Images



- Train classifier

Validation Images



- Measure error
- Tune model hyperparameters

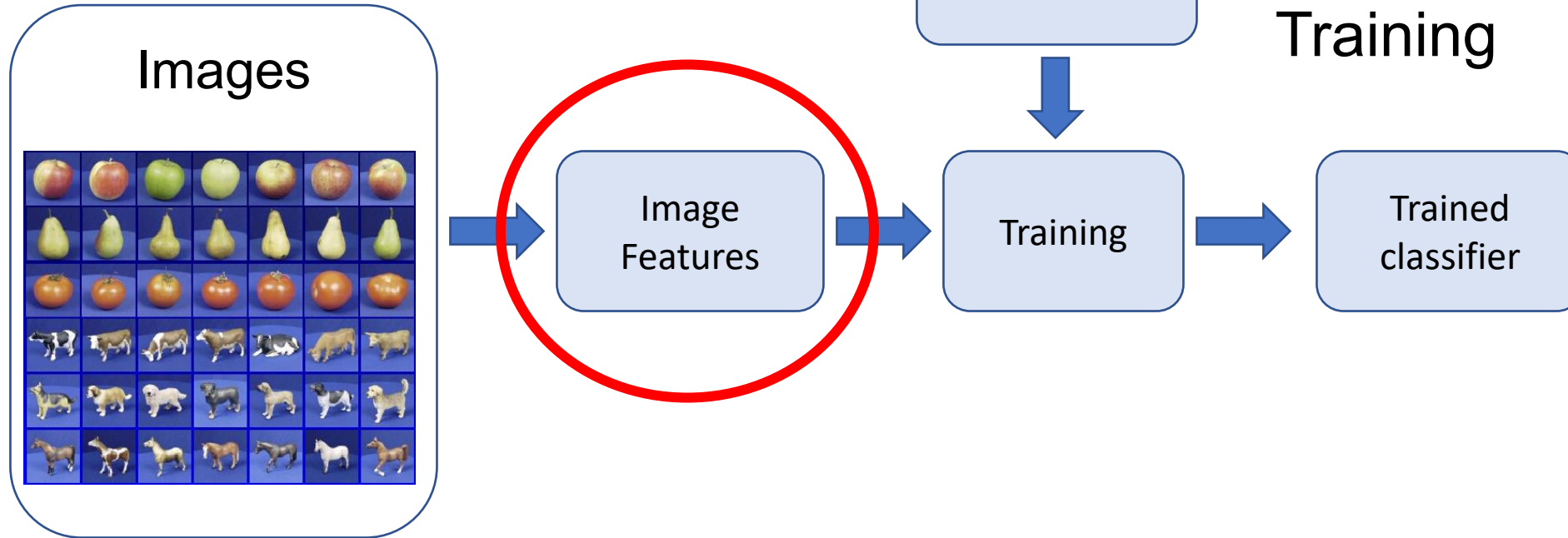
Testing Images



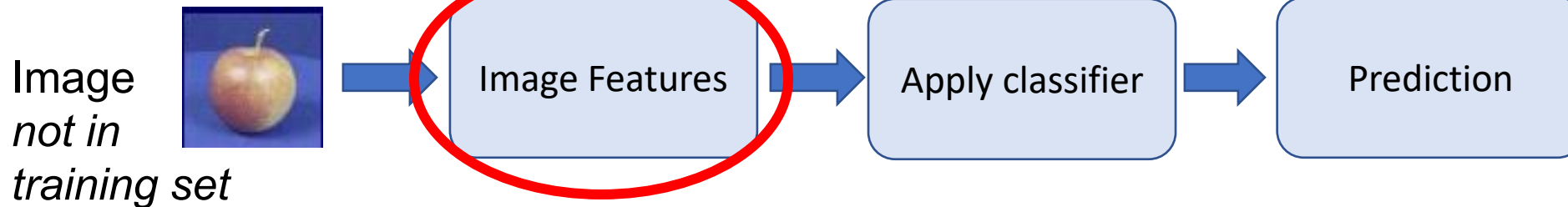
- Secret labels
- Measure error

Random train/validate splits = cross validation

Learning phases

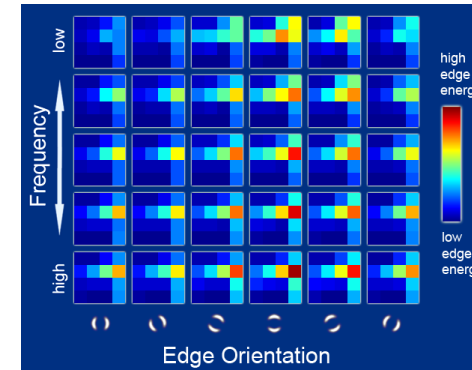
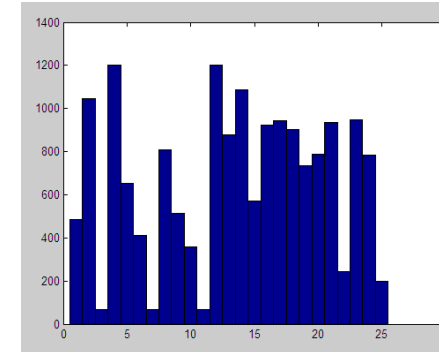


Testing

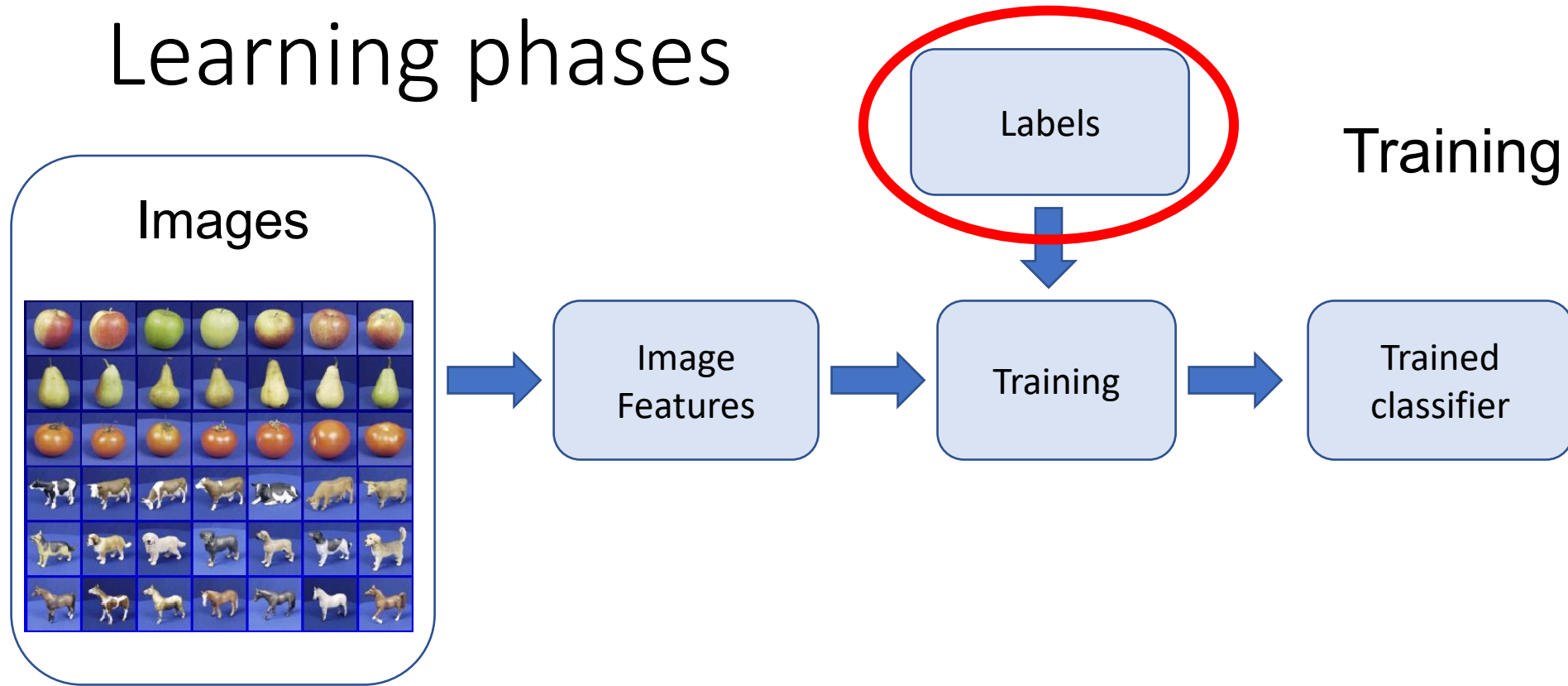


Features

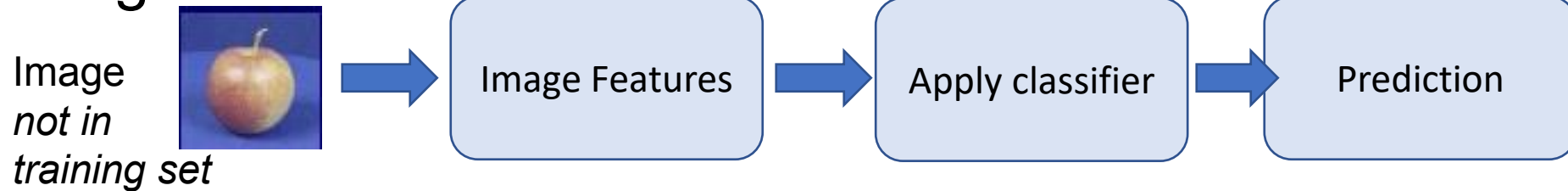
- Raw pixels
- Histograms
- Templates
- SIFT descriptors
 - GIST
 - ORB
 - HOG....
- Learned features



Learning phases



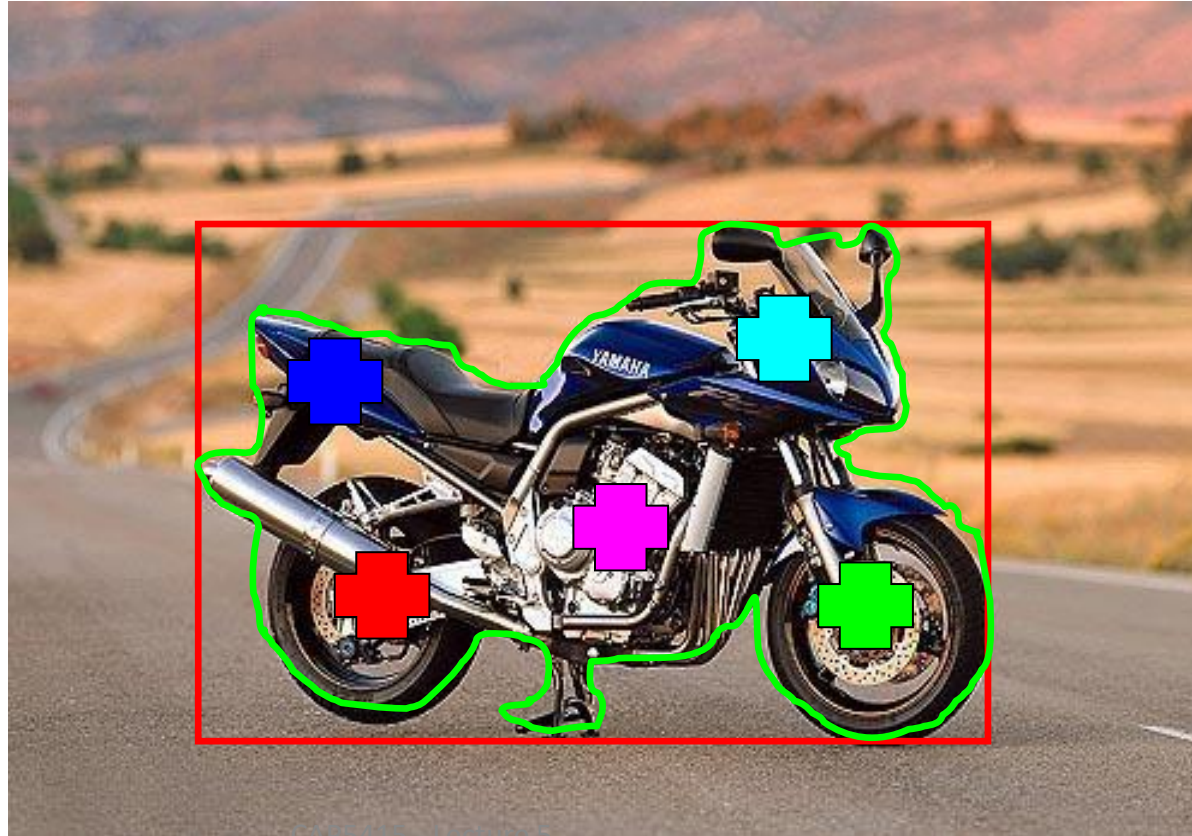
Testing



Recognition task and supervision

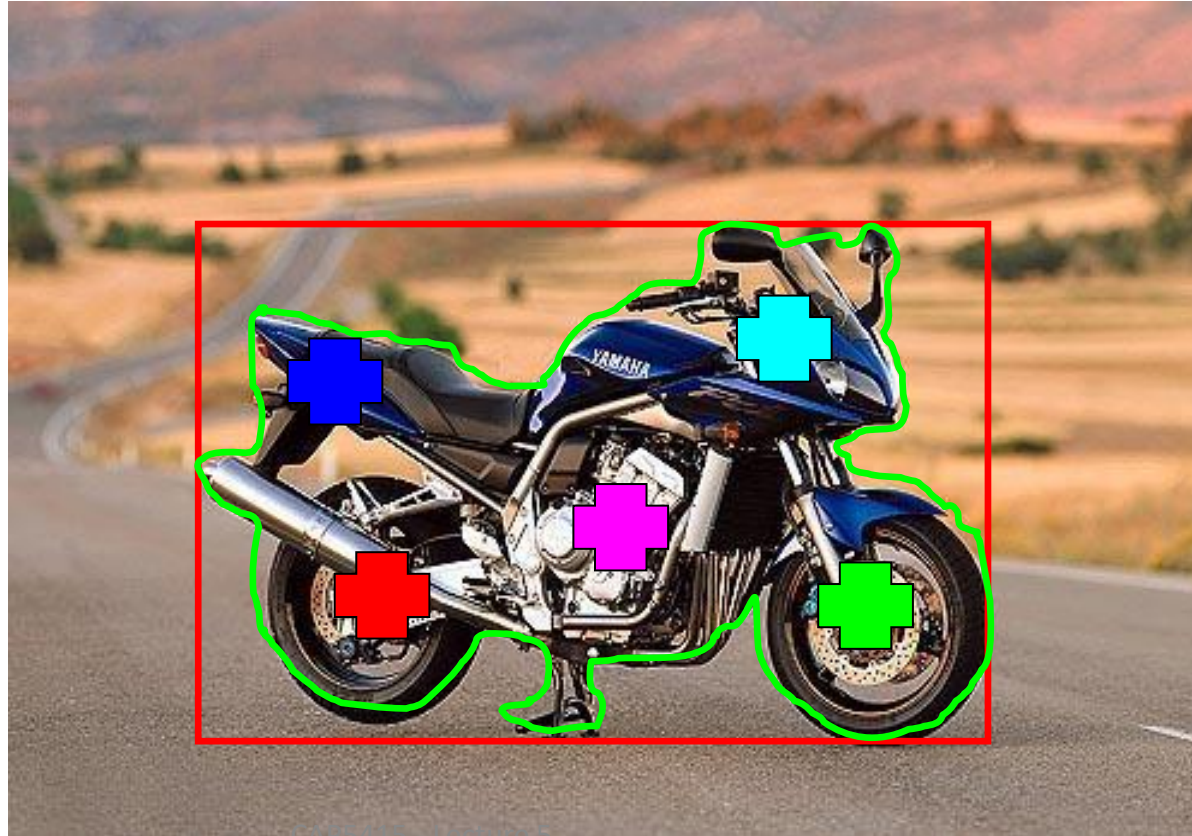
What are all the possible supervision ('label') *types* to consider?

Contains a motorbike



Recognition task and supervision

- Images in the training set must be annotated with the “correct answer” that the model is expected to produce



Contains a motorbike

Spectrum of supervision

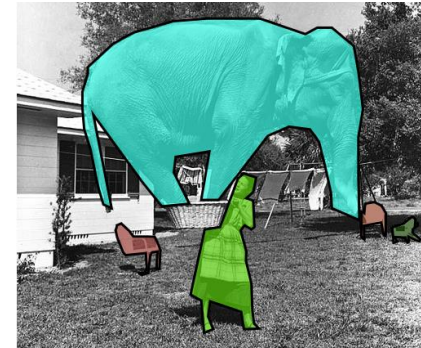
Less

More



e.g., ImageNet

e.g., MS-COCO



Unsupervised

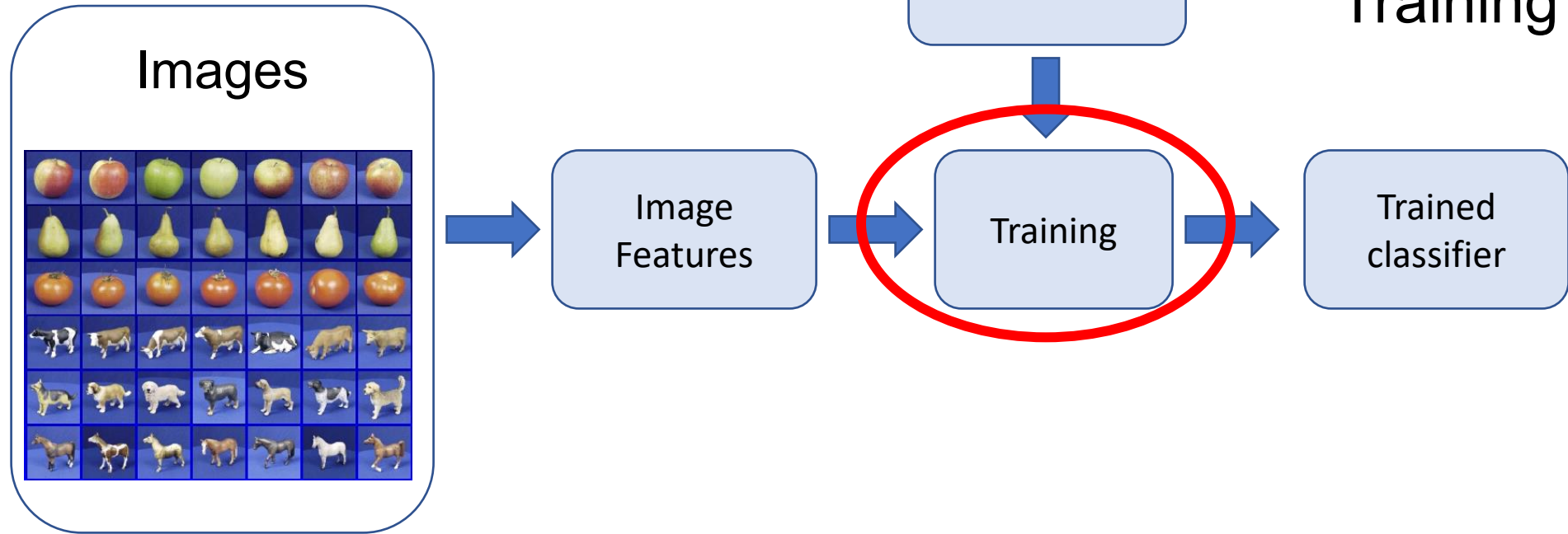
“Weakly” supervised

Fully supervised

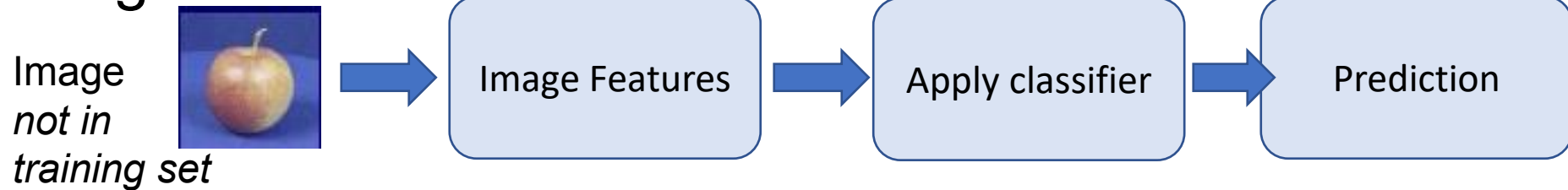
‘Semi-supervised’: small partial labeling

Fuzzy; definition depends on task

Learning phases



Testing



The machine learning framework

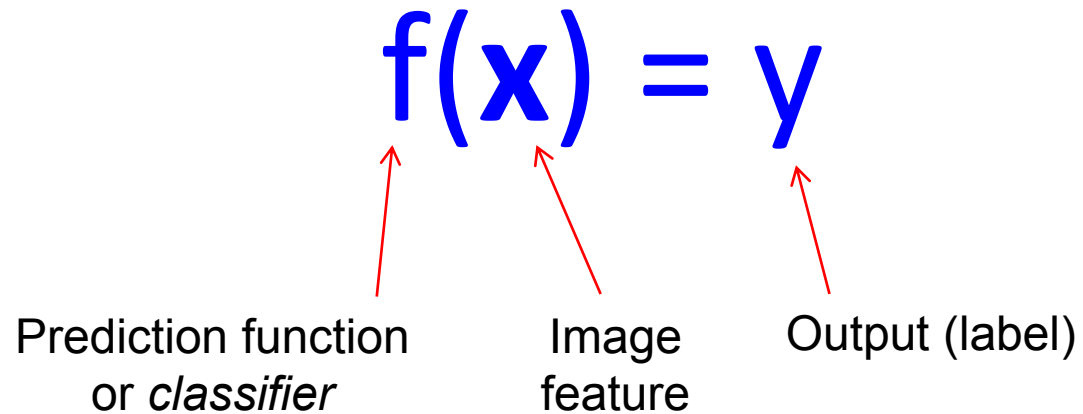
- Apply a prediction function to a feature representation of the image to get the desired output:

$f(\text{apple image}) = \text{"apple"}$

$f(\text{tomato image}) = \text{"tomato"}$

$f(\text{cow image}) = \text{"cow"}$

The machine learning framework



Training: Given a *training set* of labeled examples:

$$\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$$

Estimate the prediction function f by minimizing the prediction error on the training set.

Testing: Apply f to an unseen *test example* \mathbf{x}_u and output the predicted value $y_u = f(\mathbf{x}_u)$ to *classify* \mathbf{x}_u .

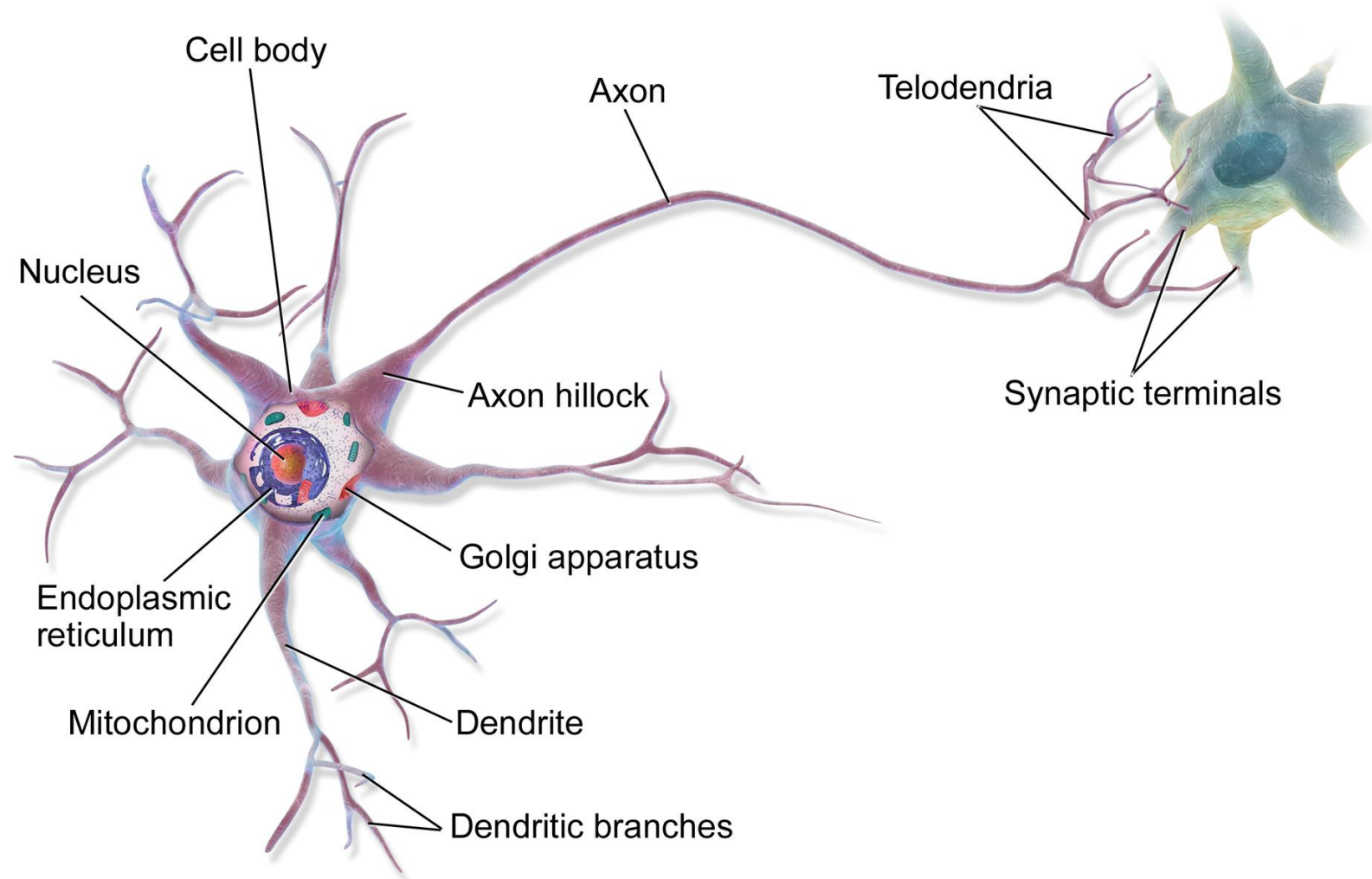
Questions?

Introduction to Neural Networks

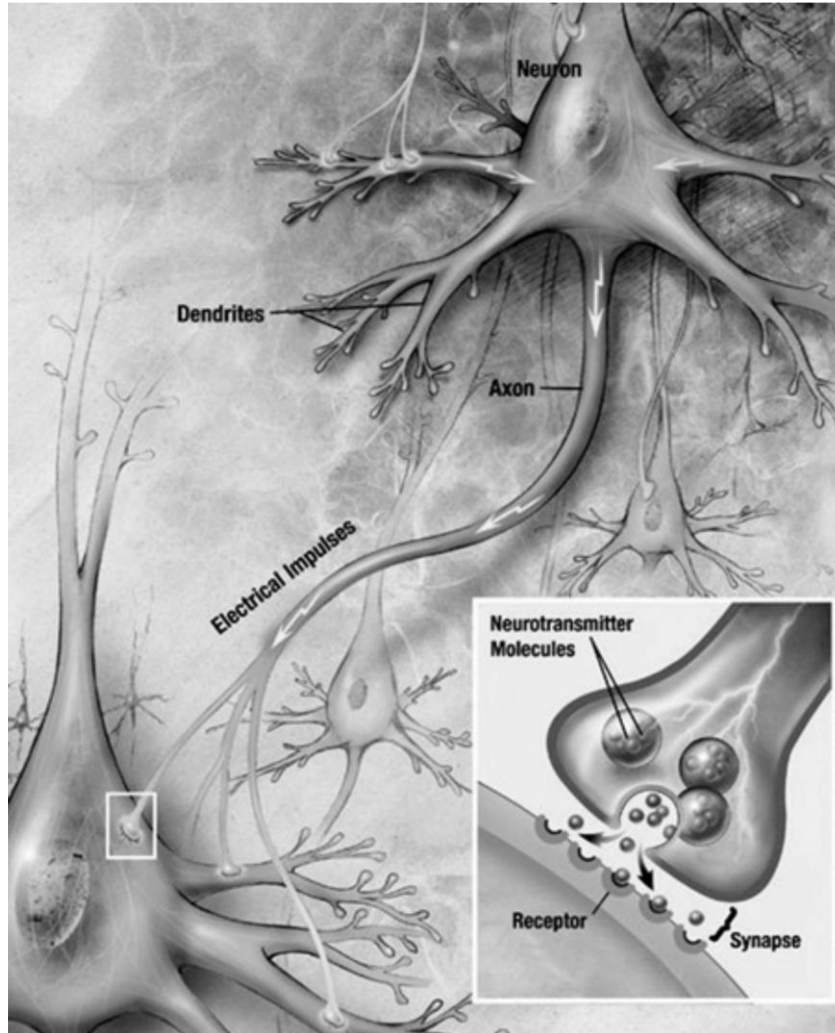
Lecture 5

Neural network basics

Neurons in the Brain



Neurons in the Brain



- Brain is composed of **neurons**
- A neuron receives input from other neurons (generally thousands) from its synapses
- Inputs are approximately **summed**
- When the input exceeds a threshold the neuron sends an electrical spike that travels from the body, down the axon, to the next neuron(s)

Background in Neural Nets (NN)

- Neural Nets can be:

- **Biological** Models
- **Artificial** Models



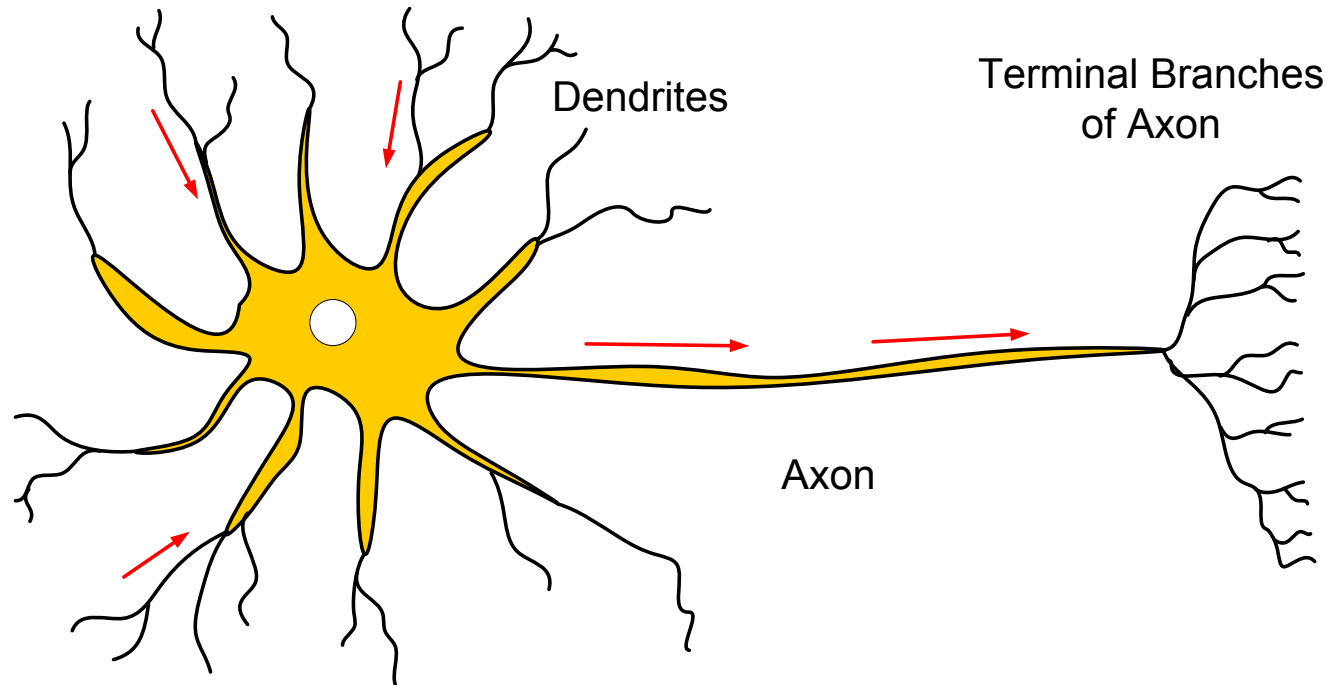
- Desire to produce **artificial systems**
 - capable of sophisticated computations
 - similar to human brain!

Brain is a remarkable Computer

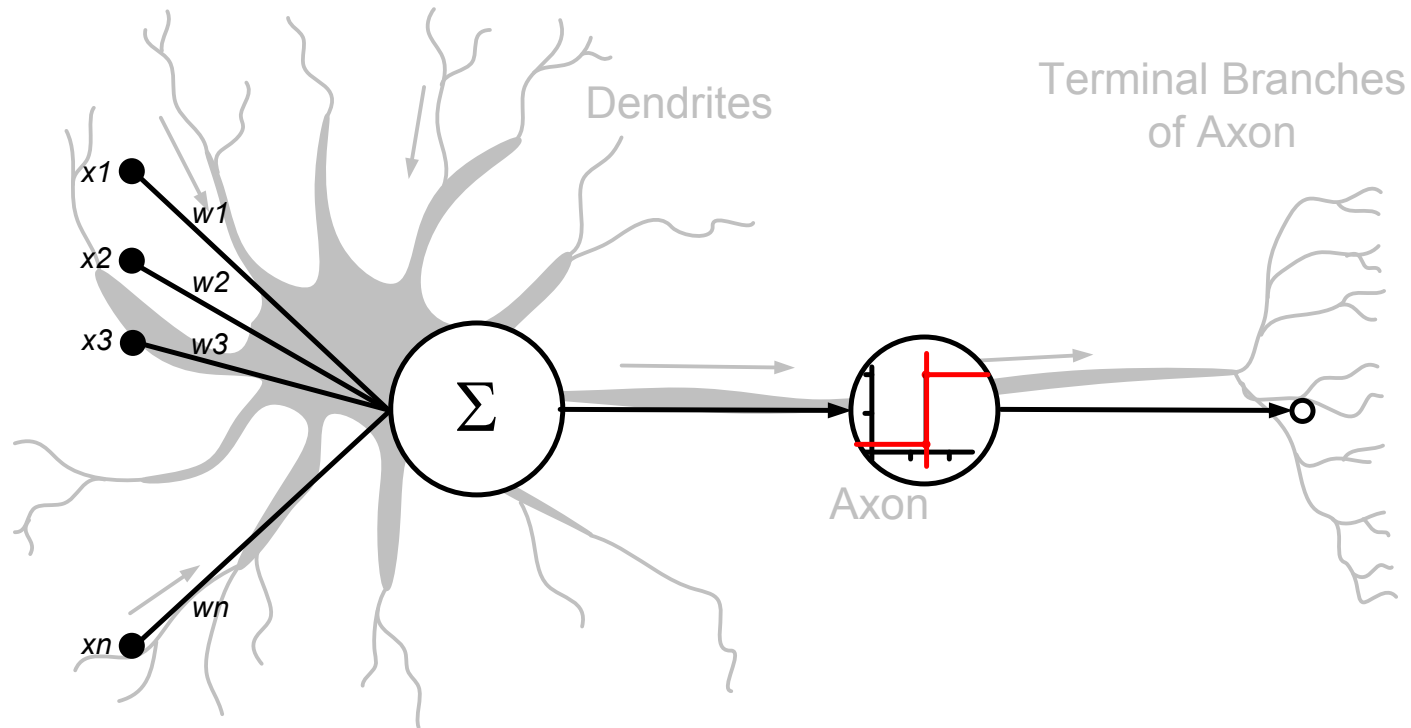


- 200 billion neurons, 32 trillion synapses
- Energy use: 25W
- Processing speed: 100 Hz
- Parallel, Distributed
- Fault Tolerant
- Learns: Yes
- Intelligent/Conscious: Usually
- 1 billion bytes RAM but trillions of bytes on disk
- Energy watt: 30-90W (CPU)
- Processing speed: 10^9 Hz
- Serial, Centralized
- Generally not Fault Tolerant
- Learns: Some
- Intelligent/Conscious: Generally No

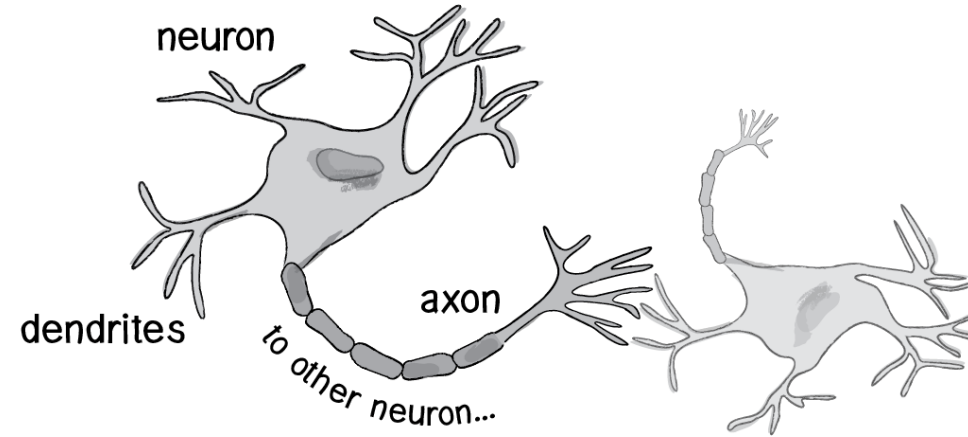
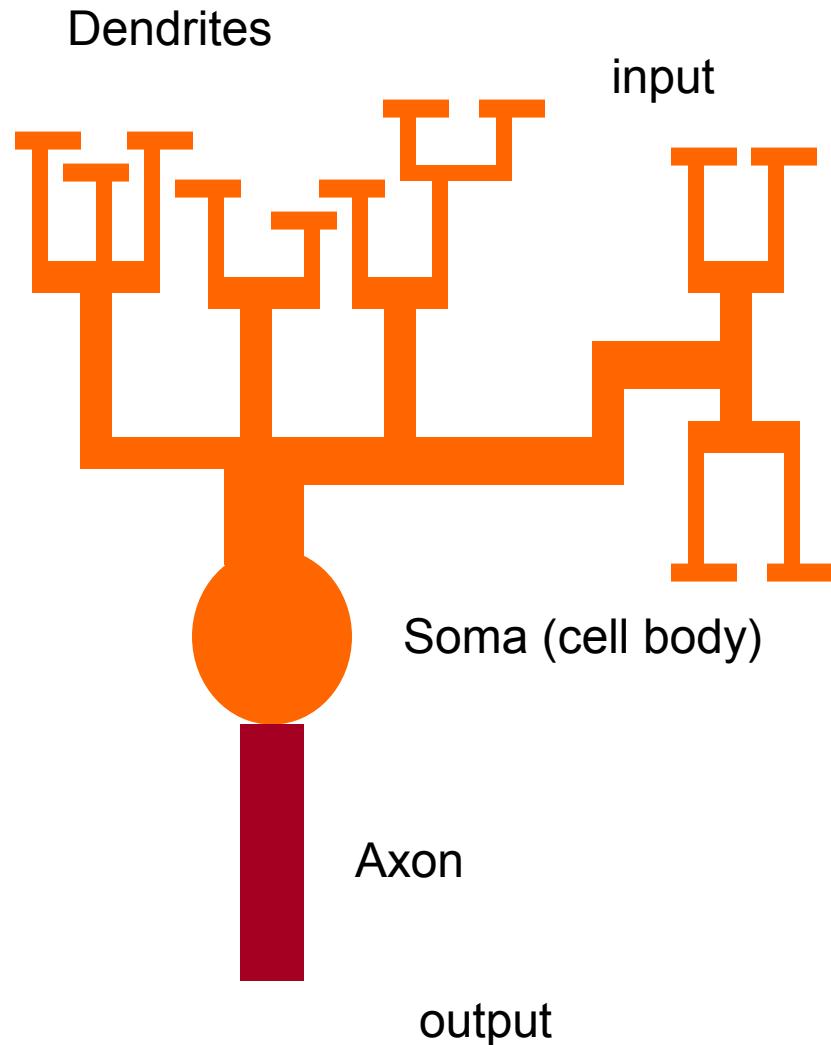
Computational Implementation of the Neural Activation Function



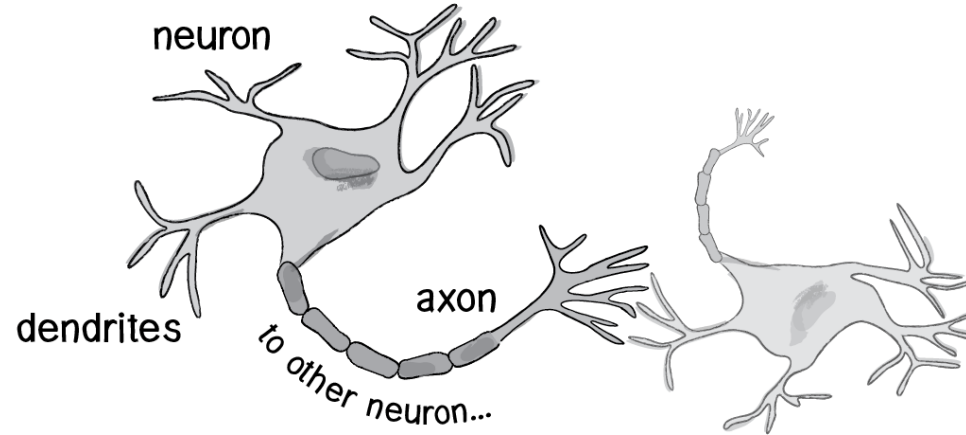
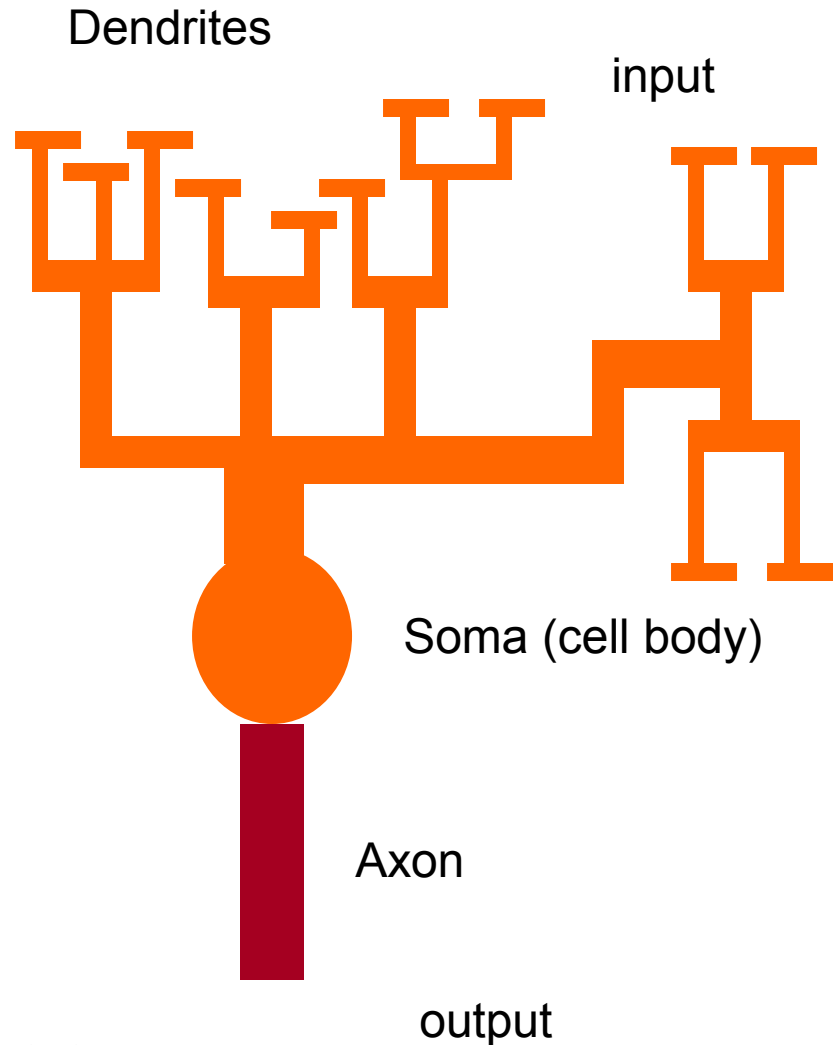
Computational Implementation of the Neural Activation Function



Neural Activation Function



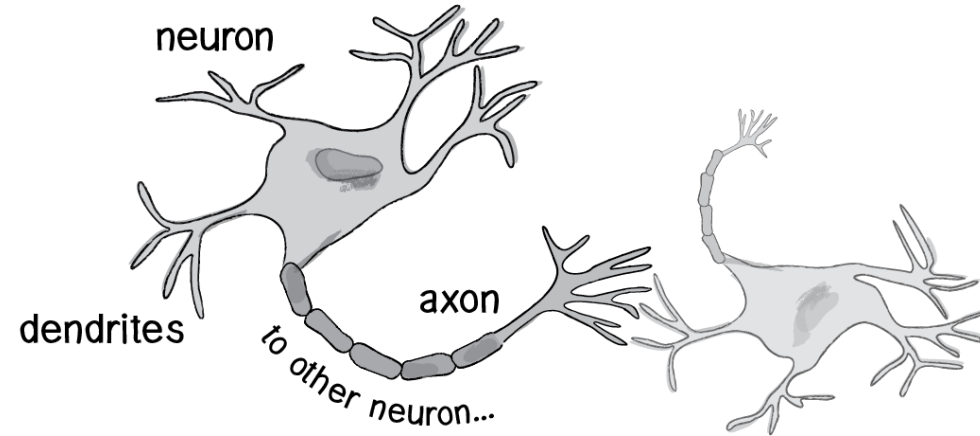
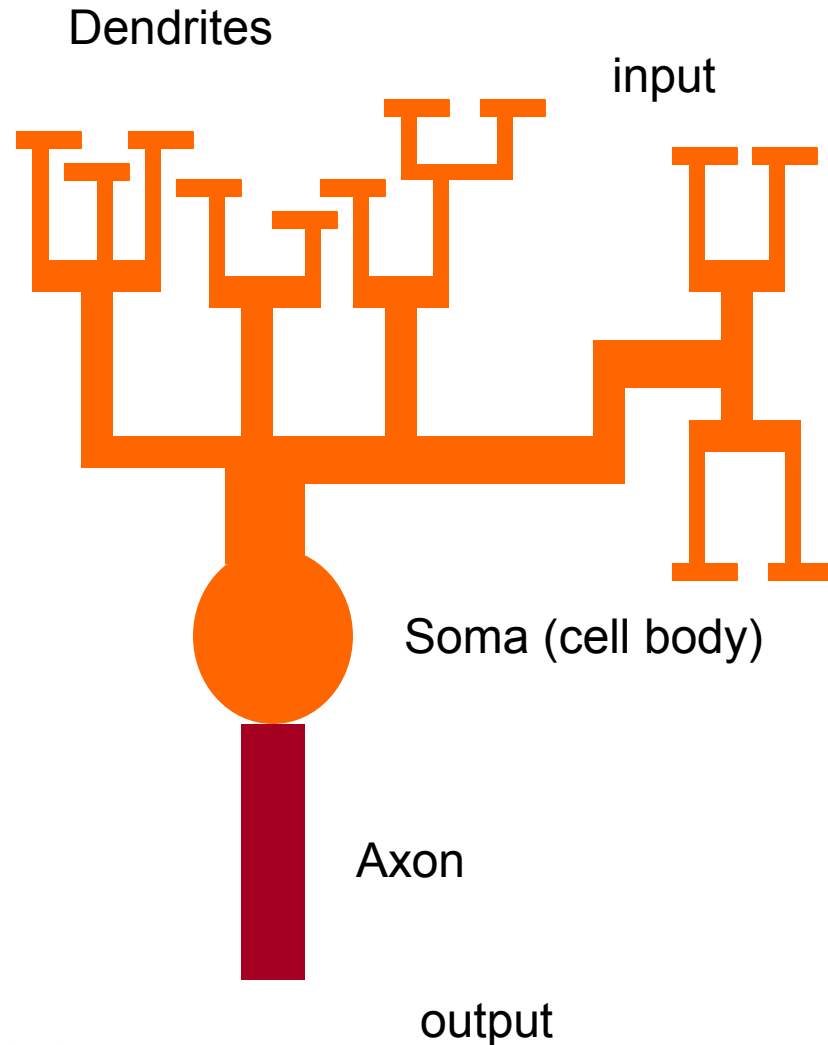
Neural Activation Function



Activation value (sum of individual inputs):

$$n_j = \sum_i x_i w_{ij}$$

Neural Activation Function



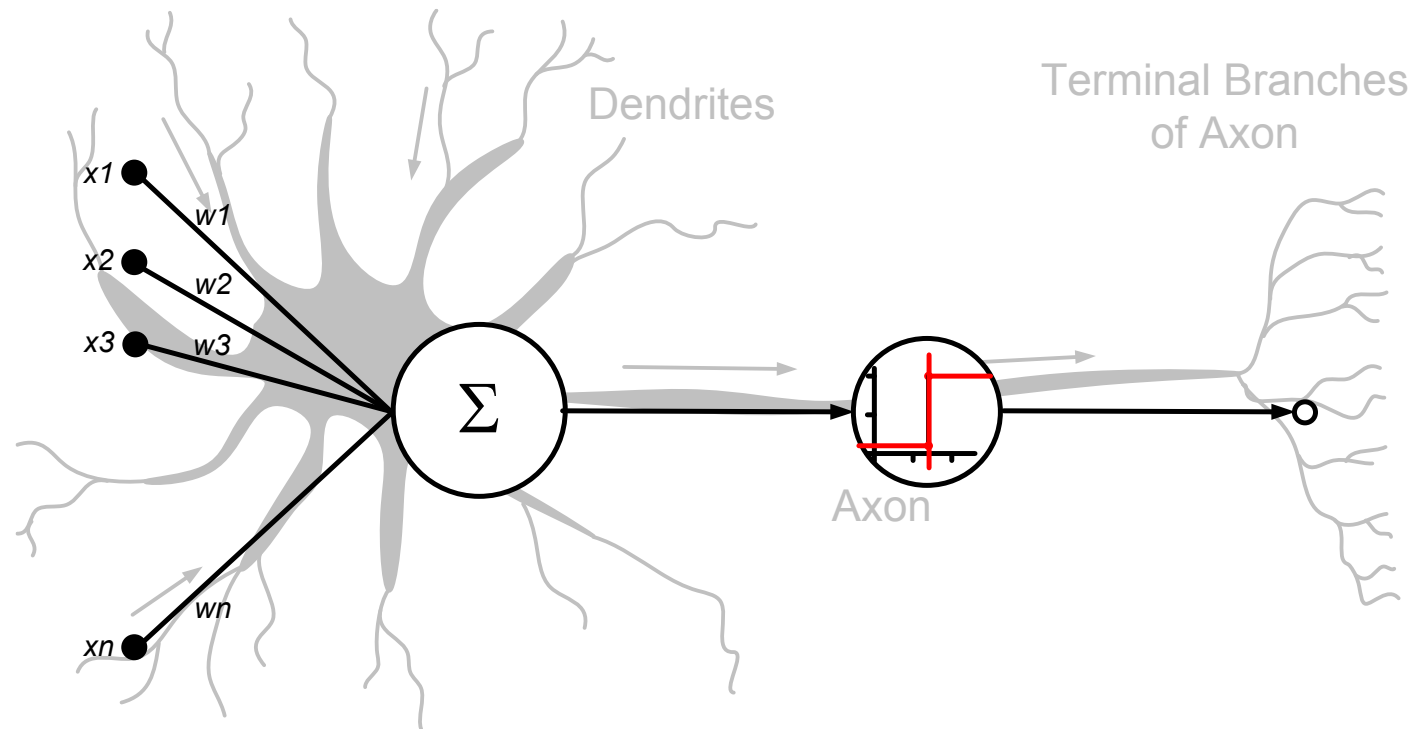
Activation value (sum of individual inputs):

$$n_j = \sum_i x_i w_{ij}$$

Activation function (sigmoid / logistic):

$$y_j = \frac{1}{1 + e^{-n_j}}$$

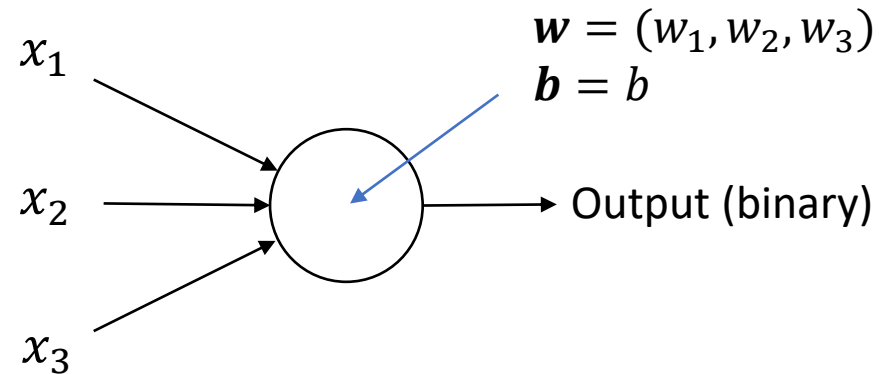
Computational Implementation of the Neural Activation Function



Neural Networks

Basic building block for composition is a *perceptron* (Rosenblatt c.1960)

Linear classifier – vector of weights w and a ‘bias’ b



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$w \cdot x \equiv \sum_j w_j x_j$$

Binary classifying an image

Each pixel of the image would be an input.

So, for a 28 x 28 image, we vectorize:

$$\mathbf{x} = 1 \times 784$$

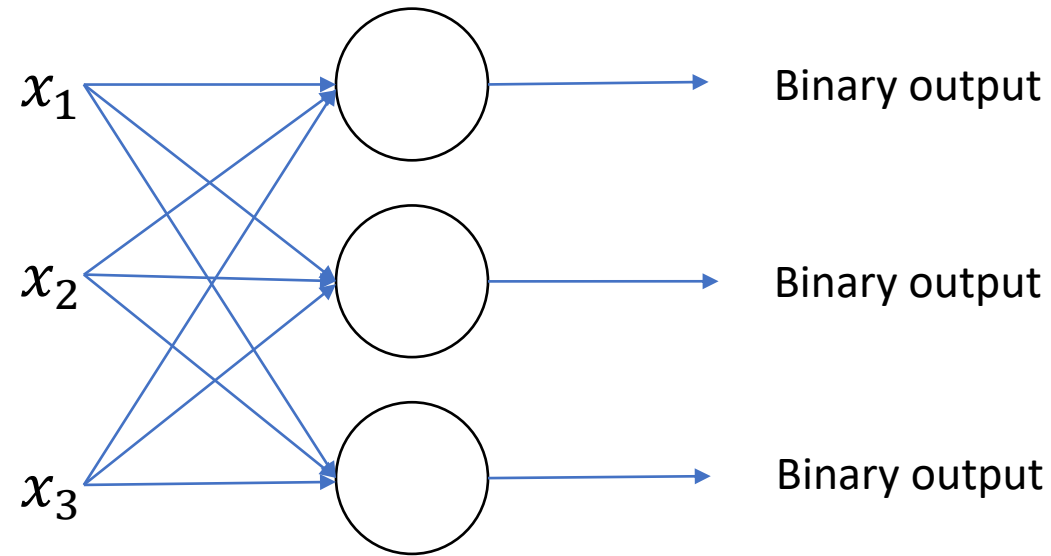
\mathbf{w} is a vector of weights for each pixel, 784 x 1

b is a scalar bias per perceptron

$$\text{Result} = \mathbf{xw} + b \quad \rightarrow (1 \times 784) \times (784 \times 1) + b = (1 \times 1) + b$$

Neural Networks - multiclass

Add more perceptrons



Multi-class classifying an image

Each pixel of the image would be an input.

So, for a 28 x 28 image, we vectorize.

$$\mathbf{x} = 1 \times 784$$

W is a matrix of weights for each pixel/each perceptron

$$\mathbf{W} = 784 \times 10 \text{ (10-class classification)}$$

b is a bias *per perceptron* (vector of biases); (1 x 10)

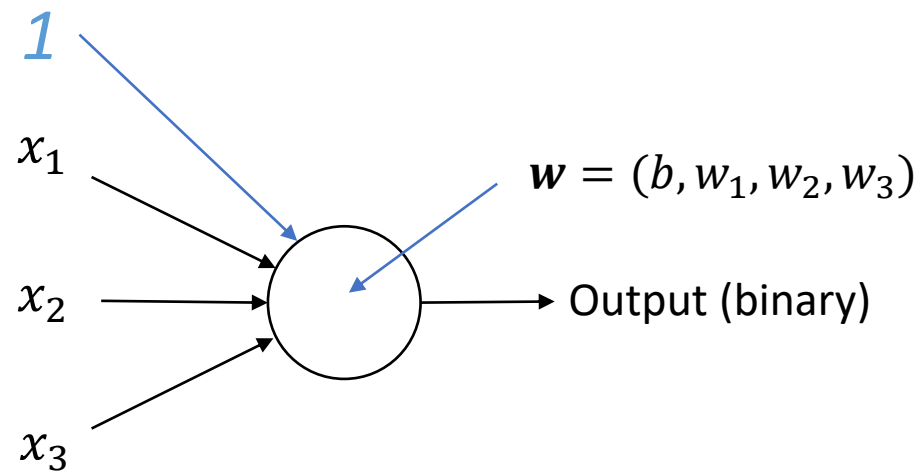
$$\text{Result} = \mathbf{xW} + \mathbf{b} \rightarrow (1 \times 784) \times (784 \times 10) + \mathbf{b}$$

$$\rightarrow (1 \times 10) + (1 \times 10) = \text{output vector}$$

Bias convenience

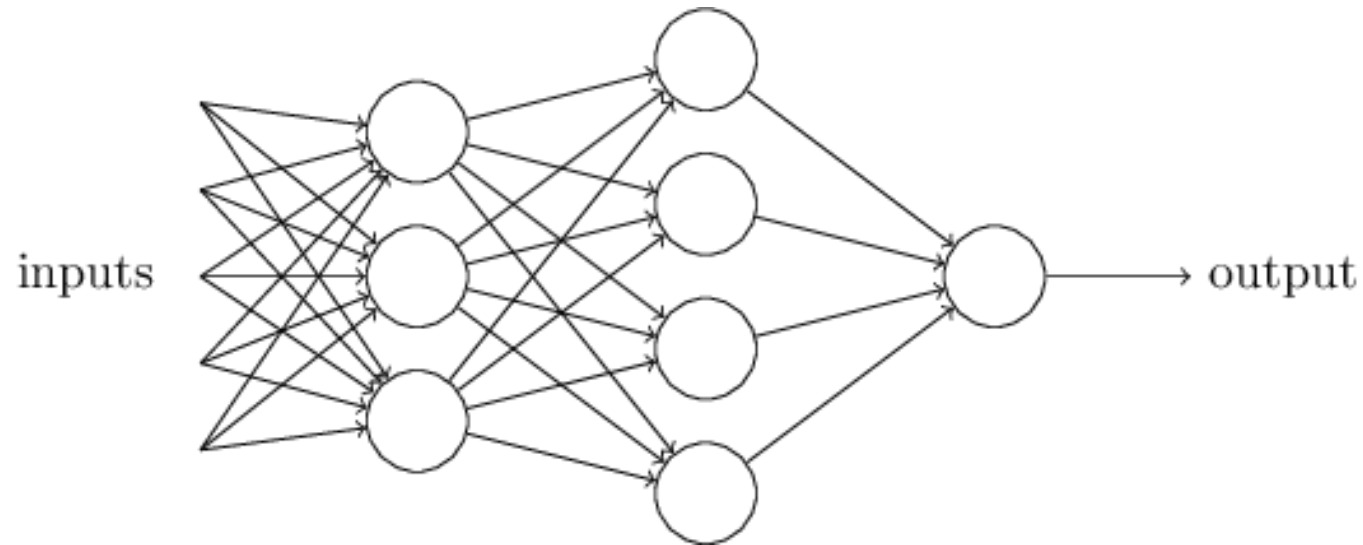
Let's turn this operation into a multiplication only:

- Create a 'fake' feature with value 1 to represent the bias
- Add an extra weight that can vary



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x \leq 0 \\ 1 & \text{if } w \cdot x > 0 \end{cases} \qquad w \cdot x \equiv \sum_j w_j x_j$$

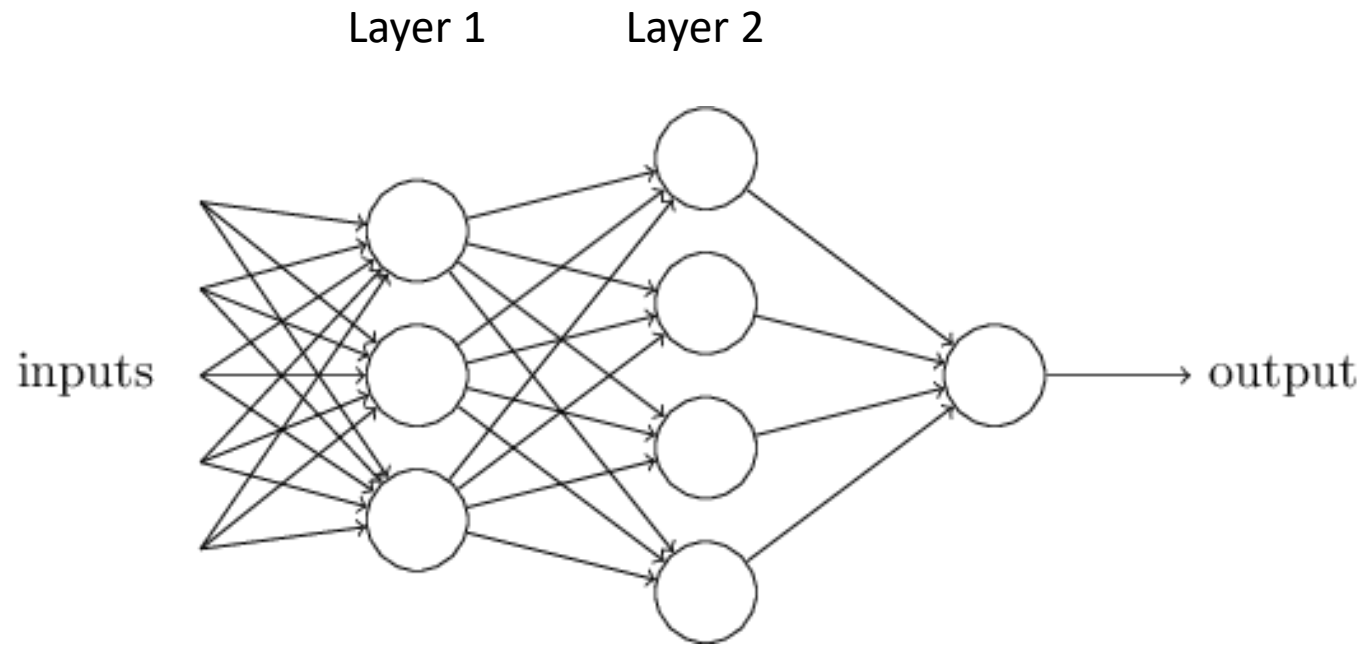
Composition



Attempt to represent complex functions as compositions of smaller functions.

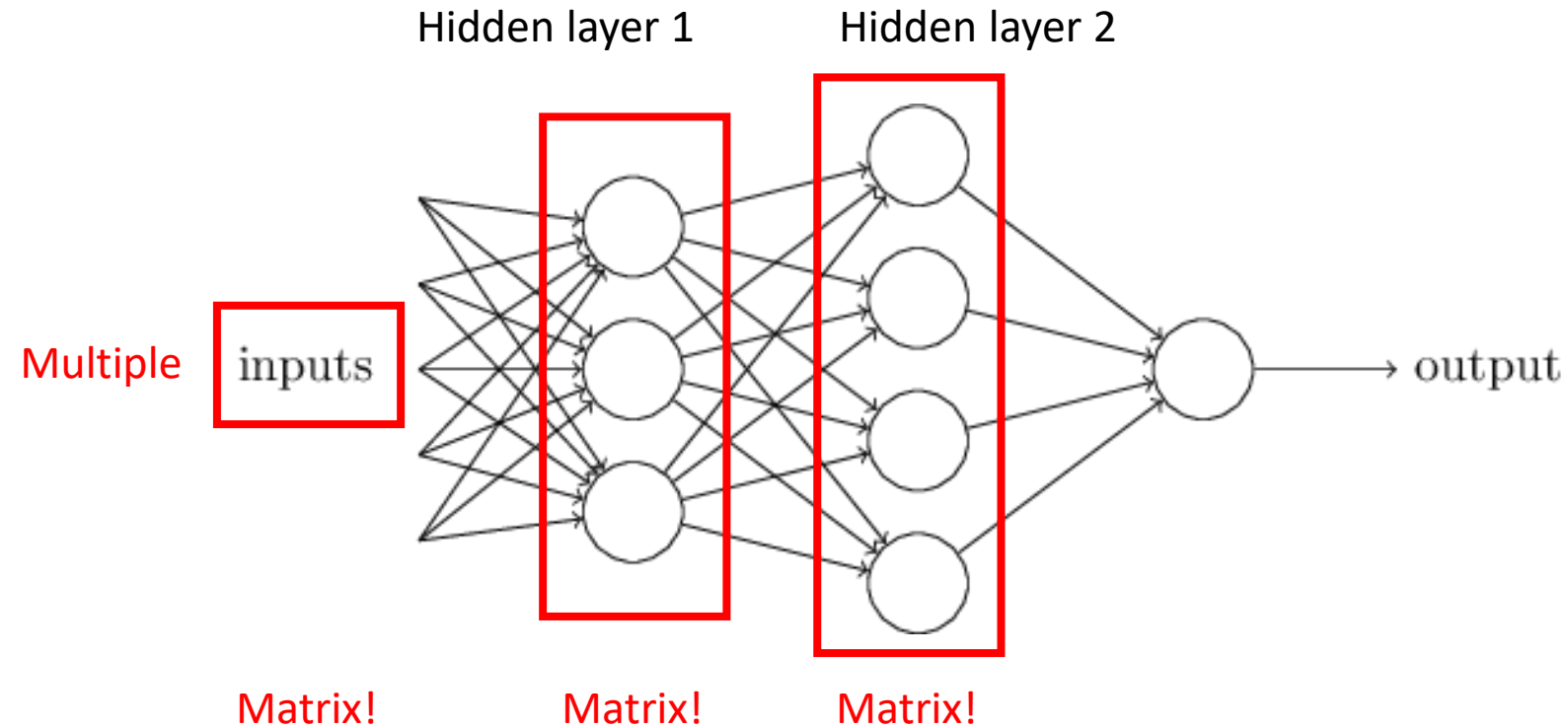
Outputs from one perceptron are fed into inputs of another perceptron.

Composition



Sets of layers and the connections (weights) between them define the *network architecture*.

Composition



It's all just matrix multiplication!
Matrix store weights/parameters!

GPUs -> special hardware for fast/large matrix multiplication.

Questions?

Questions?

Introduction to Neural Networks

Lecture 5

Introduction to Neural Networks

Lecture 5

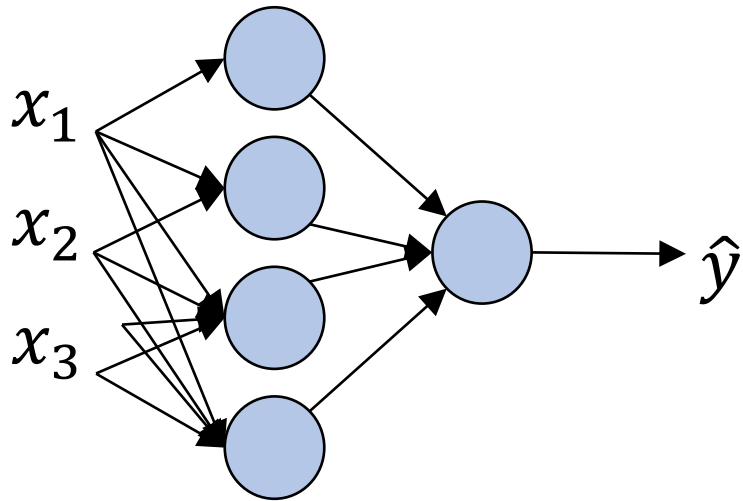
Non-linearity

Problem 1 with all linear functions

- We have formed chains of linear functions.
- We know that linear functions can be reduced
 - $g = f(h(x))$

Our composition of functions is really
just a single function : (

Problem 1 with all linear functions



$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]} \mathbf{x} + \mathbf{b}^{[1]}$$

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

$$\begin{aligned} \mathbf{z}^{[2]} &= \mathbf{W}^{[2]} \mathbf{z}^{[1]} + \mathbf{b}^{[2]} \\ &= \mathbf{W}^{[2]} [\mathbf{W}^{[1]} \mathbf{x} + \mathbf{b}^{[1]}] + \mathbf{b}^{[2]} \\ &= \mathbf{W}^{[2]} \mathbf{W}^{[1]} \mathbf{x} + \mathbf{W}^{[2]} \mathbf{b}^{[1]} + \mathbf{b}^{[2]} \\ &= \mathbf{W} \mathbf{x} + \mathbf{b} \end{aligned}$$

$$\hat{y} = \mathbf{z}^{[2]} = \mathbf{W} \mathbf{x} + \mathbf{b}$$

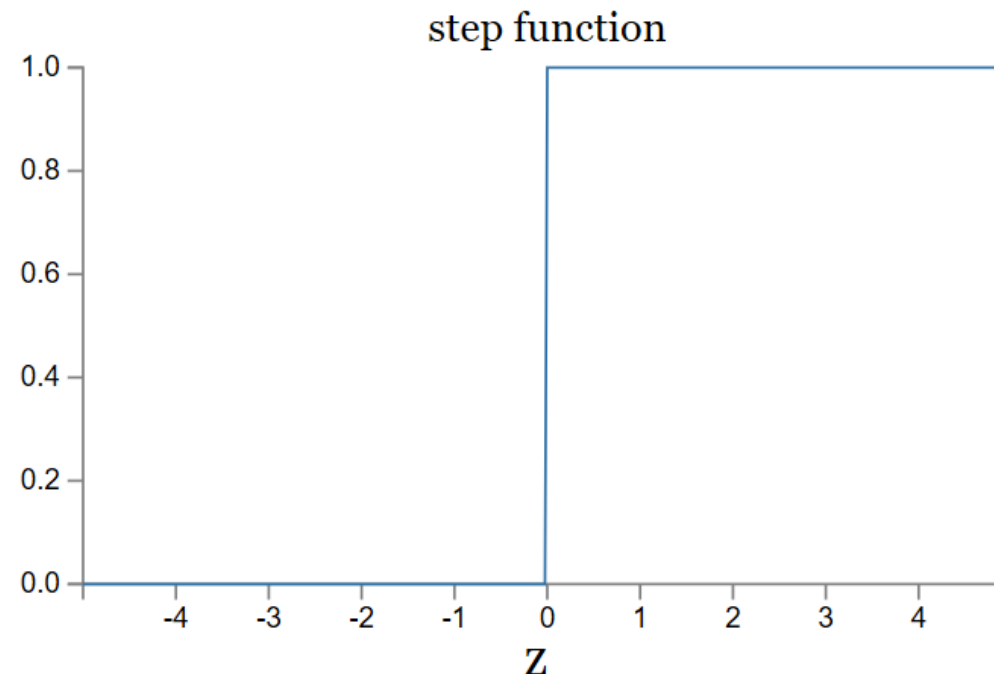
The output is always a linear function of the input!

Problem 2 with all linear functions

Linear classifiers:

small change in input can cause large change in binary output

*Activation function
for a perceptron:
Heaviside function*

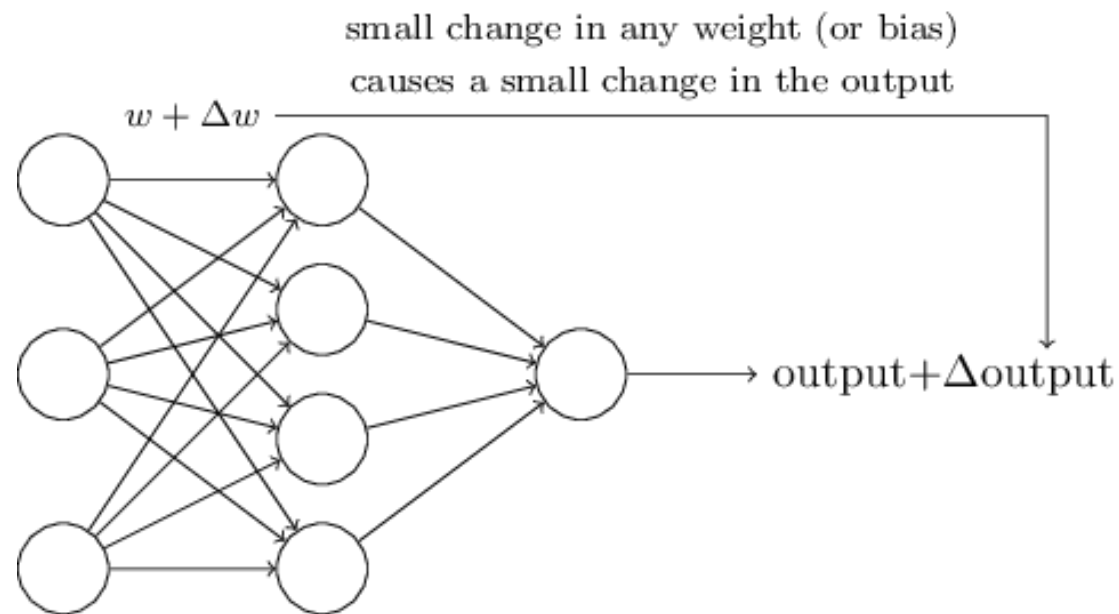


Problem 2 with all linear functions

Linear classifiers:

small change in input can cause large change in binary output.

We want:

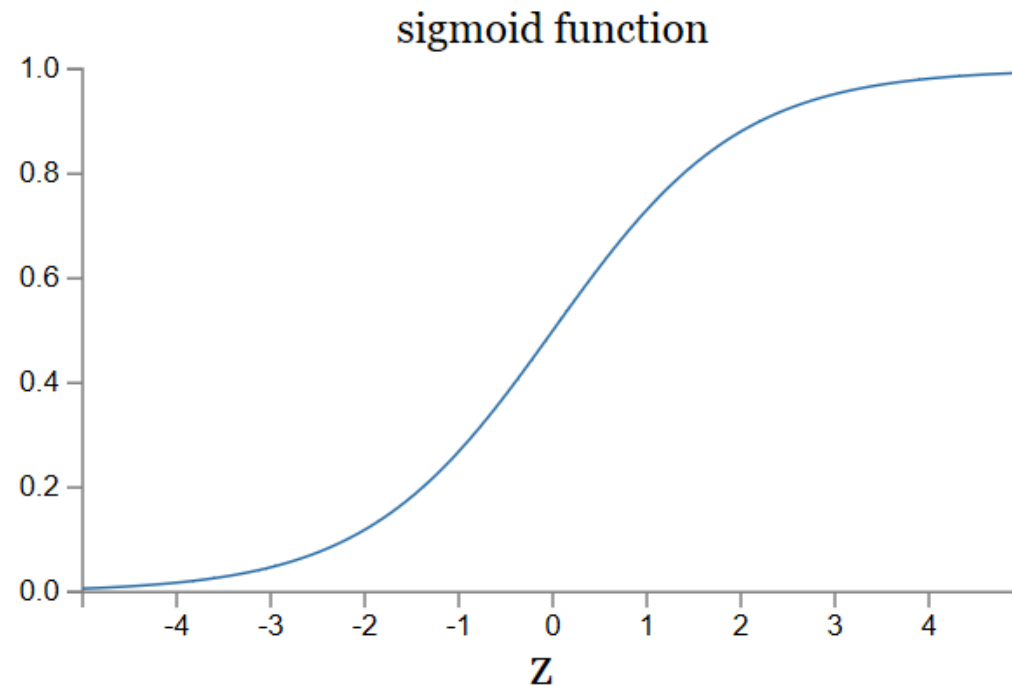


Let's introduce non-linearities

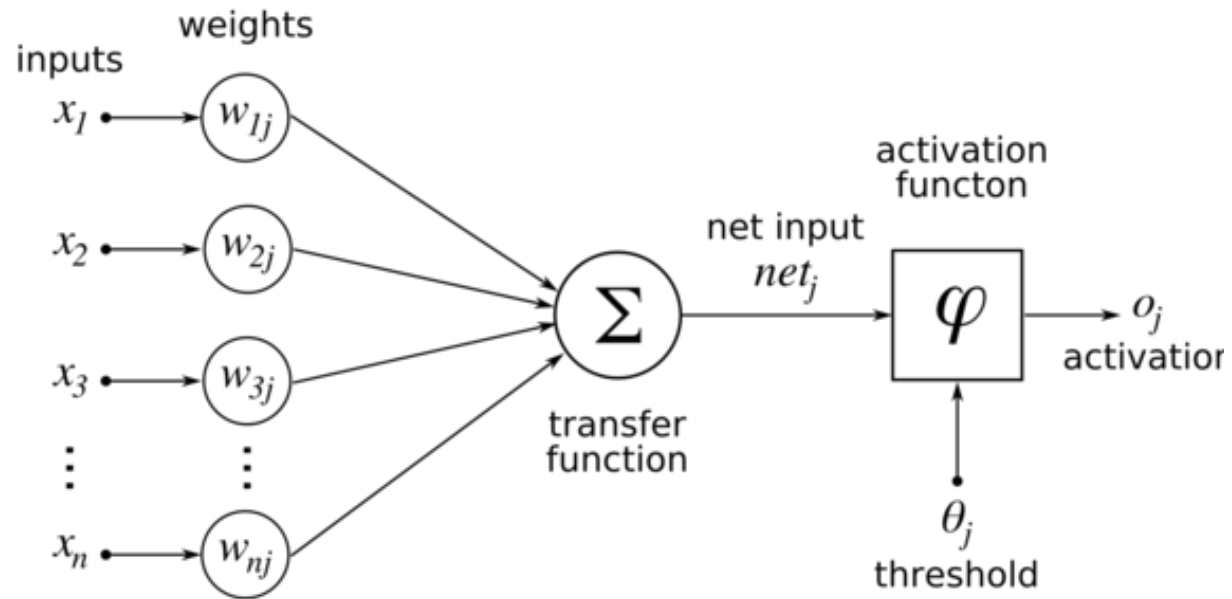
We're going to introduce non-linear functions to transform the features.

$$\sigma(w \cdot x + b)$$

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}.$$

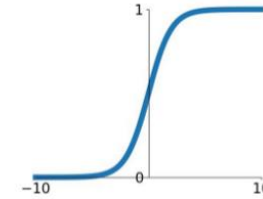


Activation Functions

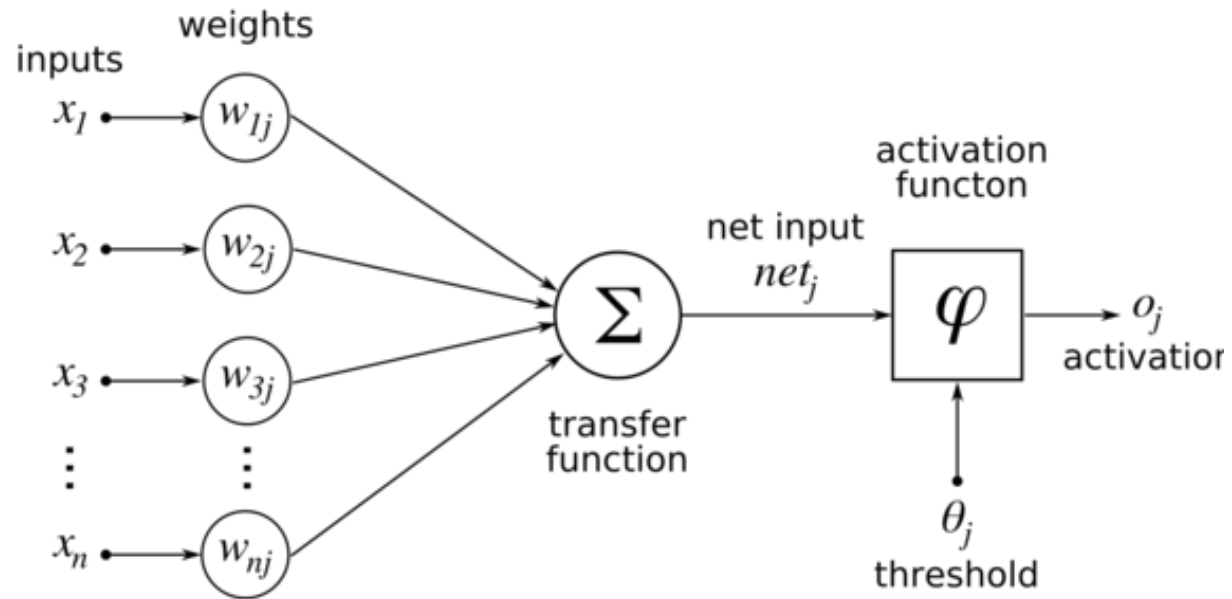


Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

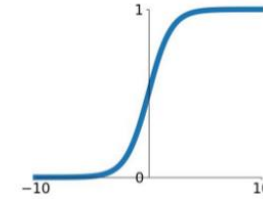


Activation Functions



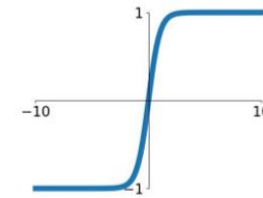
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

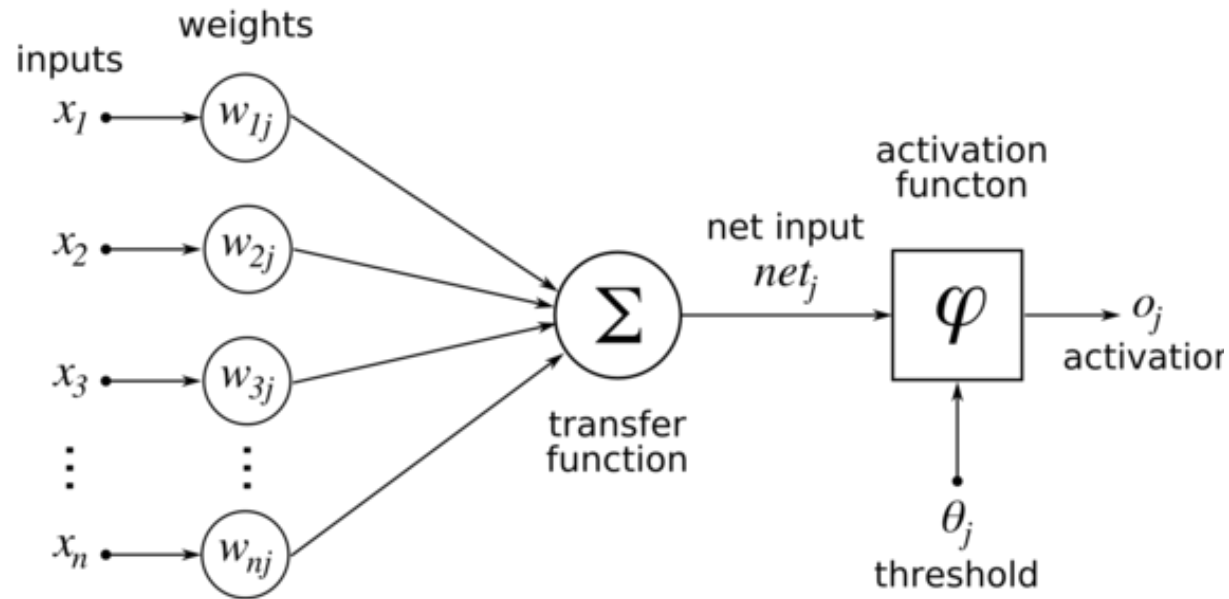


tanh

$$\tanh(x)$$

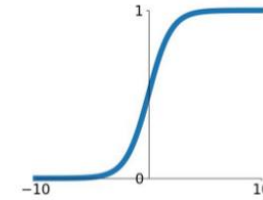


Activation Functions



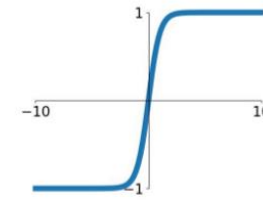
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



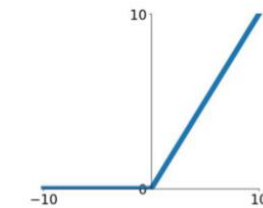
tanh

$$\tanh(x)$$



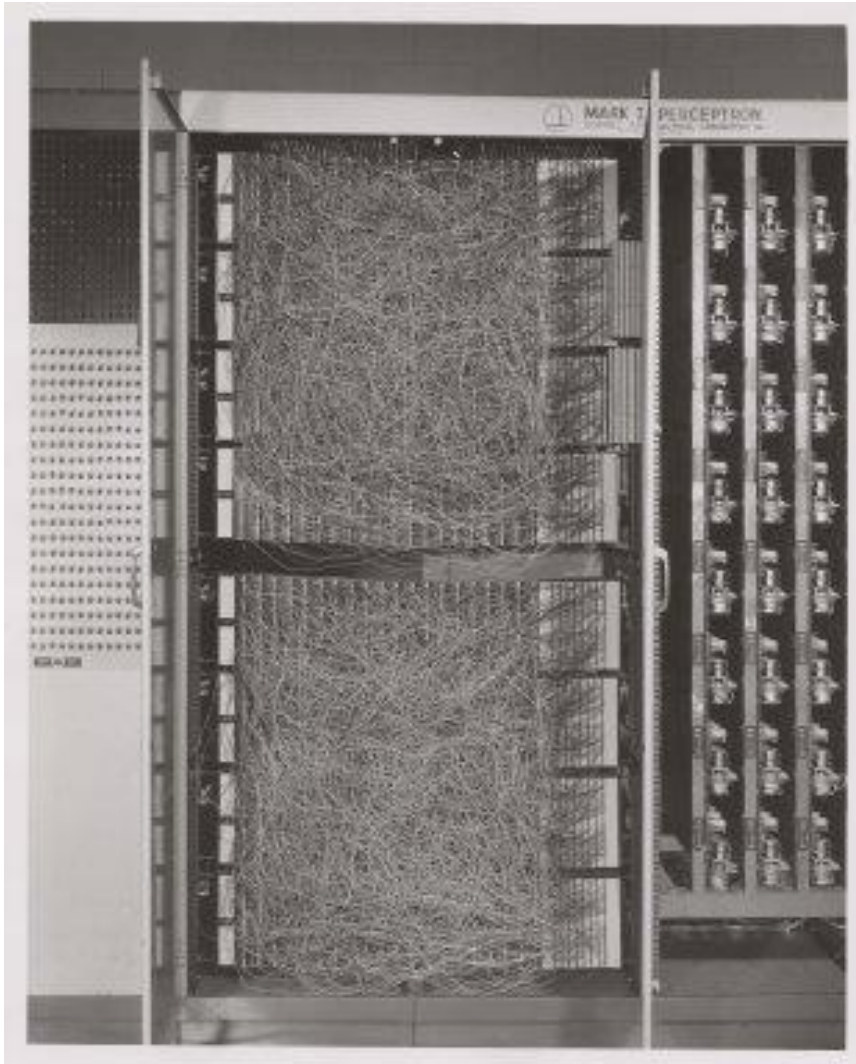
ReLU

$$\max(0, x)$$



Perceptron model

- Use is grounded in theory
 - Universal approximation theorem (Goodfellow 6.4.1)
- Can represent a NAND circuit, from which any binary function can be built by compositions of NANDs
- With enough parameters, it can approximate any function.



Mark 1 Perceptron
c.1960

20x20 pixel
camera feed

Perceptron model

*If a single-layer network can learn any function...
...given enough parameters...*

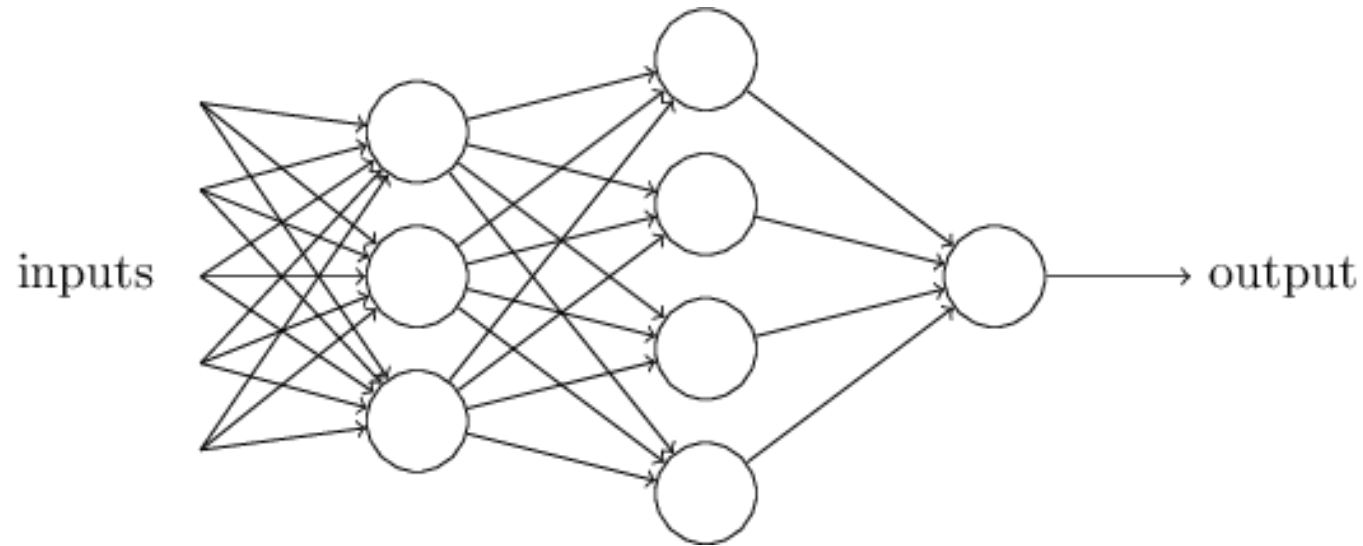
...then why do we go deeper?

Intuitively, composition is efficient because it allows *reuse*.

Empirically, deep networks do a better job than shallow networks at learning such hierarchies of knowledge.

Multi-layer perceptron (MLP)

- ...is a '*fully connected*' neural network with non-linear activation functions.



- '*Feed-forward*' neural network

Goals

- Build a classifier which is more powerful at representing complex functions *and* more suited to the learning problem.
- What does this mean?
 1. Assume that the *underlying data generating function* relies on a composition of factors.
 2. Learn a feature representation that is specific to the dataset.

MLP performance on MNIST

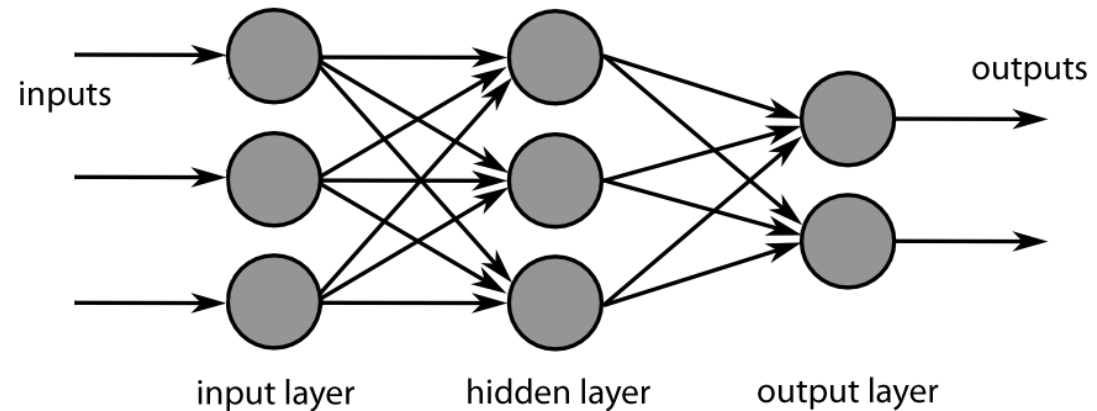
- MNIST
 - Dataset of handwritten digits
 - 60K training samples
 - 10K testing samples
- A 6-layer MLP [1]
- Number of neurons
 - 2500, 2000, 1500, 1000, 500, and 10
- Performance - 99.65%



[1] Cireşan, Dan Claudiu, et al. "Deep, big, simple neural nets for handwritten digit recognition." *Neural computation* 22.12 (2010): 3207-3220.

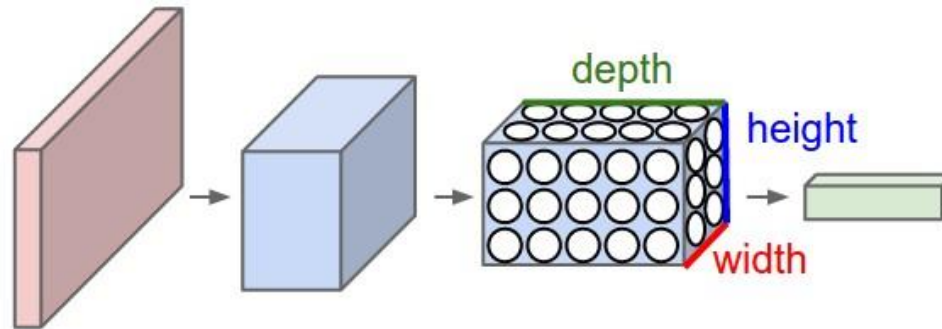
Why we need CNN?

- Neural Network
 - Fully connected layers
- Input
 - Hand-crafted features
 - Pixel values
- Hand-crafted features
 - Limitations
- Image as input
 - Size of feature vector = $H \times W \times C$
 - For 256x256 RGB image
 - 196608 dimensions



Why we need CNN?

- CNN - Special type of neural network
 - Operate with volume of data
 - Weight sharing in form of kernels



Source: <http://cs231n.github.io>

MLP still very useful



NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, ECCV 2020

Questions?

Sources for this lecture include materials from works by Abhijit Mahalanobis, James Tompkin, Sedat Ozer, and Ulas Bagci