# Intro to ML

EECS 442 Discussion
Fall 2023

# Image Classification

**Input Image** | **Output Class**

dog
cat
mountain
ship
.
.

dog
cat
mountain
ship
.
.

# IMAGENET

**22K** categories and **15M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
- Food
- Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activity

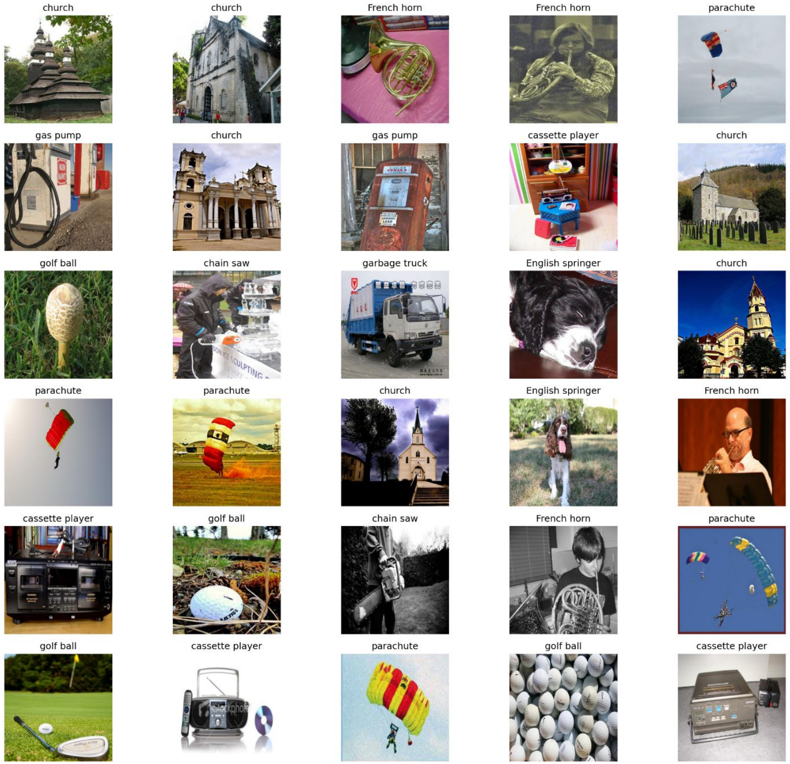
**www.image-net.org**

Deng et al. 2009,
Russakovsky et al. 2015

Slide credit: Fei-Fei Li

# Tiny ImageNet

- Smaller version of ImageNet
- 200 Classes
- Training set: 100k images
- Validation set: 10k images
- Resolution: 64x64x3



# L2 Distance

Calculate the distance between two images

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$


Square of difference

| | |
|---|---|
| 225 | 2809 |
| 49 | 43 |

Called L2 Distance between two images

Add and take square root

$$= \sqrt{225 + 2809 + 49 + 43}$$
$$= 55.91$$

Hint for broadcasting: // x-y // ^2 = // x // ^2 + // y // ^2 - 2x^T y

# K-Nearest Neighbors

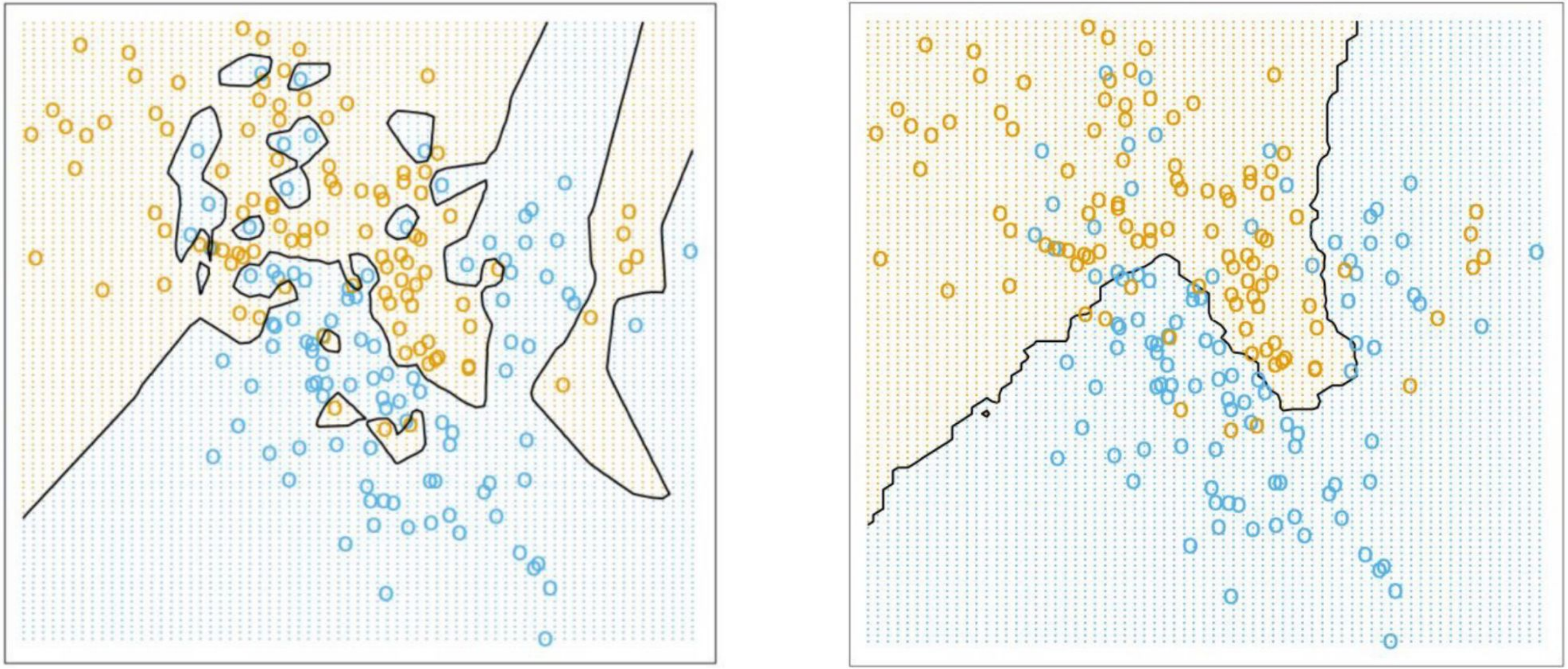
- Can be applied to any type of data with the right distance metric
- Memorizing the training data and labels and predicting the label of the most similar training image

For different values of *k*:

- Find the *k* points with the shortest L2 distance from *x*
- These *k* points are called k-Nearest Neighbors to the point *x*
- *k* is a hyperparameter that we can tune

# How this looks

Demo: http://vision.stanford.edu/teaching/cs231n-demos/knn/



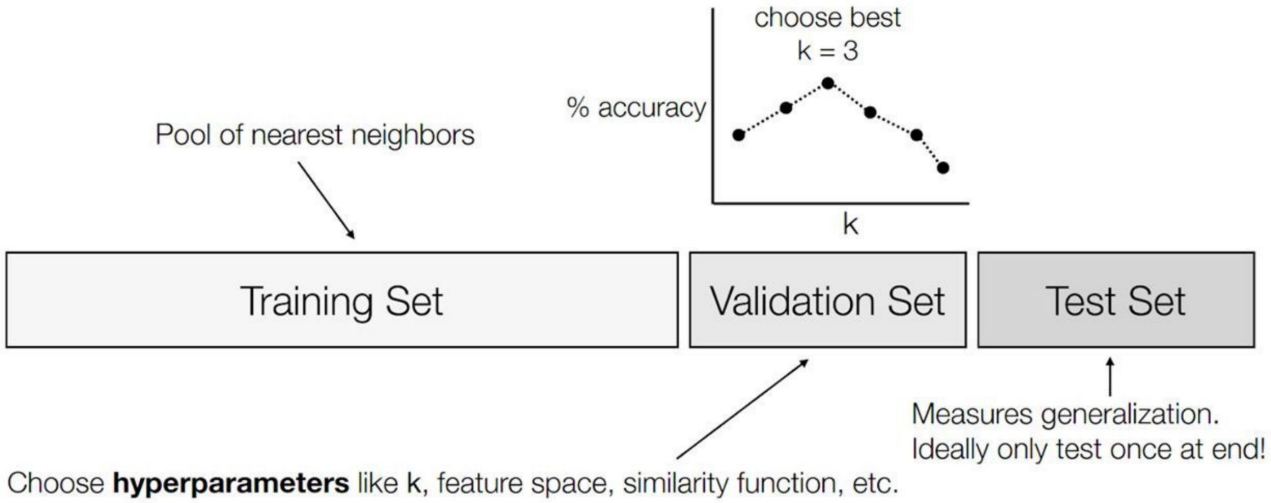
$k = 1$ $k = 15$

# Splitting dataset & Choosing *k*

Machine Learning: generalize in the wild by training on a dataset that is representative of the samples to which we want to apply our system

To measure how good our system will be:

- Calculate its accuracy on a “test set”
- Only do this once at the end of training

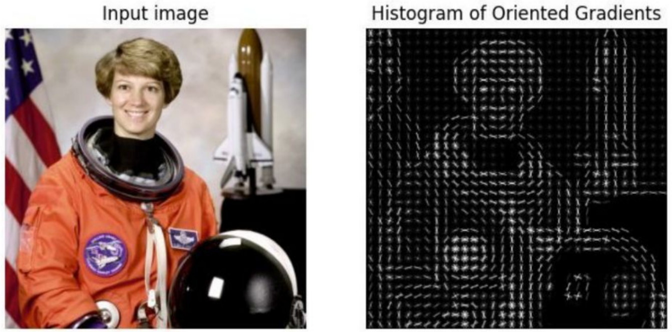


# Histogram of Oriented Gradients (HOG)

- Edge and gradient based descriptors
- Uniquely describe the features of images

Steps:

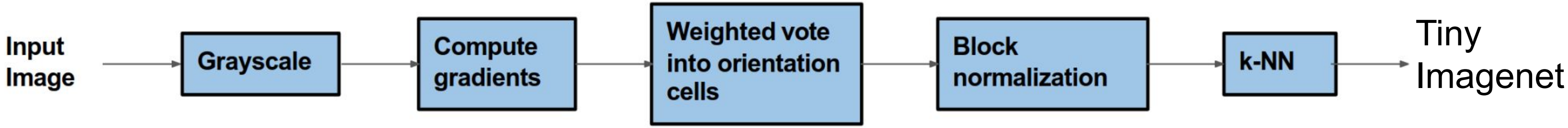
1. Compute the orientations of the gradients
2. Create a histogram of the edge orientations, with votes weighted by the gradient magnitudes
3. Perform block normalization across the histogram




*N. Dalal and B. Triggs. Histograms of oriented gradients for human detection.*

# HOG Classification Workflow

Simplified version of HOG:

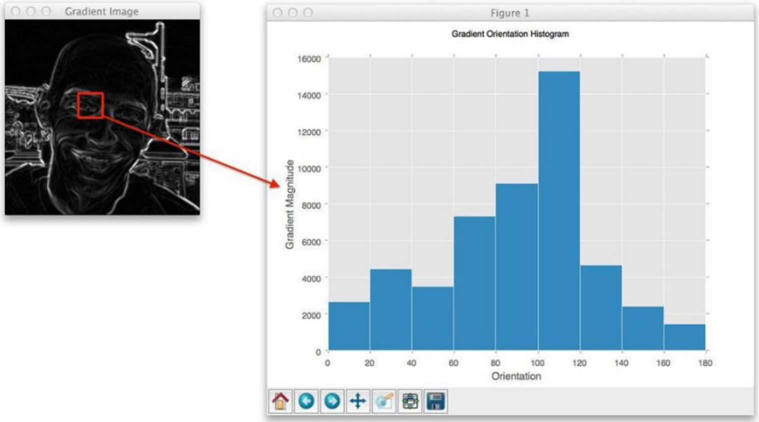


# Computing Orientations of the Image Gradients

- Compute gradients of an image along the horizontal and vertical directions
  - Convolve with dx = [1 -1] and dy = [1 -1]^T from ps1
  - Or convolve with a Sobel filter
- Compute magnitude of each gradient
  - Magnitude = sqrt(Gx^2 + Gy^2)
- Compute orientation/angle of each gradient
  - Orientation = tan^(-1) (Gy, Gx)
  - Use modulo to convert angles to degree in range [0, 180]
- Get an orientation for every pixel

# Create Histogram from Orientations and Magnitudes

- Iterate through each pixel in every cell
- Weigh the vote of the orientation base on its magnitude
- Place the vote into the bin where the orientation falls




N. Dalal and B. Triggs. *Histograms of oriented gradients for human detection.*
[Histogram of Oriented Gradients (and car logo recognition) – PyImageSearch]

# Linear Classifiers

x : (32x32x3)

Flatten image

x : (3072, )

```
img = img.reshape(img.shape[0], -1)
```

**Linear Classifier**

$f(x) = Wx + b$

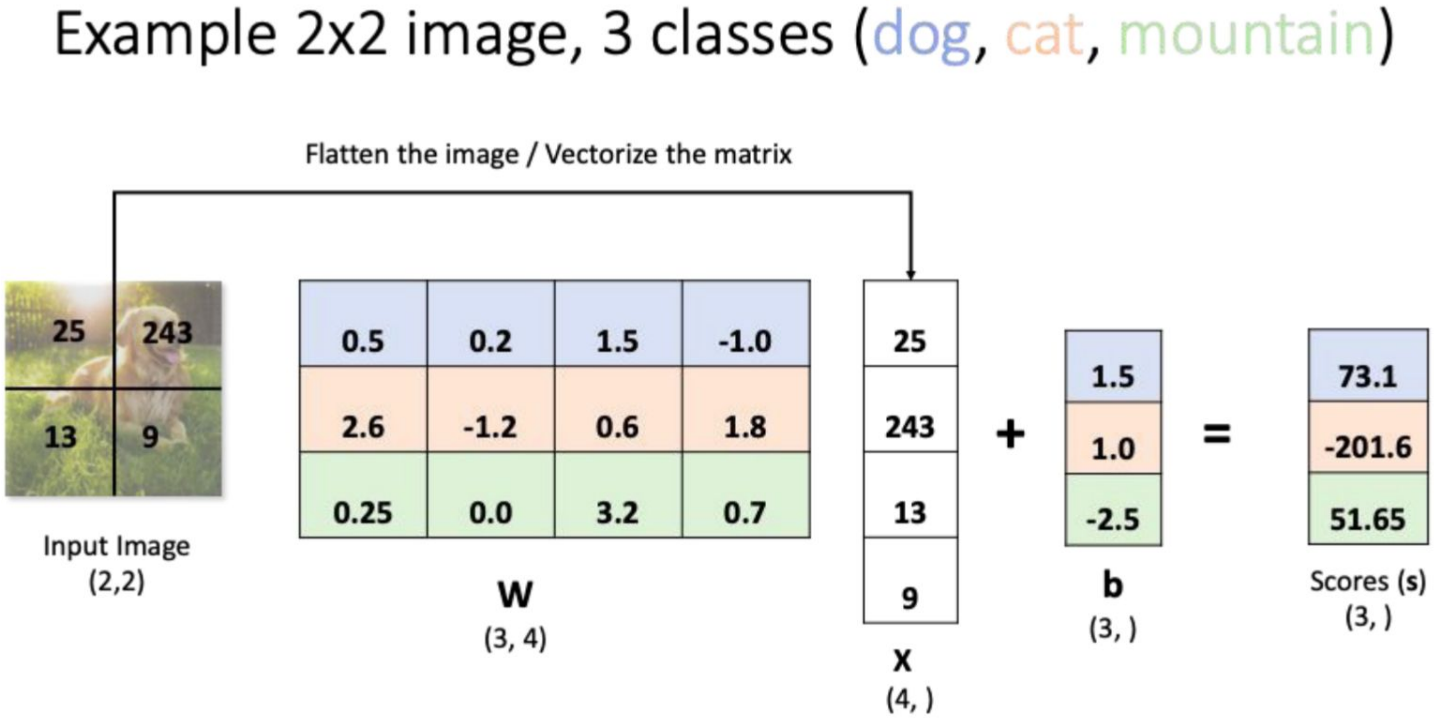
W: (10, 3072)

b: (10, )

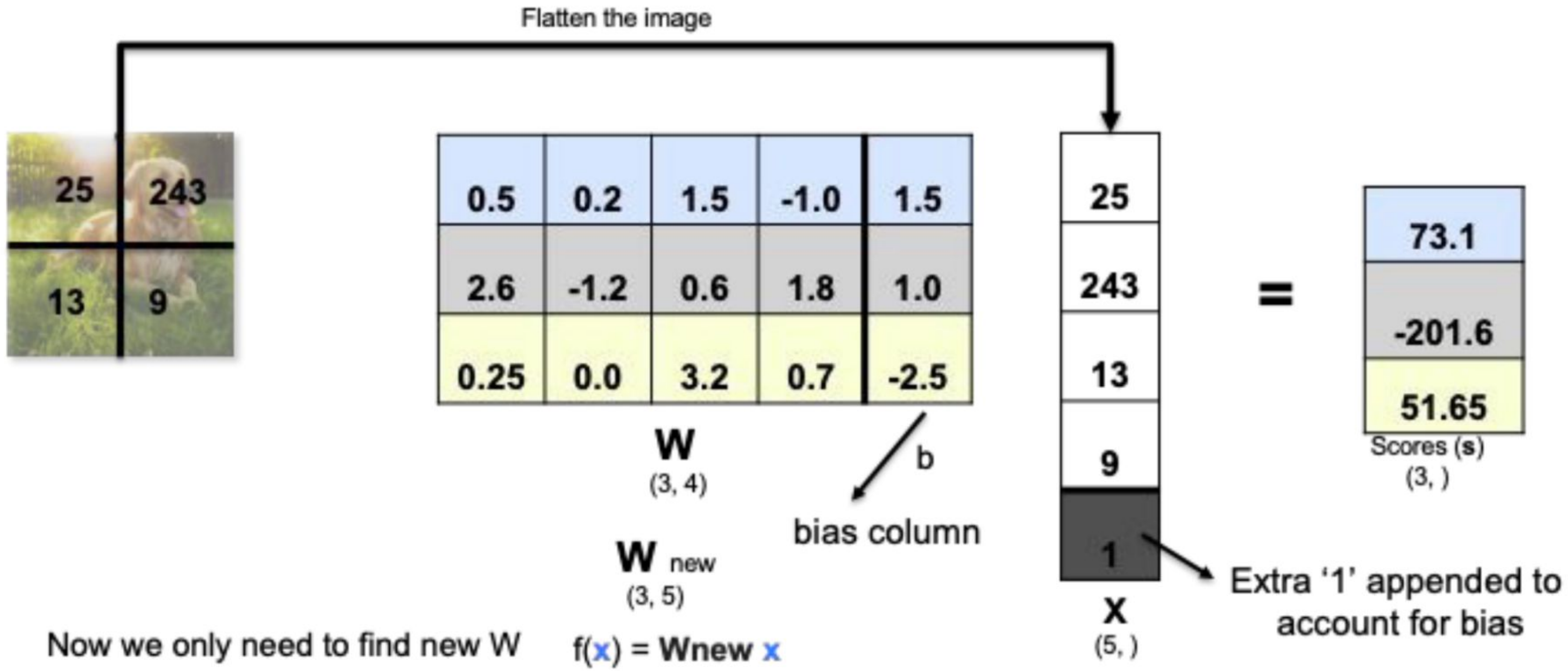
| | |
|---|---|
| **250.75** | **dog** |
| 30.5 | cat |
| 67.325 | mountain |
| -20.25 | ship |
| . | . |
| . | . |

10 possible classes in total

# Linear Classifiers

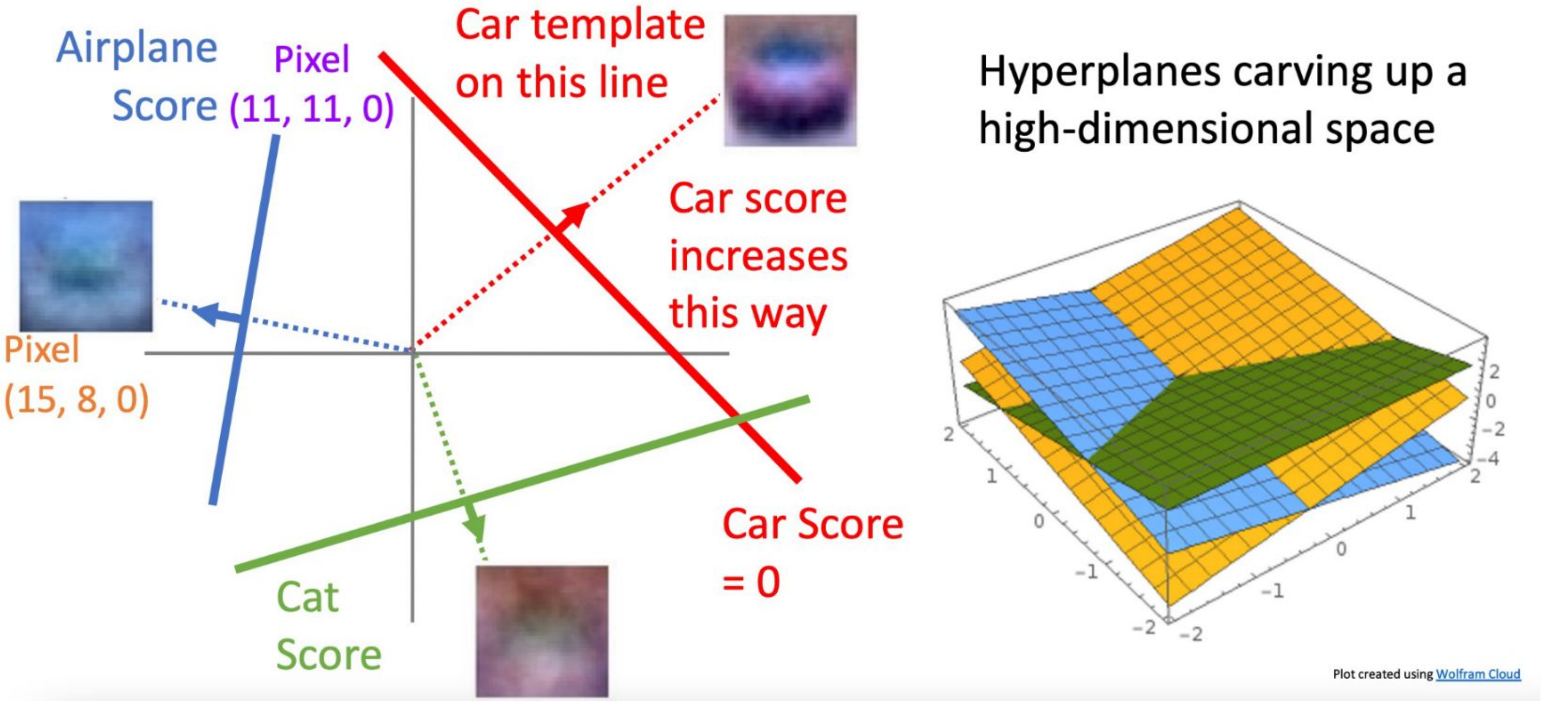


# Bias Trick



Flatten the image
25
243
13
9
0.5 0.2 1.5 -1.0 1.5
2.6 -1.2 0.6 1.8 1.0
0.25 0.0 3.2 0.7 -2.5
W
(3, 4)
b
bias column
W new
(3, 5)
25
243
13
9
1
x
(5, )
=
73.1
-201.6
51.65
Scores (s)
(3, )
Extra '1' appended to account for bias
Now we only need to find new W
f(x) = Wnew x

# Linear Classifier Boundary



# Loss Function

Computes how much current model's prediction deviates from target

K: number of classes

**Cross entropy**

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k \longleftarrow \text{easy to optimize, good approximation}$$

Adapted from: Isola, Torralba, Freeman

# Softmax loss / Multinomial logistic loss

- Softmax activation followed by a cross entropy loss
- Note: in pytorch, “cross entropy loss” function already has a softmax built in

S: score, y: label for given image

**For a single image, x** with label ’y’

$$L(s, y) = -\log \frac{e^{s_{y_i} - \max(s_j)}}{\sum_{j=1}^{C} e^{s_j - \max(s_j)}}$$

**Probabilities**

| | Scores, **s** = Wx | exp → $e^{s_y}$ | normalize → $\frac{e^{s_y}}{\sum_{j=1}^{3} e^{s_j}}$ |
|---|---|---|---|
| Dog | 2.5 | 12.18 | 0.06 |
| Cat | -1.3 | 0.27 | 0.0 |
| Mountain | 5.2 | 181.27 | 0.94 |

y = 1, 2, 3
1: Dog
2: Cat
3: Mountain

In this case, class y = 1 (Dog)

$$L(s, 1) = -\log 0.06 = 2.81$$

# Softmax loss properties

$$L(s, y) = -\log \frac{e^{s_{y_i} - \max(s_j)}}{\sum_{j=1}^{C} e^{s_j - \max(s_j)}}$$

C: Number of classes

What is the min/max possible Value of L(s, y) ? → Min 0, Max +infinity

If all scores were really small and thus approximately the same, What would be the value of L(s, y)? → log(C)

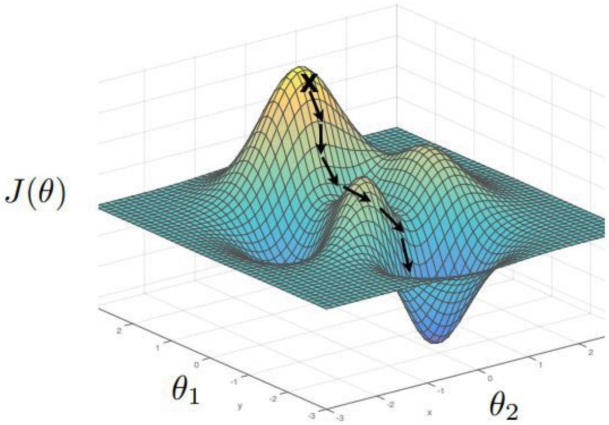
# Gradient Descent

We want to optimize some objective function ***J***

$$\boxed{\theta^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}_{J(\theta)}}$$

One iteration of gradient descent:

$$\theta^{t+1} = \theta^t - \underbrace{\eta_t}_{\text{learning rate}} \nabla_\theta J(\theta)\Big|_{\theta=\theta^t}$$



# Batch Gradient Descent

Loss function is $J(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(x_i, y_i, \theta)$

Its gradient is the sum of gradients for each example $\nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(x_i, y_i, \theta)$

Requires iterating over every training example in each gradient step.

# Stochastic Gradient Descent

Sample a point instead of looping over all training examples

$$\nabla J(\theta) \approx \frac{1}{|B|} \sum_{i \in B} \nabla L(x_i, y_i, \theta)$$

where B is a minibatch – a random subset of examples