Lecture 2:
Image Classification

Administrative: Assignment 0

- Due this Thursday 10/03 by 11:59pm
- Easy assignment
- Hardest part is learning how to use colab and how to submit on gradescope
- Worth 1% of your grade
- Used to evaluate how prepared you are to take this course

Administrative: Assignment 1

Due 10/10 11:59pm

- K-Nearest Neighbor
- Linear classifiers: SVM, Softmax

Administrative: Course Project

Project proposal due 10/29 11:59pm

Find your teammates on EdStem. We will help find teammates as well (we will release a partner form in the next few weeks)

Collaboration: EdStem

“Is X a valid project for 493G1?”

- Anything related to deep learning
- Maximum of 3 students per team
- Make a EdStem private post or come to TA Office Hours

More info on the website

Final project

- Groups of up to 3
- You can form groups yourselves
    - For students looking for groups, we will help assign you
- Anything related to deep learning

Administrative: Fridays

This Friday 9:30-10:30am and again 12:30-1:30pm

Project Info Session + Assignment 1 Prep (Python + Numpy continued)

Presenter: TBD

Syllabus

Deep learning Fundamentals	Practical training skills	Applications
Data-driven approaches	Pytorch 1.4 / Tensorflow 2.0	Image captioning
Linear classification & kNN	Activation functions	Interpreting machine learning
Loss functions	Batch normalization	Generative AI
Optimization	Transfer learning	Fairness & ethics
Backpropagation	Data augmentation	Data-centric AI
Multi-layer perceptrons	Momentum / RMSProp / Adam	Deep reinforcement learning
Neural Networks	Architecture design	Self-supervised learning
Convolutions		Diffusion
RNNs / LSTMs		LLMs
Transformers		

Image Classification

A Core Task in Computer Vision

Today:

- The image classification task
- Two basic data-driven approaches to image classification
  - K-nearest neighbor and linear classifier

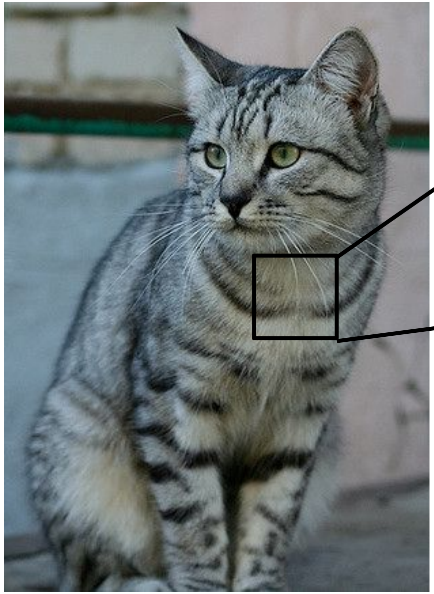
Image Classification: A core task in Computer Vision

(assume given a set of possible labels)

→

Cat
Dog
Bird
Truck
Plane

The Problem: Semantic Gap



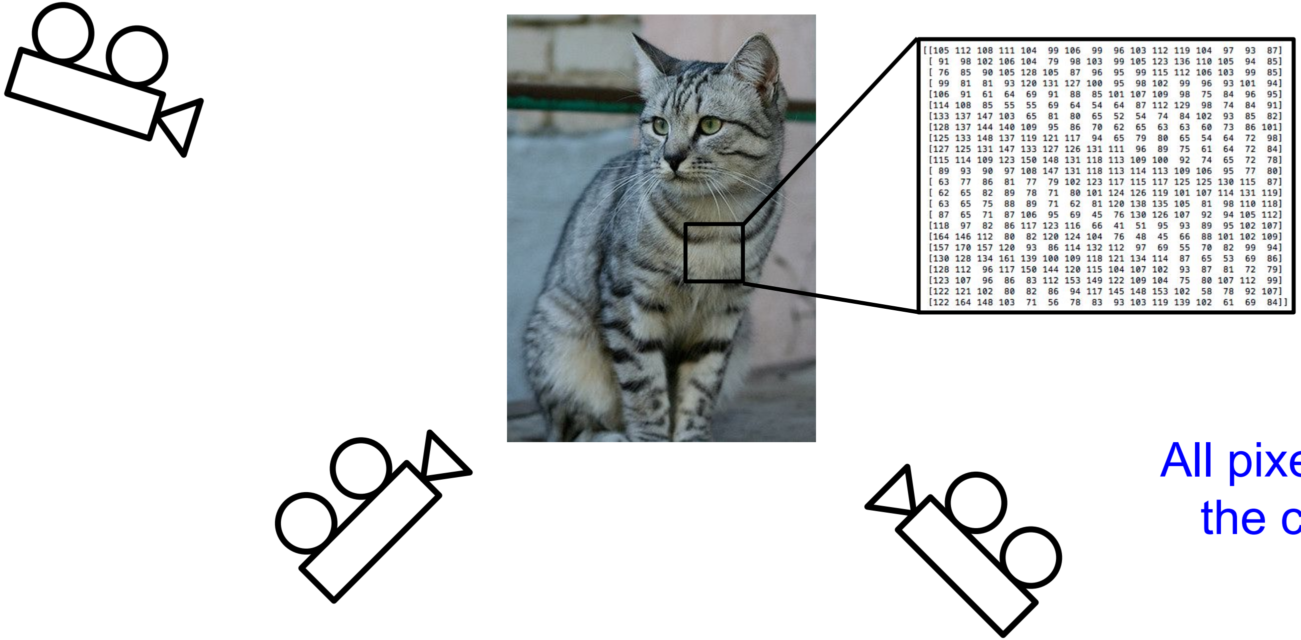

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is a tensor of integers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint variation



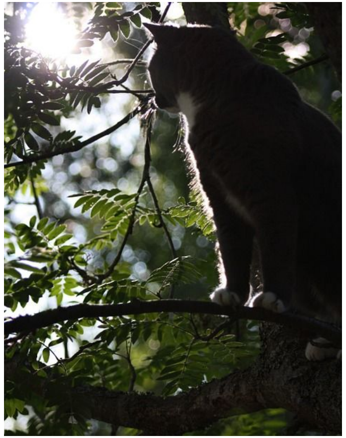
All pixels change when the camera moves!

Challenges: Illumination

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

RGB values are a function of surface materials, color, light source, etc.

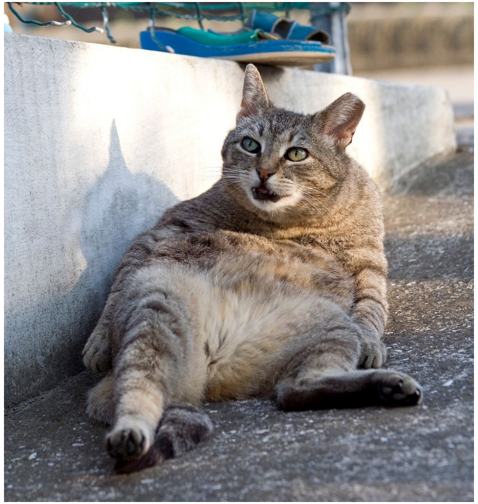
Challenges: Background Clutter

This image is CC0 1.0 public domain

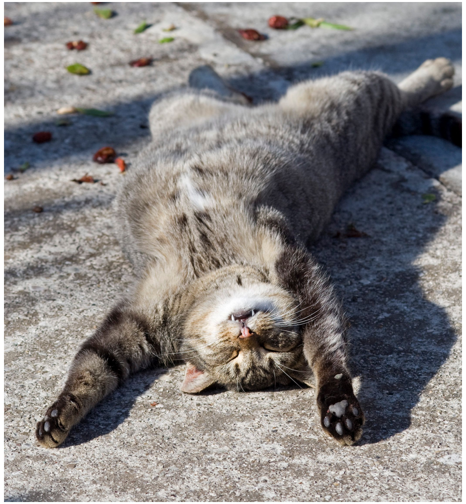
This image is CC0 1.0 public domain

Challenges: Occlusion

Challenges: Deformation

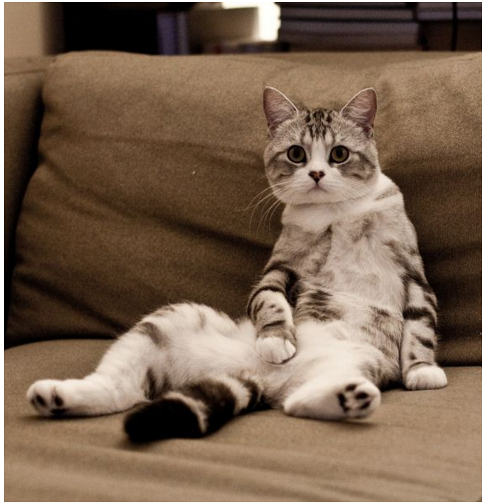


This image by Umberto Salvagnin is licensed under CC-BY 2.0



This image by Umberto Salvagnin is licensed under CC-BY 2.0

This image by sare bear is licensed under CC-BY 2.0



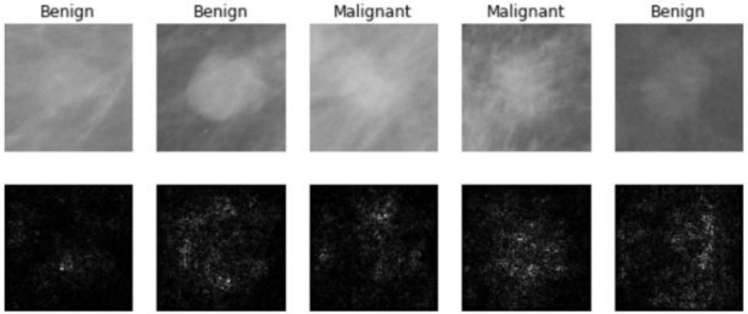
This image by Tom Thai is licensed under CC-BY 2.0

Challenges: Intraclass variation

This image is CC0 1.0 public domain

Image classification is a building block for other tasks

Medical Imaging



Levy et al, 2016 Figure reproduced with permission

Galaxy Classification

Dieleman et al, 2014

Whale recognition

[Kaggle Challenge]

Image classification is a building block for other tasks

Image Captioning

Vinyals et al, 2015
Karpathy and Fei-Fei, 2015

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

Captions generated by Justin Johnson using Neuraltalk2

Image classification is a building block for other tasks

Example: Playing Go

→

(1, 1)

(1, 2)

...

(1, 19)

...

(19, 19)

Where to play next?

Modern computer vision algorithms

Classifiers today take 1ms to classify images. And can handle thousands of categories.

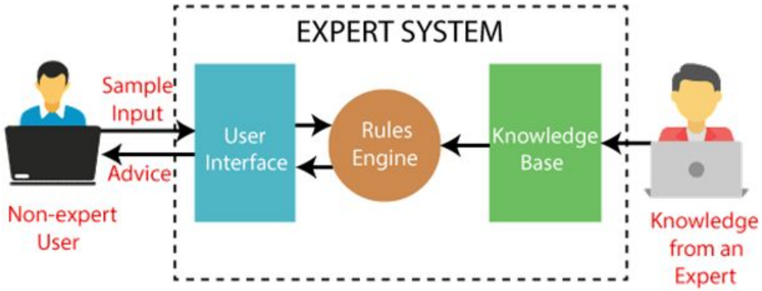
An image classifier: can we implement this as a normal software function?

```
def classify_image(image):
  # Some magic here?
  return class_label
```

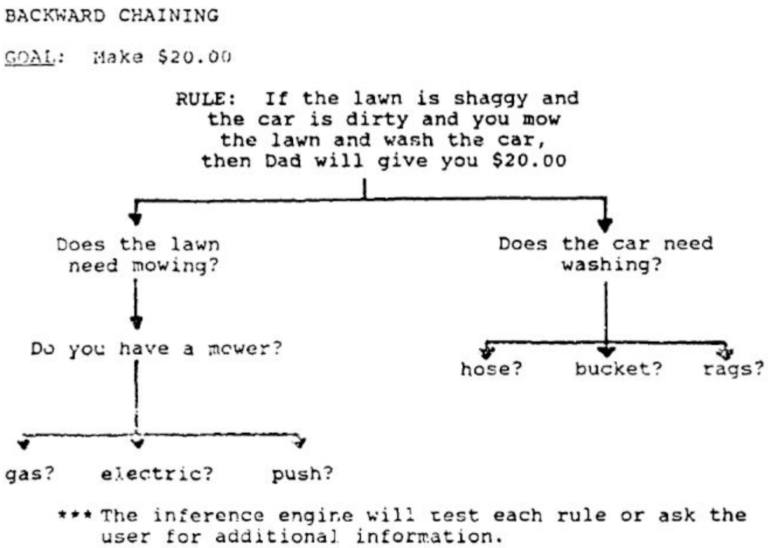
Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

This is why expert systems in the 80s led to the AI winter.



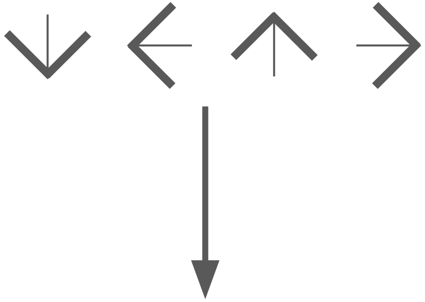
Originally called heuristic programming project.



Attempts have been made

Find edges

Find corners



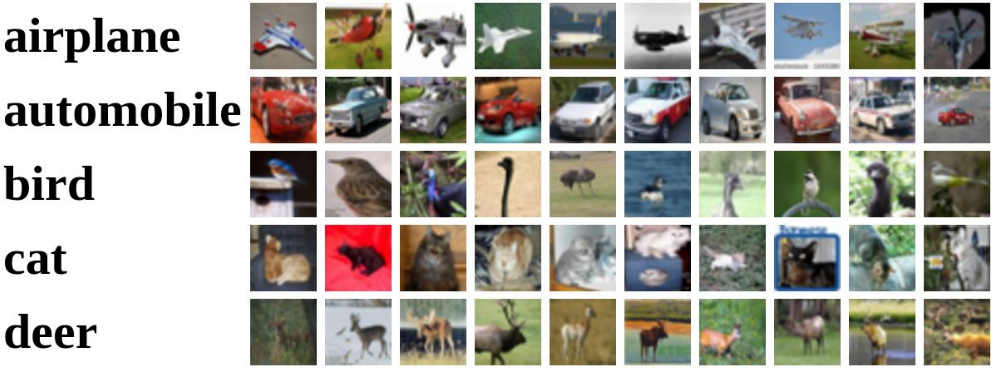
?

John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986

Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels

Example training set



Example dataset: MNIST

10 classes: Digits 0 to 9
28x28 grayscale images
50k training images
10k test images

Example dataset: CIFAR10

10 classes

50k training images (5k per class)

10k testing images (1k per class)

32x32 RGB images

We will use this dataset for homework assignments

Example dataset: CIFAR100

100 classes
50k training images (500 per class)
10k testing images (100 per class)
32x32 RGB images

20 superclasses with 5 classes each:

Aquatic mammals: beaver, dolphin, otter, seal, whale
Trees: Maple, oak, palm, pine, willow

Example dataset: ImageNet (ILSVRC challenge)

ILSVRC = ImageNet Large-Scale Visual Recognition Challenge

1000 classes

~1.3M training images (~1.3K per class)
50K validation images (50 per class)
100K test images (100 per class)

Performance metric: **Top 5 accuracy**
Algorithm predicts 5 labels for each image; one of them needs to be right

Example dataset: MIT Places

365 classes of different scene types

~8M training images
18.25K val images (50 per class)
328.5K test images (900 per class)

Images have variable size, often resize to **256x256** for training

Example dataset: Omniglot

1623 categories: characters from 50 different alphabets

20 images per category

Meant to test **few shot learning**

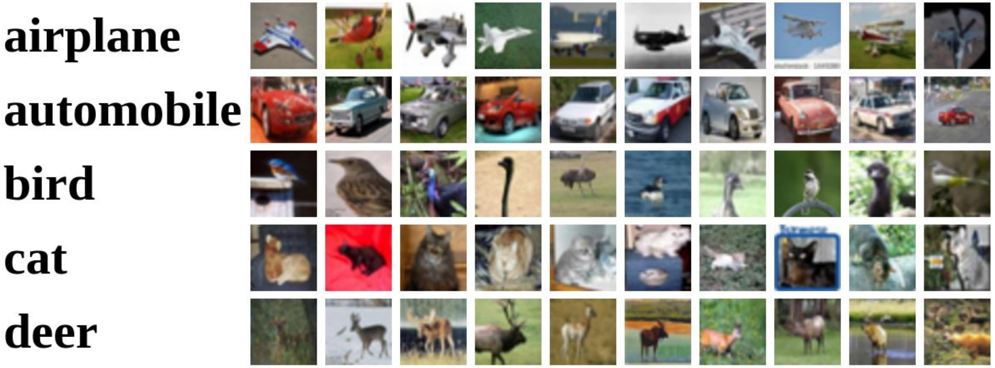
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning algorithms to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Example training set



Nearest Neighbor Classifier

First classifier: **Nearest Neighbor**

```
def train(images, labels):
  # Machine learning!
  return model
```

→ Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

→ Predict the label of the most similar training image

First classifier: **Nearest Neighbor**

deer bird plane cat car

Training data with labels

?

query data

Distance Metric $\Big($  ,  $\Big) \rightarrow \mathbb{R}$

What is a good distance metric?

Distance Metric to compare images

L1 distance: $$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

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

add → 456

Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor classifier

Memorize training data

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor classifier

For each test image:
  Find closest train image
  Predict label of nearest image

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor classifier

Q: With N examples, how fast are training and prediction?

Ans: Train O(1),
predict O(N)

This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

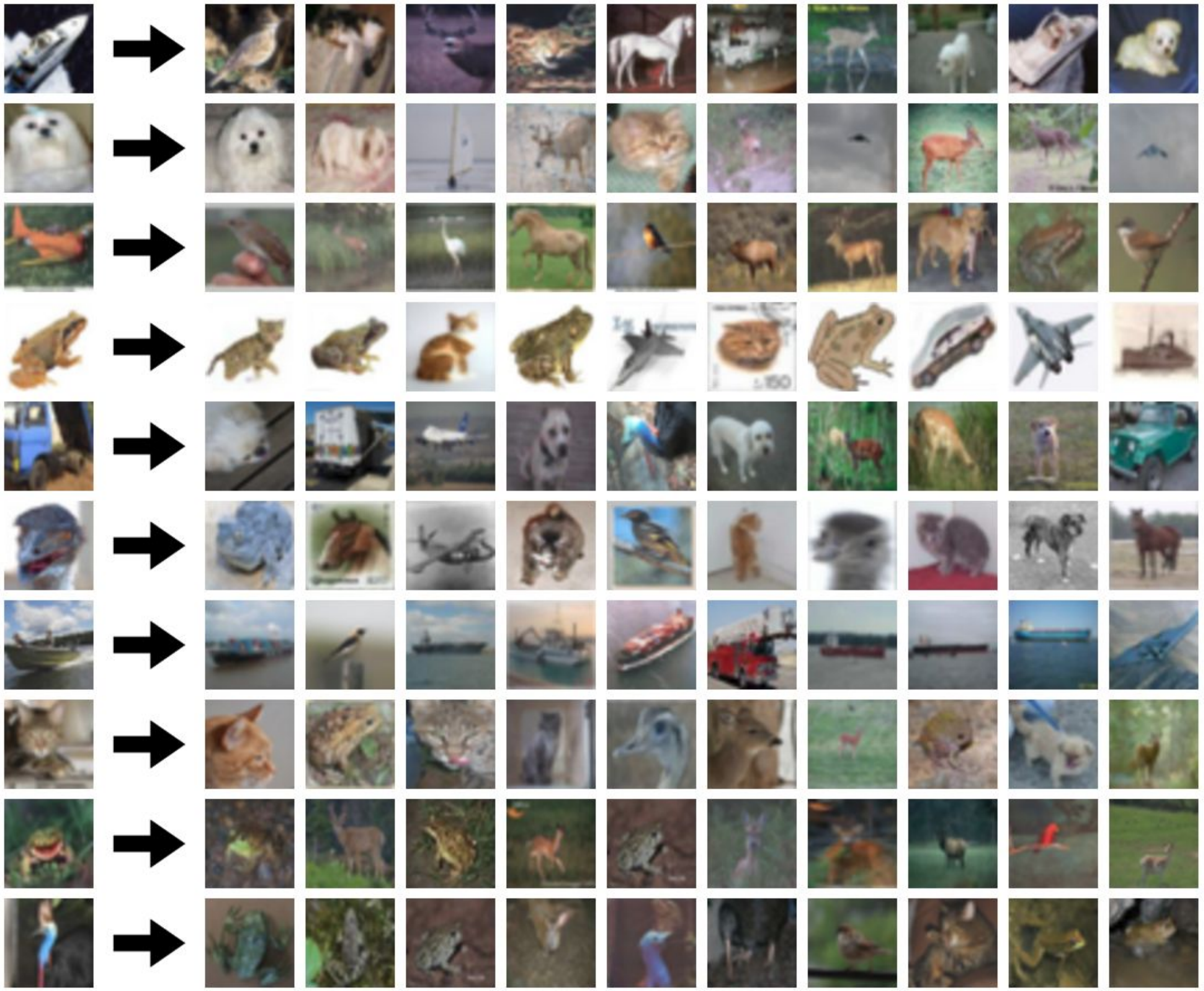
Nearest Neighbor classifier

Many methods exist for fast / approximate nearest neighbor (beyond the scope of this course!)

A good implementation:
https://github.com/facebookresearch/faiss

Johnson et al, “Billion-scale similarity search with GPUs”, arXiv 2017

Example outputs from a NN classifier on CIFAR:



Ali Farhadi, Sarah Pratt

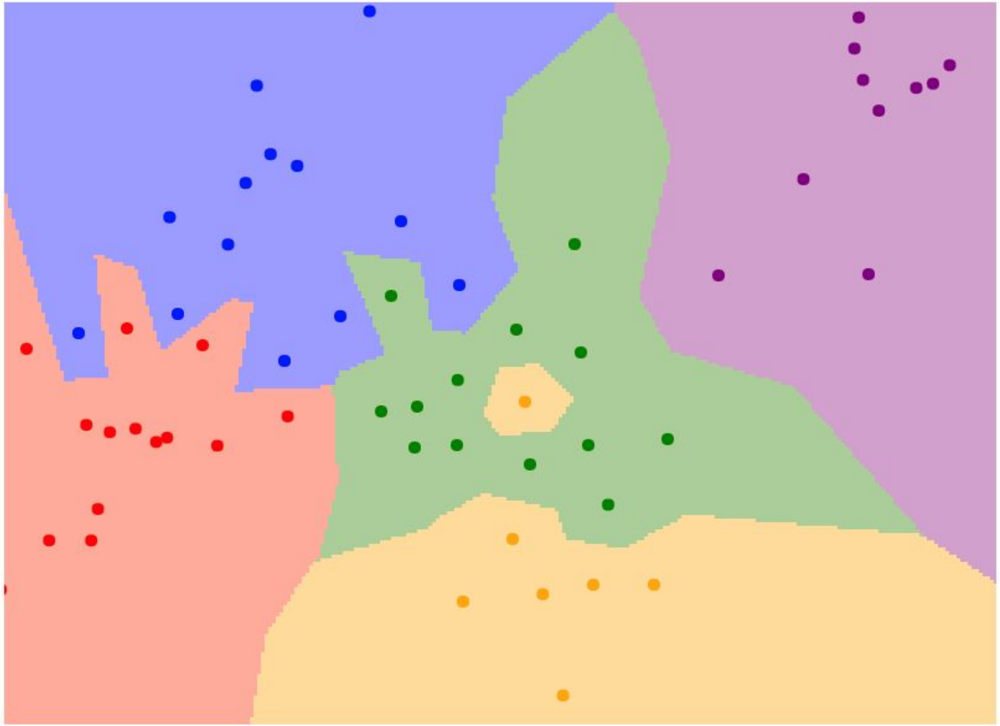
Example outputs from a NN classifier on CIFAR:

Ali Farhadi, Sarah Pratt

Assume each dot is a training image.

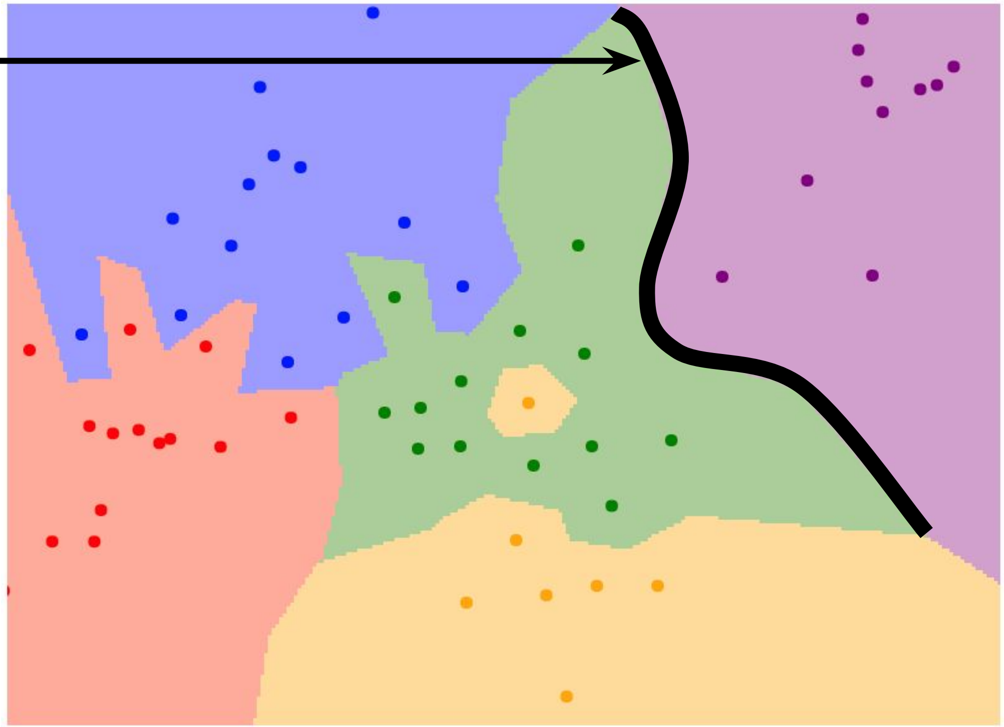
Assume all images are two dimensional.

What does this classifier look like?



1-nearest neighbor

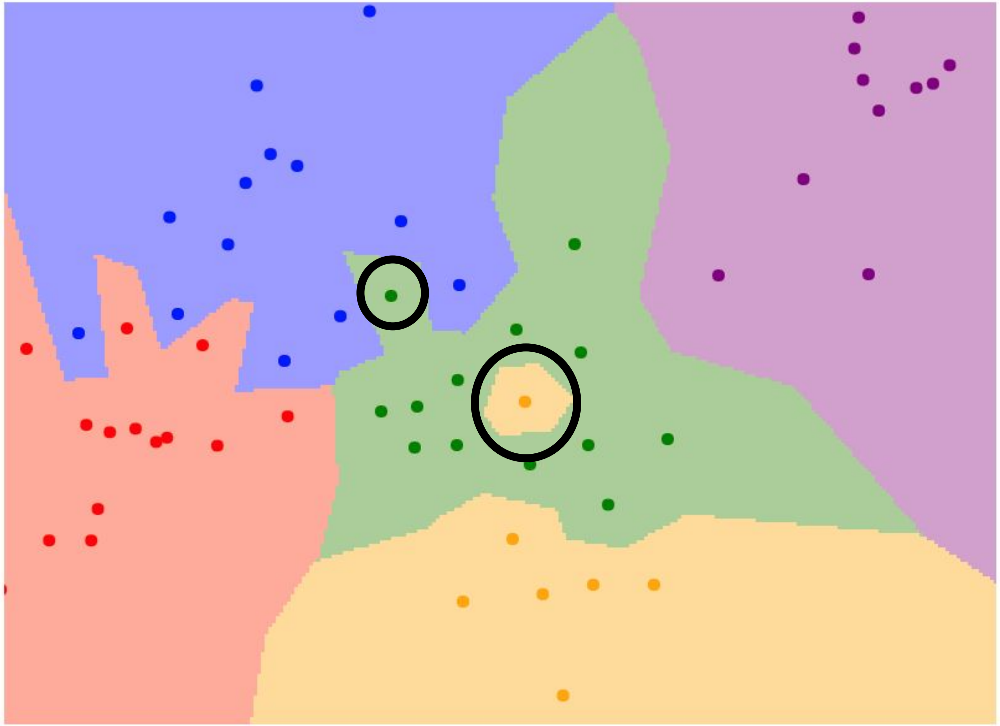
Decision boundary is the boundary between two classification regions



1-nearest neighbor

Yellow point in the middle of green might be mislabeled.

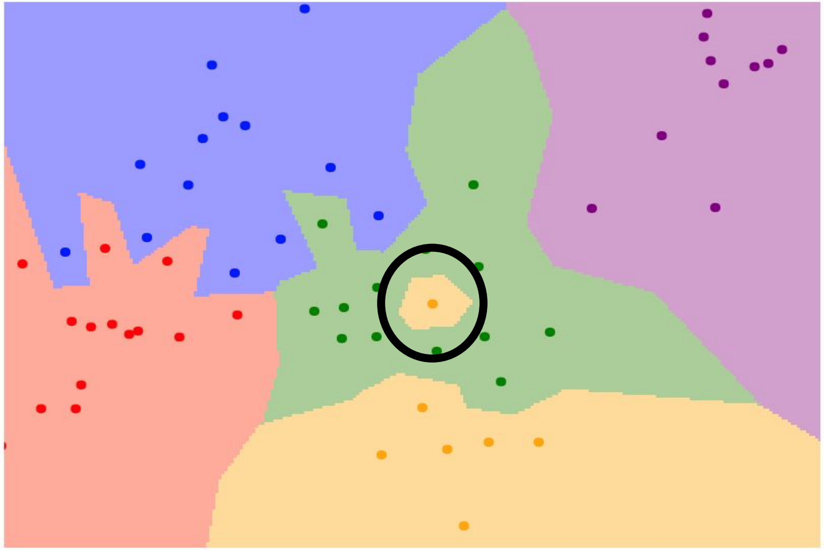
1-NN is not robust to label noise.



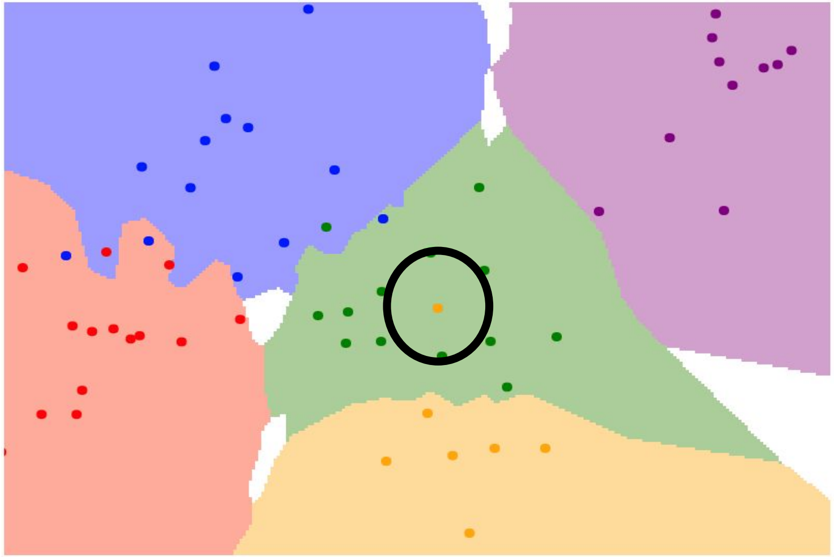
1-nearest neighbor

K-Nearest Neighbors

Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



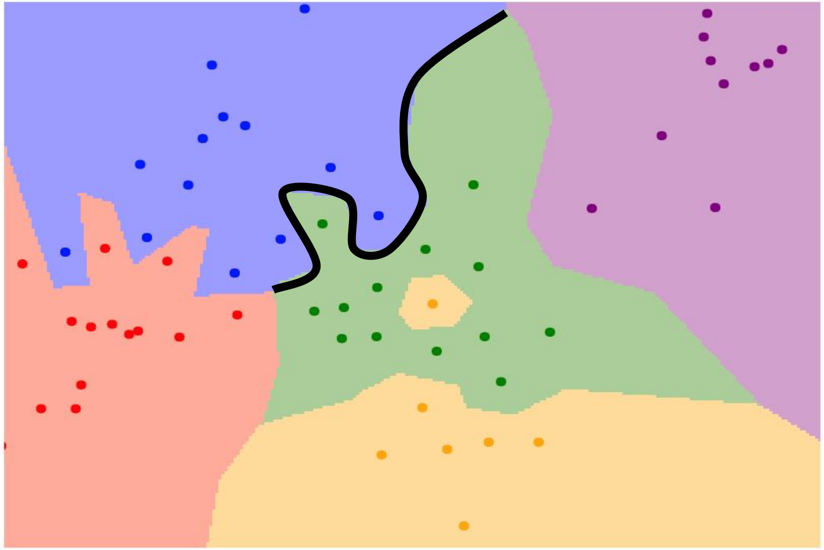
K = 1



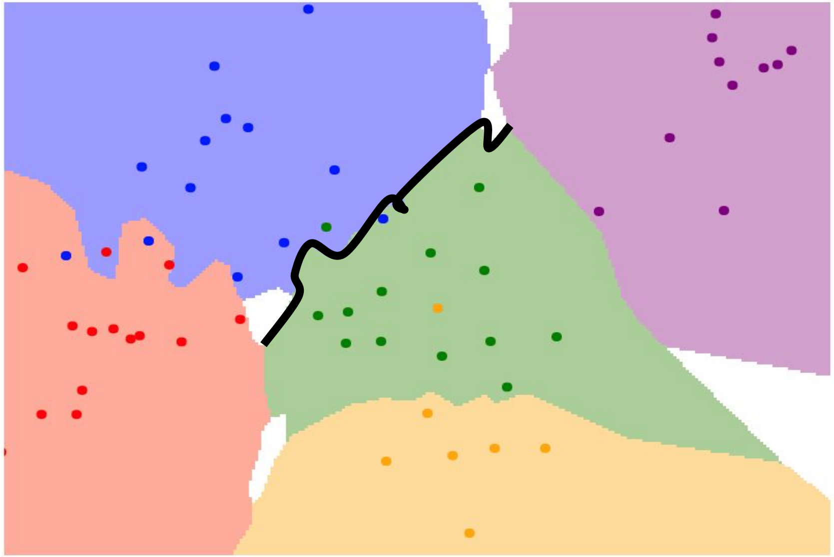
K = 3

K-Nearest Neighbors

Using more neighbors helps smooth out rough decision boundaries



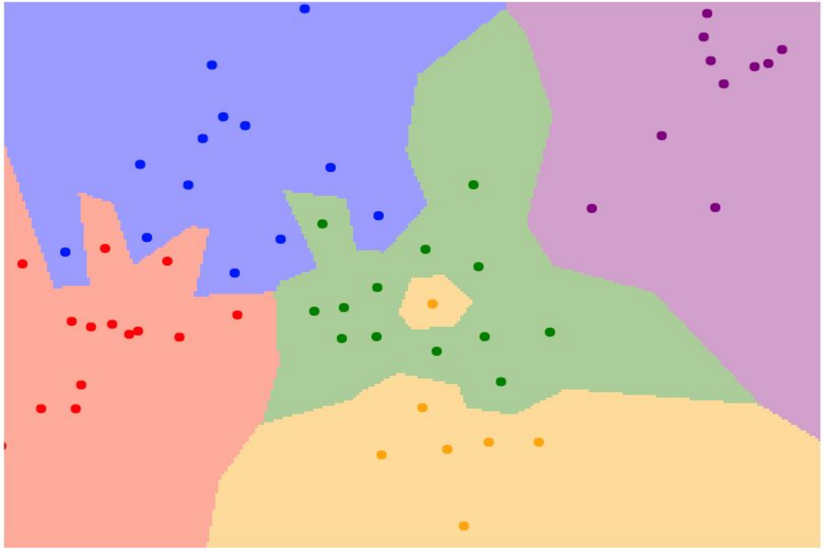
K = 1



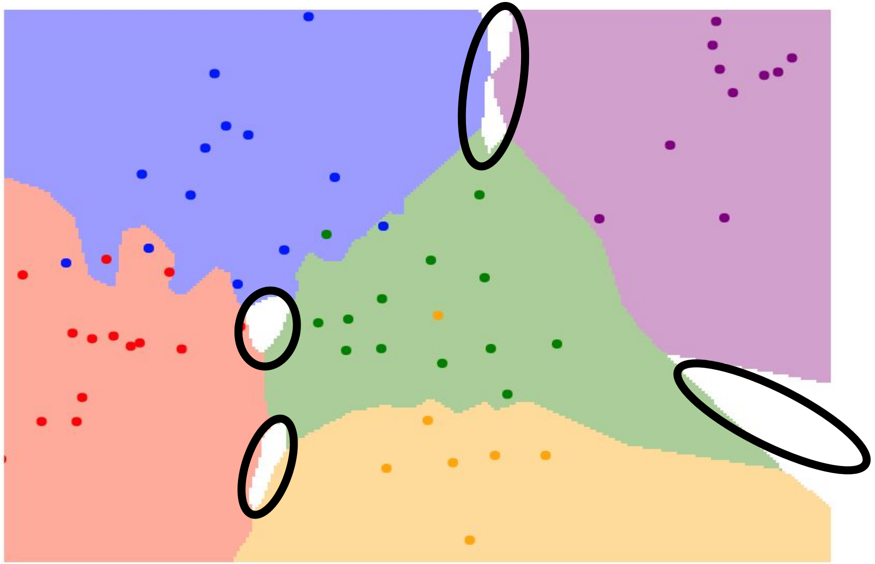
K = 3

K-Nearest Neighbors

Find more labels near uncertain white regions



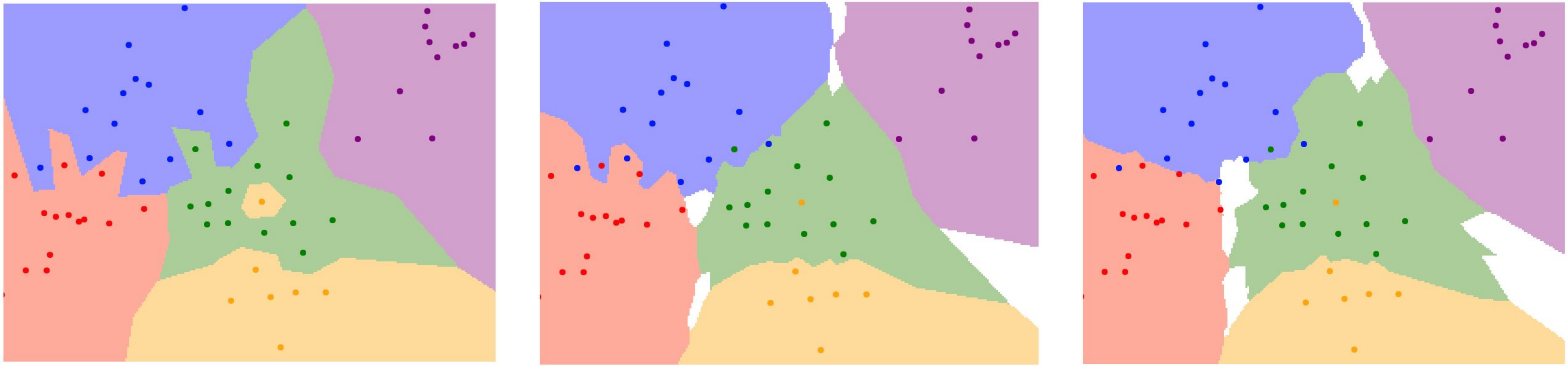
K = 1



K = 3

K-Nearest Neighbors

Larger K smooths boundaries more and leads to more uncertain regions

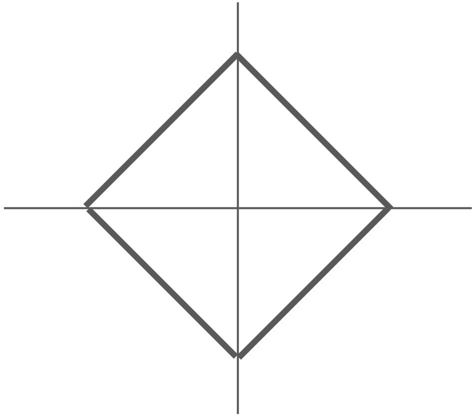


K = 1 K = 3 K = 5

K-Nearest Neighbors: Distance Metric

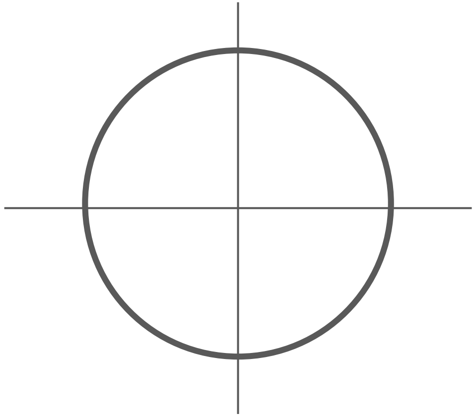
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

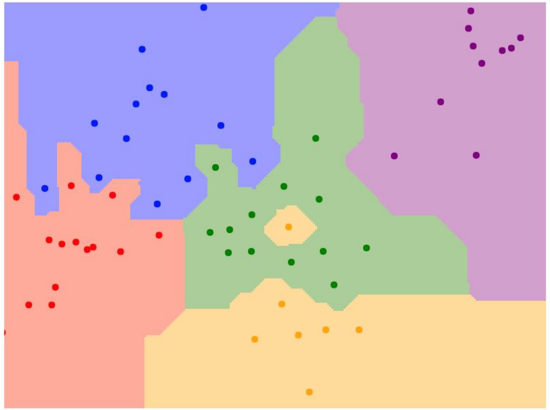
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

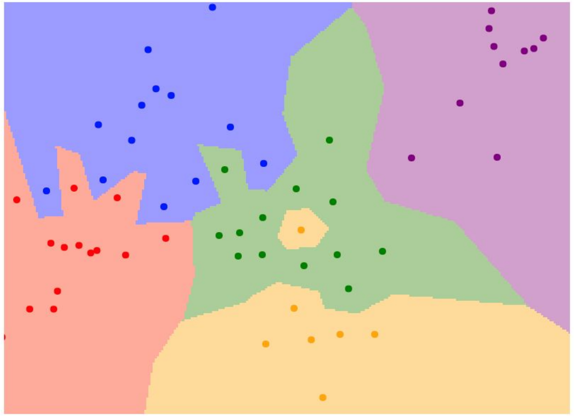
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



K = 1

k-Nearest Neighbor with pixel distance **never used.**

- Distance metrics on pixels are not informative

Original

Occluded

Shifted (1 pixel)

Tinted

(All three images on the right have the same pixel distances to the one on the left)

Hyperparameters

What is the best value of **k** to use?
What is the best **distance** to use?

These are **hyperparameters**: choices about the algorithms themselves.

Very problem/dataset-dependent.
Must try them all out and see what works best.

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**

train

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**

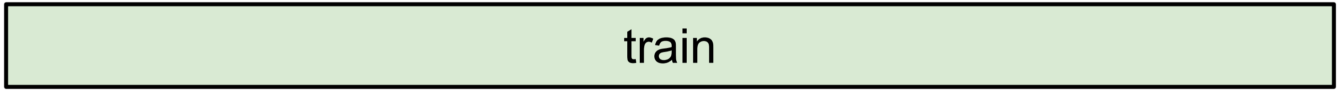
BAD: K = 1 always works perfectly on training data

train

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**

BAD: K = 1 always works perfectly on training data



Idea #2: choose hyperparameters that work best on **test** data

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**

BAD: K = 1 always works perfectly on training data

train

Idea #2: choose hyperparameters that work best on **test** data

BAD: No idea how algorithm will perform on new data

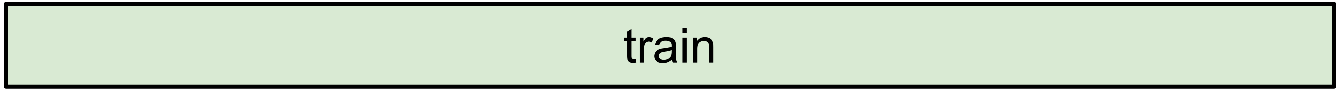
train | test

Never do this!

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the **training data**

BAD: K = 1 always works perfectly on training data



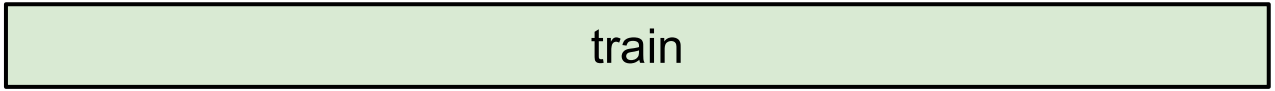
Idea #2: choose hyperparameters that work best on **test** data

BAD: No idea how algorithm will perform on new data

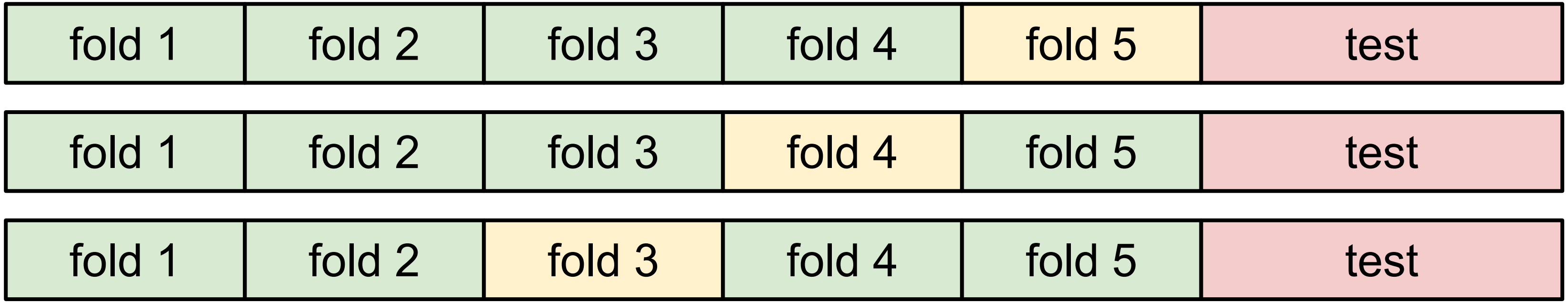
Idea #3: Split data into **train**, **val**; choose hyperparameters on val and evaluate on test

Better!

Setting Hyperparameters



Idea #4: **Cross-Validation**: Split data into **folds**,
try each fold as validation and average the results



Useful for small datasets, but not used too frequently in deep learning

Example Dataset: **CIFAR10**

10 classes
50,000 training images
10,000 testing images

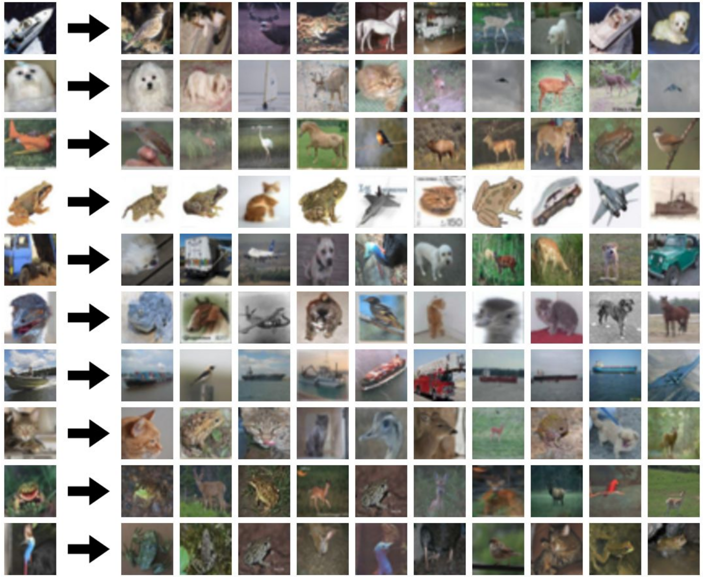
Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Example Dataset: **CIFAR10**

10 classes
50,000 training images
10,000 testing images

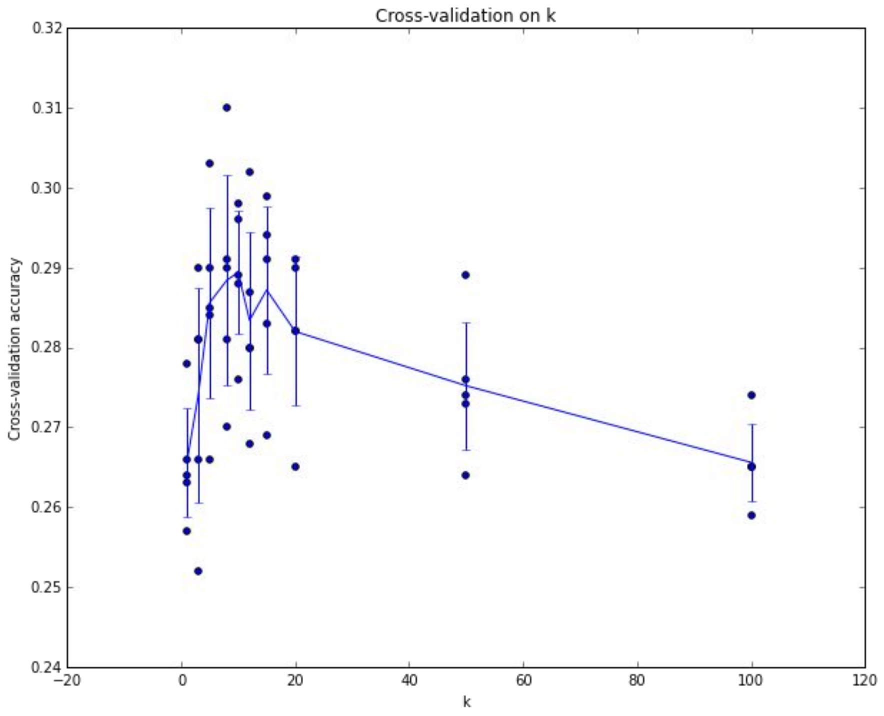
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Test images and nearest neighbors



Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Setting Hyperparameters



Example of
5-fold cross-validation
for the value of **k.**

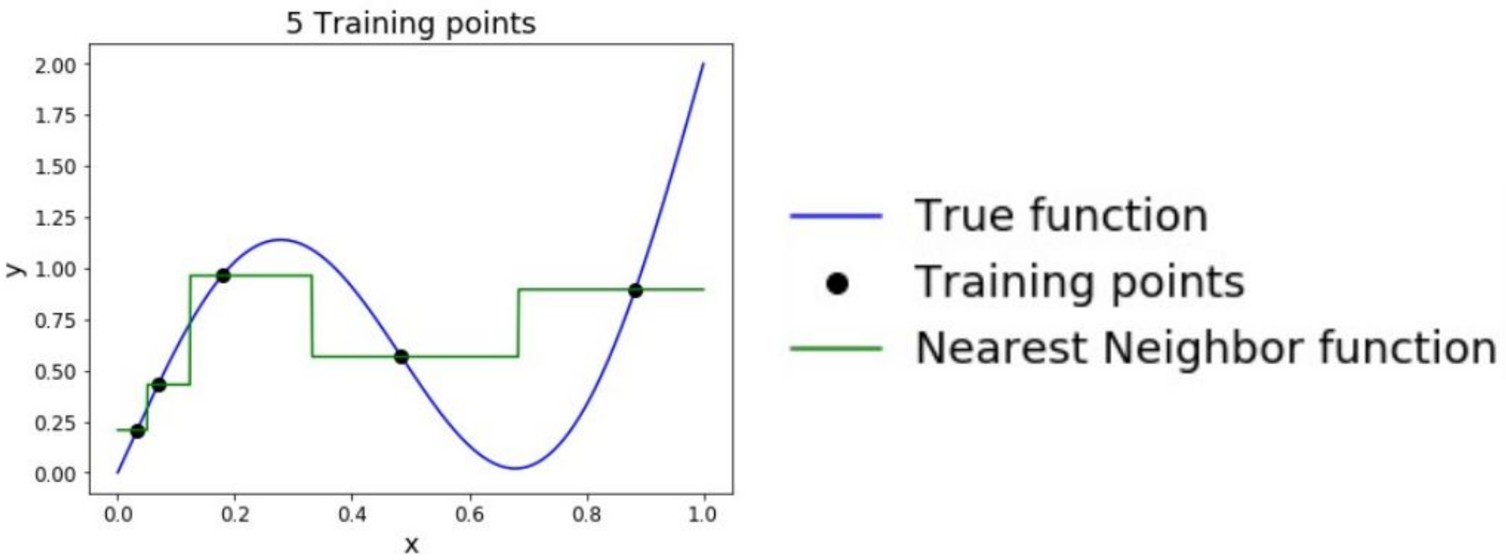
Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

(Seems that k ~= 7 works best
for this data)

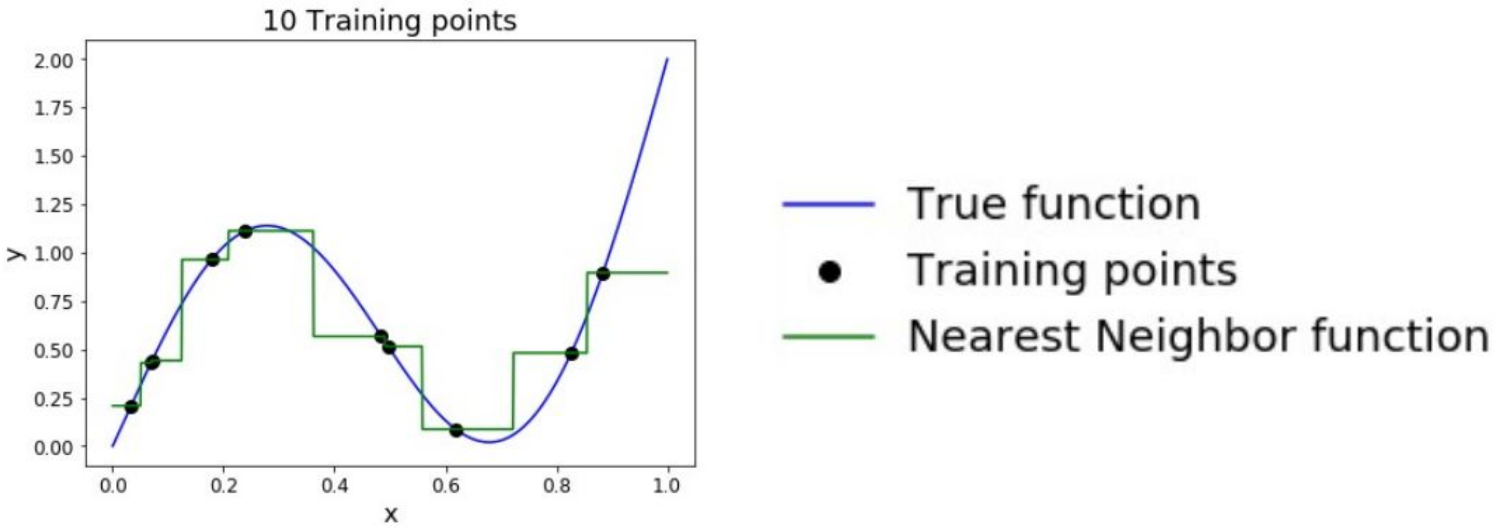
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



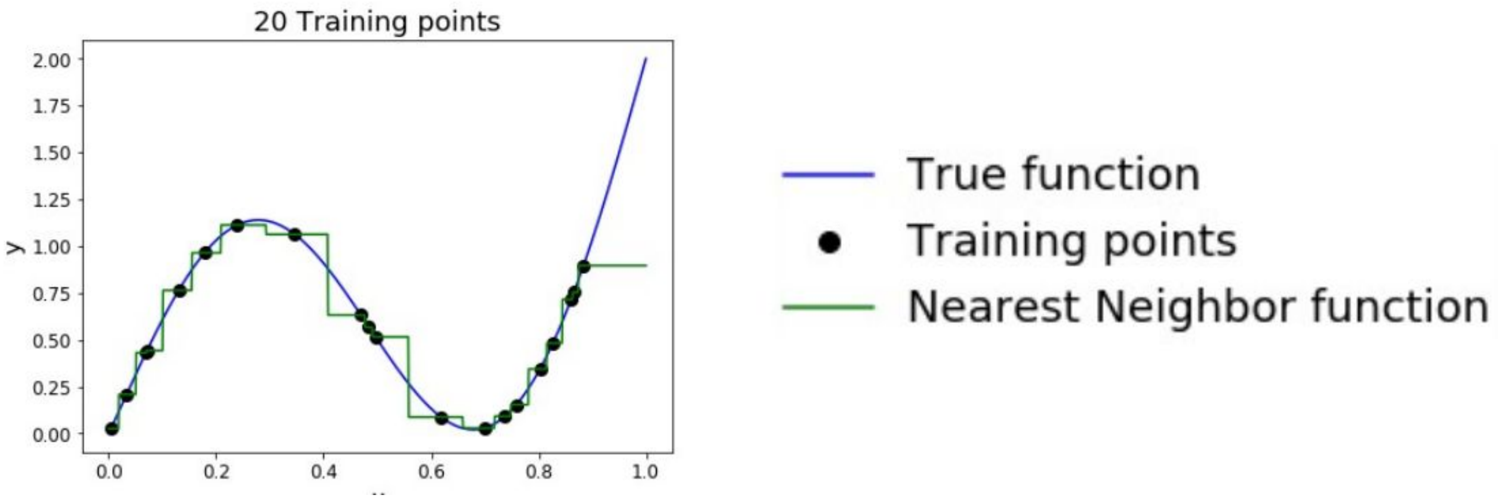
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



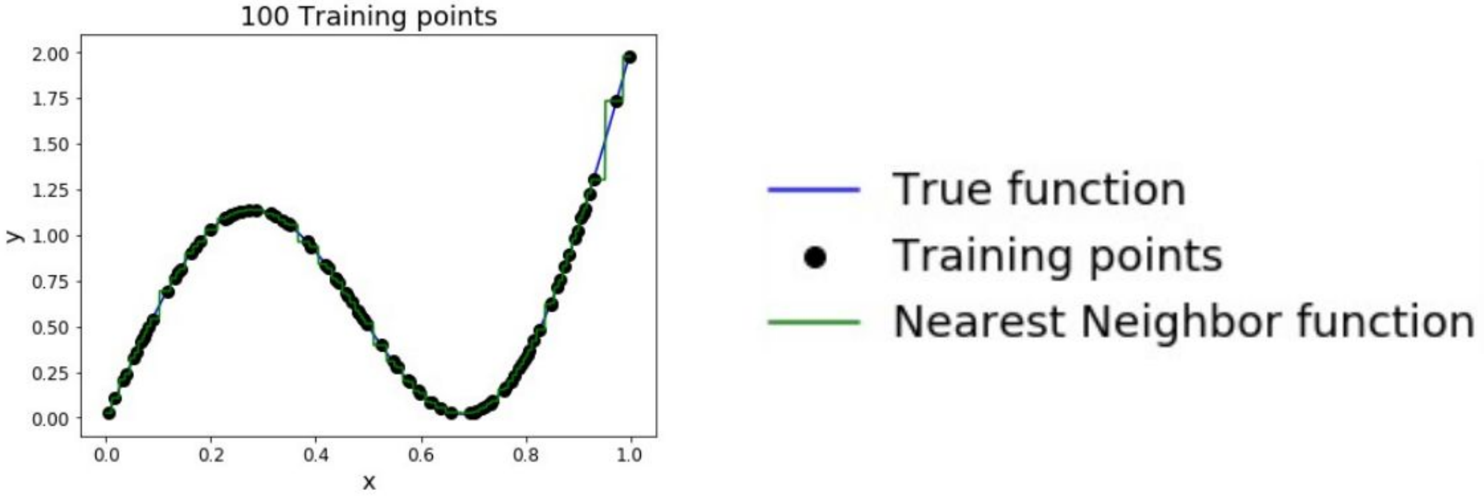
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!

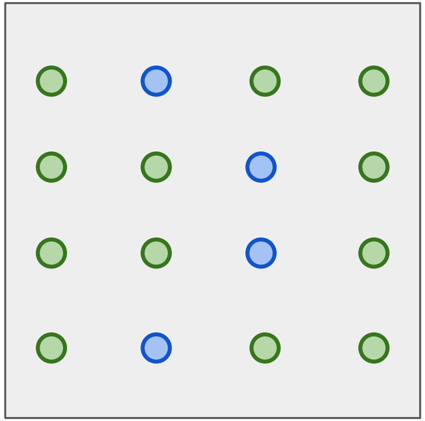


Problem: curse of dimensionality

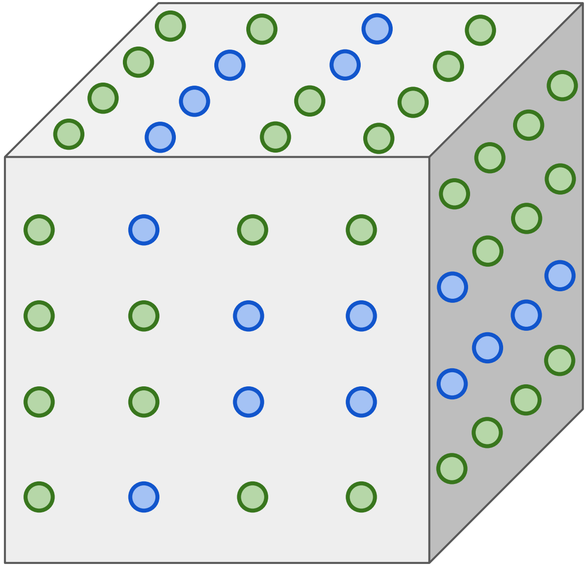
Curse of dimensionality: : For uniform coverage of space, number of training points needed grows exponentially with dimension

Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2



Dimensions = 3
Points = 4^3



Problem: curse of dimensionality

Curse of dimensionality: : For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible 32x32 binary images:

$2^{32x32} = 10^{308}$

Number of elementary particles in the visible universe:
10^{97}

K-Nearest Neighbors: Summary

In **image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**

The **K-Nearest Neighbors** classifier predicts labels based on the K nearest training examples

Distance metric and K are **hyperparameters**

Choose hyperparameters using the **validation set**;

Only run on the test set once at the very end!

Linear Classifier

Parametric Approach

Image

Array of **32x32x3** numbers
(3072 numbers total)

→ $f(\mathbf{x}, \mathbf{W})$ →

10 numbers giving
class scores

↑

W

parameters
or weights

Parametric Approach: Linear Classifier

$$f(x,W) = Wx$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

→ $f(\mathbf{x},\mathbf{W})$ → **10** numbers giving class scores

↑

W

parameters
or weights

Parametric Approach: Linear Classifier

3072x1

$$f(x,W) = Wx$$

10x1   10x3072

Image

Array of **32x32x3** numbers
(3072 numbers total)

→ f(**x**,**W**) → **10** numbers giving class scores

↑

W

parameters
or weights

Parametric Approach: Linear Classifier

3072x1

$$f(x,W) = Wx + b$$

10x1 10x3072 10x1

Image

Array of **32x32x3** numbers
(3072 numbers total)

$\rightarrow$ $f(\mathbf{x},\mathbf{W})$ $\rightarrow$ **10** numbers giving class scores

$\uparrow$

W

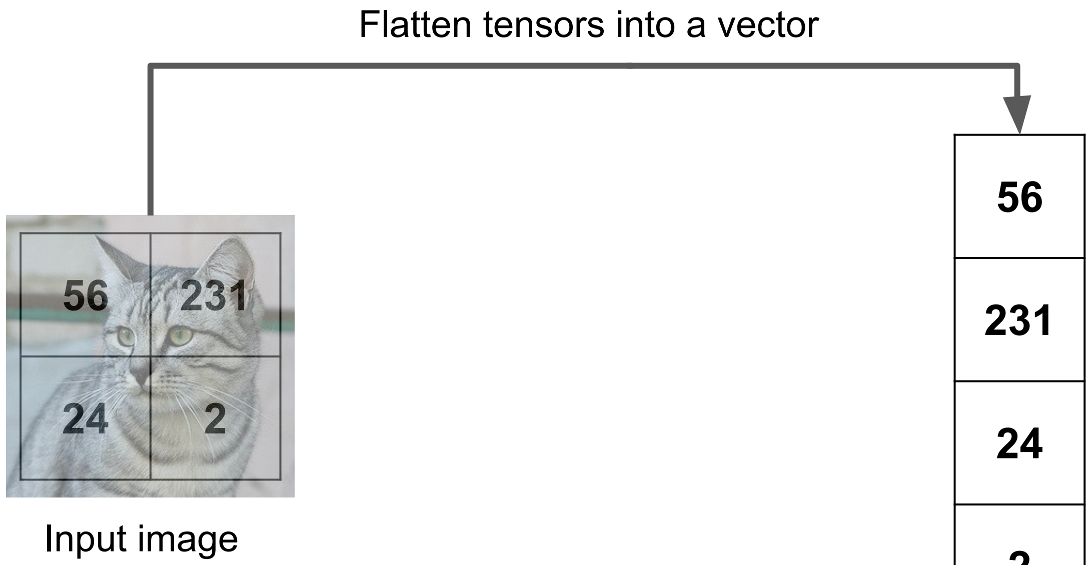
parameters
or weights

Recall CIFAR10

50,000 training images
each image is **32x32x3**

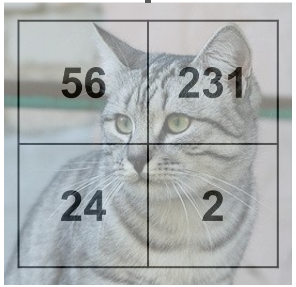
10,000 test images.

Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector

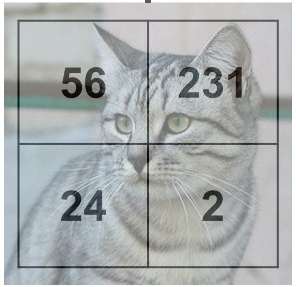


Input image

$$\underbrace{\begin{bmatrix}0.2 & -0.5 & 0.1 & 2.0\\ 1.5 & 1.3 & 2.1 & 0.0\\ 0 & 0.25 & 0.2 & -0.3\end{bmatrix}}_{W}\begin{bmatrix}56\\ 231\\ 24\\ 2\end{bmatrix}+\underbrace{\begin{bmatrix}1.1\\ 3.2\\ -1.2\end{bmatrix}}_{b}=\begin{bmatrix}-96.8\\ 437.9\\ 61.95\end{bmatrix}\begin{matrix}\text{Cat score}\\ \text{Dog score}\\ \text{Ship score}\end{matrix}$$

Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector



Input image

(2,2)

W (3,4)

0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0	0.25	0.2	-0.3

(4,)

56
231
24
2

+ **b** (3,)

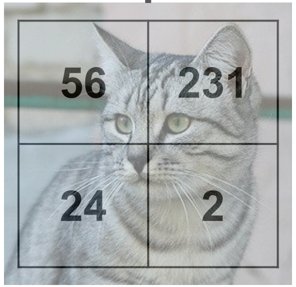
1.1
3.2
-1.2

= (3,)

-96.8	Cat score
437.9	Dog score
61.95	Ship score

Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector



Input image

W

0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0	0.25	0.2	-0.3

56
231
24
2

+

b

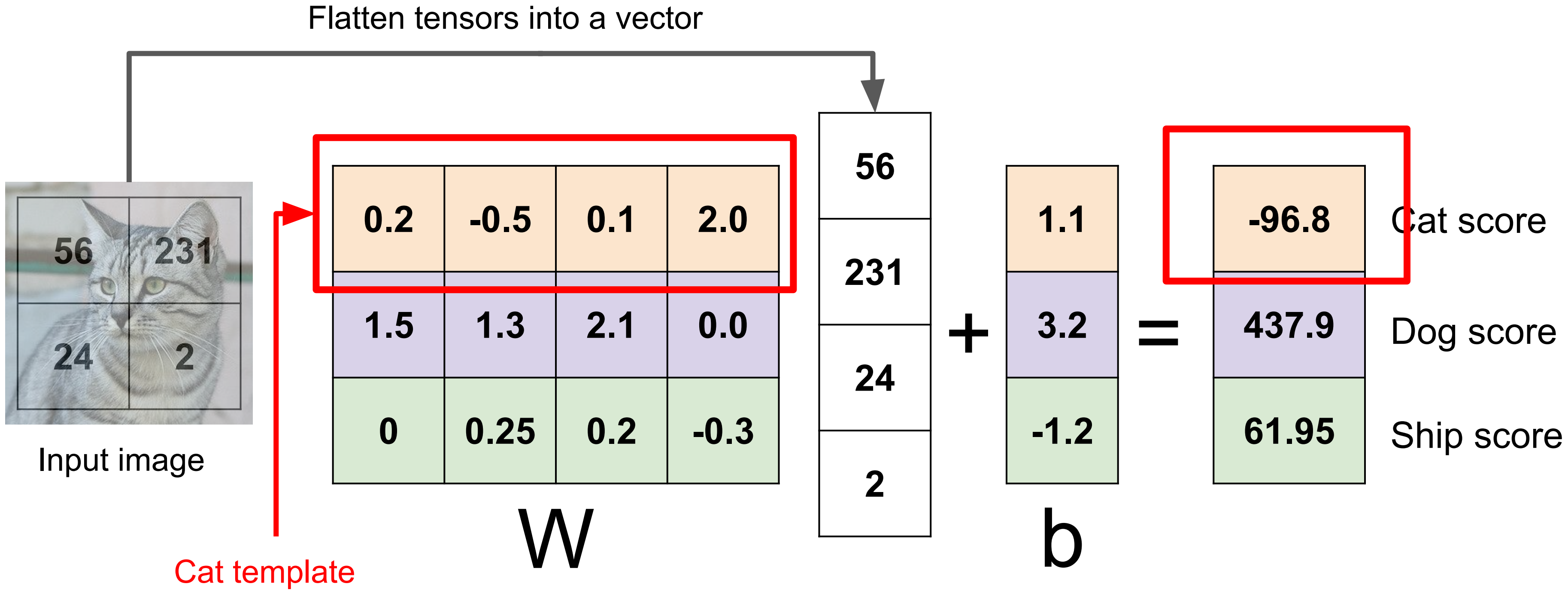
1.1
3.2
-1.2

=

Likelihood of being a cat

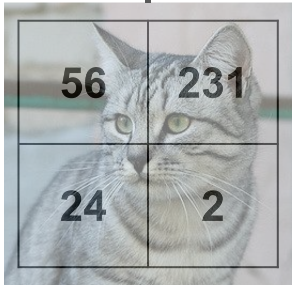
-96.8	Cat score
437.9	Dog score
61.95	Ship score

Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Algebraic viewpoint: Bias trick to simply computation

Flatten tensors into a vector



Input image

0.2	-0.5	0.1	2.0	1.1
1.5	1.3	2.1	0.0	3.2
0	0.25	0.2	-0.3	-1.2

W;b

(3,5)

56
231
24
2
1

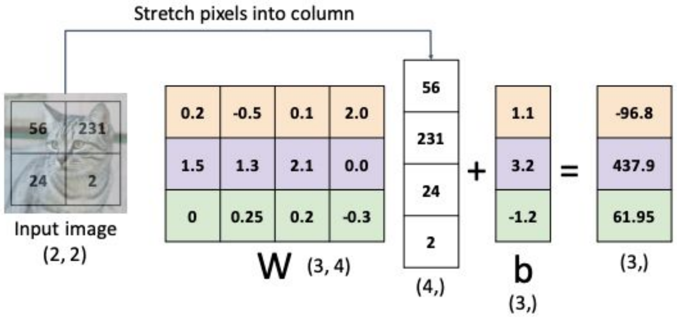
(5,)

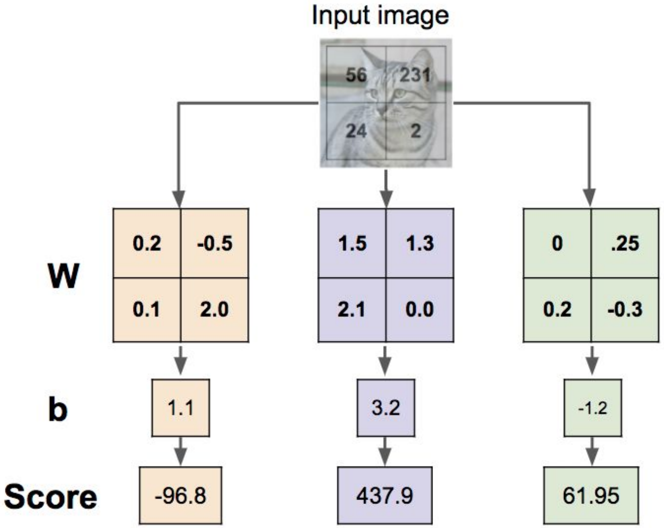
=

-96.8	Cat score
437.9	Dog score
61.95	Ship score

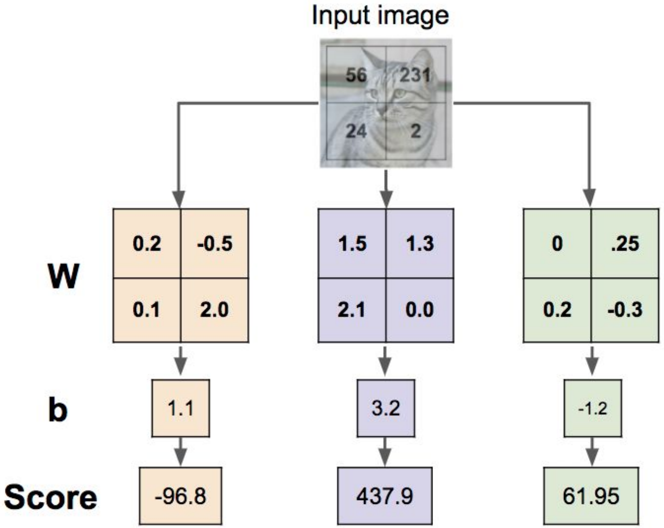
Visual Viewpoint: learning templates

Algebraic viewpoint:

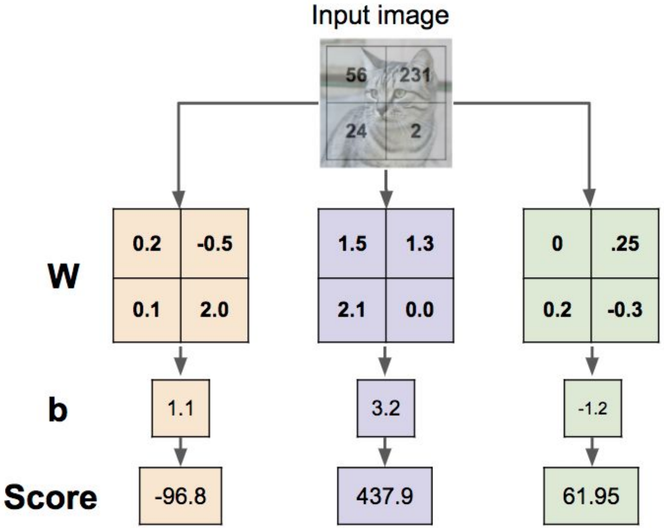




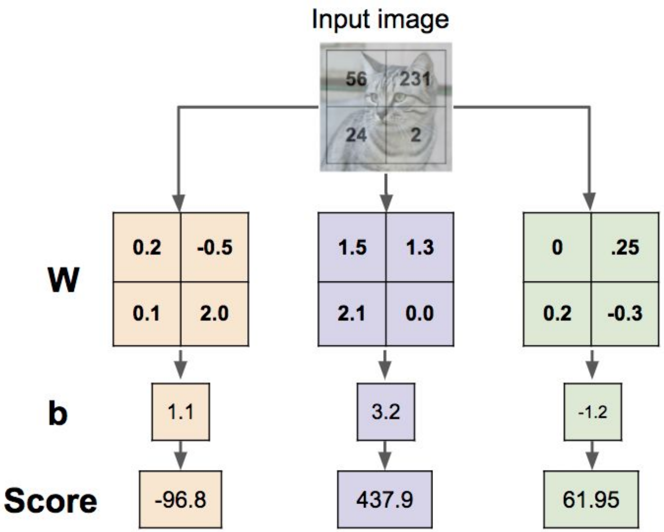
Visual Viewpoint: learning templates

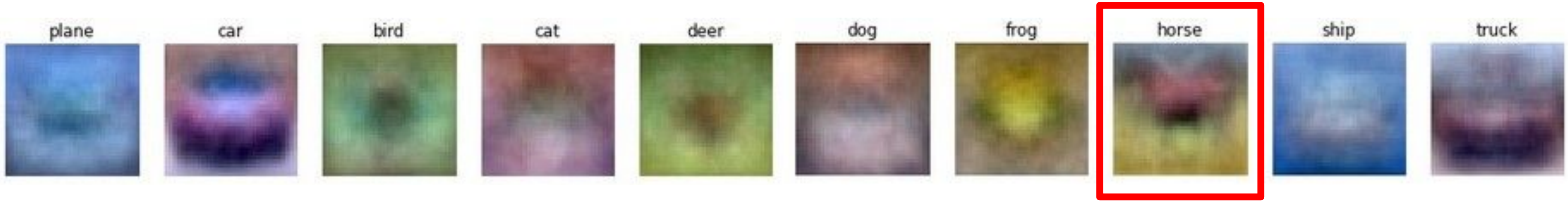


Visual Viewpoint: learning templates

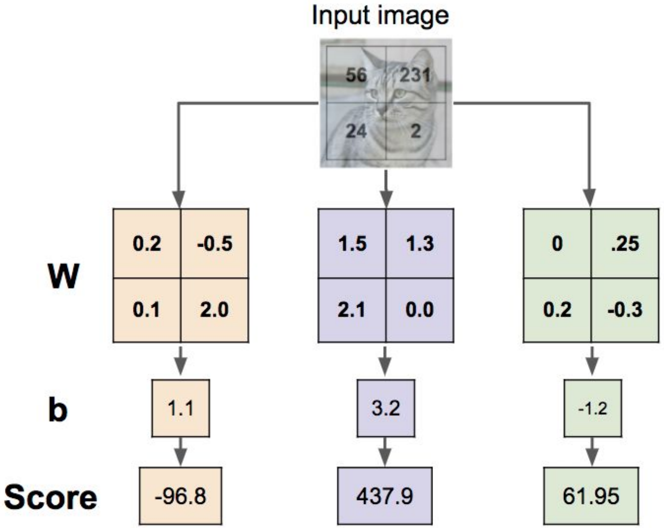


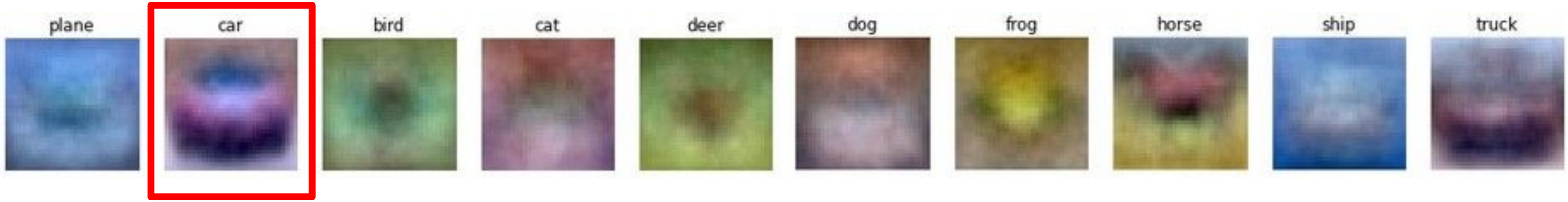
Visual Viewpoint: learning templates



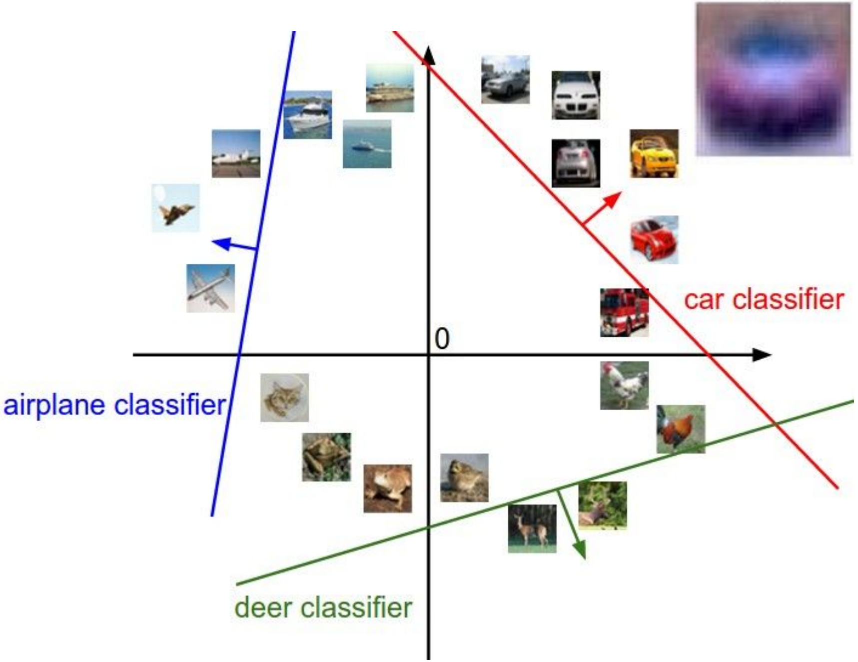


Visual Viewpoint: learning templates





Geometric Viewpoint: linear decision boundaries

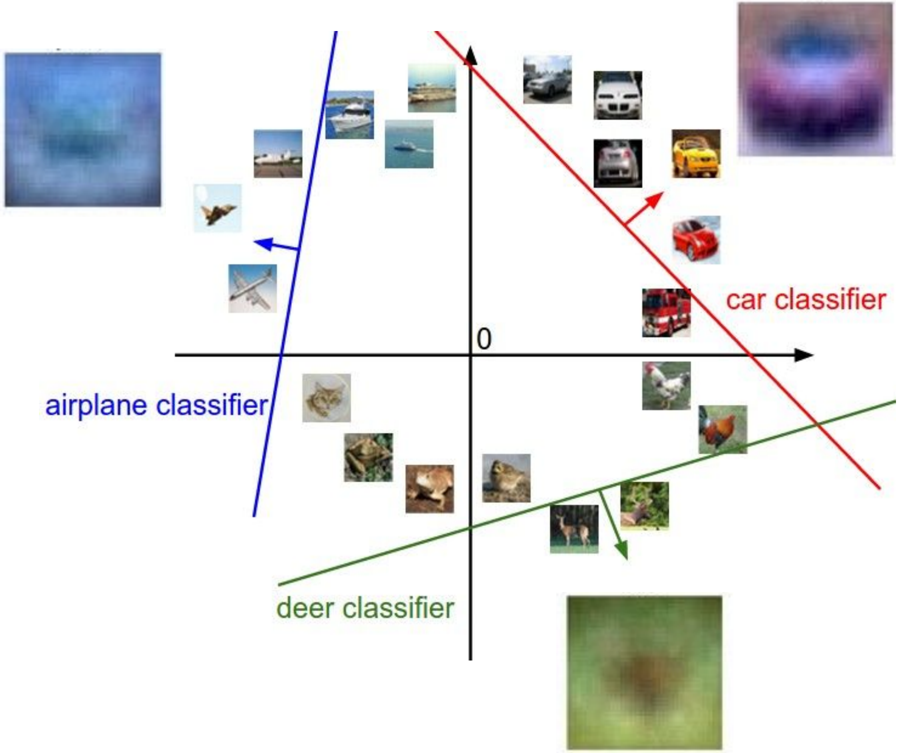


$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

Plot created using Wolfram Cloud

Geometric Viewpoint: linear decision boundaries

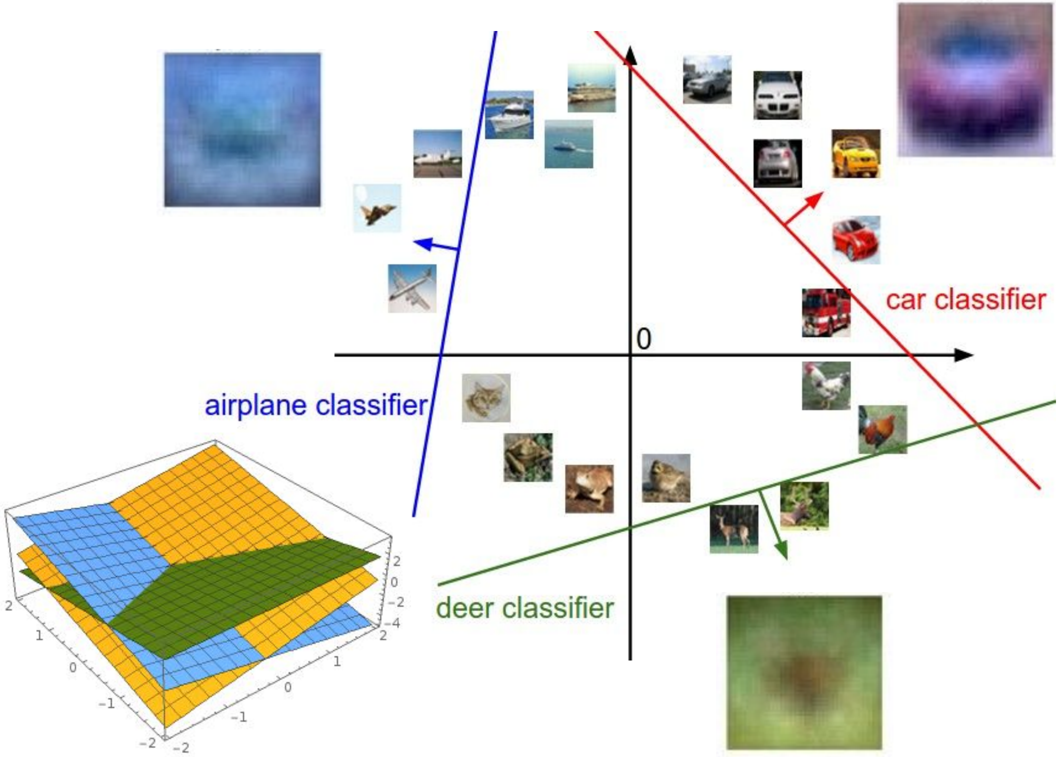


$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

Plot created using Wolfram Cloud

Geometric Viewpoint: linear decision boundaries



$$f(x,W) = Wx + b$$

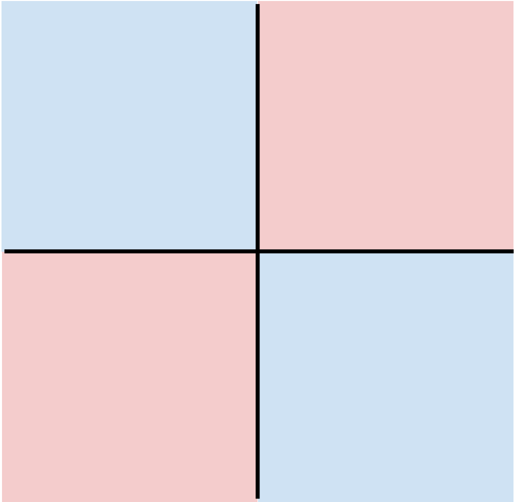
Array of **32x32x3** numbers
(3072 numbers total)

Plot created using Wolfram Cloud

Hard cases for a linear classifier

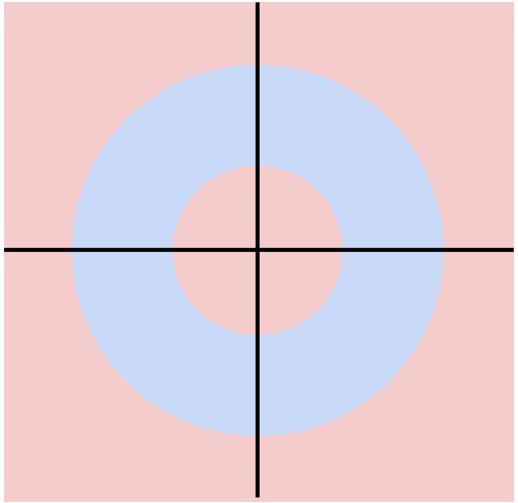
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



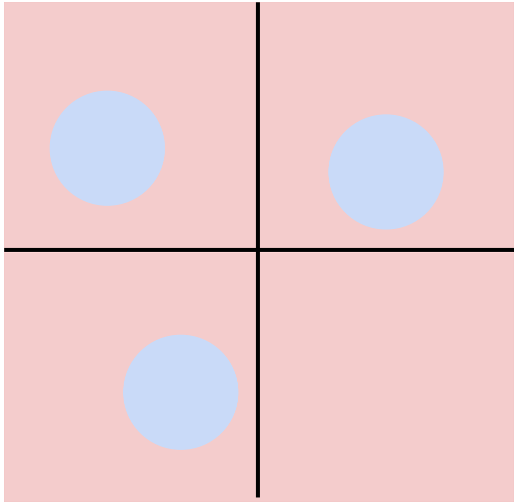
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

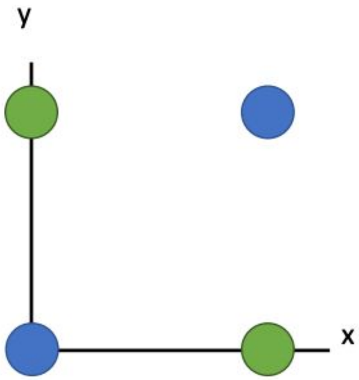
Class 2:
Everything else

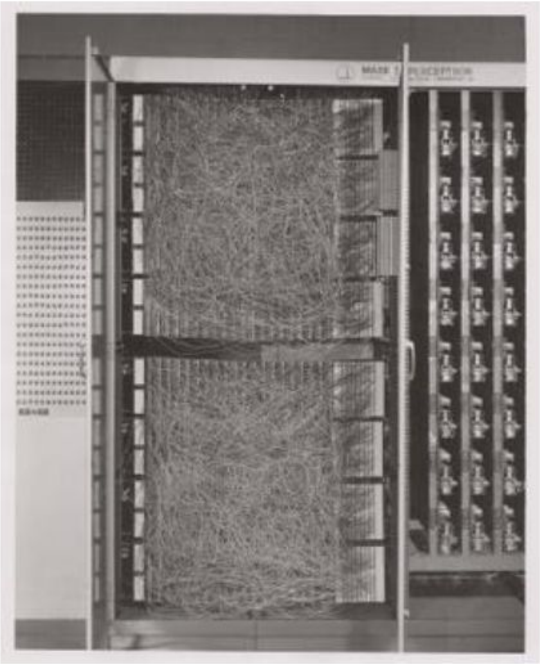


Recall the Minsky report 1969 from last lecture

Unabled to learn the XNOR function

X	Y	F(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

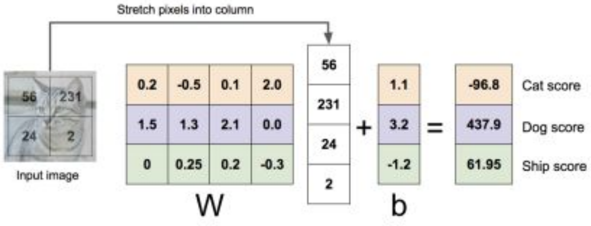




Three viewpoints for interpreting linear classifiers

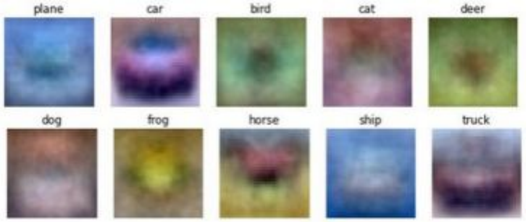
Algebraic Viewpoint

f(x,W) = Wx



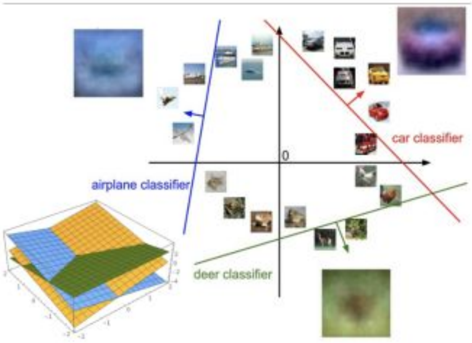
Visual Viewpoint

One template
per class



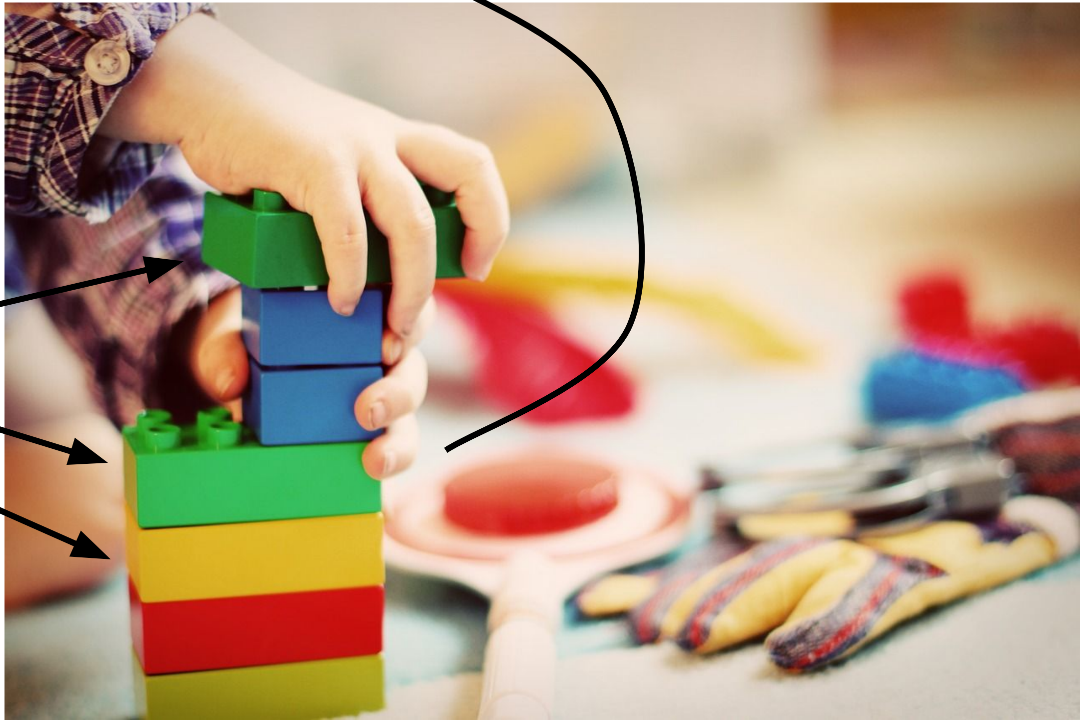
Geometric Viewpoint

Hyperplanes
cutting up space

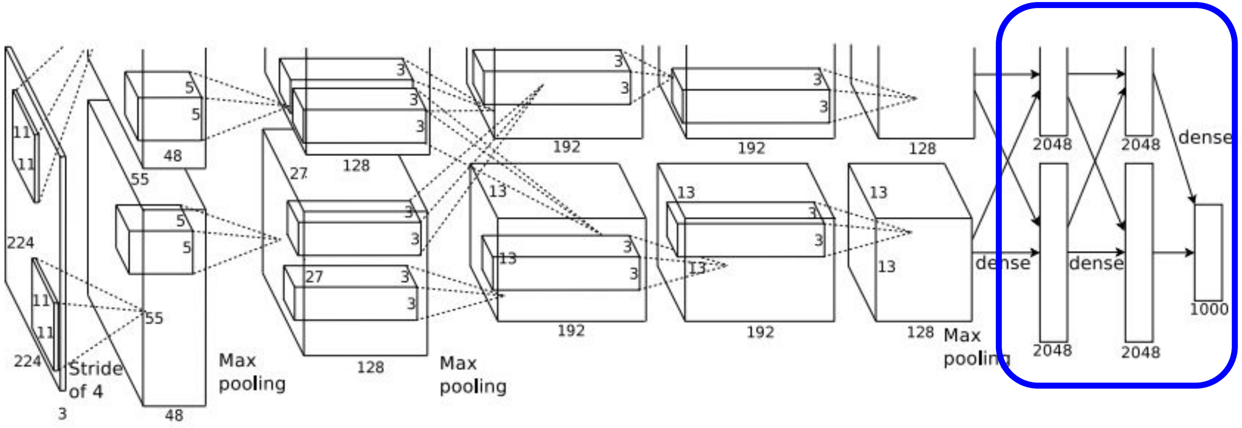


Neural Network

Linear classifiers

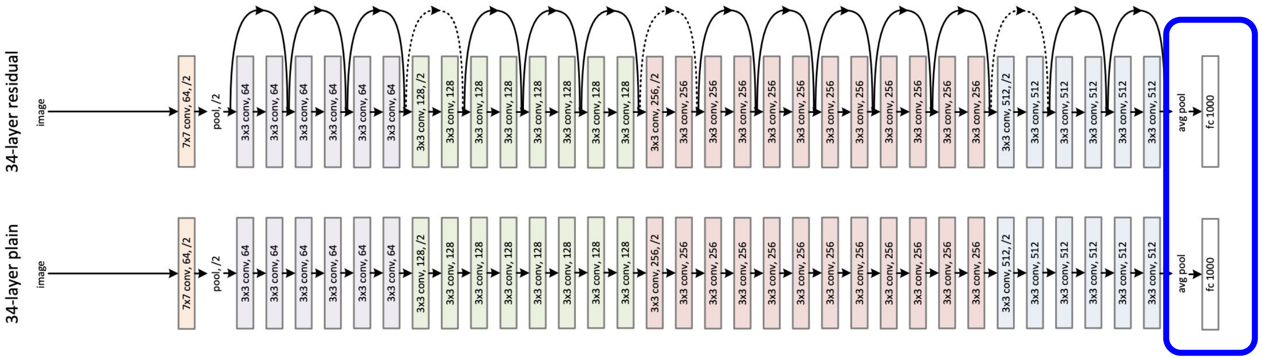


This image is CC0 1.0 public domain



[Krizhevsky et al. 2012]

Linear layers



[He et al. 2015]

Coming up:
- Loss function
- Optimization
- Neural Networks

$$f(x,W) = Wx + b$$

(quantifying what it means to have a "good" W)

(start with random W and find a W that minimizes the loss)

(tweak the functional form of f)