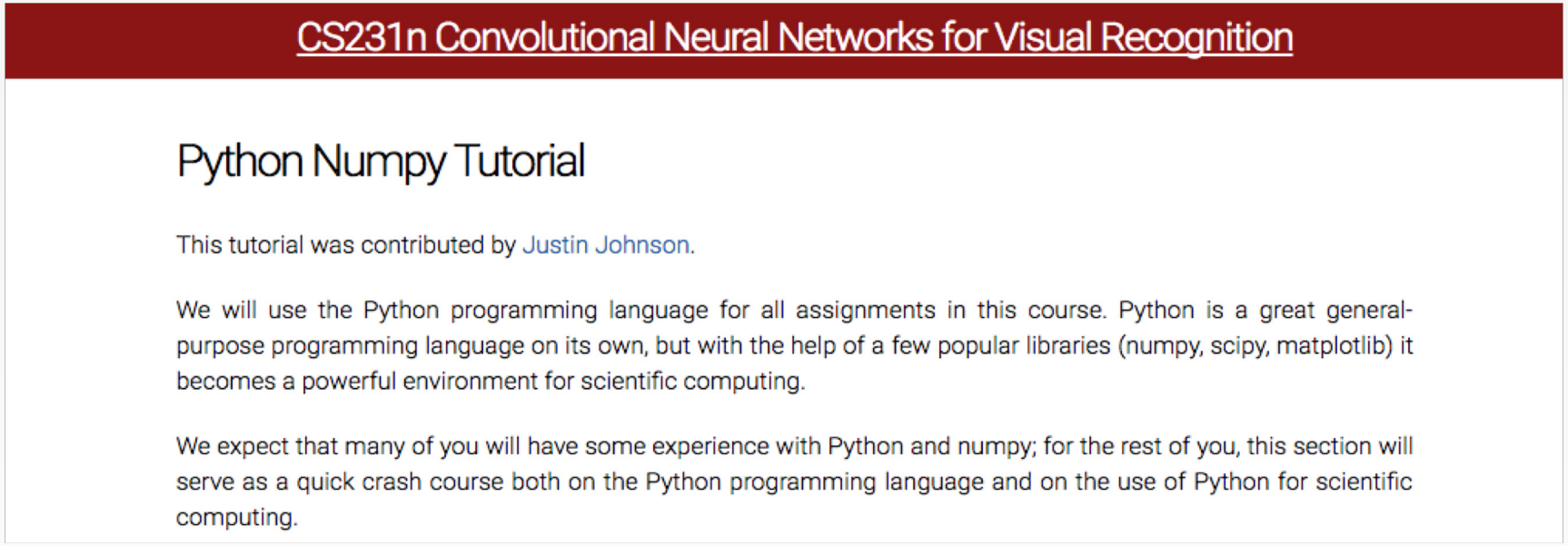
DATA ANALYTICS USING DEEP LEARNING

GT 8803 // FALL 2019 // JOY ARULRAJ

LECTURE #02: IMAGE CLASSIFICATION

CREATING THE NEXT®

PYTHON + NUMPY TUTORIAL



CS231n Convolutional Neural Networks for Visual Recognition

Python Numpy Tutorial

This tutorial was contributed by Justin Johnson.

We will use the Python programming language for all assignments in this course. Python is a great general-purpose programming language on its own, but with the help of a few popular libraries (numpy, scipy, matplotlib) it becomes a powerful environment for scientific computing.

We expect that many of you will have some experience with Python and numpy; for the rest of you, this section will serve as a quick crash course both on the Python programming language and on the use of Python for scientific computing.

http://cs231n.github.io/python-numpy-tutorial/

ASSIGNMENT #0

- Hand in **one page** with following details
  - Digital picture (ideally 2x2 inches of face)
  - Name (last name, first name)
  - Year in School
  - Major Field
  - Final Degree Goal (e.g., B.S., M.S., Ph.D.)
  - Previous Education (degrees, institutions)
  - Previous Courses
  - More details on Gradescope

ASSIGNMENT #0

- The purpose is to help us:
  - know more about your background for tailoring the course, and
  - recognize you in class
- Due on next Aug 26 (Monday)

LAST CLASS

- History of Computer Vision
- Overview of Visual Recognition problems
  - Focus on the Image Classification problem

TODAY'S AGENDA

- Image Classification
- Nearest Neighbor Classifier
- Linear Classifier

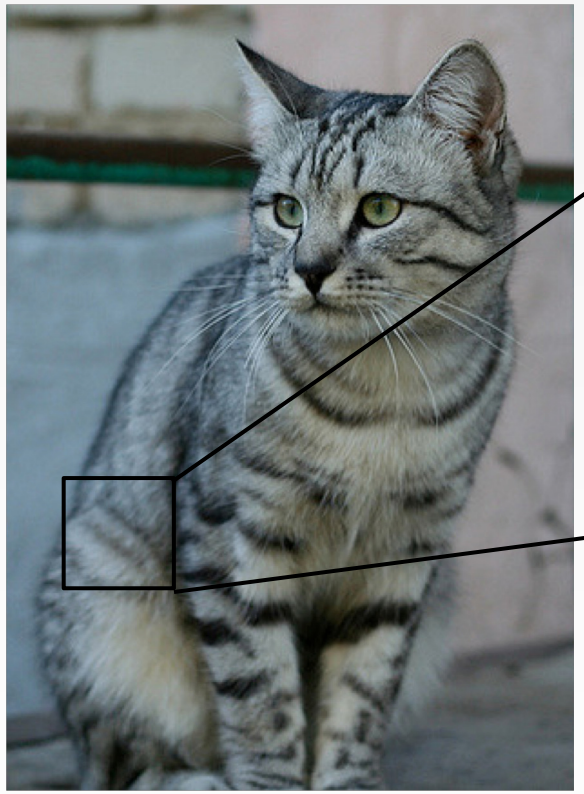
IMAGE CLASSIFICATION

IMAGE CLASSIFICATION: A CORE CV TASK

(assume given set of DISCRETE LABELS)
{dog, cat, truck, plane, ...}

CAT

THE PROBLEM: "SEMANTIC" GAP

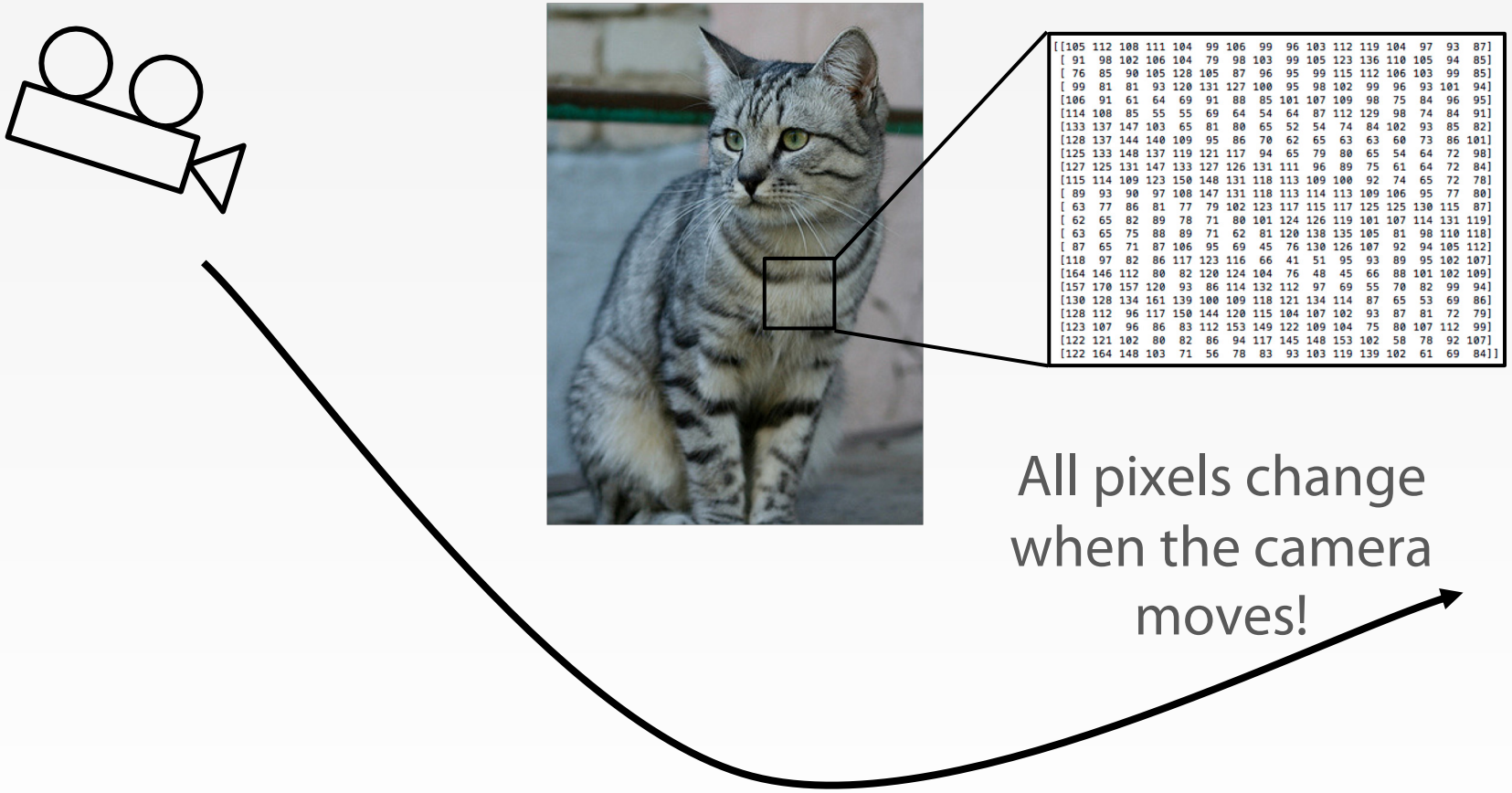


```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  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

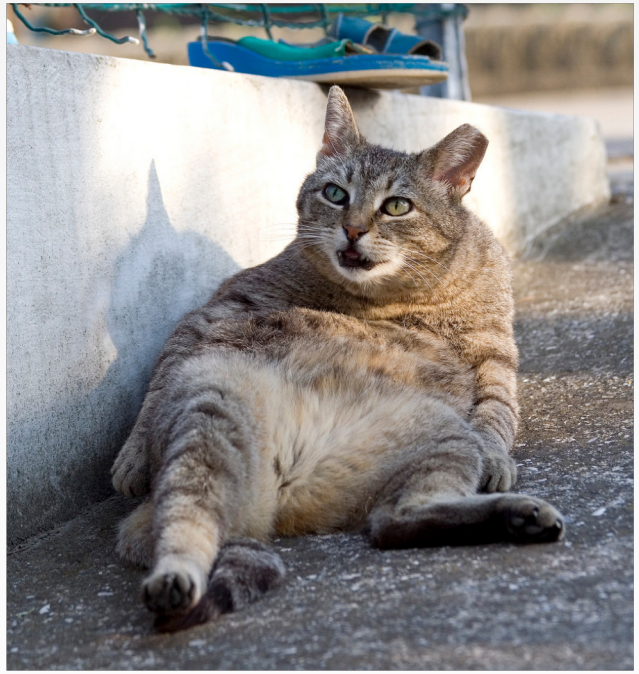
An image is just a big grid of numbers between [0, 255]:
e.g. 800 x 600 x 3 (3 channels RGB)

CHALLENGES: VIEWPOINT VARIATION



CHALLENGES: ILLUMINATION

CHALLENGES: DEFORMATION



CHALLENGES: OCCLUSION

CHALLENGES: BACKGROUND CLUTTER

CHALLENGES: INTRACLASS VARIATION

CHALLENGES: IMAGE CLASSIFICATION

- Hard to appreciate the complexity of this task
  - Because your brains are tuned for dealing with this
  - But, this is a fantastically challenging problem for computer programs
- It is miraculous that a program works at all in practice
  - But, it actually works very close to human accuracy (with certain constraints)

AN IMAGE CLASSIFIER

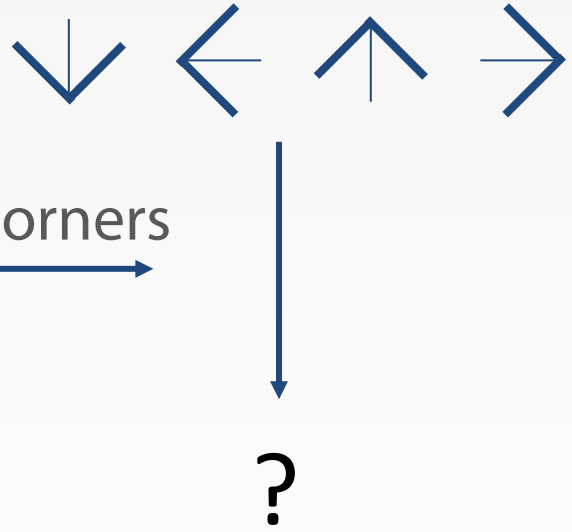
```
def classify_image(image):
  # Some magic here?
  return class_label
```

- Unlike e.g. sorting a list of numbers,
  - **No obvious way** to hard-code the algorithm for recognizing a cat, or other classes

RULE-BASED APPROACH

Find edges

Find corners



CHALLENGES: RULE-BASED APPROACH

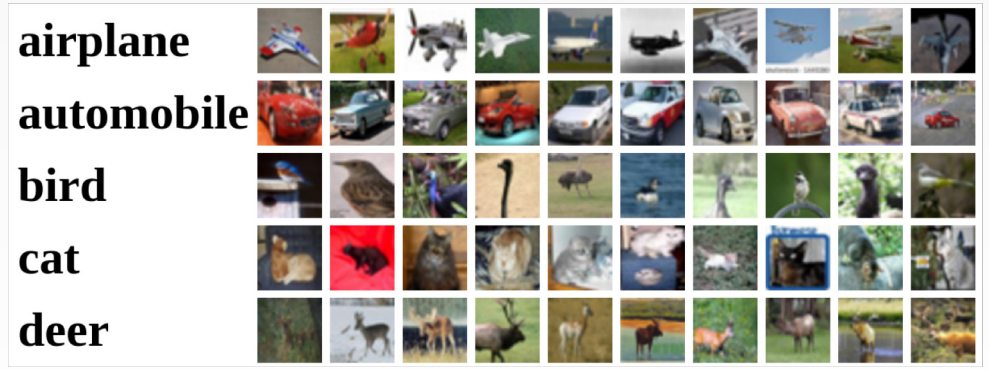
- Challenges
  - Not robust enough to handle different image transformations
  - Does not generalize to other classes (e.g., dogs)
- Need a robust scalable approach

DATA-DRIVEN APPROACH: MACHINE LEARNING

1. Collect a dataset of images and labels
2. Use machine learning 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

NEAREST NEIGHBOR CLASSIFIER

- This class is primarily about neural networks
  - But, this data driven approach is more general
  - We will start with a simpler classifier

FIRST CLASSIFIER: NEAREST NEIGHBOR

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

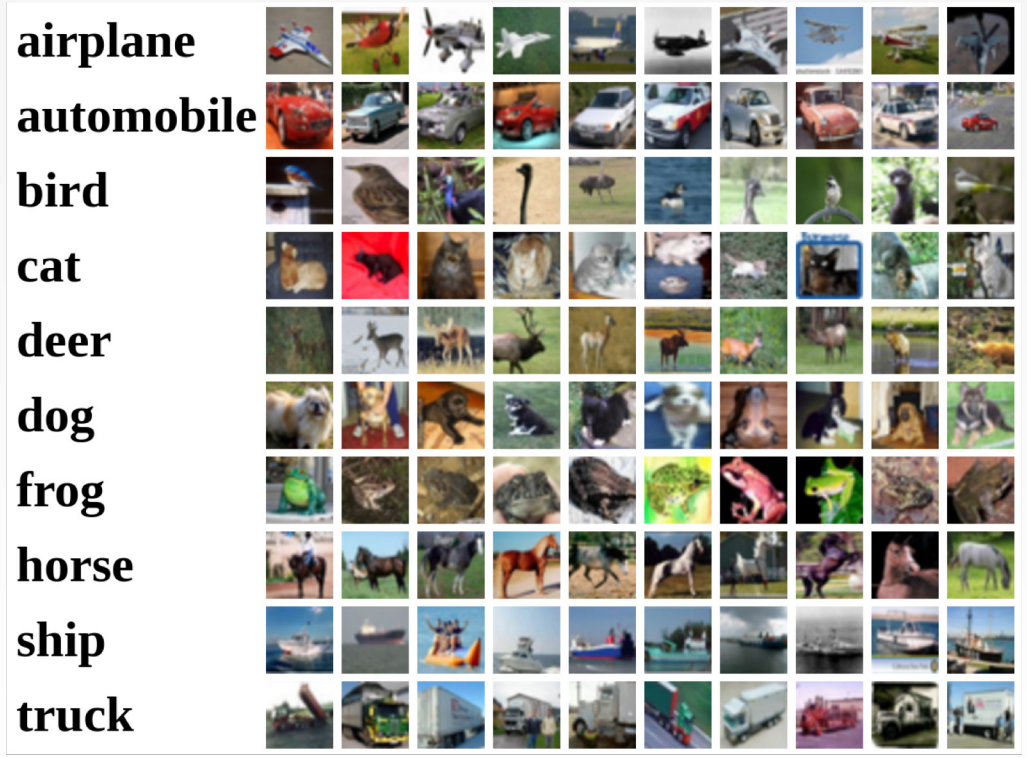
→ Memorize all data and labels

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

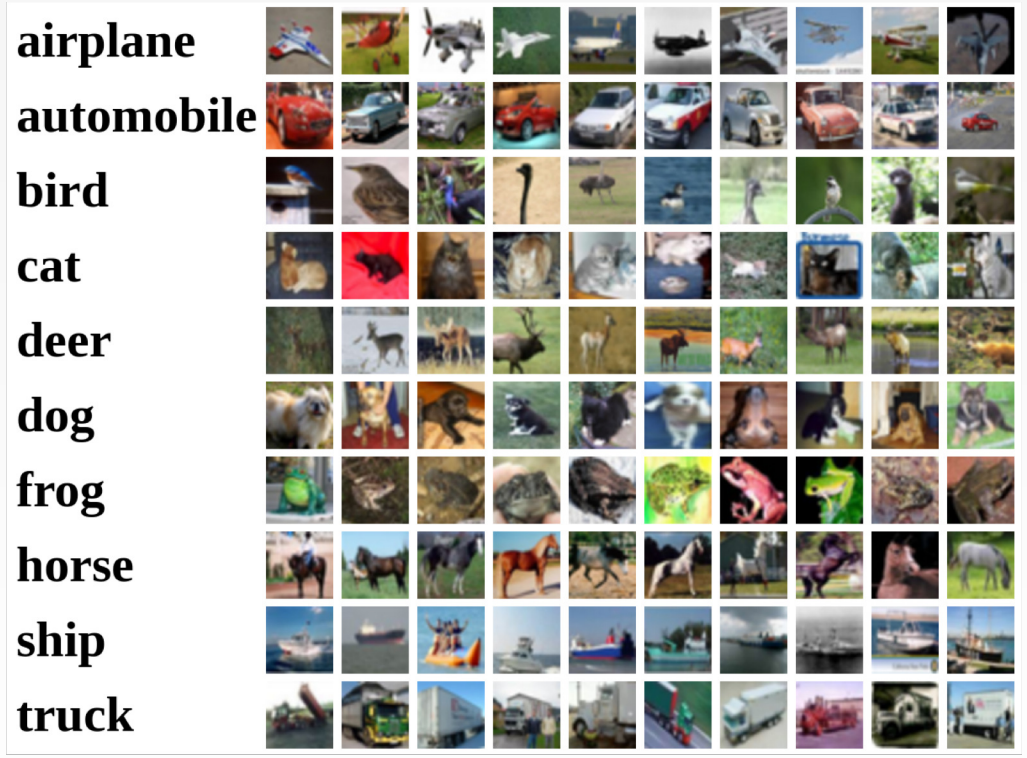
→ Predict the label of the most similar training image

EXAMPLE DATASET: CIFAR10

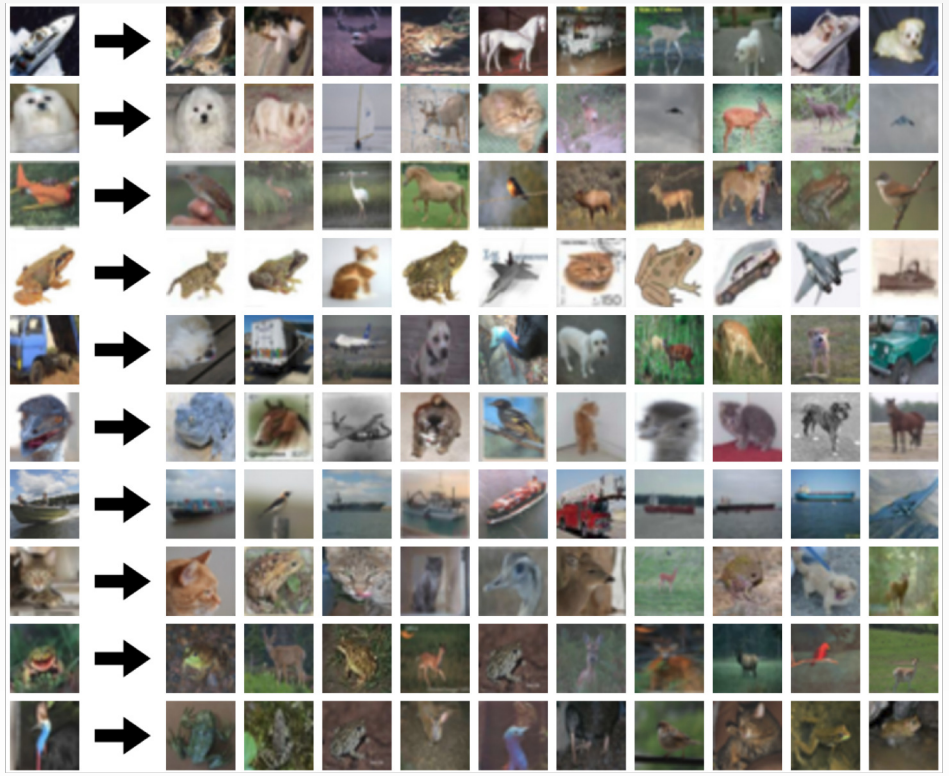
- **10** classes; **50K** training and **10K** testing images



EXAMPLE DATASET: CIFAR10



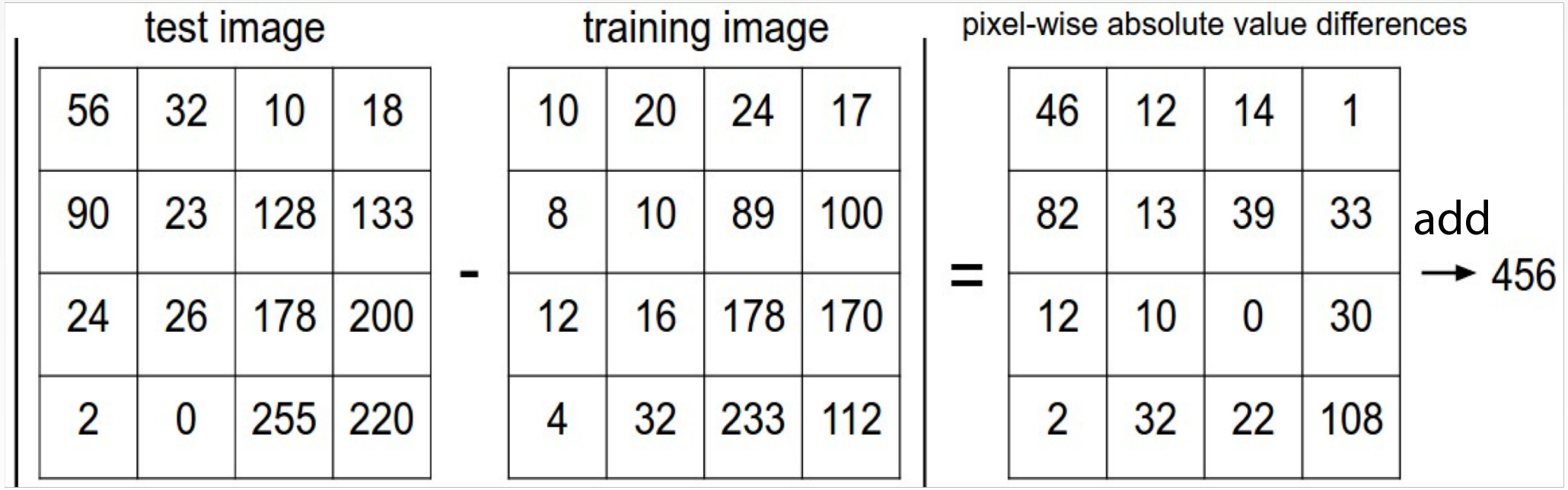
TEST IMAGES AND NEAREST NEIGHBORS



DISTANCE METRIC TO COMPARE IMAGES

L1 DISTANCE

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



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
```

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
```

Memorize training data

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
```

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

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
```

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

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
```

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

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

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
```

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

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

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

NEAREST NEIGHBOR CLASSIFIER

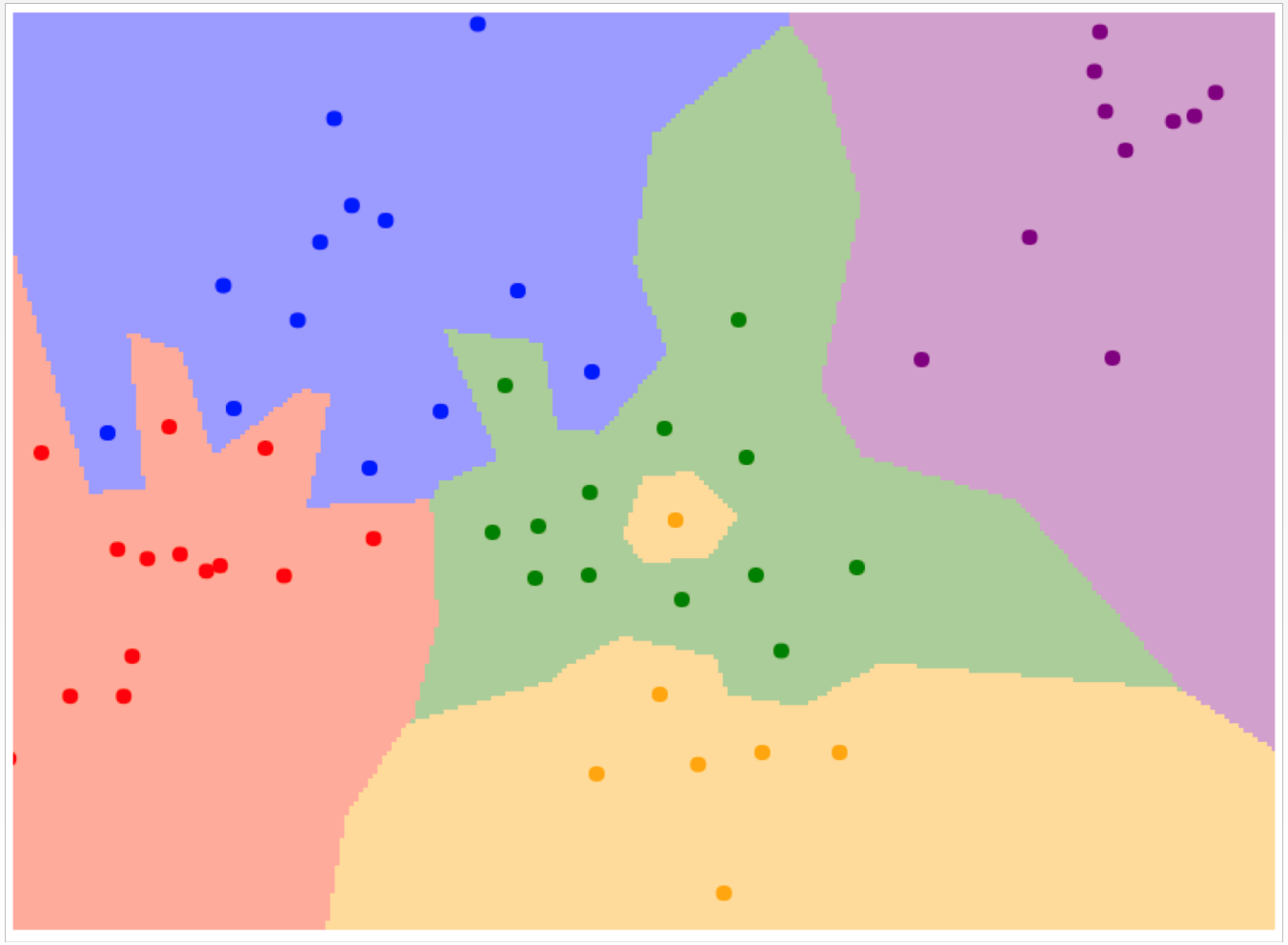
```python
import numpy as np

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
```

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

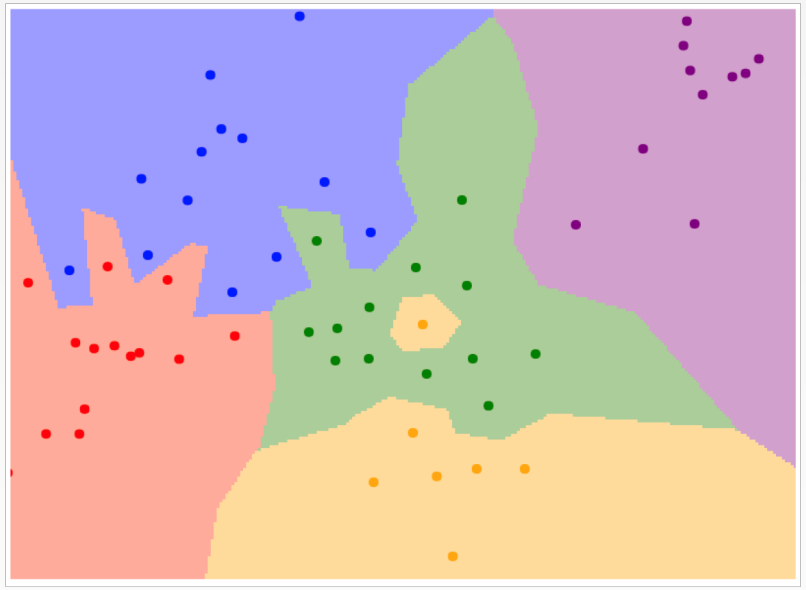
WHAT DO THE DECISION REGIONS LOOK LIKE?



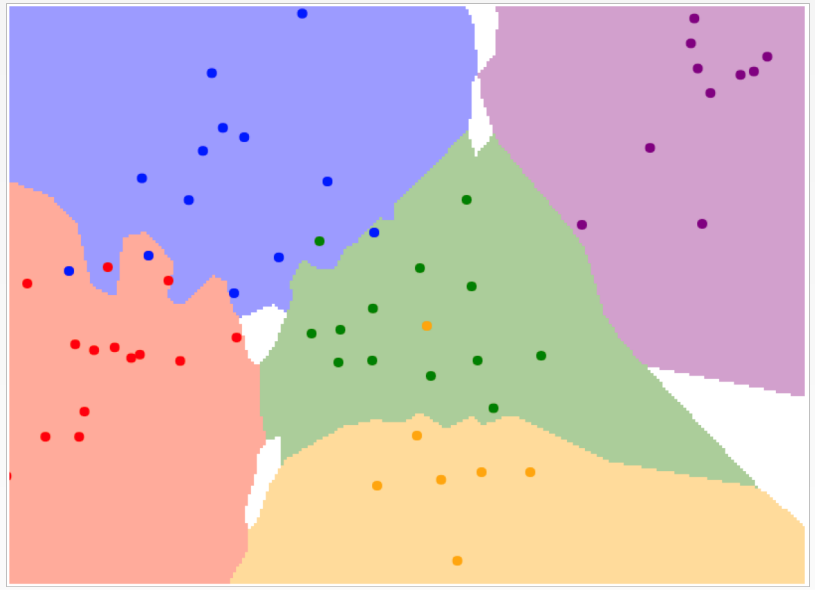
LIMITATIONS

- Island
  - Yellow island within the green cluster
- Fingers
  - Green region pushing into blue region
  - Noisy or spurious points
- Generalization
  - Instead of copying label from nearest neighbor, take **majority vote** from K nearest neighbors (i.e., closest points)

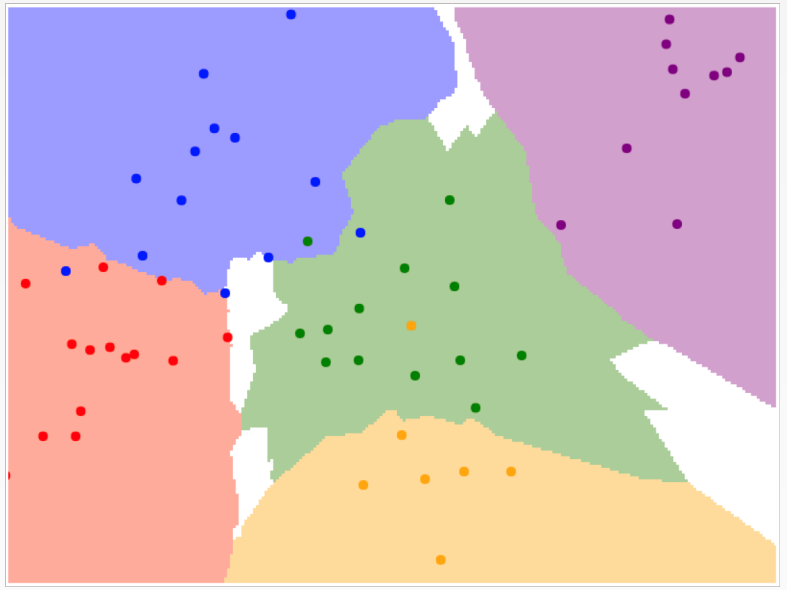
K-NEAREST NEIGHBORS



K = 1



K = 3

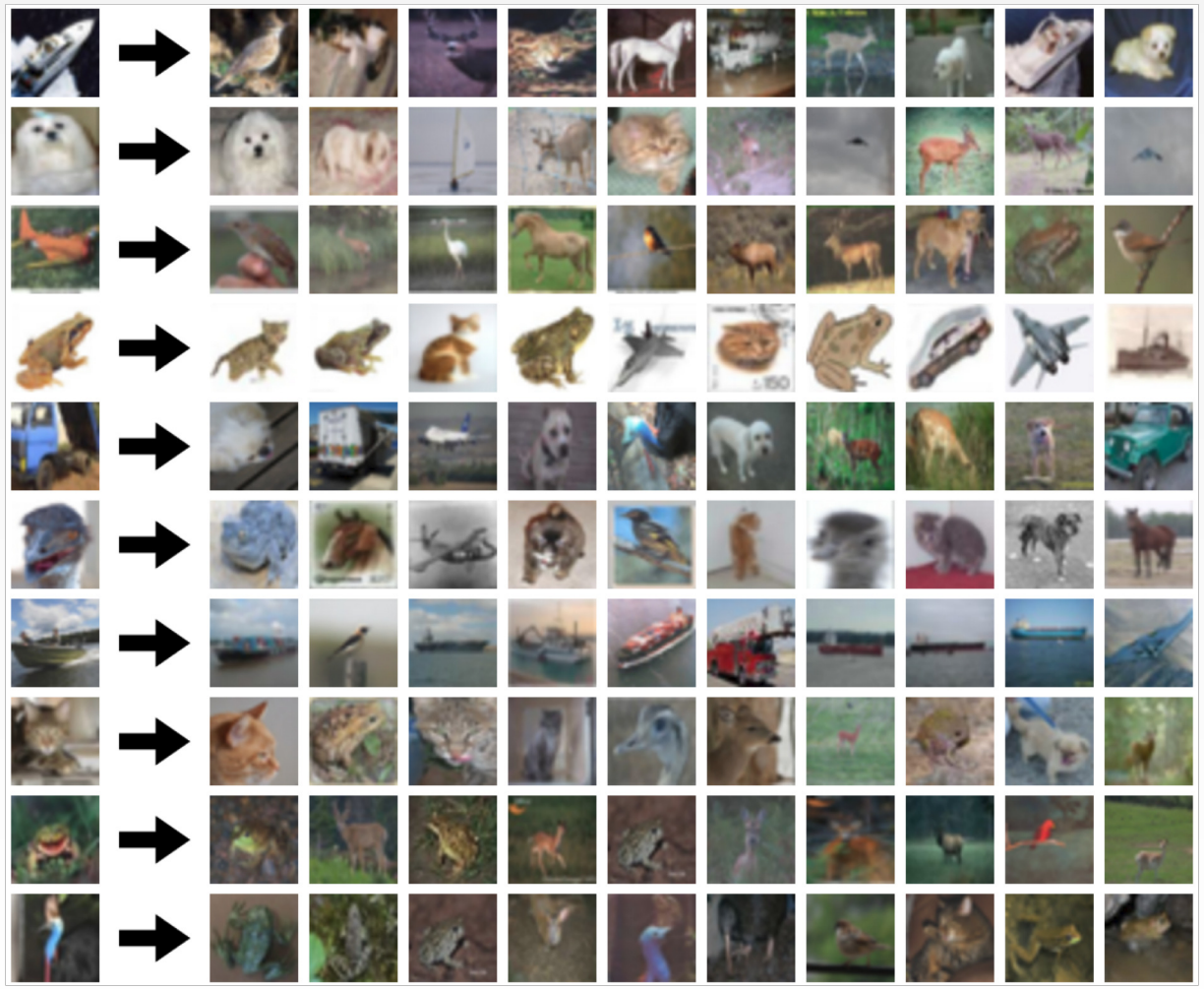


K = 5

COMPUTER VISION VIEWPOINTS

- Whenever we think of computer vision, it is useful to flip between different viewpoints:
  - **Geometric viewpoint:** Points in a high-dimensional space
  - **Visual viewpoint:** Concrete pixels in images
  - **Algebraic viewpoint:** In terms of vectors and matrices
- Images are high-dimensional vectors

WHAT DOES IT LOOK LIKE? (VISUAL VIEWPOINT)

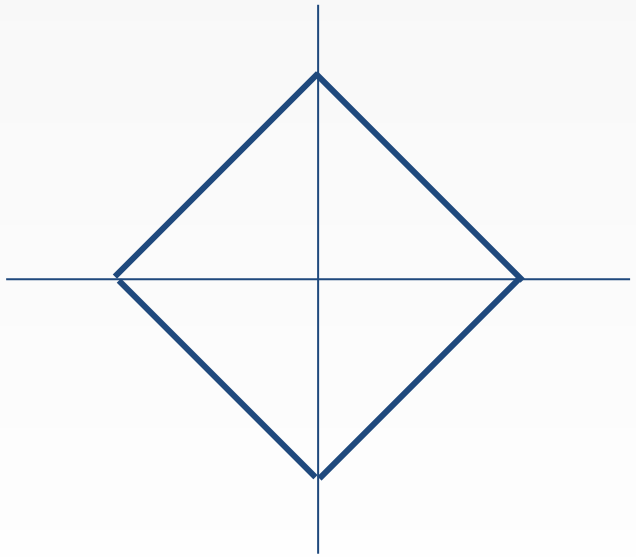


WHAT DOES IT LOOK LIKE? (VISUAL VIEWPOINT)

K-NEAREST NEIGHBORS: DISTANCE METRIC

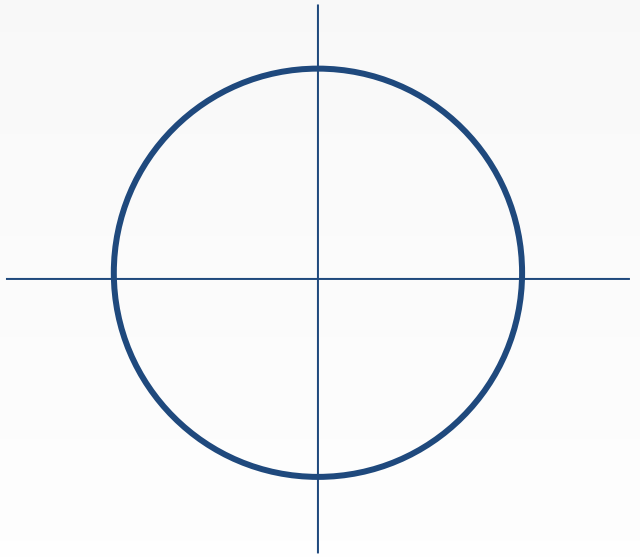
L1 (MANHATTAN) DISTANCE

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



L2 (EUCLIDEAN) DISTANCE

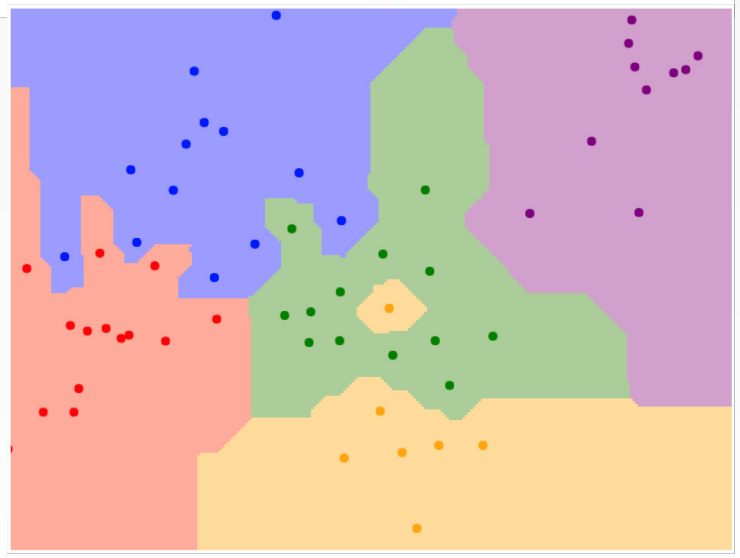
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



K-NEAREST NEIGHBORS: DISTANCE METRIC

L1 (MANHATTAN) DISTANCE

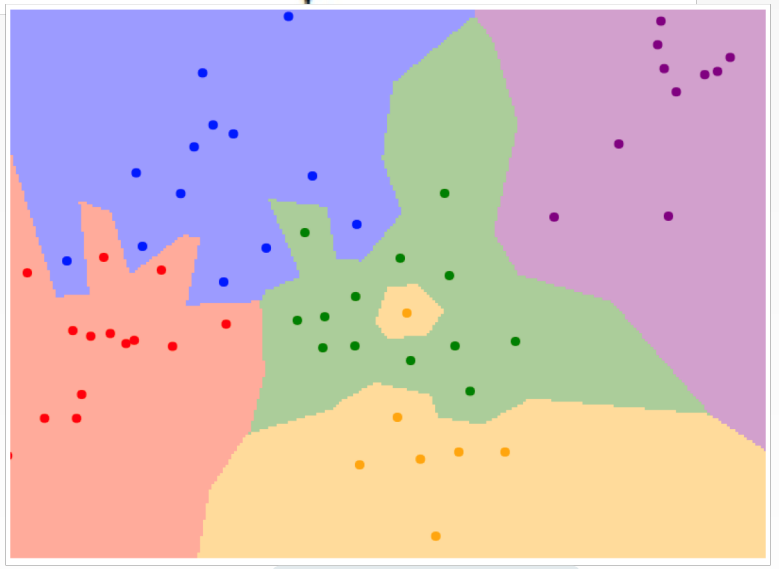
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (EUCLIDEAN) DISTANCE

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



K = 1

K-NEAREST NEIGHBORS: DEMO

- All examples are from an interactive demo
  - http://vision.stanford.edu/teaching/cs231n-demos/knn/

HYPERPARAMETERS

- What is the best value of **K** to use?
- What is the best **distance metric** to use?
- These are **hyperparameters**
  - Choices about the algorithm that we set rather than learn directly from the data

HYPERPARAMETERS

- What is the best value of **K** to use?
- What is the best **distance metric** to use?
- These are **hyperparameters**
  - Choices about the algorithm that we set rather than learn directly from the data
  - Very problem-dependent.
  - Must try them all out and see what works best.

SETTING HYPERPARAMETERS

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

SETTING HYPERPARAMETERS

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

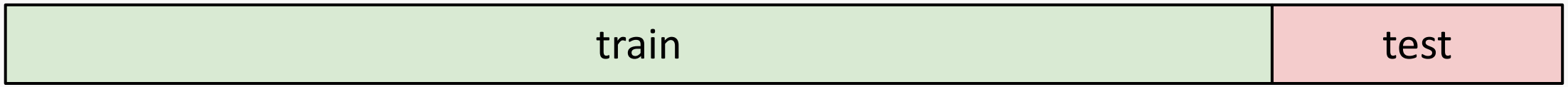
BAD: K = 1 always works perfectly on training data

SETTING HYPERPARAMETERS

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

BAD: K = 1 always works perfectly on training data

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data



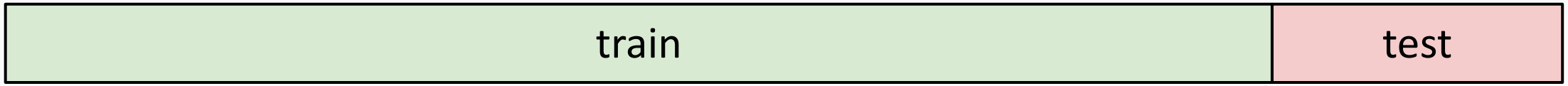
SETTING HYPERPARAMETERS

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

BAD: K = 1 always works perfectly on training data

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data



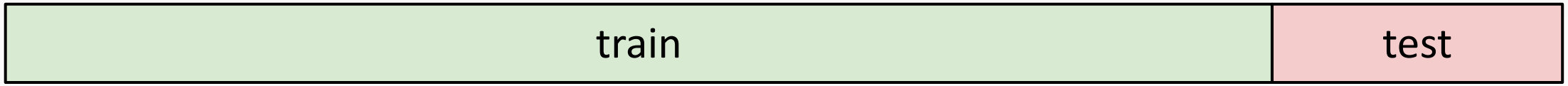
SETTING HYPERPARAMETERS

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

BAD: K = 1 always works perfectly on training data

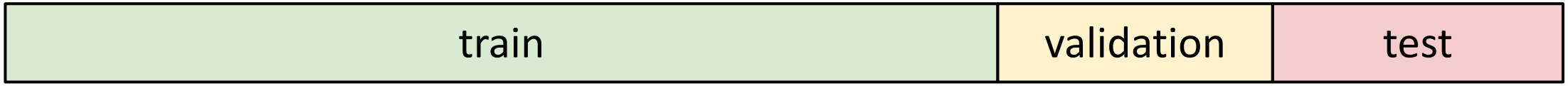
Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data



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

Better!



SETTING HYPERPARAMETERS

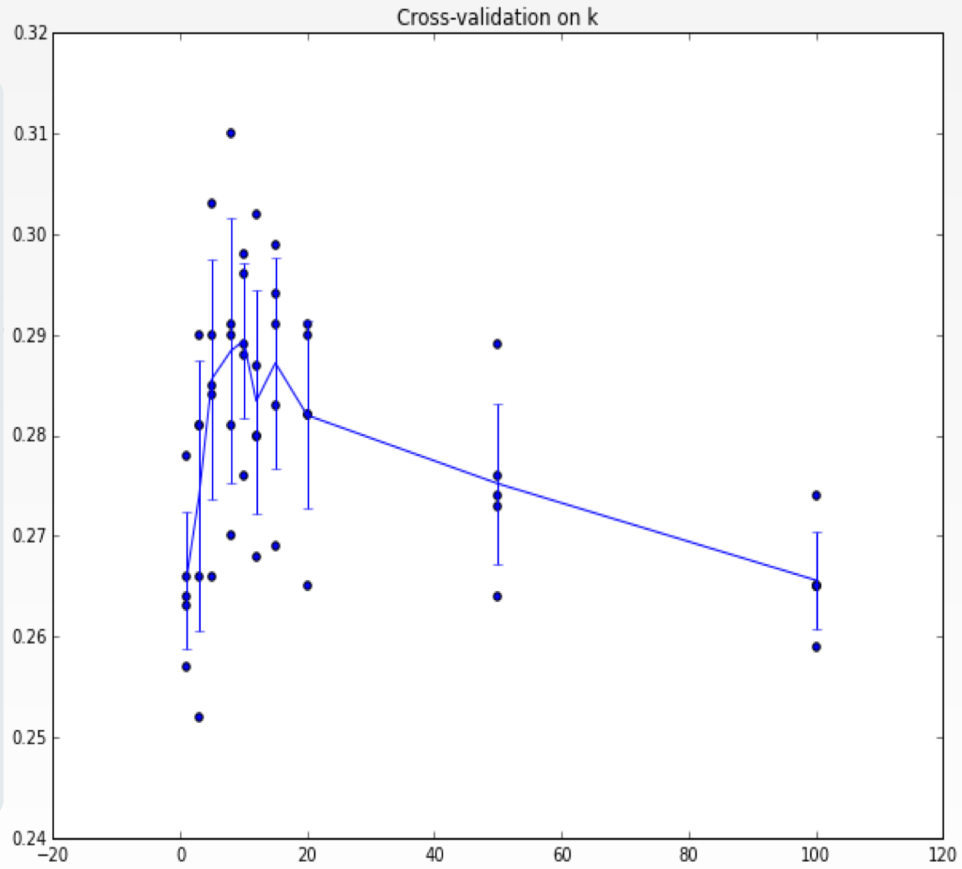
Your Dataset

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

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

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

SETTING HYPERPARAMETERS



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

Each point: single outcome.

The line goes through the mean, bars indicate standard deviation

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

K-NEAREST NEIGHBOR: LIMITATIONS

- K-Nearest Neighbor **never used** on images
  - Very slow at test time
  - Distance metrics on pixels are not informative

ORIGINAL

BOXED

SHIFTED

TINTED

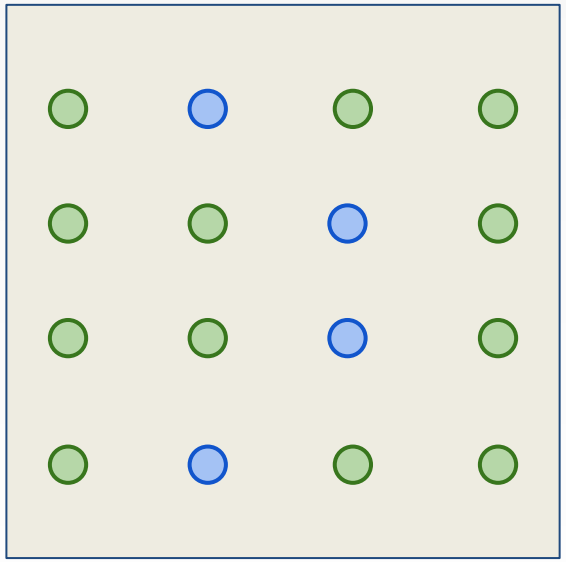
(all three images have same L2 distance to the one on the left)

K-NEAREST NEIGHBOR: LIMITATIONS

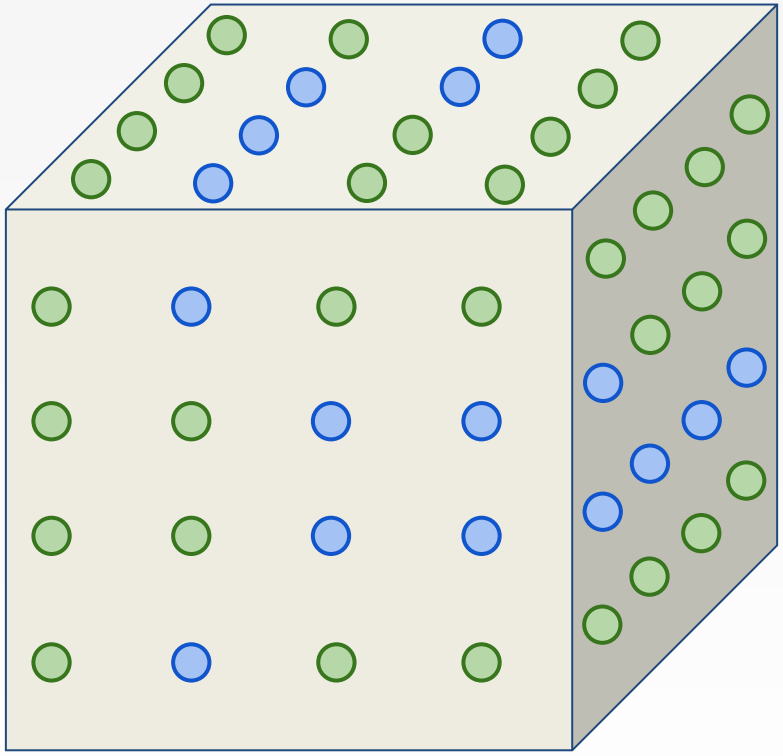
- Curse of dimensionality

Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2



Dimensions = 3
Points = 4^3

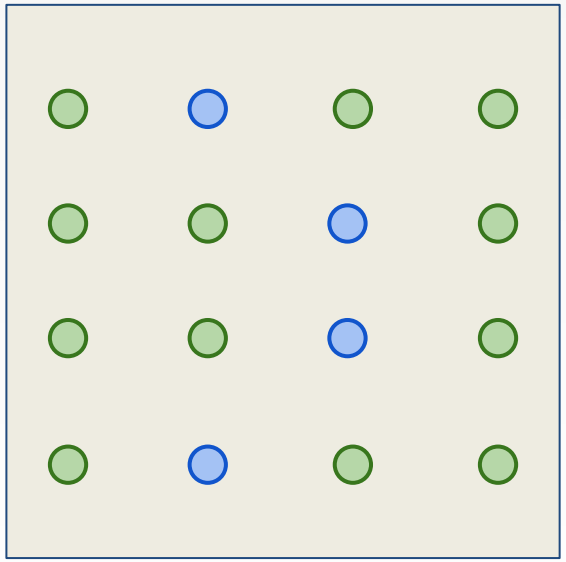


K-NEAREST NEIGHBOR: LIMITATIONS

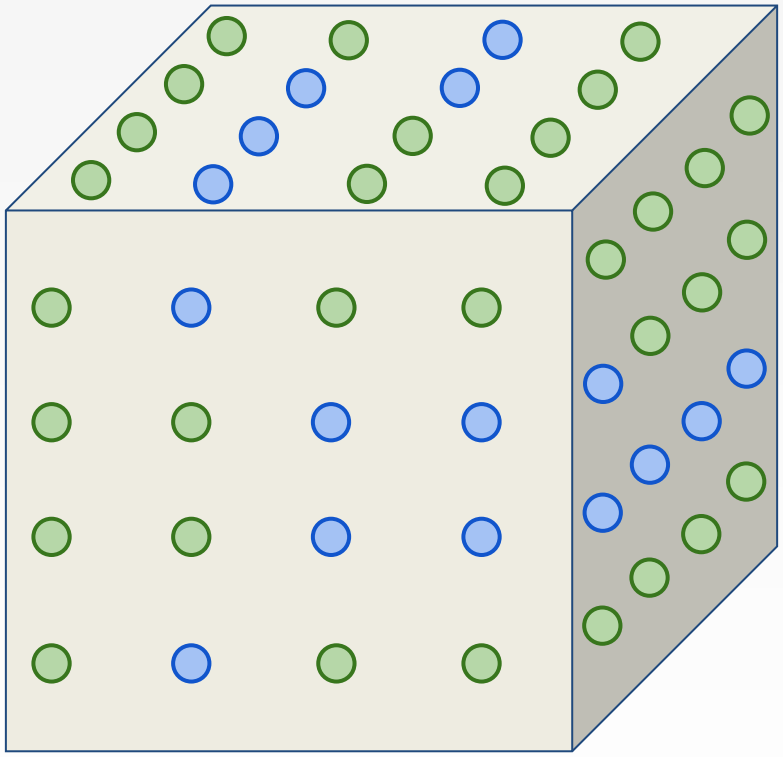
- Curse of dimensionality

Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2



Dimensions = 3
Points = 4^3



K-NEAREST NEIGHBOR: SUMMARY

- In **Image classification**, we start with a **training set** of images and labels, and must predict labels on the **test set**
- **K-Nearest Neighbor** classifier predicts labels based on 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

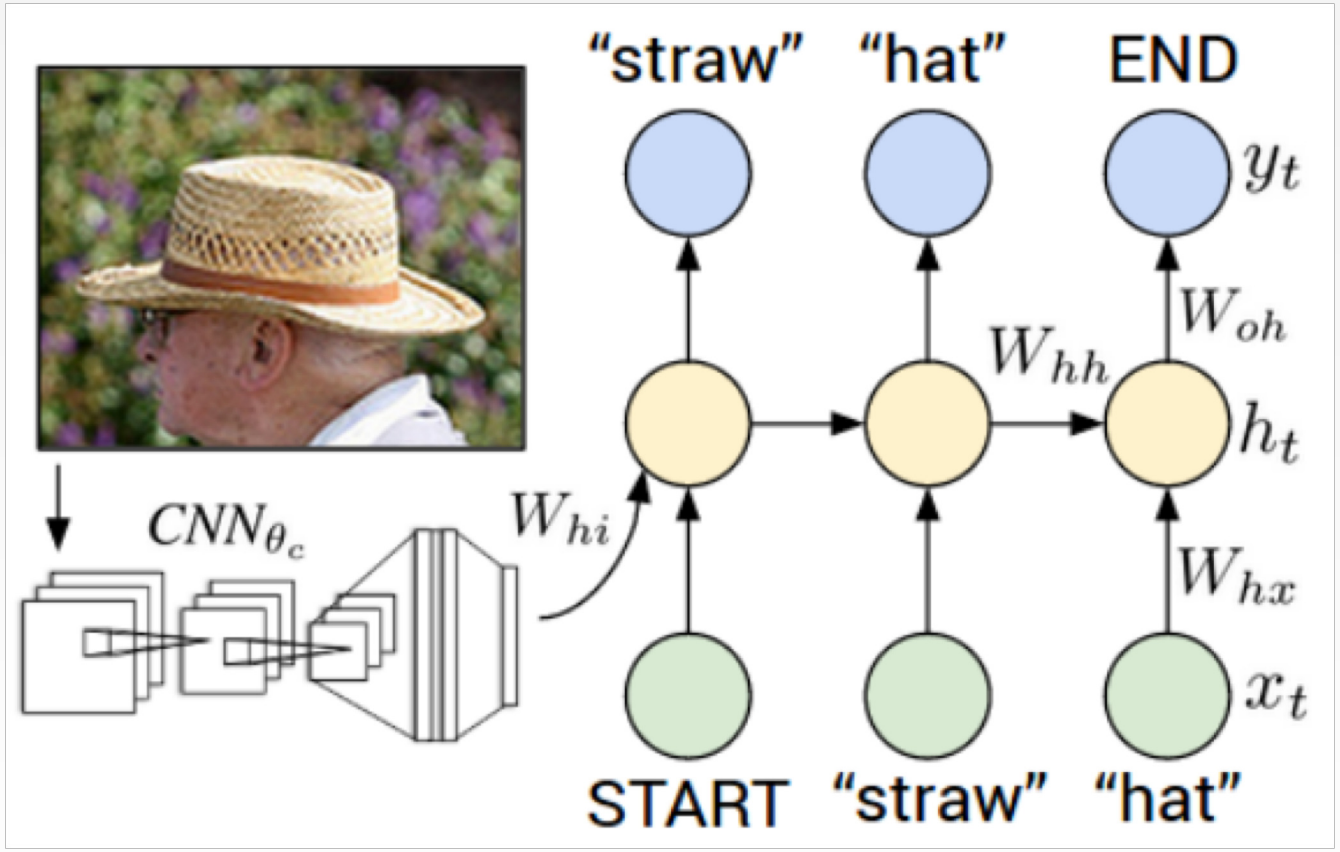
NEURAL NETWORK: LEGO BLOCKS

LINEAR CLASSIFIERS

NEURAL NETWORK: LEGO BLOCKS

Two young girls are playing with lego toy.

IMAGE CAPTIONING



CNN + RNN

RECALL: CIFAR10 DATASET

10 classes.
50K training images.
10K test images.
Each image is **32x32x3**

PARAMETRIC APPROACH

Image

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

f(**x**,**W**)

W
parameters
or weights

10 numbers
giving class scores

Highest score
maps to
predicted class

PARAMETRIC APPROACH

- In K-Nearest Neighbor classifier,
  - We use the training data during prediction
- With a parametric approach,
  - We summarize our knowledge of training data in the parameters.
  - At test time, can discard training data since only parameters are needed
  - Deep learning is all about coming up with right structure for the parametric function **f()**

PARAMETRIC APPROACH: LINEAR CLASSIFIER

Image

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

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

$$f(\mathbf{x},\mathbf{W})$$

10 numbers
giving class scores

W
parameters
or weights

PARAMETRIC APPROACH: LINEAR CLASSIFIER

Image

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

$f(x,W) = Wx$

10x1 3072x1 10x3072

$f(\mathbf{x},\mathbf{W})$

10 numbers
giving class scores

W
parameters
or weights

PARAMETRIC APPROACH: LINEAR CLASSIFIER

Image

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

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

10x1 (f(x,W)) = 10x3072 (W) · 3072x1 (x) + 10x1 (b)

$$f(\mathbf{x},\mathbf{W})$$

10 numbers
giving class scores

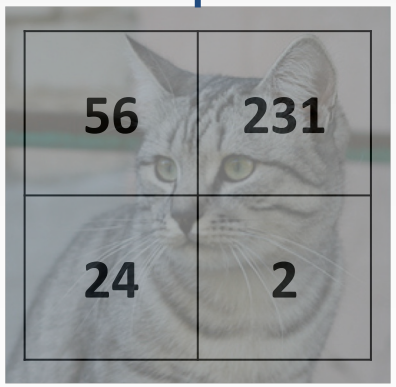
W

parameters
or weights

INTERPRETATION: ALGEBRAIC VIEWPOINT

Image with 4 pixels, and 3 classes (cat/dog/ship)

Stretch pixels into column



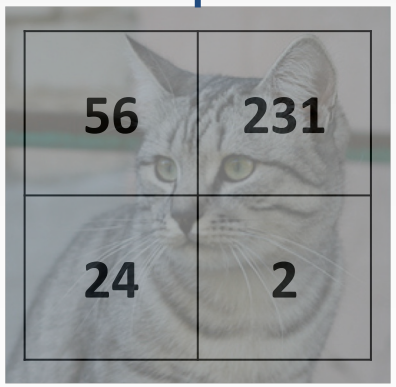
Input image

56
231
24
2

INTERPRETATION: ALGEBRAIC VIEWPOINT

Image with 4 pixels, and 3 classes (cat/dog/ship)

Stretch pixels into column



Input image

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

W

56
231
24
2

+

1.1
3.2
-1.2

b

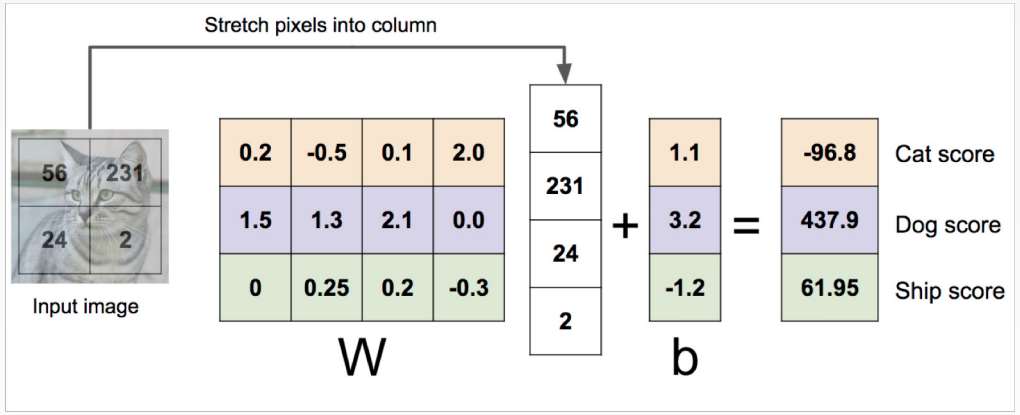
=

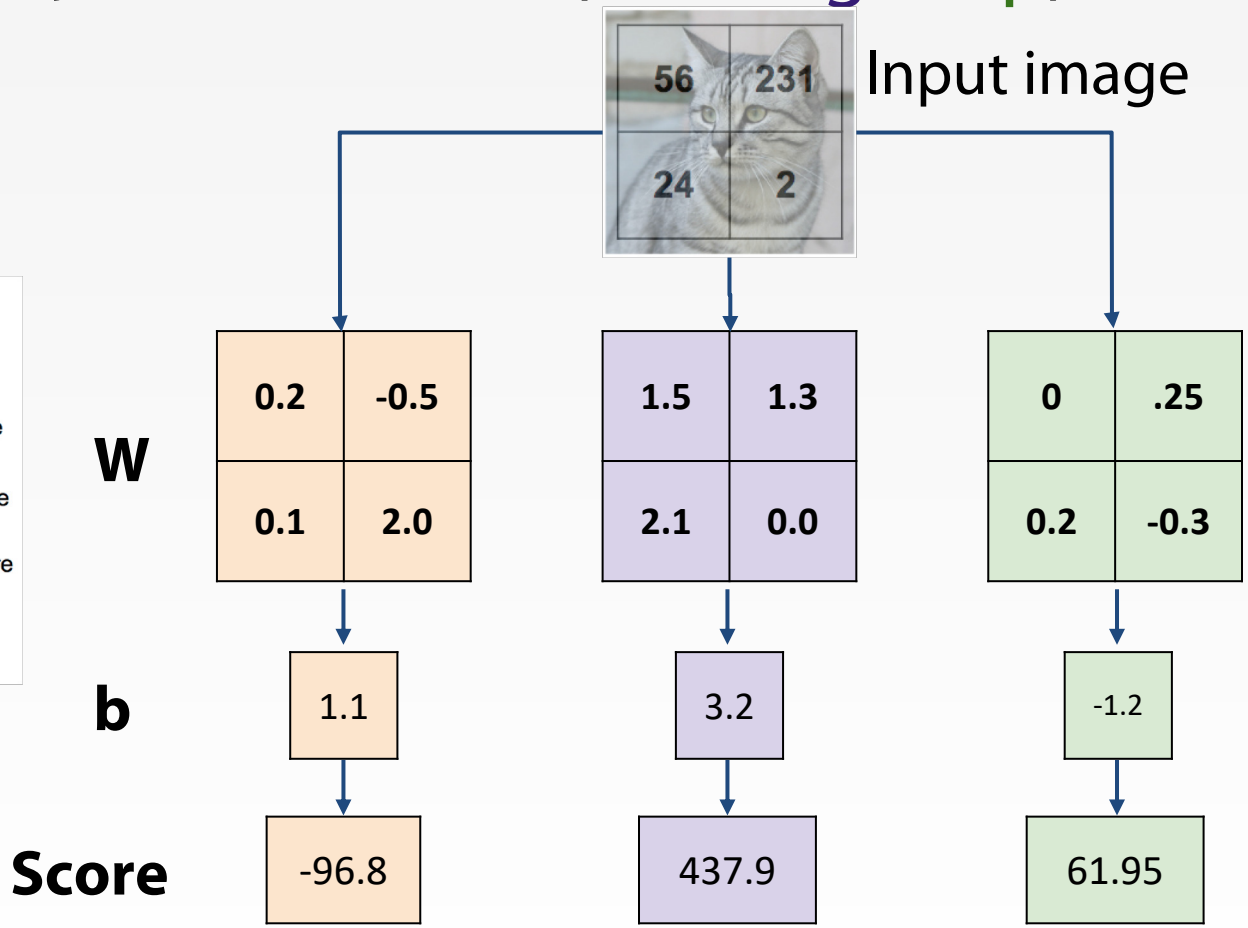
-96.8	Cat score
437.9	Dog score
61.95	Ship score

INTERPRETATION: ALGEBRAIC VIEWPOINT

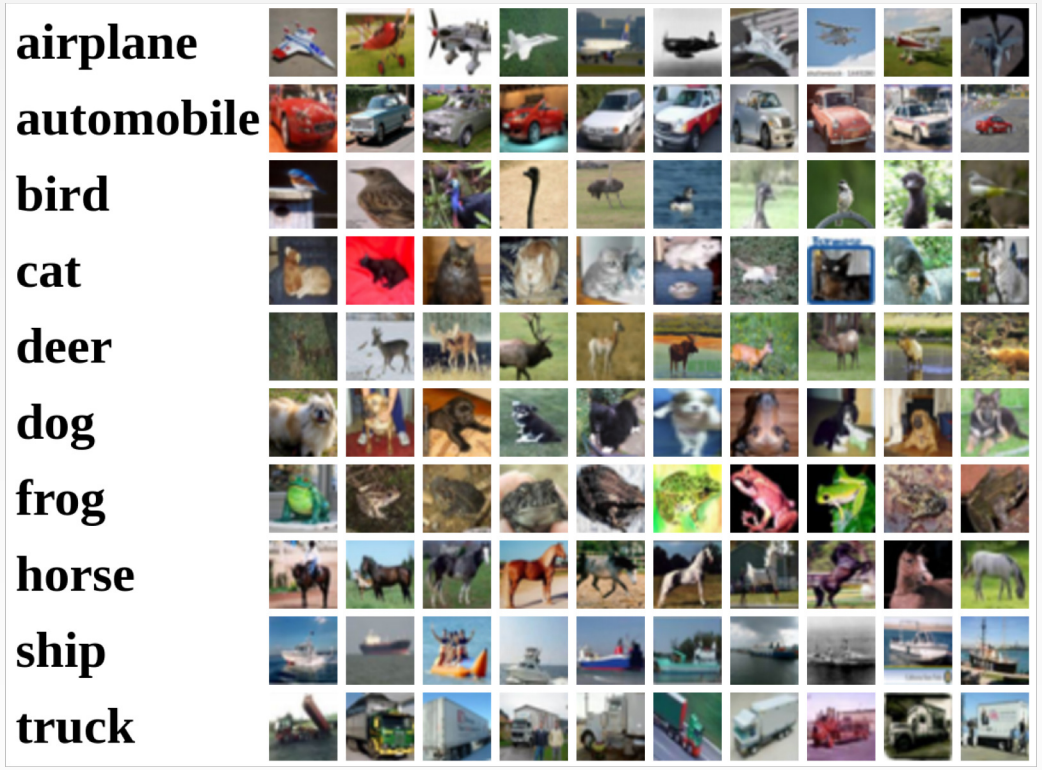
Image with 4 pixels, and 3 classes (cat/dog/ship)

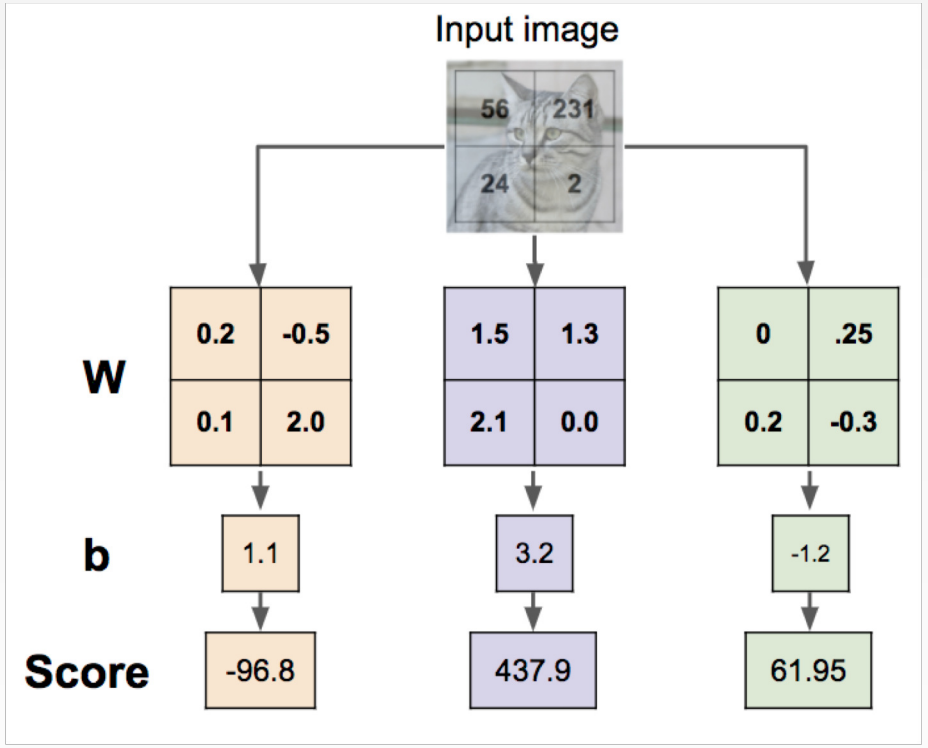
$$f(x,W) = Wx$$



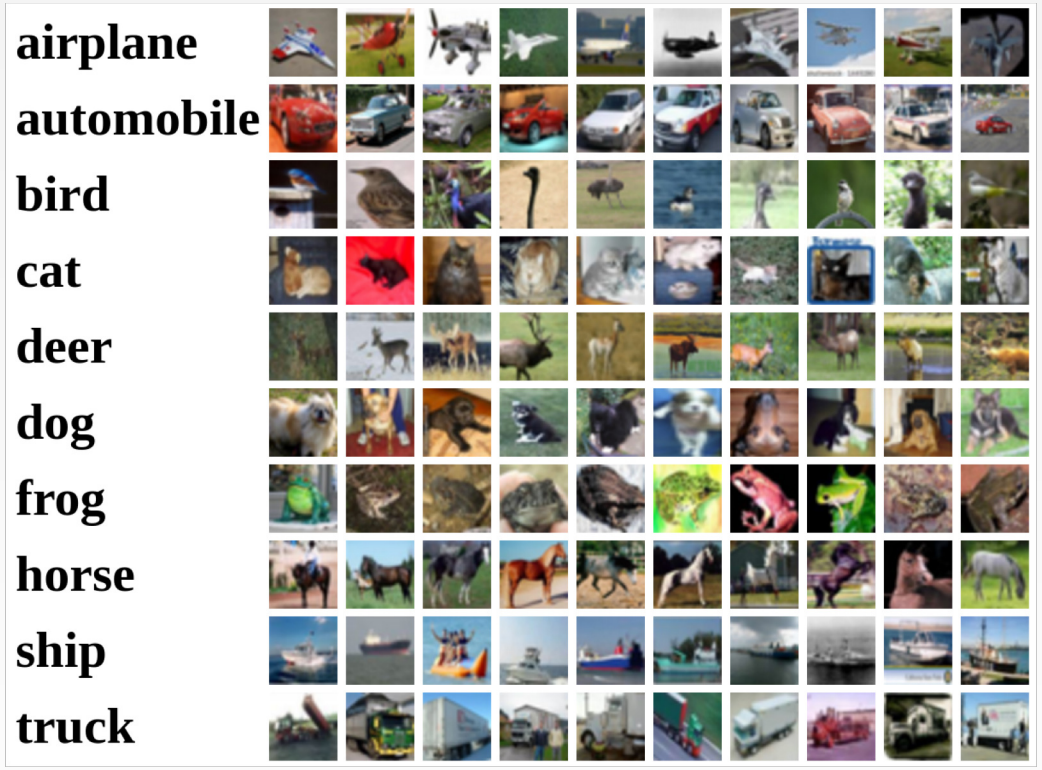


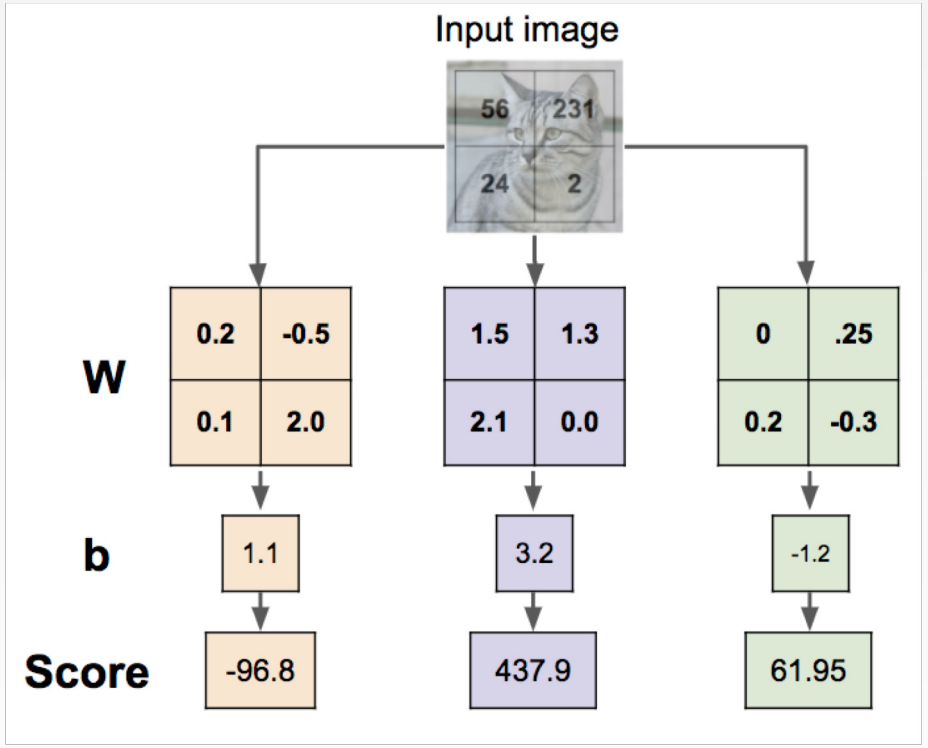
INTERPRETATION: VISUAL VIEWPOINT





VISUAL VIEWPOINT





VISUAL VIEWPOINT

- Each row in the weight matrix can be unraveled into an image which serves a template for that class
  - Problem is that the linear classifier is only learning **one template** for each class
  - Averages out variations in the class
  - Example: Two-headed horse template

VISUAL VIEWPOINT

- Neural networks can achieve better accuracy than a linear classifier since they can learn multiple templates for each class

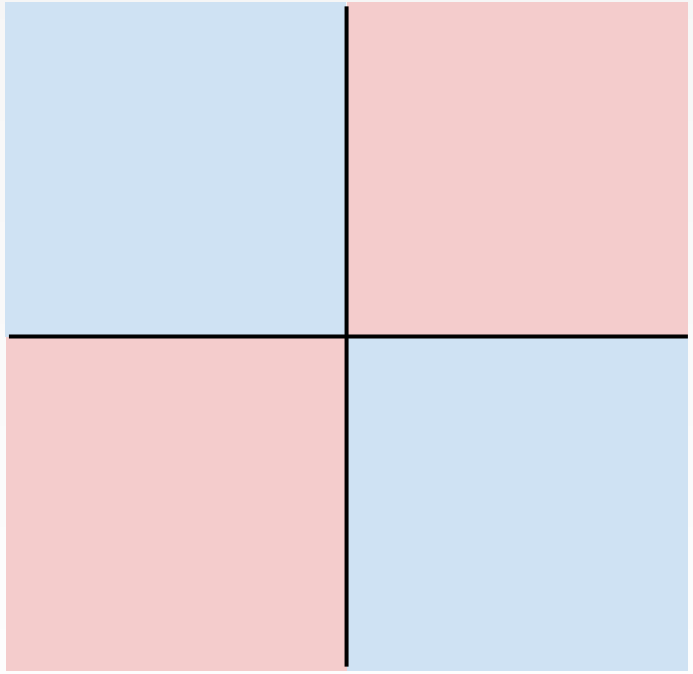
GEOMETRIC VIEWPOINT

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

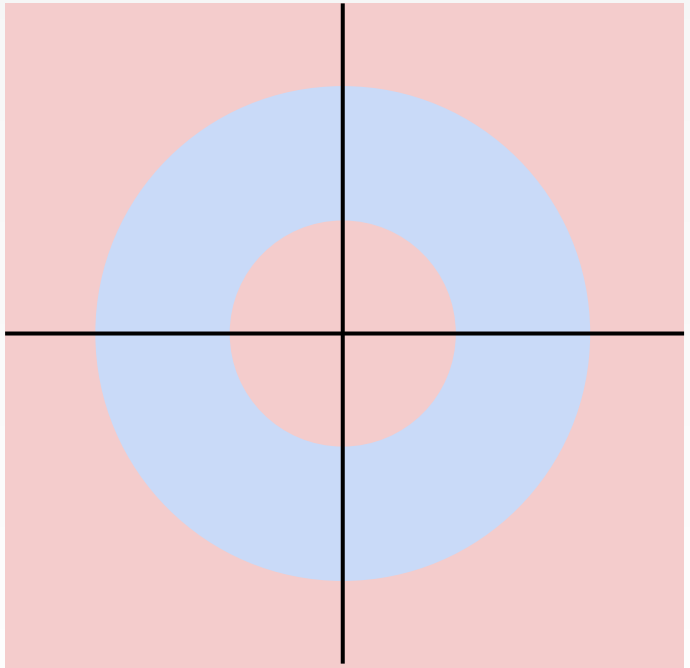
Array of **32x32x3** numbers
(3072 numbers total)

HARD CASES FOR LINEAR CLASSIFIER

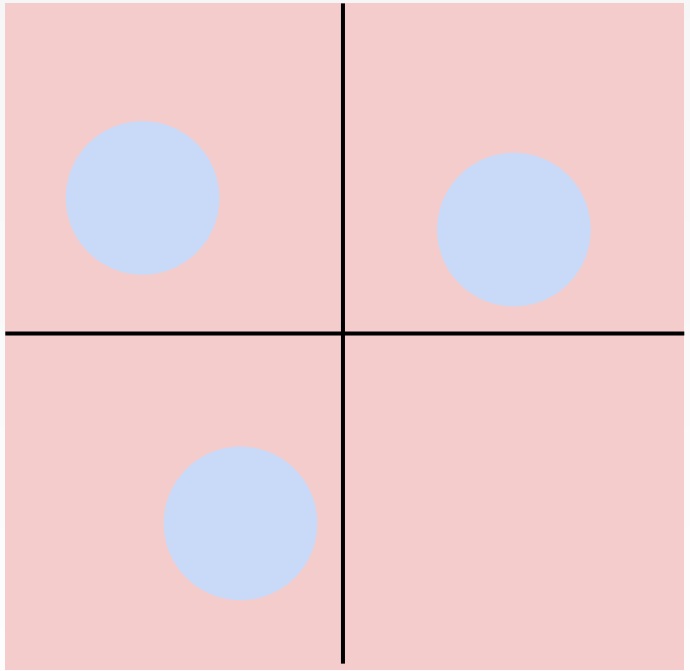
Class 1:
First and third quadrants
Class 2:
Second and fourth quadrants



Class 1:
1 <= L2 norm <= 2
Class 2:
Everything else



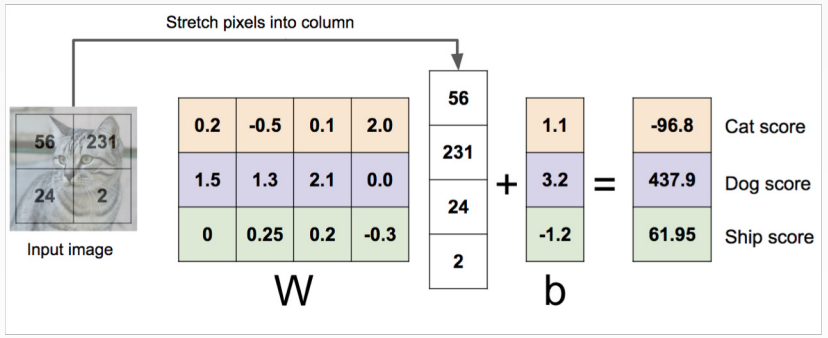
Class 1:
Three modes
Class 2:
Everything else



LINEAR CLASSIFIER: THREE VIEWPOINTS

Algebraic Viewpoint

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

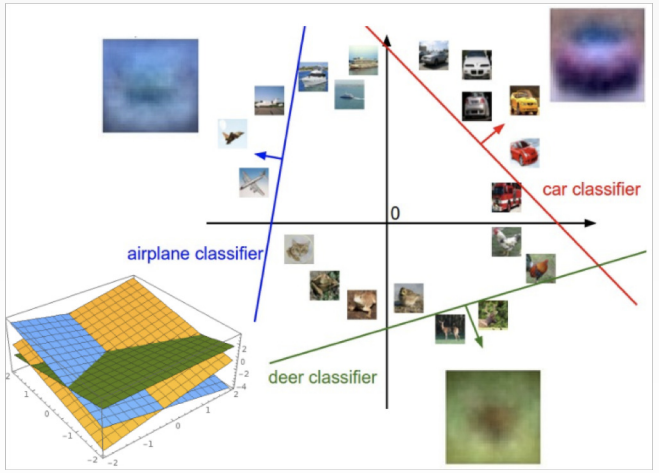


Visual Viewpoint

One template per class

Geometric Viewpoint

Hyperplanes cutting up space



SO FAR: DEFINED A (LINEAR) SCORE FUNCTION

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



Example class scores for 3 images for some W:
How can we tell whether this **W** is good or bad?

airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

PARTING THOUGHTS

- Image classification is a core vision task
  - Nearest neighbor is a non-parametric approach that works well for non-visual data
  - Linear classifier is a parametric approach that works well for visual data
  - It is useful to flip between different viewpoints to interpret a given classifier

NEXT WEEK

COMING UP:

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

- LOSS FUNCTION (quantifying what it means to have a "good" W)
- OPTIMIZATION (start with random W and find a W that minimizes the loss)
- CONVNETS! (tweak the functional form of f)