# CS163: Deep Learning for Computer Vision

# Lecture 2:
# Image Classification

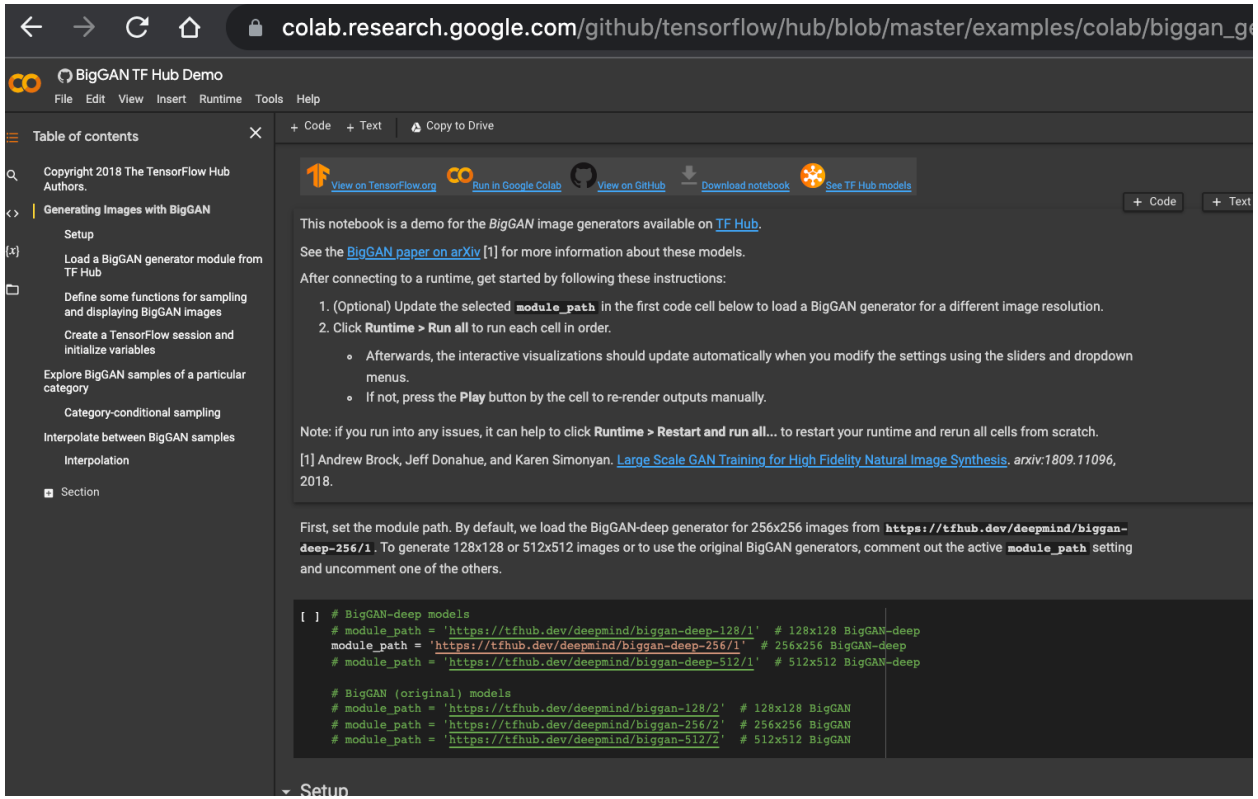
Courtesy of Prof. Justin Johnson at Umich and the team of Stanford CS231n for part of the slides

# Announcement: Assignment 1 is released

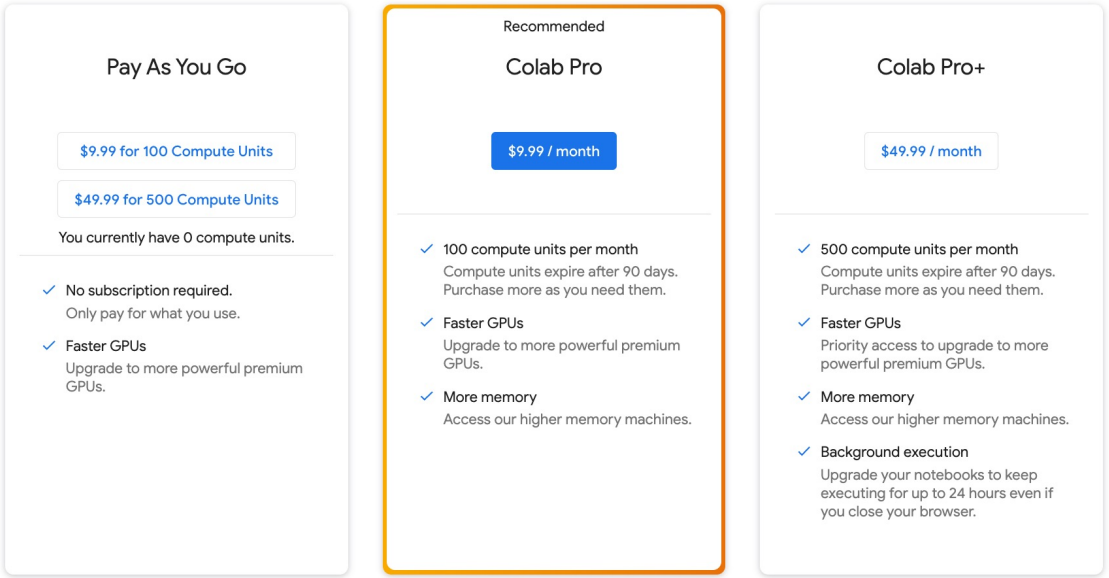
- How to use Google Colab
- How to implement a PyTorch data loader to load and preprocess the MiniPlaces dataset.
- How to implement a linear/logistic/softmax regression classifier
- How to build and train a fully connected neural network from scratch and using built-in PyTorch modules.
- Due: Sunday, Oct 20 (please start early, a lot of questions to solve)
- https://github.com/UCLAdeepvision/CS163-Assignments-2024Fall

# Google Colab: Cloud Computing in the Browser

Free GPU and TPU to use (a maximum of 12-hour training time)




Monthly plan with better GPUs and longer training time



# Google Colab: Cloud Computing in the Browser

Useful Colab Tutorials:

- https://towardsdatascience.com/getting-started-with-google-colab-f2fff97f594c
- https://colab.research.google.com/notebooks/intro.ipynb#recent=true
- https://medium.com/swlh/the-best-place-to-get-started-with-ai-google-colab-tutorial-for-beginners-715e64bb603b

# Announcement: TA Discussion Session: Intro to Colab

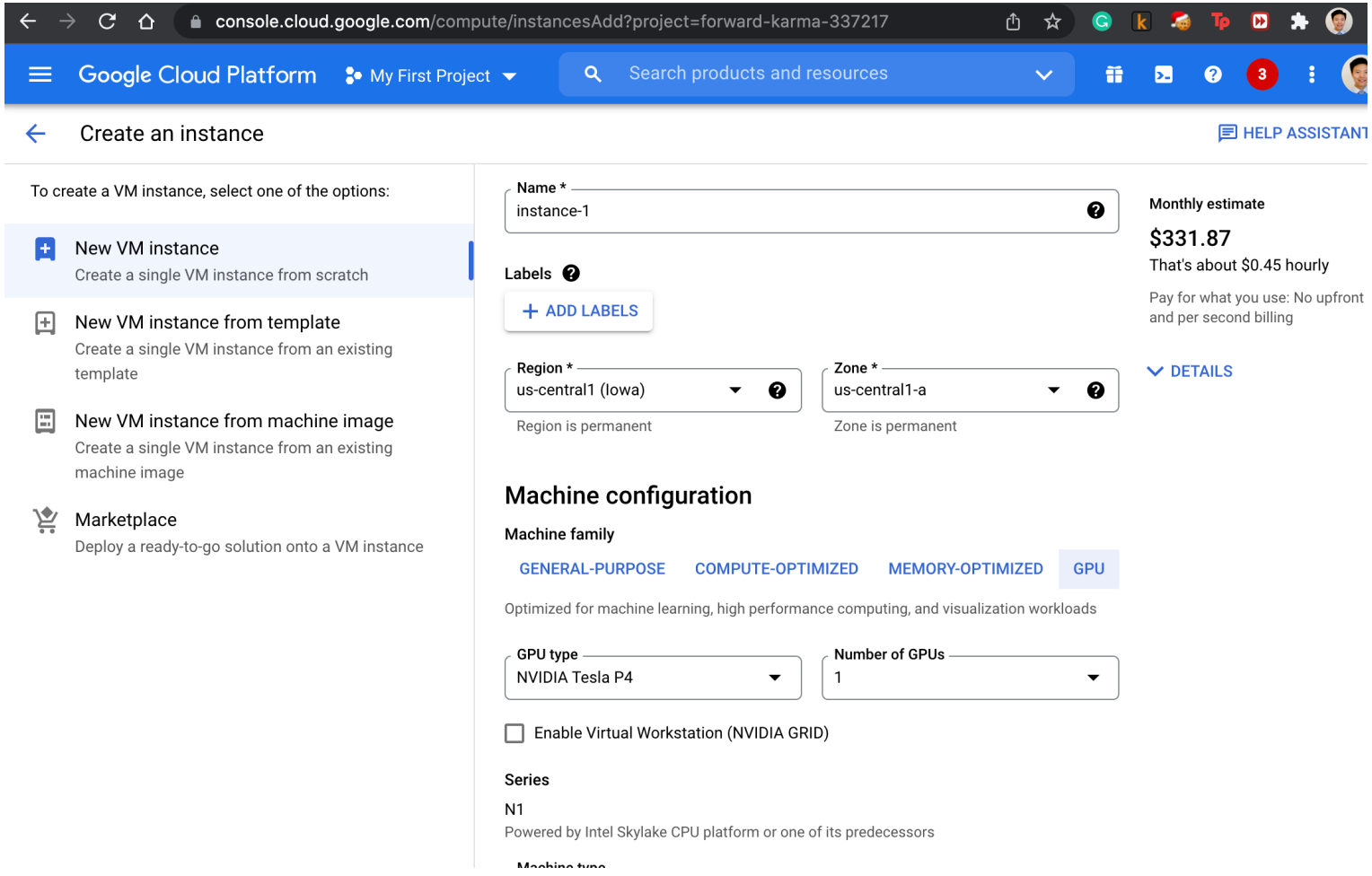
**Two discussion sessions:**

- DIS 1A F 8:00 am-9:50 am PUB AFF 1234
- DIS 1B F 10:00 am-11:50 am PUB AFF 1234
- TA Office Hour F 2:00pm-3:30pm HAINES A2

- Similar content, choose one to go (but not so sure about the classroom size…)
- Limited number of presentation slot per session
- General plans: programming basics, colab, google cloud, assignments, project presentations

# Google Cloud Education Credit for Course Project

- $50 credit per enrolled student (invitation will be sent later)
- If 2 students in a group, ~ 220 hours for VM instance with P4 GPU



# A relevant seminar talk

**THURSDAY: OCTOBER 3, 2024**
**PLACE: 3400 BOELTER HALL**
**TIME: 4:15 PM – 5:45 PM**

# (Compositional, efficient, and robust): Visual Concept Learning in the #GenAI era

**Yezhou "YZ" Yang**
***Arizona State University***

**Abstract:**

The goal of Computer Vision, as coined by Marr, is to develop algorithms to answer What are Where at When from visual appearance. The speaker, among others, recognizes the importance of studying underlying entities and relations beyond visual appearance, following an Active Perception paradigm. This talk will present the speaker's efforts over the last decade, ranging from 1) reasoning beyond appearance for vision and language tasks (VQA, captioning, T2I, etc.), and addressing their evaluation misalignment (SMURF, ConceptBed), through 2) reasoning about implicit properties (such as spatial consistency – SPRIGHT and REVISION), to 3) their roles in efficient (ECLIPSE) and reliable (WOUAF, R.A.C.E.) image #GenAI models.

**Bio:**

Yezhou (YZ) Yang is an Associate Professor and a Fulton Entrepreneurial Professor in the School of Computing and Augmented Intelligence (SCAI) at Arizona State University. He founded and directs the Active Perception Group, and currently serves as the topic lead (situation awareness) at the Institute of Automated Mobility, Arizona Commerce Authority. He is also a thrust lead (AVAI) at Advanced Communications Technologies (ACT, a Science and Technology Center under the New Economy Initiative, Arizona). His work includes exploring visual primitives and representation learning in visual (and language) understanding, grounding them by natural language and high-level reasoning over the primitives for intelligent systems, secure/robust AI/GenAI, and V&L model evaluation alignment. Yang is a recipient of the Qualcomm Innovation Fellowship 2011, the NSF CAREER Award 2018, the Amazon AWS Machine Learning Research Award 2019, and the best paper award at the 2024 CVPR Vision Datasets Understanding workshop. He received his Ph.D. from the University of Maryland at College Park and a B.E. from Zhejiang University, China.

# **Image Classification**: A core computer vision task

**Input**: image

**Output**: Assign image to one of a fixed set of categories

chair
bed
sofa
table
cabinet

Source image is from https://www.ikea.com/en/products/coffee-and-side-table/armchairs/ekenaset-art-30453028

# **Problem**: Semantic Gap

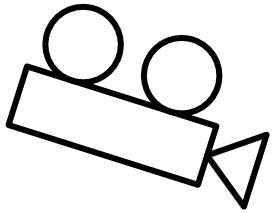
```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88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

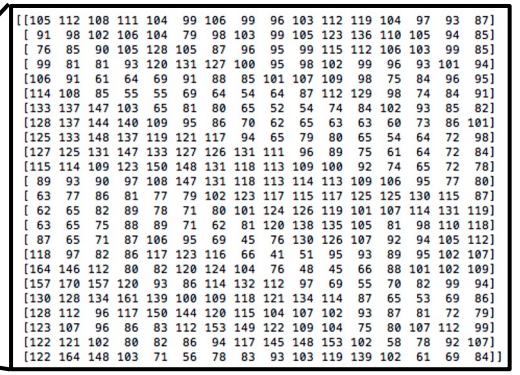
What the computer sees

An image is just a big grid of numbers between [0, 255]:

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

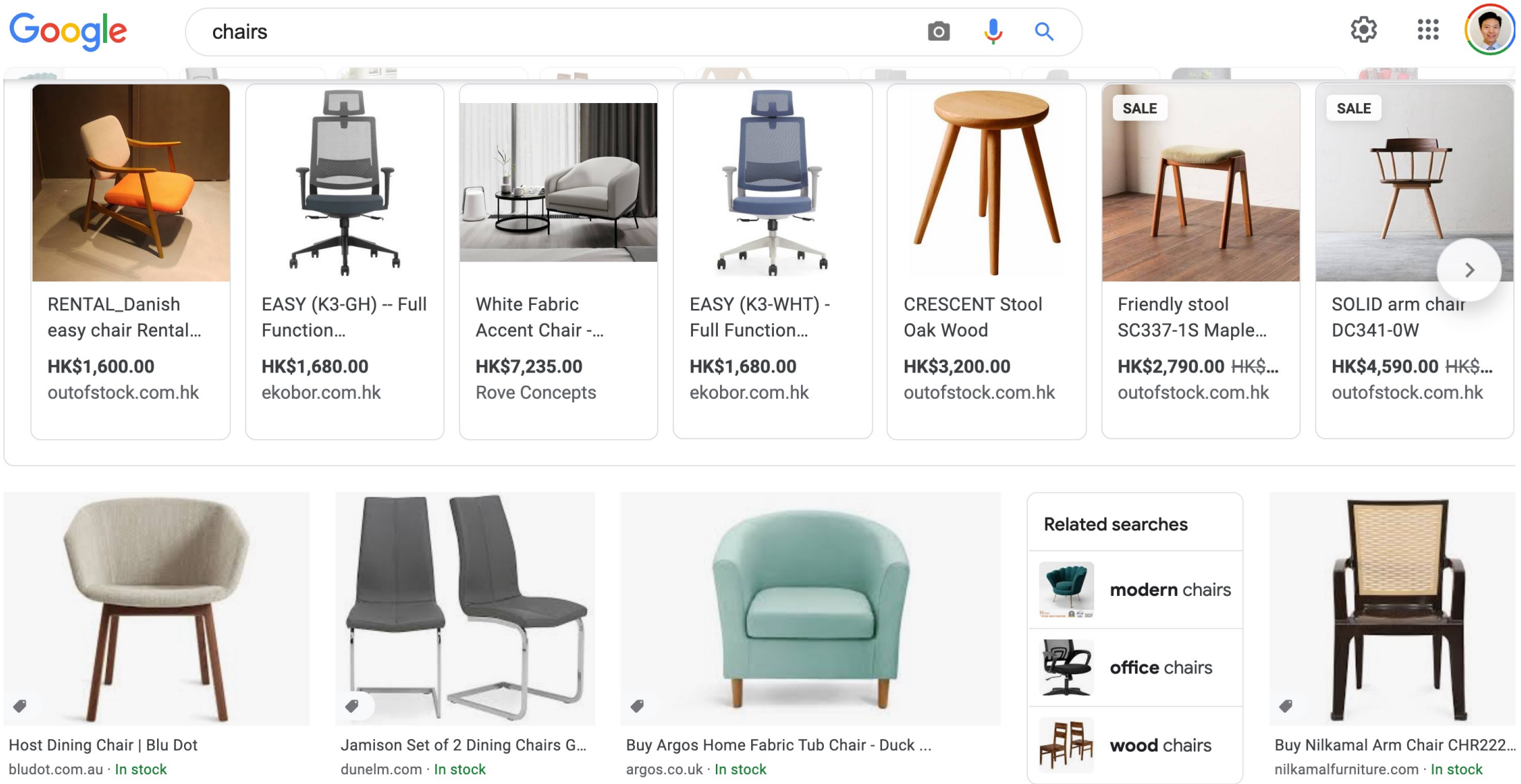
# Challenges: Viewpoint Variation





All pixels change when the camera moves!

# Challenges: Intraclass Variation



# **Challenges**: Fine-Grained Categories

## Office chairs

Buy Office Chair Clearance, ...
ubuy.hk

Office Chairs – Kokuyo
kokuyo-furniture.com

LÅNGFJÄLL Office chair wit...
ikea.com · In stock

Ergonomic Chair - HOMELE...
homeless.hk

Of
wa

## Wooden chairs

BE STYLE Dining Chair wooden sea...
**HK$4,400.00**
outofstock.com.hk

Friendly chair DC340-1N Walnut...
**HK$4,000.00**
outofstock.com.hk

Wooden Chair - AGAIN Co...
againhk.com

Anders Wood Dining Chair O
decor8.com.hk · In stock

## Avocado chairs?

Images created by OpenAI DALLE

# **Challenges**: Chairs in Scene Context

Occlusion, non-canonical view, clutters

Chairs in my office

# **Challenges**: Domain Changes

Art clip


Drawing


Photo

# **Challenges**: Variation in Materials and Textures

Mold Chair by Anders Johnsson and Petter Thörne


The Shopping Cart Chair by Fernando Paullada

The Octopus Chair from the Animal Chair Collection by Máximo Riera

https://furniturefashion.com/well-thats-interesting-10-bizarre-awesome-chairs/

# Challenges: Functionality

Definition of a chair: anything people can sit?

Chairless chair

Chair as a weapon?

Chair as a transportation?

Handy Geng

# Challenges: Cross-class similarity

Chair

Sofa

Samples from chair and sofa categories
https://www.kaggle.com/c/day-3-kaggle-competition/overview

# Image Classification: Very Useful!

**Predictions**:

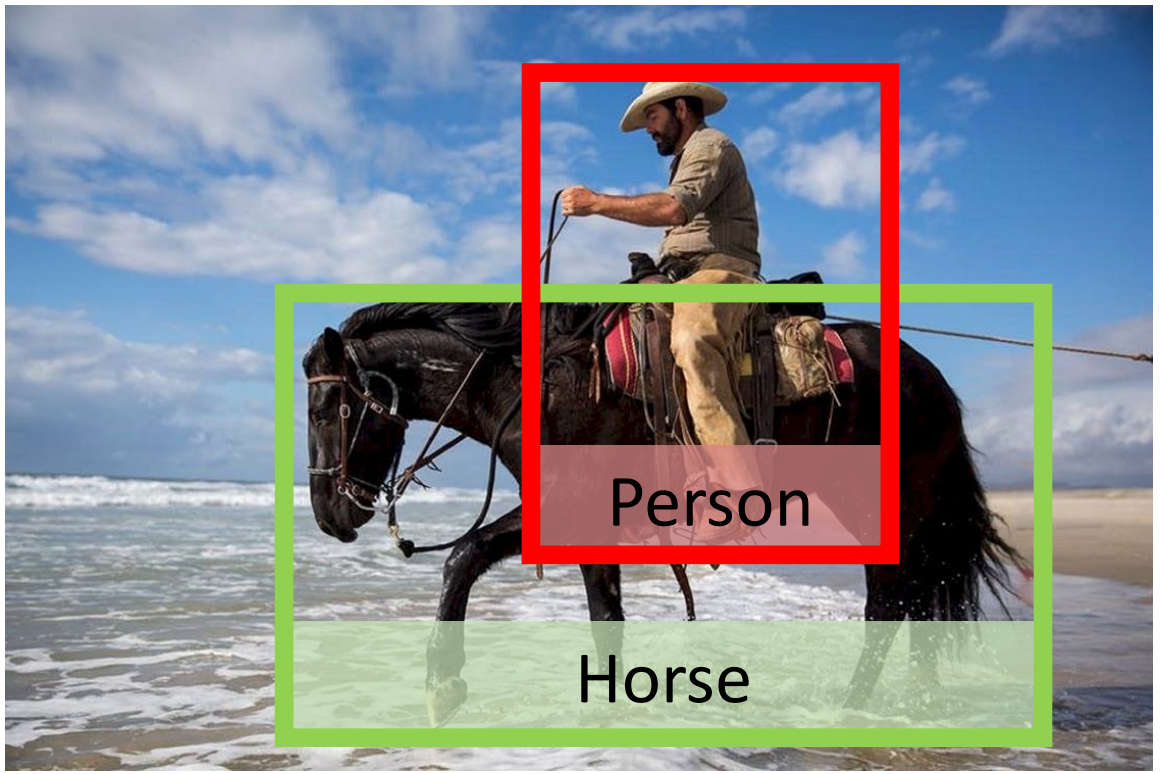
- **Type of environment:** outdoor
- **Scene categories:** playground (0.999)
- **Scene attributes:** man-made, natural light, open area, playing, sunny, plastic, touring, no horizon, dry

**Predictions**:

- **Type of environment:** indoor
- **Scene categories:** conference_center (0.506), conference_room (0.228), ballroom (0.166)
- **Scene attributes:** no horizon, enclosed area, indoor lighting, man-made congregating, wood, spectating, glossy, cloth

# Image Classification: Building Block for other tasks!

## Example: Object Detection



# Image Classification: Building Block for other tasks!

## Example: Object Detection



Background

Horse

Person

Car

Truck

# Image Classification: Building Block for other tasks!

## Example: Object Detection



→

Background
Horse
Person
Car
Truck

# Image Classification: Building Block for other tasks!

## Example: Image Captioning



riding
cat
horse
man
when
…
<STOP>

What word to say next?

Caption:
Man

# Image Classification: Building Block for other tasks!

## Example: Image Captioning



riding
cat
horse
man
when
…
<STOP>

What word to say next?

Caption:
Man riding

# Image Classification: Building Block for other tasks!

## Example: Image Captioning



riding
cat
horse
man
when
…
<STOP>

What word
to say next?

Caption:
Man riding horse

# Image Classification: Building Block for other tasks!

## Example: Image Captioning



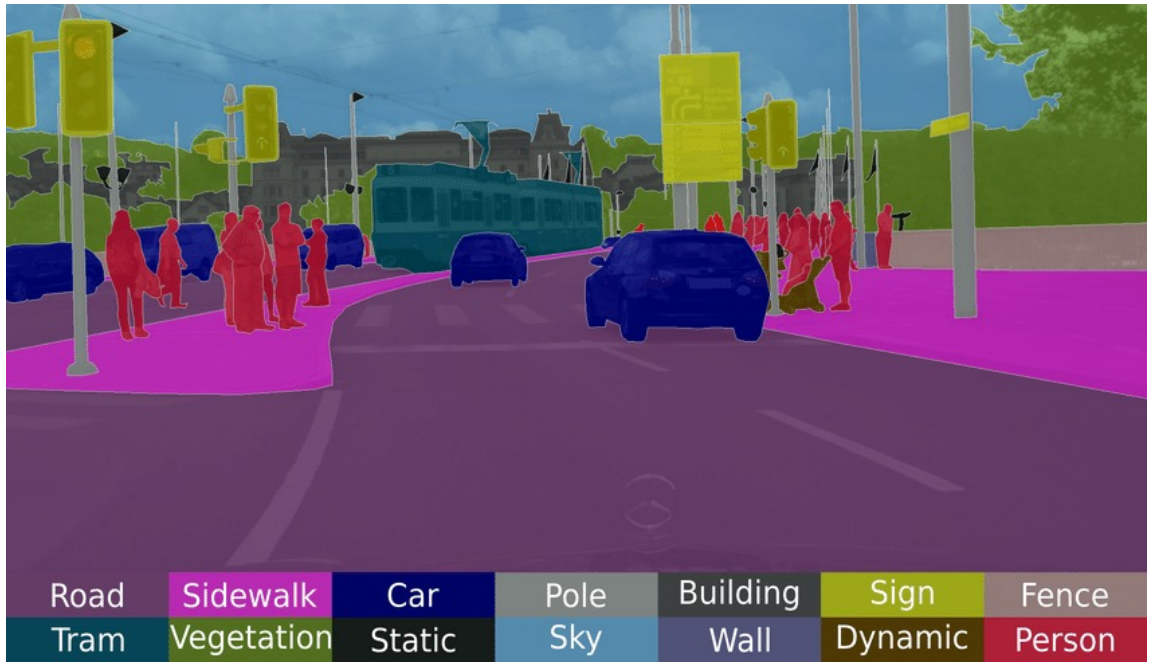
riding
cat
horse
man
when
…
<STOP>

What word
to say next?

Caption:
Man riding horse

# Image Classification: Building Block for other tasks!

Semantic segmentation (Pixel-wise classification)




https://www.youtube.com/watch?v=0yCYro9H1kM

# Image Classification: Building Block for other tasks!

## Example: Visual question answering

**Question: what is the sport**

**Predictions:**:

- **baseball** (score: 13.13 = 4.82 [image] + 8.32 [word])
- **soccer** (score: 9.08 = 0.13 [image] + 8.95 [word])
- **tennis** (score: 8.66 = 0.44 [image] + 8.22 [word])

**Based on image only**: pitcher (5.16), batting (4.95), baseball (4.82),
**Based on word only**: soccer (8.95), baseball (8.32), tennis (8.22),

**Question: what is the color of the pant**

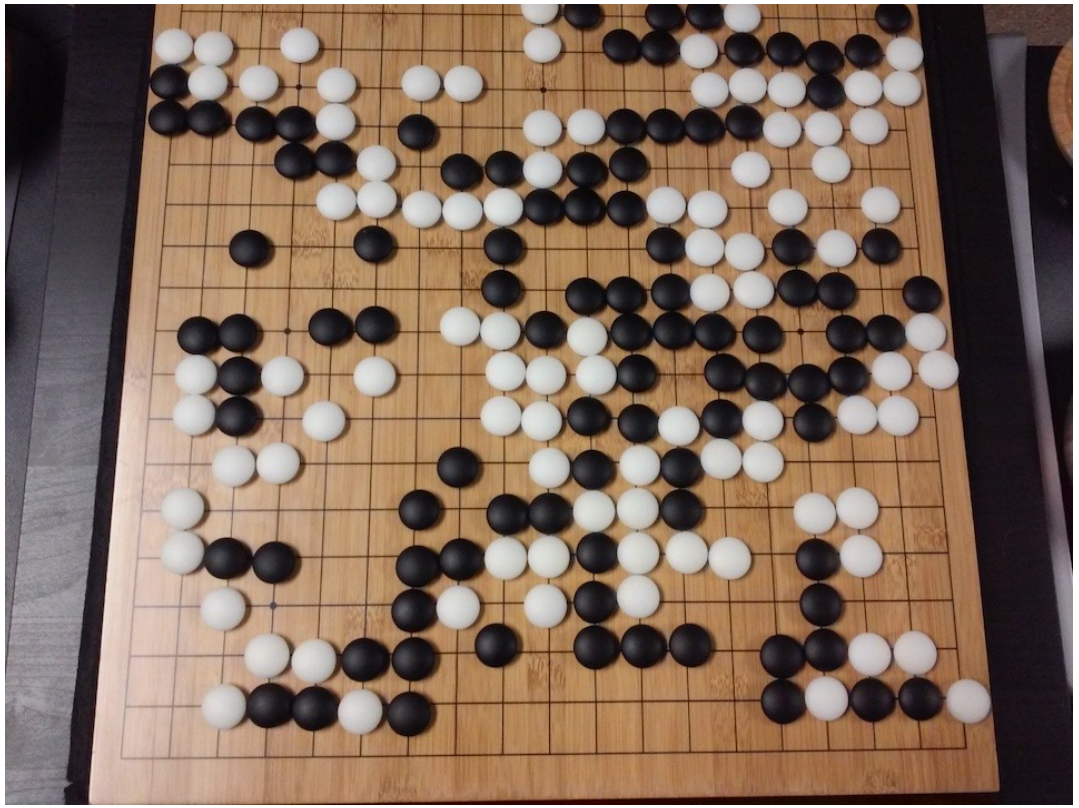
**Predictions:**:

- **white** (score: 14.06 = 2.77 [image] + 11.28 [word])
- **black** (score: 13.05 = 2.23 [image] + 10.82 [word])
- **green** (score: 12.18 = 1.40 [image] + 10.79 [word])

**Based on image only**: pitcher (5.16), batting (4.95), baseball (4.82),
**Based on word only**: white (11.28), brown (11.05), black (10.82),

# Image Classification: Building Block for other tasks!

Example: Playing Go



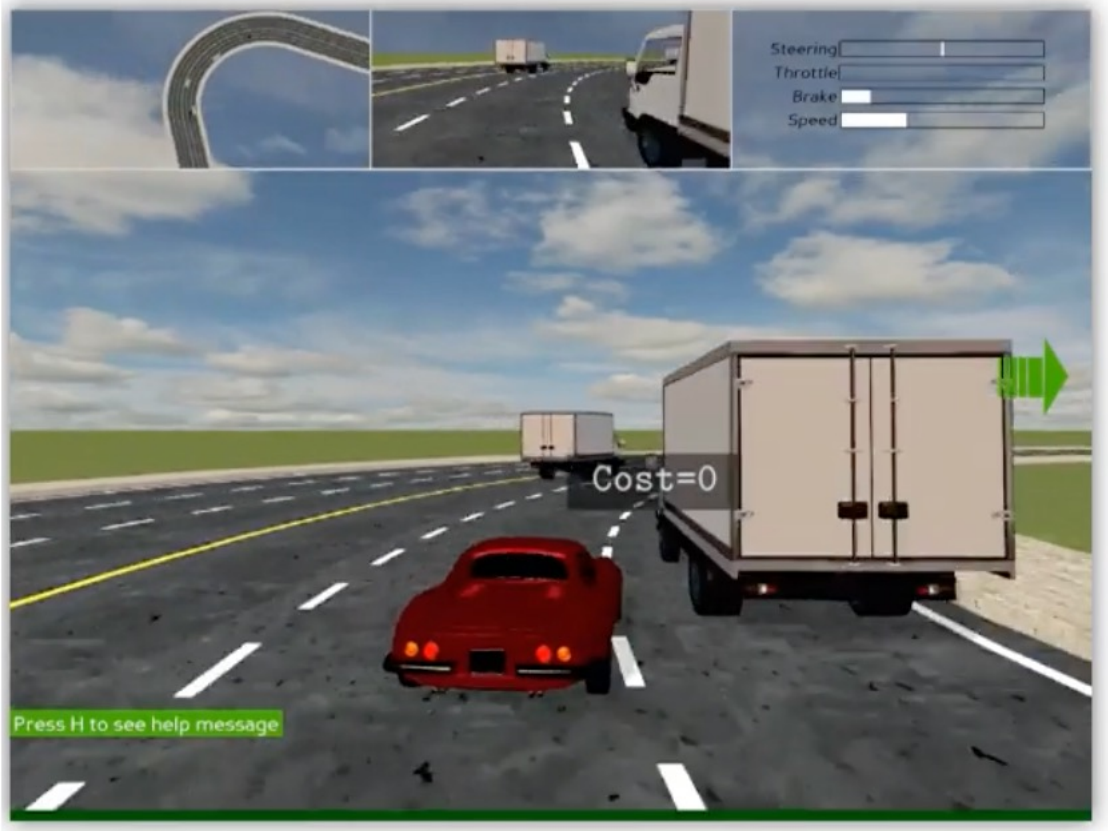
This image is CC0 public domain

(1, 1)
(1, 2)
…
(1, 19)
…
(19, 19)

Where to play next?

# Image Classification: Building Block for other tasks!

Example: Decision making in driving scene




Turn left

Keep in lane

Turn right

# An Image Classifier

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

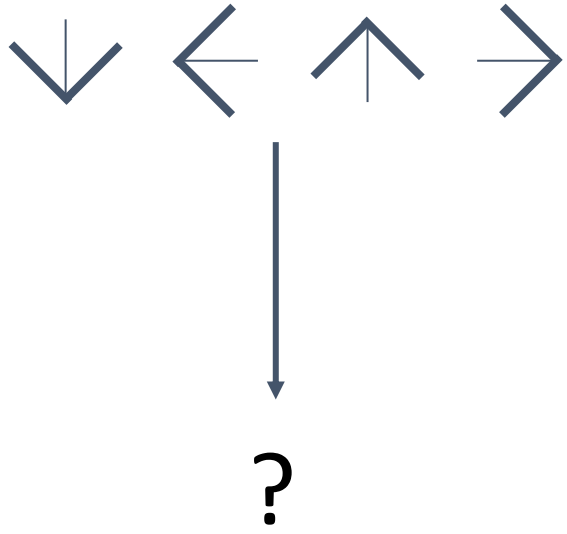
Unlike e.g. sorting a list of numbers,

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

# You could try …

Find edges

Find corners



?

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

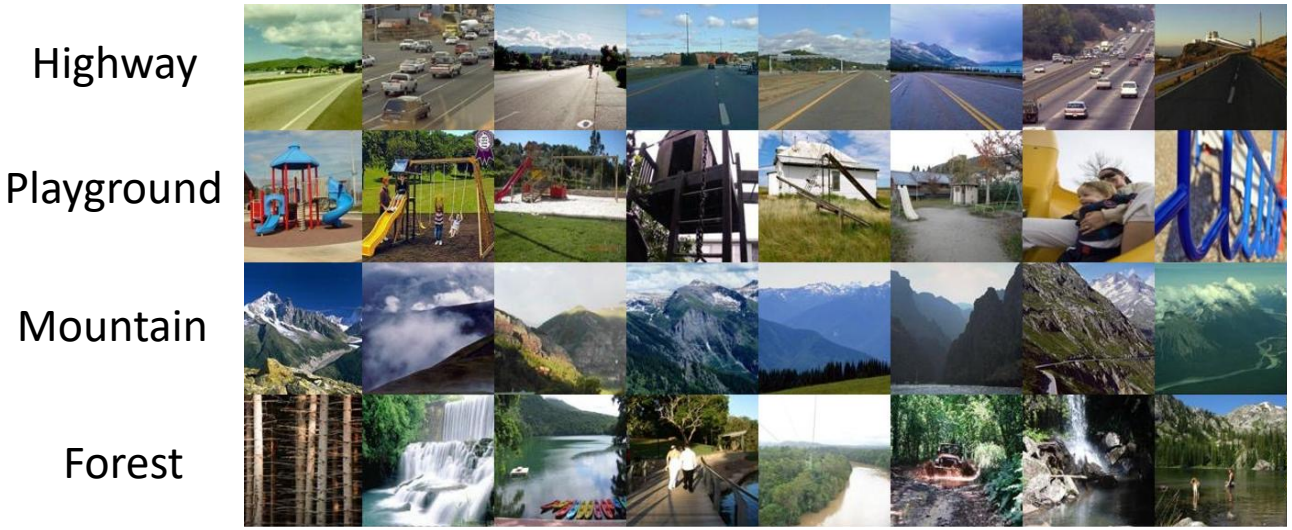
# Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
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
```

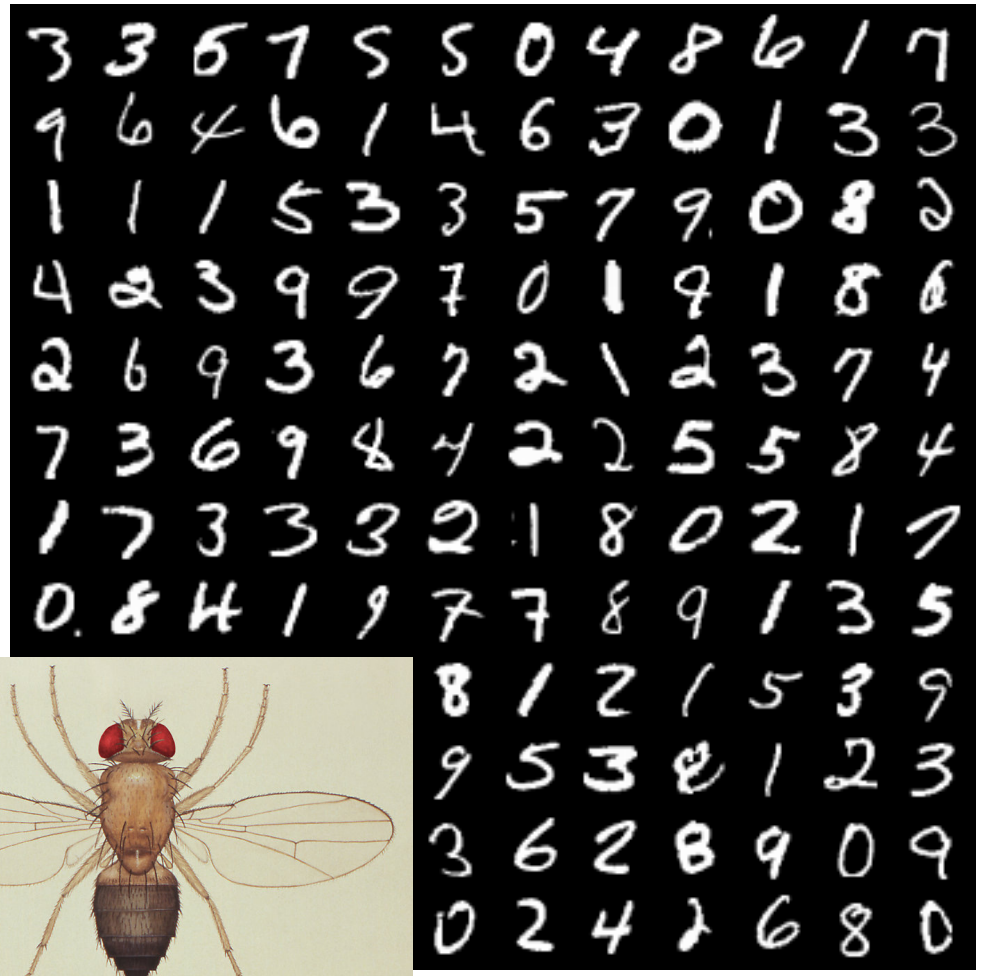
**Example training set**



# Image Classification Datasets: MNIST

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

# Image Classification Datasets: MNIST



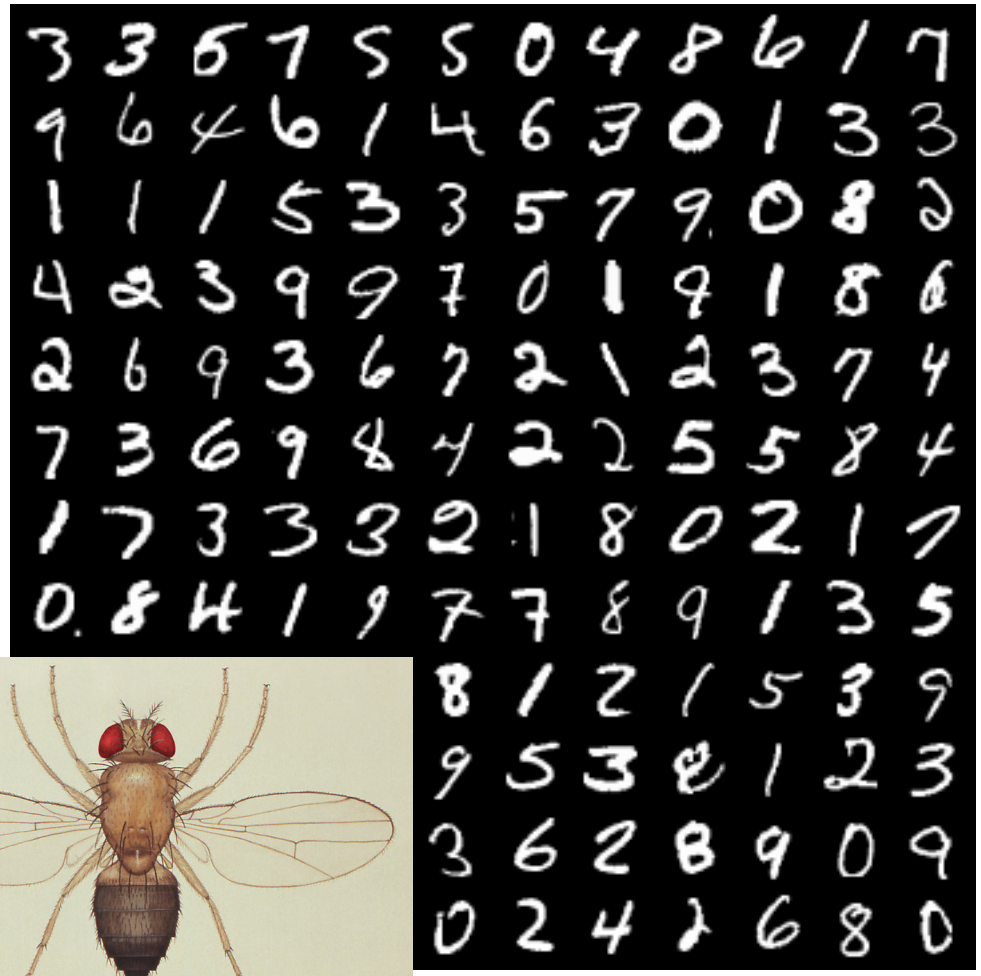


**10 classes**: Digits 0 to 9
**28x28** grayscale images
**50k** training images
**10k** test images

"Drosophila of computer vision"

"Results from MNIST often do not hold on more complex datasets!"
(10 years ago when ImageNet bloomed)

# Image Classification Datasets: MNIST





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

"Drosophila of computer vision"

"Results from MNIST often do not hold on more complex datasets!"
(10 years ago when ImageNet bloomed)

Now ImageNet becomes the next MNIST

# Image Classification Datasets: CIFAR10


**10** classes

**50k** training images (5k per class)

**10k** testing images (1k per class)

**32x32 RGB** images

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

# Image Classification Datasets: places


**Zhou** et al, "Places: A 10 million Image Database for Scene Recognition", TPAMI 2017

**365 classes** of different scene types
Standard set: **1.8 M** training images
Challenge set: **~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

http://places2.csail.mit.edu/

# Image Classification Datasets: places

- The construction pipeline for Places Database

1. Collect scene names from a dictionary

~1000 scene names

2. Query and download images

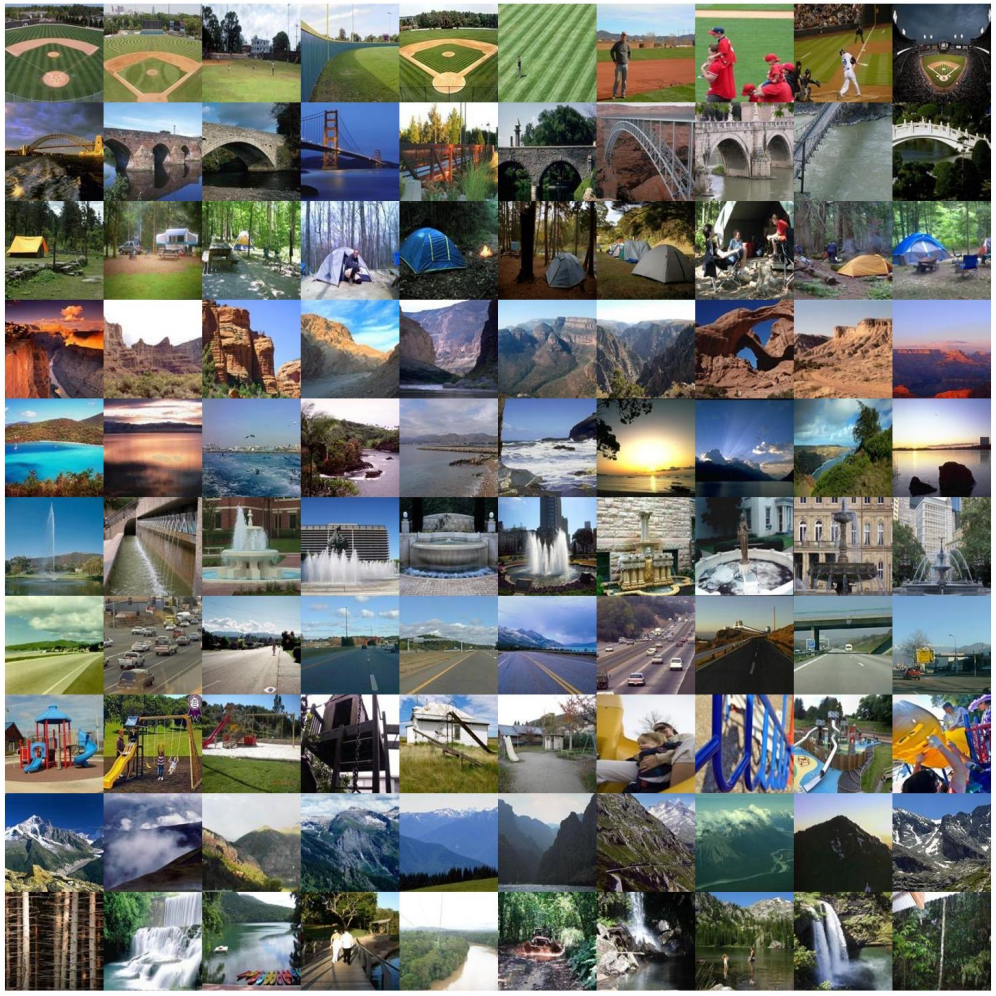

696 adjectives + scene names
~ 60 million raw images downloaded

3. Annotate through Amazon Mechanical Turk


Three rounds of annotations

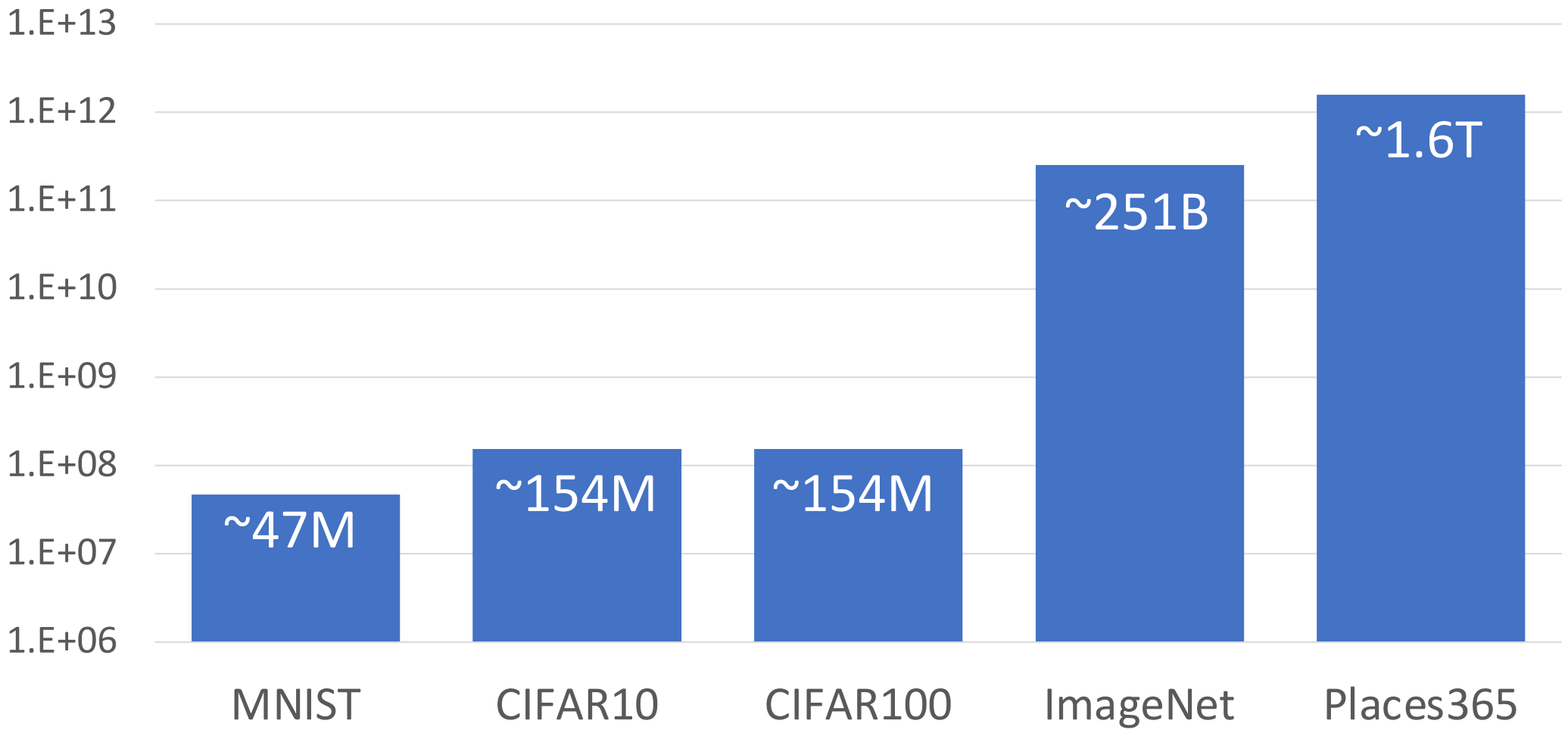
# TinyPlaces for Assignment 1 and Assignment 2

baseball_field, bridge, campsite, canyon, coast, fountain, highway, playground, mountain, rainforest



bathroom, bedroom, bookstore, classroom, dining_room, food_court, kitchen, lobby, living_room, office

# Classification Datasets: Number of Training Pixels



# Other Image Classification Datasets: Omniglot

**1623 categories**: characters from 50 different alphabets

**20 images per category**

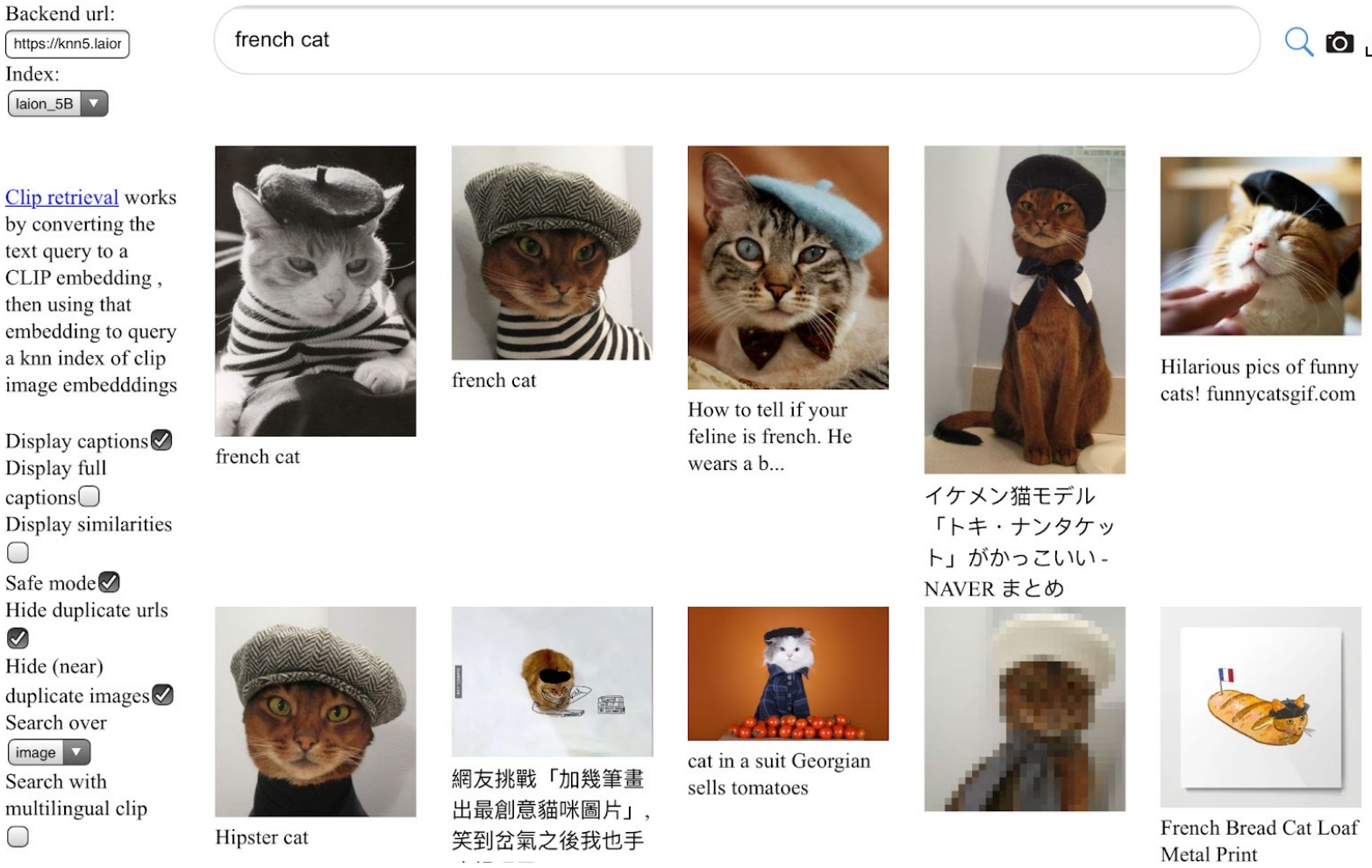
Meant to test **few shot learning**

Lake et al, “Human-level concept learning through probabilistic program induction”, Science, 2015

# More recent image datasets: LAION-5B

- https://laion.ai/blog/laion-5b/

But be cautious about your training data!!!




**Office of the Vice Chancellor for Research & Creative Activities**

**Information Technology Services**

**To: Academic Employees**

**What you need to know:**

- Links to thousands of child sexual abuse material (“CSAM”) have been found in the LAION-5B machine learning dataset frequently used by the AI research community for training AI models.
- Anyone who downloaded copies of images from links in this dataset, or any other LAION dataset, may have inadvertently downloaded illegal material.
- Do not attempt to delete, modify, or use any LAION dataset if you have already downloaded it. Instead, please contact Lisa Snyder at LMS@ucla.edu by **Jan. 19** to discuss steps to determine if any CSAM was downloaded, and if so, to safely and securely expunge any CSAM.

# 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

# Distance Metric to compare images

**L1 distance:**

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

| test image | | | | | training image | | | | | pixel-wise absolute value differences | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 56 | 32 | 10 | 18 | - | 10 | 20 | 24 | 17 | = | 46 | 12 | 14 | 1 | add → 456 |
| 90 | 23 | 128 | 133 | | 8 | 10 | 89 | 100 | | 82 | 13 | 39 | 33 | |
| 24 | 26 | 178 | 200 | | 12 | 16 | 178 | 170 | | 12 | 10 | 0 | 30 | |
| 2 | 0 | 255 | 220 | | 4 | 32 | 233 | 112 | | 2 | 32 | 22 | 108 | |

# 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 nearest training image

Return 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 is training?

```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 is training?

**A**: O(1)

```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 is training?

**A**: O(1)

**Q**: With N examples, how fast is testing?

```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 is training?

**A**: O(1)

**Q**: With N examples, how fast is testing?

**A**: O(N)

```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 is training?

**A**: O(1)

**Q**: With N examples, how fast is testing?

**A**: O(N)

This is **bad**: We can afford slow training, but we need fast testing!

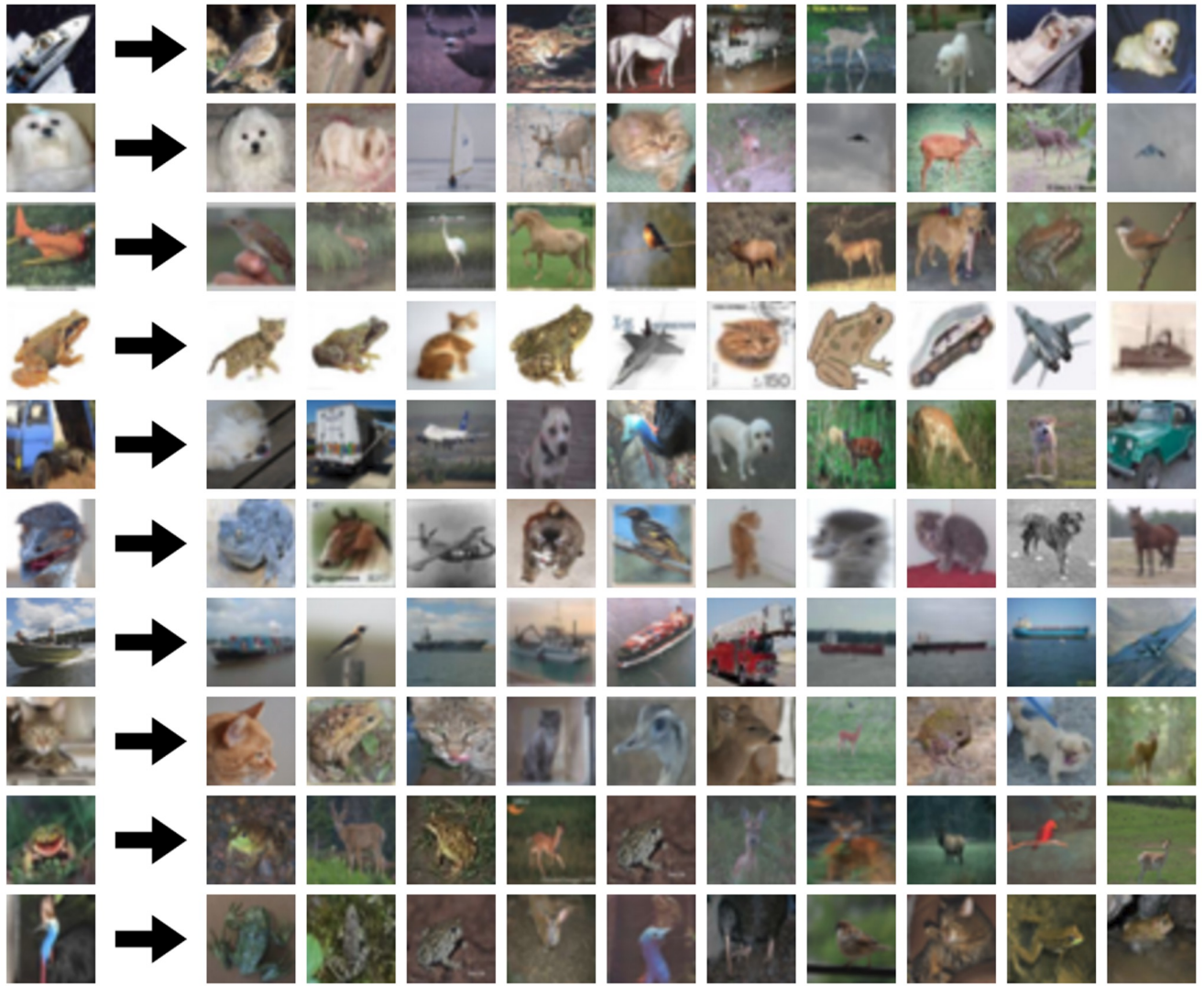
```python
import numpy as np

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

There are many methods for fast / approximate nearest neighbors; e.g. see

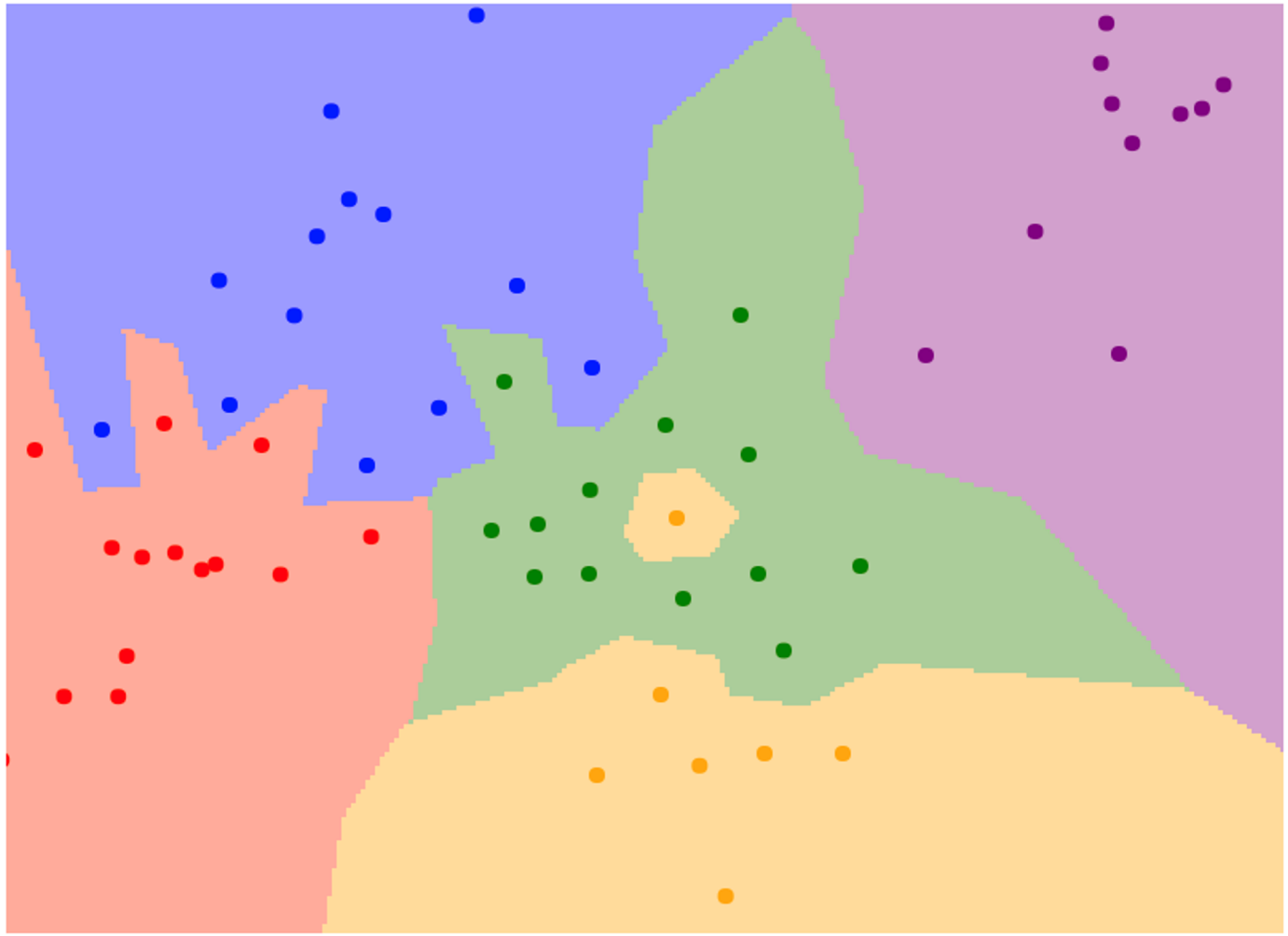
https://github.com/facebookresearch/faiss

# What does this look like?



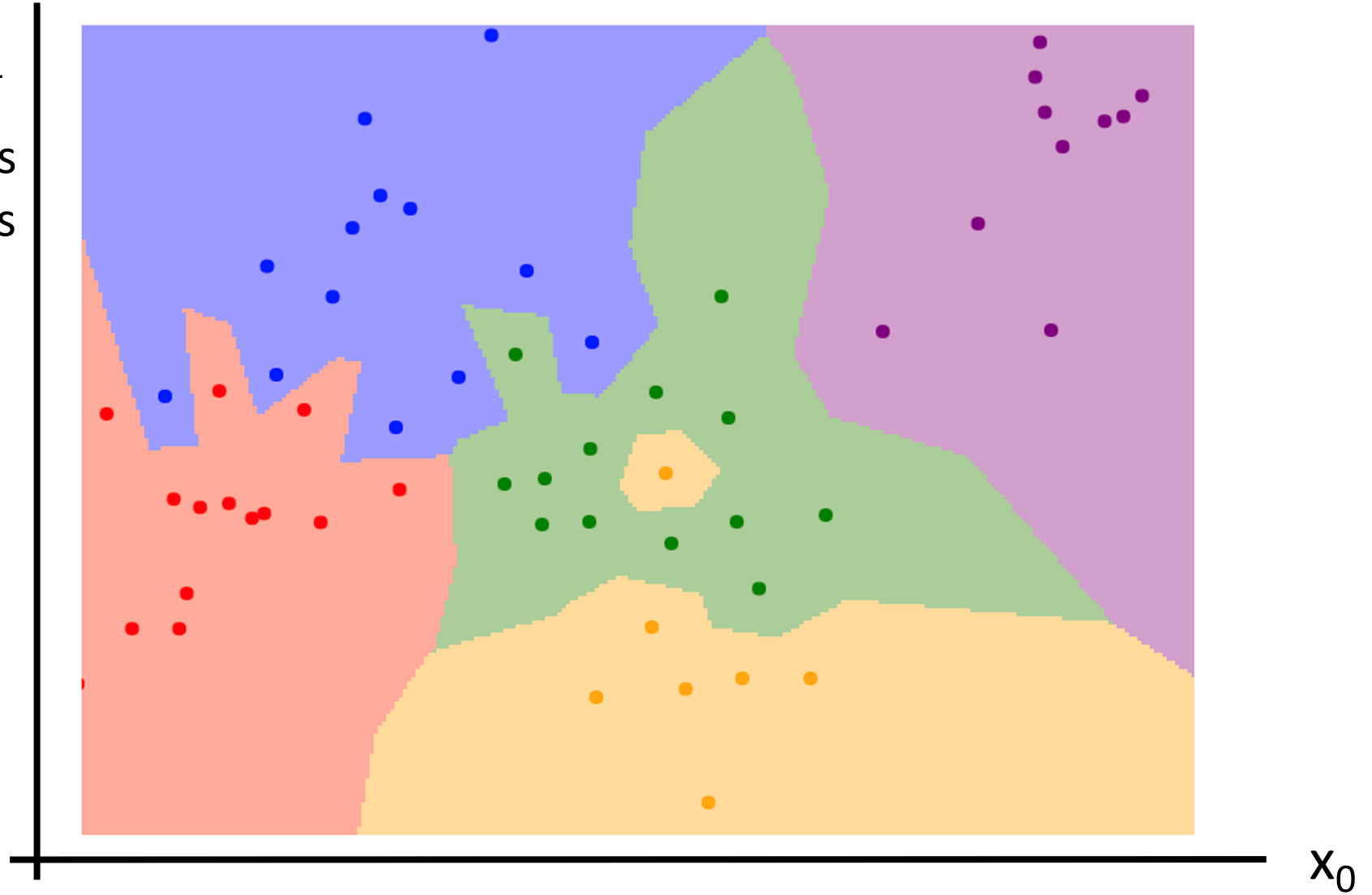
# What does this look like?

# Nearest Neighbor **Decision Boundaries**



# Nearest Neighbor **Decision Boundaries**

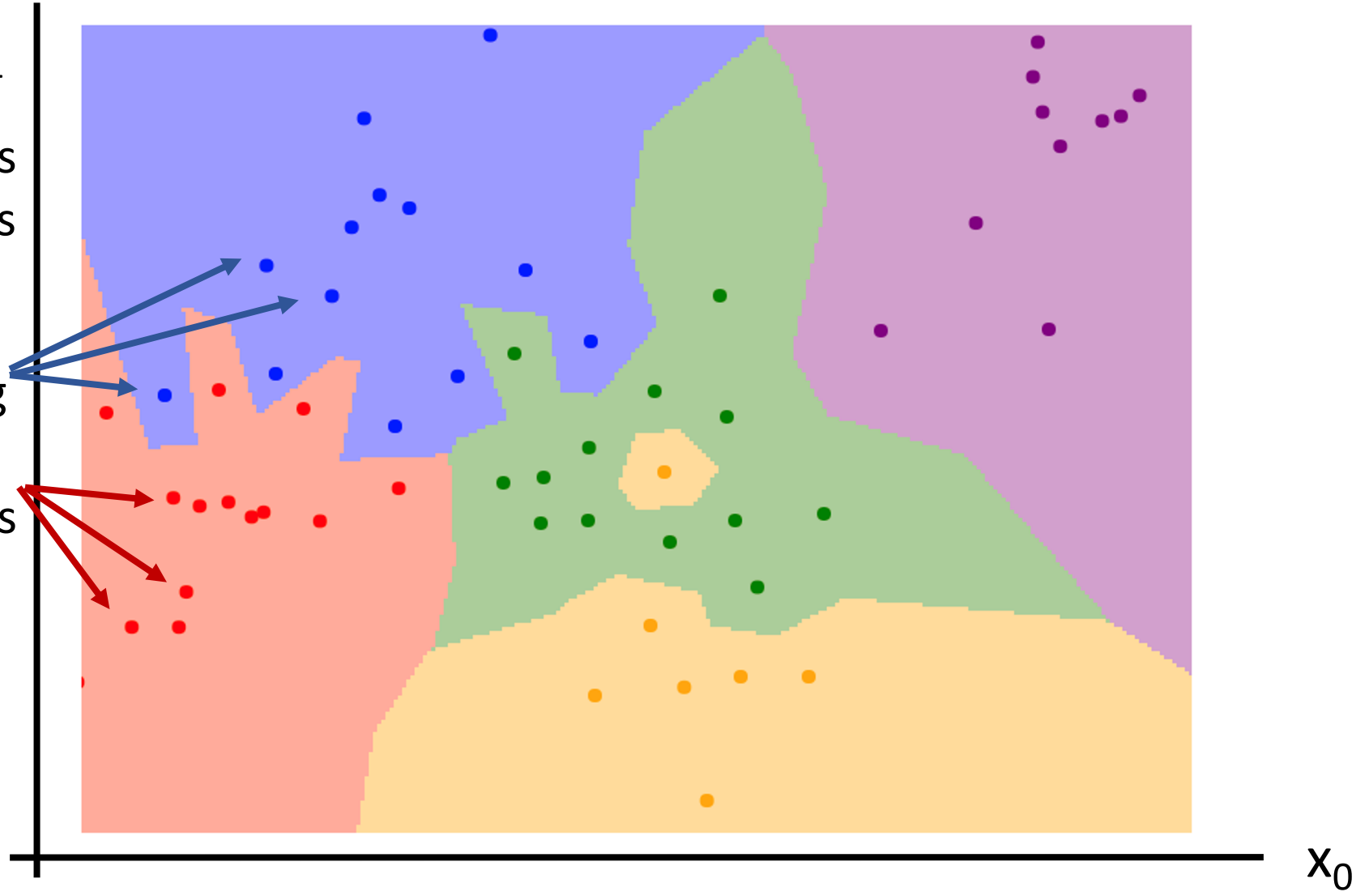
Nearest neighbors in two dimensions



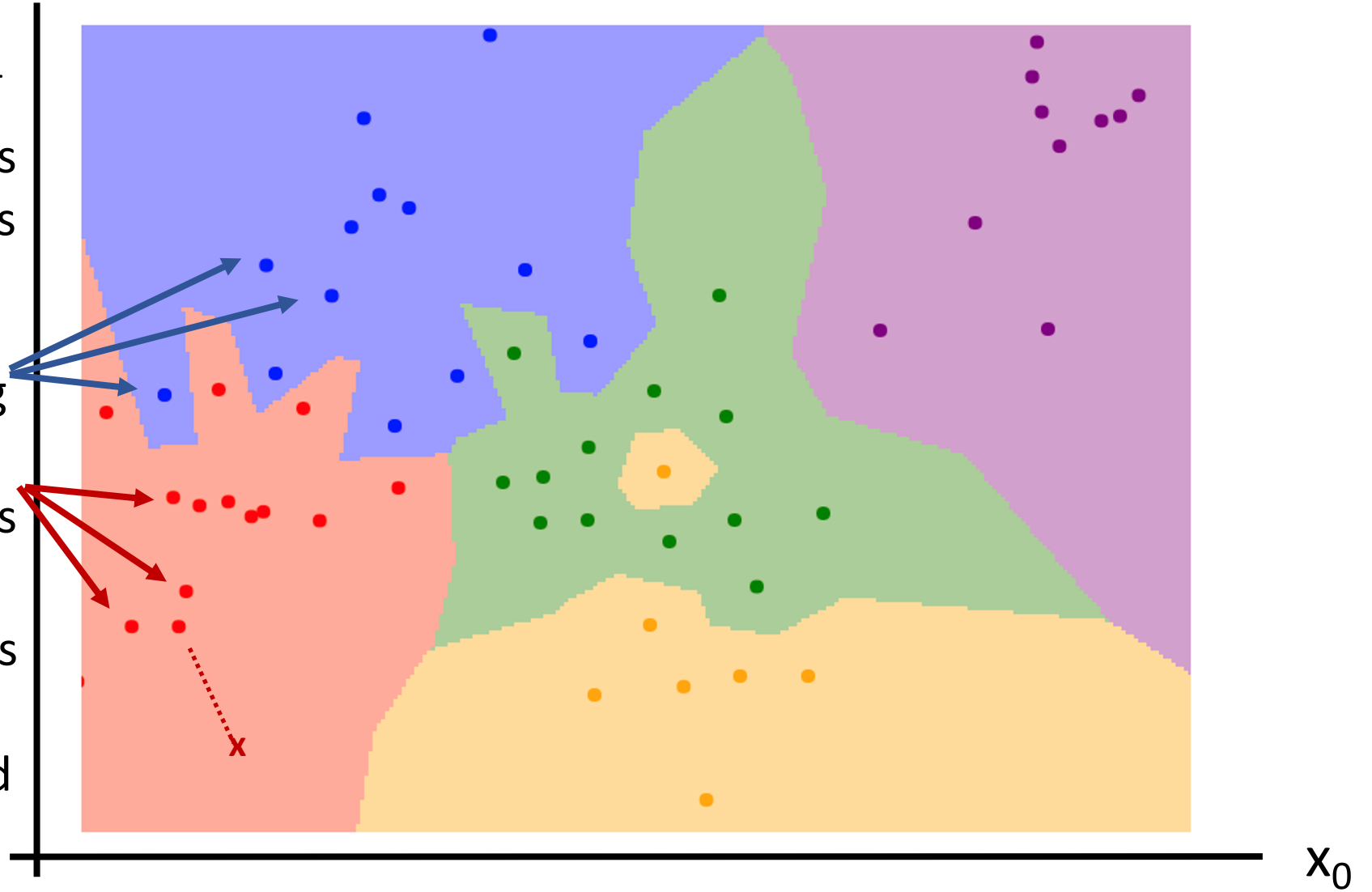
# Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels



# Nearest Neighbor **Decision Boundaries**

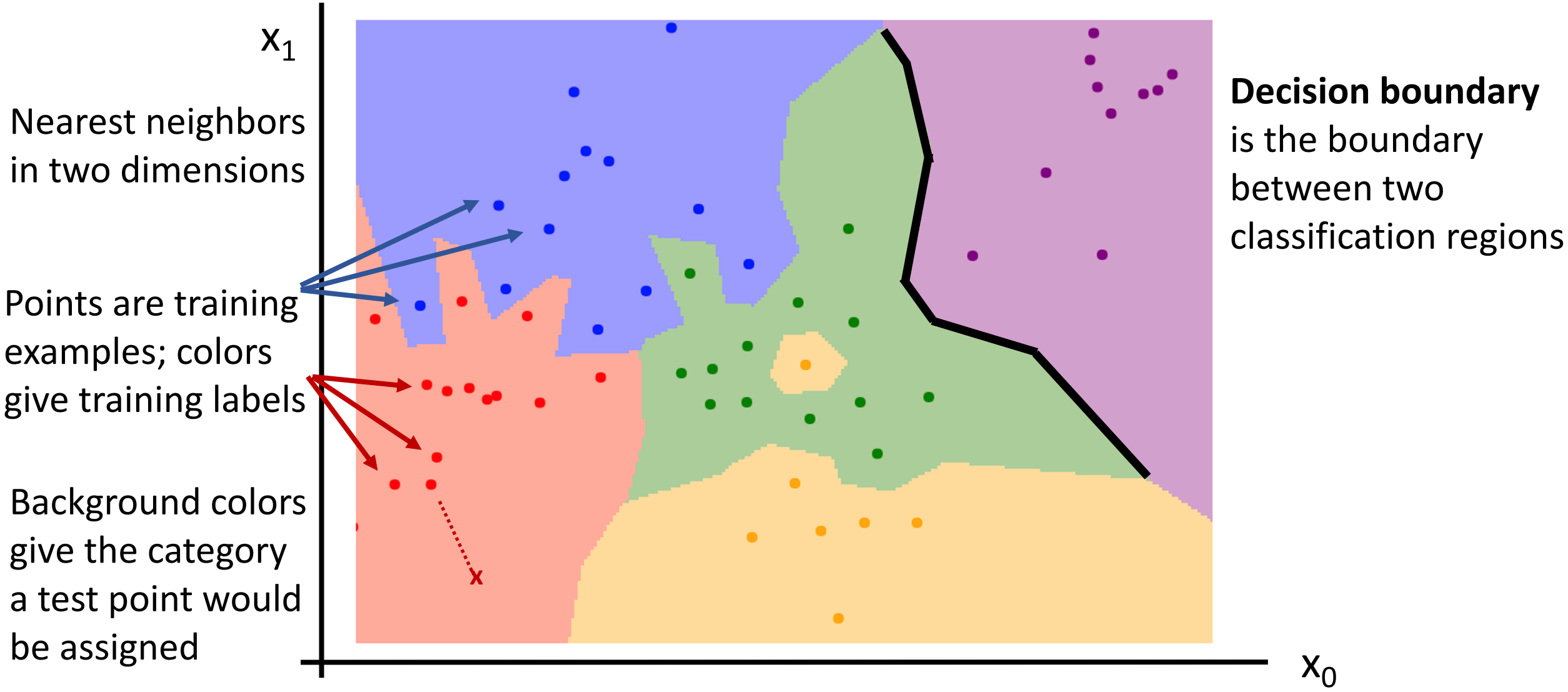



Nearest neighbors in two dimensions

Points are training examples; colors give training labels

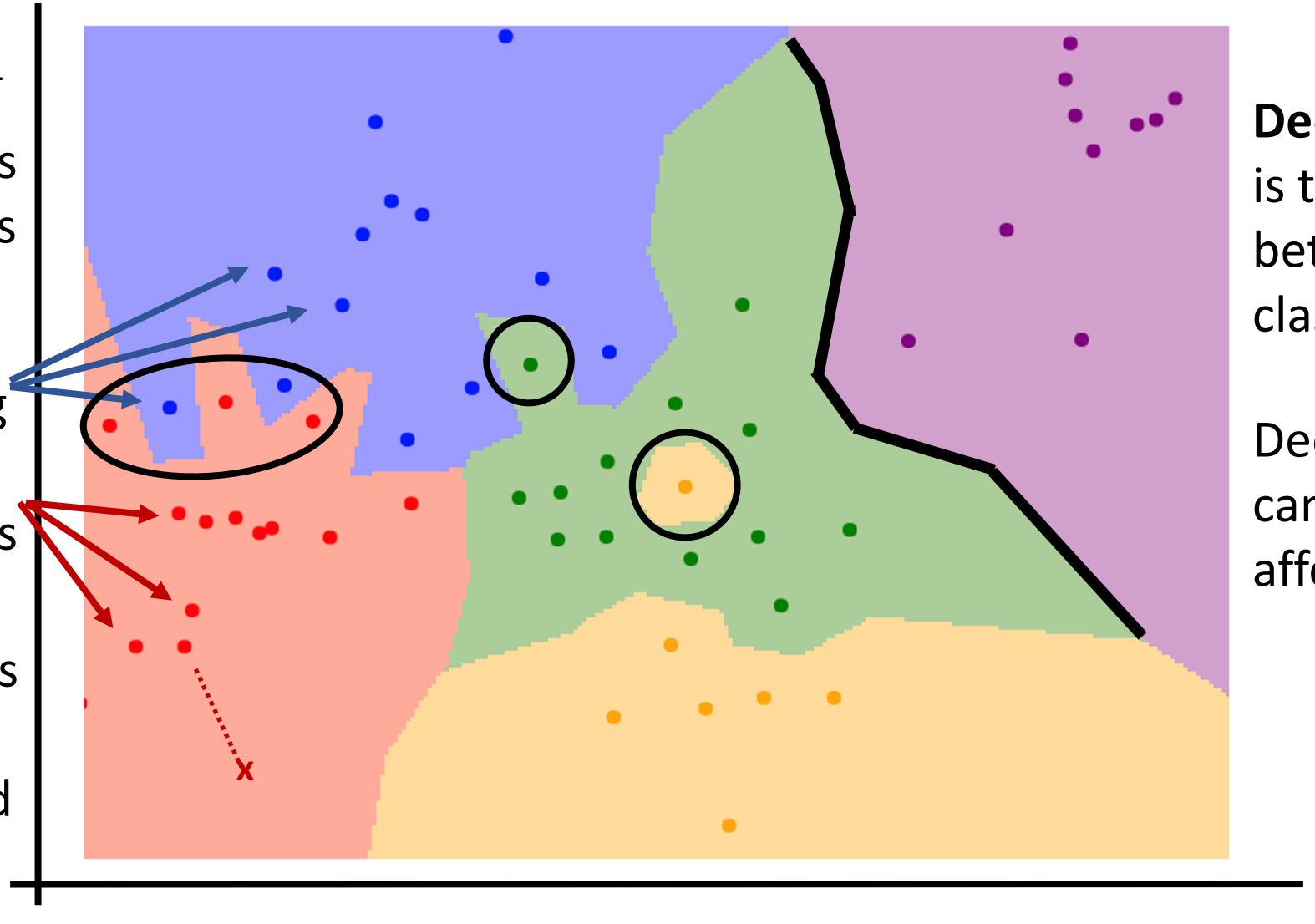
Background colors give the category a test point would be assigned

# Nearest Neighbor **Decision Boundaries**




**Decision boundary** is the boundary between two classification regions

# Nearest Neighbor **Decision Boundaries**




Nearest neighbors in two dimensions

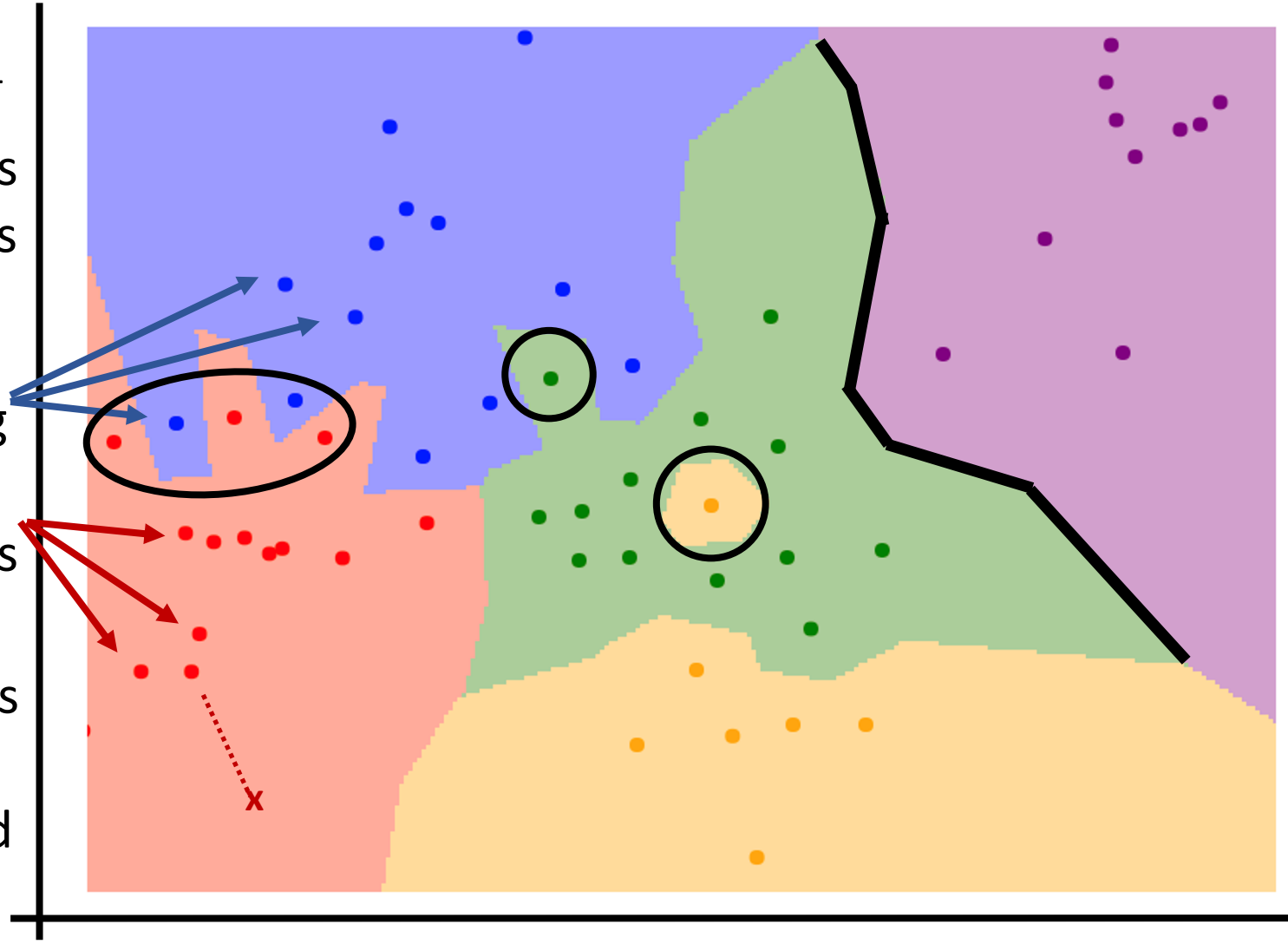
Points are training examples; colors give training labels

Background colors give the category a test point would be assigned

**Decision boundary** is the boundary between two classification regions

Decision boundaries can be noisy; affected by outliers

# Nearest Neighbor **Decision Boundaries**




Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned

**Decision boundary** is the boundary between two classification regions

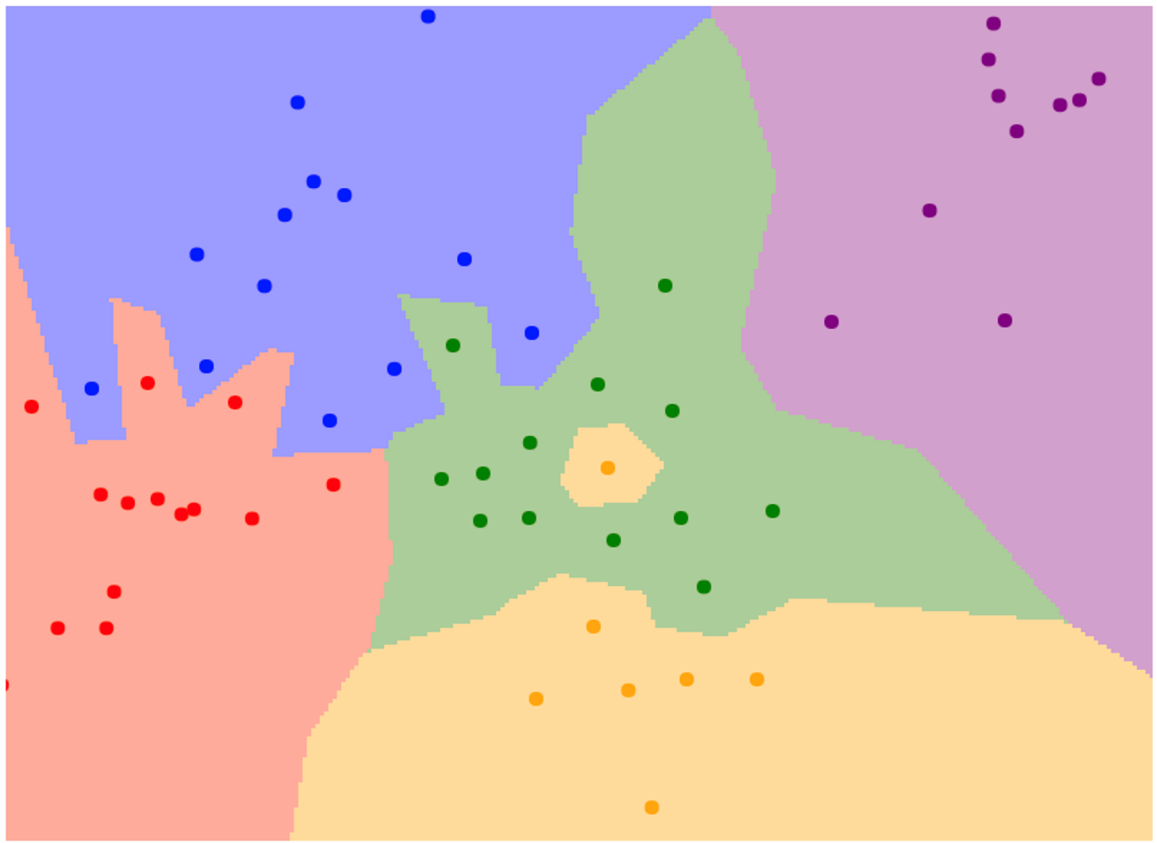
Decision boundaries can be noisy; affected by outliers

How to smooth out decision boundaries? Use more neighbors!

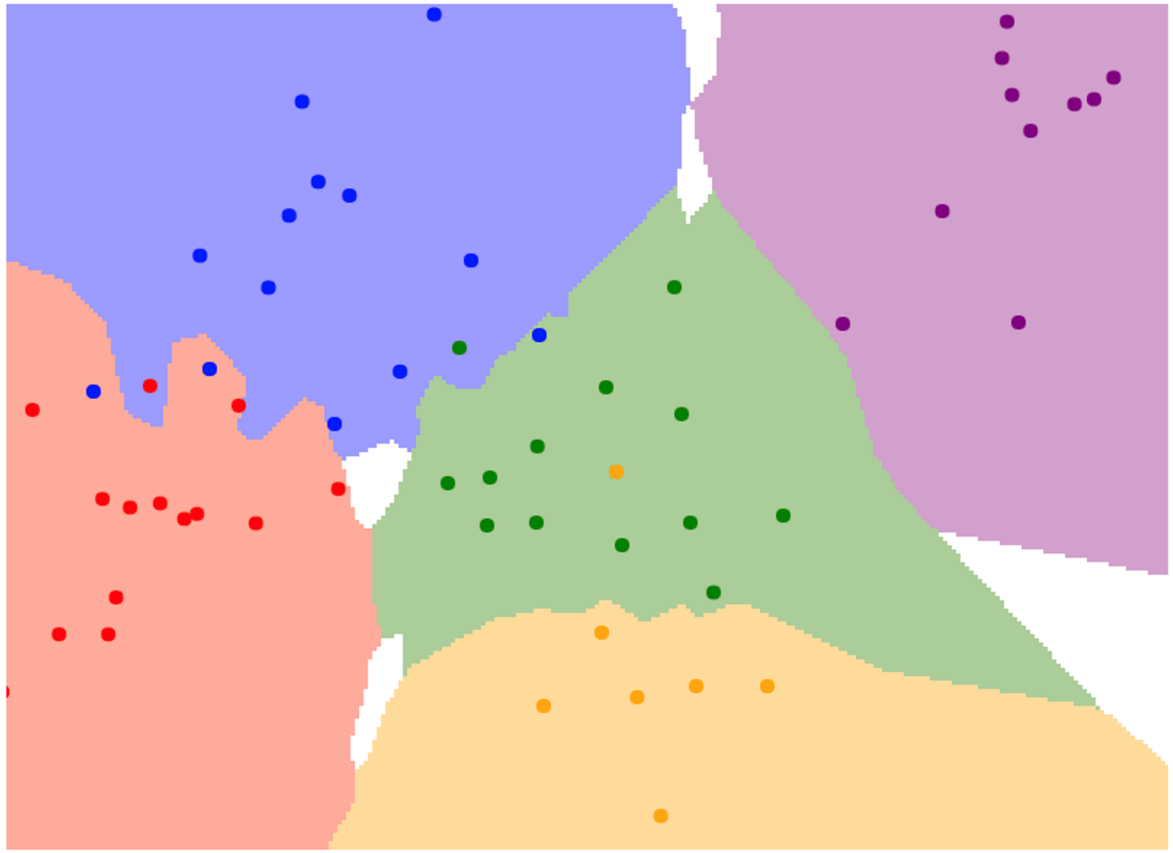
# K-Nearest Neighbors

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

K = 1



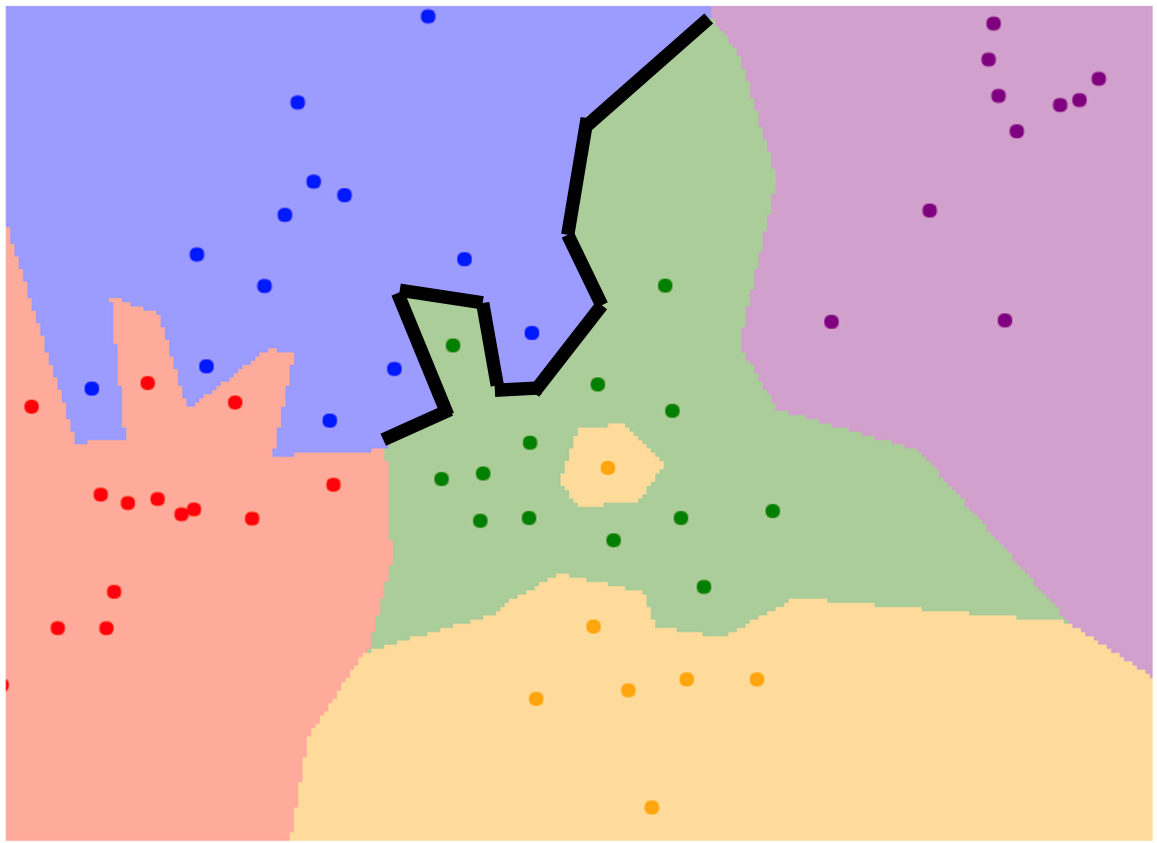
K = 3



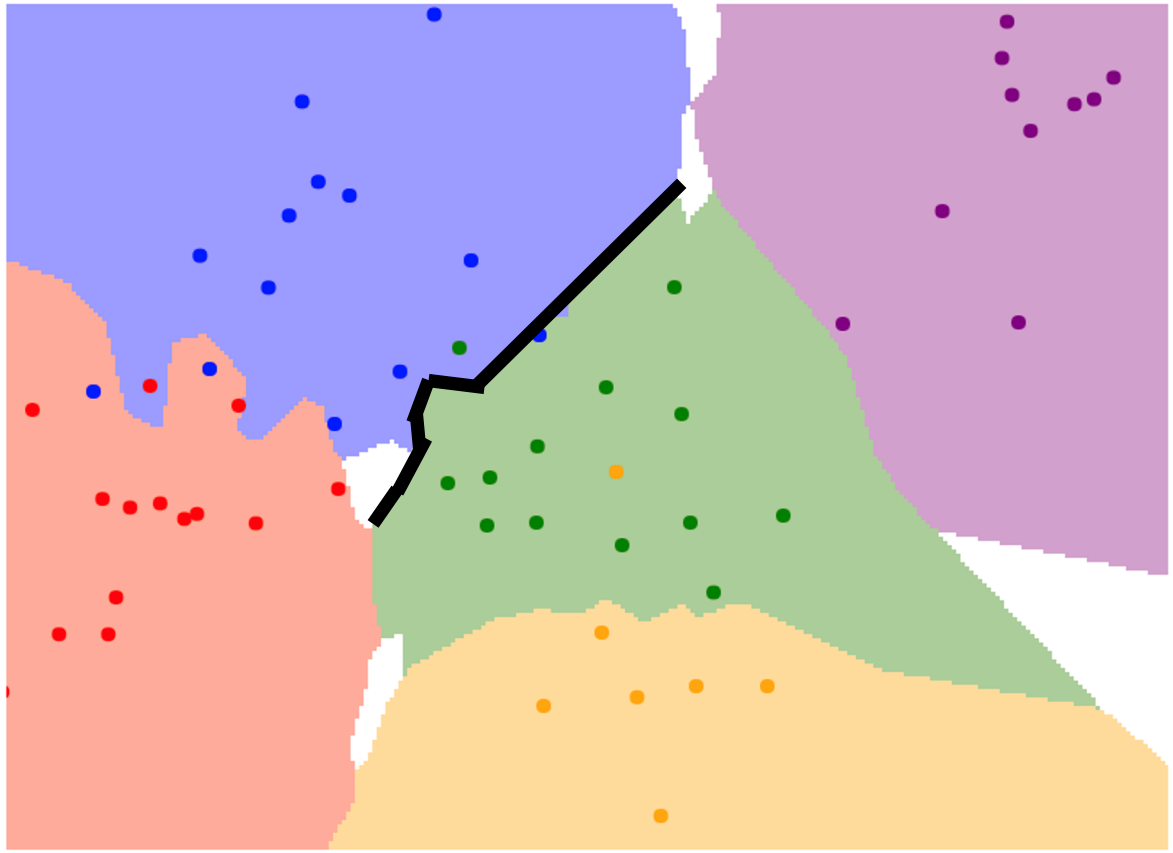
# K-Nearest Neighbors

Using more neighbors helps smooth out rough decision boundaries

K = 1



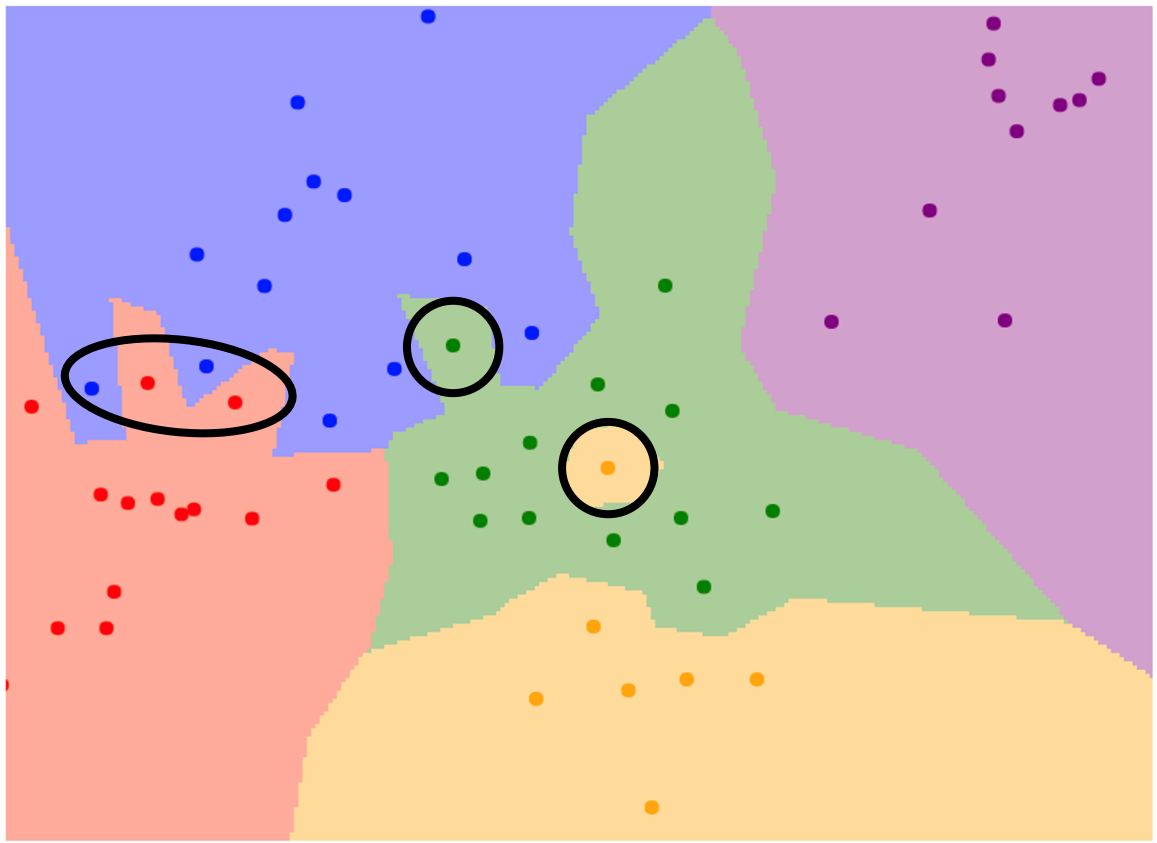
K = 3



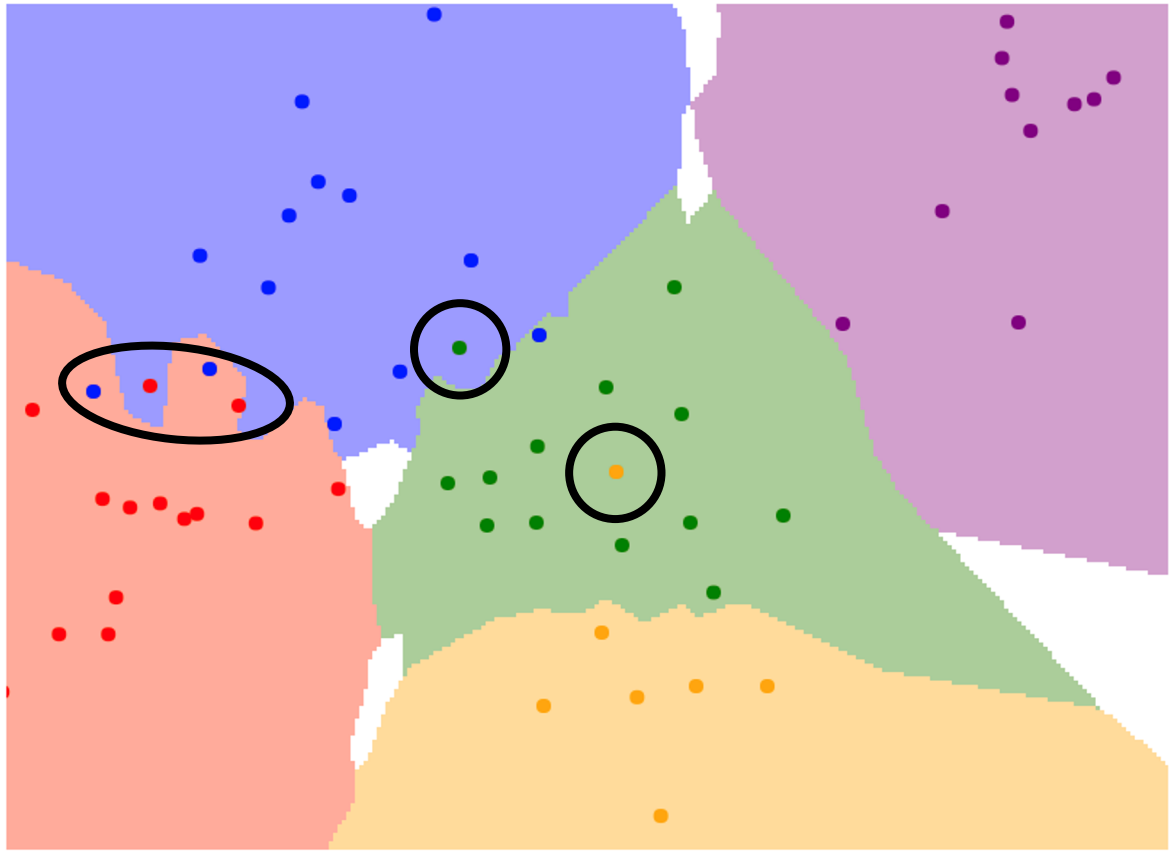
# K-Nearest Neighbors

Using more neighbors helps reduce the effect of outliers

K = 1



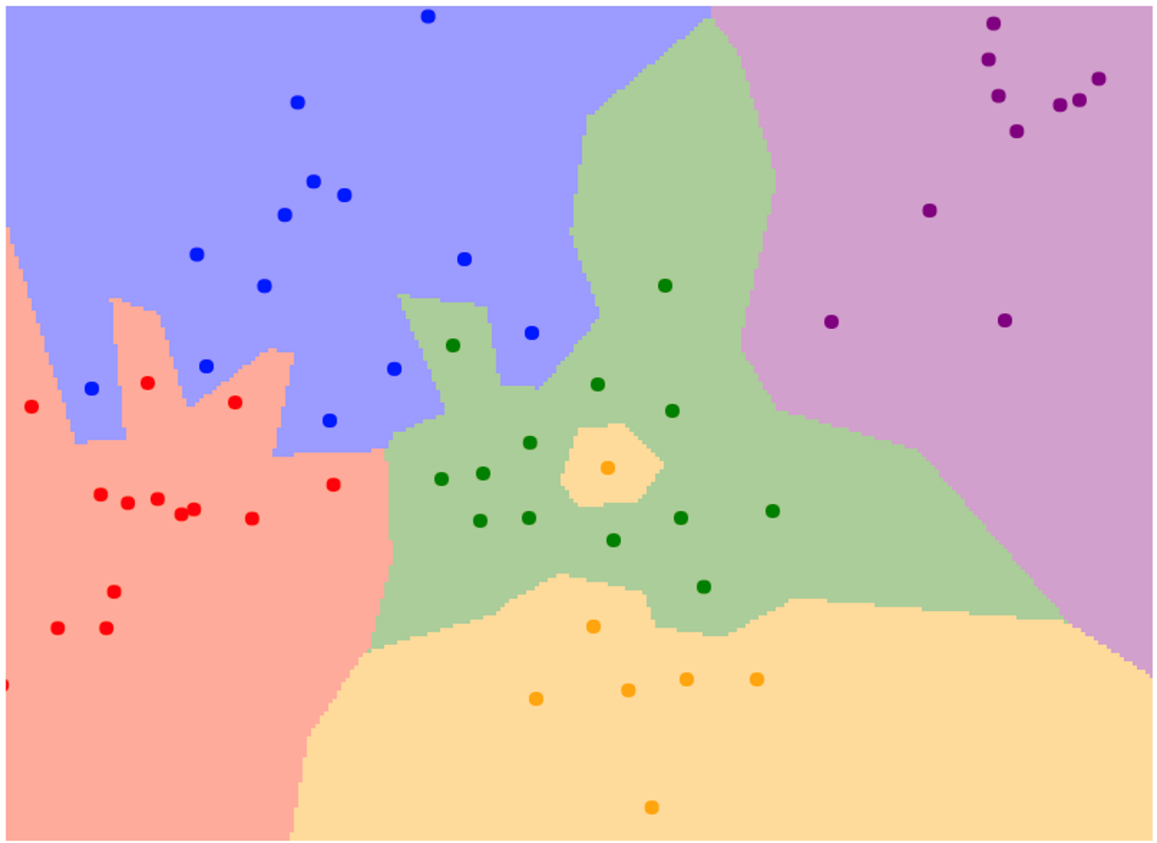
K = 3



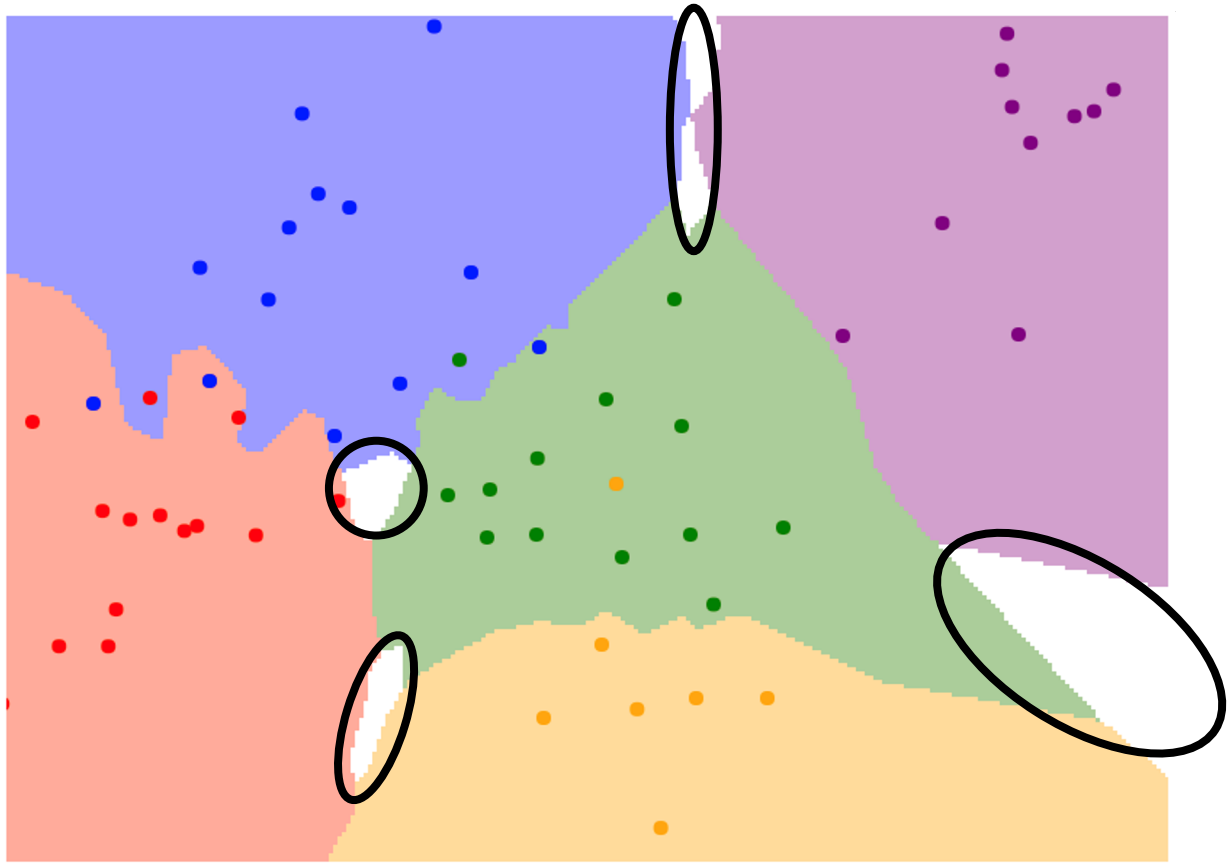
# K-Nearest Neighbors

When K > 1 there can be ties between classes.
Need to break somehow!

K = 1



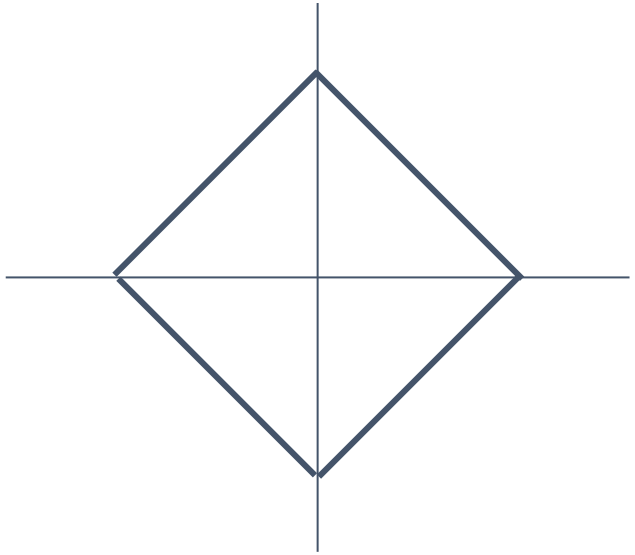
K = 3



# K-Nearest Neighbors: Distance Metric

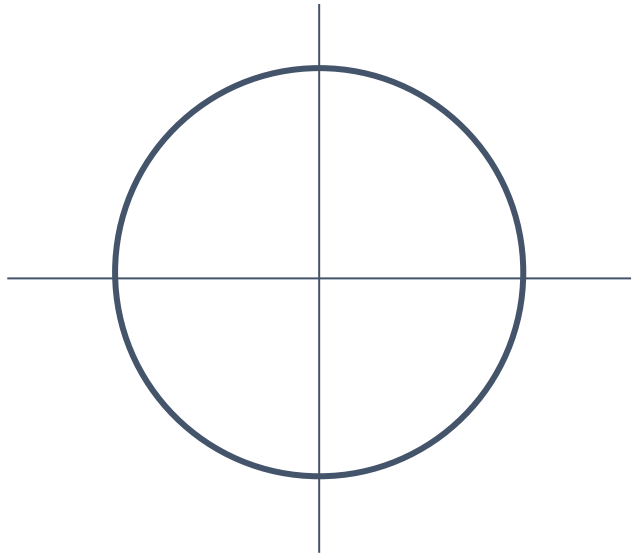
L1 (Manhattan) distance

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



L2 (Euclidean) distance

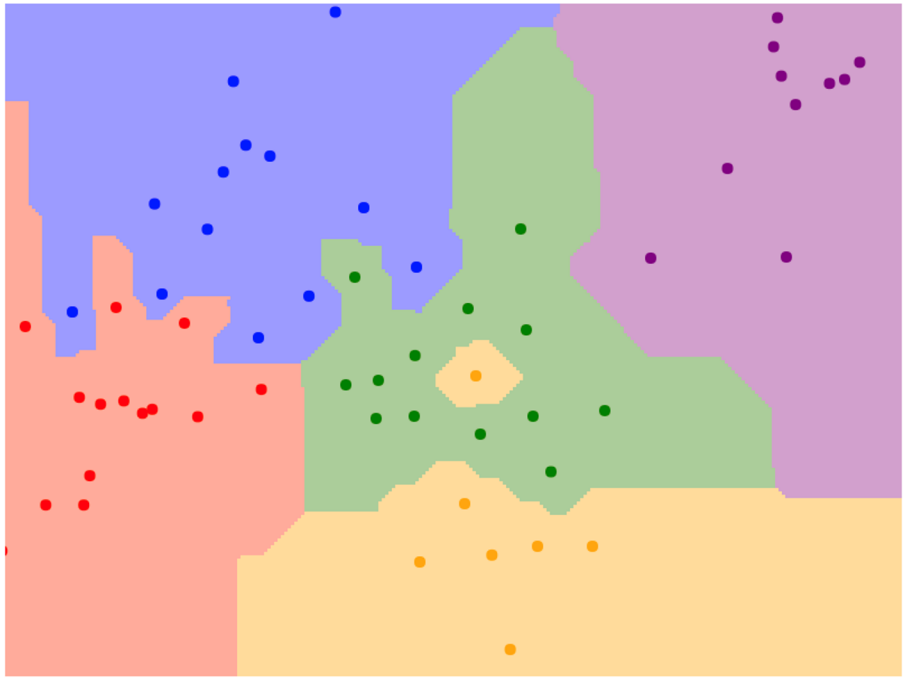
$$d_1(I_1, I_2) = \left( \sum_p \left(I_1^p - I_2^p\right)^2 \right)^{\frac{1}{2}}$$



# K-Nearest Neighbors: Distance Metric

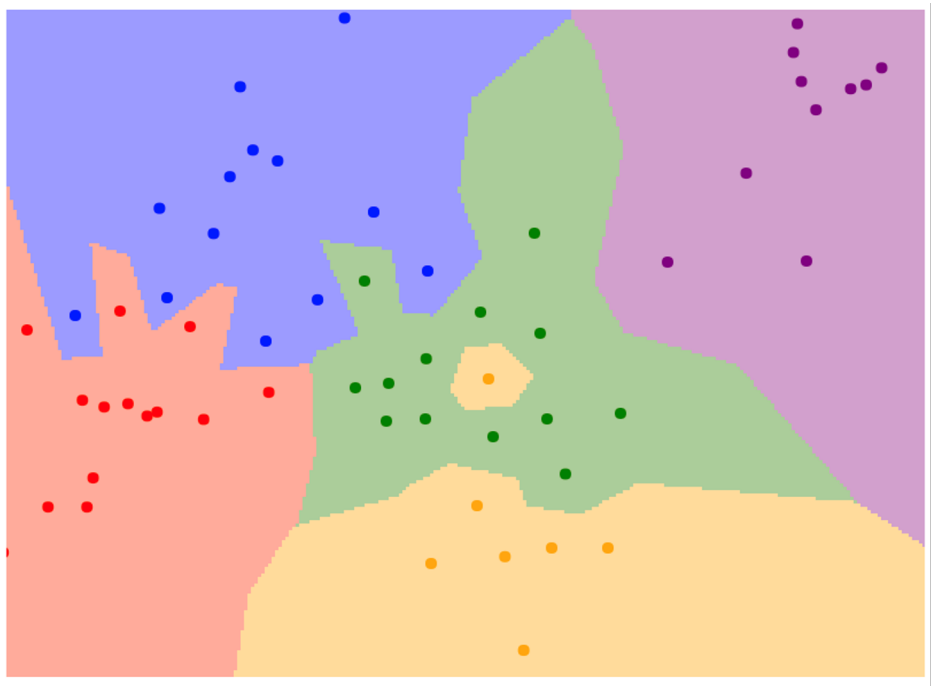
L1 (Manhattan) distance

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



L2 (Euclidean) distance

$$d_1(I_1, I_2) = \left( \sum_p \left(I_1^p - I_2^p\right)^2 \right)^{\frac{1}{2}}$$



K = 1

# K-Nearest Neighbors: Distance Metric

With the right choice of distance metric, we can apply K-Nearest Neighbor to any type of data!

# K-Nearest Neighbors: **Distance Metric**

With the right choice of distance metric, we can apply K-Nearest Neighbor to any type of data!

Example: Compare research papers using tf-idf similarity

**Mesh R-CNN**
Georgia Gkioxari, Jitendra Malik, Justin Johnson
6/6/2019 cs.CV

[illegible]02739v1 pdf
show similar | discuss

Rapid advances in 2D perception have led to systems that accurately detect objects in real-world images. However, these systems make predictions in 2D, ignoring the 3D structure of the world. Concurrently, advances in 3D shape prediction have mostly focused on synthetic benchmarks and isolated objects. We unify advances in these two areas. We propose a system that detects objects in real-world images and produces a triangle mesh giving the full 3D shape of each detected object. Our system, called Mesh R-CNN, augments Mask R-CNN with a mesh prediction branch that outputs meshes with varying topological structure by first predicting coarse voxel representations which are converted to meshes and refined with a graph convolution network operating over the mesh's vertices and edges. We validate our mesh prediction branch on ShapeNet, where we outperform prior work on single-image shape prediction. We then deploy our full Mesh R-CNN system on Pix3D, where we jointly detect objects and predict their 3D shapes.

http://www.arxiv-sanity.com/search?q=mesh+r-cnn

# K-Nearest Neighbors: Distance Metric

Most similar papers:

**Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era**
1906.06543v2 pdf
show similar | discuss
Xian-Feng Han, Hamid Laga, Mohammed Bennamoun
6/18/2019 (v1: 6/15/2019) cs.CV | cs.CG | cs.GR | cs.LG

3D reconstruction is a longstanding ill-posed problem, which has been explored for decades by the computer vision, computer graphics, and machine learning communities. Since 2015, image-based 3D reconstruction using convolutional neural networks (CNN) has attracted increasing interest and demonstrated an impressive performance. Given this new era of rapid evolution, this article provides a comprehensive survey of the recent developments in this field. We focus on the works which use deep learning techniques to estimate the 3D shape of generic objects either from a single or multiple RGB images. We organize the literature based on the shape representations, the network architectures, and the training mechanisms they use. While this survey is intended for methods which reconstruct generic objects, we also review some of the recent works which focus on specific object classes such as human body shapes and faces. We provide an analysis and comparison of the performance of some key papers, summarize some of the open problems in this field, and discuss promising directions for future research.

**Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images**
1804.01654v2 pdf
show similar | discuss
Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, Yu-Gang Jiang
8/3/2018 (v1: 4/5/2018) cs.CV

We propose an end-to-end deep learning architecture that produces a 3D shape in triangular mesh from a single color image. Limited by the nature of deep neural network, previous methods usually represent a 3D shape in volume or point cloud, and it is non-trivial to convert them to the more ready-to-use mesh model. Unlike the existing methods, our network represents 3D mesh in a graph-based convolutional neural network and produces correct geometry by progressively deforming an ellipsoid, leveraging perceptual features extracted from the input image. We adopt a coarse-to-fine strategy to make the whole deformation procedure stable, and define various of mesh related losses to capture properties of different levels to guarantee visually appealing and physically accurate 3D geometry. Extensive experiments show that our method not only qualitatively produces mesh model with better details, but also achieves higher 3D shape estimation accuracy compared to the state-of-the-art.

**Pixel2Mesh++: Multi-View 3D Mesh Generation via Deformation**
1908.01491v2 pdf
show similar | discuss
Chao Wen, Yinda Zhang, Zhuwen Li, Yanwei Fu
8/16/2019 (v1: 8/5/2019) cs.CV
Accepted by ICCV 2019

We study the problem of shape generation in 3D mesh representation from a few color images with known camera poses. While many previous works learn to hallucinate the shape directly from priors, we resort to further improving the shape quality by leveraging cross-view information with a graph convolutional network. Instead of building a direct mapping function from images to 3D shape, our model learns to predict series of deformations to improve a coarse shape iteratively. Inspired by traditional multiple view geometry methods, our network samples nearby area around the initial mesh's vertex locations and reasons an optimal deformation using perceptual feature statistics built from multiple input images. Extensive experiments show that our model produces accurate 3D shape that are not only visually plausible from the input perspectives, but also well aligned to arbitrary viewpoints. With the help of physically driven architecture, our model also exhibits generalization capability across different semantic categories, number of input images, and quality of mesh initialization.

**GEOMetrics: Exploiting Geometric Structure for Graph-Encoded Objects**
1901.11461v1 pdf
show similar | discuss
Edward J. Smith, Scott Fujimoto, Adriana Romero, David Meger
1/31/2019 cs.CV
18 pages

Mesh models are a promising approach for encoding the structure of 3D objects. Current mesh reconstruction systems predict uniformly distributed vertex locations of a predetermined graph through a series of graph convolutions, leading to compromises with respect to performance or resolution. In this paper, we argue that the graph representation of geometric objects allows for additional structure, which should be leveraged for enhanced reconstruction. Thus, we propose a system which properly benefits from the advantages of the geometric structure of graph encoded objects by introducing (1) a graph convolutional update preserving vertex information; (2) an adaptive splitting heuristic allowing detail to emerge; and (3) a training objective operating both on the local surfaces defined by vertices as well as the global structure defined by the mesh. Our proposed method is evaluated on the task of 3D object reconstruction from images with the ShapeNet dataset, where we demonstrate state of the art performance, both visually and numerically, while having far smaller space requirements by generating adaptive meshes

http://www.arxiv-sanity.com/1906.02739v1

# Hyperparameters

What is the best value of **K** to use?
What is the best **distance metric** to use?

These are examples of **hyperparameters**: choices about our learning algorithm that we don't learn from the training data; instead we set them at the start of the learning process

# Hyperparameters

What is the best value of **K** to use?
What is the best **distance metric** to use?

These are examples of **hyperparameters**: choices about our learning algorithm that we don't learn from the training data; instead we set them at the start of the learning process

Very problem-dependent. In general we have to try them all and see what works best for our data / task.

# Setting Hyperparameters

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

# Setting Hyperparameters

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

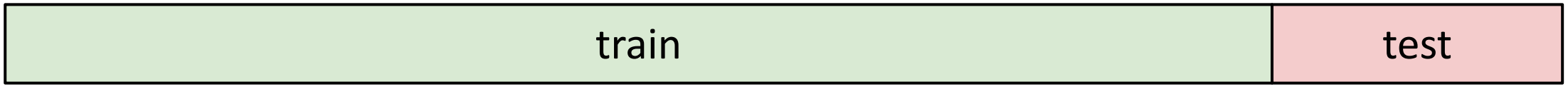
**BAD**: K = 1 always works perfectly on training data

# 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



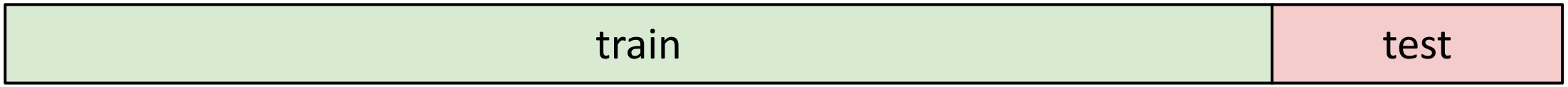
# Setting Hyperparameters

**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



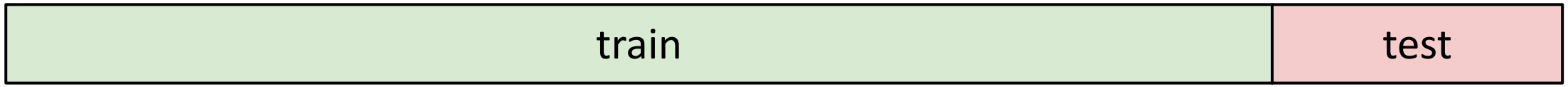
# Setting Hyperparameters

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

**BAD**: K = 1 always works perfectly on training data

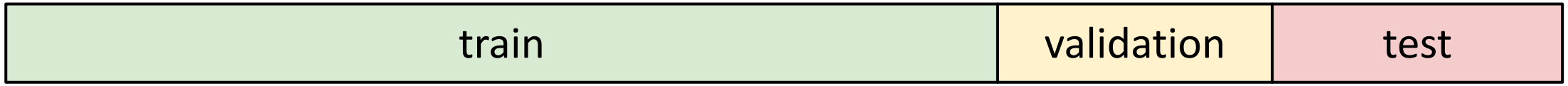

**Idea #2**: Split data into **train** and **test**, choose hyperparameters that work best on test data

**BAD**: No idea how algorithm will perform on new data



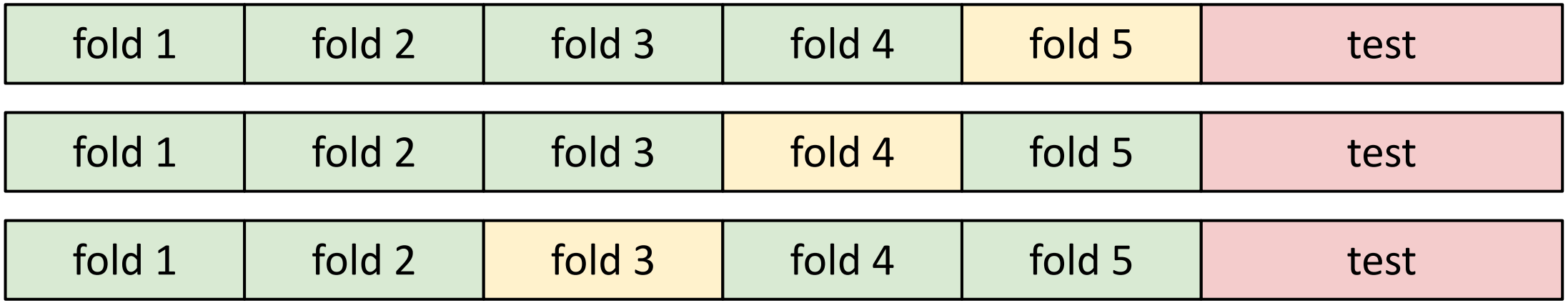

**Idea #3**: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

**Better!**



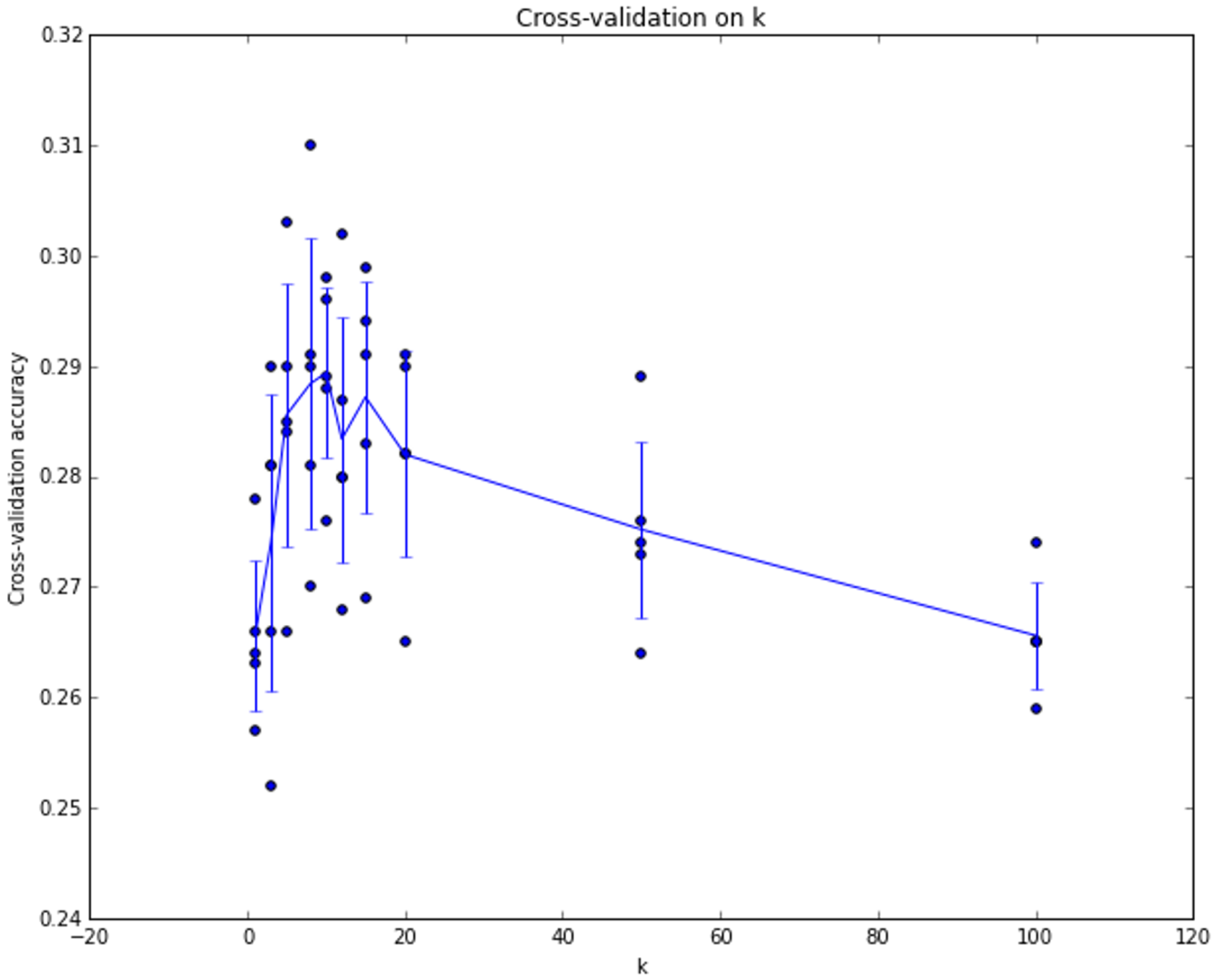
# Setting Hyperparameters


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




Useful for small datasets, but (unfortunately) 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 indicated standard deviation

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

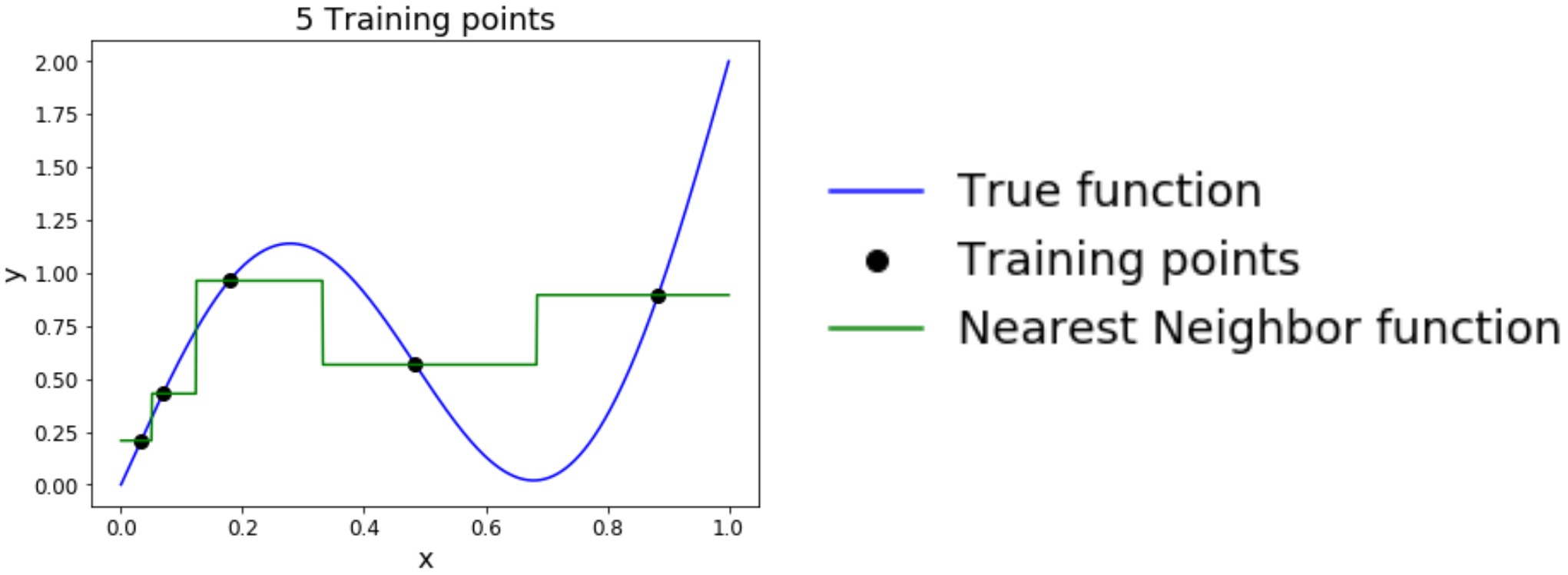
# K-Nearest Neighbor: Universal Approximation

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

(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

# K-Nearest Neighbor: **Universal Approximation**

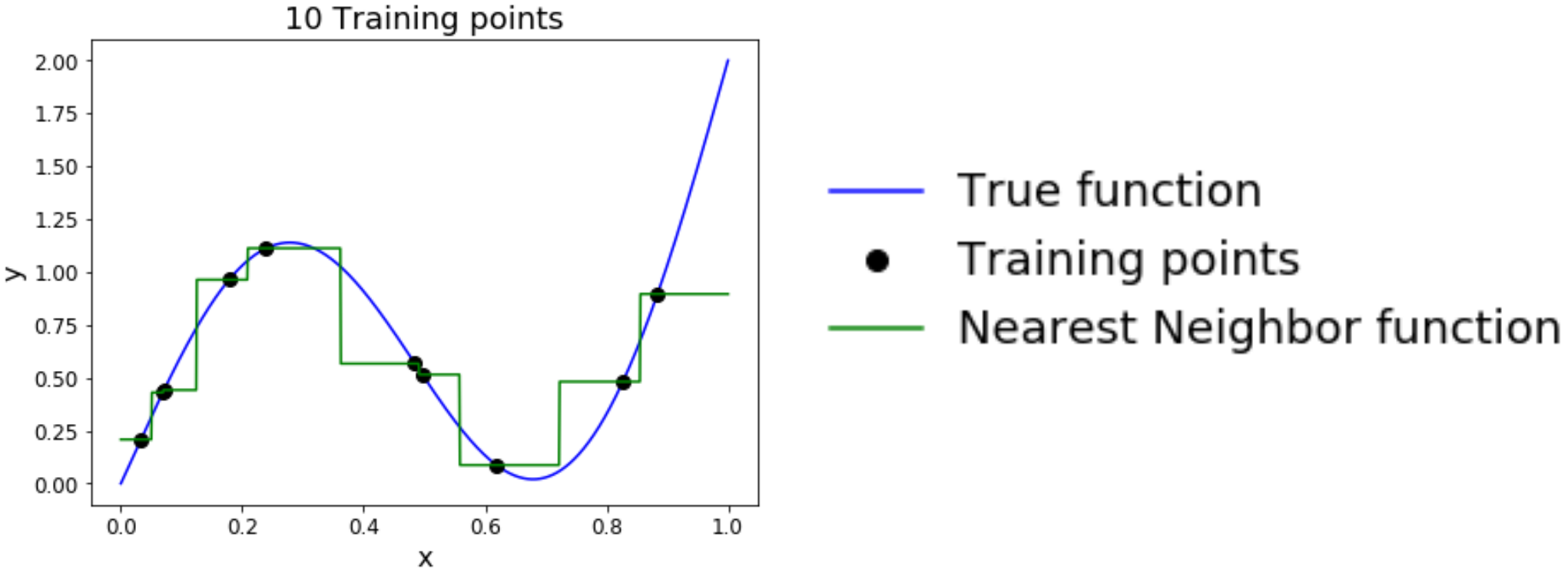
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!




(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

# K-Nearest Neighbor: **Universal Approximation**

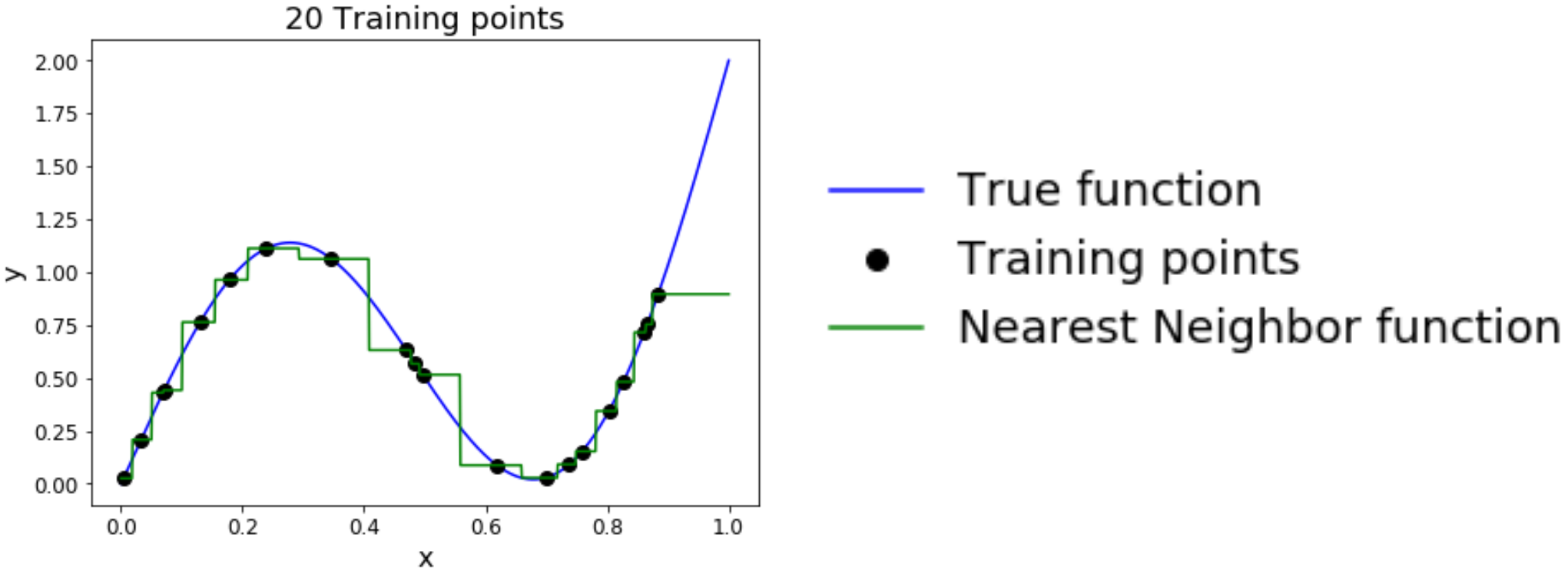
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!




(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

# K-Nearest Neighbor: **Universal Approximation**

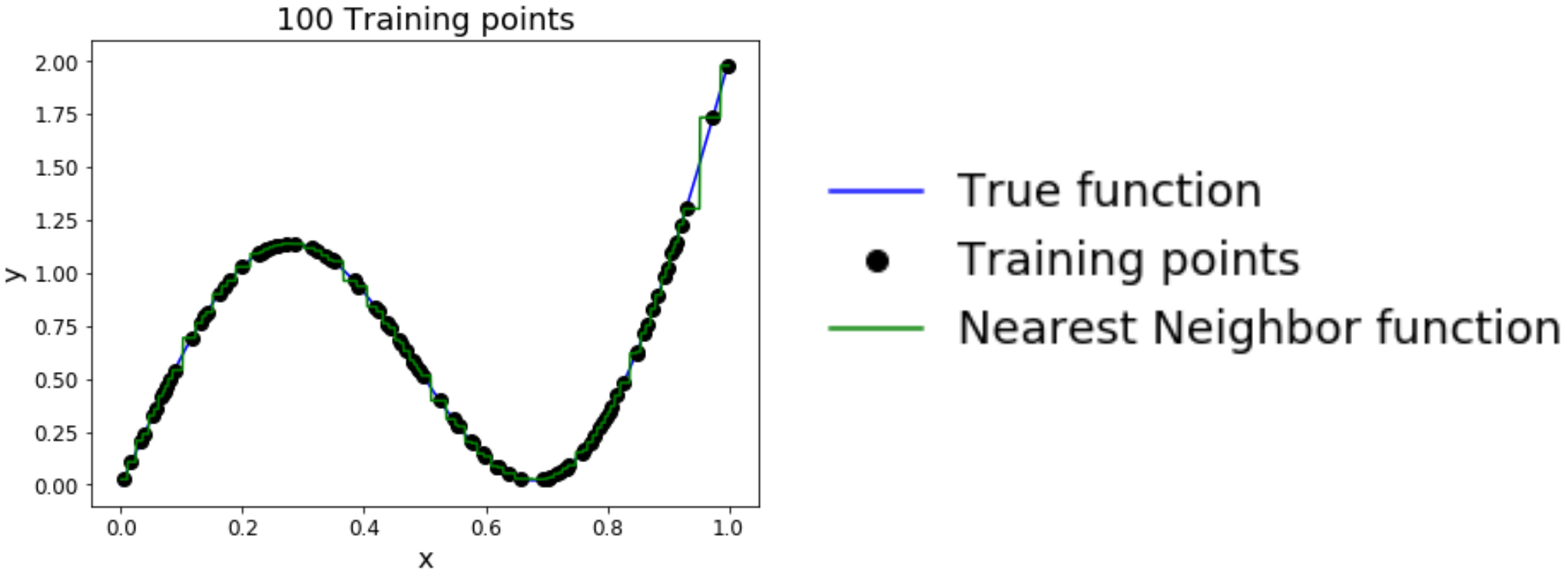
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!




(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

# K-Nearest Neighbor: **Universal Approximation**

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



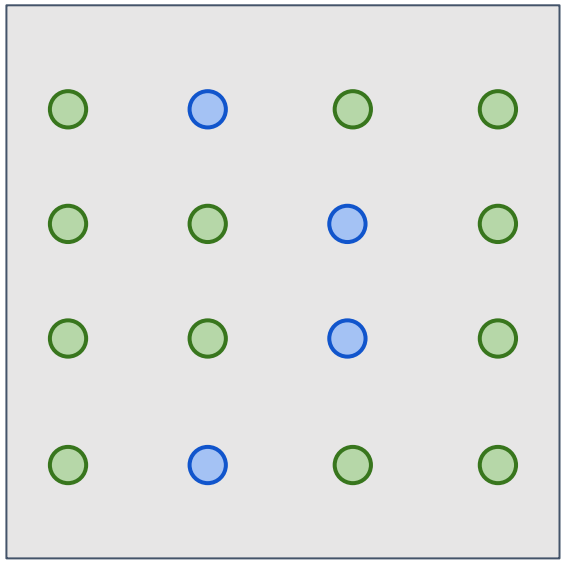

(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

# Problem: Curse of Dimensionality

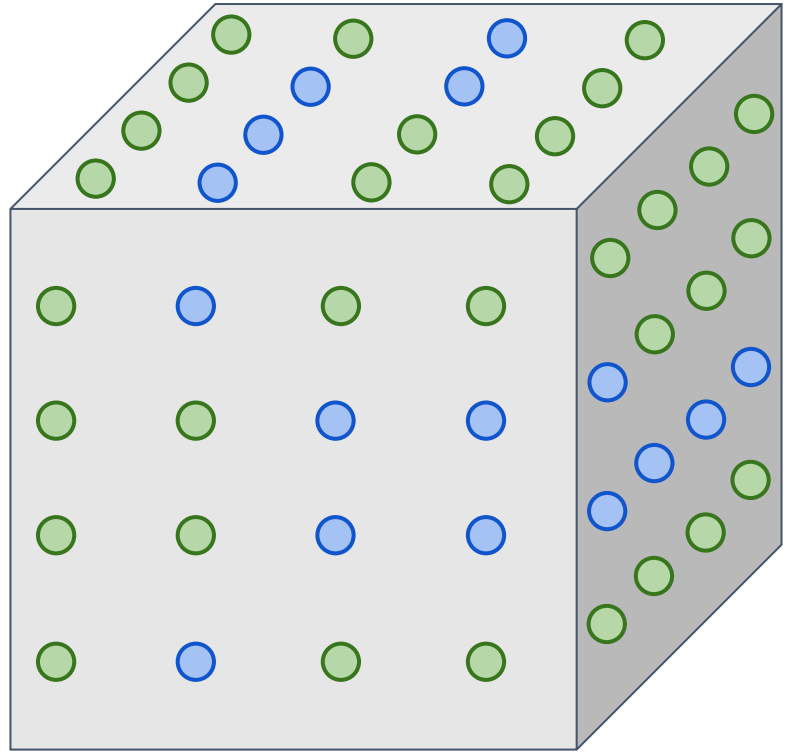
**Curse of dimensionality**: For uniform coverage of space, number of training points needed grows exponentially with dimension

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:

$$2^{32x32} \approx 10^{308}$$

# 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:

$2^{32x32} \approx 10^{308}$

Number of elementary particles in the visible universe: (source)

$\approx 10^{97}$

# Problem: Pixel distance might not be an accurate metric

- Very slow at test time
- Distance metrics on pixels are not informative

Original

Boxed

Shifted

Tinted

(all 3 images have same L2 distance to the one on the left)

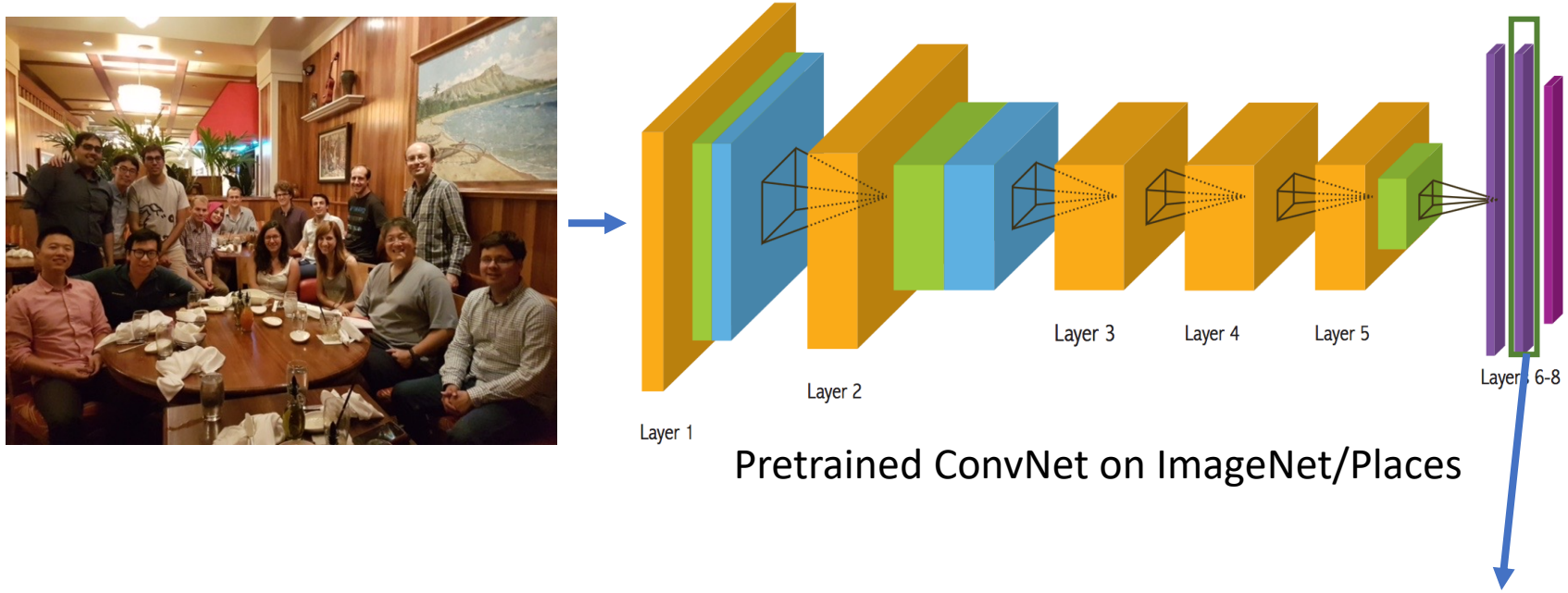
# But is pixel metric really that bad?

- Image retrieval on pixels under different image numbers


Torralba, Fergus, Freeman: 80 million tiny images: a large dataset for non-parametric object and scene recognition. TPAMI 2008

https://people.csail.mit.edu/torralba/publications/80millionImages.pdf

# Using pre-trained ConvNet features



# Nearest Neighbor with ConvNet features works well!

Devlin et al, “Exploring Nearest Neighbor Approaches for Image Captioning”, 2015

# Nearest Neighbor with ConvNet features works well!

## Example: Image Captioning with Nearest Neighbor

A bedroom with a bed and a couch.

A cat sitting in a bathroom sink.

A train is stopped at a train station.

A wooden bench in front of a building.

Devlin et al, “Exploring Nearest Neighbor Approaches for Image Captioning”, 2015

# Why ConvNet feature works so well?

- Stay tuned for visualizing and understanding ConvNet and other DNN

# Summary

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

Image classification is challenging due to the semantic gap: we need invariance to occlusion, deformation, lighting, intraclass variation, etc

Image classification is a **building block** for other vision tasks

The **K-Nearest Neighbors** 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!

# Practices on Google Colab

- Go through the Google Colab Tutorials
  - https://towardsdatascience.com/getting-started-with-google-colab-f2fff97f594c
  - https://colab.research.google.com/notebooks/intro.ipynb#recent=true
  - https://medium.com/swlh/the-best-place-to-get-started-with-ai-google-colab-tutorial-for-beginners-715e64bb603b
- Play with some Colab examples to deepen your understanding:
  - Scene parsing: https://colab.research.google.com/github/CSAILVision/semantic-segmentation-pytorch/blob/master/notebooks/DemoSegmenter.ipynb
  - BigGAN image generation: https://colab.research.google.com/github/tensorflow/hub/blob/master/examples/colab/biggan_generation_with_tf_hub.ipynb

# Go through the List of Potential Project Topics

- You can find a partner through Piazza or now and form a team
- Think about what you want to work on
  - End of week 2: finalize groups
  - End of week 4: finalize topics (instead of week 3 as stated in the slide above)
- Please see and sign up the team and project information at the Google Doc (https://docs.google.com/document/d/1oeQWkJmCuHfJKAY4QAKgU6o3u33CJcgLxXZInDap7ug/edit?usp=sharing)

# Next time: Linear Neural Networks/Classifiers