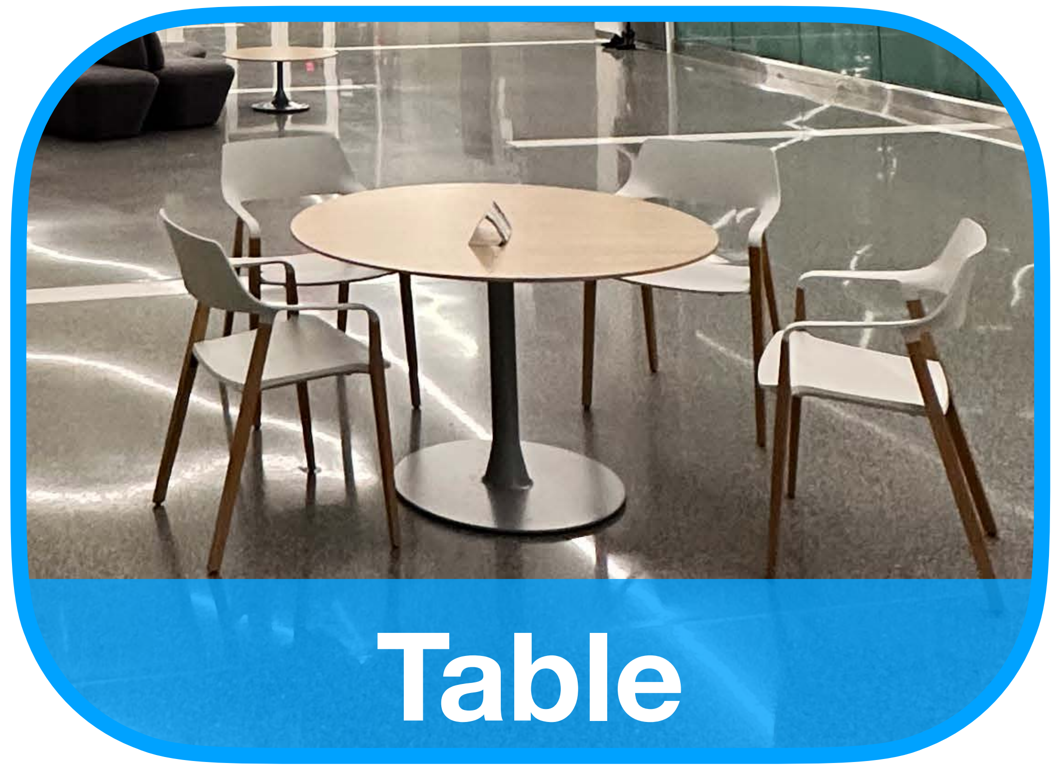
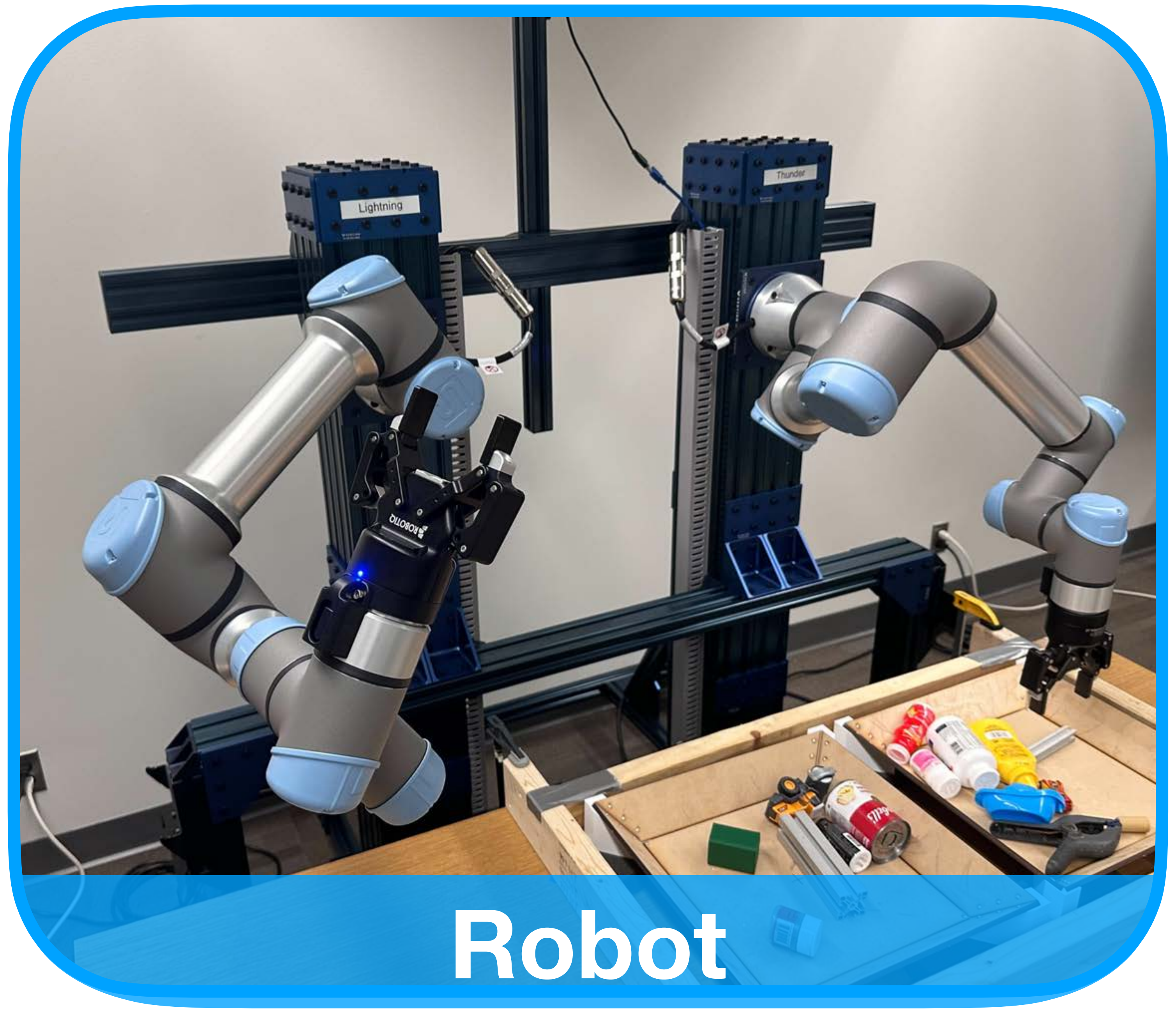
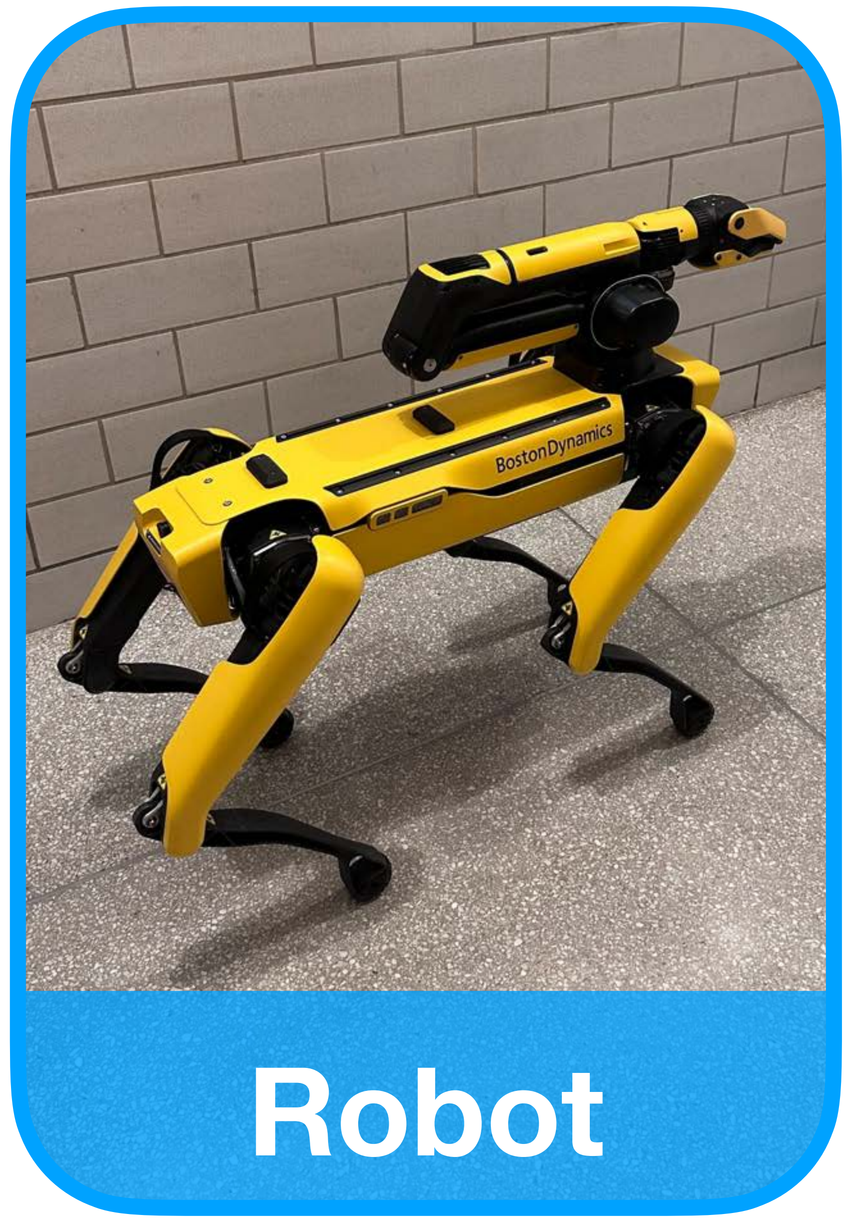




DeepRob

Lecture 2
Image Classification
University of Minnesota

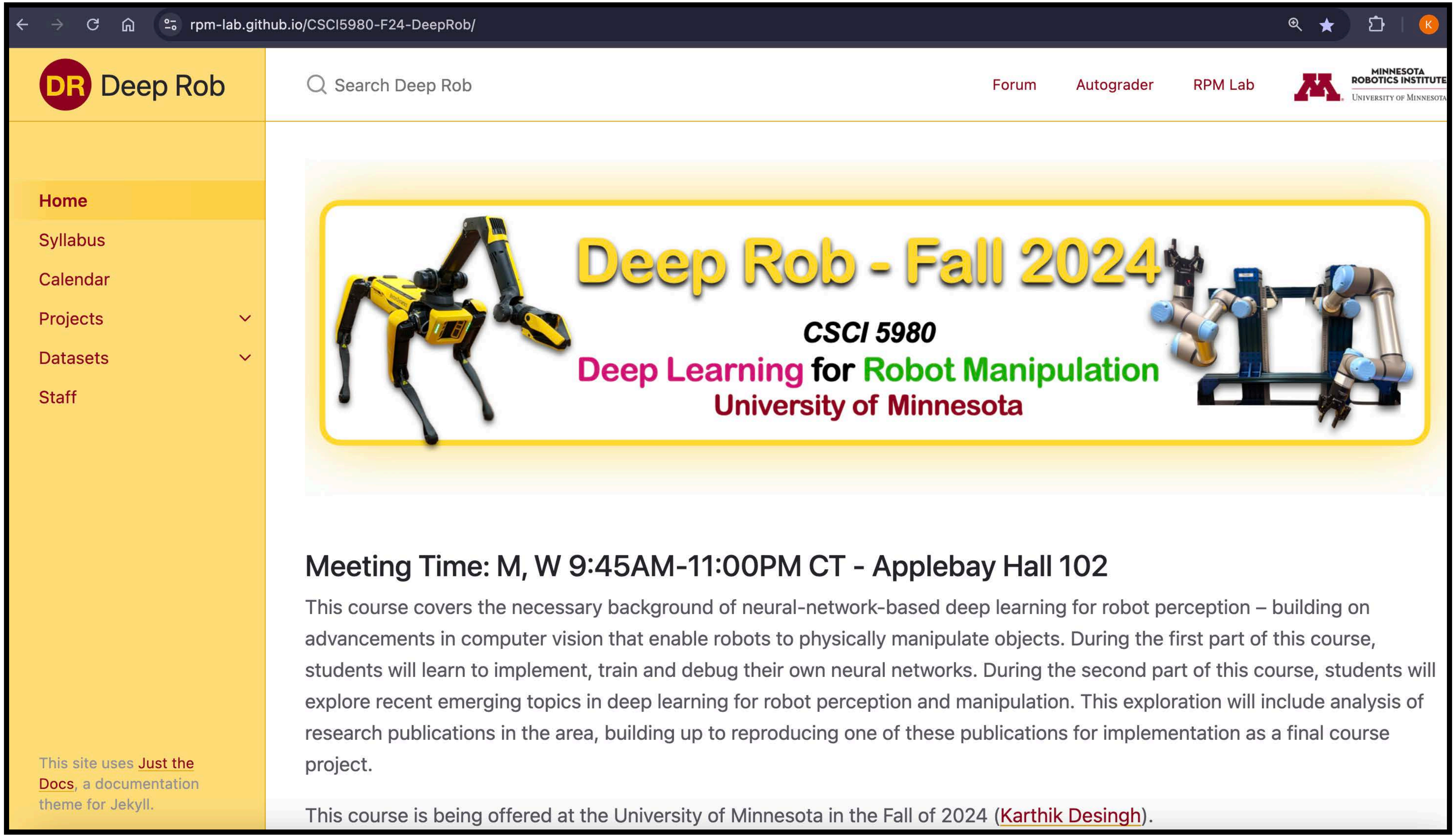


Slides adapted from Justin Johnson's Deep Learning for Computer Vision course

Course Resources

- Course Website: https://rpm-lab.github.io/CSCI5980-F24-DeepRob/
  - Syllabus, calendar, project files, slides, links, etc.
- Ed Stem: https://edstem.org/us/courses/66160/discussion/
  - Forum for communication and question answering

Course Website: https://rpm-lab.github.io/CSCI5980-F24-DeepRob/



Course Website:

https://rpm-lab.github.io/CSCI5980-F24-DeepRob/

rpm-lab.github.io/CSCI5980-F24-DeepRob/syllabus/

DR Deep Rob

Search Deep Rob

Home
Syllabus
Calendar
Projects
Datasets
Staff

Course Syllabus

TABLE OF CONTENTS

rpm-lab.github.io/CSCI5980-F24-DeepRob/calendar/

DR Deep Rob

Home
Syllabus
Calendar
Projects
Datasets
Staff

Current Running Schedule

Week 1

Sept 04: LEC 1 Course Introduction

Snapshot of Planned Schedule

DeepRob_Fall24_Calendar : Sheet1

Week	Lec #	Date	Topic	Proj Release	Proj Due	Final Project Phases
1	1	09/04	Introduction			
2	2	09/09	Image Classification	P0 (optional)		Individual Tasks: - Brainstorming on robot tasks - Reading 3 papers
	3	09/11	Linear Classifiers	P1	P0 (optional)	
3	4	09/16	Regularization - Optimization			
	5	09/18	Neural Networks			
4	6	09/23	Backpropagation			
	7	09/25	CNNs	P2	P1	
5	8	09/30	CNNs			Team Init: - Team formation tasks - Reading 3 papers as a team - Converge on a topic and paper(s) - Project proposal
	9	10/02	Training NN1			
6	10	10/07	Training NN2 + DL Software			
	11	10/09	Object detection	P3	P2	
7	12	10/14	Semantic segmentation			
	13	10/16	Grasp Pose Learning			
8	14	10/21	Imitation Learning - I			Diving into the research topic: - Prepare in-lecture presentation
	15	10/23	Imitation Learning - II	P4	P3	
	25	11/27	Thanksgiving break			
14	26	12/02	*Student lecture 9*			Evaluation: - Metrics to evaluate the performance - Demonstration on simulated or real-robot
	27	12/04	Student lecture 10			
15	28	12/09	Extra OH / Guest lecture			
	29	12/11	Extra OH / Guest lecture		FP Posters Due	
16	30	12/16	Poster presentation day (tentative)		FP Videos Due	

This site uses Just the Docs, a documentation

Instructors

Karthik Desingh
kdesingh@umn.edu
Office Hours: F 10:00am - 12:00pm 159-Shepherd Labs

rpm-lab.github.io/CSCI5980-F24-DeepRob/syllabus/

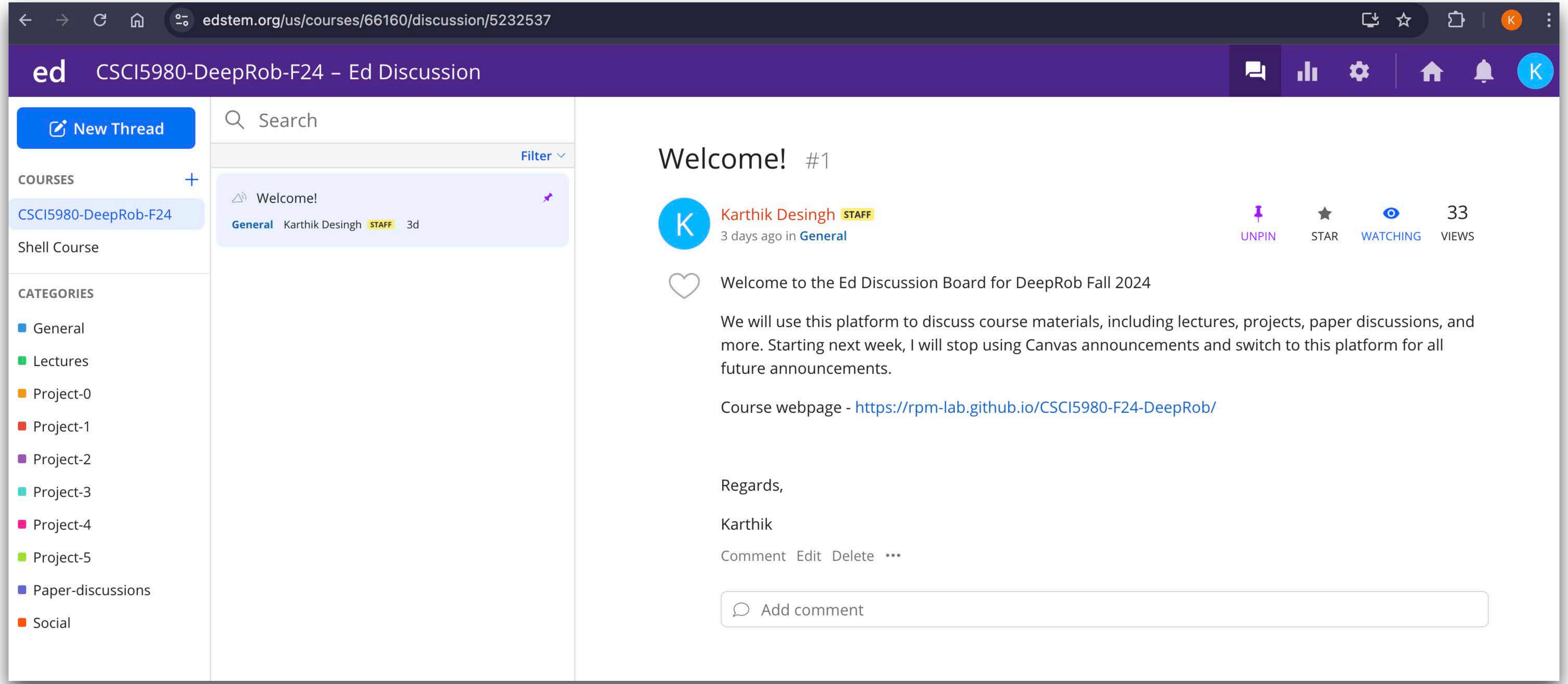
DR Deep Rob

Search Deep Rob

Forum

Autograder

Discussion Forum



Project Grading

- Projects 1-5
  - 2 total late days available
  - 25% daily penalty after deadline and late days

- Final project graded manually by course staff

Overall Grading Policy

rpm-lab.github.io/CSCI5980-F24-DeepRob/syllabus/#grading-policy

DR Deep Rob

- Home
- **Syllabus**
- Calendar
- Projects
- Datasets
- Staff

Grading Policy

Course grades will be determined according to the following criteria:

- Project 0 (optional and **not graded**)
- Project 1 (Linear classfication): 5%
- Project 2 (Fully-connected and CNNs) : 10%
- Project 3 (Object detection with CNNs): 10%
- Project 4 (Object pose estimation): 10%
- Project 5 (Imitation learning): 10%
- Final Project:
  - Individual brainstorming and reading: 5%
  - In-class presentation background: 5%
  - In-class presentation paper in detail: 5%
  - Data acquisition/Simulation setup: 10%
  - Network development: 10%
  - Training strategy and evaluation: 10%
  - Video and poster: 10%

Textbook

rpm-lab.github.io/CSCI5980-F24-DeepRob/syllabus/#textbook

DR Deep Rob

- Home
- **Syllabus**
- Calendar
- Projects

Textbook

There is no required textbook for this course, however optional readings will be suggested from the textbook, "Deep Learning" by Ian Goodfellow and Yoshua Bengio and Aaron Courville.

For additional references, consider the following textbooks:

"Introduction to Robotics and Perception" by Frank Dellaert and Seth Hutchinson "Robotics, Vision and Control" by Peter Corke "Computer Vision: Algorithms and Applications" by Richard Szeliski "Foundations of Computer Vision" by Antonio Torralba, Phillip Isola, and William T. Freeman

No textbook required!

Collaboration Policy

rpm-lab.github.io/CSCI5980-F24-DeepRob/syllabus/#collaboration-policy

Deep Rob

Home

Syllabus

Calendar

Collaboration Policy

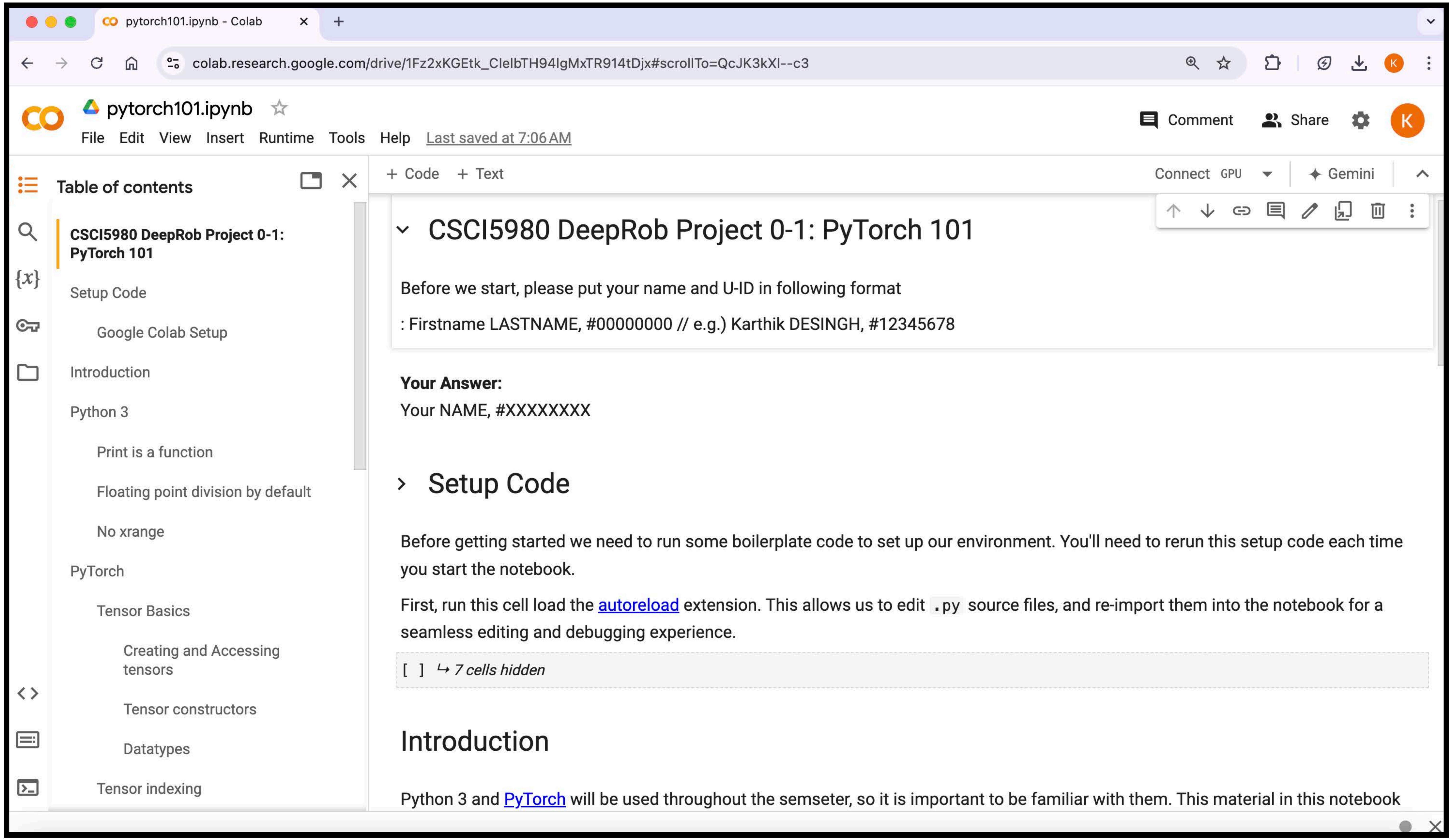
The free flow of discussion and ideas is encouraged. **But, everything you turn in must be your own work**, and you must note the names of anyone you collaborated with on each problem and cite resources that you used to learn about the problem. **If you have any doubts about whether a particular action may be construed as cheating, ask the instructor for clarification before you do it.** Cheating in this course will result in a grade of F for course and the University policies will be followed.

No code can be communicated, including verbally. Explicit use of external sources must be clearly cited in your presentations and code.

Project 0

- Instructions and code available on the website
- Released today: https://rpm-lab.github.io/CSCI5980-F24-DeepRob/projects/project0
- **Due next Monday, Sept 16th 11:59 PM CT**
- **Autograder will be made available soon!**

Project 0


pytorch101.ipynb - Colab
colab.research.google.com/drive/1Fz2xKGEtk_ClelbTH94lgMxTR914tDjx#scrollTo=QcJK3kXl--c3
pytorch101.ipynb
File Edit View Insert Runtime Tools Help Last saved at 7:06 AM
Comment
Share
Table of contents
CSCI5980 DeepRob Project 0-1: PyTorch 101
Setup Code
Google Colab Setup
Introduction
Python 3
Print is a function
Floating point division by default
No xrange
PyTorch
Tensor Basics
Creating and Accessing tensors
Tensor constructors
Datatypes
Tensor indexing
+ Code + Text
Connect GPU
Gemini
CSCI5980 DeepRob Project 0-1: PyTorch 101
Before we start, please put your name and U-ID in following format
: Firstname LASTNAME, #00000000 // e.g.) Karthik DESINGH, #12345678
Your Answer:
Your NAME, #XXXXXXXX
Setup Code
Before getting started we need to run some boilerplate code to set up our environment. You'll need to rerun this setup code each time you start the notebook.
First, run this cell load the autoreload extension. This allows us to edit .py source files, and re-import them into the notebook for a seamless editing and debugging experience.
[ ] 7 cells hidden
Introduction
Python 3 and PyTorch will be used throughout the semseter, so it is important to be familiar with them. This material in this notebook

Project 0 Suggestions

- If you choose to develop locally
  - **PyTorch Version 1.13.0**

- Ensure you save your notebook file before uploading submission

- Close any Colab notebooks not in use to avoid usage limits

Image Classification

Image Classification—A Core Computer Vision Task

Input: image

Output: assign image to one of a fixed set of categories

Chocolate Pretzels

Granola Bar

Potato Chips

Water Bottle

Popcorn

Problem—Semantic Gap

Input: image

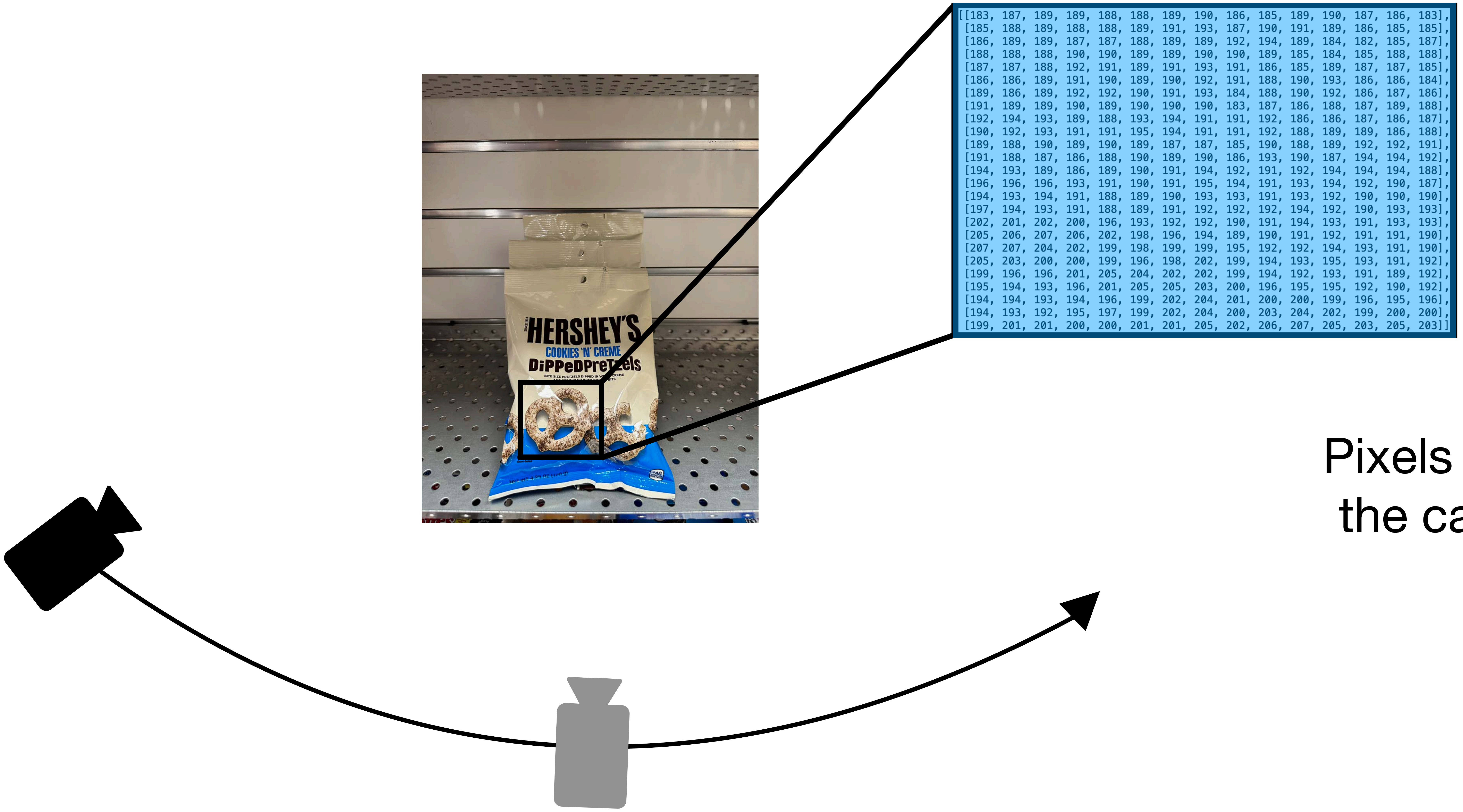

```
[[183, 187, 189, 189, 188, 188, 189, 190, 186, 185, 189, 190, 187, 186, 183],
 [185, 188, 189, 188, 188, 189, 191, 193, 187, 190, 191, 189, 186, 185, 185],
 [186, 189, 189, 187, 187, 188, 189, 189, 192, 194, 189, 184, 182, 185, 187],
 [188, 188, 188, 190, 190, 189, 189, 190, 190, 189, 185, 184, 185, 188, 188],
 [187, 187, 188, 192, 191, 189, 191, 193, 191, 186, 185, 189, 187, 187, 185],
 [186, 186, 189, 191, 190, 189, 190, 192, 191, 188, 190, 193, 186, 186, 184],
 [189, 186, 189, 192, 192, 190, 191, 193, 184, 188, 190, 192, 186, 187, 186],
 [191, 189, 189, 190, 189, 190, 190, 190, 183, 187, 186, 188, 187, 189, 188],
 [192, 194, 193, 189, 188, 193, 194, 191, 191, 192, 186, 186, 187, 186, 187],
 [190, 192, 193, 191, 191, 195, 194, 191, 191, 192, 188, 189, 189, 186, 188],
 [189, 188, 190, 189, 190, 189, 187, 187, 185, 190, 188, 189, 192, 192, 191],
 [191, 188, 187, 186, 188, 190, 189, 190, 186, 193, 190, 187, 194, 194, 192],
 [194, 193, 189, 186, 189, 190, 191, 194, 192, 191, 192, 194, 194, 194, 188],
 [196, 196, 196, 193, 191, 190, 191, 195, 194, 191, 193, 194, 192, 190, 187],
 [194, 193, 194, 191, 188, 189, 190, 193, 193, 191, 193, 192, 190, 190, 190],
 [197, 194, 193, 191, 188, 189, 191, 192, 192, 192, 194, 192, 190, 193, 193],
 [202, 201, 202, 200, 196, 193, 192, 192, 190, 191, 194, 193, 191, 193, 193],
 [205, 206, 207, 206, 202, 198, 196, 194, 189, 190, 191, 192, 191, 191, 190],
 [207, 207, 204, 202, 199, 198, 199, 199, 195, 192, 192, 194, 193, 191, 190],
 [205, 203, 200, 200, 199, 196, 198, 202, 199, 194, 193, 195, 193, 191, 192],
 [199, 196, 196, 201, 205, 204, 202, 202, 199, 194, 192, 193, 191, 189, 192],
 [195, 194, 193, 196, 201, 205, 205, 203, 200, 196, 195, 195, 192, 190, 192],
 [194, 194, 193, 194, 196, 199, 202, 204, 201, 200, 200, 199, 196, 195, 196],
 [194, 193, 192, 195, 197, 199, 202, 204, 200, 203, 204, 202, 199, 200, 200],
 [199, 201, 201, 200, 200, 201, 201, 205, 202, 206, 207, 205, 203, 205, 203]]
```

What the computer sees

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

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

Challenges—Viewpoint Variation



Pixels change when
the camera moves

Challenges—Intraclass Variation

Challenges—Fine-Grained Categories

Milk Chocolate

White Chocolate

Cookies N' Creme

Peanut Butter

Ambiguous Category

Challenges—Background Clutter

Challenges—Image Resolution

iPhone 14 Camera

4032x3024

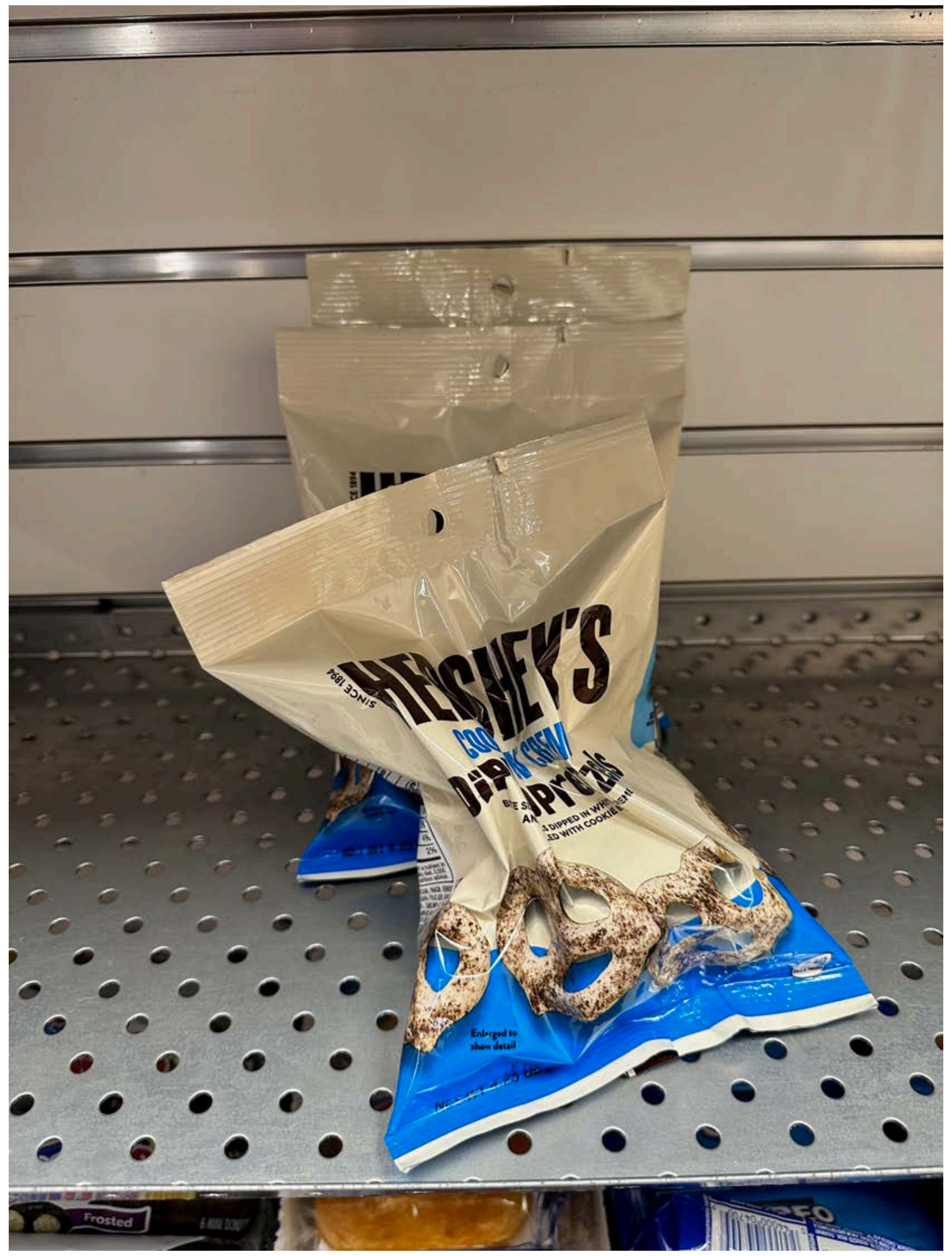
ASUS RGB-D Camera

640x480

Challenges—Illumination Changes

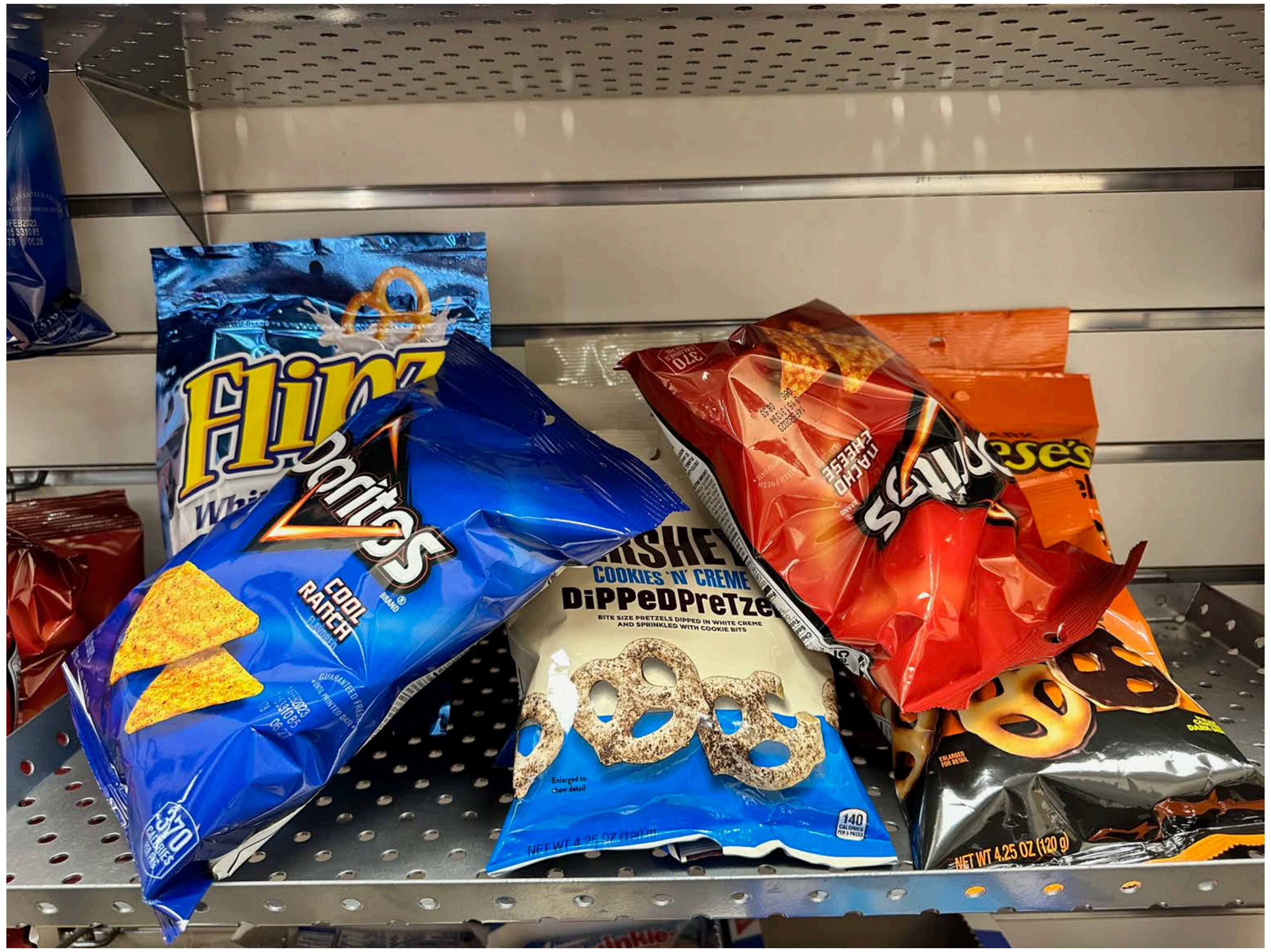
Want our robot's perception system to be reliable in all conditions

Challenges—Subject Deformation

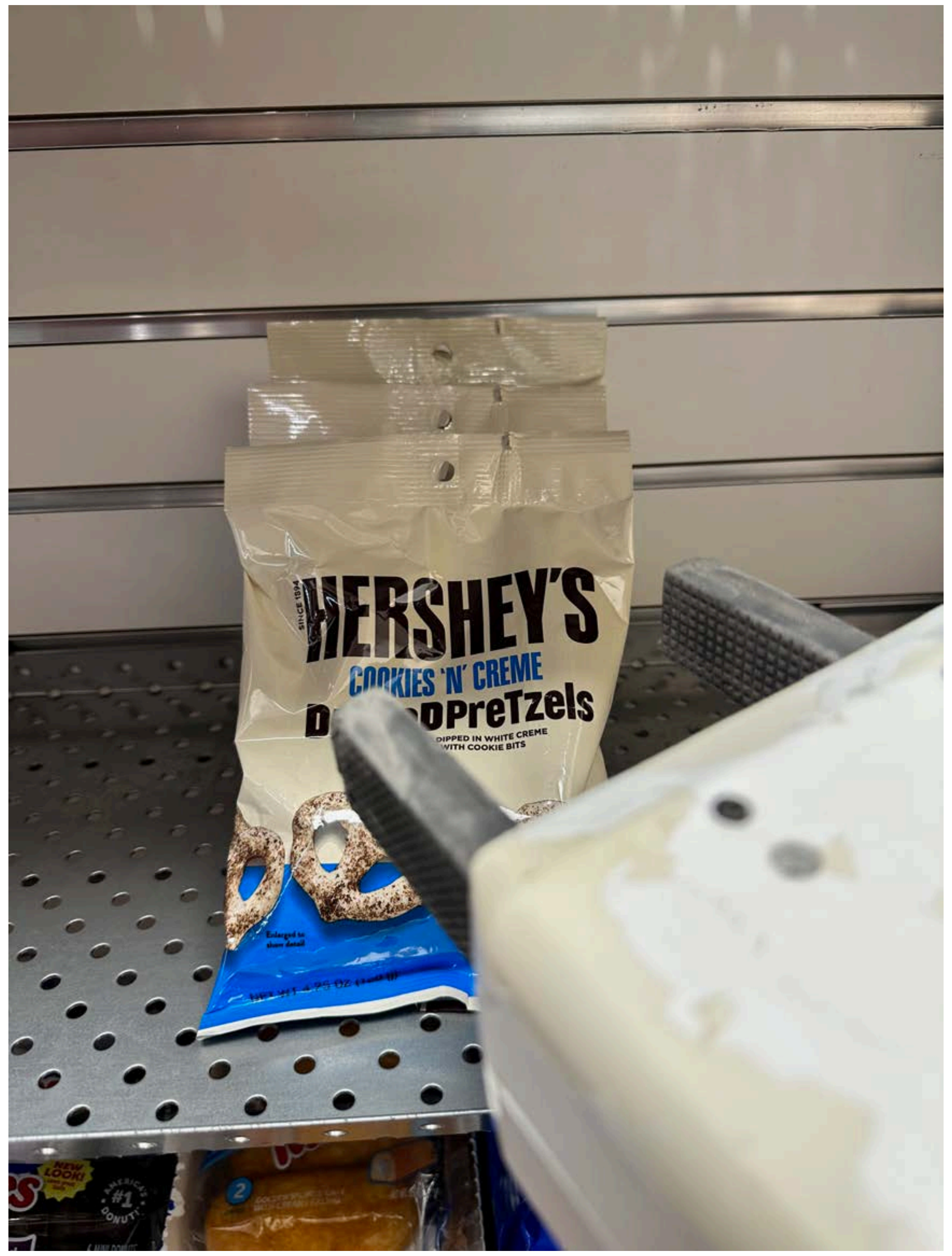


Challenges—Occlusion

Scene Clutter



Robot Actuator



Transparency

Challenges—Semantic Relationships

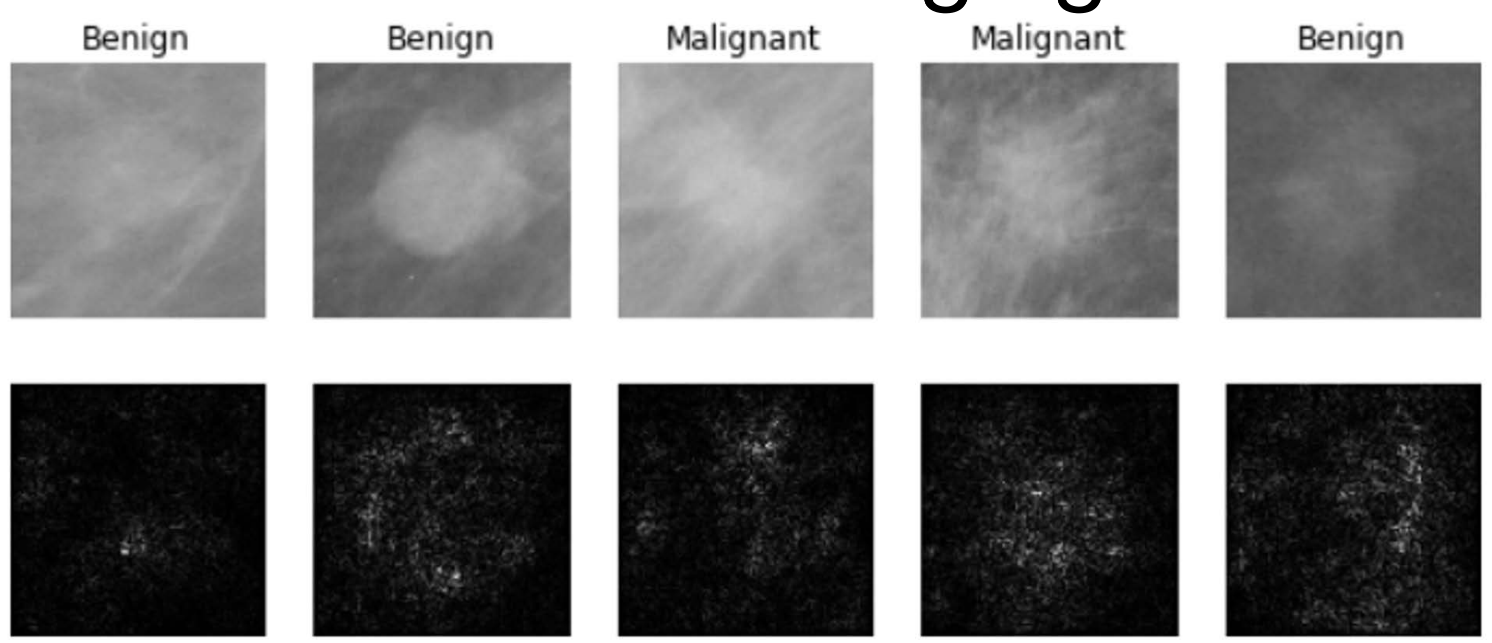
Reflections

Contact Relationships

<u>Robots have to act</u> on the state they perceive

Applications of Image Classification

Medical Imaging



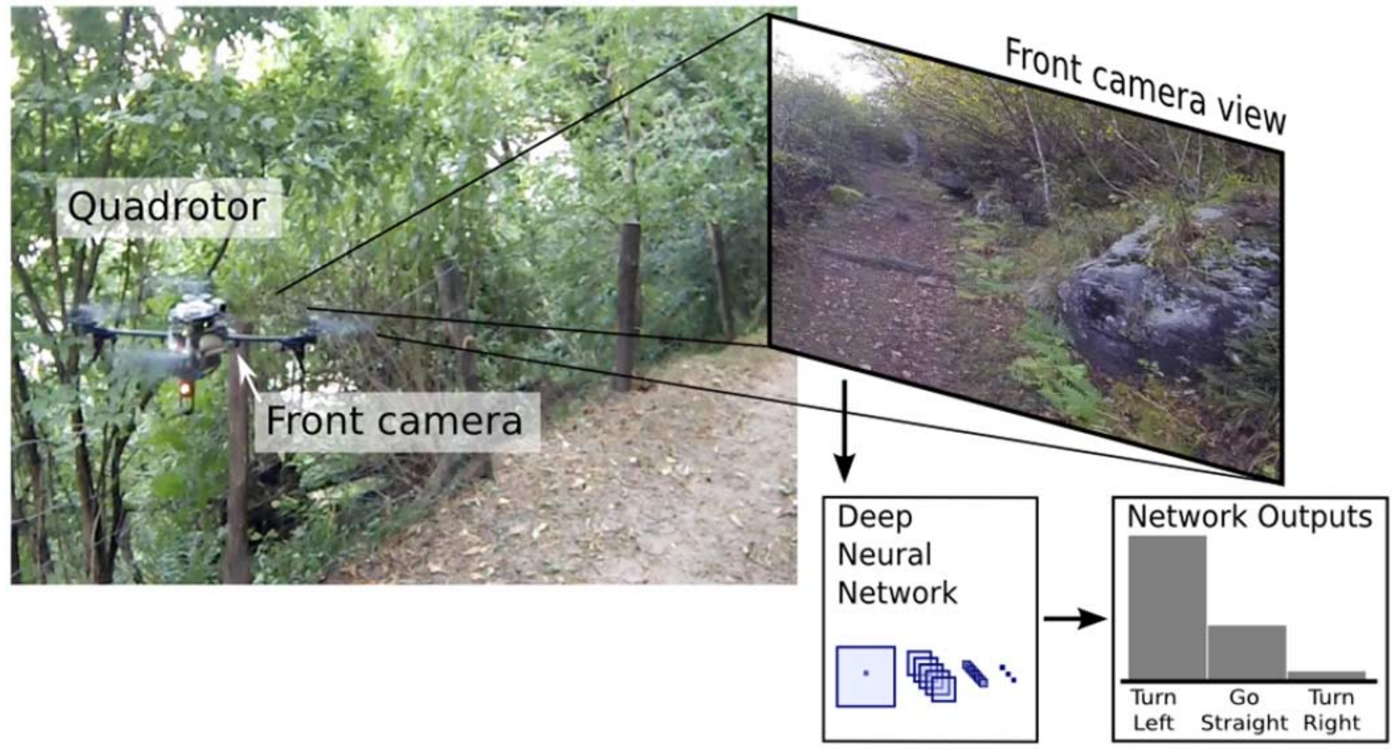
Lévy et al., “Breast Mass Classification from Mammograms using Deep Convolutional Neural Networks”, arXiv:1612.00542, 2016

Galaxy Classification

Dieleman et al., “Rotation-invariant convolutional neural networks for galaxy morphology prediction”, 2015

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain

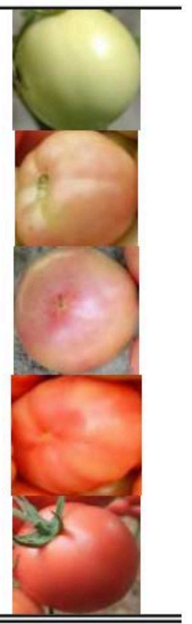
Trail Direction Classification

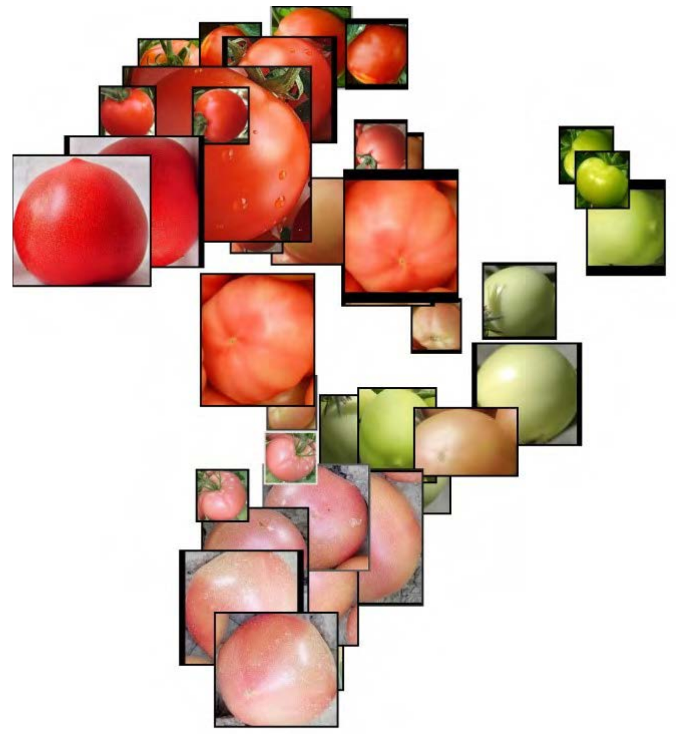


Giusti et al., “A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots”, IEEE RAL, 2016

Tomato Ripeness Classification

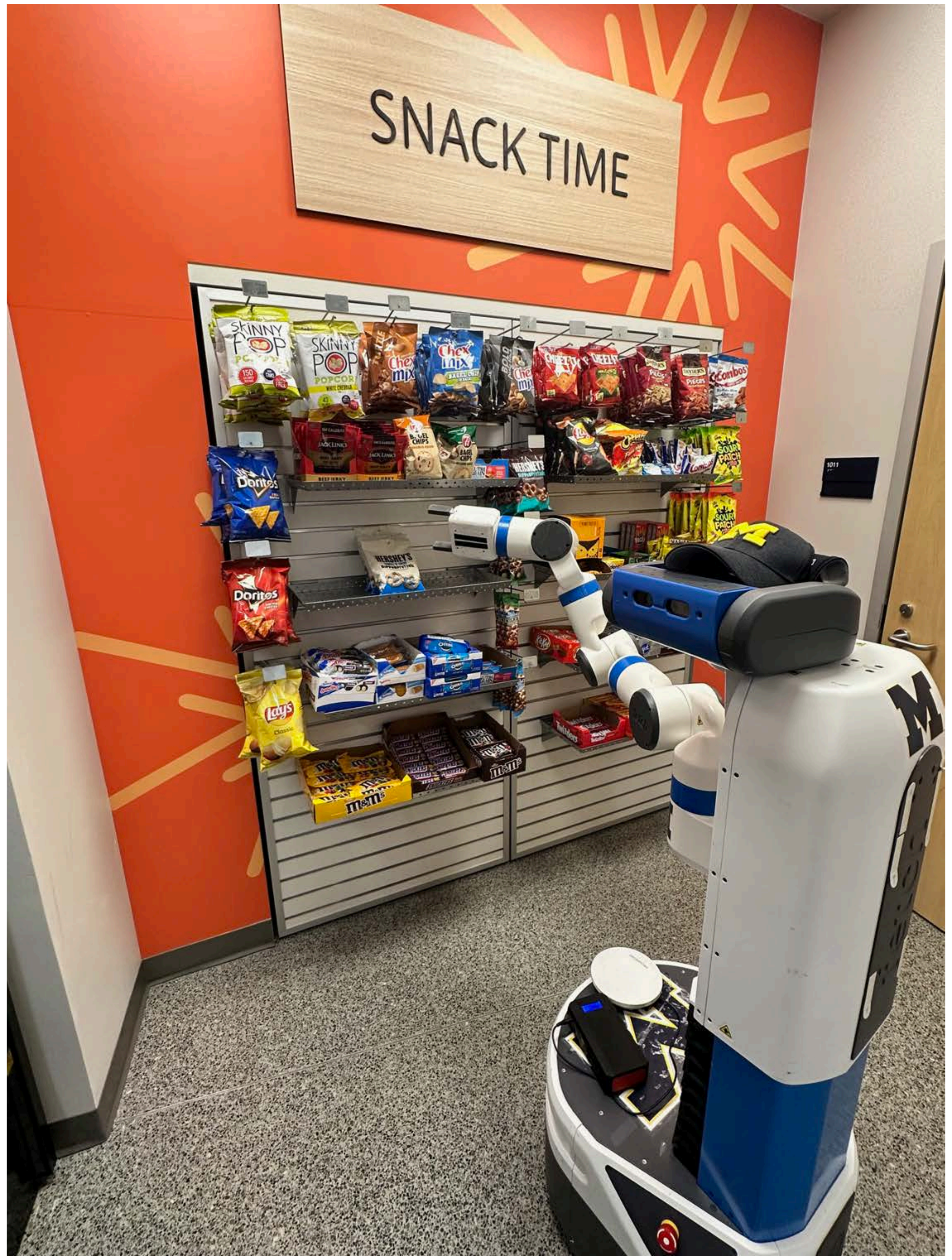
Name	Color	Storage Time (Days)	Sample
LV1	Breakers	21 ~ 28	
LV2	Turning	15 ~ 20	
LV3	Pink	7 ~ 14	
LV4	Light red	5 ~ 6	
LV5	Red	2 ~ 4	





Zhang et al., “Deep Learning Based Improved Classification System for Designing Tomato Harvesting Robot”, IEEE Access, 2016

Image Classification—Building Block for Other Tasks



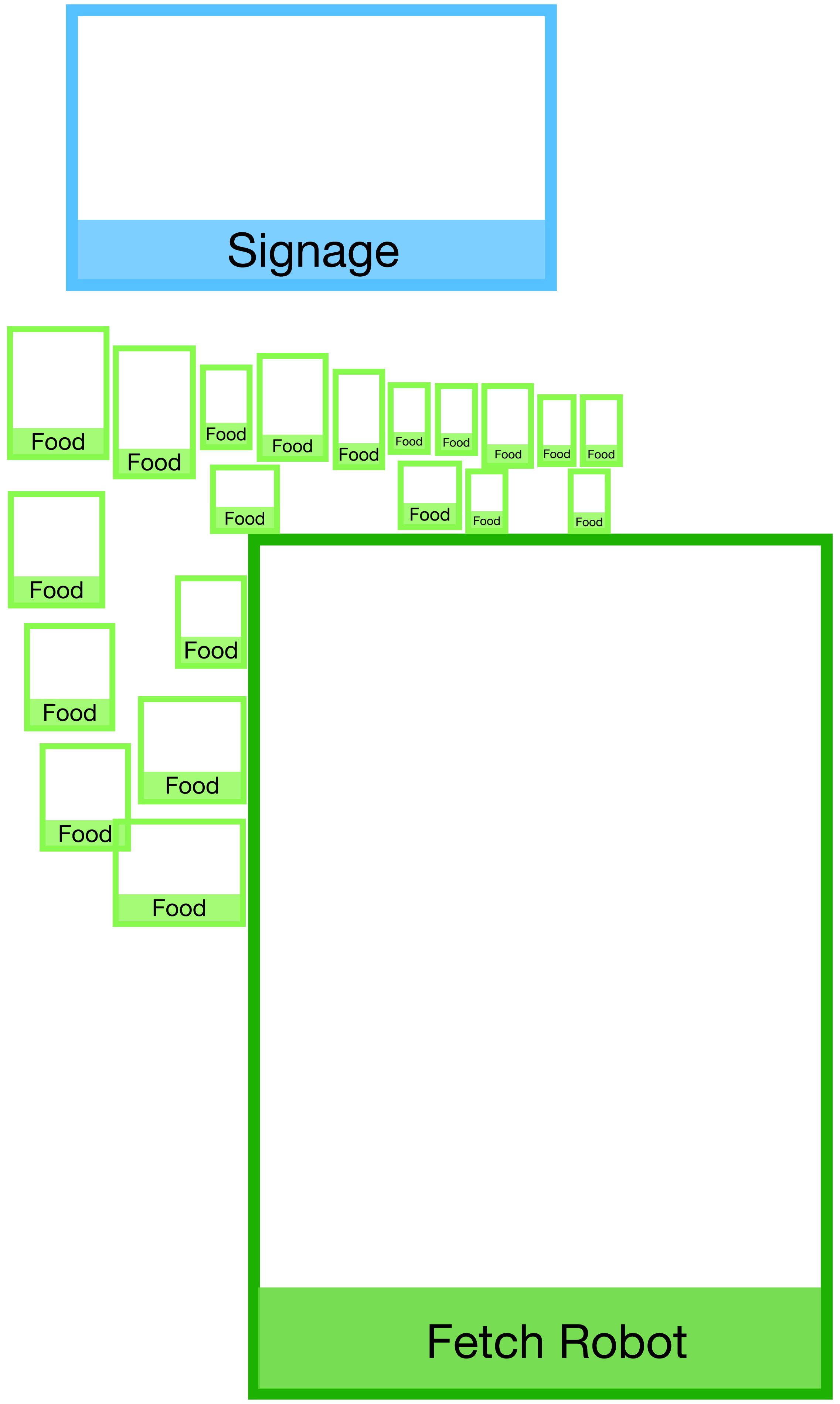
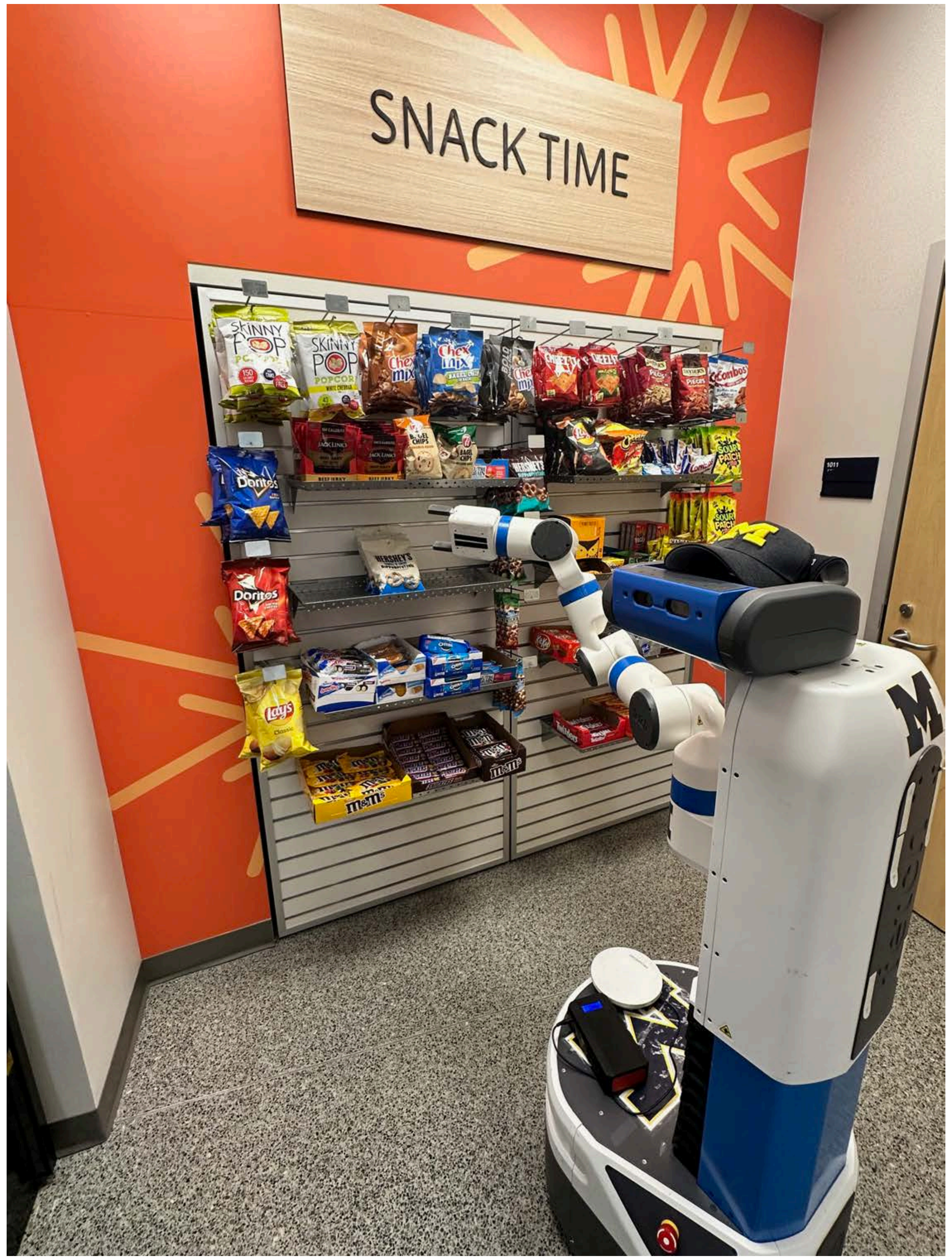


Image Classification—Building Block for Other Tasks



Example: Object Detection

Wall

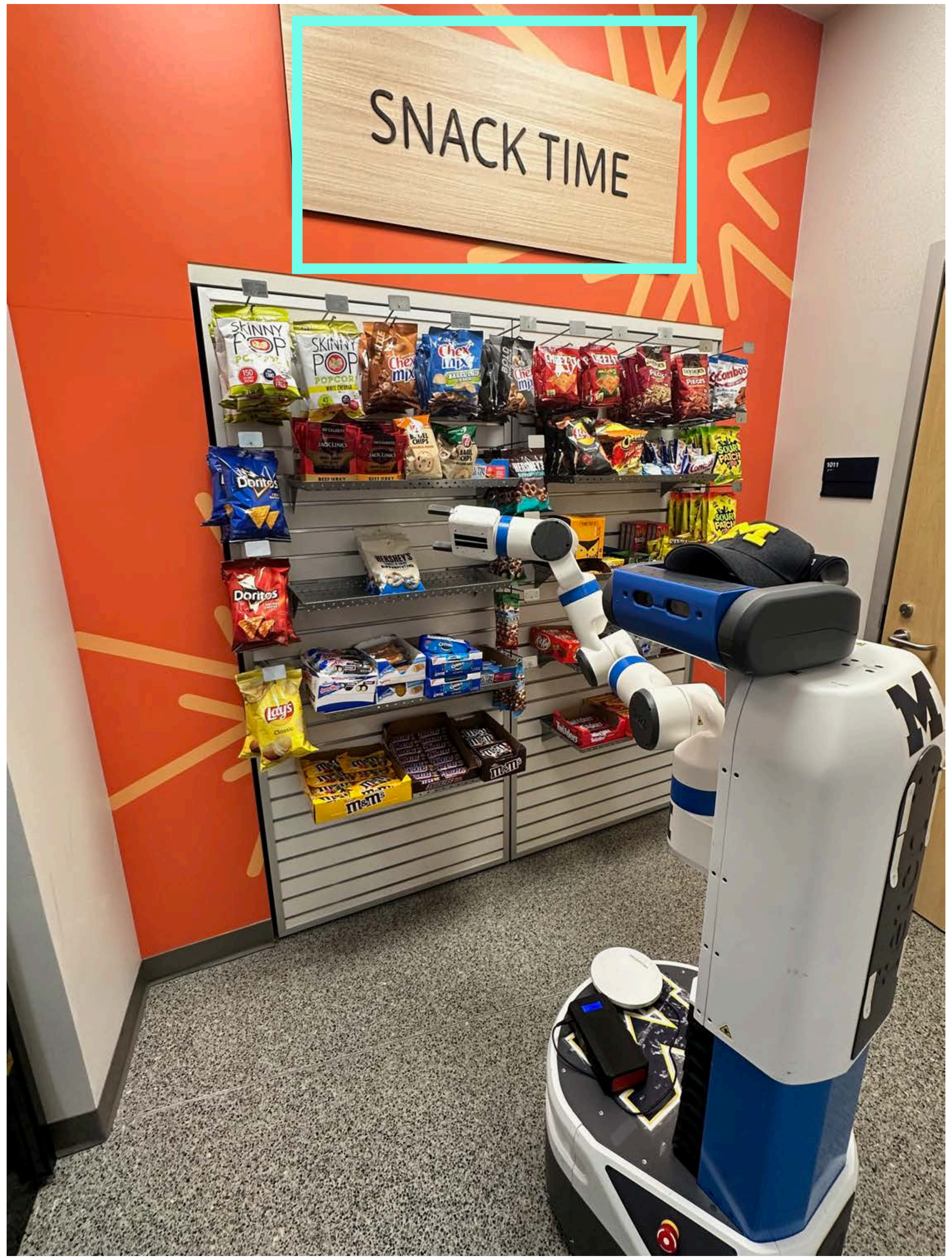
Floor

Signage

Fetch Robot

Snacks

Image Classification—Building Block for Other Tasks



Example: Object Detection

Wall

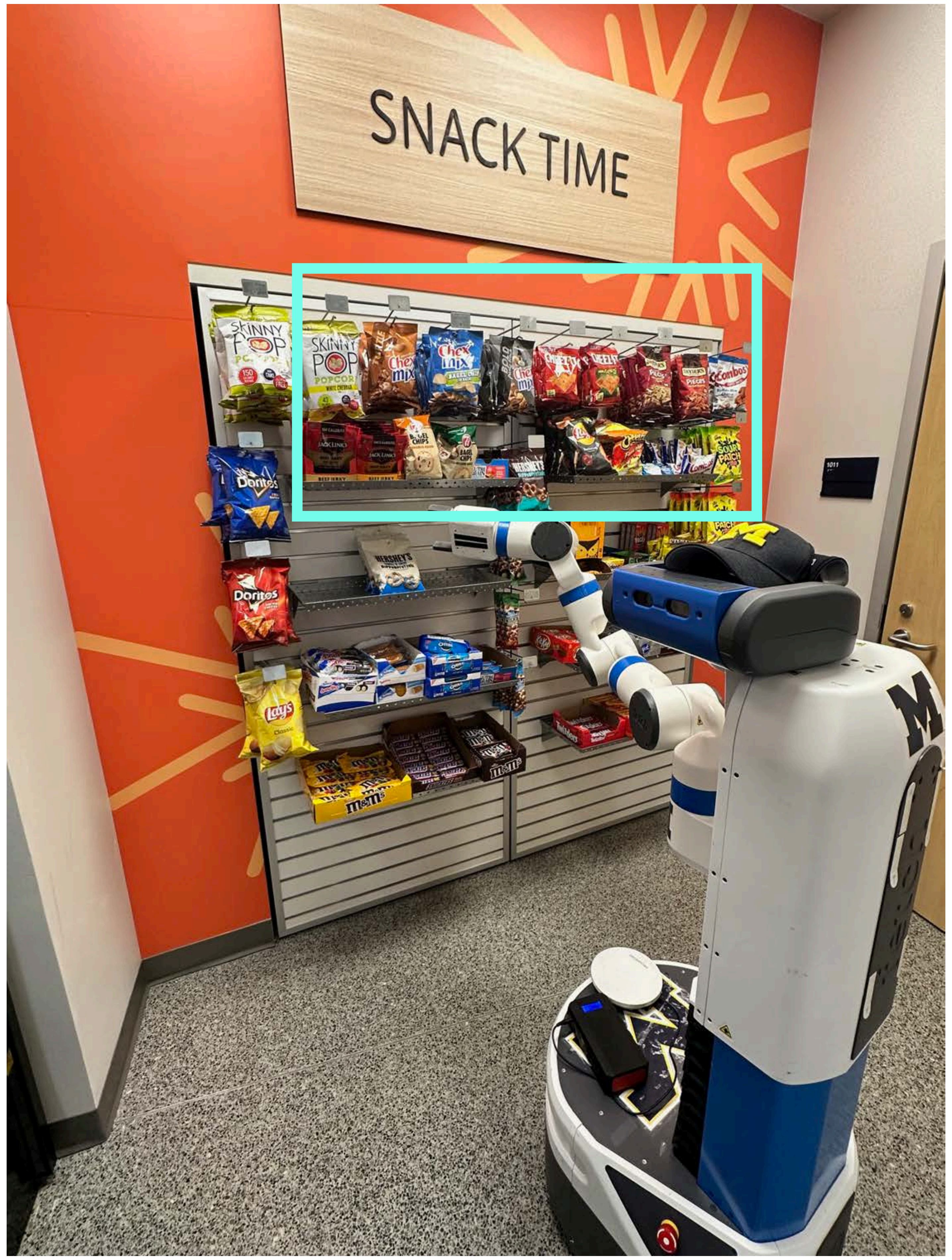
Floor

Signage

Fetch Robot

Snacks

Image Classification—Building Block for Other Tasks



Example: Object Detection

Wall

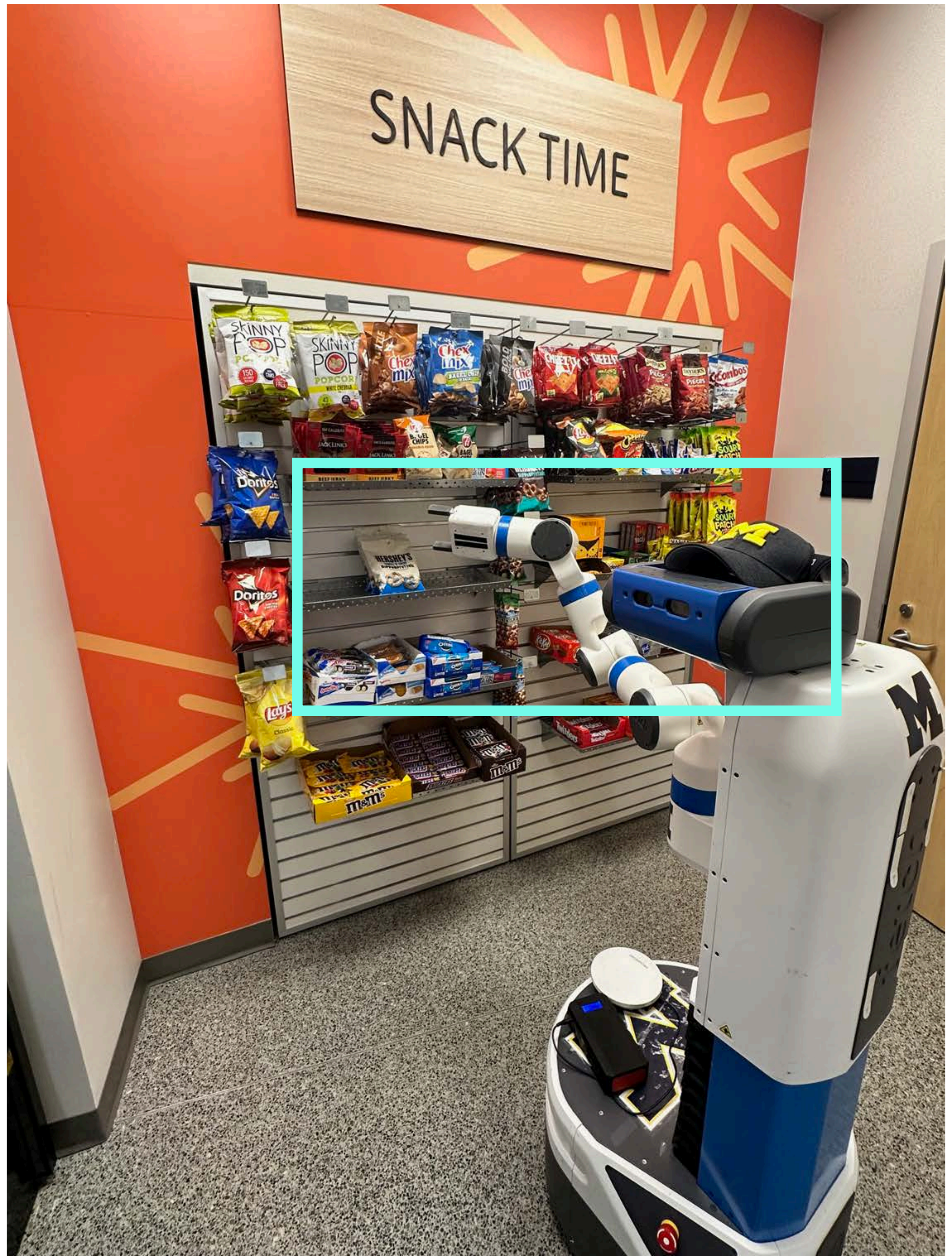
Floor

Signage

Fetch Robot

Snacks

Image Classification — Building Block for Other Tasks



Example: Object Detection

Wall

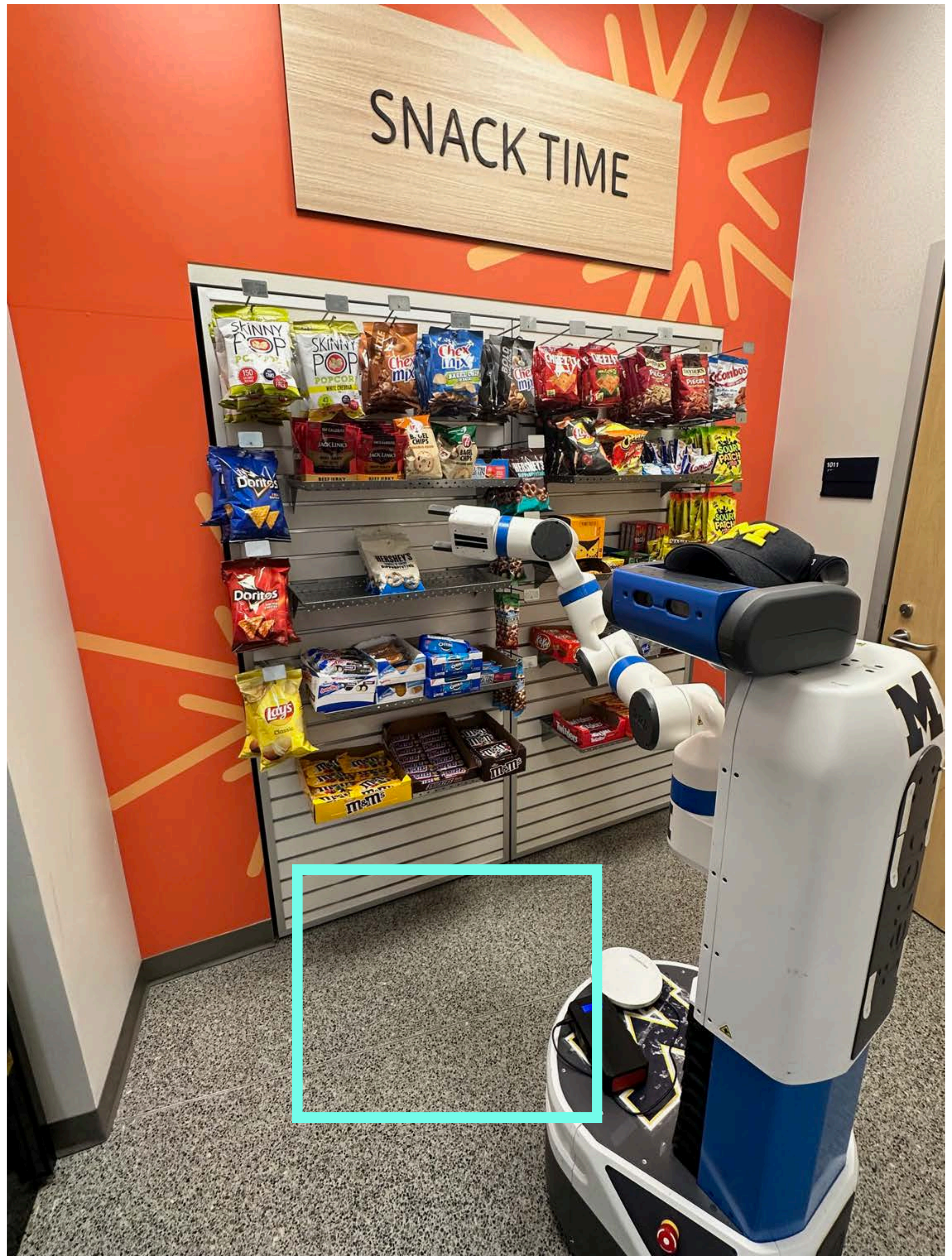
Floor

Signage

Fetch Robot

Snacks

Image Classification—Building Block for Other Tasks



Example: Object Detection

Wall

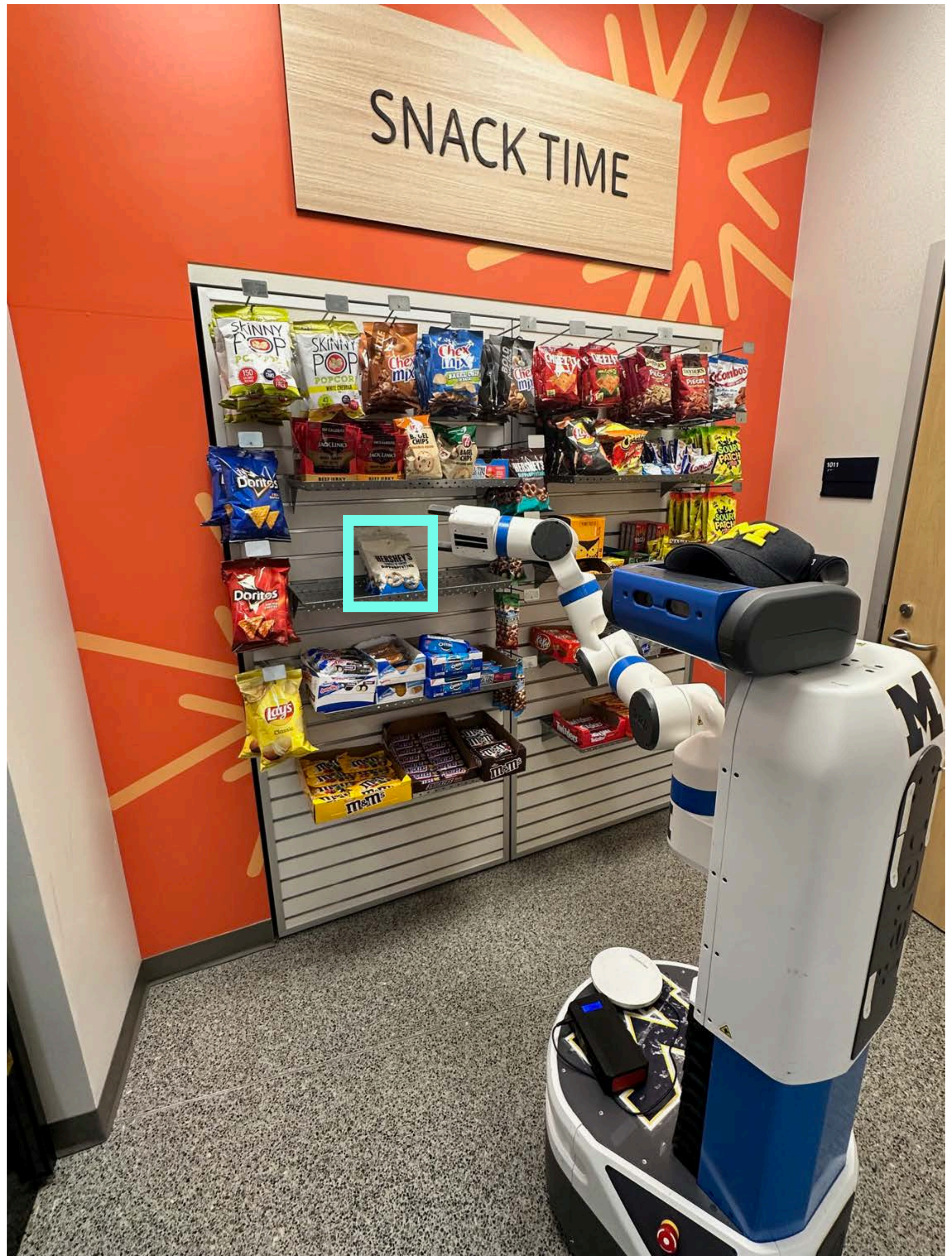
Floor

Signage

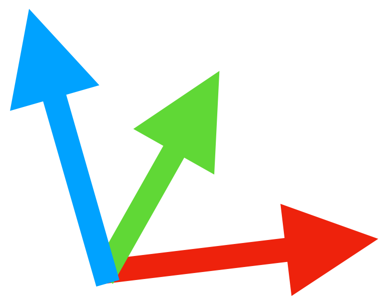
Fetch Robot

Snacks

Image Classification—Building Block for Other Tasks



Example: Pose Estimation



An Image Classifier

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

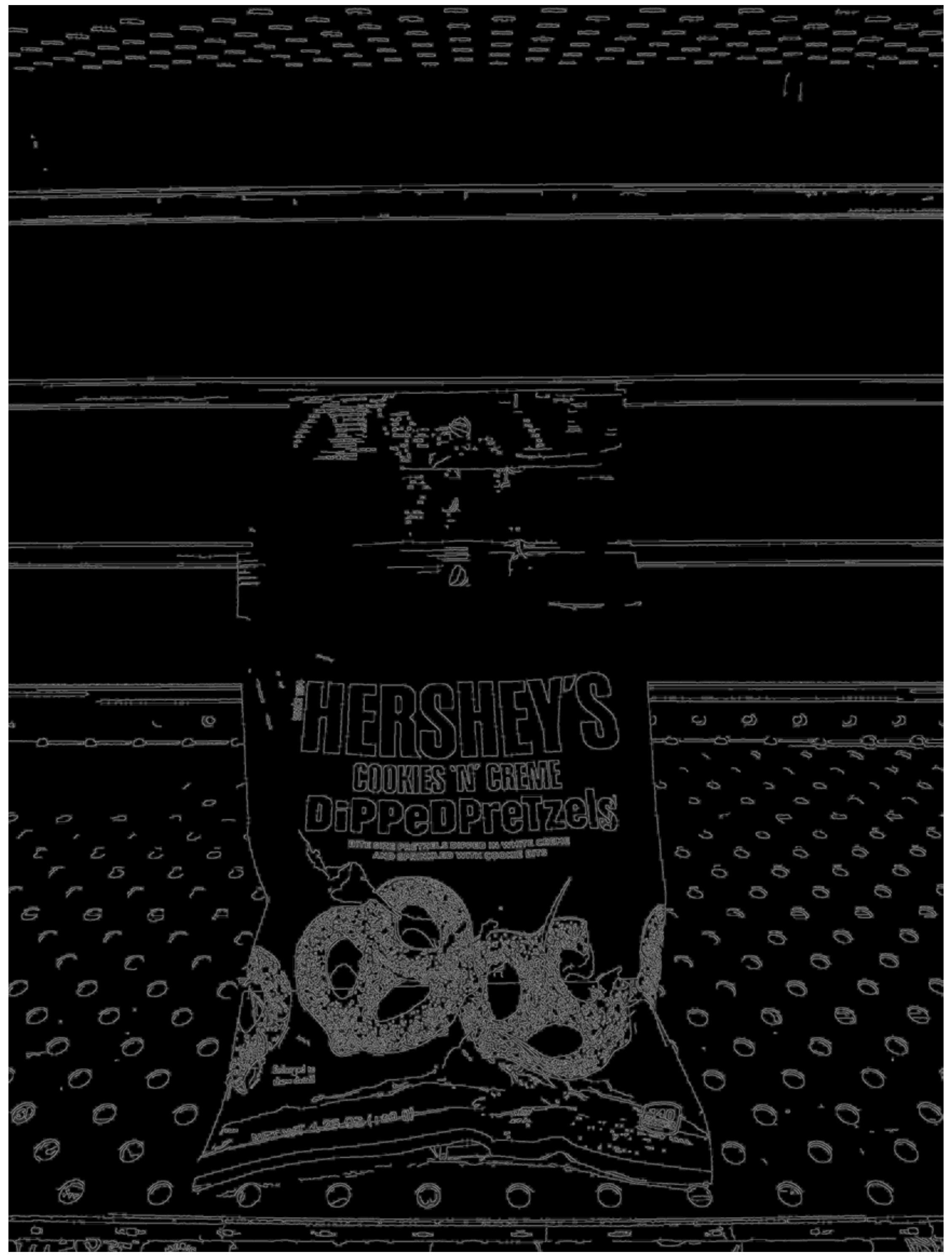
Unlike well defined programming (e.g. sorting a list)

No obvious way to hard-code the algorithm
for recognizing each class

An Image Classifier

Input: image

Detect: Edges



Detect: Corners

???

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

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

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

Example training set

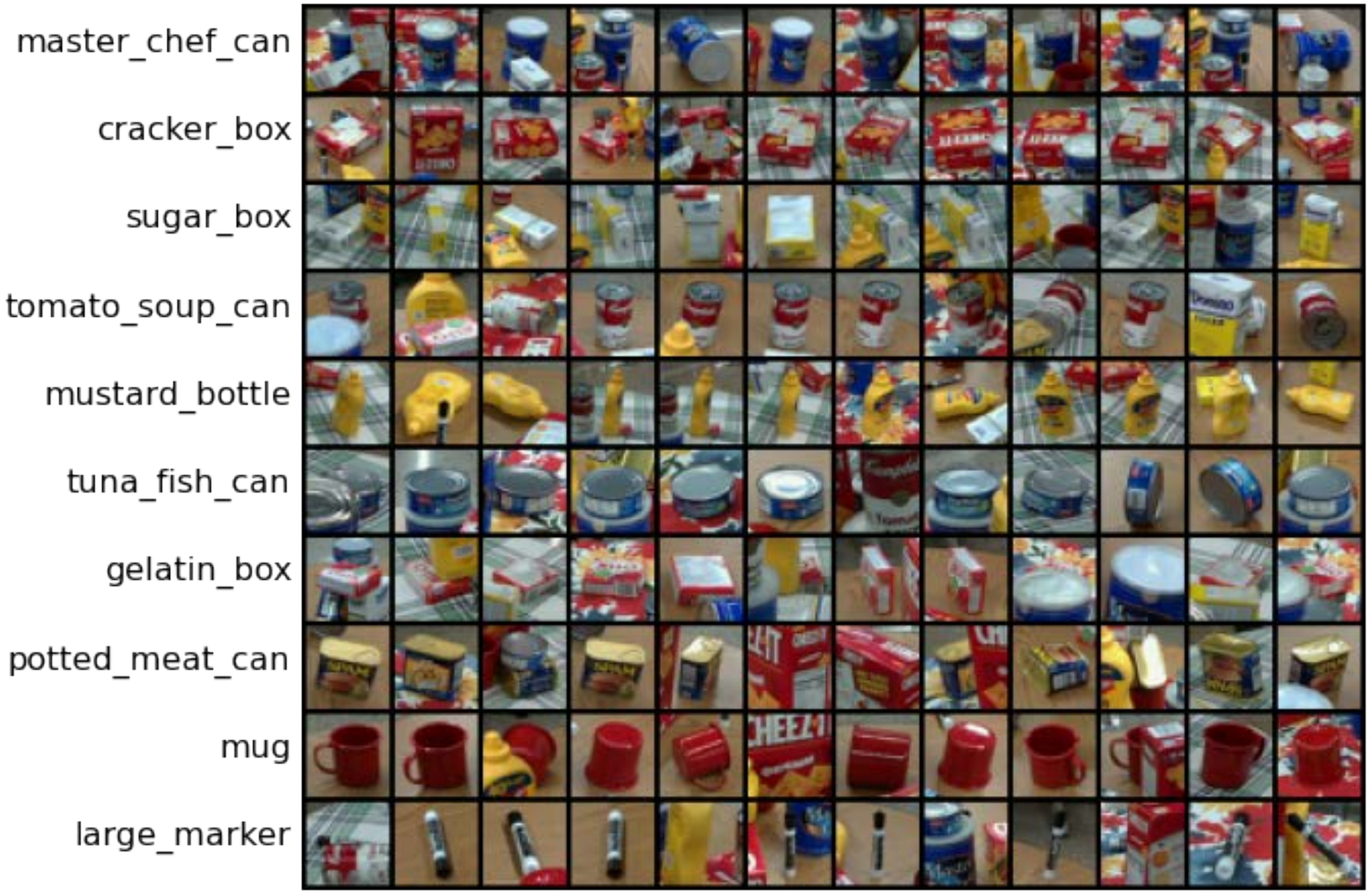


Image Classification Datasets—MNIST

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

Due to relatively small size, results on MNIST often do not hold on more complex datasets

Image Classification Datasets—CIFAR10

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

10 classes
32x32 RGB images
50k training images (5k per class)
10k test images (1k per class)

Image Classification Datasets—CIFAR100

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

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

20 superclasses with 5 classes each:

Aquatic mammals: beaver, dolphin, otter, seal, whale

Trees: maple, oak, palm, pine, willow

Image Classification Datasets—ImageNet

Deng et al., "ImageNet: A Large-Scale Hierarchical Image Database", CVPR, 2009.
Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge", IJCV, 2015.

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 must be right

Image Classification Datasets—ImageNet

Deng et al., "ImageNet: A Large-Scale Hierarchical Image Database", CVPR, 2009.
Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge", IJCV, 2015.

1000 classes

~1.3M training images (~1.3K per class)
50k validation images (50 per class)
100K test images (100 per class)
test labels are secret!

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

There is also a 22K category version of ImageNet, but less commonly used

Image Classification Datasets—MIT Places

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

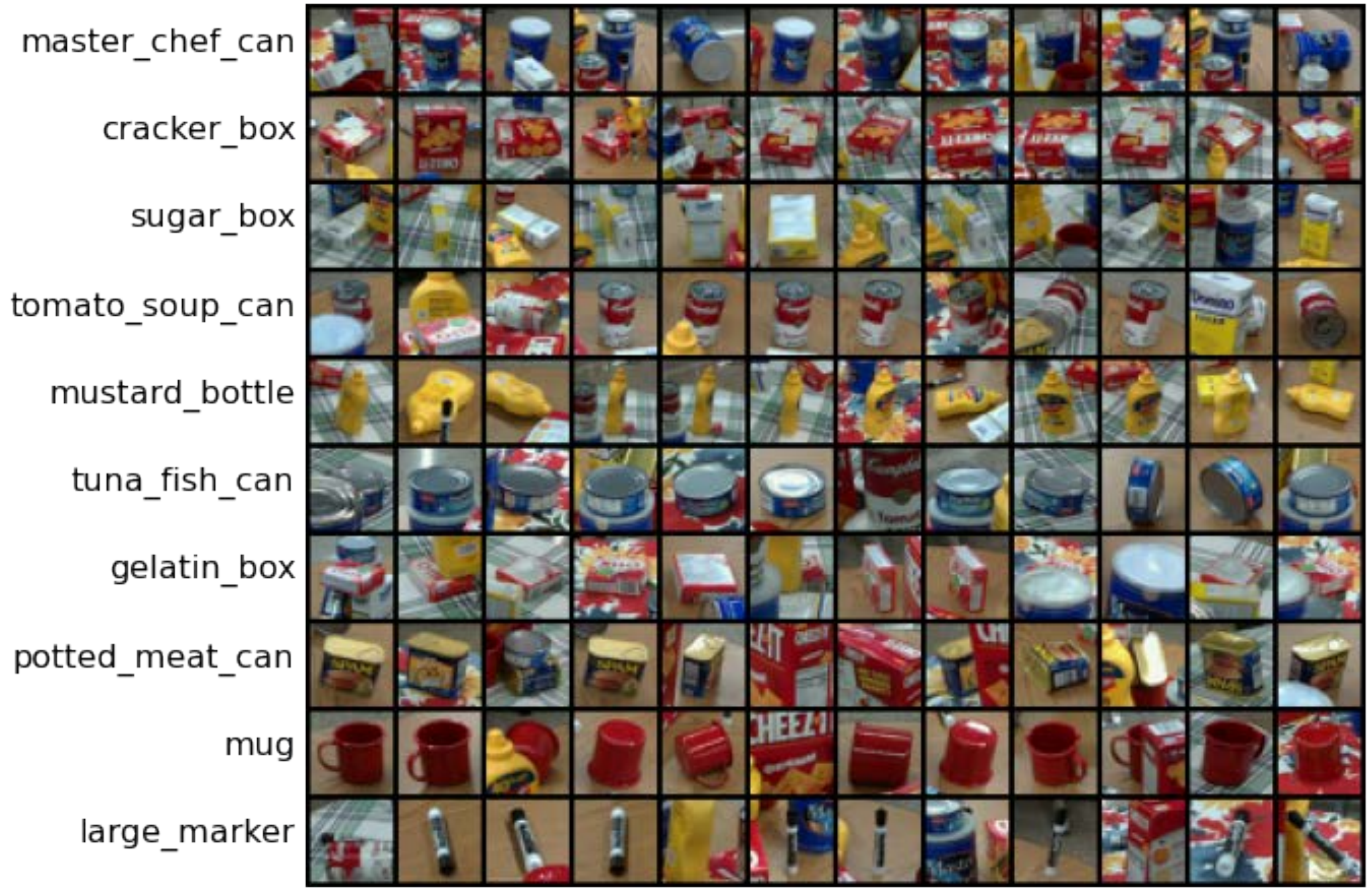
365 classes of different scene types

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

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

Image Classification Datasets—PROPS

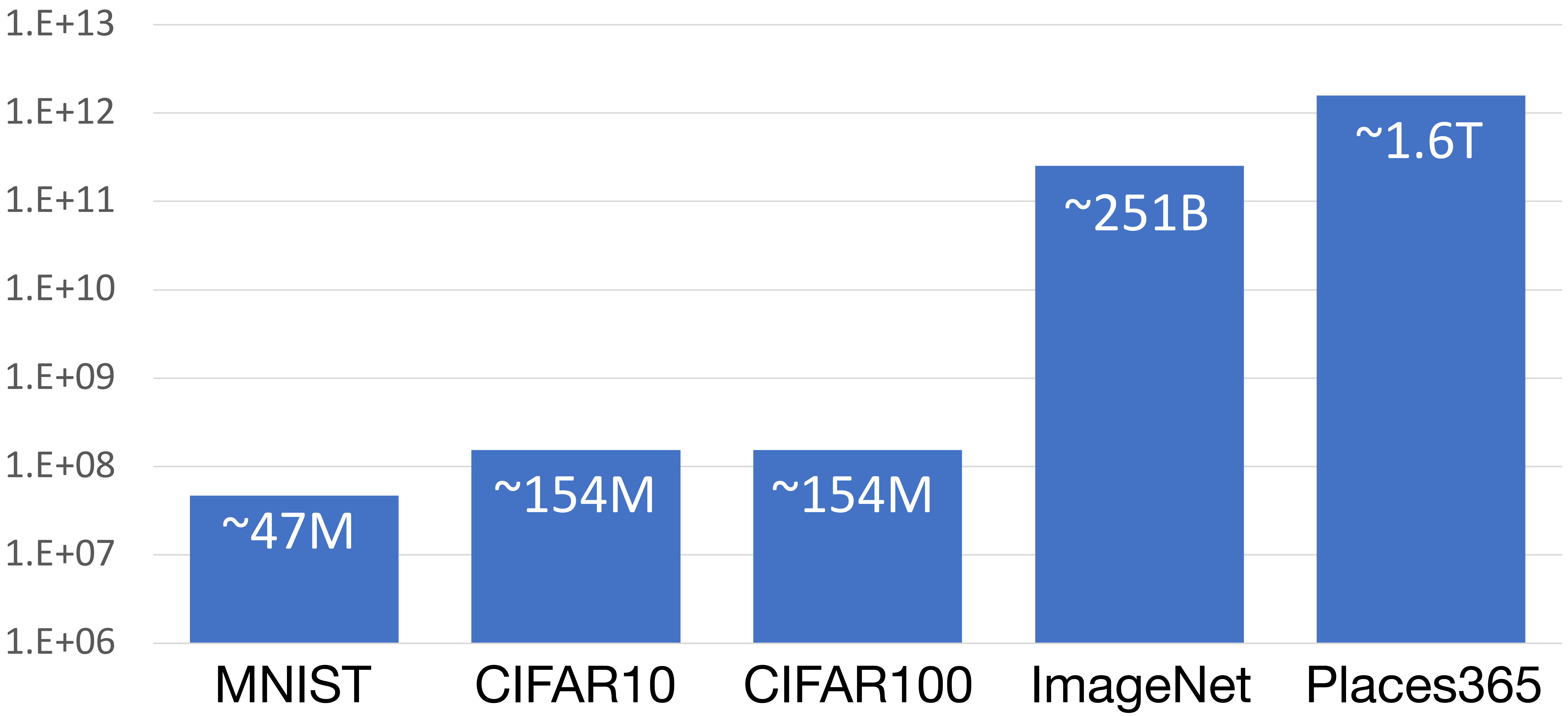
Progress **R**obot **O**bject **P**erception **S**amples **D**ataset



Chen et al., "ProgressLabeller: Visual Data Stream Annotation for Training Object-Centric 3D Perception", IROS, 2022.

10 classes
32x32 RGB images
50k training images (5k per class)
10k test images (1k per class)

Classification Datasets—Number of Training Pixels



PROPS

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

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

$\xrightarrow{\text{add}}$ 456

Nearest Neighbor Classifier

```
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

```
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

Nearest Neighbor Classifier

```
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 is training?

A: O(1)

Nearest Neighbor Classifier

```
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 is training?

A: O(1)

Q: With N examples how fast is testing?

A: O(N)

Nearest Neighbor Classifier

```
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 is training?

A: O(1)

Q: With N examples how fast is testing?

A: O(N)

This is a problem: we can train slow offline but need fast testing!

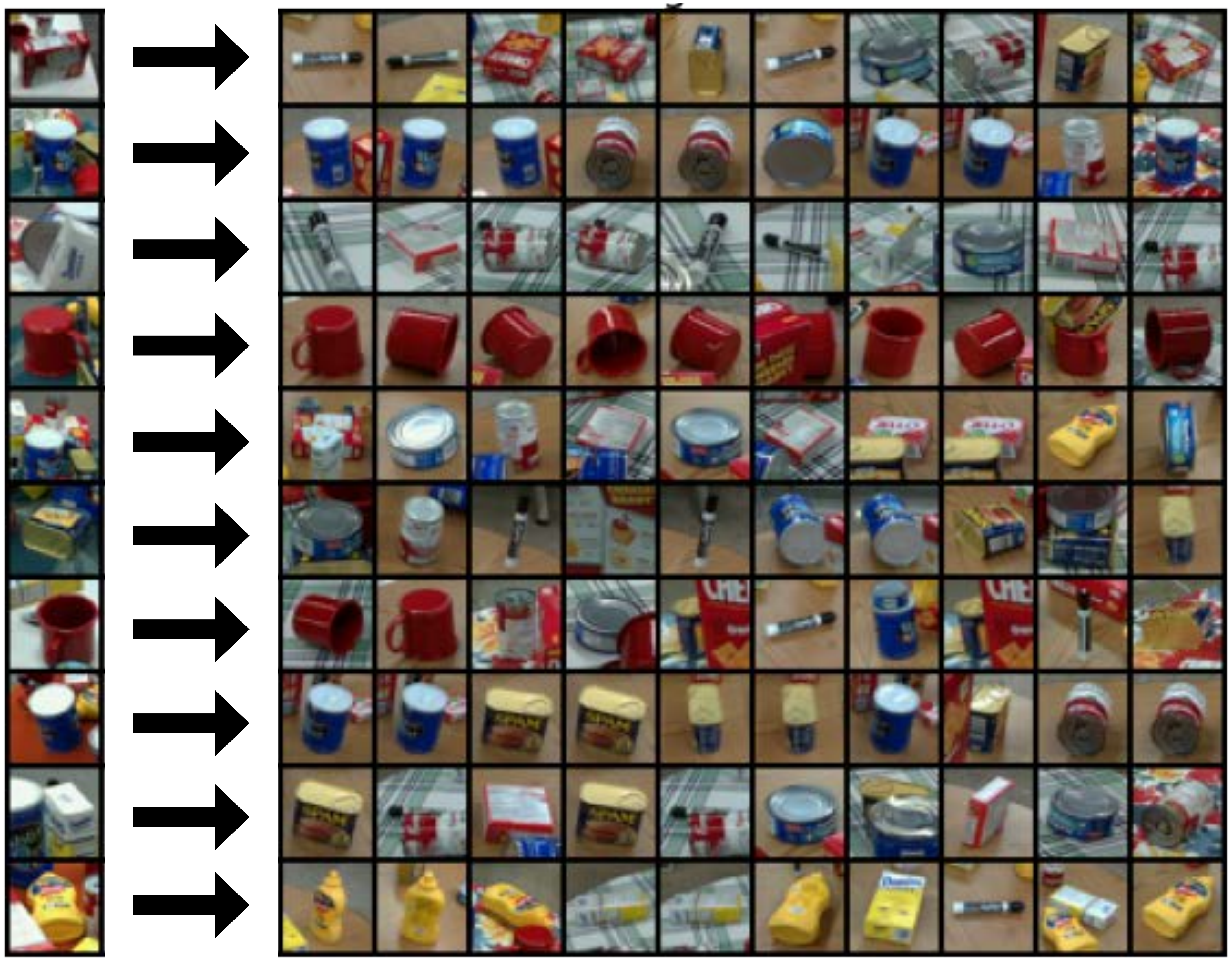
Nearest Neighbor Classifier

```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
```

There are many methods for fast / approximate nearest neighbors

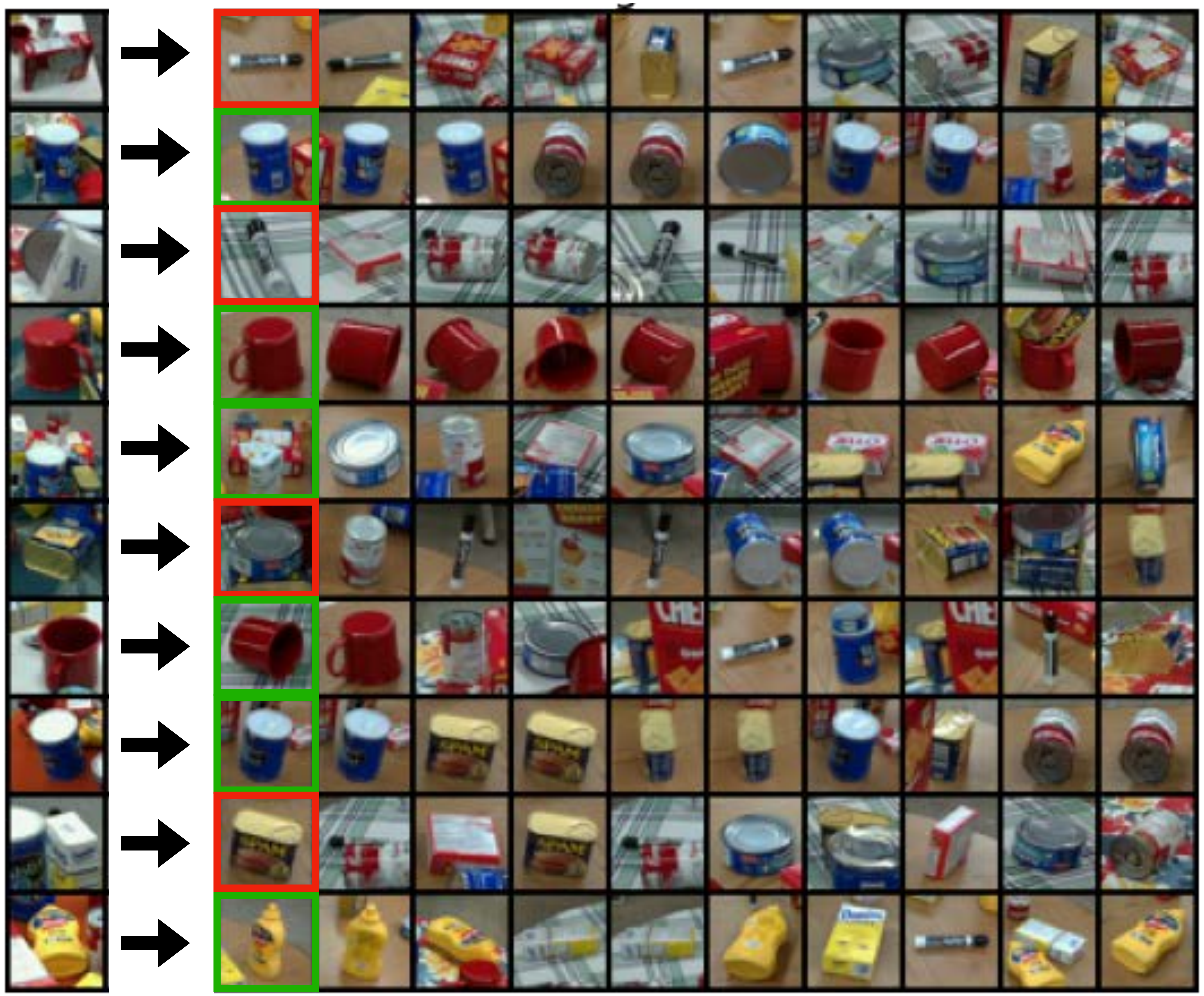
e.g. github.com/facebookresearch/faiss

What does this look like?

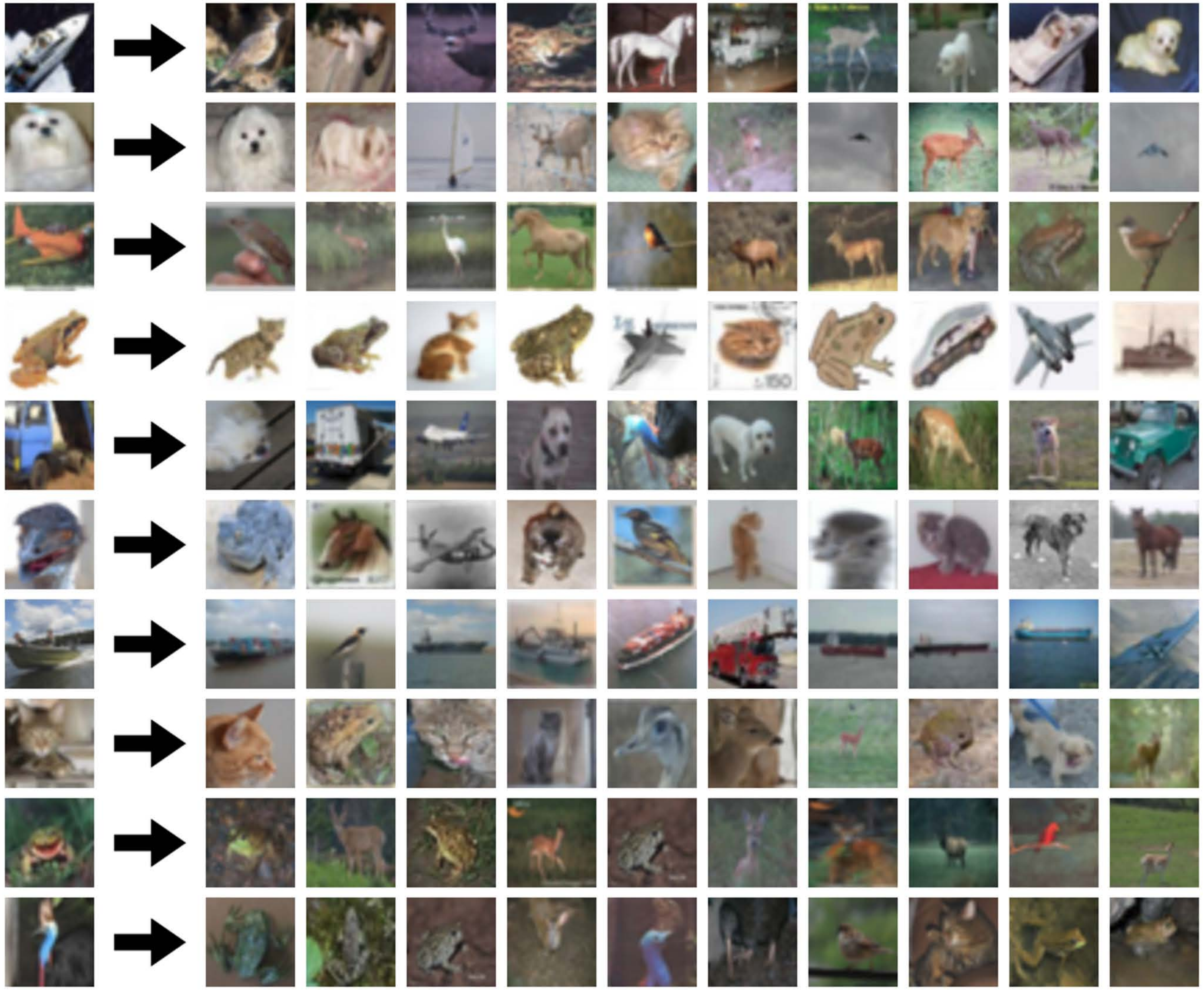


What does this look like?

PROPS dataset is instance-level



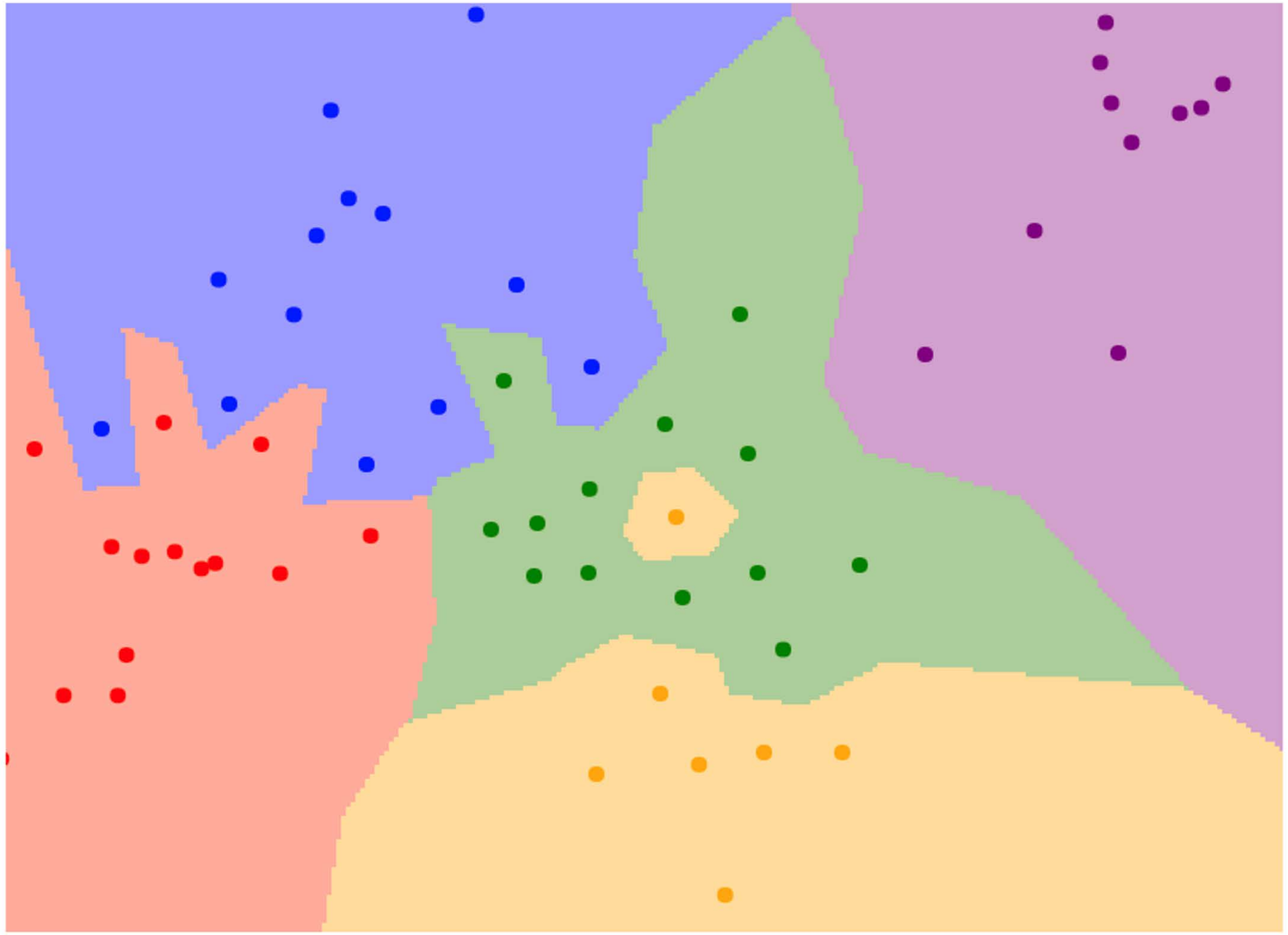
What does this look like?



What does this look like?

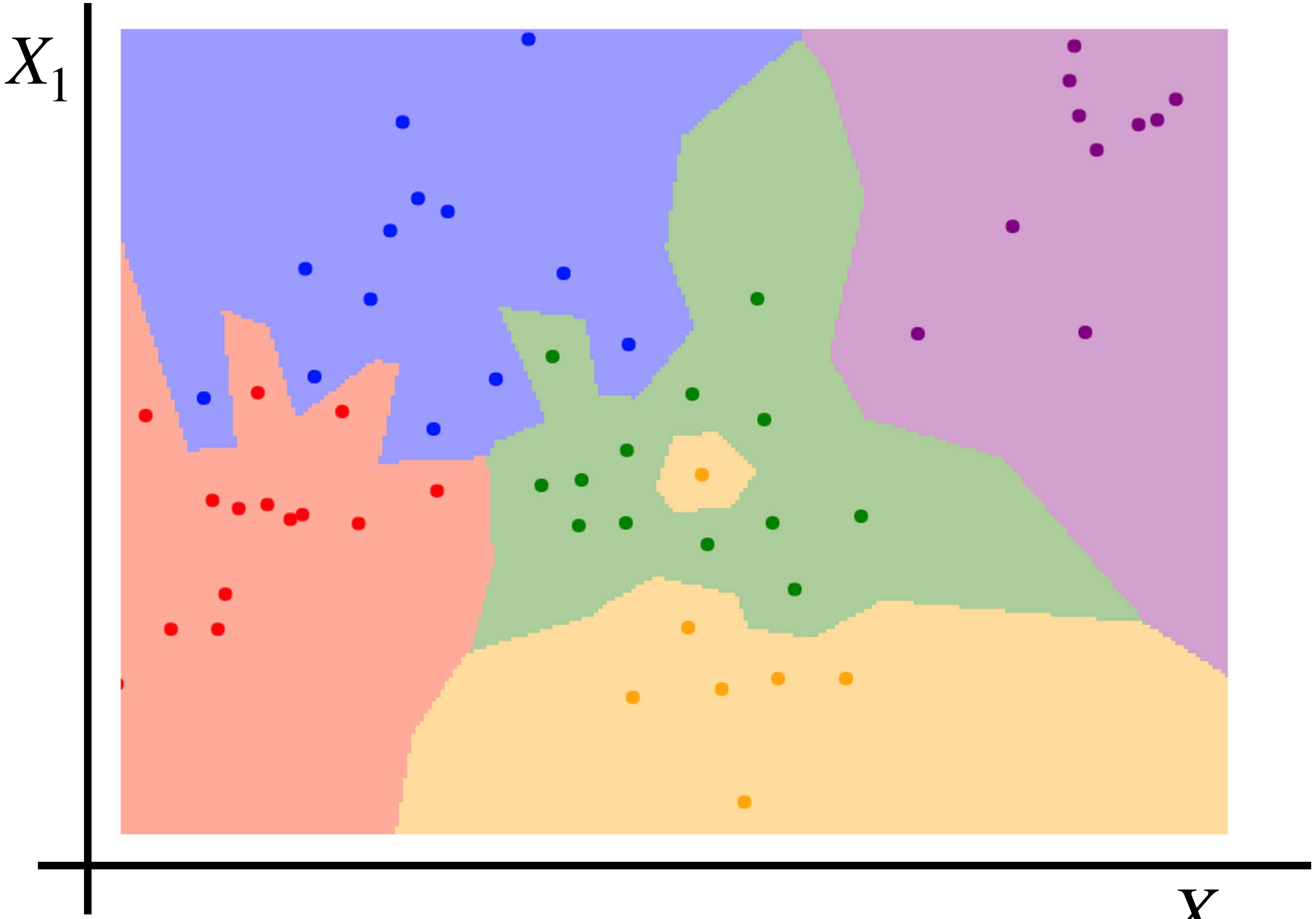
CIFAR10 dataset is category-level

K-Nearest Neighbors Decision Boundaries



K-Nearest Neighbors Decision Boundaries

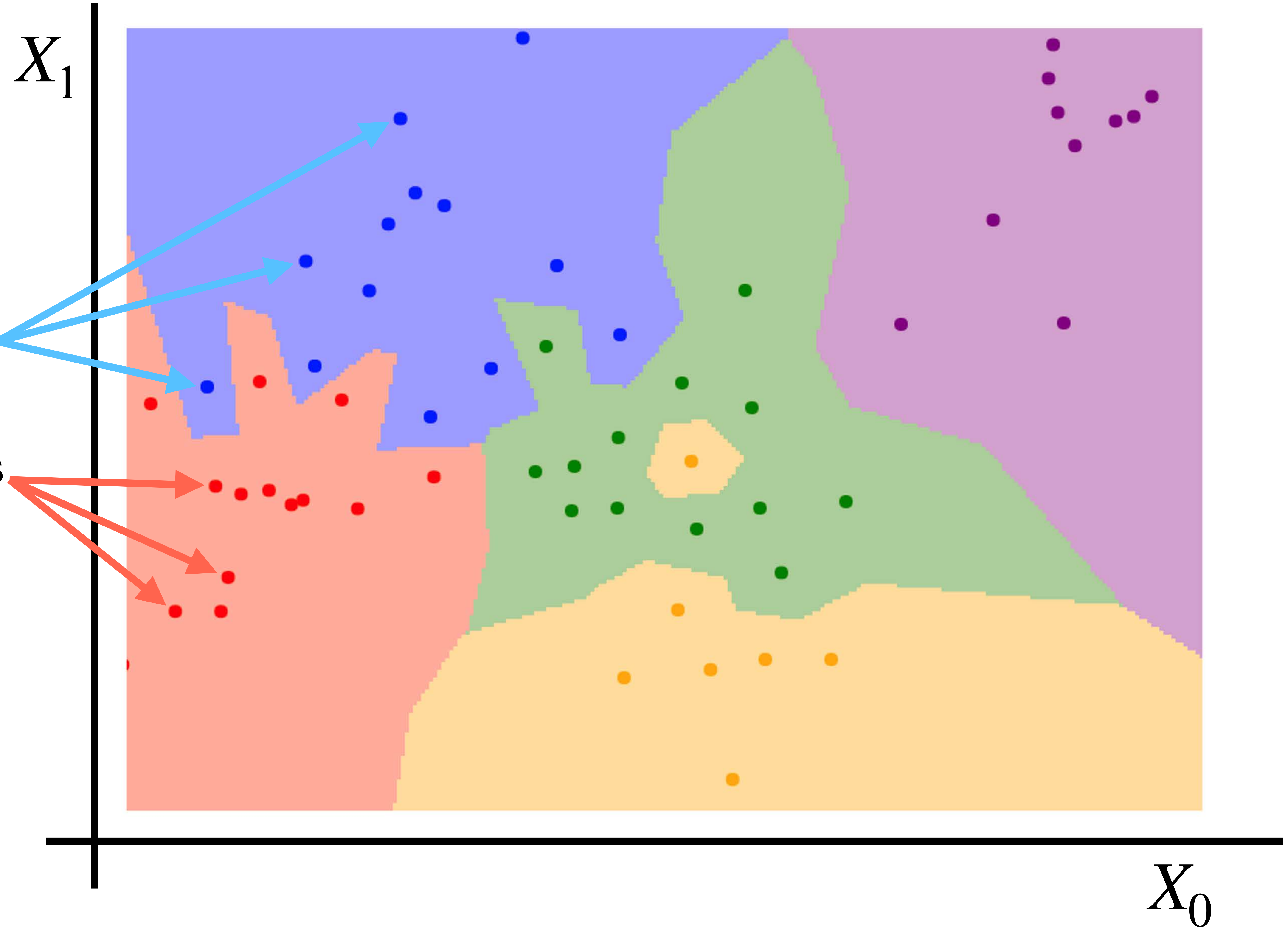
Nearest neighbors in two dimensions



K-Nearest Neighbors Decision Boundaries

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

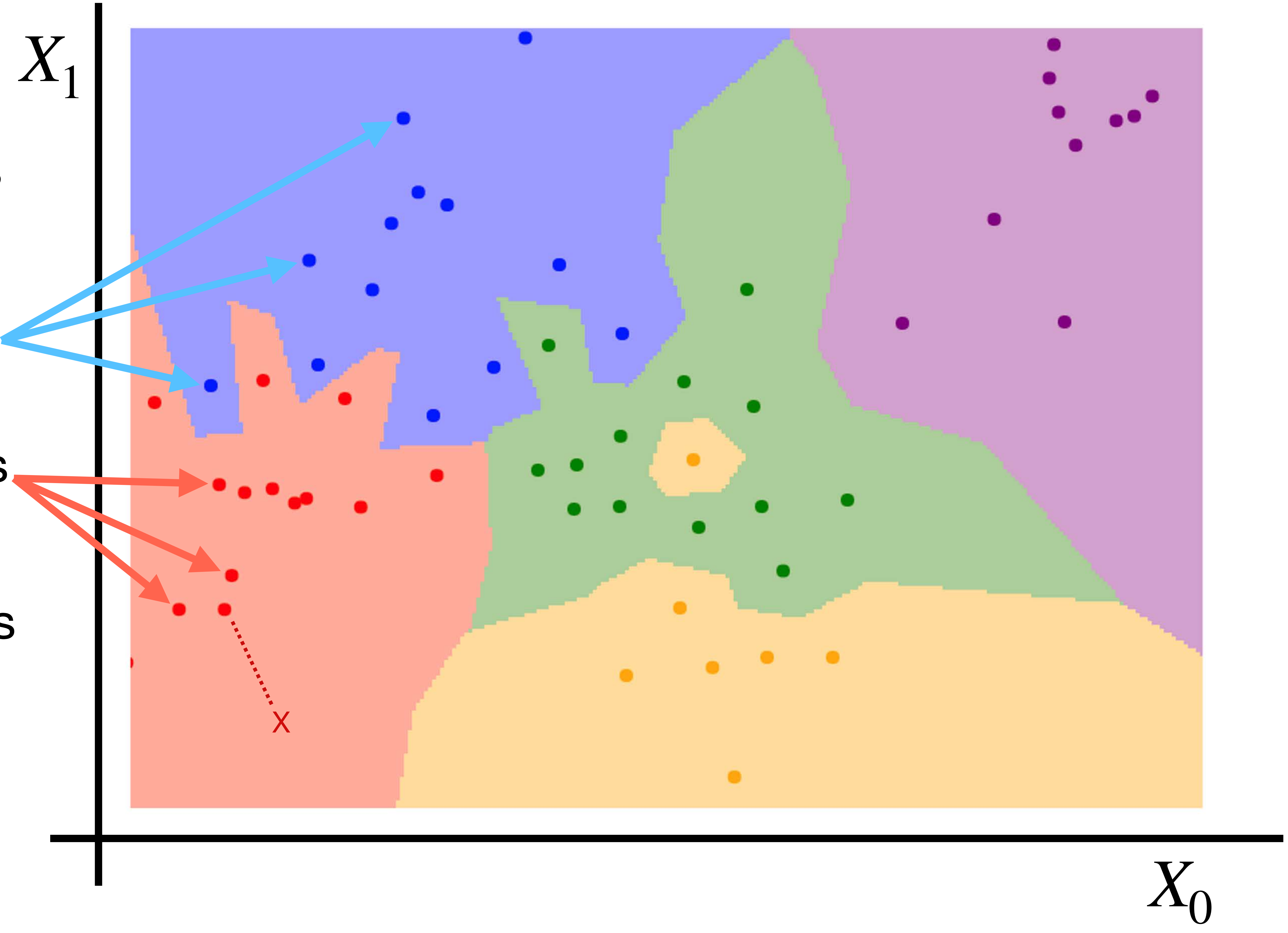


K-Nearest Neighbors Decision Boundaries

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned

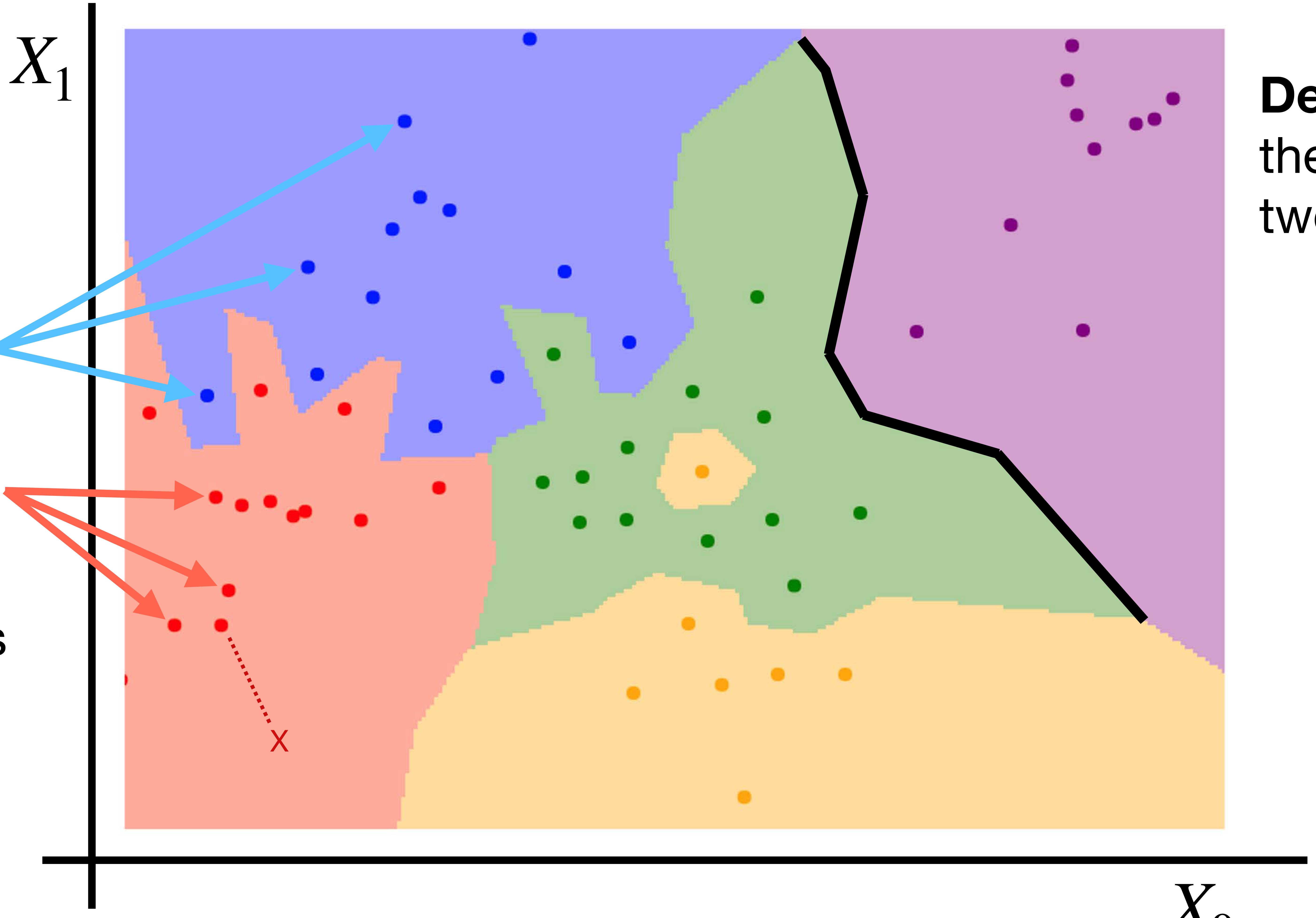


K-Nearest Neighbors Decision Boundaries

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



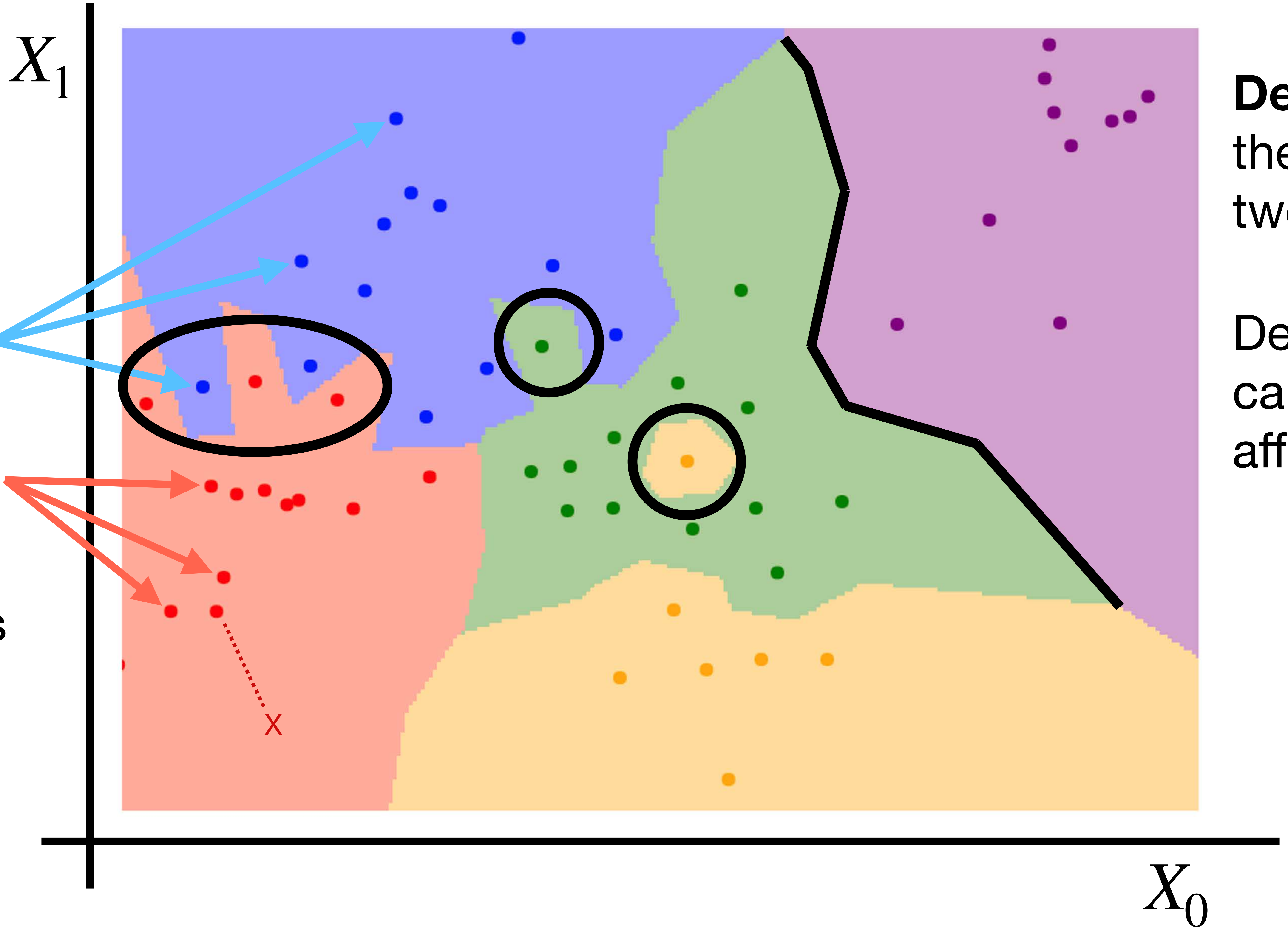
Decision boundary is the boundary between two classification regions

K-Nearest Neighbors 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

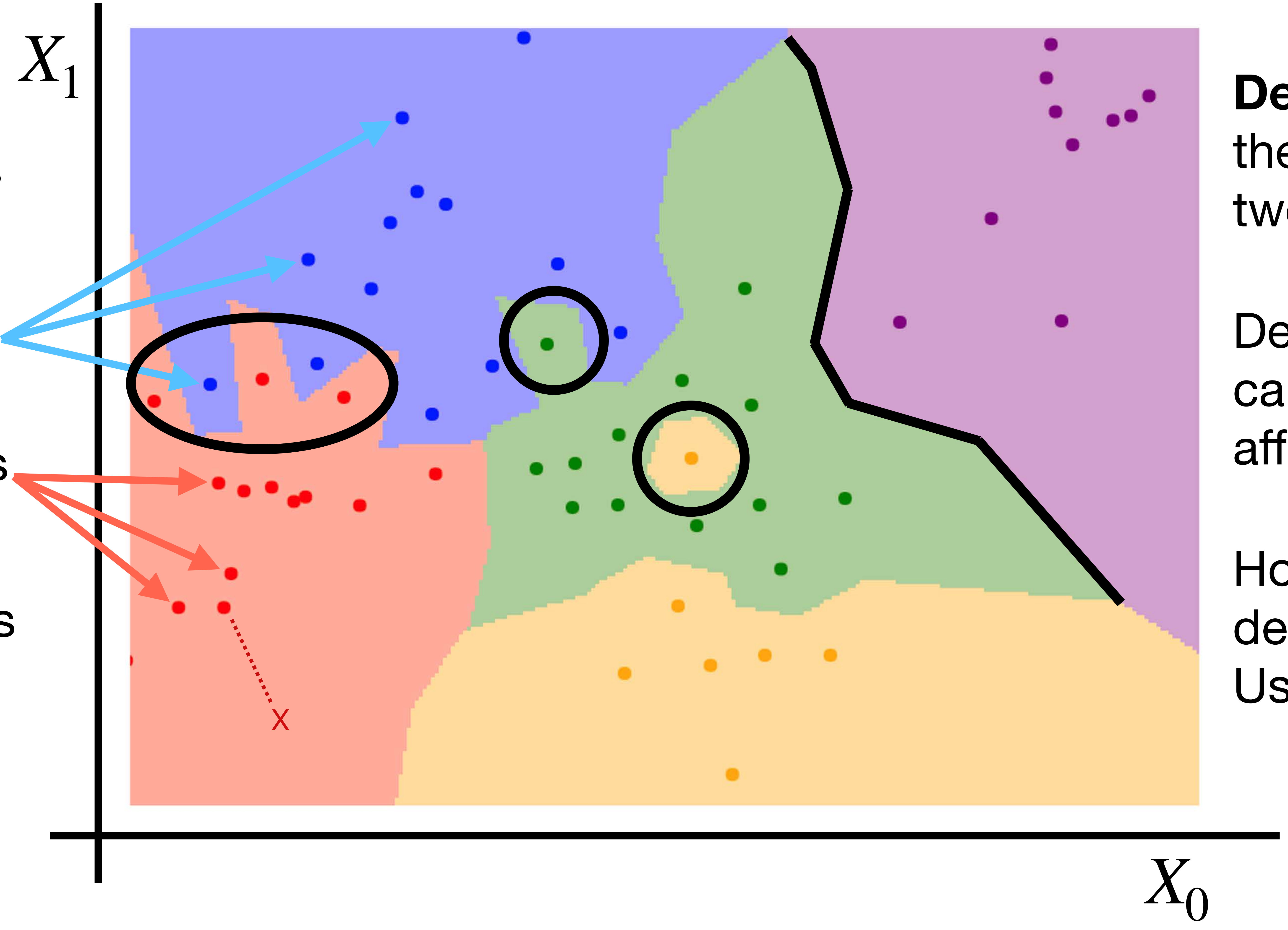
Decision boundaries can be noisy; affected by outliers

K-Nearest Neighbors 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



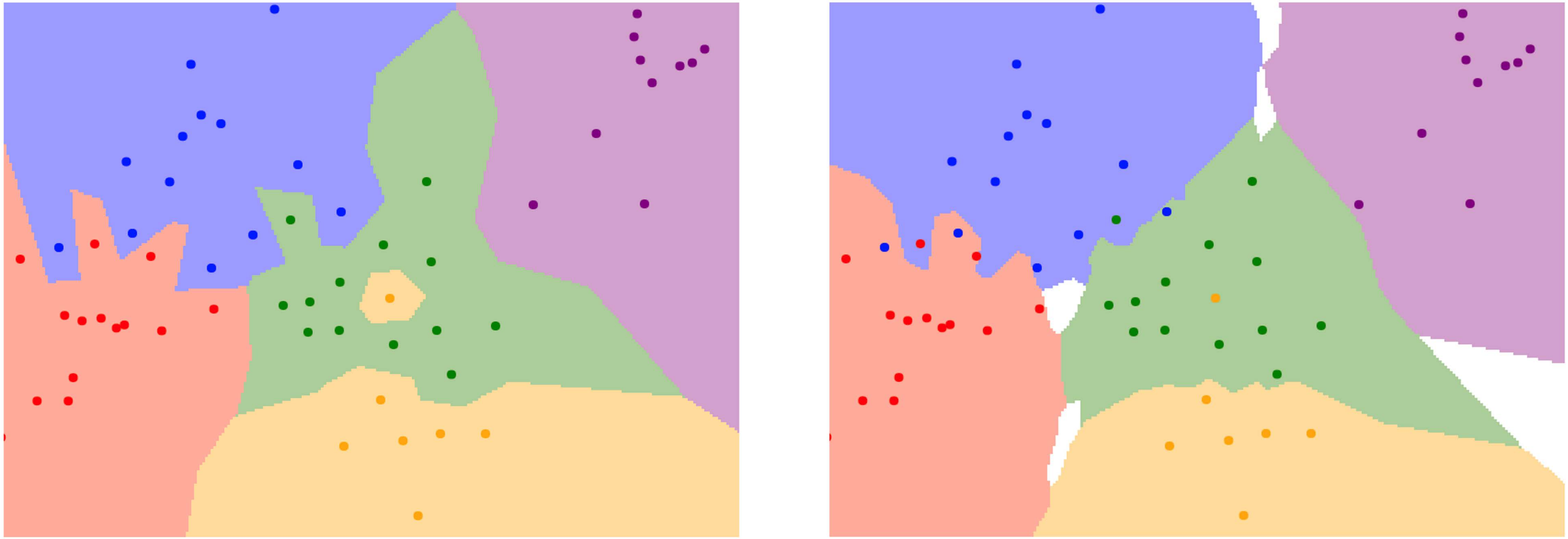
Decision boundary is the boundary between two classification regions

Decision boundaries can be noisy; affected by outliers

How to smooth the decision boundaries? Use more neighbors!

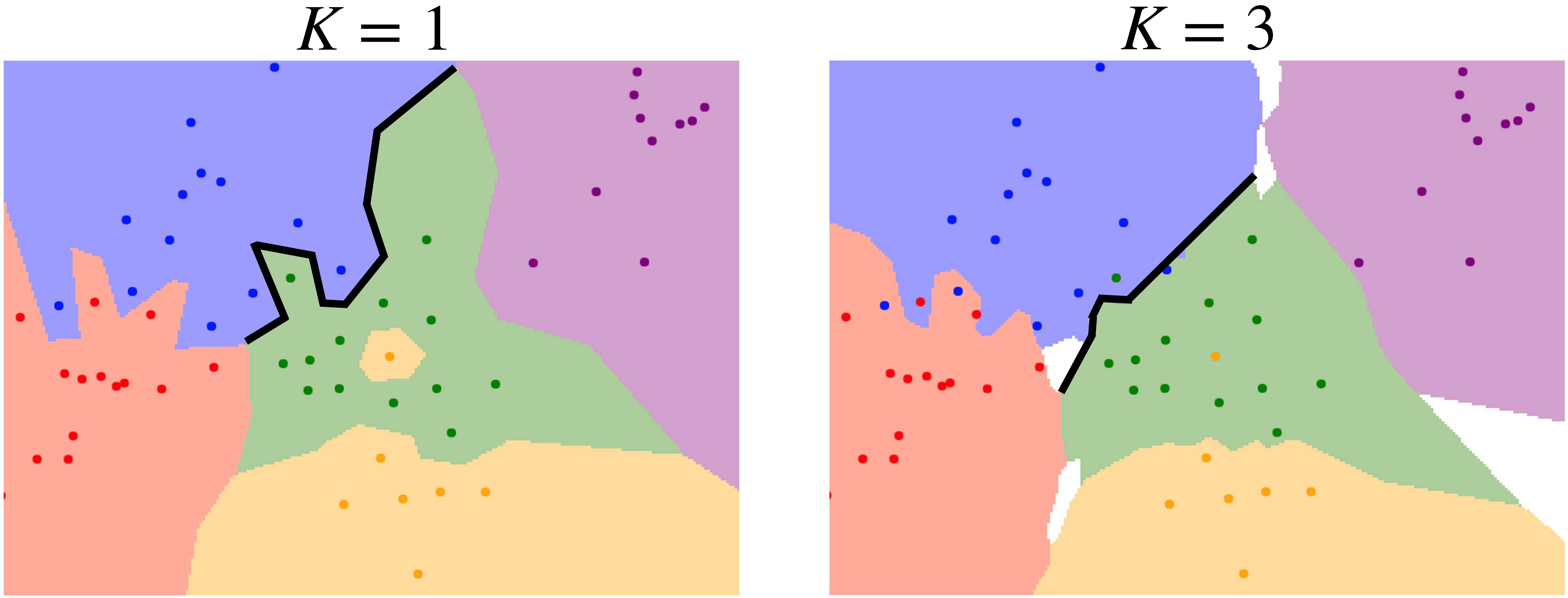
K-Nearest Neighbors Classification

$K = 1$ $K = 3$



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

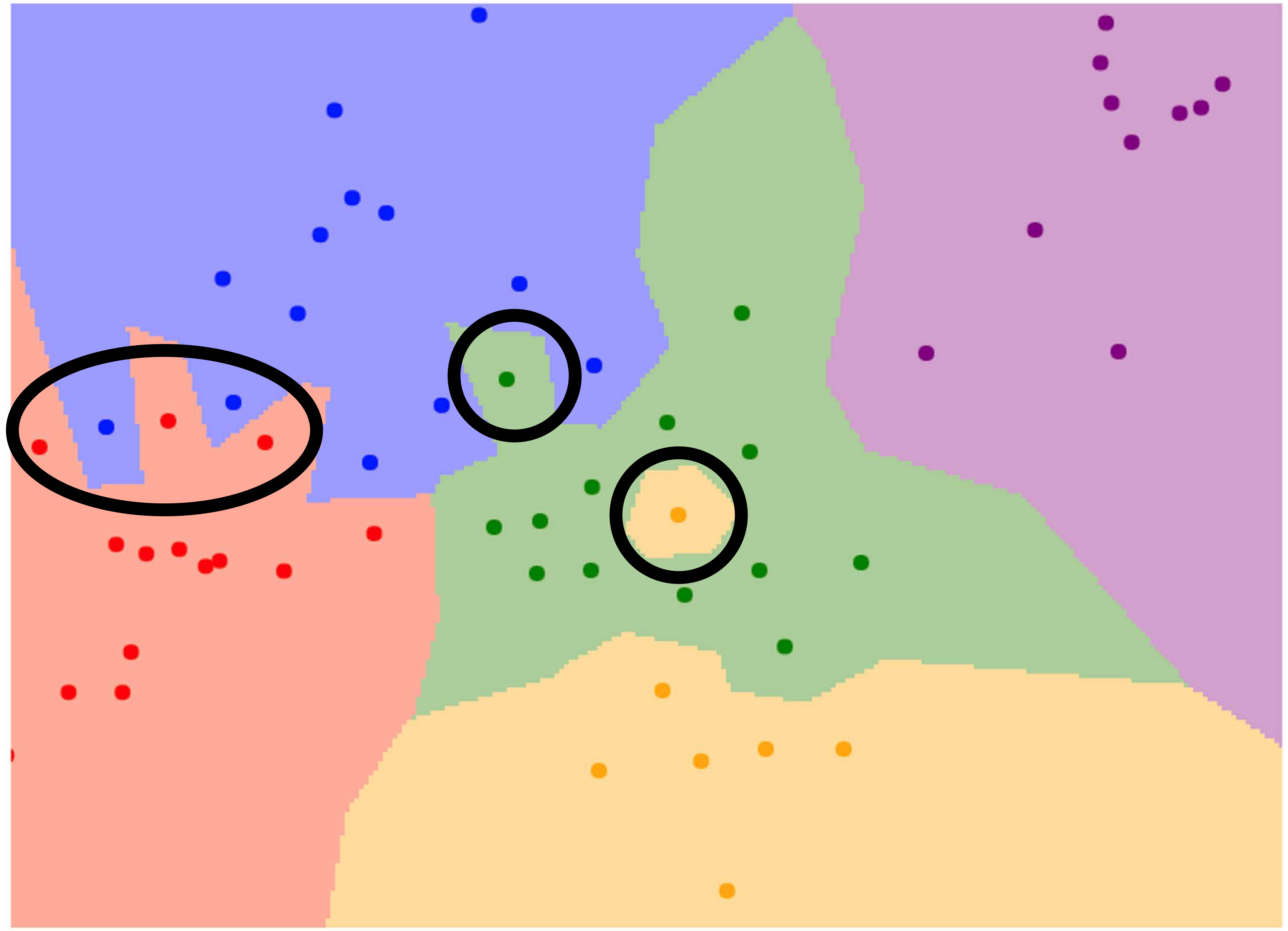
K-Nearest Neighbors Classification



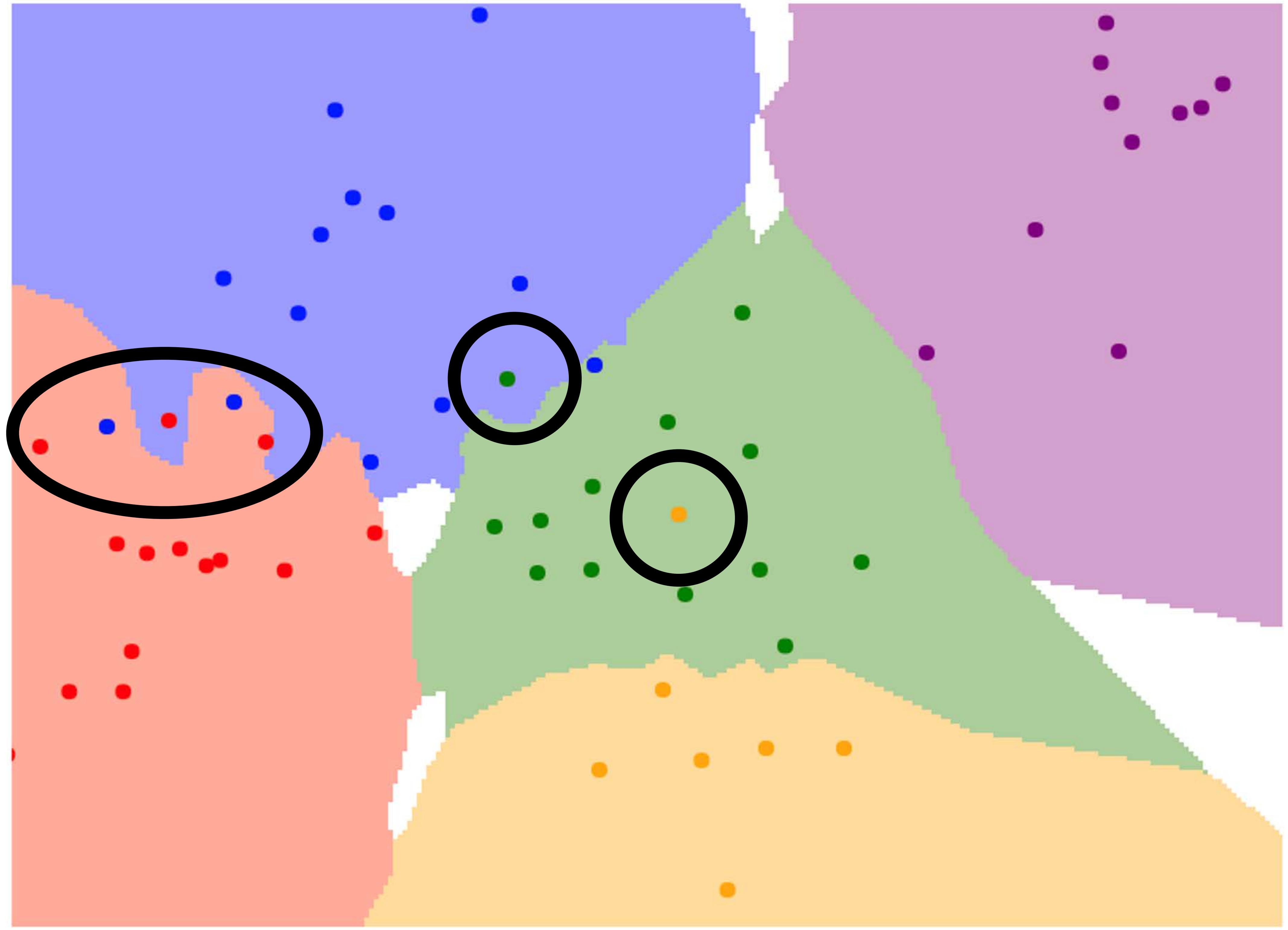
Using more neighbors helps smooth out rough decision boundaries

K-Nearest Neighbors Classification

$K = 1$



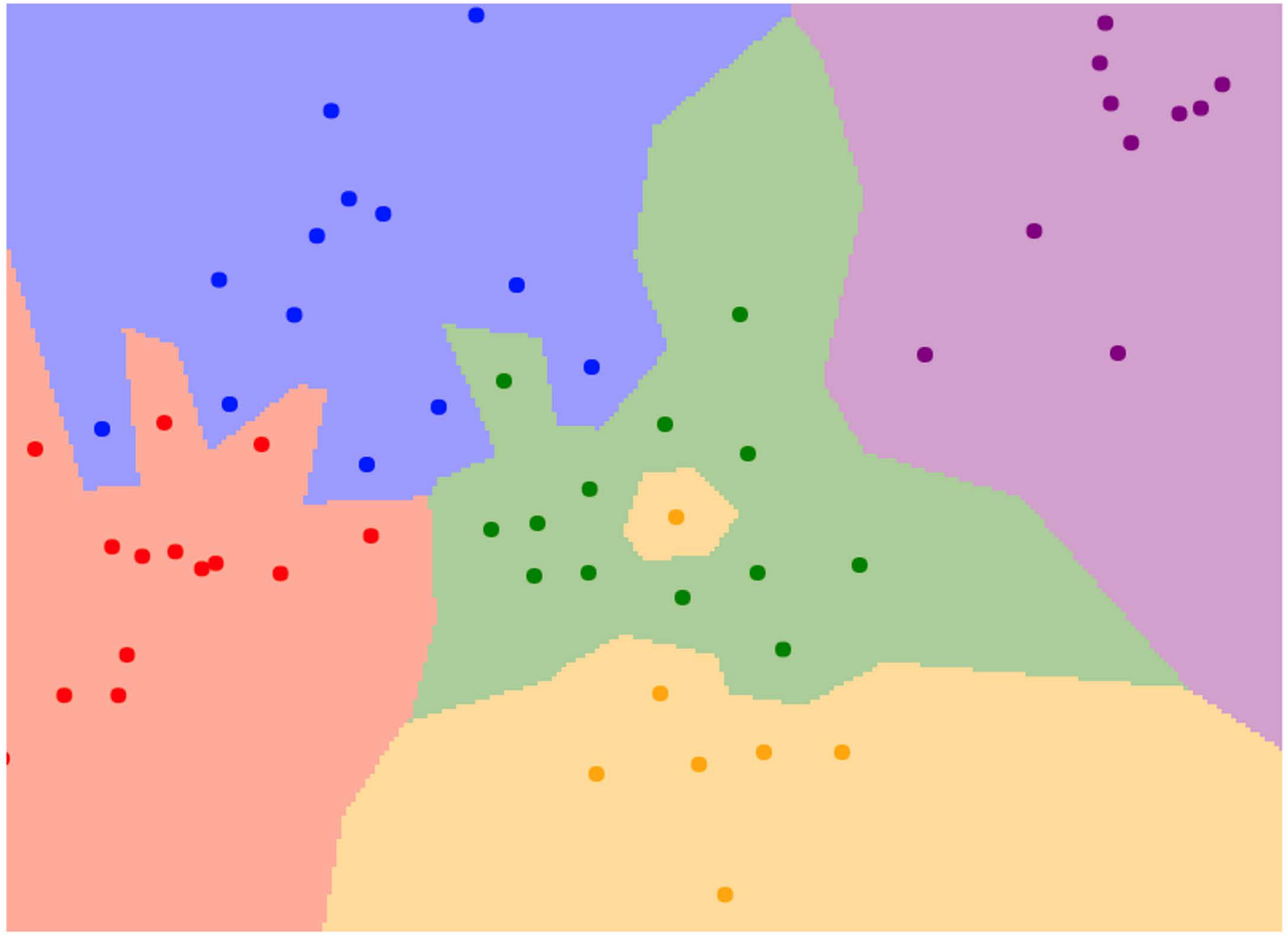
$K = 3$



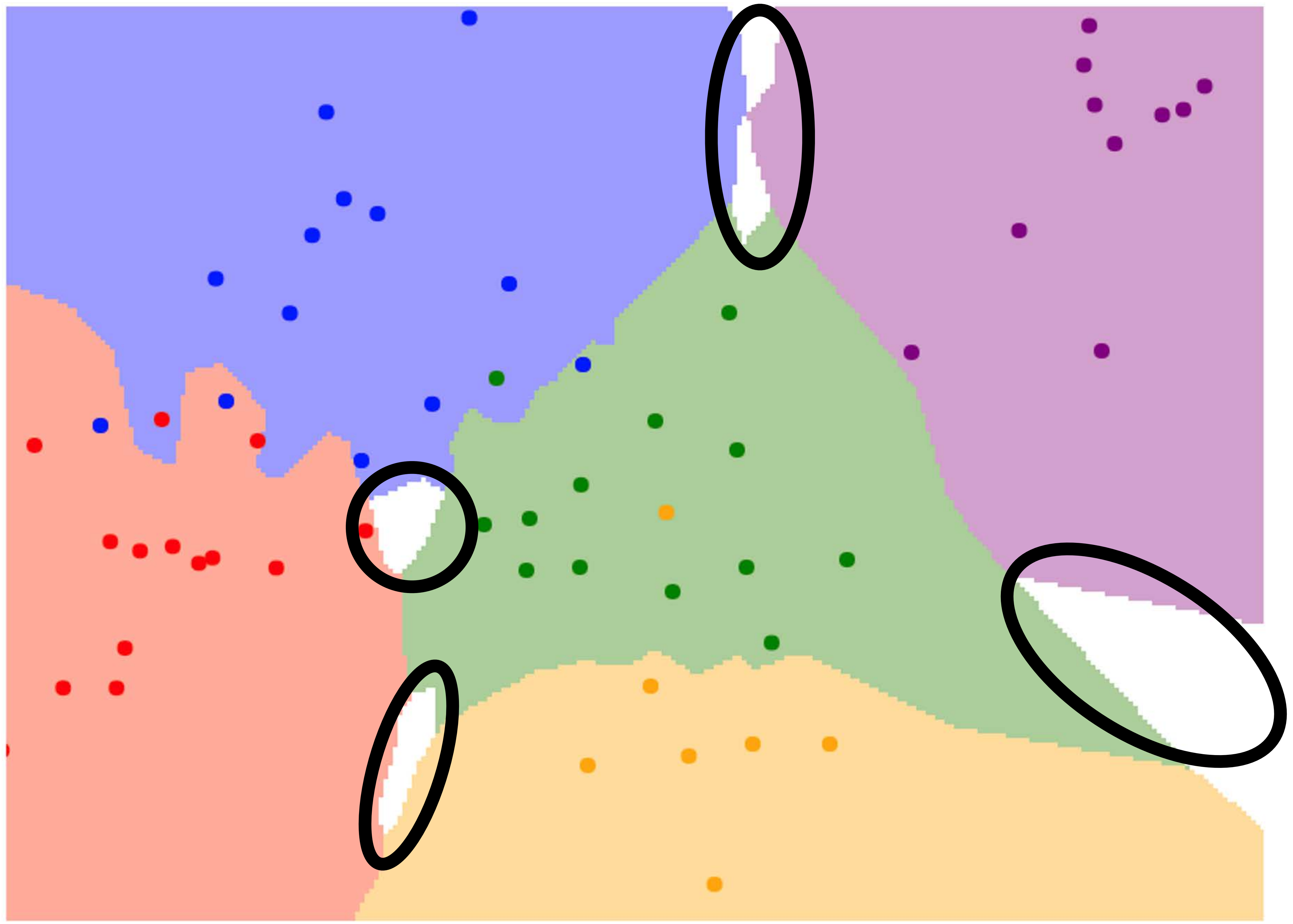
Using more neighbors helps reduce the effect of outliers

K-Nearest Neighbors Classification

$K = 1$



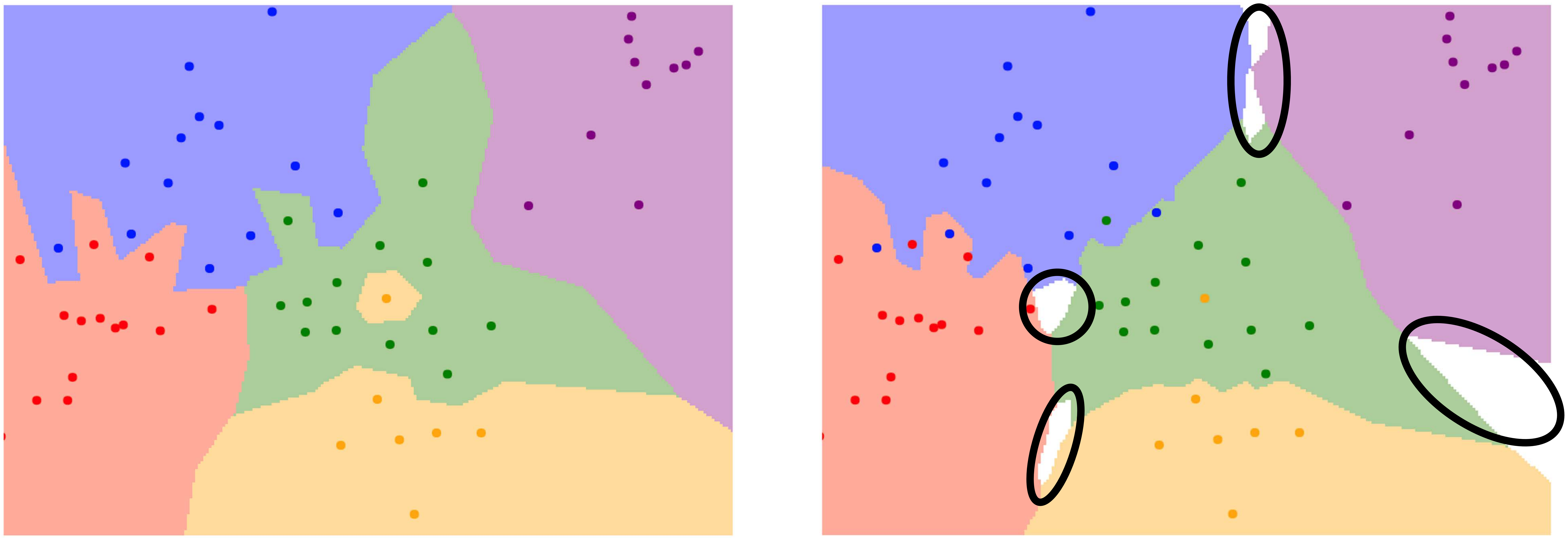
$K = 3$



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

K-Nearest Neighbors Classification

$K = 1$ $\qquad\qquad$ $K = 3$

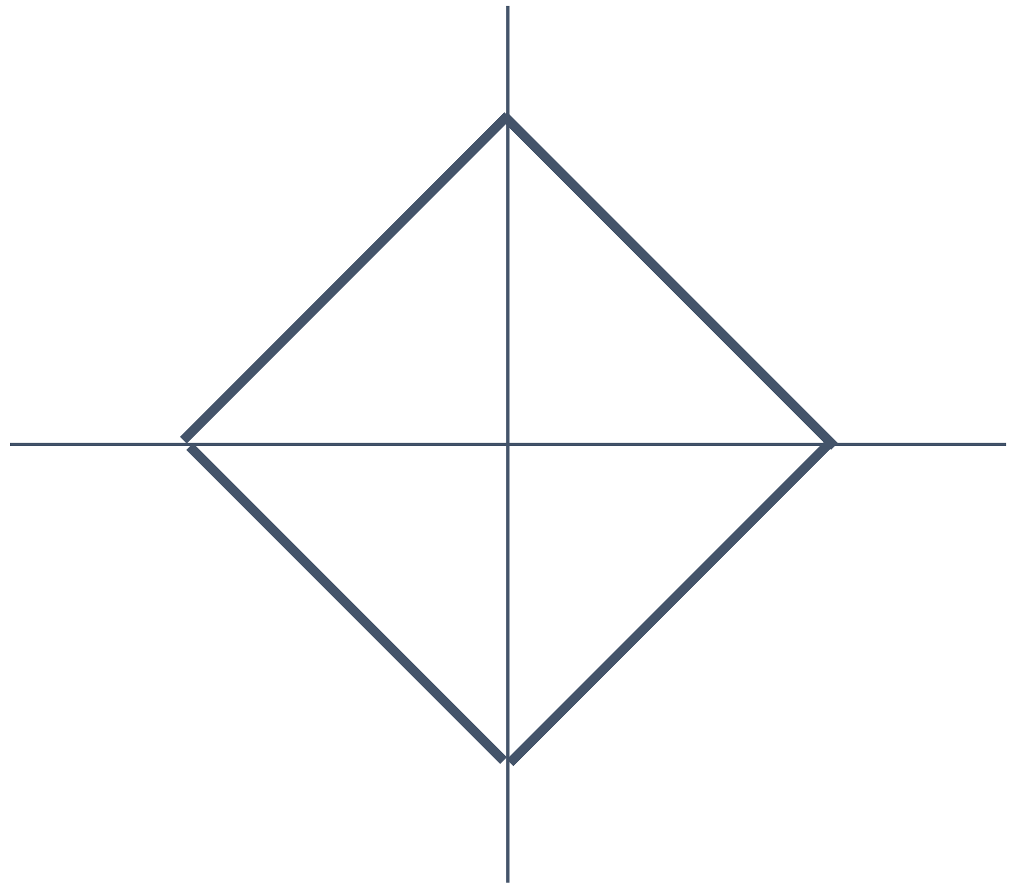


When $K > 1$ there can be ties between classes.
Need to break ties somehow!

K-Nearest Neighbors—Distance Metric

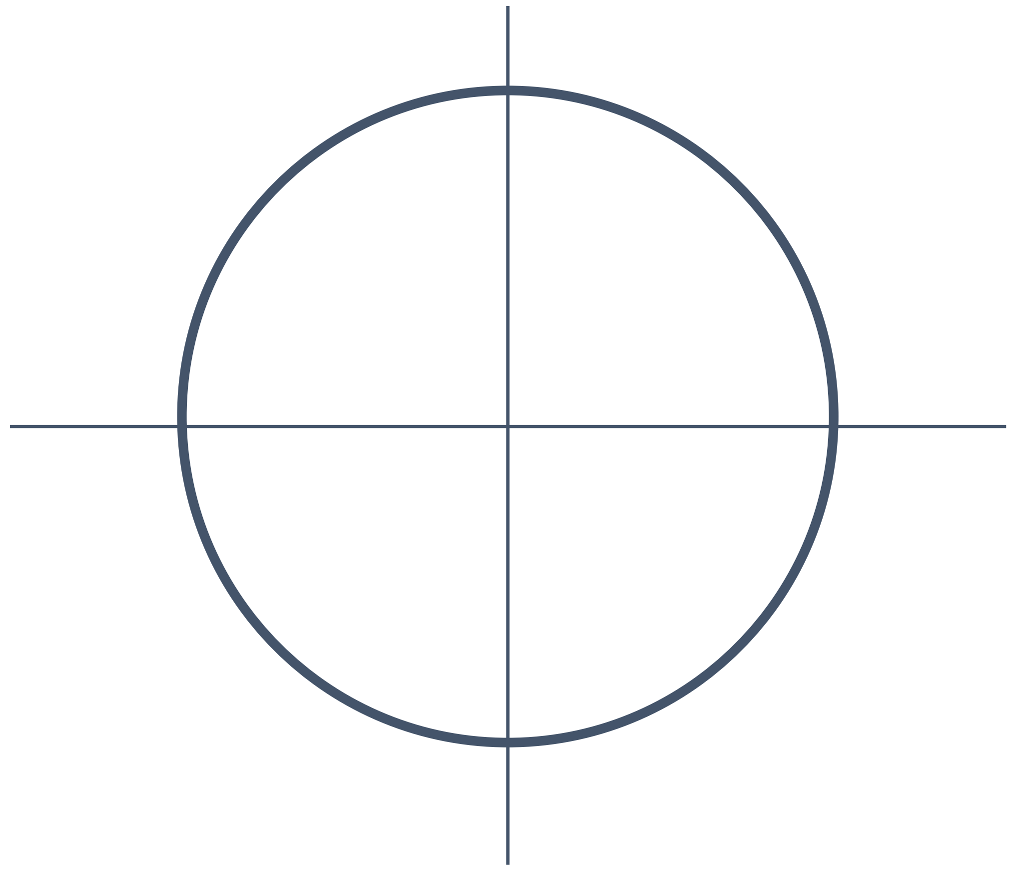
L1 (Manhattan) distance

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



L2 (Euclidean) distance

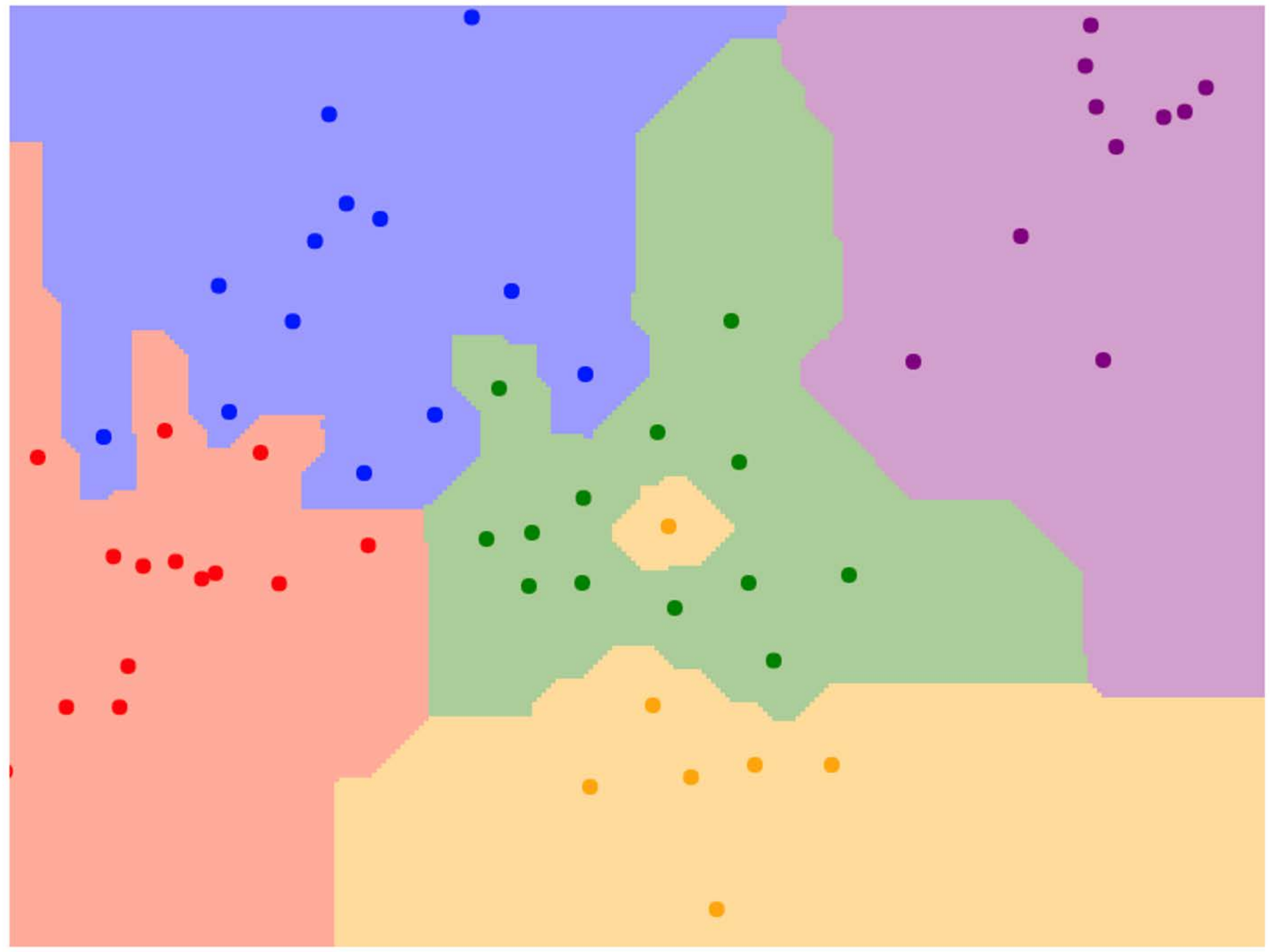
$$d_2(I_1, I_2) = \left(\sum_p (I_1^p - I_2^p)^2\right)^{\frac{1}{2}}$$



K-Nearest Neighbors—Distance Metric

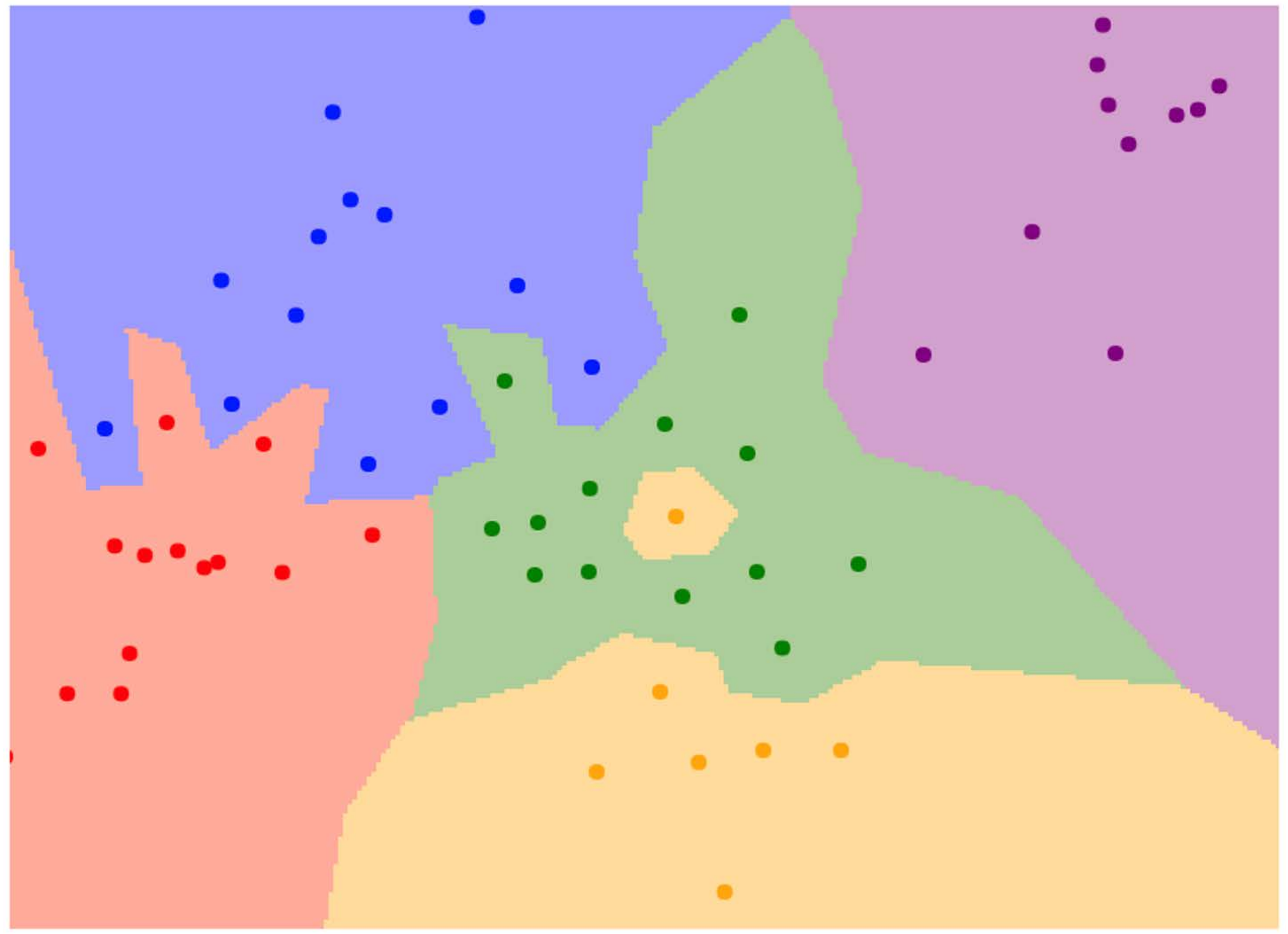
L1 (Manhattan) distance

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



L2 (Euclidean) distance

$$d_2(I_1, I_2) = \left(\sum_p (I_1^p - I_2^p)^2\right)^{\frac{1}{2}}$$



$K = 1$

K-Nearest Neighbors—Distance Metric

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

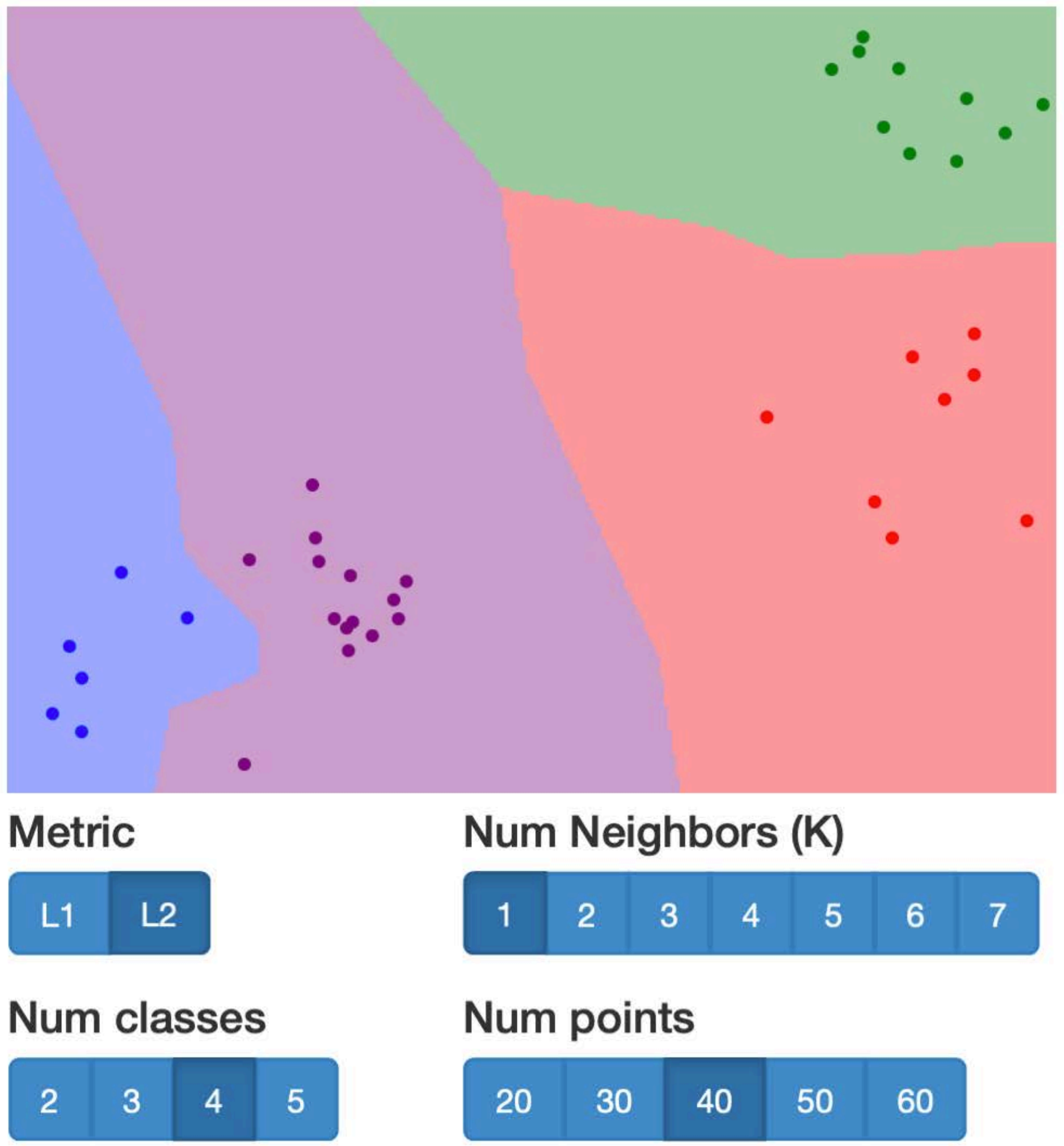
K-Nearest Neighbors—Web Demo

Interactively move points around and see decision boundaries change

Observe results with L1 vs L2 metrics

Observe results with changing number of training points and value of K

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?

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 need to try them all and observe what works best for our data.

Setting Hyperparameters

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

Your Dataset

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

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

train	test

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

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

train	test

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

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

train	test

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

Better!

train	validation	test

Setting Hyperparameters

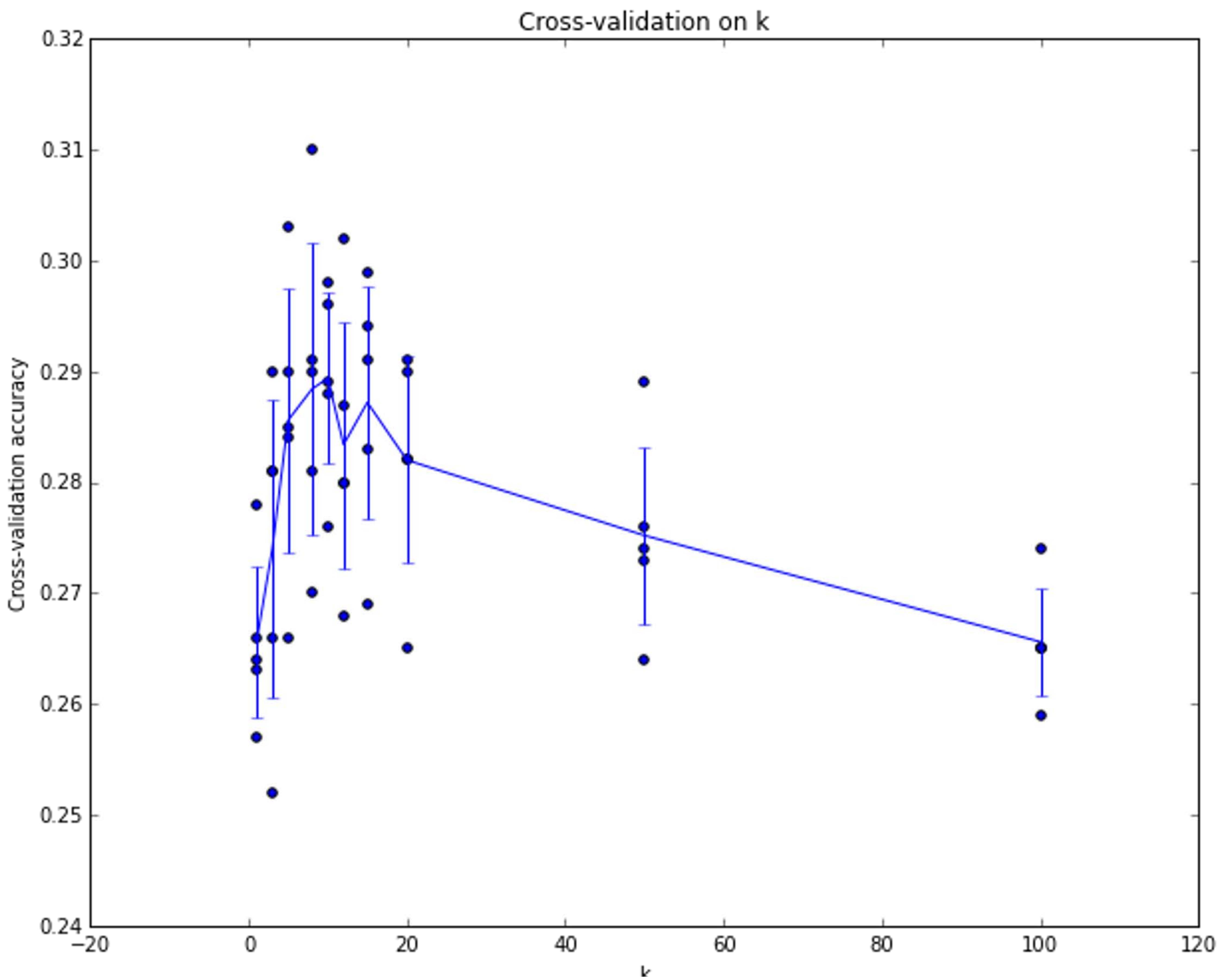
Your Dataset

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 (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)

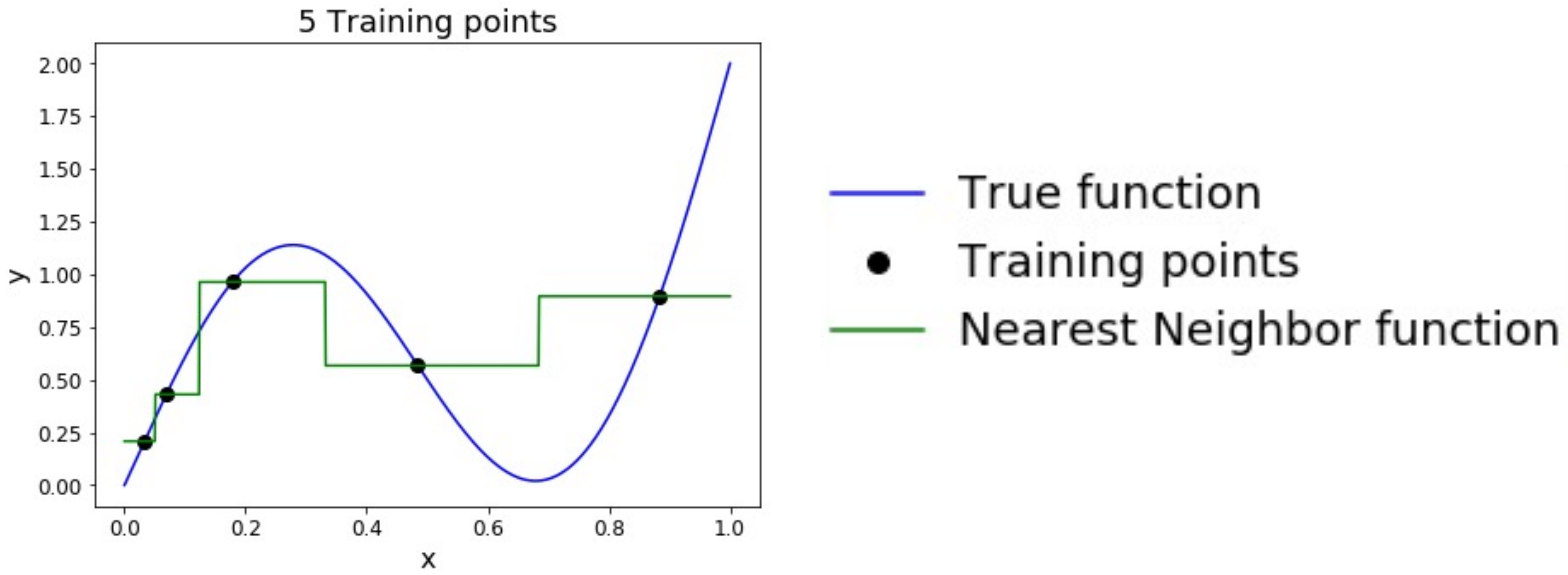
K-Nearest Neighbors—Universal Approximation

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 Neighbors—Universal Approximation

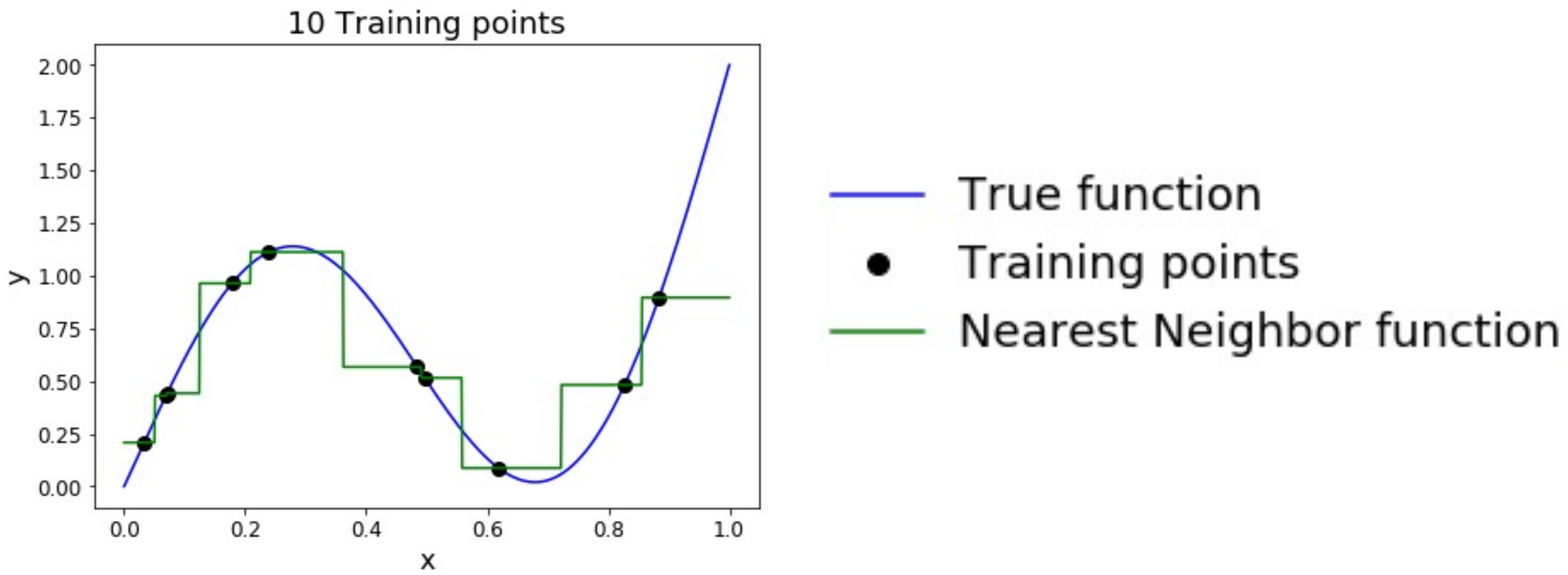
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



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

K-Nearest Neighbors—Universal Approximation

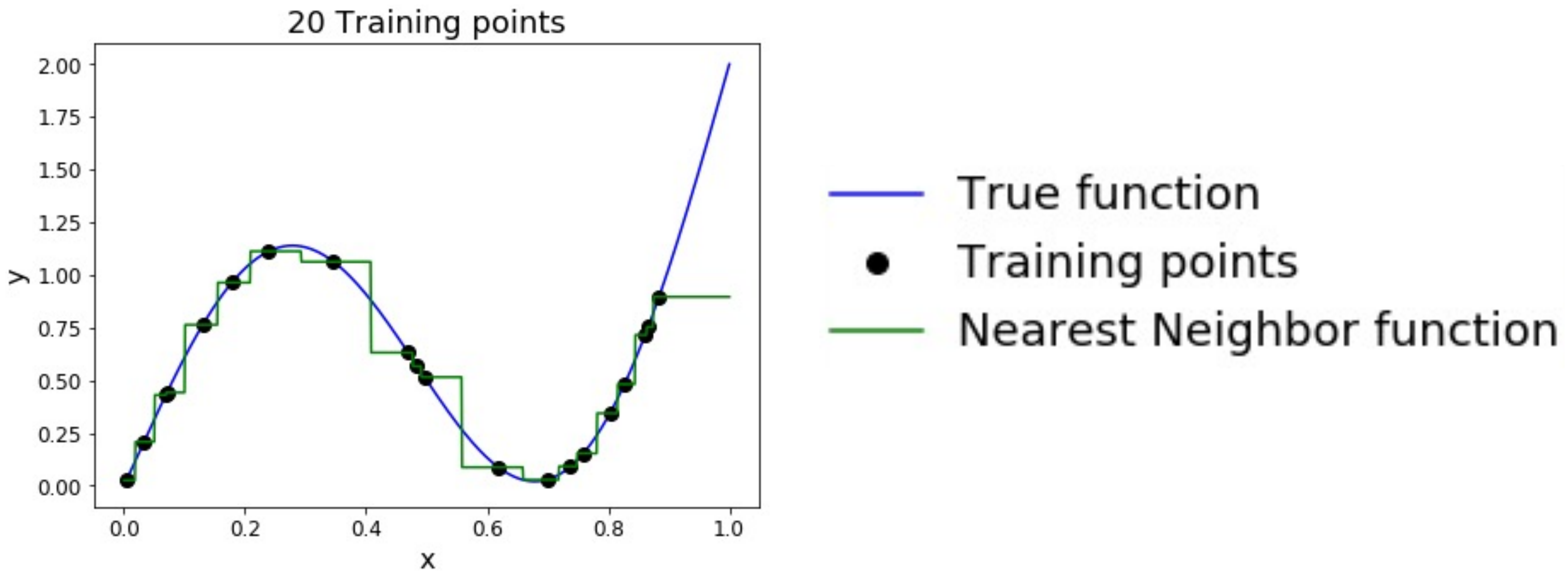
As the number of training samples goes to infinity, nearest neighbor can represent any$^{(*)}$ function!



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

K-Nearest Neighbors—Universal Approximation

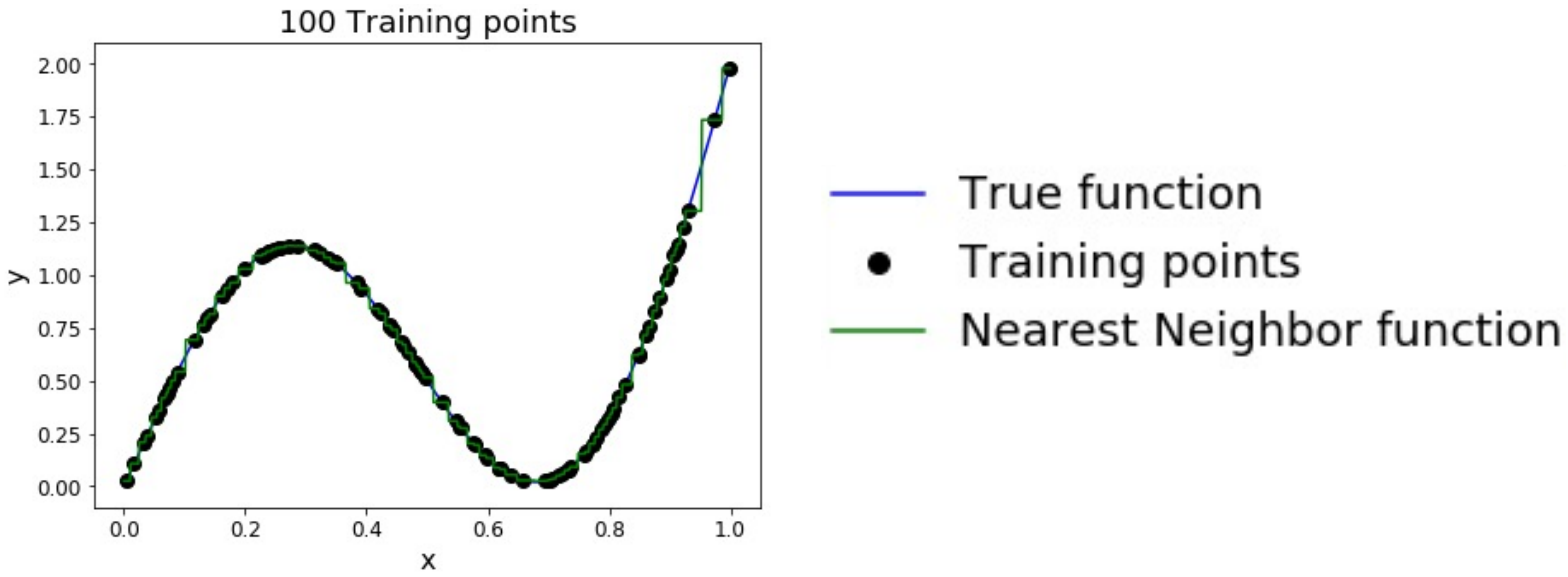
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



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

K-Nearest Neighbors—Universal Approximation

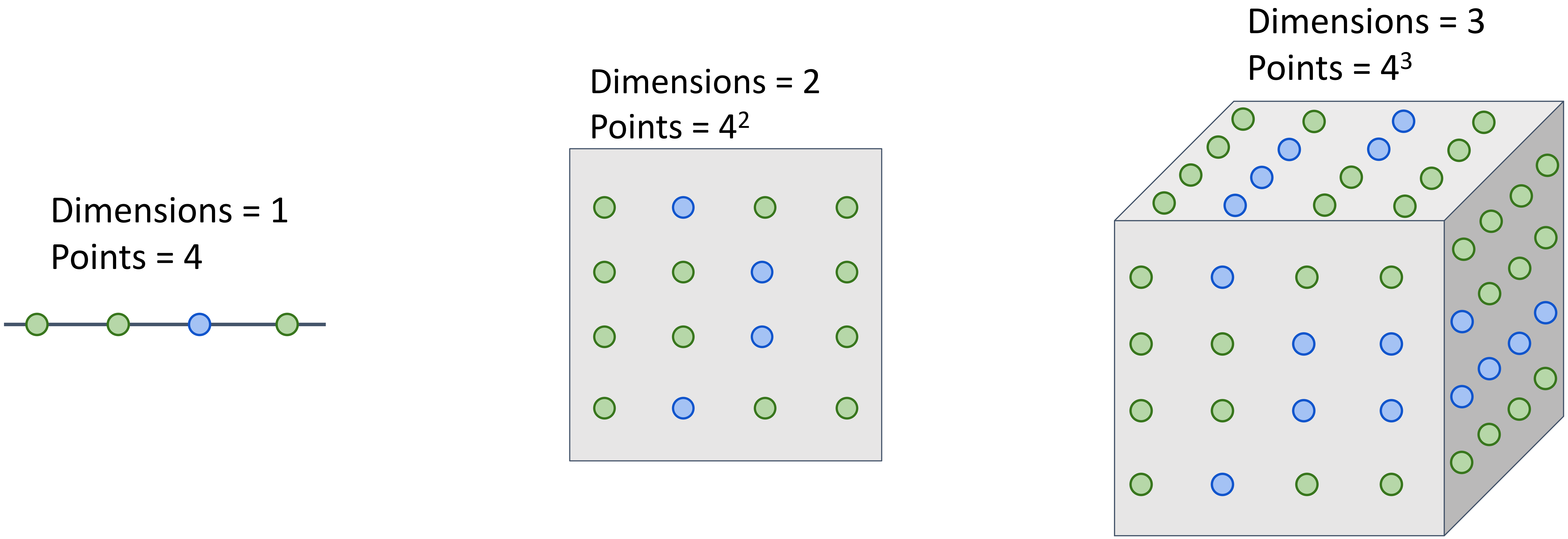
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



(*) 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



Problem—Curse of Dimensionality

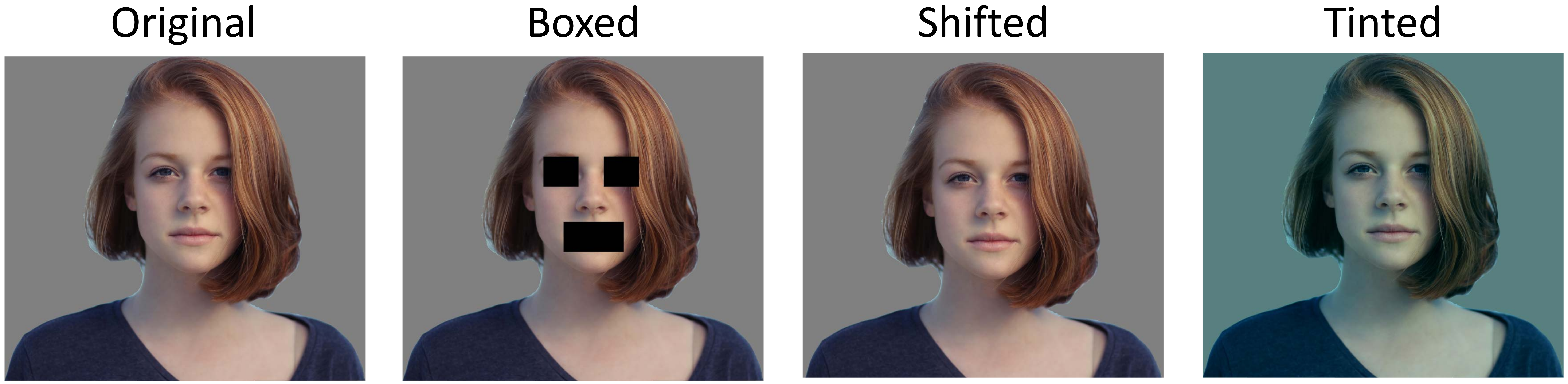
Curse of dimensionality: For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible
32x32 binary images

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

K-Nearest Neighbors Seldom Used on Raw Pixels

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



All 3 images have same L2 distance to the original

K-Nearest Neighbors with ConvNet Features Works Well

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

Summary

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

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

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

The **K-Nearest Neighbors** classifier predicts labels from nearest training samples

Distance metric and **K** are **hyperparameters**

Choose hyper parameters using the **validation set**; only run on the test set once at the very end!

Lets brainstorm on what your fav robot should do!!!

The Telegraph - Moley Robotics - https://www.youtube.com/watch?v=GyEHRXA_aA4

Next time: Linear Classifiers