Lecture 2: Recognition Problems in Vision and k-Nearest Neighbors

CS 444: Deep Learning for Computer Vision

Saurabh Gupta

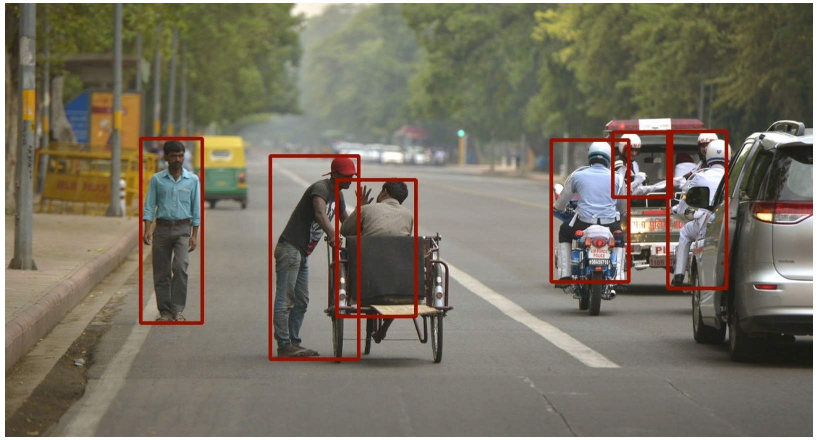
Many slides from Justin Johnson

Lecture overview

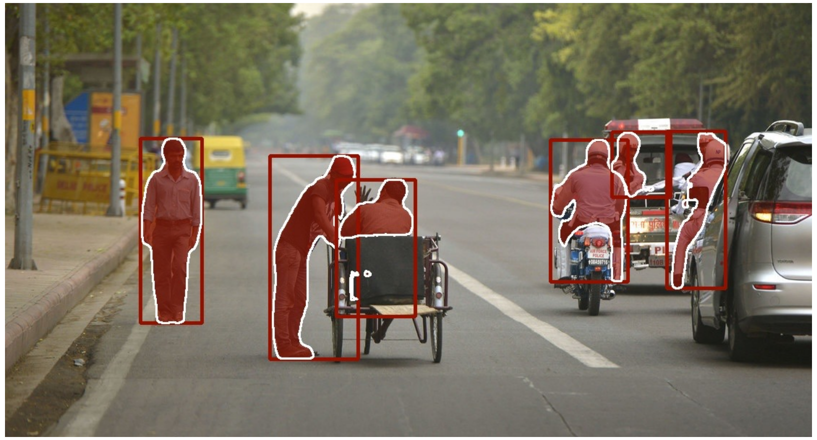
- Different problems in computer vision
- Supervised classification
- Beyond supervised classification: A taxonomy of prediction problems and types of supervision
- Image classification
- k-Nearest Neighbors

Different Recognition Problems

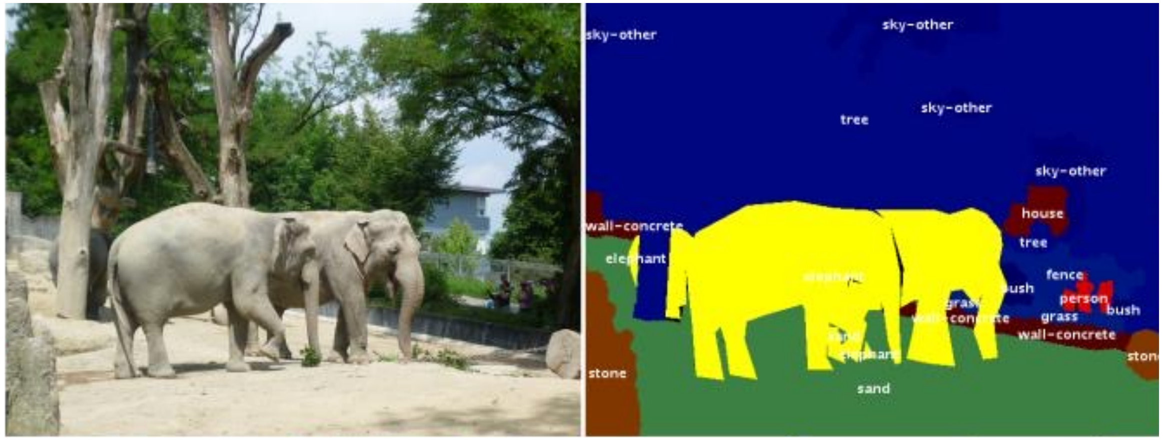
Classification: Assign image to one of a fixed set of categories



Object Detection: Put a bounding box around each instance of a class



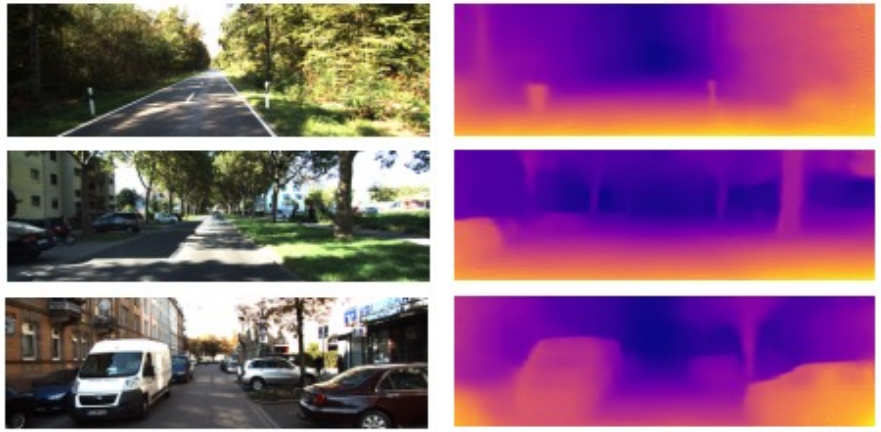
Instance Segmentation: Mark pixels for each instance of a class



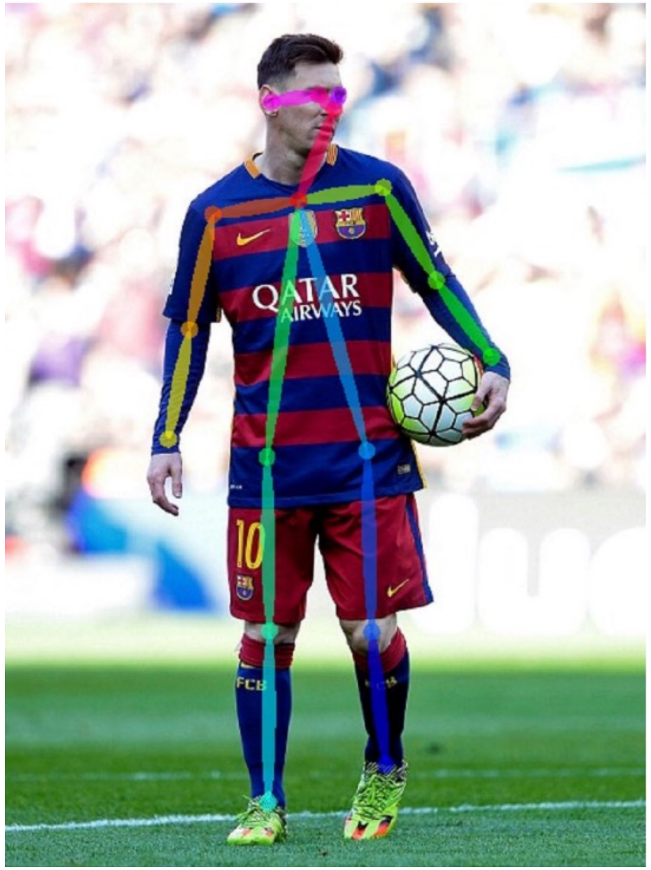
Semantic Segmentation: Label each pixels with its category

Different Recognition Problems

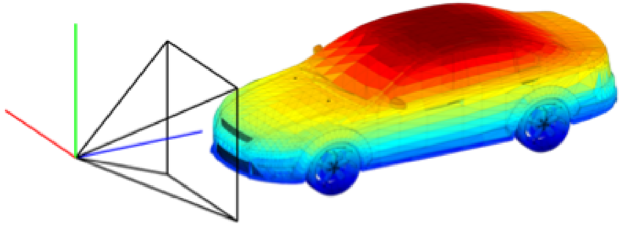
Image Captioning: Man riding a horse on a beach



Depth Prediction: how far is each pixel in the image

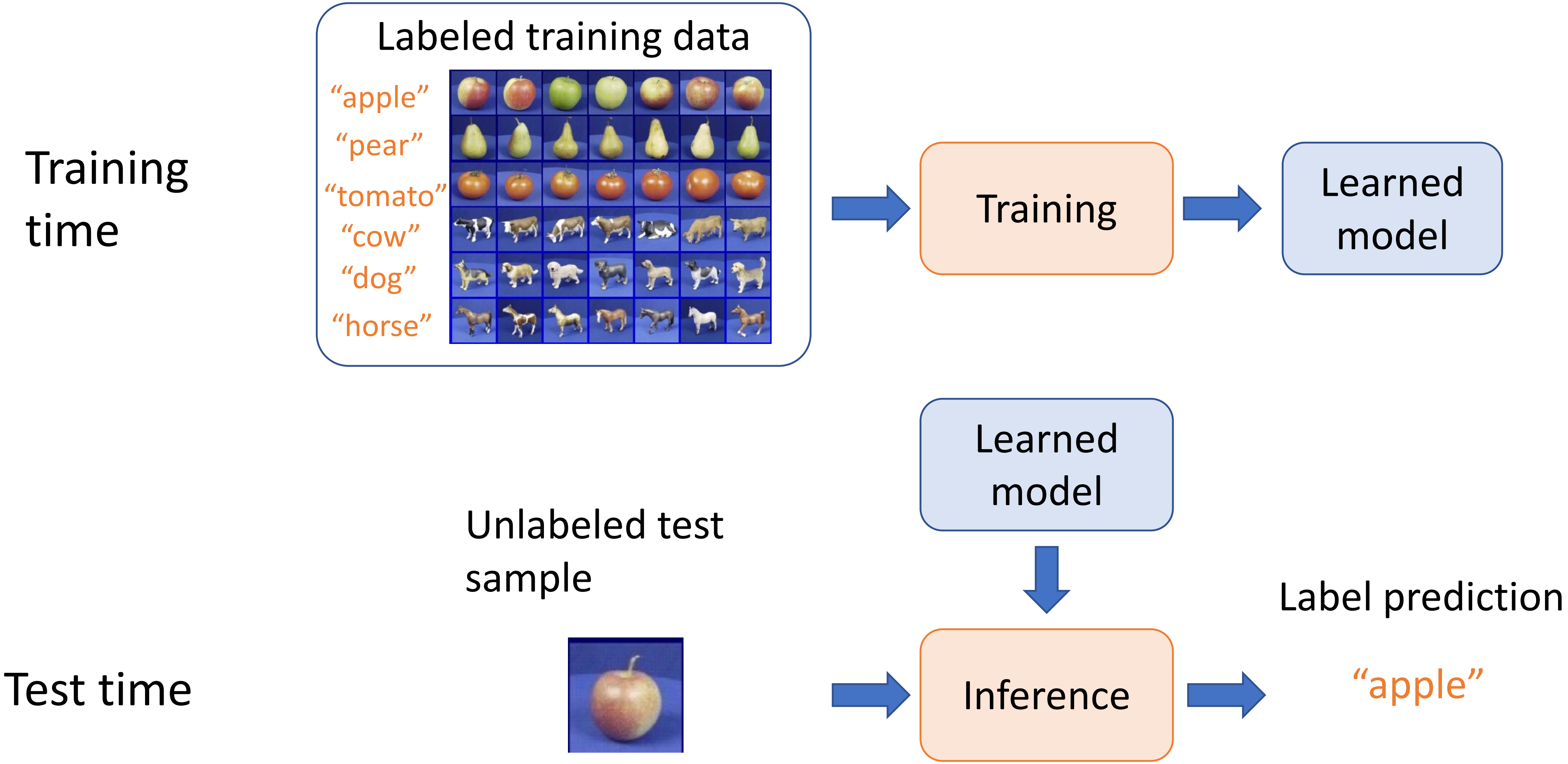


Keypoint prediction



Pose Prediction: Rotation that aligns object to a canonical pose

The basic ML framework (for supervised learning)



The basic ML framework (for supervised learning)

$$y = f(x)$$

output   prediction function   input

- **Training** (or **learning**): given a *training set* of labeled examples $\{(x_1, y_1), \dots, (x_N, y_N)\}$, instantiate a predictor f
- **Testing** (or **inference**): apply f to a new *test example* x and output the predicted value $y = f(x)$

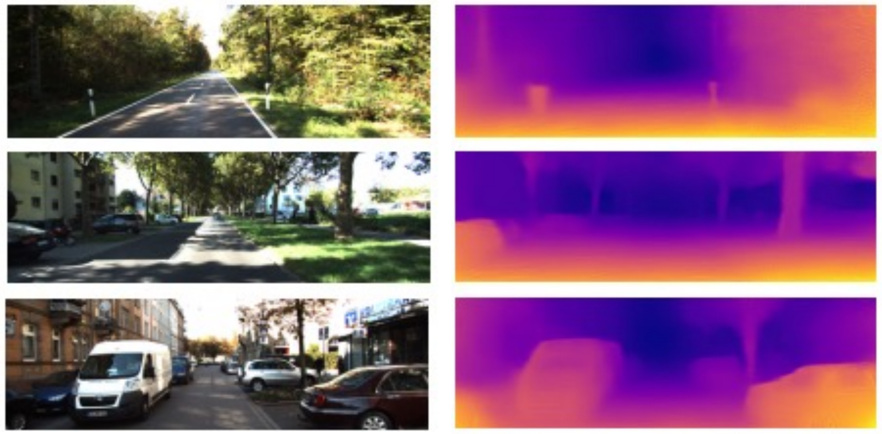
- Rather than hand-defining how 2D projections of apples are different from pears, f will learn this from the data.

Is an image classifier all you need?

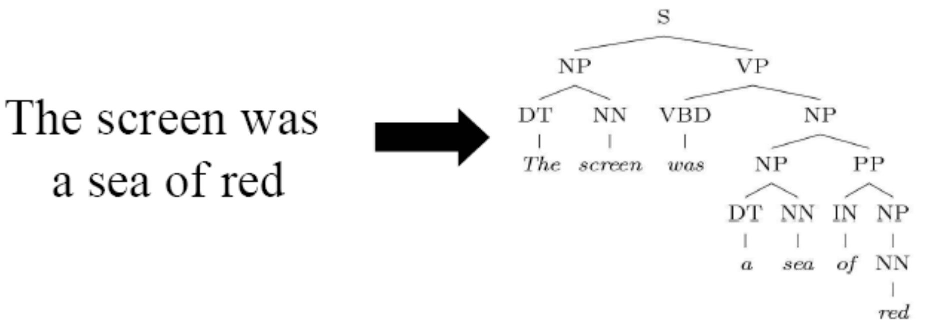
- Image Classification
- Object Detection
- Instance Segmentation
- Semantic Segmentation
- Image Captioning
- Depth Prediction
- Keypoint Prediction
- Pose Prediction
- …

Taxonomy of learning problems

- **Type of output**
  - Classification
  - Regression
    - $y = f(x)$. y is an arbitrary scalar and not a class label.
  - Structured prediction
    - $y = f(x)$. y is a structured object.



Depth Prediction: how far is each pixel in the image



Sentence

Parse tree

Several computer vision problems have structure in the output space, but often solving a classification problem with some simple post-processing (or even without) ends up being sufficient.

Taxonomy of learning problems

- **Type of output**
  - Classification
  - Regression
    - $y = f(x)$. y is an arbitrary scalar and not a class label.
  - Structured prediction
    - $y = f(x)$. y is a structured object.
- **Type of supervision**
  - Supervised learning
  - Unsupervised learning
  - Self-supervised or predictive learning

Type of supervision

Semi-supervised:
labels for a small portion of training data

Weakly supervised: noisy labels, labels not exactly for the task of interest

Unsupervised: no labels

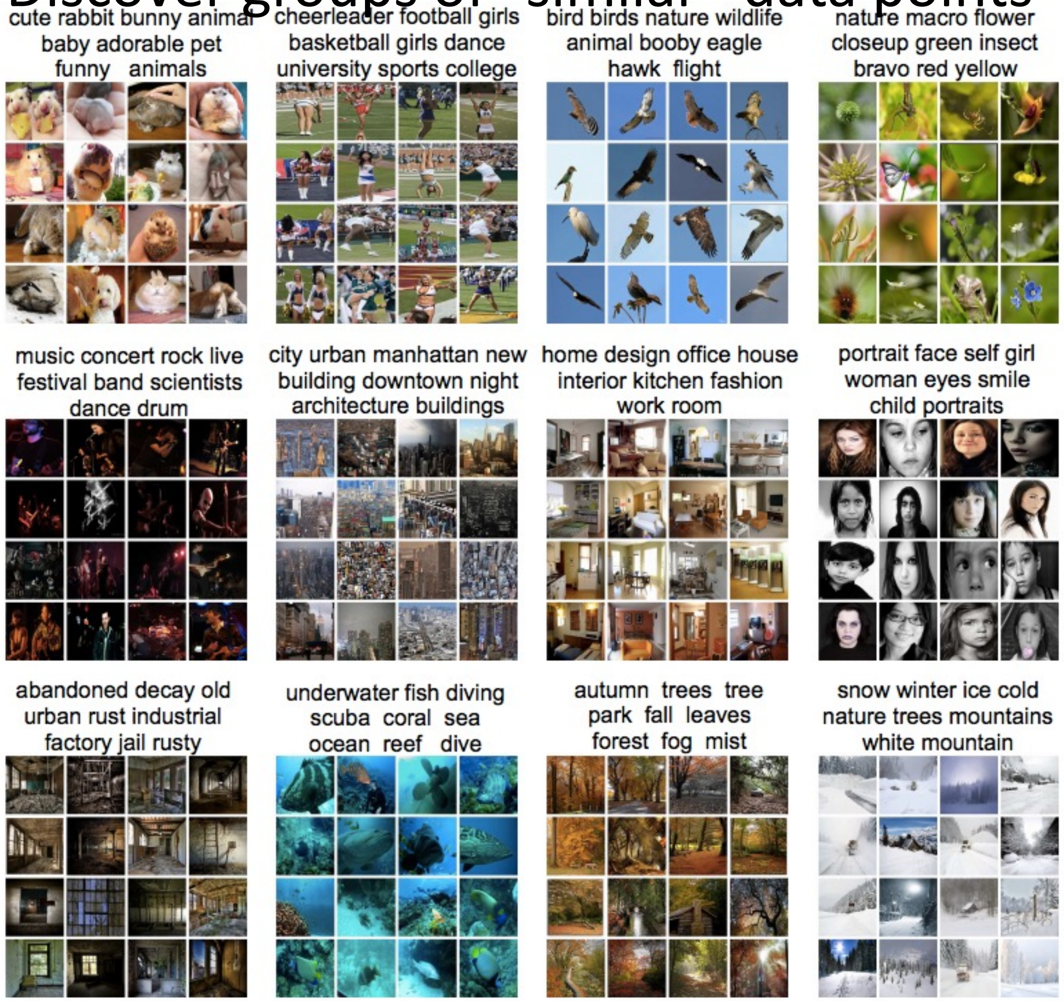
Self-supervised:
same as unsupervised?

Supervised:
clean, complete training labels for the task of interest

Unsupervised learning

Unsupervised learning

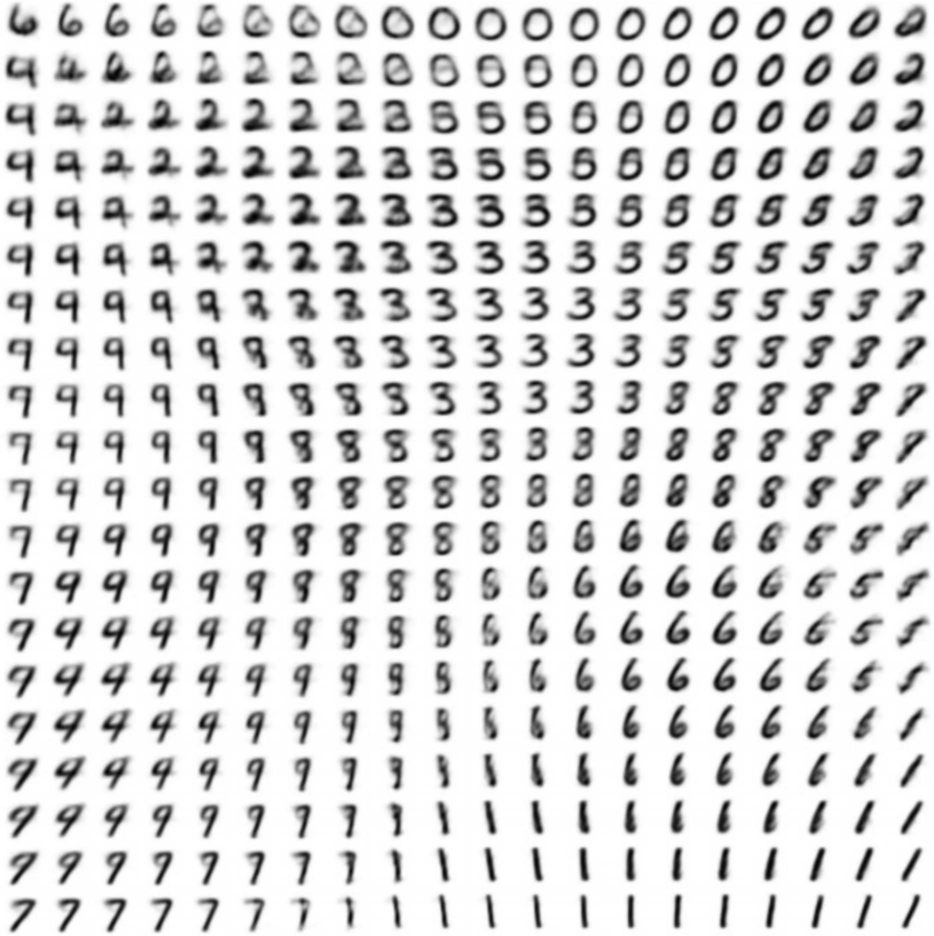
- **Clustering**
  - Discover groups of "similar" data points



Y. Gong, Q. Ke, M. Isard, and S. Lazebnik. [A Multi-View Embedding Space for Modeling Internet Images, Tags, and Their Semantics.]() IJCV 2014

Unsupervised learning

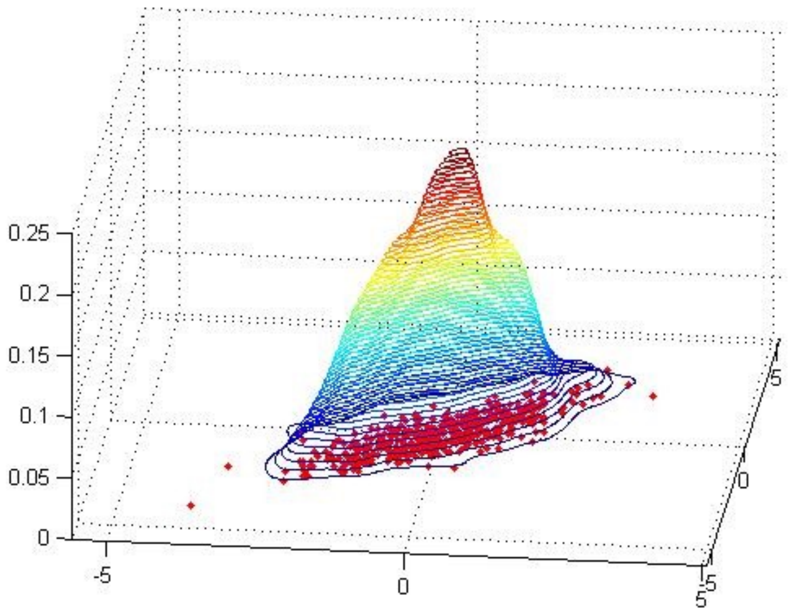
- **Dimensionality reduction, manifold learning**
  - Discover a lower-dimensional surface on which the data lives



D. Kingma and M. Welling, Auto-Encoding Variational Bayes, ICLR 2014

Unsupervised learning

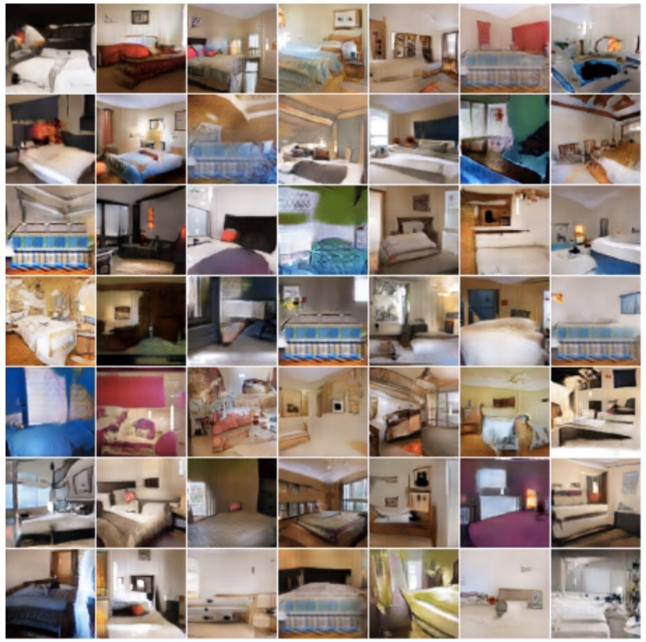
- Learning the data distribution
  - **Density estimation**: Find a function that approximates the probability density of the data (i.e., value of the function is high for "typical" points and low for "atypical" points)
  - An extremely hard problem for high-dimensional data…



Unsupervised learning

- Learning the data distribution
  - **Learning to sample:** Produce samples from a data distribution that mimics the training set

E.g. [Generative adversarial networks](#) (GANs)

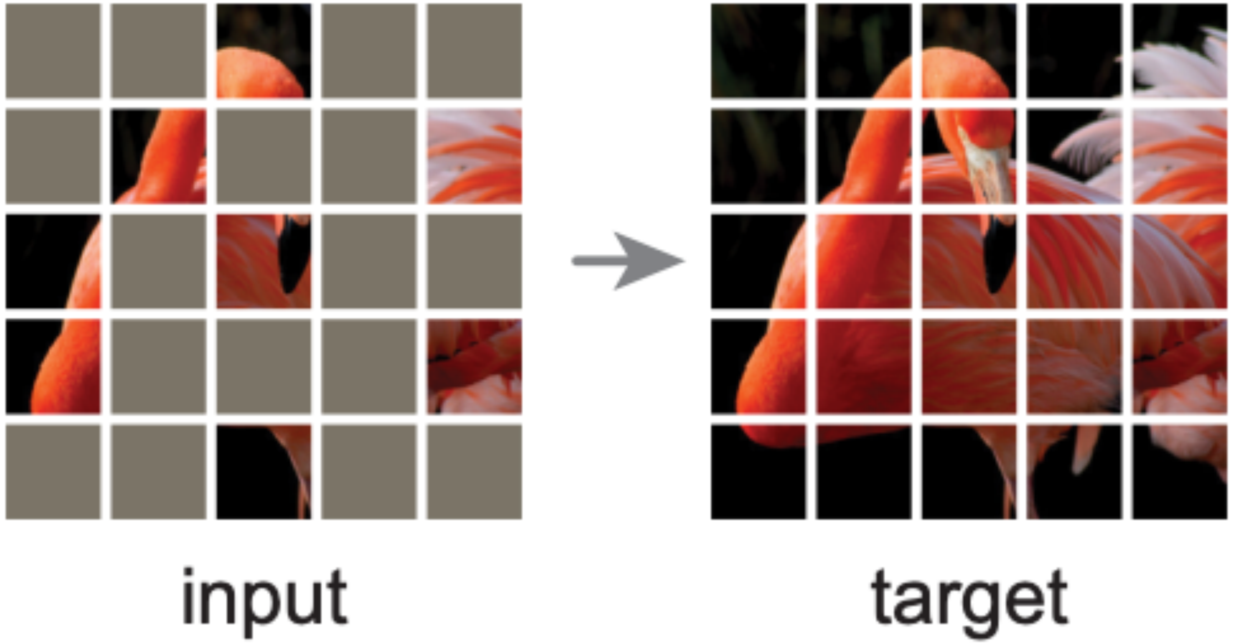


“Bedroom” (circa 2015)

“Face” (circa 2015)

Self-supervised or predictive learning

- Use part of the data to predict other parts of the data
  - Example: Masked patch prediction



K. He et al. [Masked autoencoders are scalable vision learners](). CVPR 2022

Taxonomy of learning problems

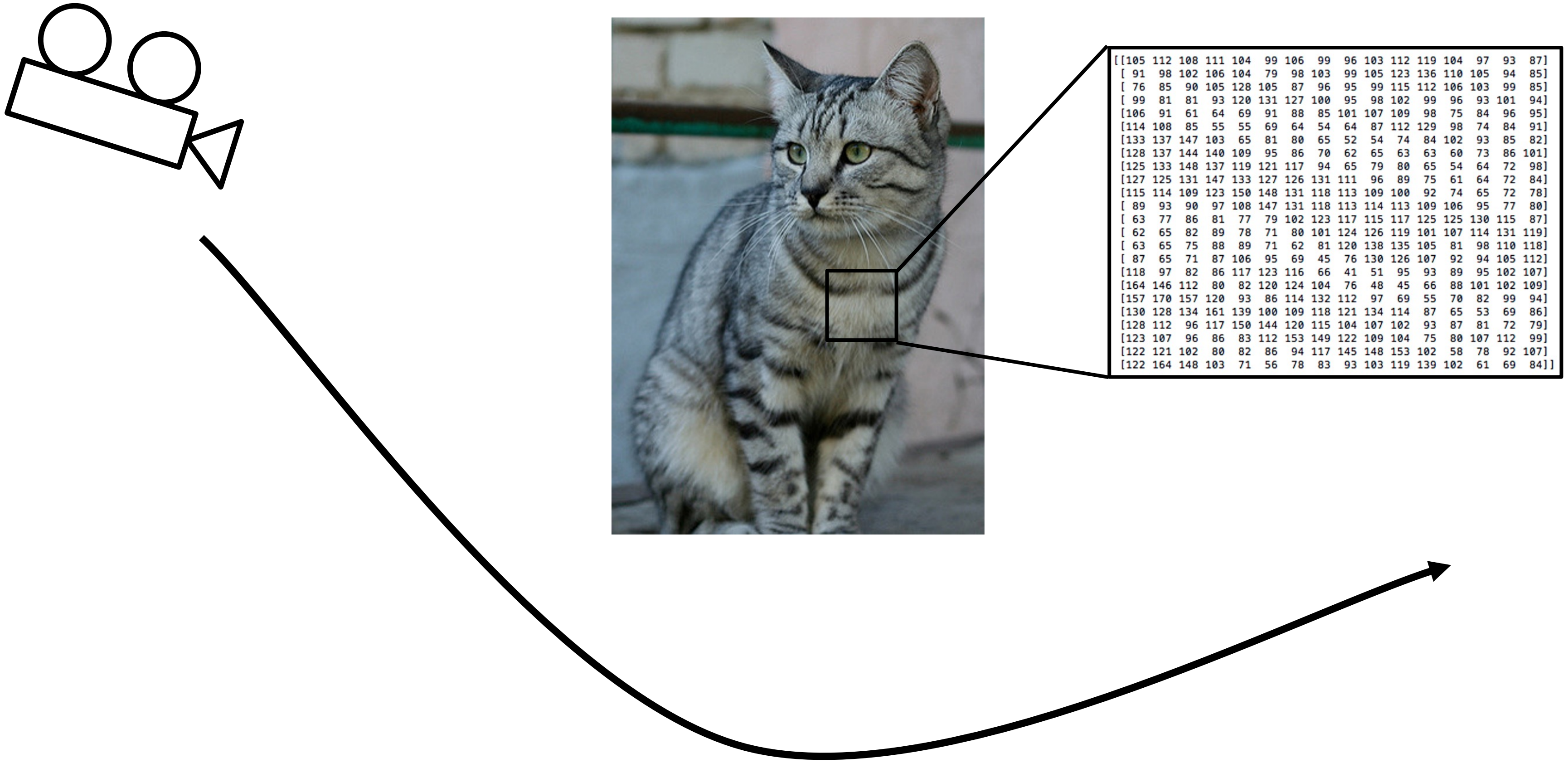
- **Type of output**
  - Classification
  - Regression
    - $y = f(x)$. y is an arbitrary scalar and not a class label.
  - Structured prediction
    - $y = f(x)$. y is a structured object.
- **Type of supervision**
  - Supervised learning
  - Unsupervised learning
  - Self-supervised or predictive learning

Taxonomy of learning problems

- **Type of output**
  - **Classification**
  - Regression
    - $y = f(x)$. y is an arbitrary scalar and not a class label.
  - Structured prediction
    - $y = f(x)$. y is a structured object.
- **Type of supervision**
  - **Supervised learning**
  - Unsupervised learning
  - Self-supervised or predictive learning

Image Classification

Challenges: Viewpoint Variation



All pixels change when the camera moves!

Slide from Justin Johnson

Challenges: Intraclass Variation

Slide from Justin Johnson

Challenges: Fine-Grained Categories

Maine Coon

Ragdoll

American Shorthair

Slide from Justin Johnson

Challenges: Background Clutter

This image is CC0 1.0 public domain

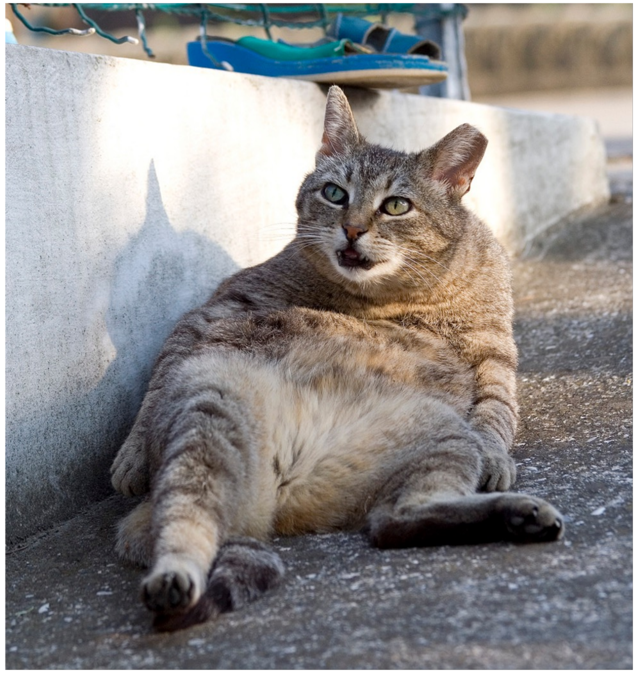
This image is CC0 1.0 public domain

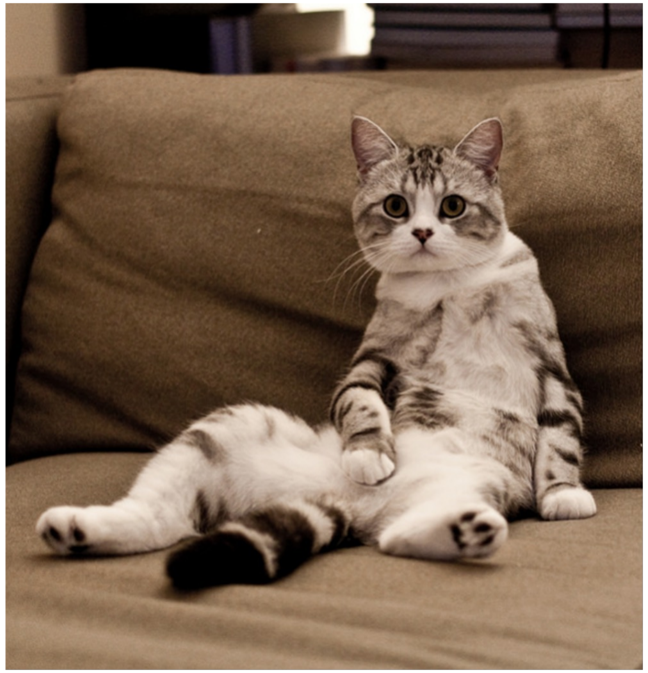
Slide from Justin Johnson

Challenges: Illumination Changes

Slide from Justin Johnson

Challenges: Deformation





Slide from Justin Johnson

Challenges: Occlusion

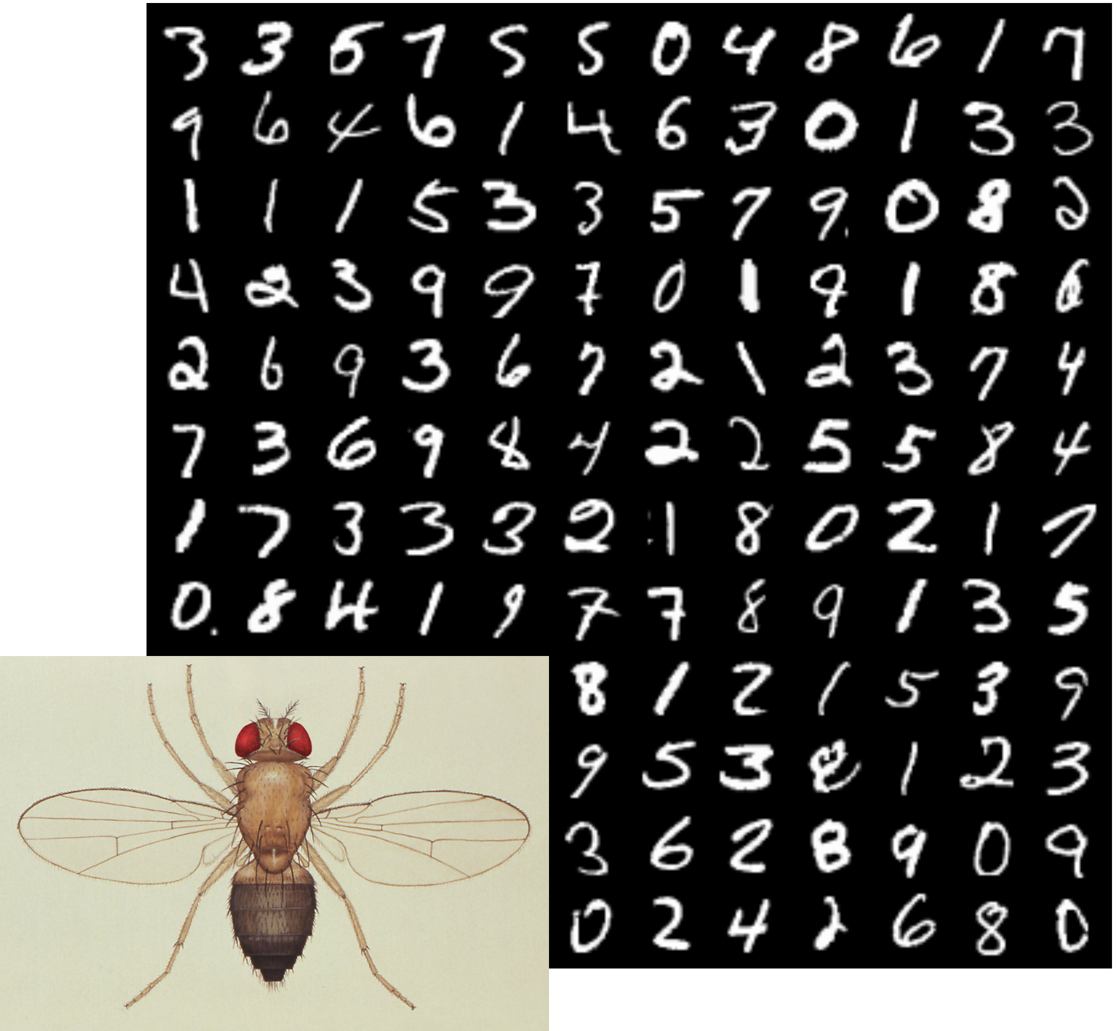
Slide from Justin Johnson

Image Classification Datasets: MNIST

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

Slide from Justin Johnson

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!

Slide from Justin Johnson

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.

Slide from Justin Johnson

Image Classification Datasets: **CIFAR100**

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

20 superclasses with 5 classes each:

<u>Aquatic mammals</u>: beaver, dolphin, otter, seal, whale
<u>Trees</u>: Maple, oak, palm, pine, willow

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

Slide from Justin Johnson

Image Classification Datasets: **ImageNet**

1000 classes

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

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

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

Slide from Justin Johnson

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

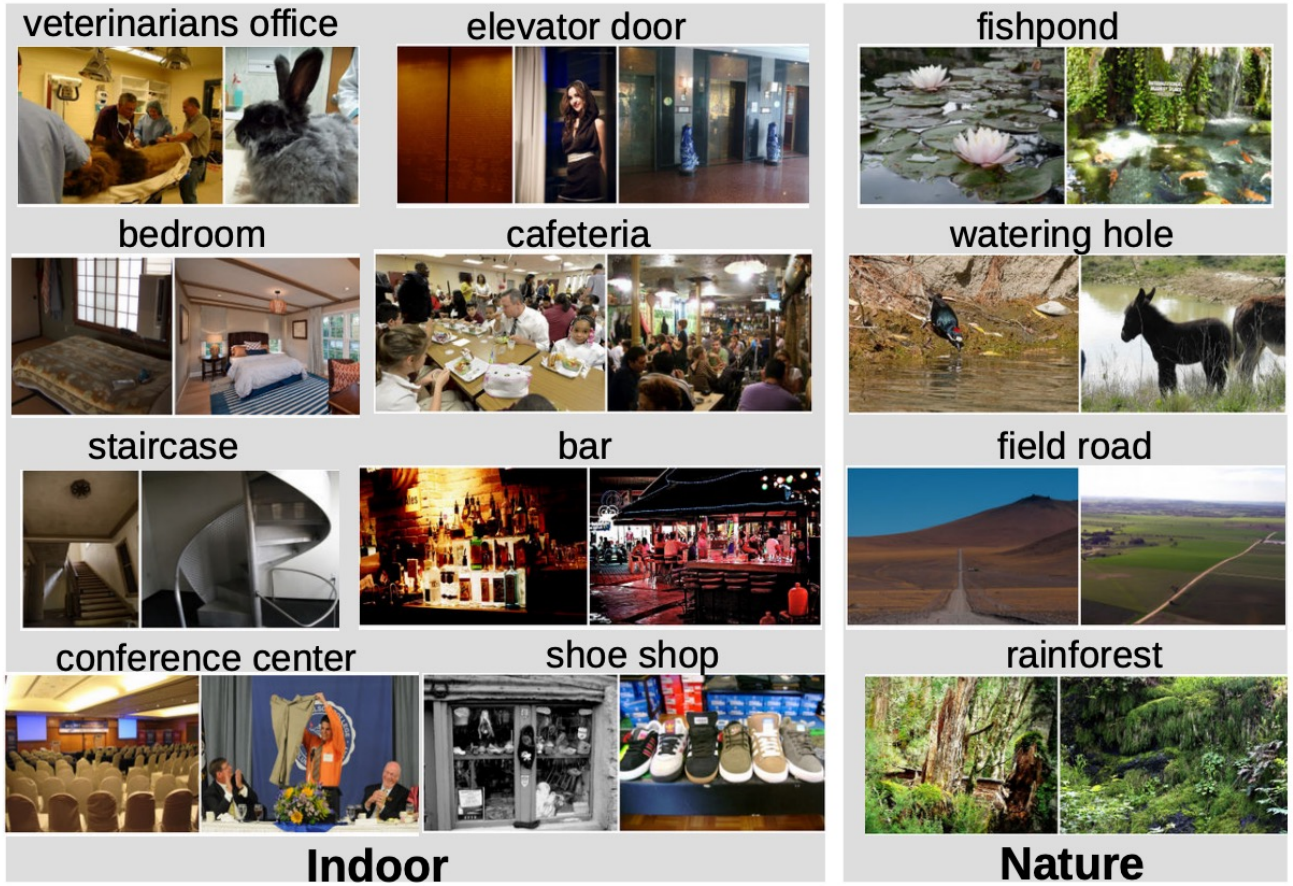
~1.3M training images (~1.3K per class)
50K validation images (50 per class)
100K test images (100 per class)
test labels are secret!

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

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

Slide from Justin Johnson

Image Classification Datasets: **MIT Places**



365 classes of different scene types

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

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

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

Slide from Justin Johnson

k-Nearest Neighbors

Slide from Justin Johnson

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

Slide from Justin Johnson

Distance Metric to compare images

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

$$\left| \begin{array}{c} \text{test image} \\ \begin{array}{|c|c|c|c|} \hline 56 & 32 & 10 & 18 \\ \hline 90 & 23 & 128 & 133 \\ \hline 24 & 26 & 178 & 200 \\ \hline 2 & 0 & 255 & 220 \\ \hline \end{array} \end{array} - \begin{array}{c} \text{training image} \\ \begin{array}{|c|c|c|c|} \hline 10 & 20 & 24 & 17 \\ \hline 8 & 10 & 89 & 100 \\ \hline 12 & 16 & 178 & 170 \\ \hline 4 & 32 & 233 & 112 \\ \hline \end{array} \end{array} \right| = \begin{array}{c} \text{pixel-wise absolute value differences} \\ \begin{array}{|c|c|c|c|} \hline 46 & 12 & 14 & 1 \\ \hline 82 & 13 & 39 & 33 \\ \hline 12 & 10 & 0 & 30 \\ \hline 2 & 32 & 22 & 108 \\ \hline \end{array} \end{array} \xrightarrow{\text{add}} 456$$

Slide from Justin Johnson

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
```

Slide from Justin Johnson

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```

Nearest Neighbor Classifier

Memorize training data

Slide from Justin Johnson

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

Slide from Justin Johnson

```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?

Slide from Justin Johnson

```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)

Slide from Justin Johnson

```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?

Slide from Justin Johnson

```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)

Slide from Justin Johnson

```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!

Slide from Justin Johnson

1-Nearest Neighbor Complexity

Method	Query Time Complexity
Exact Algorithms:	
RP Tree	$O((d' \log d')^{d'} + \log n)$
Spill Tree	$O(d'^{d'} + \log n)$
Karger & Ruhl (2002)	$O(2^{3d'} \log n)$
Navigating Net	$2^{O(d')} \log n$
Cover Tree	$O(2^{12d'} \log n)$
Rank Cover Tree	$O(2^{O(d' \log h)} n^{2/h})$ for $h \geq 3$
DCI	$O(d \max(\log n, n^{1-1/d'}))$
Prioritized DCI (Proposed Method)	$O(d \max(\log n, n^{1-m/d'})$ $+(m \log m) \max(\log n, n^{1-1/d'}))$ for $m \geq 1$
Approximate Algorithms:	
k-d Tree	$O((1/\epsilon)^d \log n)$
BBD Tree	$O((6/\epsilon)^d \log n)$
LSH	$\approx O(dn^{1/(1+\epsilon)^2})$

Table 1. Query time complexities of various algorithms for 1-NN search. Ambient dimensionality, intrinsic dimensionality, dataset size and approximation ratio are denoted as d, d', n and $1 + \epsilon$. A visualization of the growth of various time complexities as a function of the intrinsic dimensionality is shown in Figure 1.

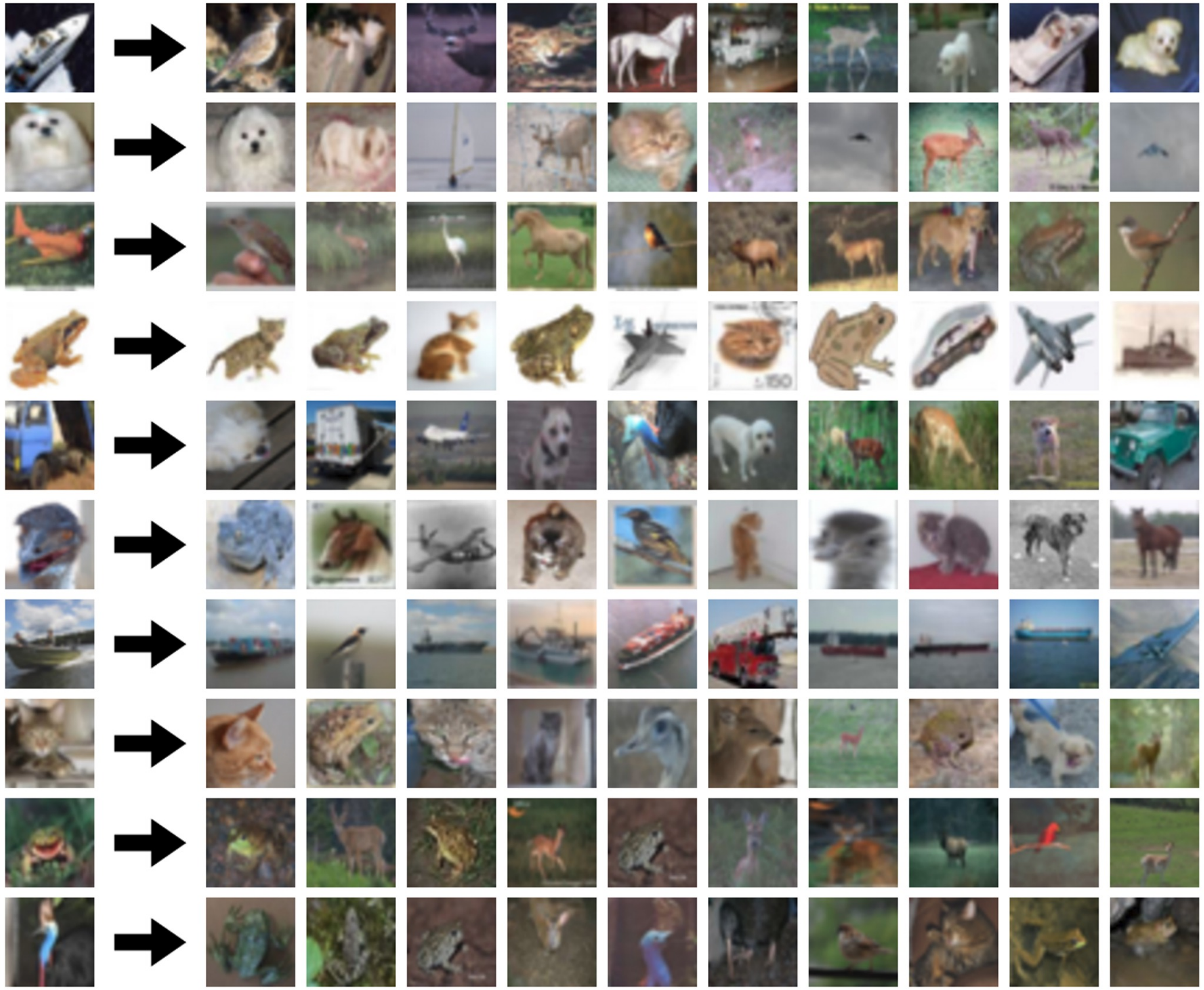
Bad news overall:
Exponential in dimensionality or (almost) linear in number of data points.

Good Implementation:

https://github.com/facebookresearch/faiss

Fast k-Nearest Neighbour Search via Prioritized DCI. Ke Li, Jitendra Malik. ICML 2017

What does this look like?



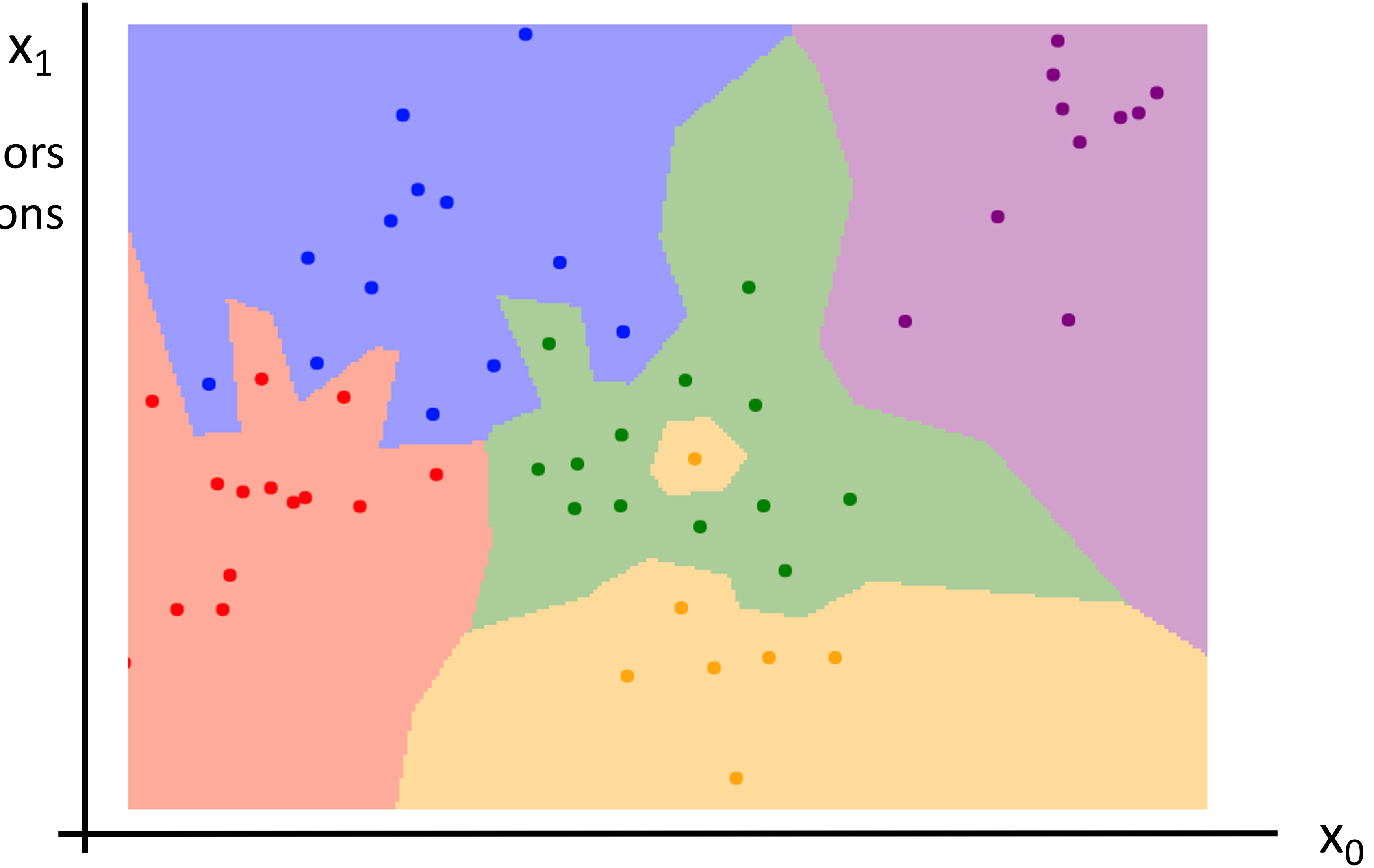
Slide from Justin Johnson

What does this look like?

Slide from Justin Johnson

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

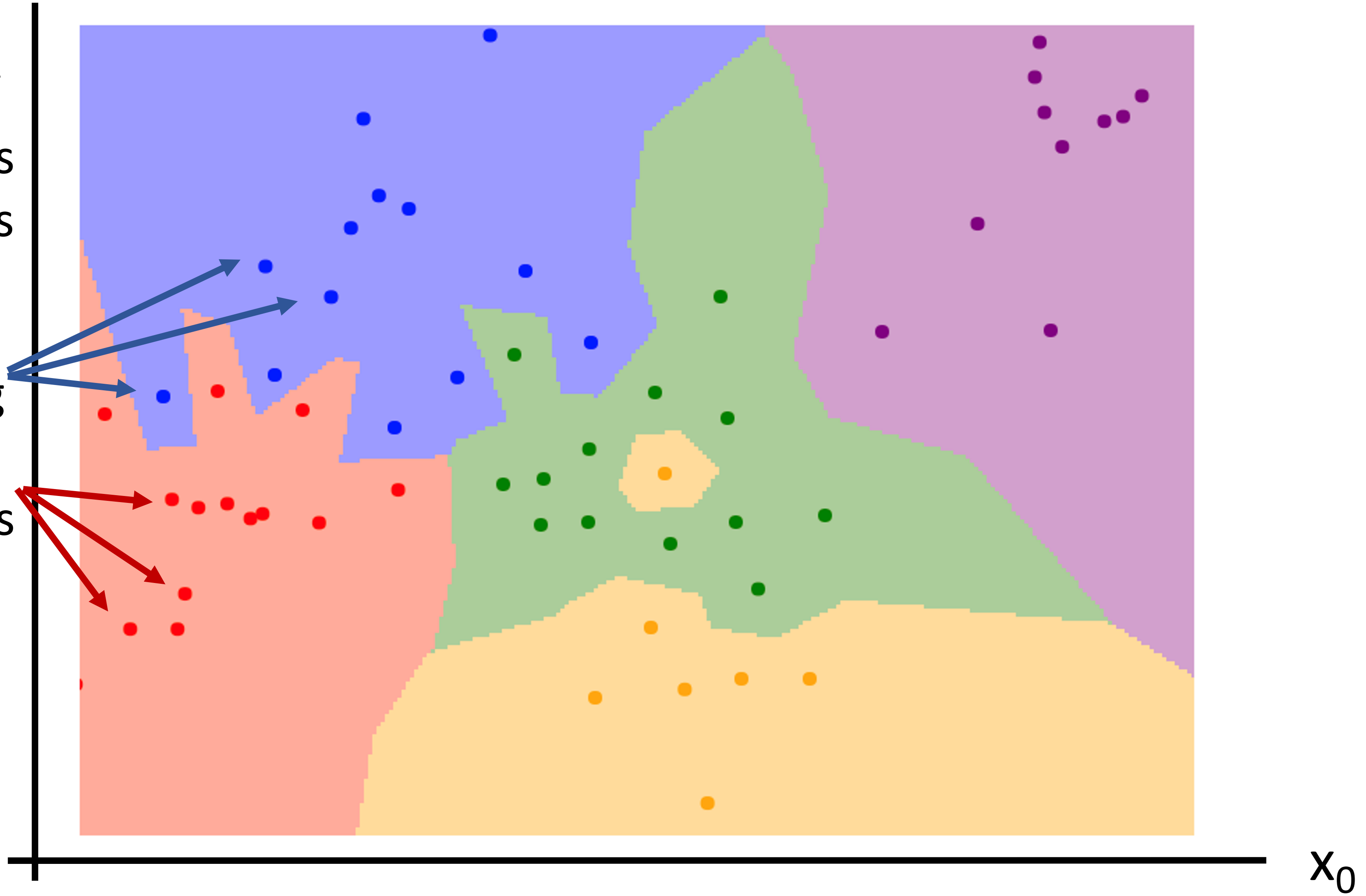


Slide from Justin Johnson

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels



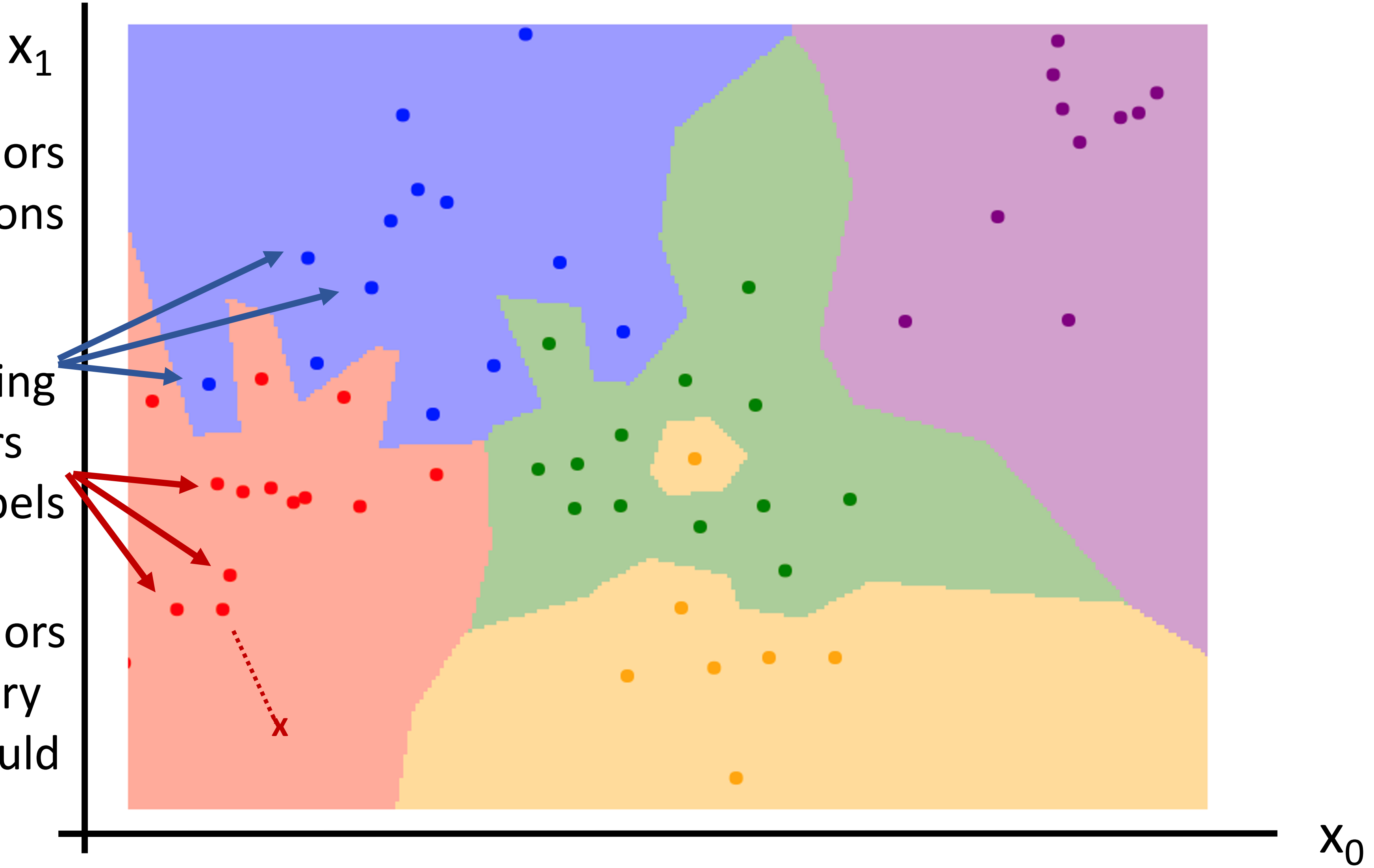
Slide from Justin Johnson

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



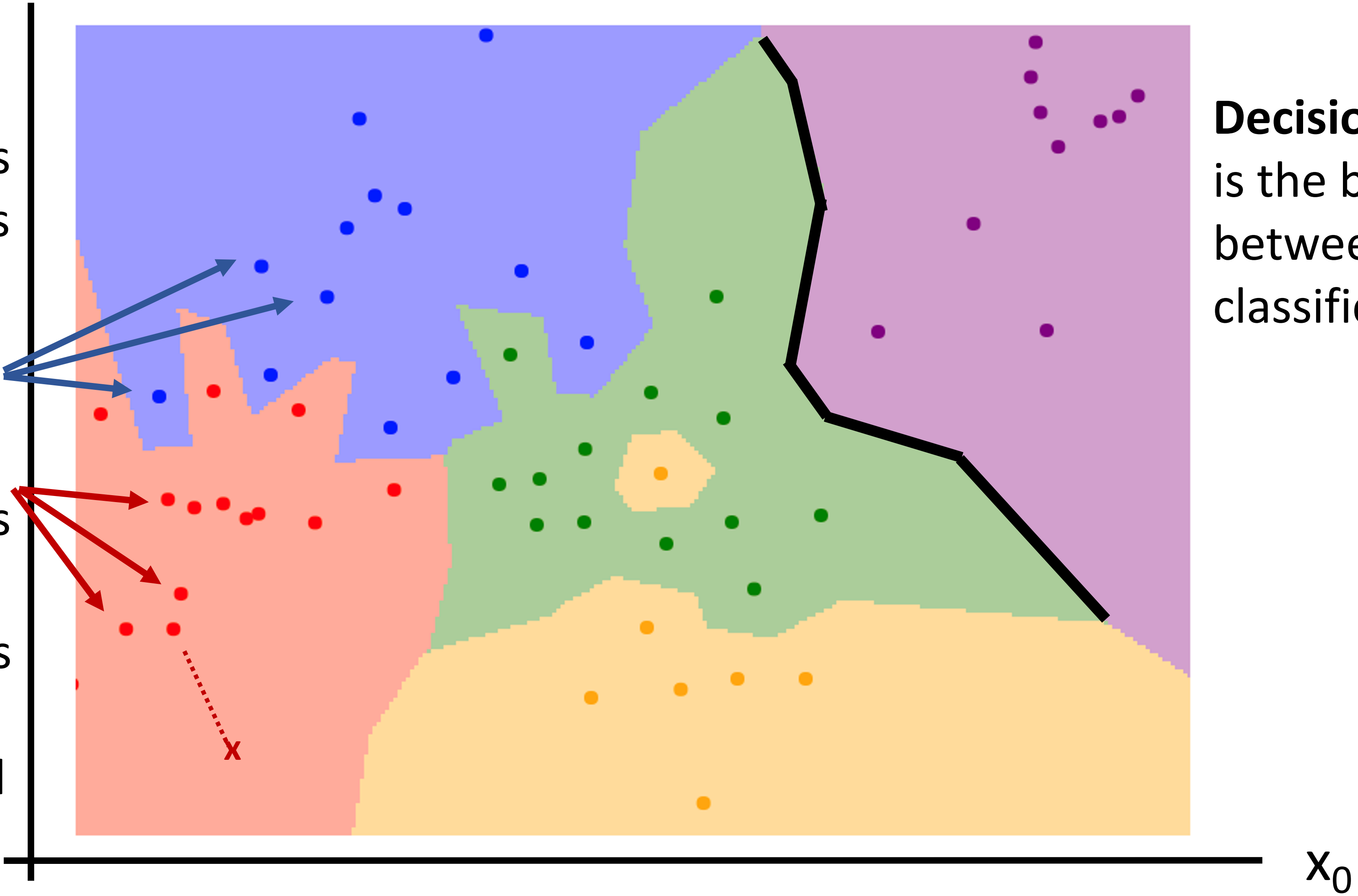
Slide from Justin Johnson

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

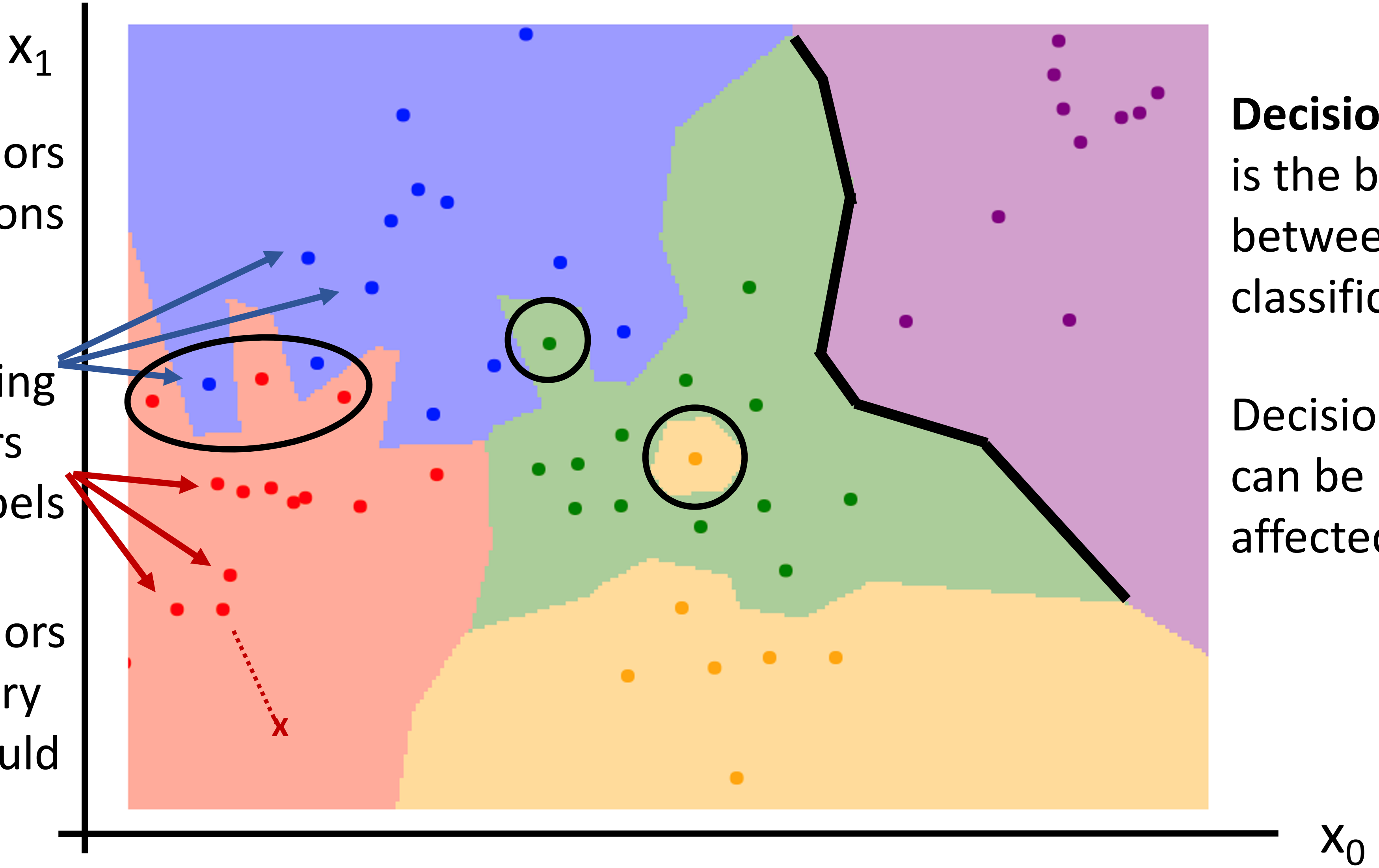
Slide from Justin Johnson

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

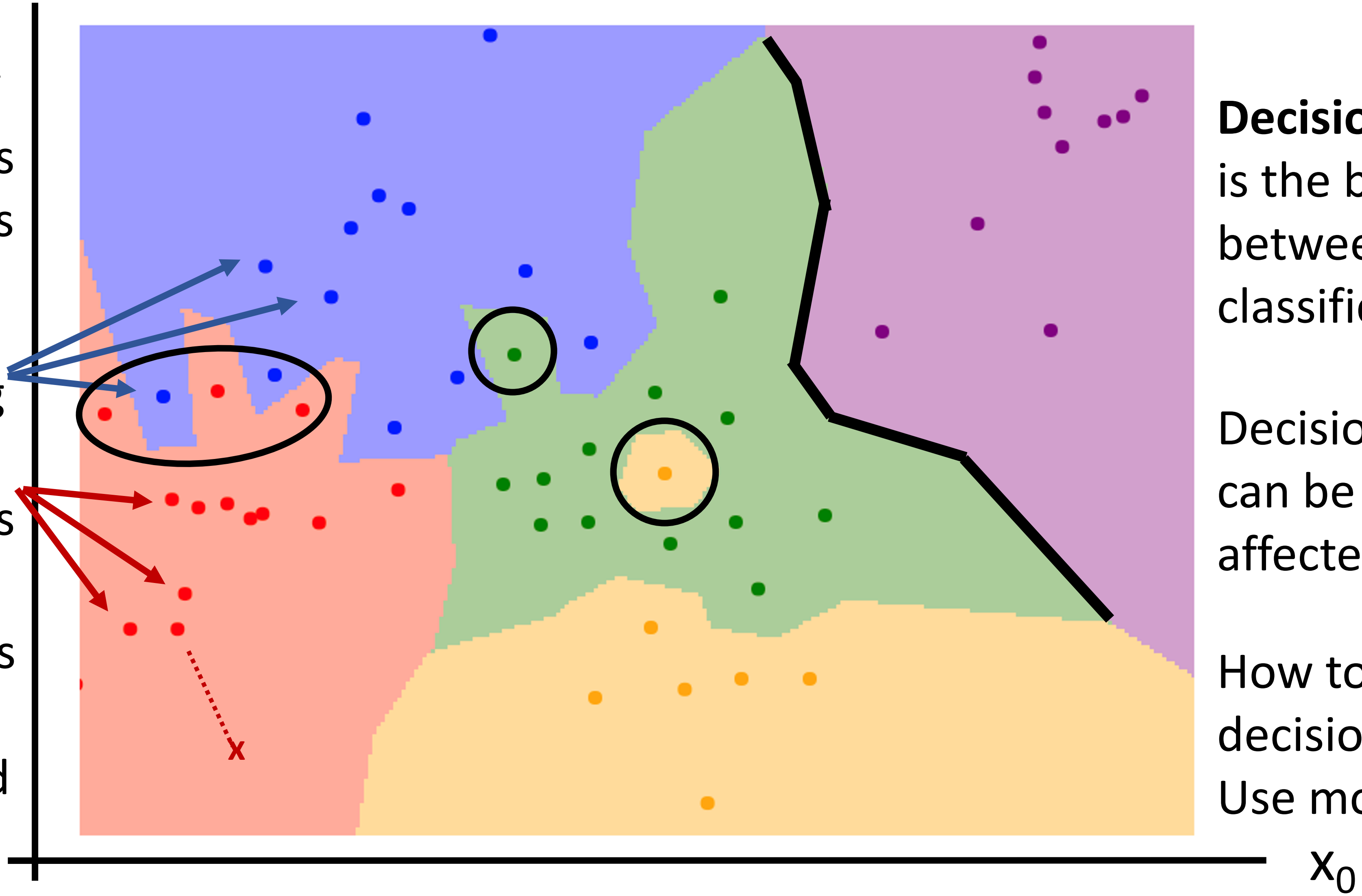
Slide from Justin Johnson

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

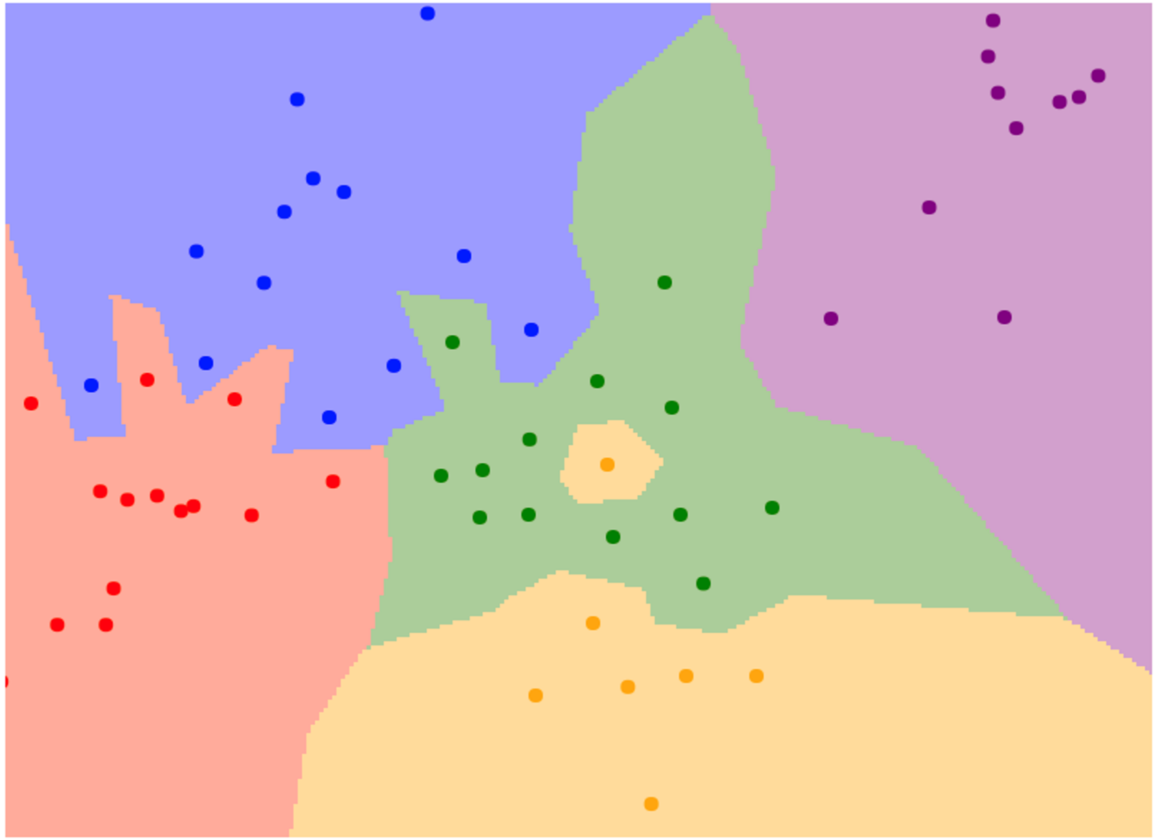
How to smooth out decision boundaries? Use more neighbors!

Slide from Justin Johnson

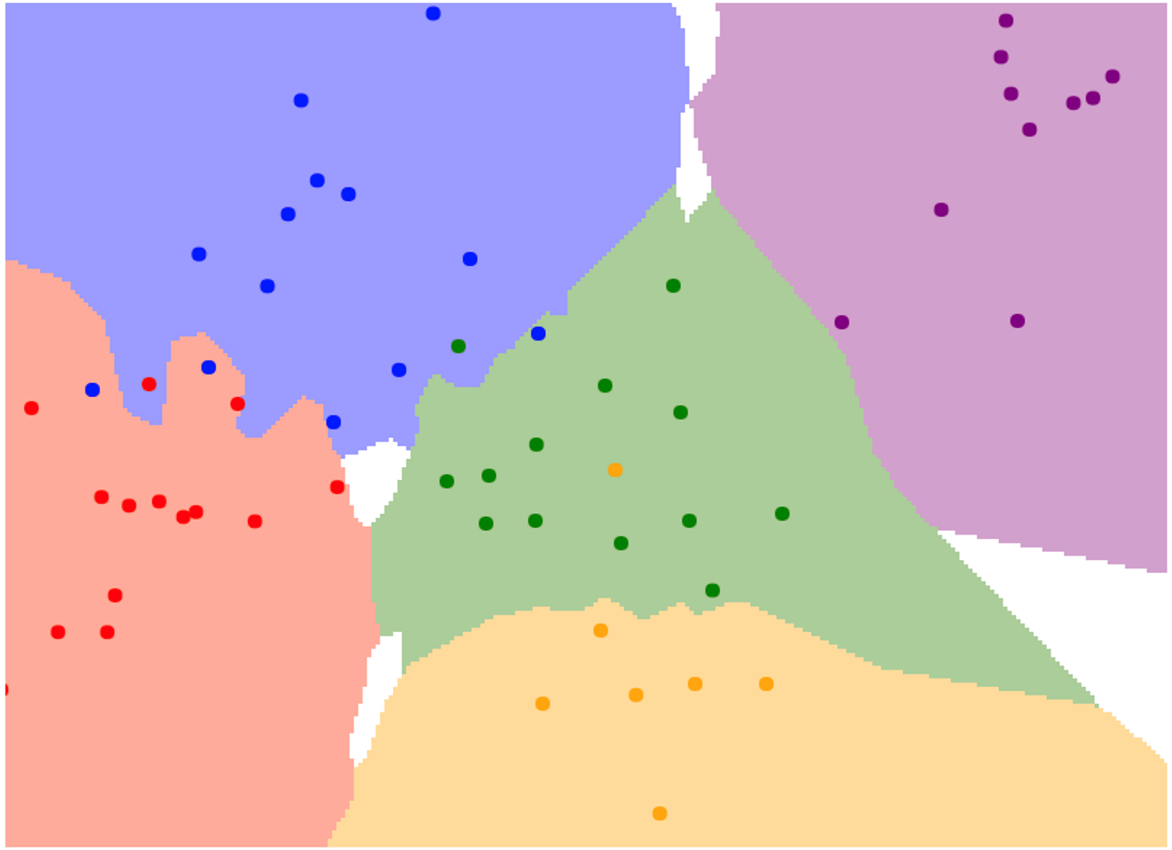
K-Nearest Neighbors

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

K = 1



K = 3

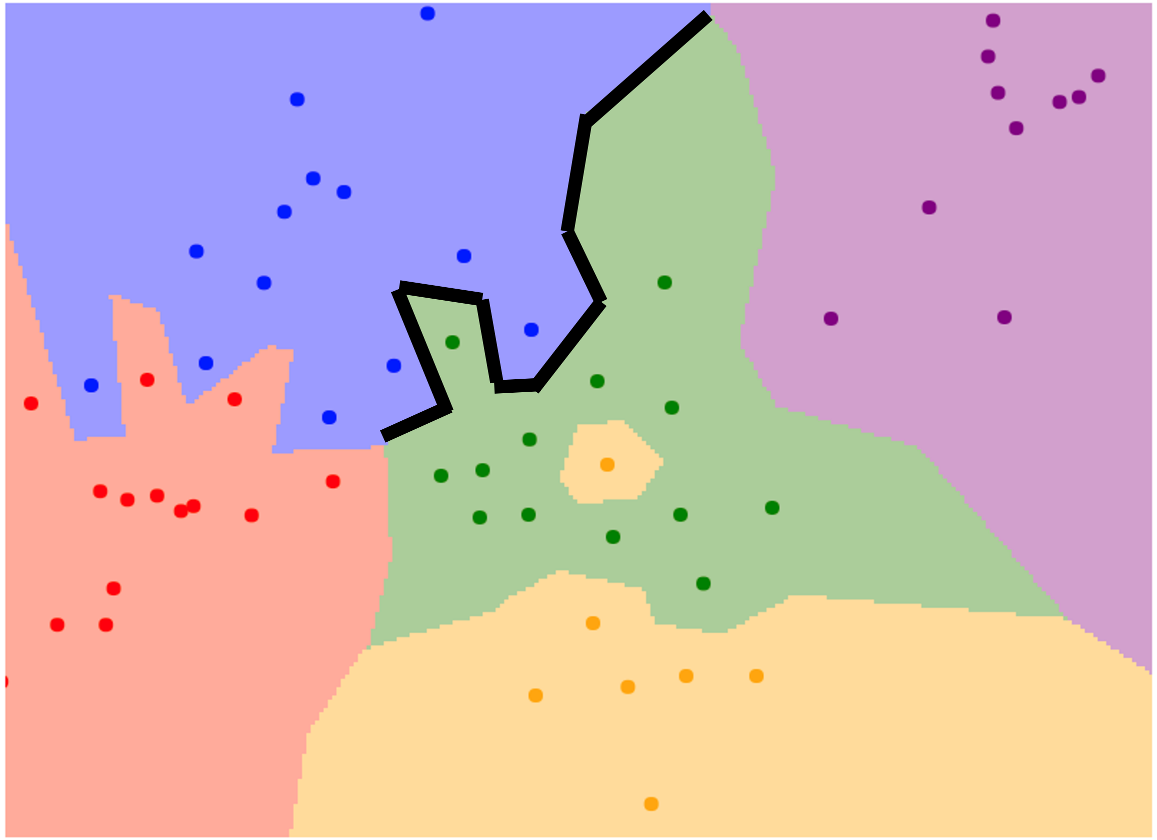


Slide from Justin Johnson

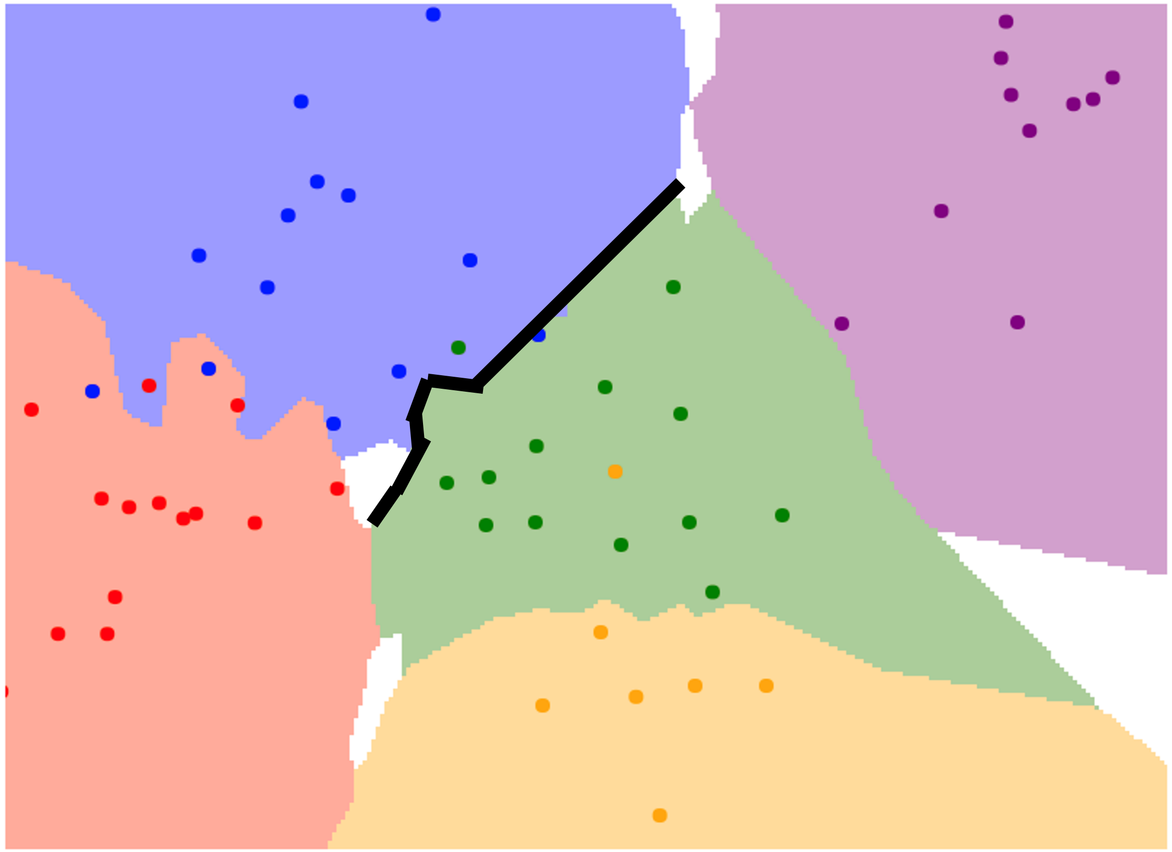
K-Nearest Neighbors

Using more neighbors helps smooth out rough decision boundaries

K = 1



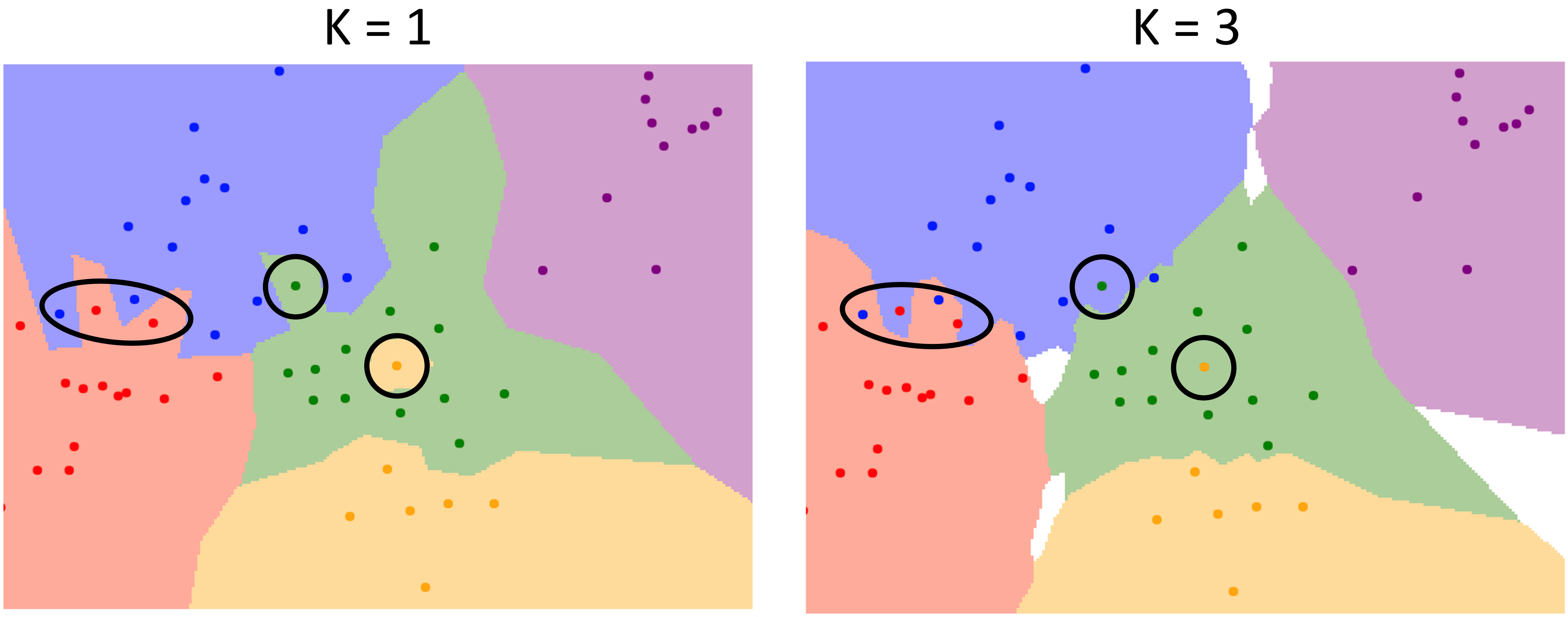
K = 3



Slide from Justin Johnson

K-Nearest Neighbors

Using more neighbors helps reduce the effect of outliers

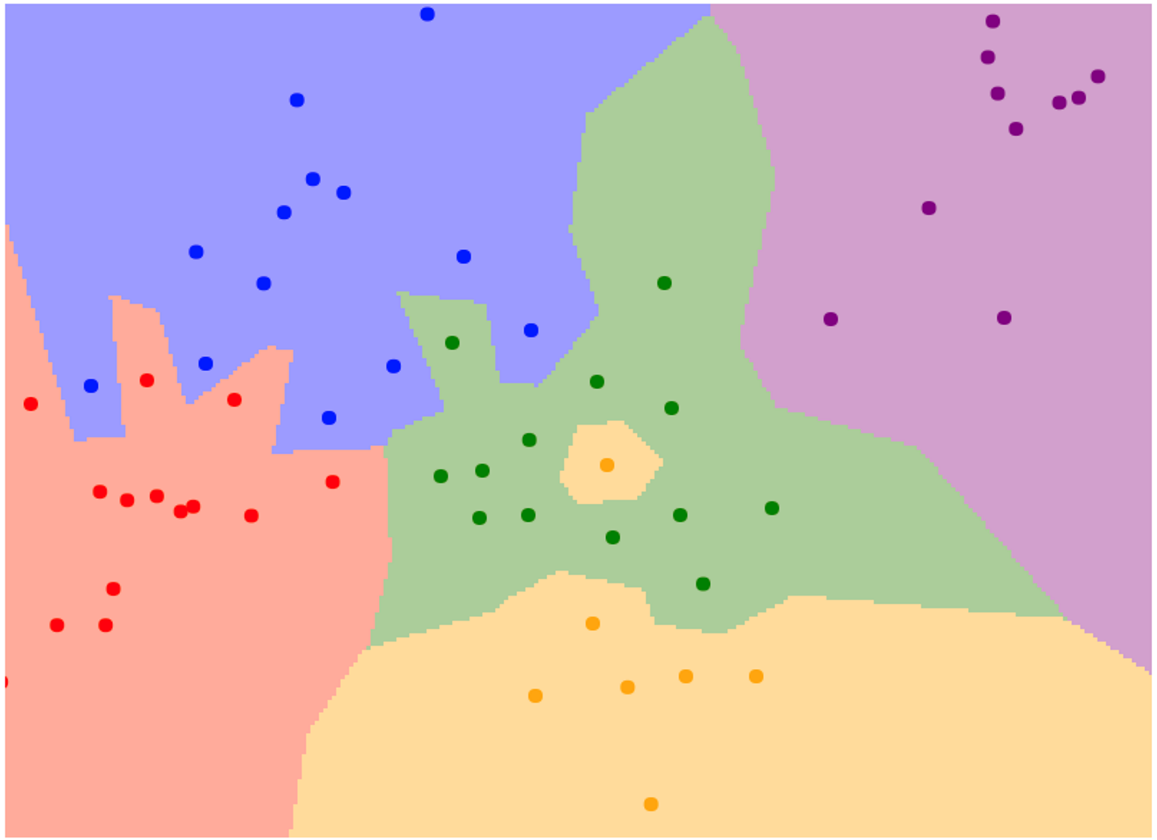


Slide from Justin Johnson

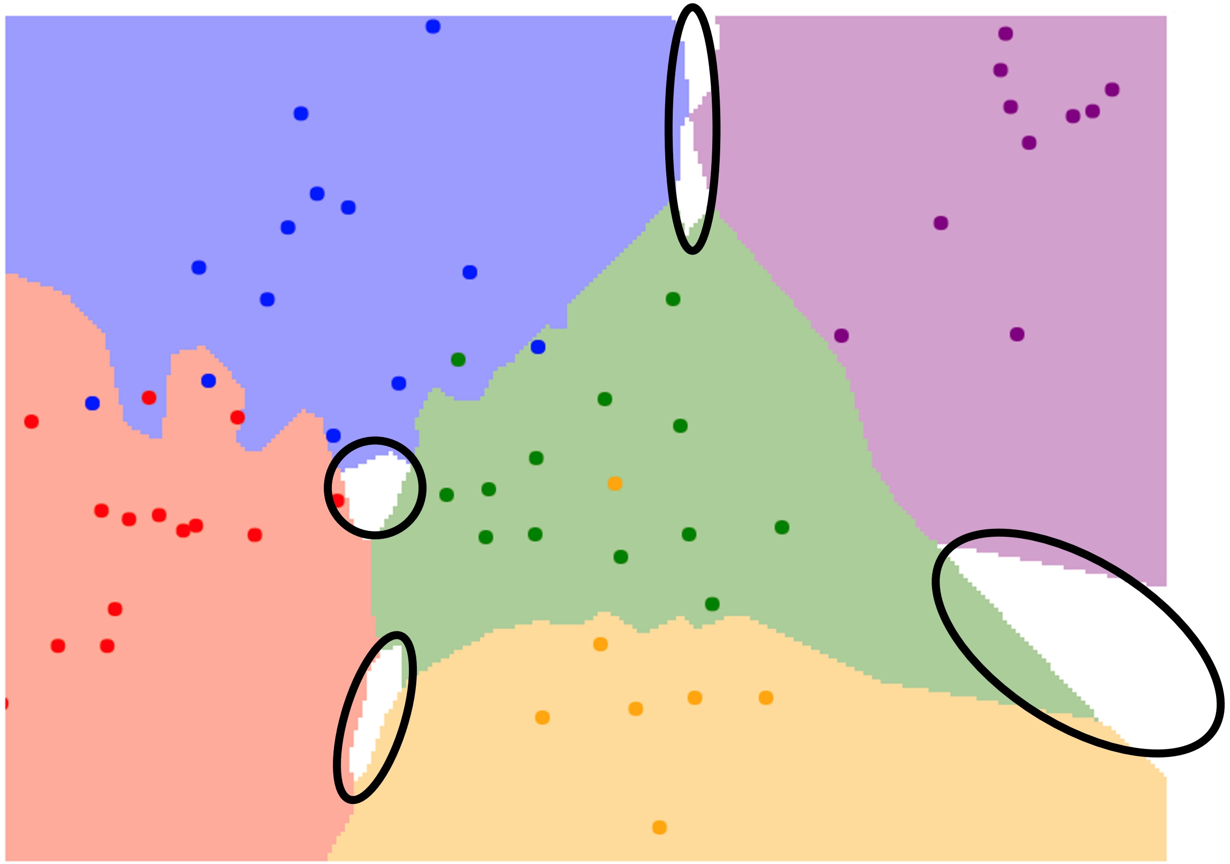
K-Nearest Neighbors

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

K = 1



K = 3

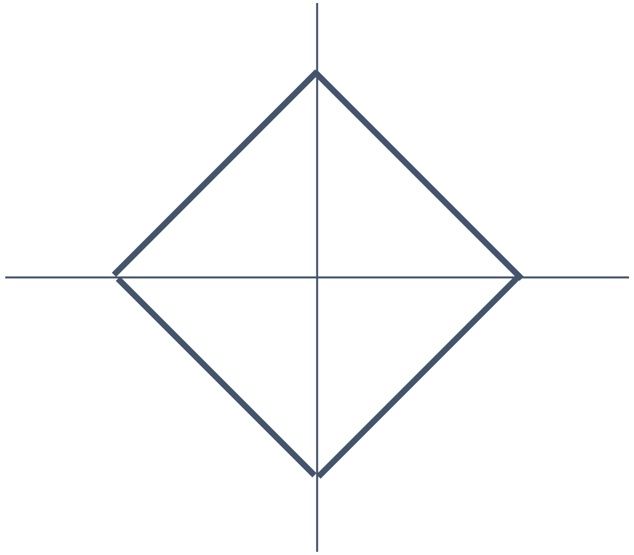


Slide from Justin Johnson

K-Nearest Neighbors: **Distance Metric**

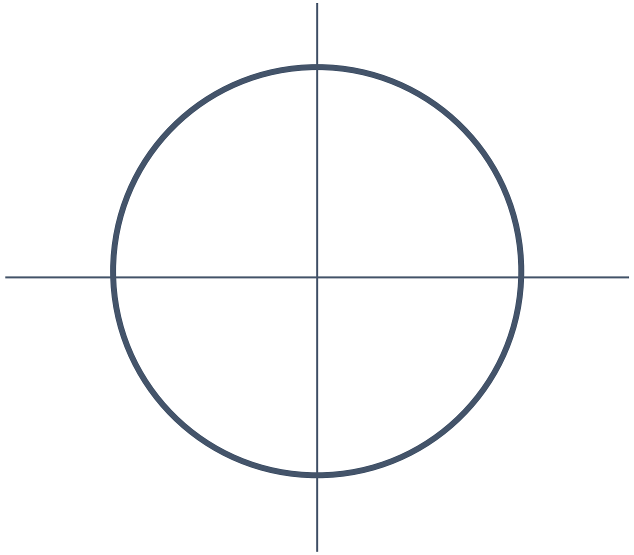
L1 (Manhattan) distance

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



L2 (Euclidean) distance

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

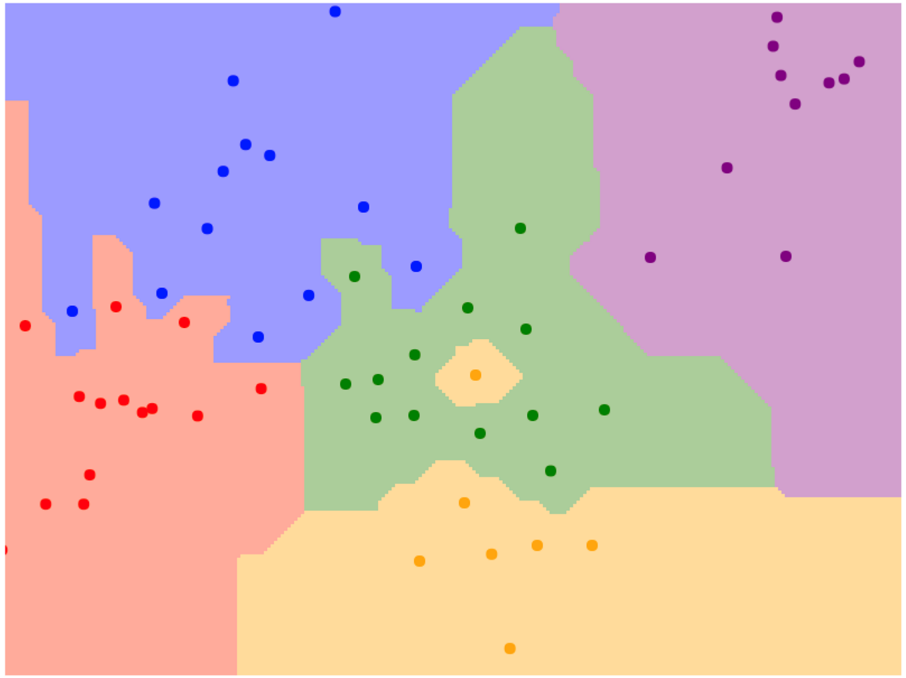


Slide from Justin Johnson

K-Nearest Neighbors: **Distance Metric**

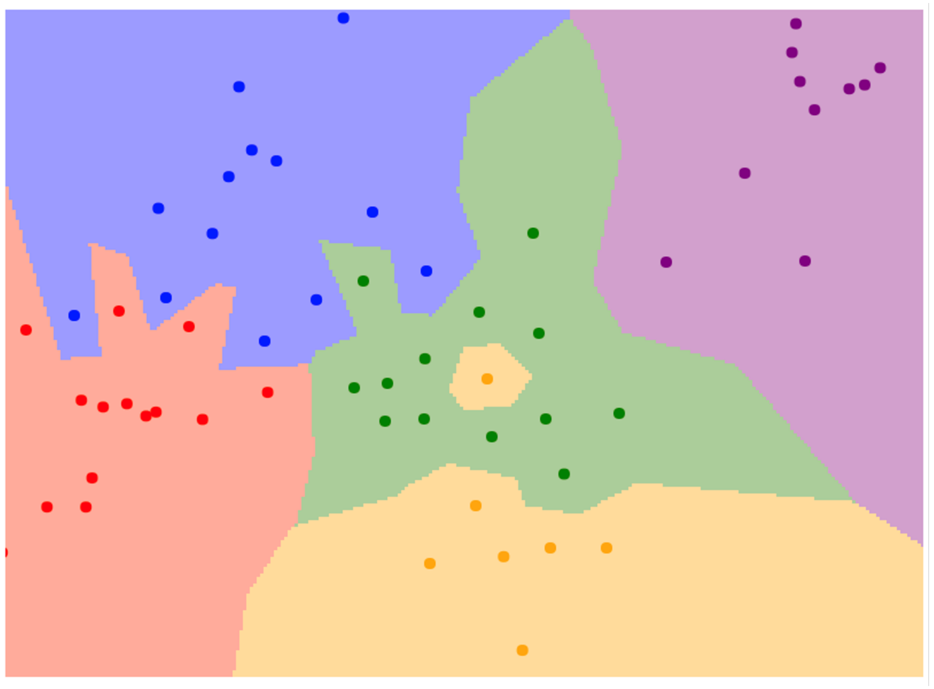
L1 (Manhattan) distance

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



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

Slide from Justin Johnson

K-Nearest Neighbors: **Distance Metric**

Can get quite creative with the distance function, e.g. bending energy (how much do you need to transform one example to look like another)

Fig. 1. Examples of two handwritten digits. In terms of pixel-to-pixel comparisons, these two images are quite different, but to the human observer, the shapes appear to be similar.

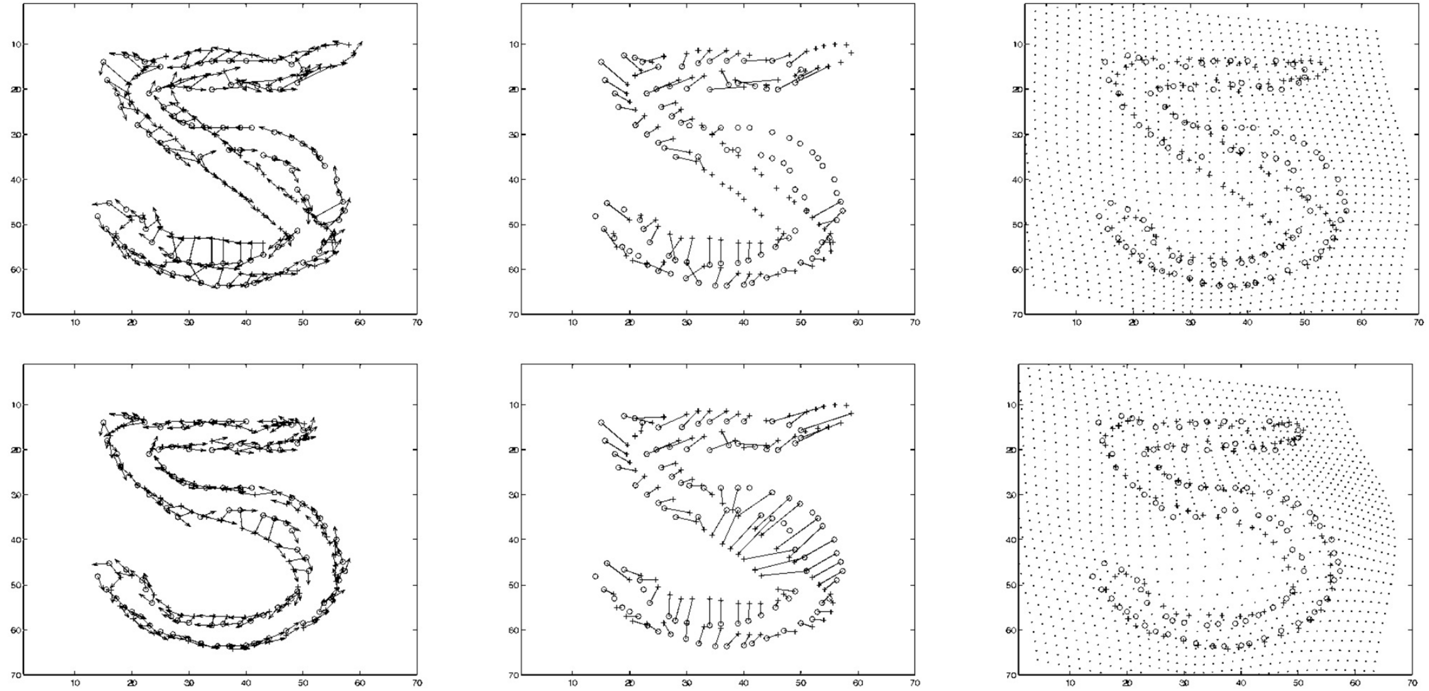


Fig. 4. Illustration of the matching process applied to the example of Fig. 1. Top row: 1st iteration. Bottom row: 5th iteration. Left column: estimated correspondences shown relative to the transformed model, with tangent vectors shown. Middle column: estimated correspondences shown relative to the untransformed model. Right column: result of transforming the model based on the current correspondences; this is the input to the next iteration. The grid points illustrate the interpolated transformation over $\mathbb{R}^2$. Here, we have used a regularized TPS model with $\lambda_o = 1$.

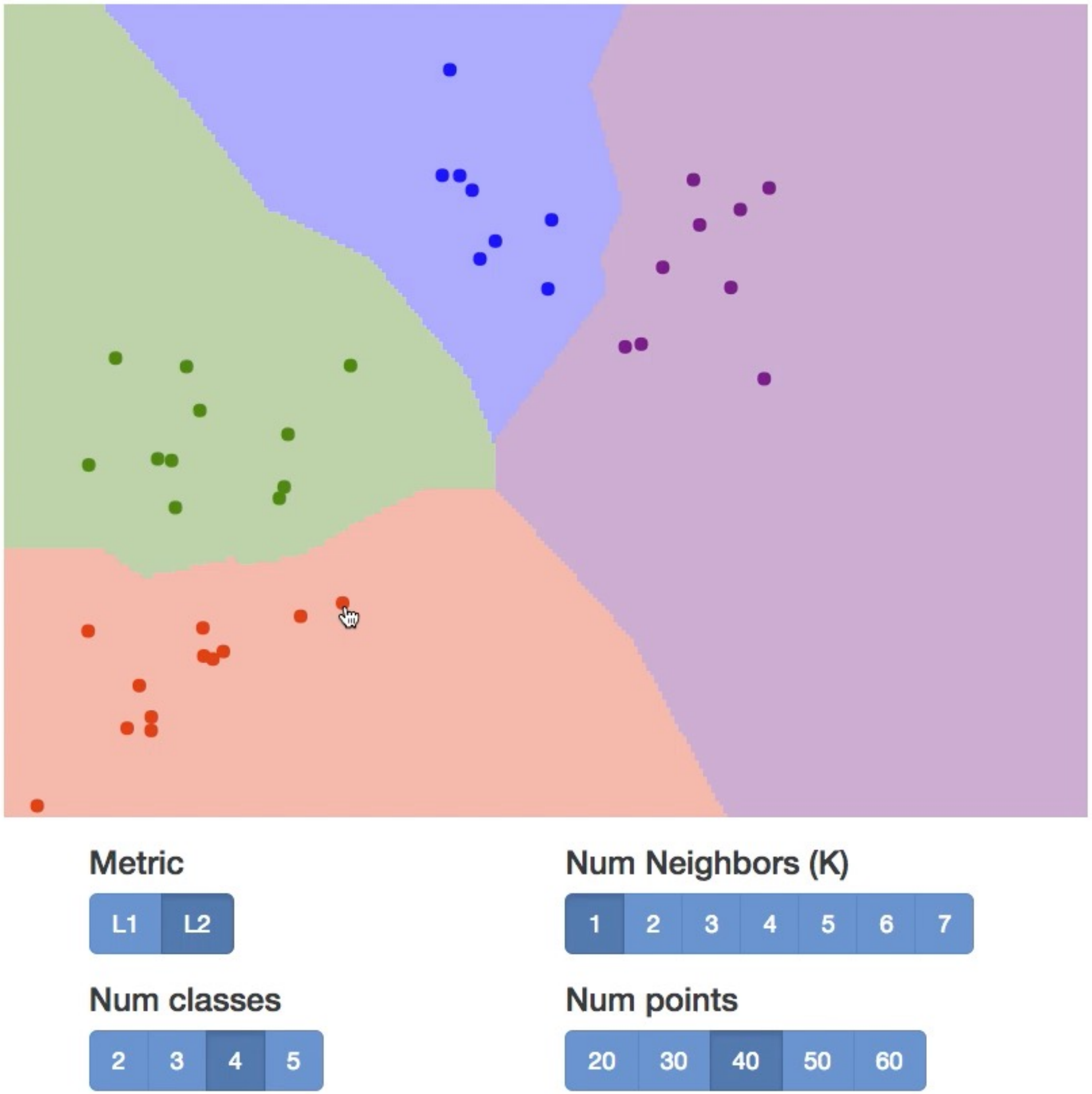
[Shape Matching and Object Recognition Using Shape Context](#) Serge Belongie, Jitendra Malik, and Jan Puzicha, PAMI 2002

K-Nearest Neighbors: Web Demo

Interactively move points around and see decision boundaries change

Play with L1 vs L2 metrics

Play with changing number of training points, value of K



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

Slide from Justin Johnson

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

Slide from Justin Johnson

Hyperparameters

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

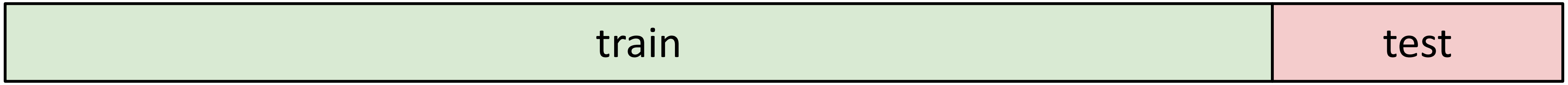
These are examples of **hyperparameters**: choices about our learning algorithm that we don't learn from the training data; instead we set them at the start of the learning process

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

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data (in general, memorization is sufficient for acing the train set)

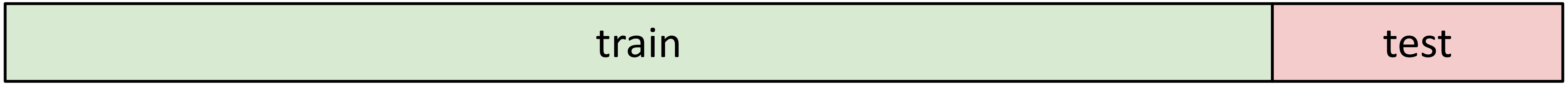


Slide from Justin Johnson

Setting Hyperparameters

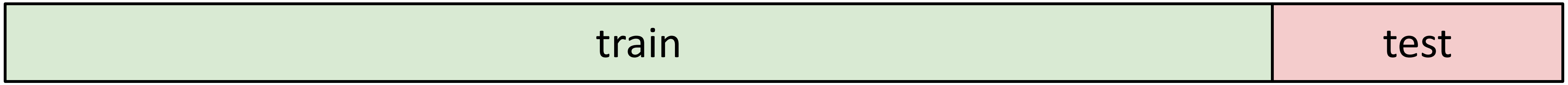
Idea #1: Choose hyperparameters that work best on the training data

BAD: K = 1 always works perfectly on training data (in general, memorization is sufficient for acing the train set)



Idea #2: Choose hyperparameters that work best on test data

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

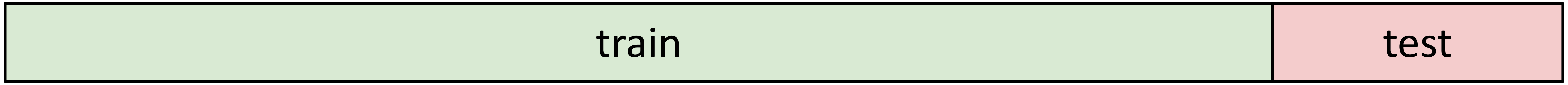


Slide from Justin Johnson

Setting Hyperparameters

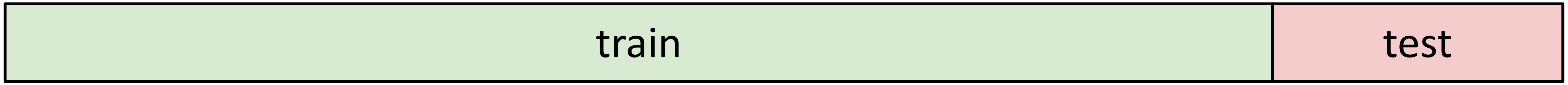
Idea #1: Choose hyperparameters that work best on the training data

BAD: K = 1 always works perfectly on training data (in general, memorization is sufficient for acing the train set)



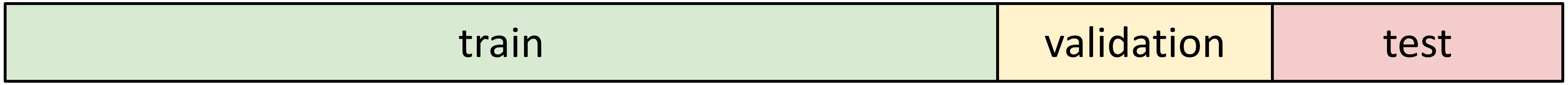
Idea #2: Choose hyperparameters that work best on test data

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



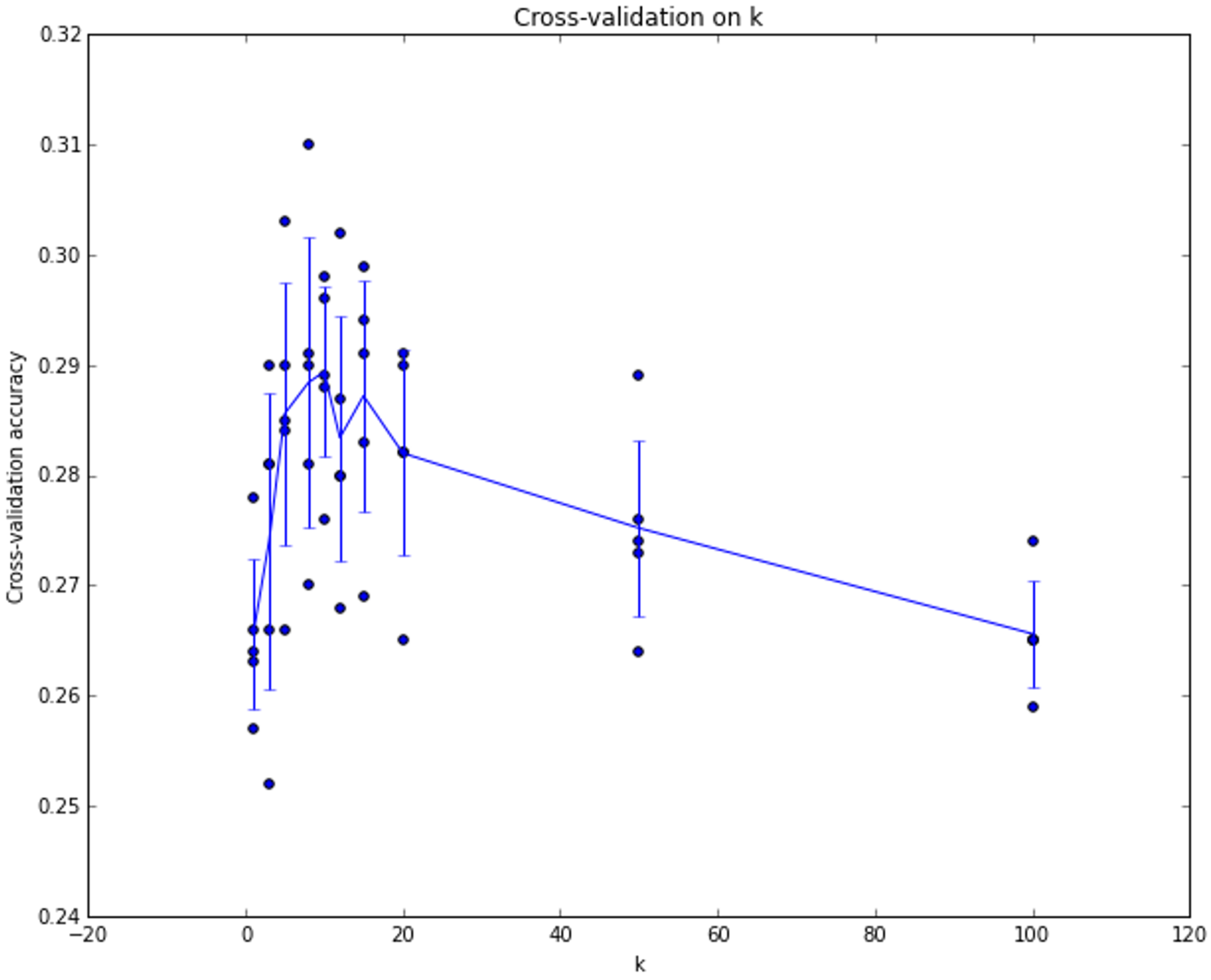
Idea #3: Split dataset into **train** and **val**; choose hyperparameters on val and evaluate on test.

Better!



Slide from Justin Johnson

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)

Slide from Justin Johnson

K-Nearest Neighbor: Universal Approximation

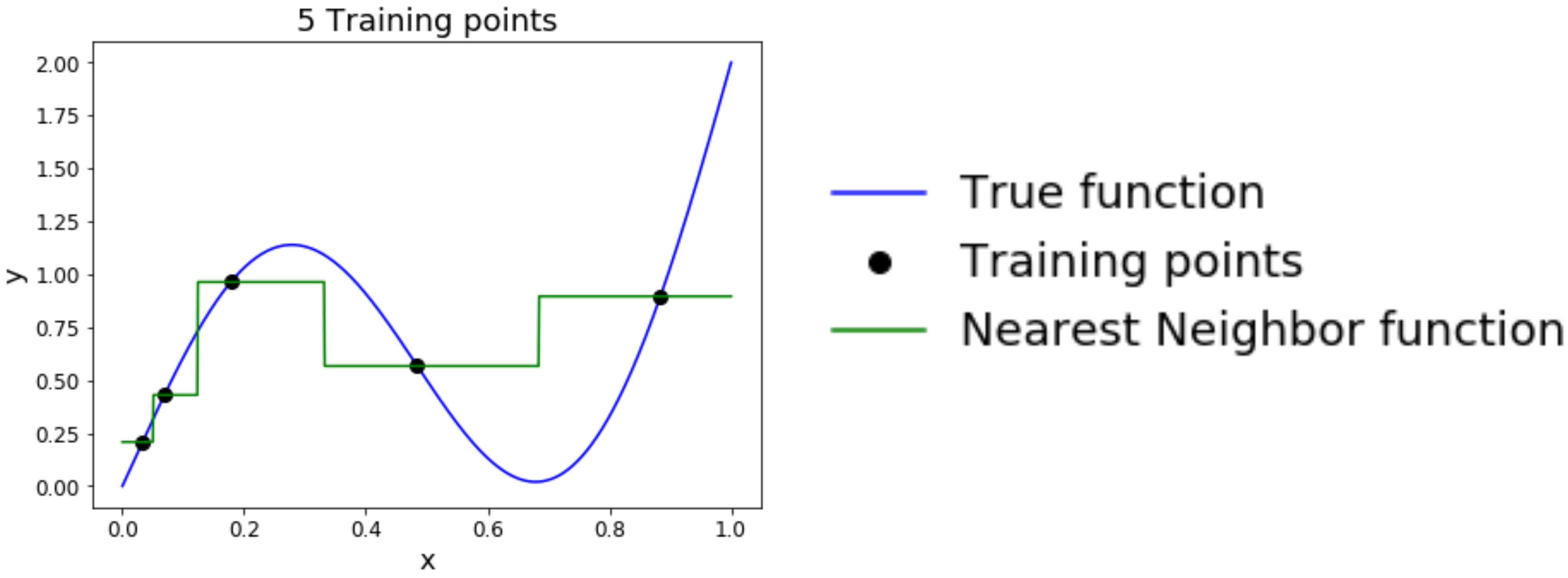
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!

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

Slide from Justin Johnson

K-Nearest Neighbor: **Universal Approximation**

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

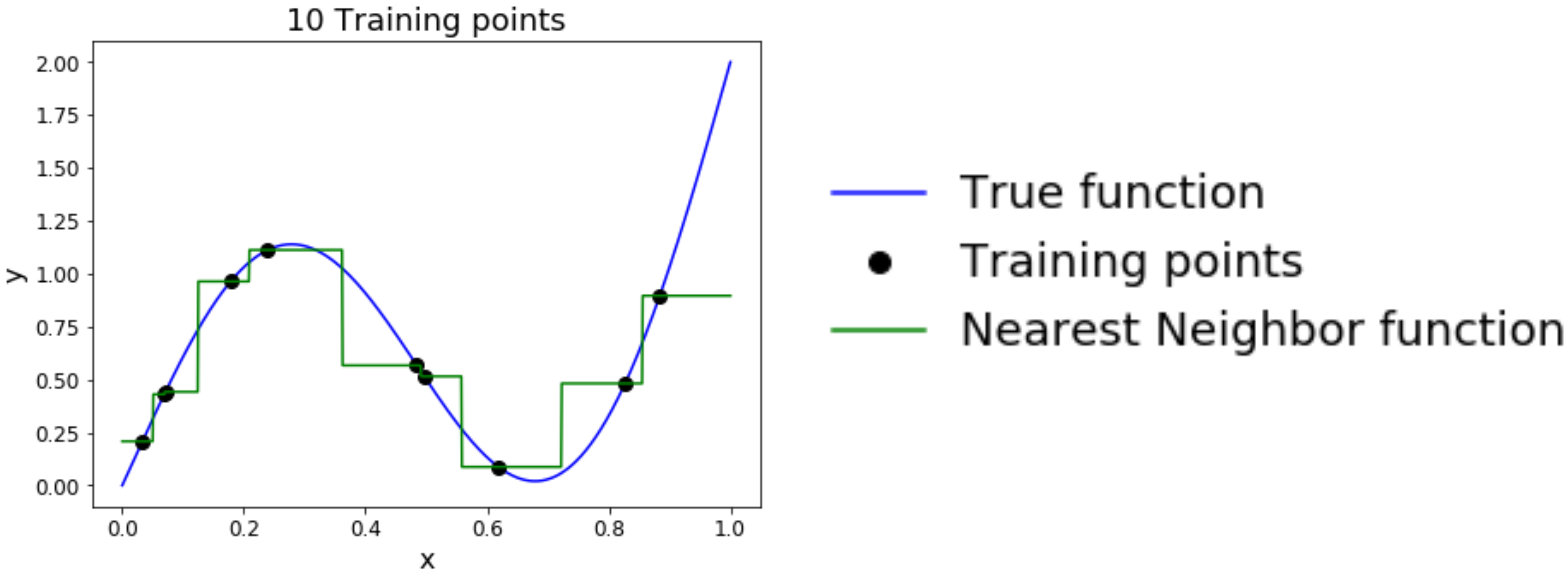


(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

Slide from Justin Johnson

K-Nearest Neighbor: **Universal Approximation**

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

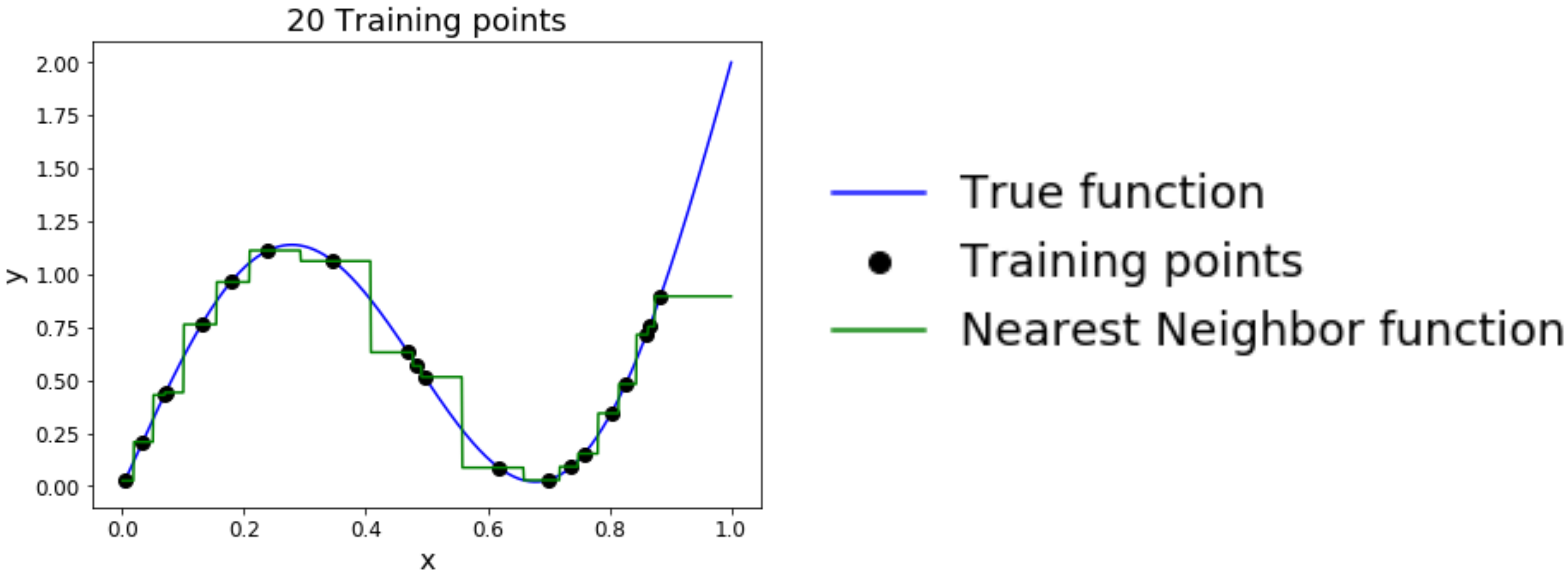


(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

Slide from Justin Johnson

K-Nearest Neighbor: **Universal Approximation**

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

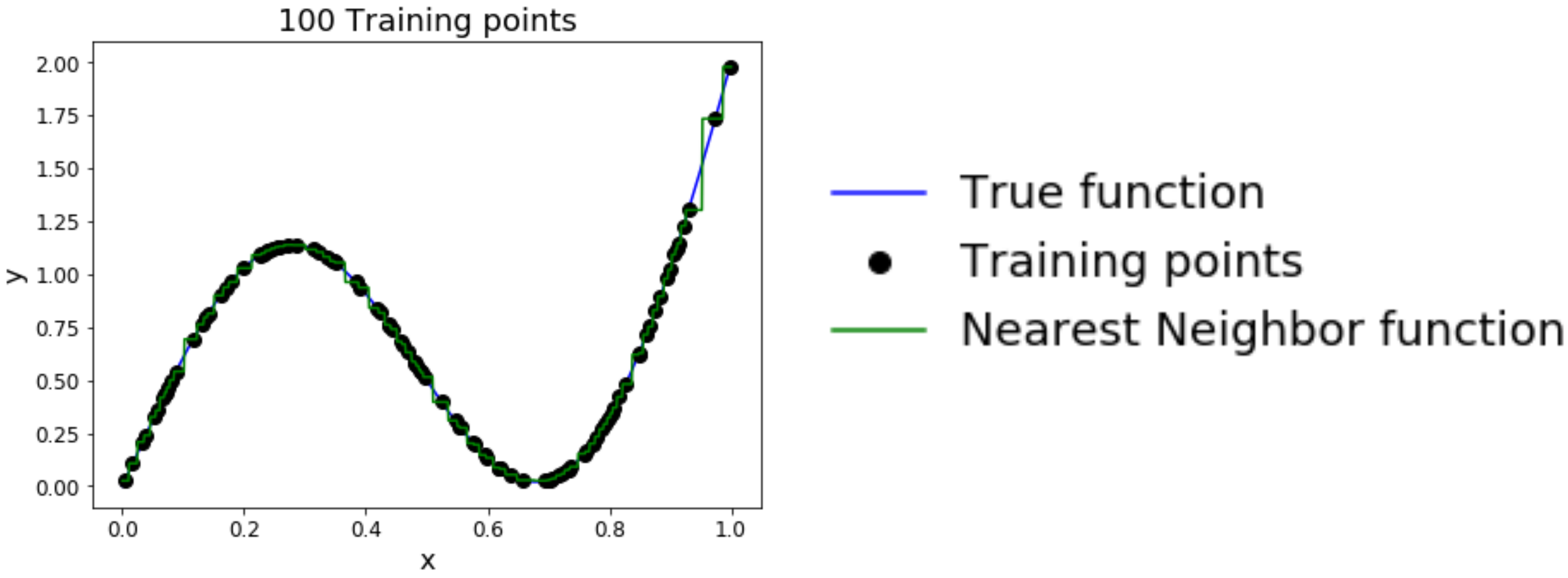


(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

Slide from Justin Johnson

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.

Slide from Justin Johnson

K-Nearest Neighbor on raw pixels is **seldom used**

- 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)

Slide from Justin Johnson

Nearest Neighbor with ConvNet features works well!

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

Slide from Justin Johnson

Can transfer more than just label!

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

Slide from Justin Johnson

Can transfer more than just label!

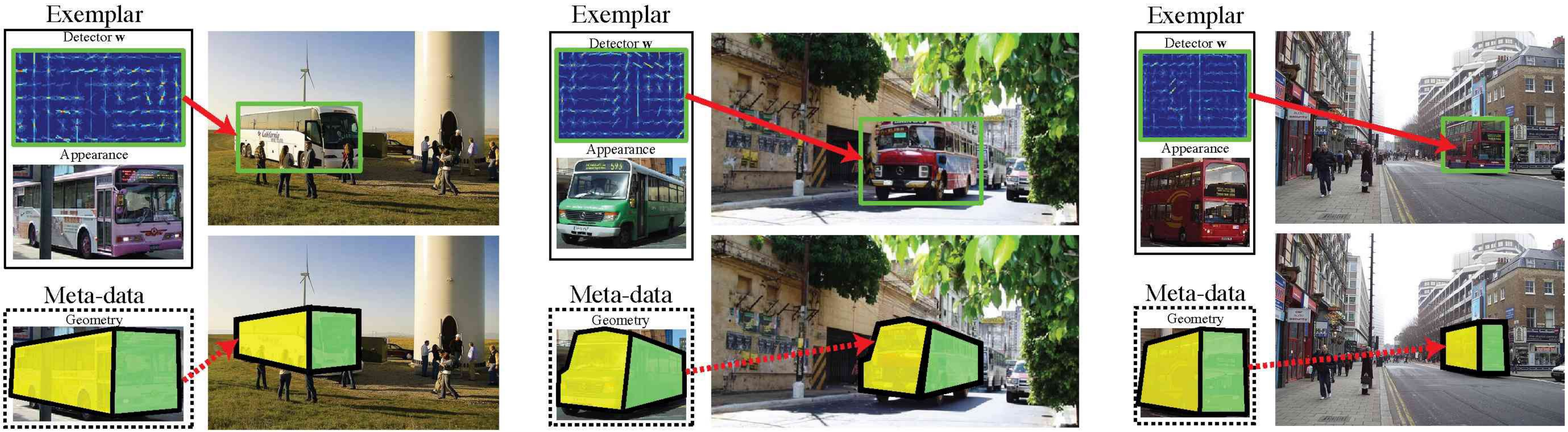
Image Captioning with Nearest Neighbor

Method	BLEU 4
ME + DMSM [8]	**56.7**
LRCN [6]	53.4
Vinyals et al. [35]	53.8
Xu et al. [36]	52.3
m-RNN [25]	54.3
MLBL [18], [19]	51.7
NeuralTalk [16]	44.6
fc7-fine (CIDEr)	54.2 (2)
Human	47.1

Outperformed many other End-to-end models at the time.

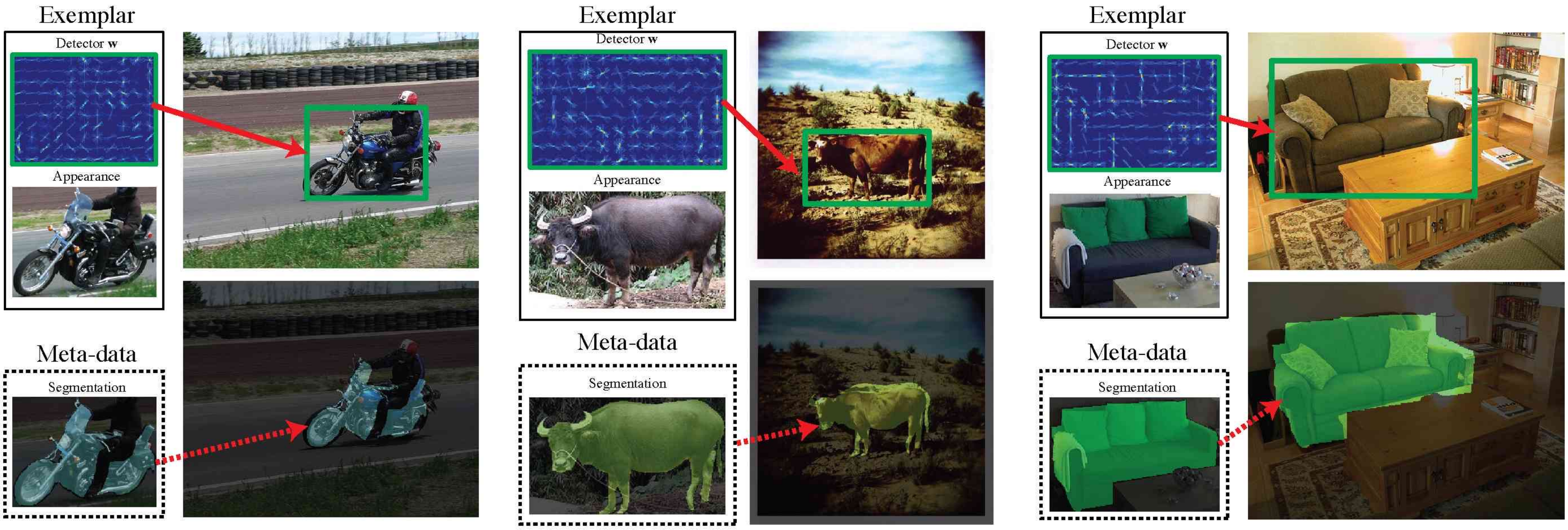
Devlin et al, "Exploring Nearest Neighbor Approaches for Image Captioning", 2015

Can transfer more than just label!



Malisiewicz et al, "[Ensemble of Exemplar-SVMs for Object Detection and Beyond]()", ICCV 2011

Can transfer more than just label!



Malisiewicz et al, "[Ensemble of Exemplar-SVMs for Object Detection and Beyond](#)", ICCV 2011

Lecture overview

- Different problems in computer vision
- Supervised classification
- Beyond supervised classification: A taxonomy of prediction problems and types of supervision
- Image classification
- k-Nearest Neighbors