Image classification

Image Classification

- It is the task of assigning an input image one label from a fixed set of categories.
- This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications.

Image Classification

- As shown in the image, keep in mind that to a computer an image is represented as one large 3-dimensional array of numbers.
- There is a huge gap between the semantic idea of a cat and those numbers the computer are seeing

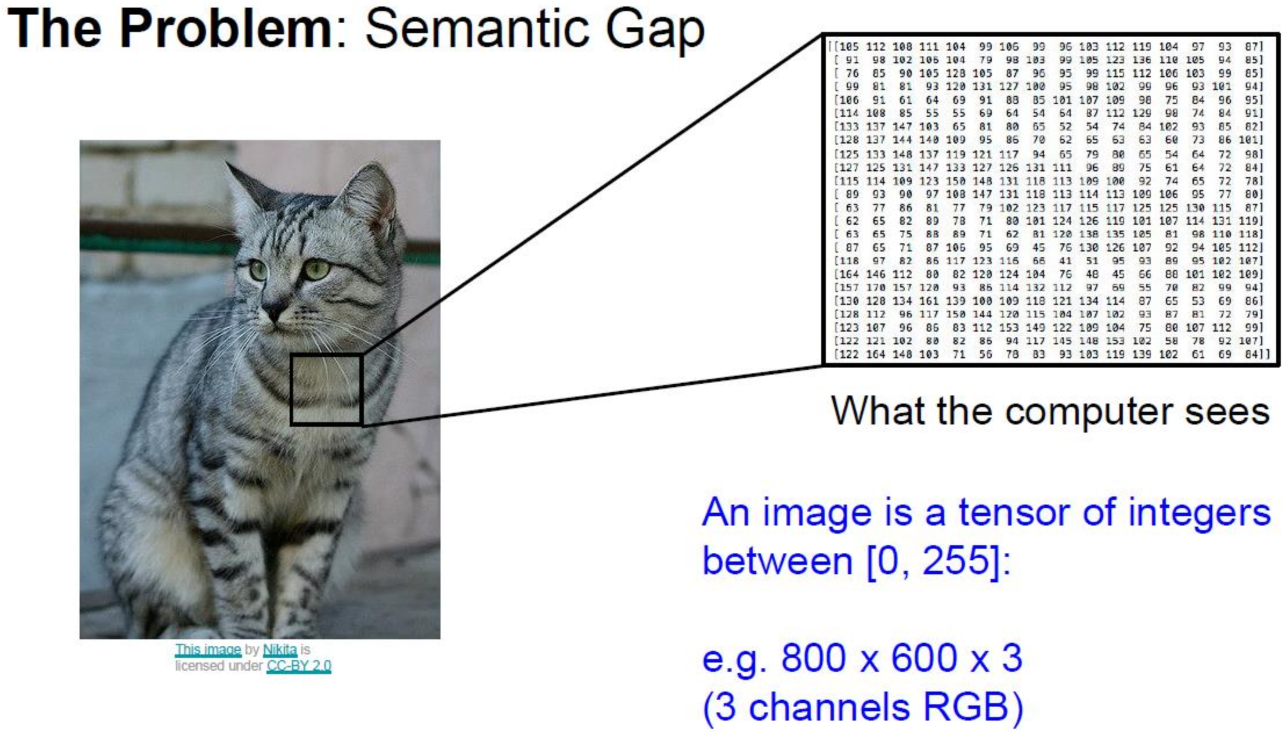
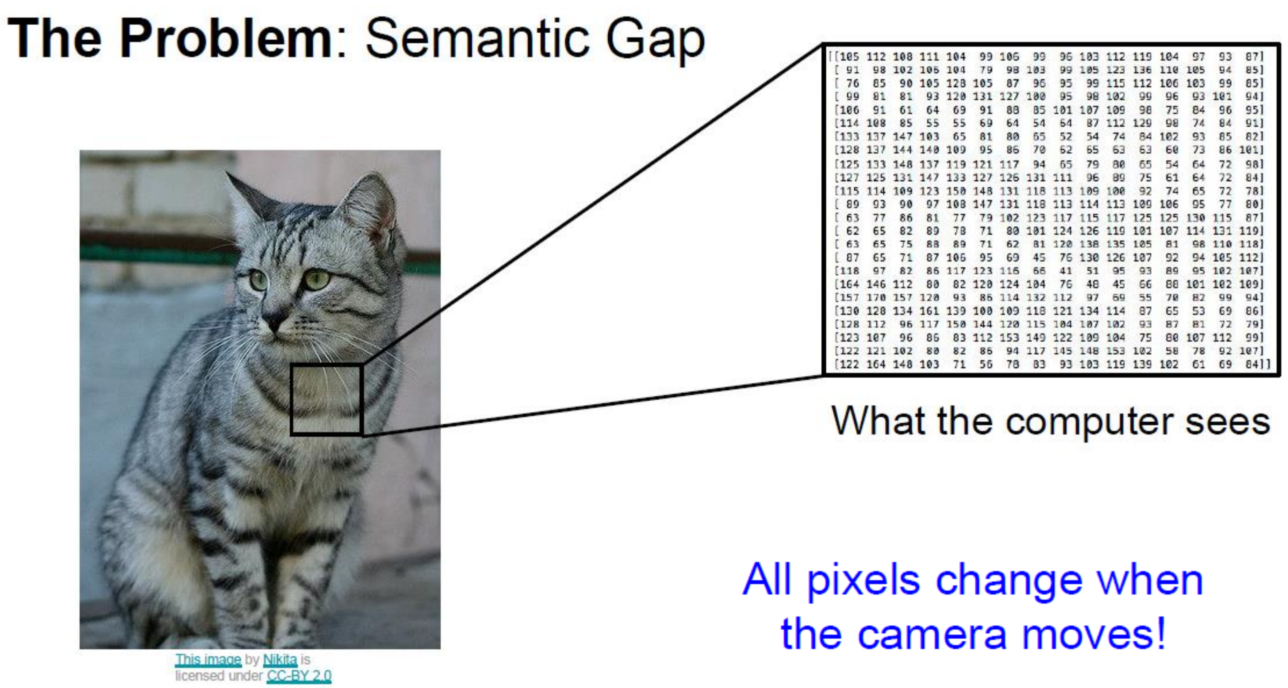


Image Classification

- As shown in the image, keep in mind that to a computer an image is represented as one large 3-dimensional array of numbers.
- There is a huge gap between the semantic idea of a cat and those numbers the computer are seeing



Challenges

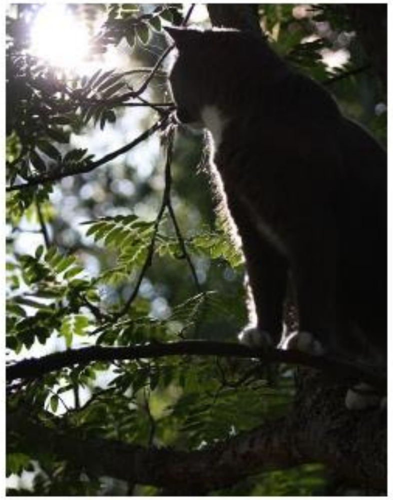
- Since this task of recognizing a visual concept is relatively trivial for a human to perform, it is worth considering the challenges involved from the perspective of a Computer Vision algorithm.

Challenges: Illumination

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

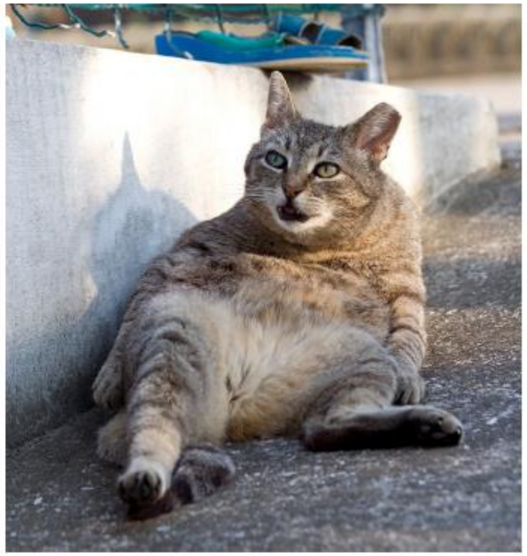


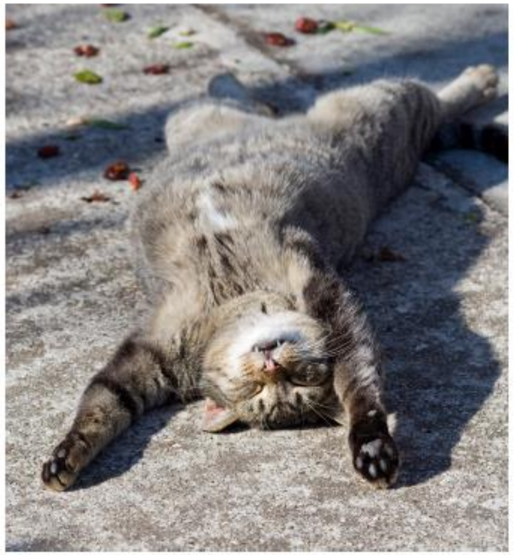
This image is CC0 1.0 public domain

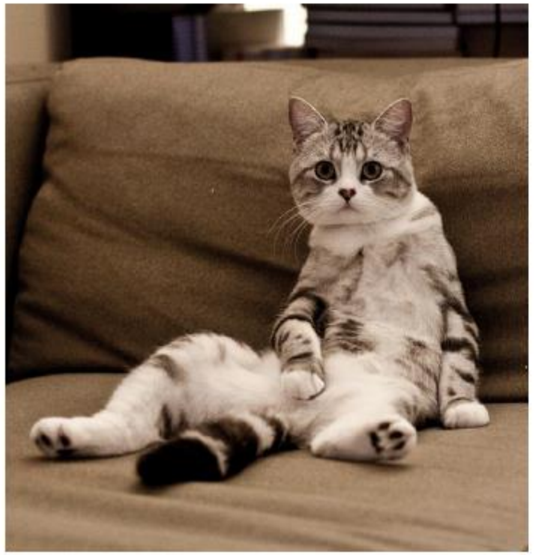
Challenges: Background Clutter

Challenges: Occlusion

Challenges: Deformation

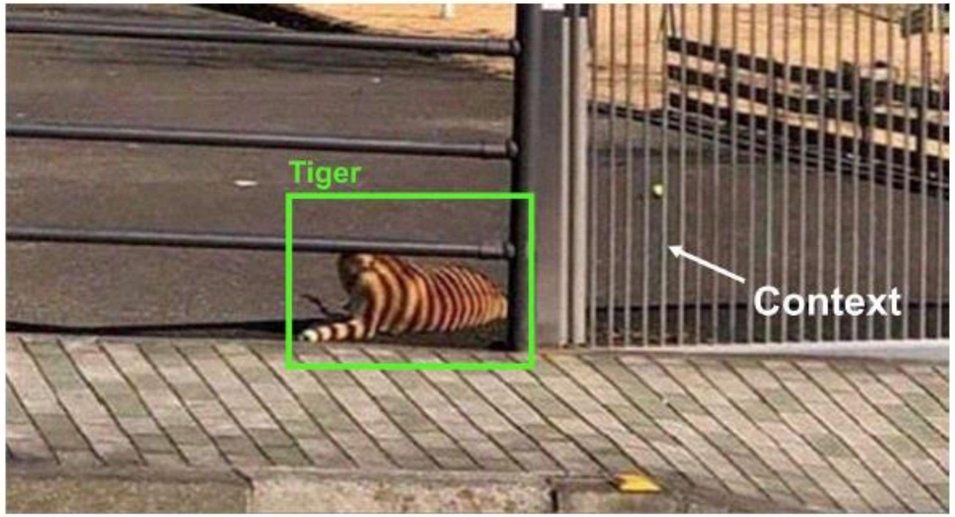




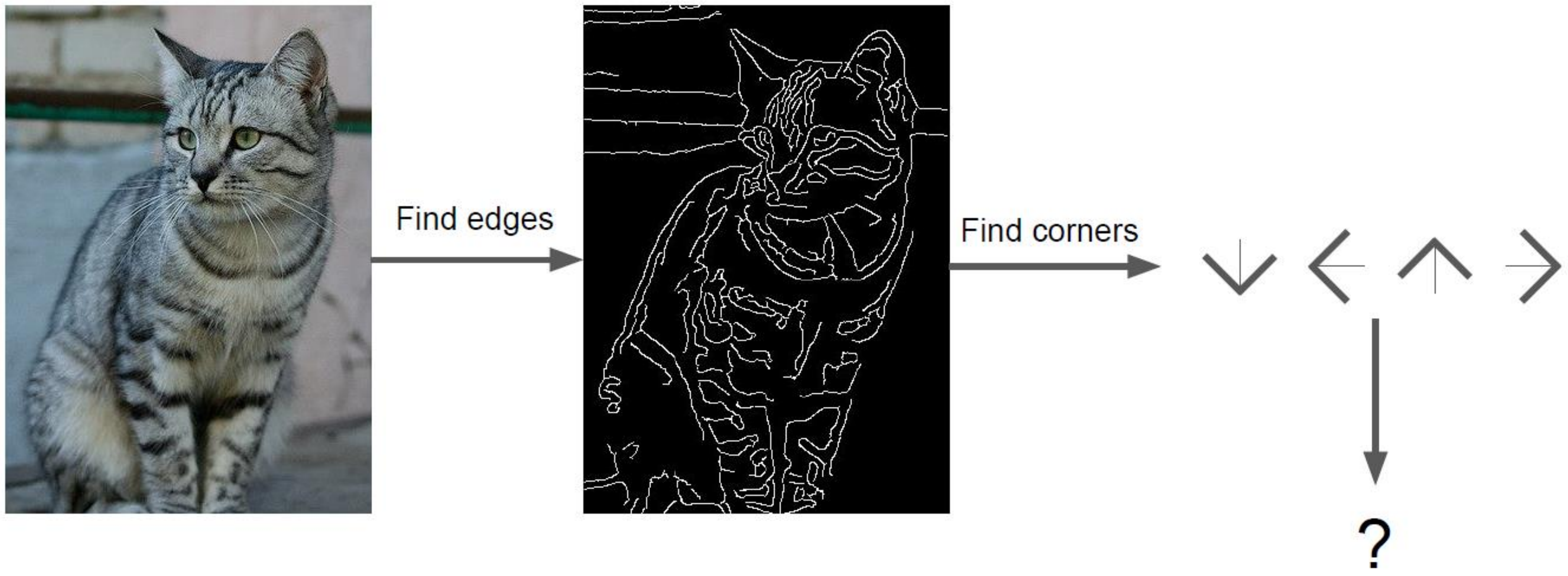


Challenges: Intraclass variation

Challenges: Context



Attempts have been made



Data driven approach

- To classify images into distinct categories, like identifying cats, a direct algorithm is challenging.
- Instead, we use a data-driven approach. We provide the computer with many labeled examples of each class and develop learning algorithms.
- These algorithms analyze the examples, learning about the visual appearance of each category.

The data driven approach:

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

Data driven approach

The image classification pipeline

- **Input**: Our input consists of a set of N images, each labeled with one of K different classes. We refer to this data as the training set.
- **Learning**: Our task is to use the training set to learn what every one of the classes looks like. We refer to this step as training a classifier or learning a model.
- **Prediction**: In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier. Intuitively, we're hoping that a lot of the predictions match up with the true answers (which we call the ground truth).

The image classification pipeline

```
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

Nearest Neighbor Classifier

Nearest Neighbor Classifier

- As our first approach, we will develop what we call a Nearest Neighbor Classifier.
- This classifier has nothing to do with Convolutional Neural Networks and it is very rarely used in practice, but it will allow us to get an idea about the basic approach to an image classification problem.

Nearest Neighbor Classifier

Training data with labels

?

query data

Distance Metric $\Big|$  ,  $\Big| \rightarrow \mathbb{R}$

Distance Metric to compare images

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

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

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

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```

Nearest Neighbor classifier

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

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```

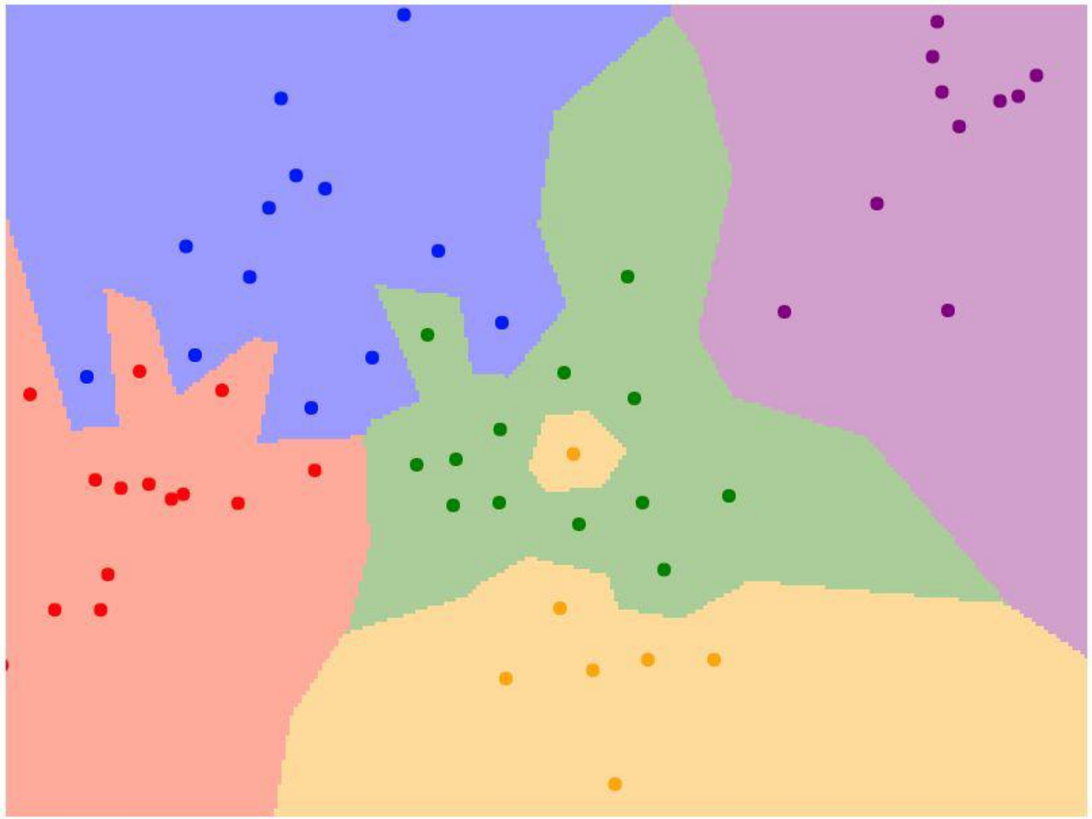
Nearest Neighbor classifier

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

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

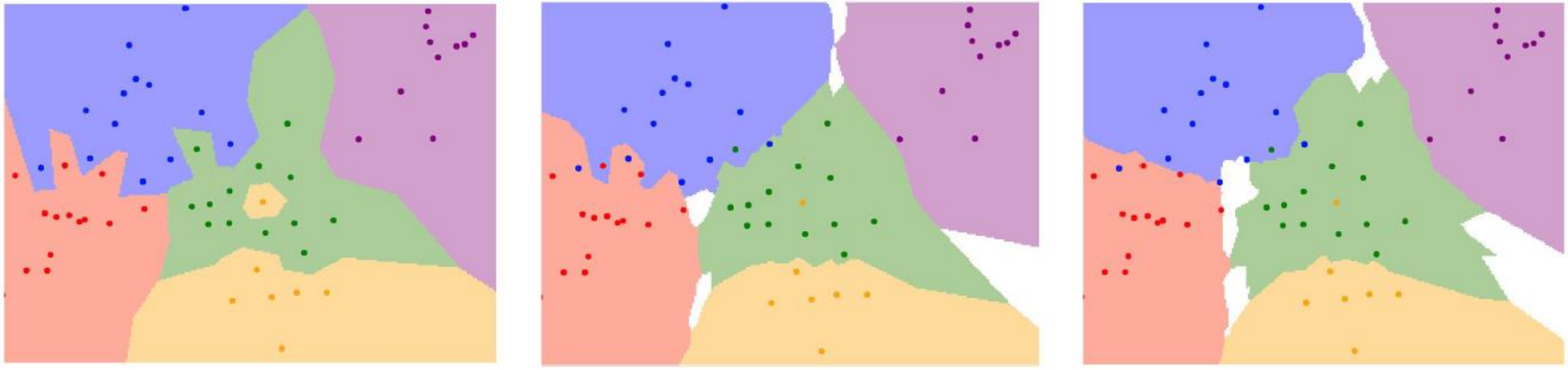
This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok

1-nearest neighbor



K-Nearest Neighbors

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



K = 1 K = 3 K = 5

K-Nearest Neighbors

- The idea is very simple: instead of finding the single closest image in the training set, we will find the top k closest images, and have them vote on the label of the test image.

- Intuitively, higher values of k have a smoothing effect that makes the classifier more resistant to outliers

The choice of distance

- There are many other ways of computing distances between vectors.

- Another common choice could be to instead use the L2 distance, which has the geometric interpretation of computing the Euclidean distance between two vectors.

- The distance takes the form:

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

Hyperparameters

What is the best value of k to use?

What is the best distance to use?

- These are hyperparameters: choices about the algorithms themselves.
- Every problem/dataset-dependent.

- You might be tempted to suggest that we should try out many different values and see what works best. That is a fine idea and that’s indeed what we will do, but this must be done very carefully.

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

train

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

BAD: No idea how algorithm will perform on new data

train | test

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

Better!

train | validation | test

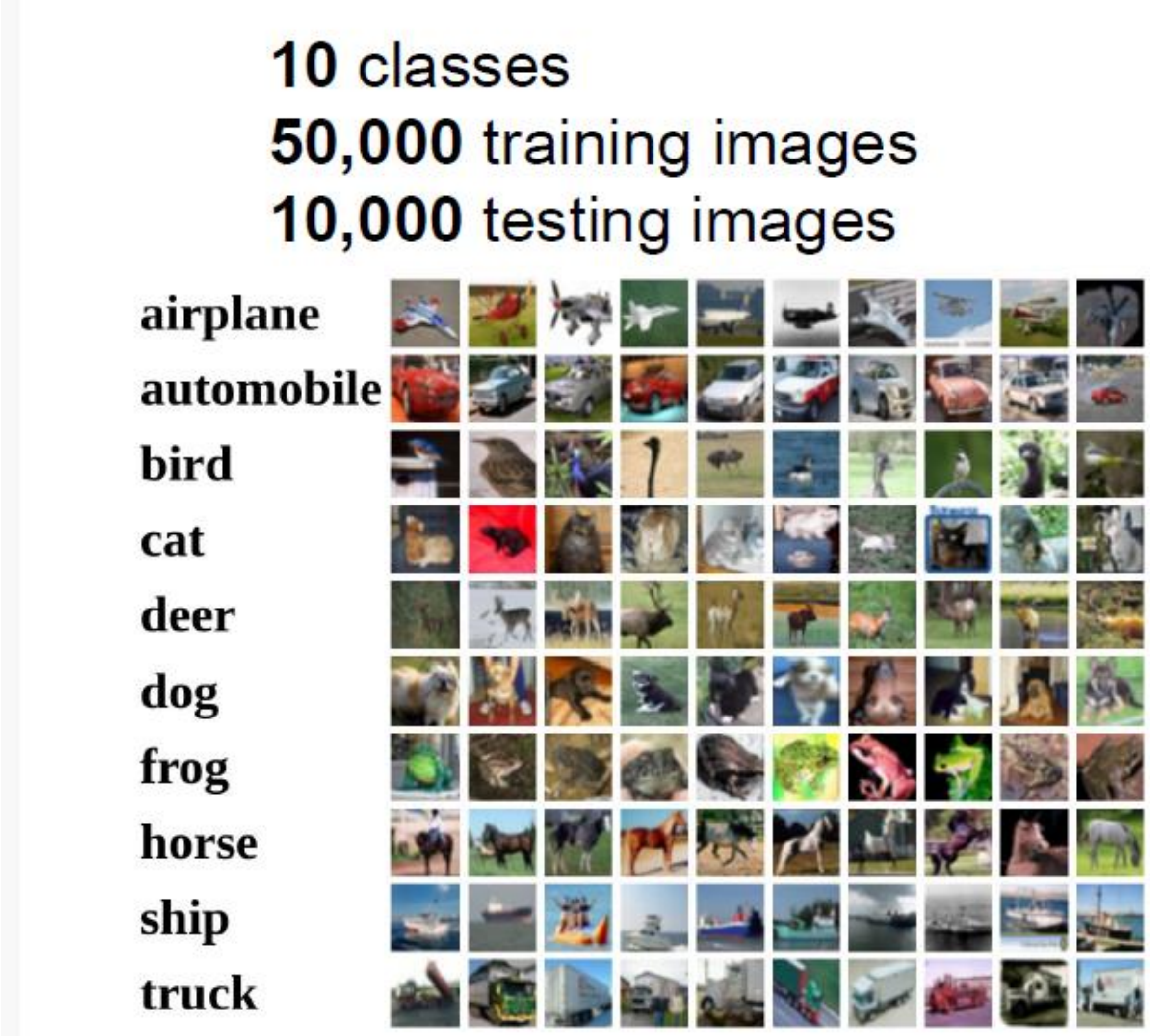
Setting Hyperparameters

- we cannot use the test set for the purpose of tweaking hyperparameters.

- Whenever you’re designing Machine Learning algorithms, you should think of the test set as a very precious resource that should ideally never be touched until one time at the very end.

- Otherwise, the very real danger is that you may tune your hyperparameters to work well on the test set, but if you were to deploy your model you could see a significantly reduced performance.

Example Dataset: CIFAR10

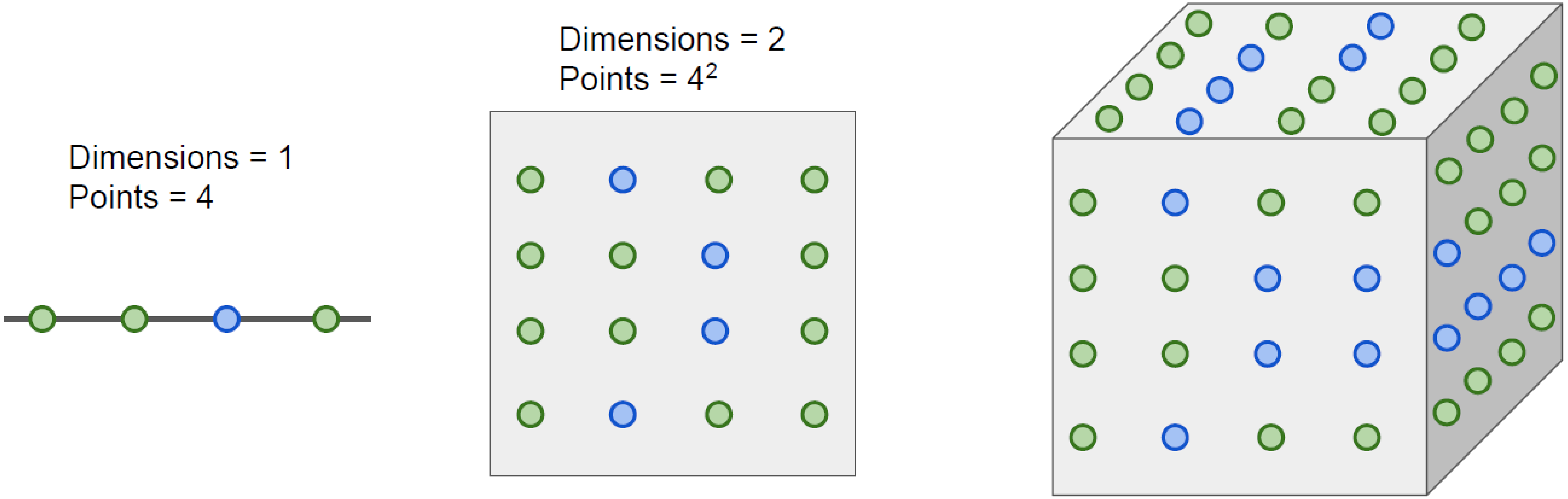


Example Dataset: CIFAR10

Example Dataset: **CIFAR10**

K-Nearest Neighbors

- k-Nearest Neighbor with pixel distance is actually **never used**.
  - Curse of dimensionality



K-Nearest Neighbors: Summary

- In image classification we start with a training set of images and labels, and must predict labels on the test set.
- The K-Nearest Neighbors classifier predicts labels based on the K nearest training examples.
- Distance metric and K are hyperparameters
- Choose hyperparameters using the validation set
- Only run on the test set once at the very end!