

Image classification

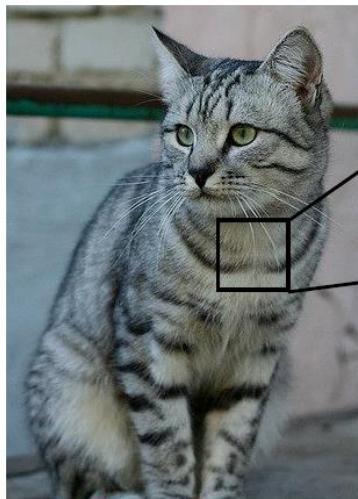
Image Classification

- 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

The Problem: Semantic Gap



```
[185 112 108 111 104 99 105 99 96 183 112 119 104 97 93 87]
[ 91 88 102 105 104 79 98 103 99 105 123 136 110 105 94 89]
[ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]
[ 09 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]
[128 137 144 148 109 95 86 79 62 65 63 63 60 73 85 101]
[125 133 148 137 119 121 117 94 65 79 88 65 54 64 72 98]
[127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
[ 89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]
[115 114 109 123 150 147 131 118 113 109 108 92 74 65 72 78]
[ 63 77 86 81 77 79 102 123 117 115 117 125 130 135 87]
[ 63 65 82 80 88 89 71 81 100 103 105 105 105 110 118]
[ 07 65 71 87 106 95 69 76 130 126 107 92 94 105 112]
[118 67 82 86 117 123 116 66 41 51 98 93 89 98 102 107]
[164 146 112 88 92 120 124 104 76 48 45 66 68 101 102 109]
[157 170 157 178 93 88 114 132 312 97 69 55 78 82 91 94]
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 66]
[128 112 96 117 159 144 120 115 104 107 102 93 87 81 72 79]
[123 187 96 86 83 112 153 149 122 109 104 75 88 107 112 99]
[122 121 102 88 82 86 94 117 145 148 153 102 58 78 92 92 107]
[122 104 148 103 71 56 78 83 93 103 119 139 102 61 69 84]
```

What the computer sees

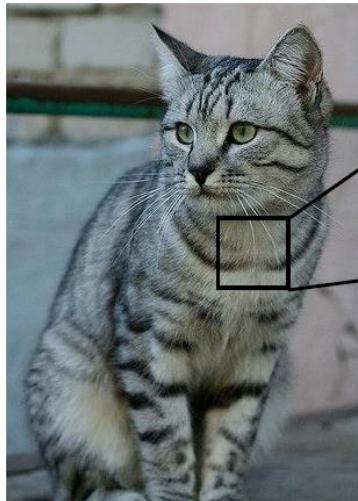
An image is a tensor of integers between [0, 255]:

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

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

The Problem: Semantic Gap



```
[185 112 108 111 104 99 105 99 96 183 112 119 104 97 93 87]
[ 91 88 102 105 104 79 98 103 99 105 123 136 110 105 94 89]
[ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]
[ 09 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]
[128 137 144 148 109 95 86 79 62 65 63 63 60 73 85 101]
[125 133 148 137 119 121 117 94 65 79 88 65 54 64 72 98]
[127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
[ 89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]
[115 114 109 123 150 147 131 118 113 109 108 92 74 65 72 78]
[ 63 77 86 81 77 79 102 123 117 115 117 125 130 135 87]
[ 63 65 82 80 88 89 71 81 100 103 105 105 107 114 123 111]
[ 63 65 82 80 88 89 71 81 100 103 105 105 107 114 123 111]
[ 07 65 71 87 106 95 69 66 76 130 126 107 92 84 105 112]
[118 67 82 86 117 123 116 66 41 51 95 93 89 98 102 107]
[164 146 112 89 92 120 124 104 76 46 45 56 66 101 102 109]
[157 170 157 178 93 88 114 132 312 97 69 55 78 82 91 94]
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 66]
[128 112 96 117 159 144 120 115 104 187 102 93 87 81 72 79]
[123 187 96 86 83 112 153 149 122 109 104 75 88 107 112 99]
[122 121 102 88 82 86 94 117 145 148 153 102 58 78 92 92 107]
[122 104 148 103 71 56 78 83 93 103 119 139 102 61 69 84]
```

What the computer sees

All pixels change when
the camera moves!

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



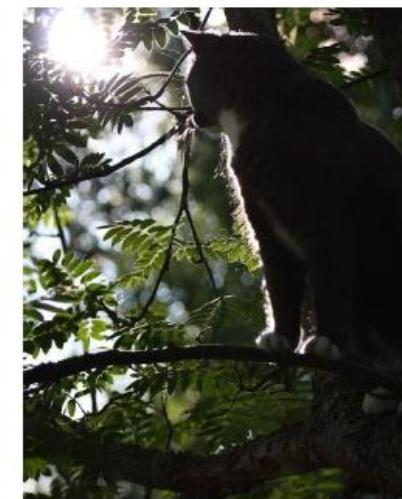
[This image](#) is CC0 1.0 public domain



[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



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

Challenges: Occlusion



[This image](#) is CC0 1.0 public domain



[This image](#) is CC0 1.0 public domain



[This image](#) by [jonsson](#) is licensed under CC-BY 2.0

Challenges: Deformation



[This image](#) by [Umberto Salvagnin](#)
is licensed under [CC-BY 2.0](#)



[This image](#) by [Umberto Salvagnin](#)
is licensed under [CC-BY 2.0](#)



[This image](#) by [sare bear](#) is
licensed under [CC-BY 2.0](#)



[This image](#) by [Tom Thai](#) is
licensed under [CC-BY 2.0](#)

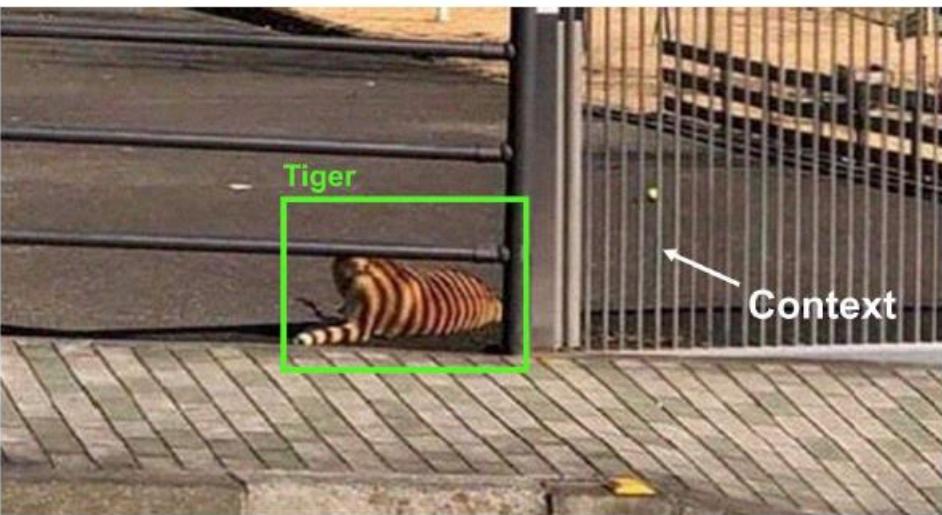
Challenges: Intraclass variation



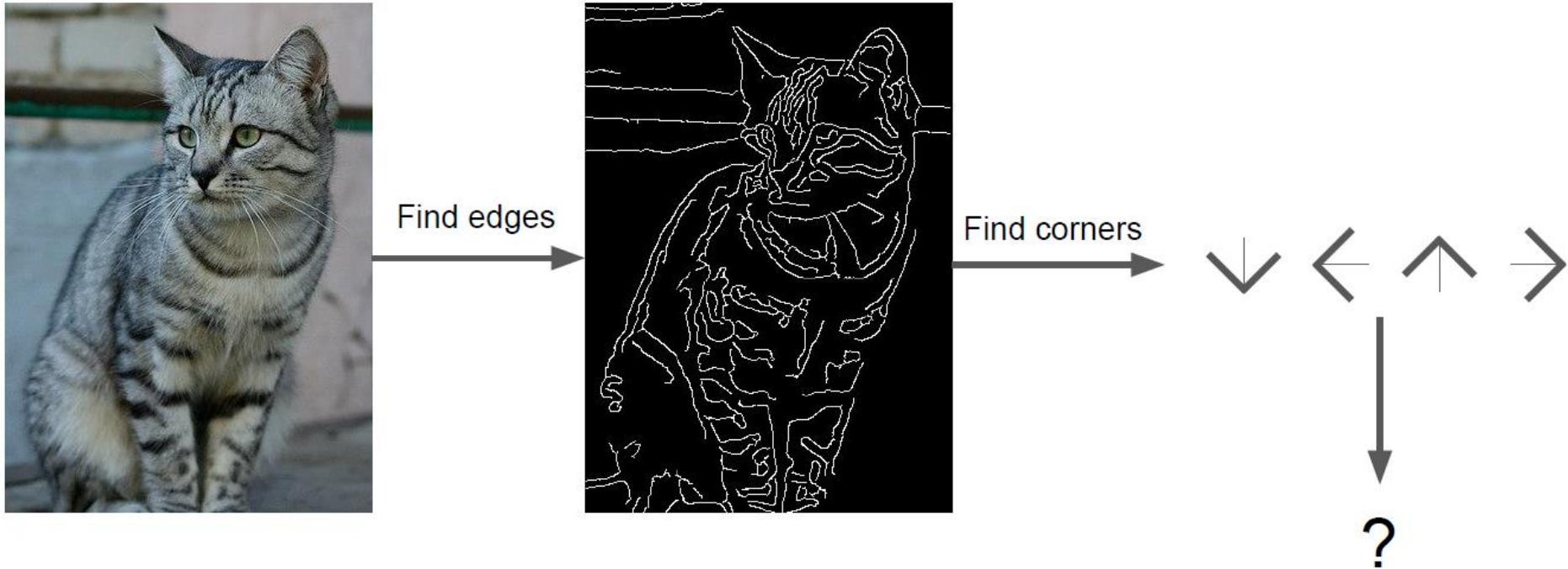
[This image](#) is CC0 1.0 public domain



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

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



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

deer



bird



plane



cat



car



Training data with labels

?



query data

Distance Metric



$\rightarrow \mathbb{R}$

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

=

add → 456

```
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

```
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

```
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
```

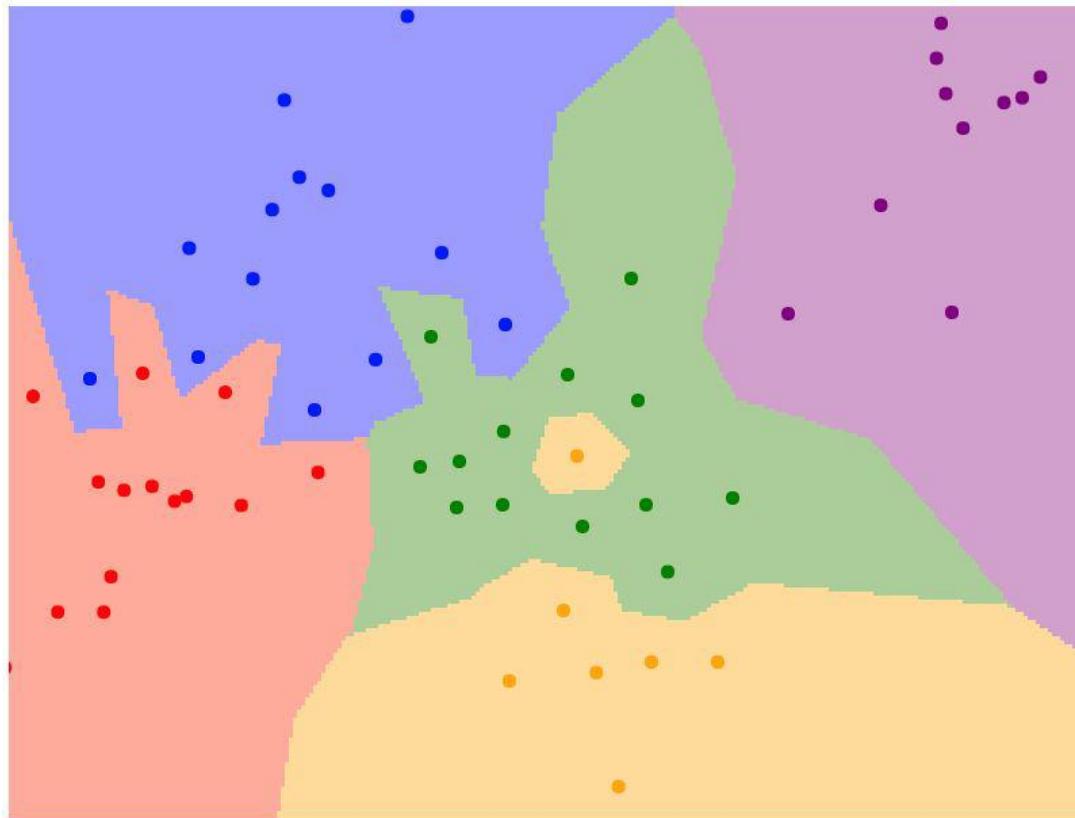
Nearest Neighbor classifier

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

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

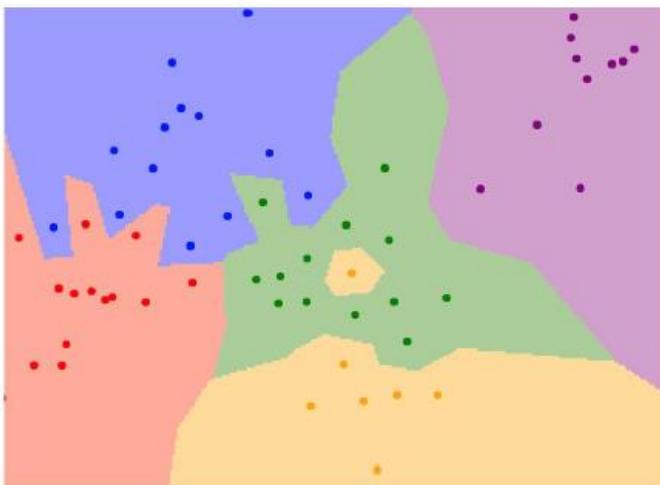
This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok

1-nearest neighbor

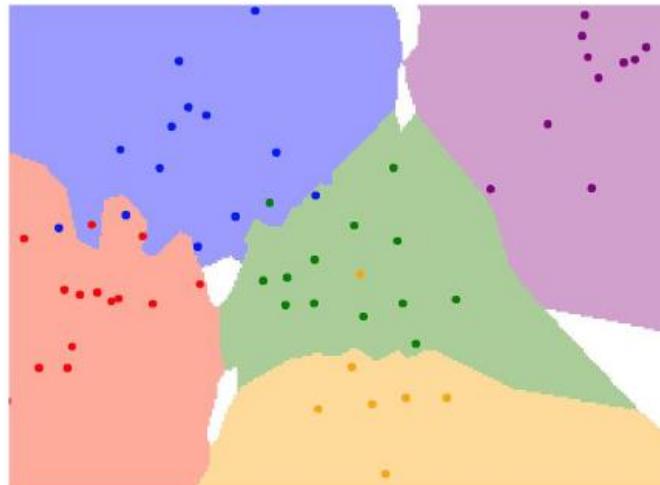


K-Nearest Neighbors

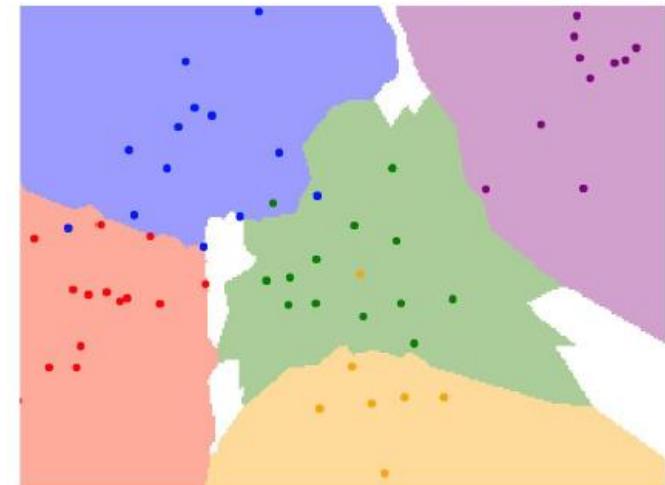
Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



$K = 1$



$K = 3$



$K = 5$

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:

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

Hyperparameters

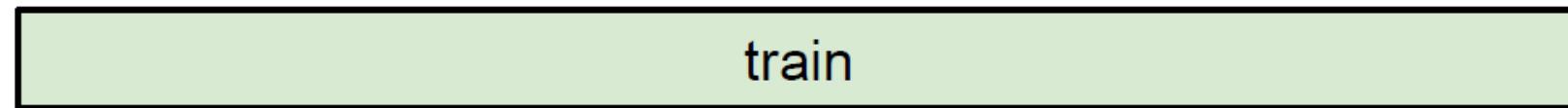
What is the best value of k to use?

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

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



BAD: No idea how algorithm will perform on new data

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

Better!



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

10 classes

50,000 training images

10,000 testing images

airplane



automobile



bird



cat



deer



dog



frog



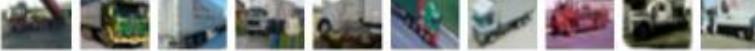
horse



ship



truck



Example Dataset: CIFAR10

10 classes

50,000 training images

10,000 testing images

airplane



automobile



bird



cat



deer



dog



frog



horse



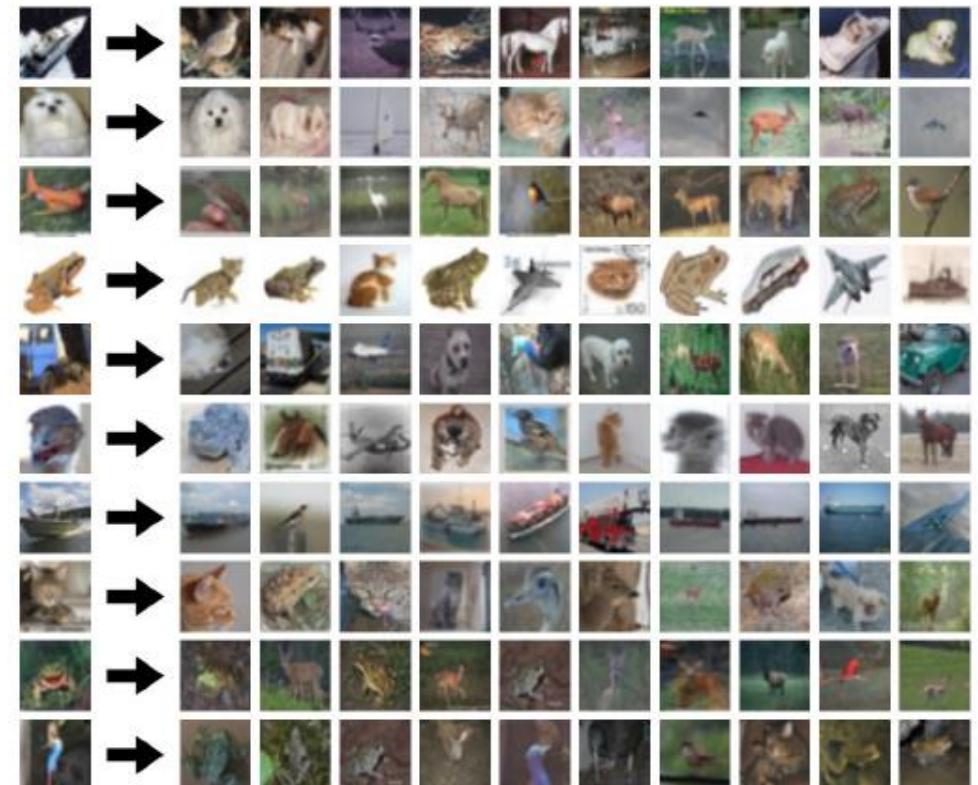
ship



truck



Test images and nearest neighbors

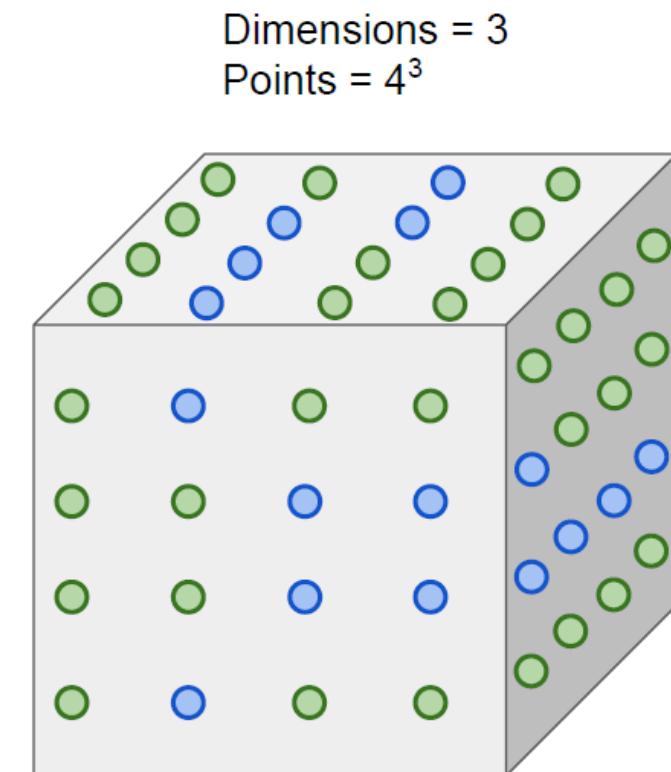
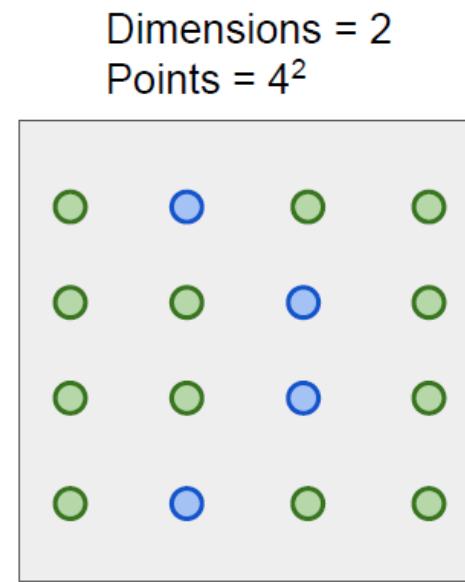
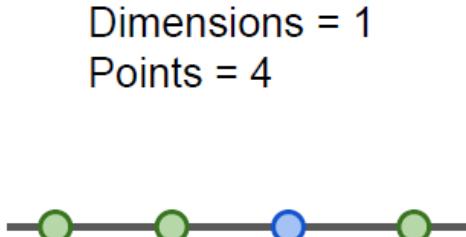


Example Dataset: CIFAR10



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!