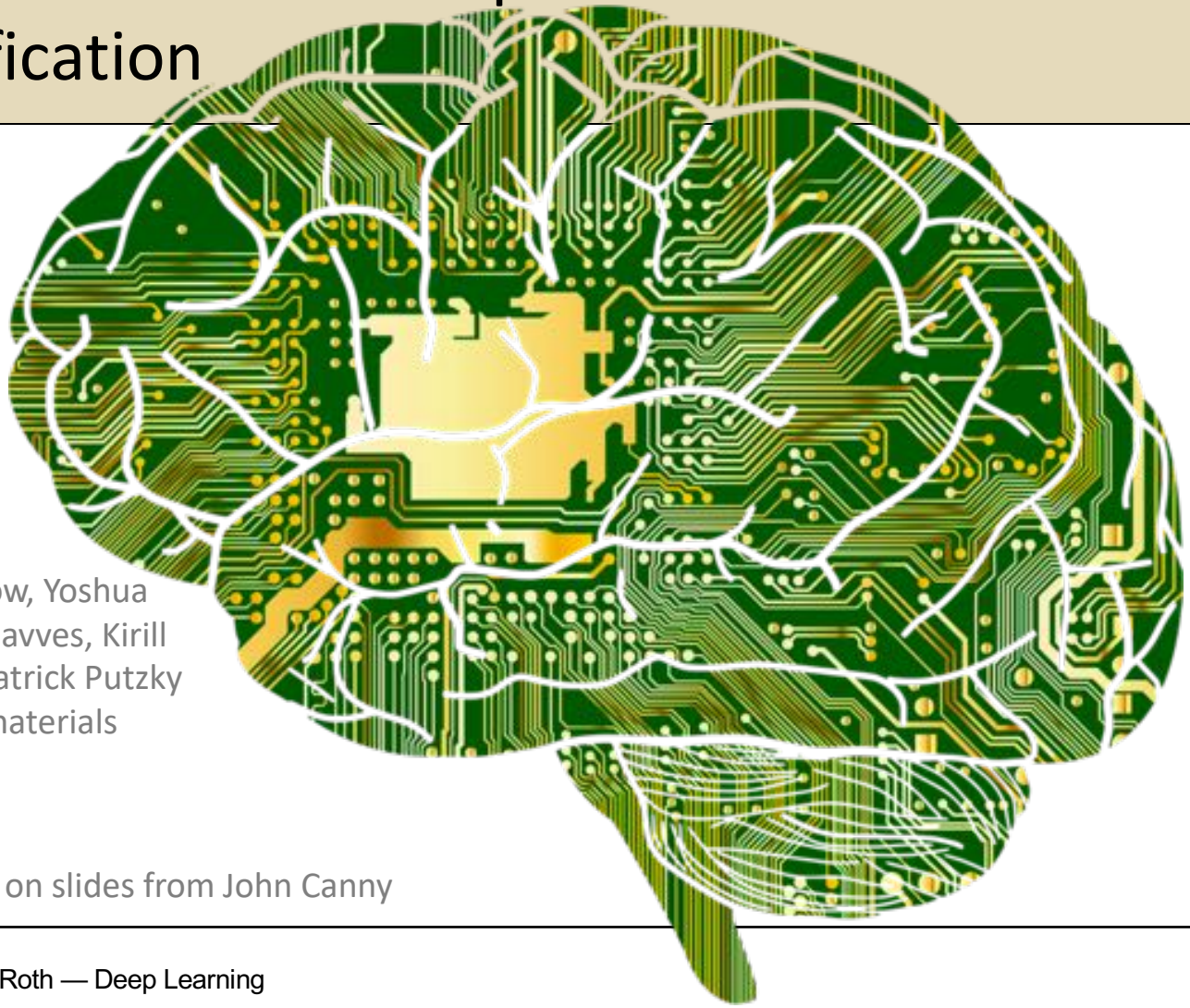
Deep Learning

TECHNISCHE UNIVERSITÄT DARMSTADT

Architectures and Methods: Computer Vision and Image Classification



Thanks to John Canny, Ian Goodfellow, Yoshua Bengio, Aaron Courville, Efstratios Gavves, Kirill Gavrilyuk, Berkay Kicanaoglu, and Patrick Putzky and many others for making their materials publically available.

The present slides are mainly based on slides from John Canny

Computer Vision – a brief history

TECHNISCHE UNIVERSITÄT DARMSTADT

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence Group — July 7, 1966
Vision Memo. No. 100.

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

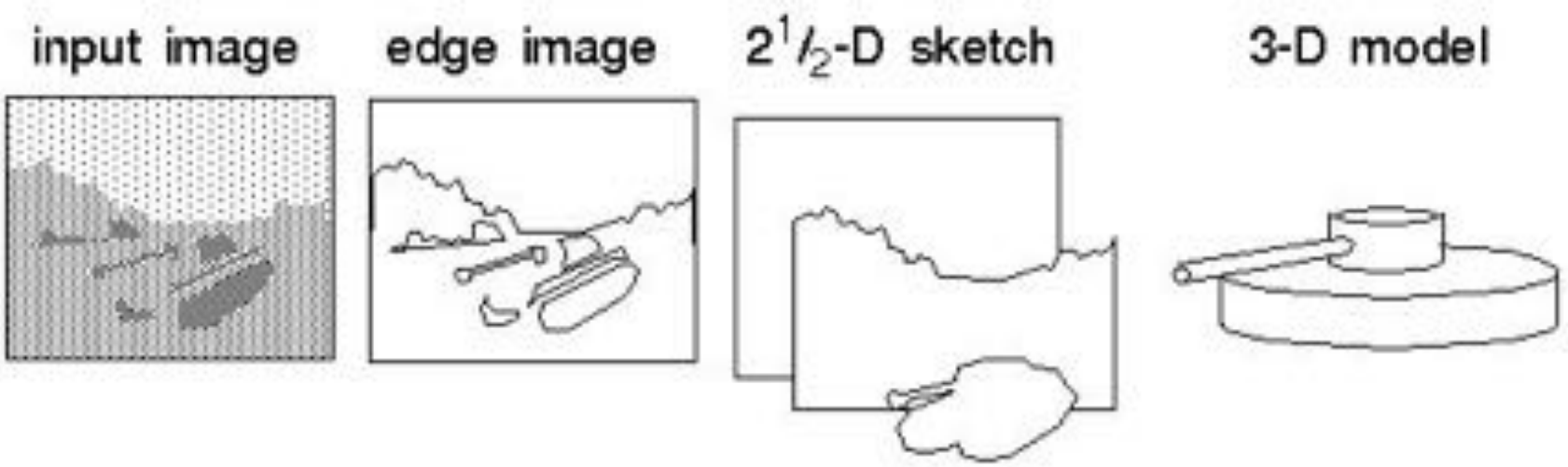
Computer Vision – a brief history

TECHNISCHE UNIVERSITÄT DARMSTADT

David Marr's approach 1970-80s

- Computation
- Algorithm
- Implementation

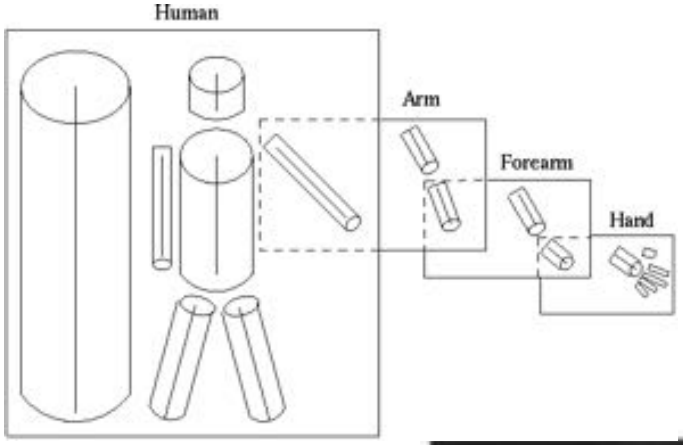
primal sketch and 2 ½ D sketch



Computer Vision - Shape description

TECHNISCHE UNIVERSITÄT DARMSTADT

Skeletons and Cylinders: Marr and Nishihara 1978:



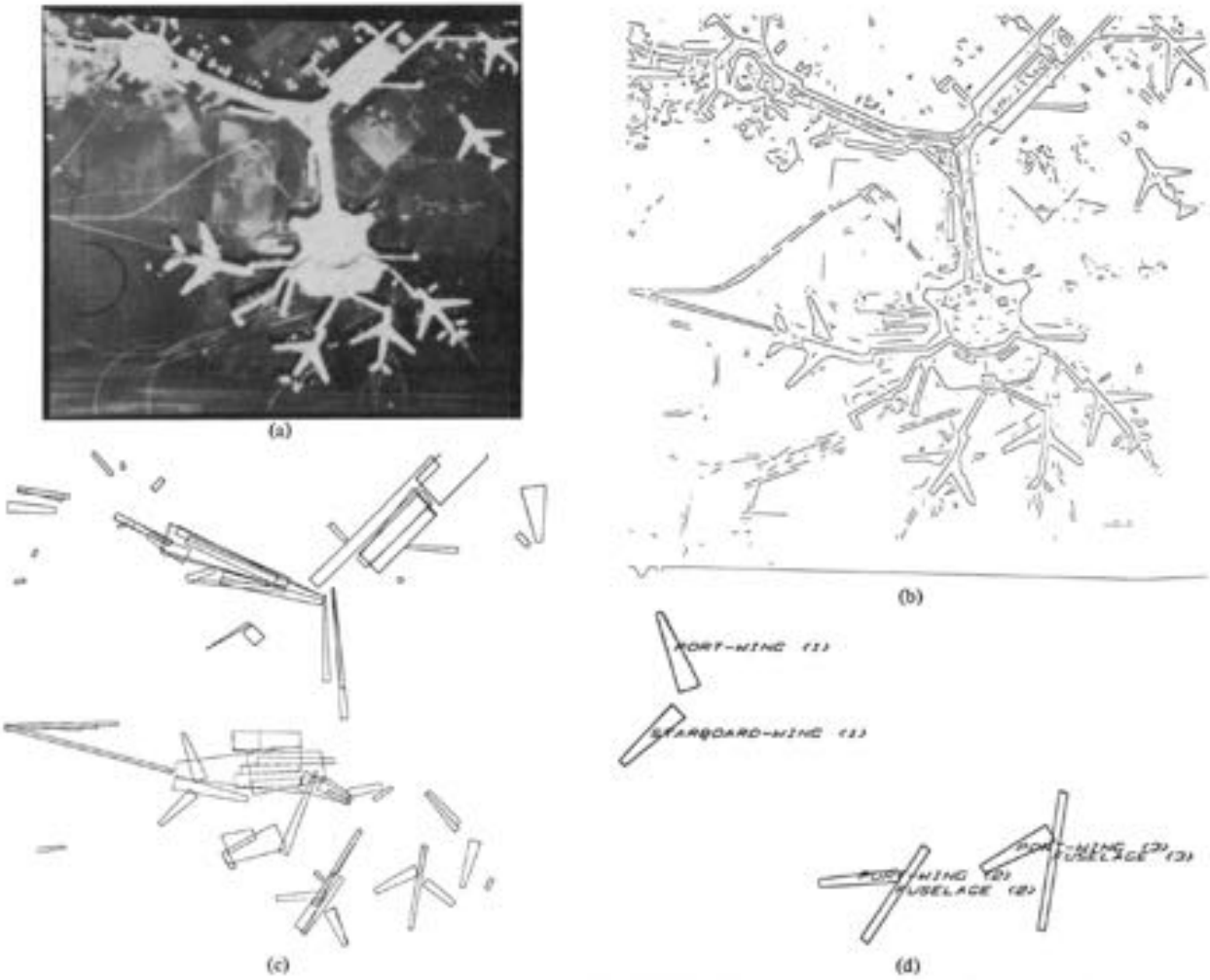
Tom Binford, generalized cylinders 1971

a) b) c)

Computer Vision - Shape matching

TECHNISCHE UNIVERSITÄT DARMSTADT

ACRONYM (Brooks 1982) matching with generalized cylinders + search!:



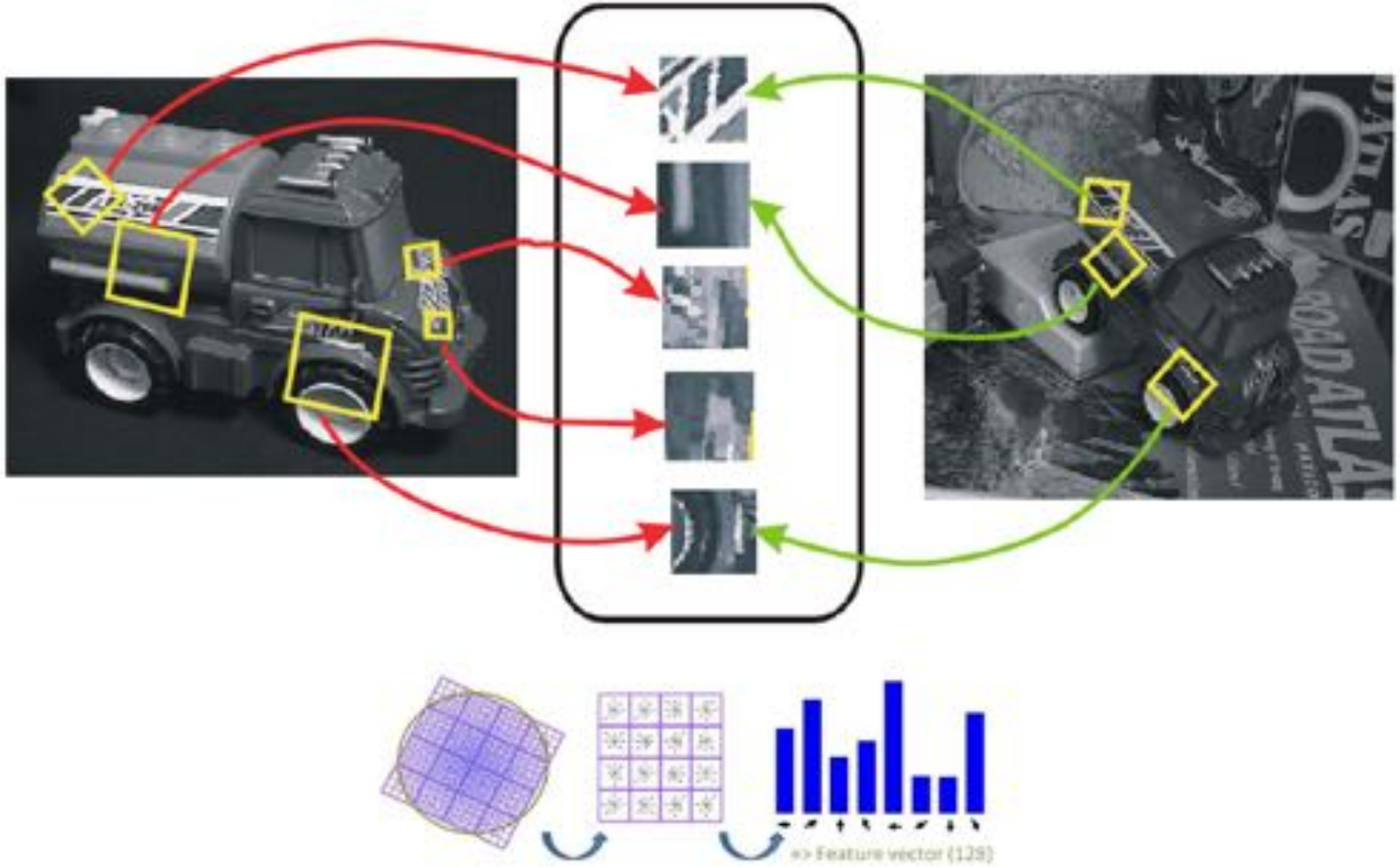
Computer Vision - Eigenfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

Turk and Pentland 1991:

A data-driven approach?

Computer Vision – Invariant Features

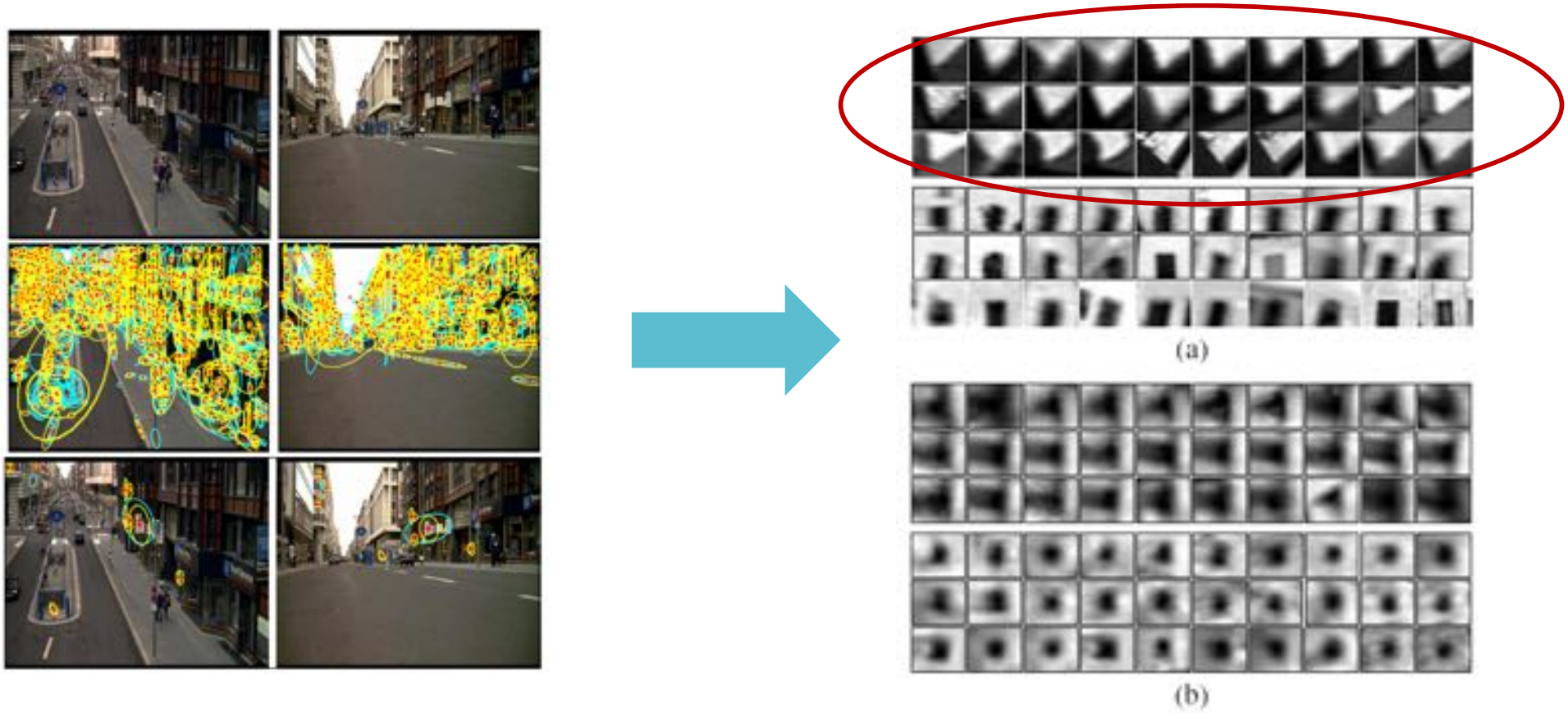


"SIFT" & Object Recognition, David Lowe, 1999

Computer Vision – Bag-of-words

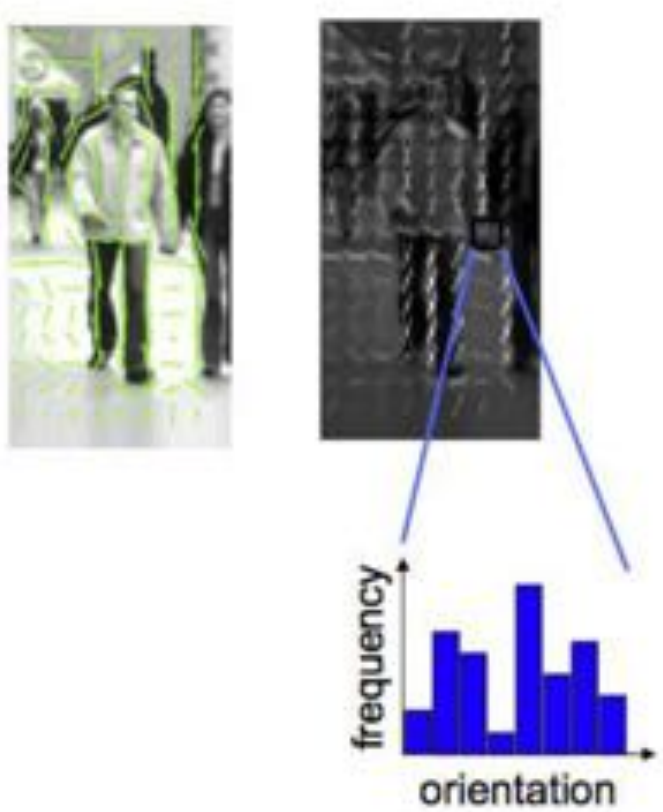
TECHNISCHE UNIVERSITÄT DARMSTADT

Sivic and Zisserman (2003) “Video Google”

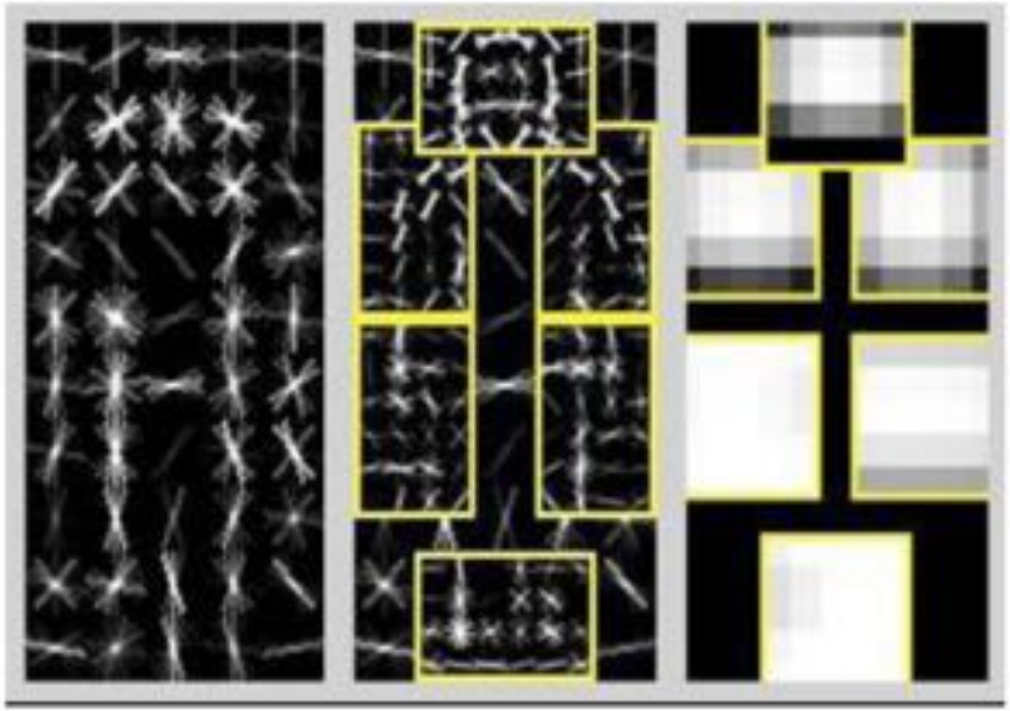


SIFT → SA and MS regions code words

Computer Vision - Shape matching



Histogram of Gradients (HoG)
Dalal & Triggs, 2005



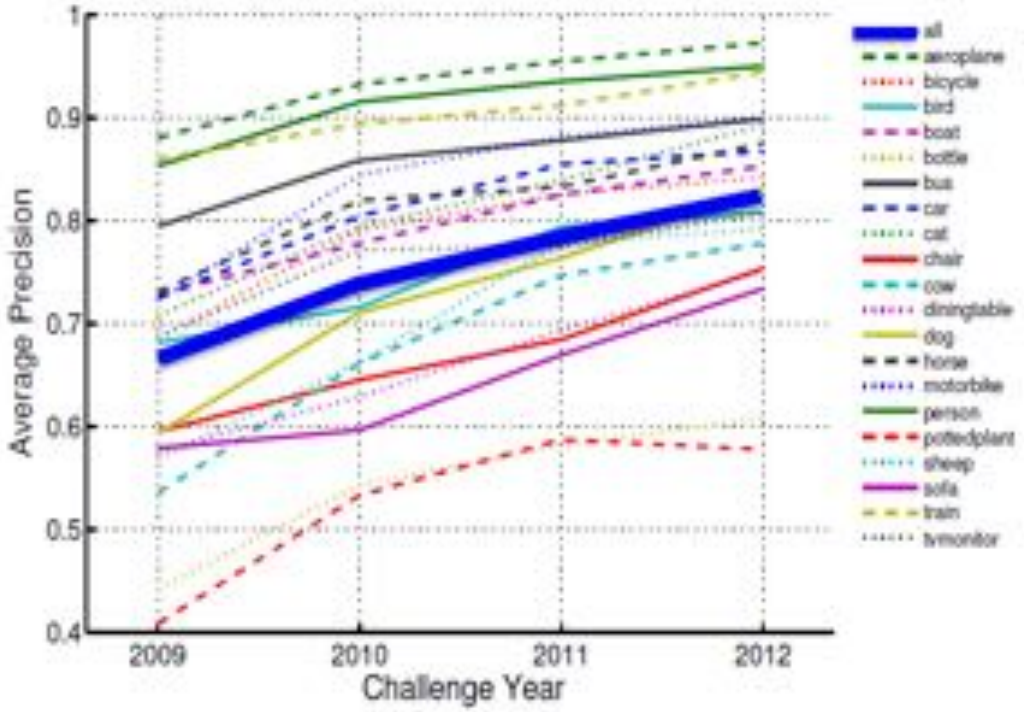
Deformable Part Model
Felzenswalb, McAllester, Ramanan, 2009

Computer Vision – Challenge Datasets

TECHNISCHE UNIVERSITÄT DARMSTADT

PASCAL Visual Object Challenge (20 object categories)

[Everingham et al. 2006-2012]



IMAGENET

www.image-net.org

22K categories and **14M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
  - Food
  - Materials
- Structures
  - Artifact
    - Tools
    - Appliances
    - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activities

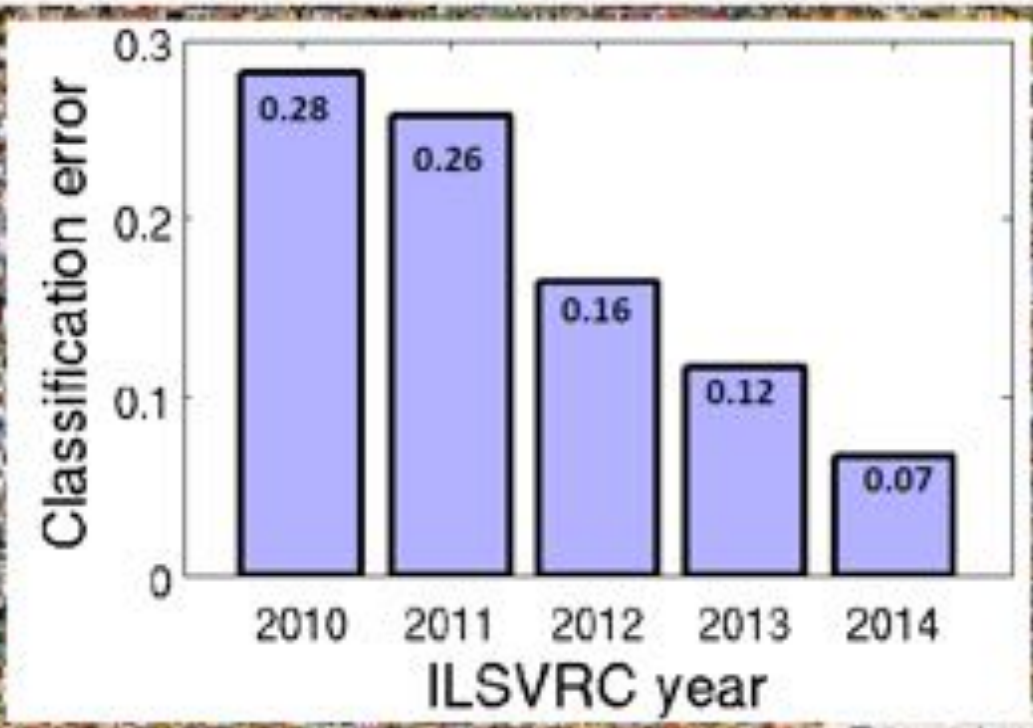
Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009

IMAGENET **Large Scale Visual Recognition Challenge**

The Image Classification Challenge:

1,000 object classes

1,431,167 images

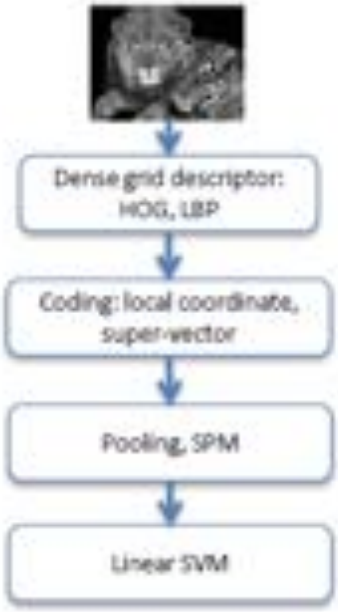


Russakovsky et al. arXiv, 2014

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

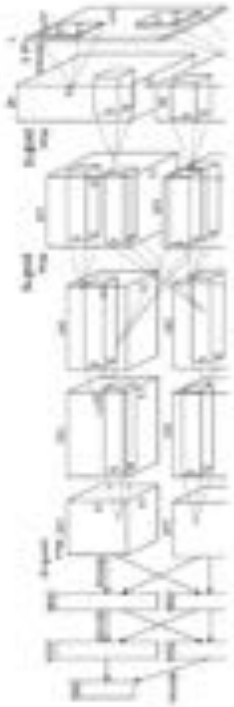
NEC-UIUC



[Lin CVPR 2011]

Year 2012

SuperVision



[Krizhevsky NIPS 2012]

Year 2014

GoogLeNet

[Szegedy arxiv 2014]

VGG

[Simonyan arxiv 2014]

Year 2015

MSRA

Image Classification: a core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ **cat**

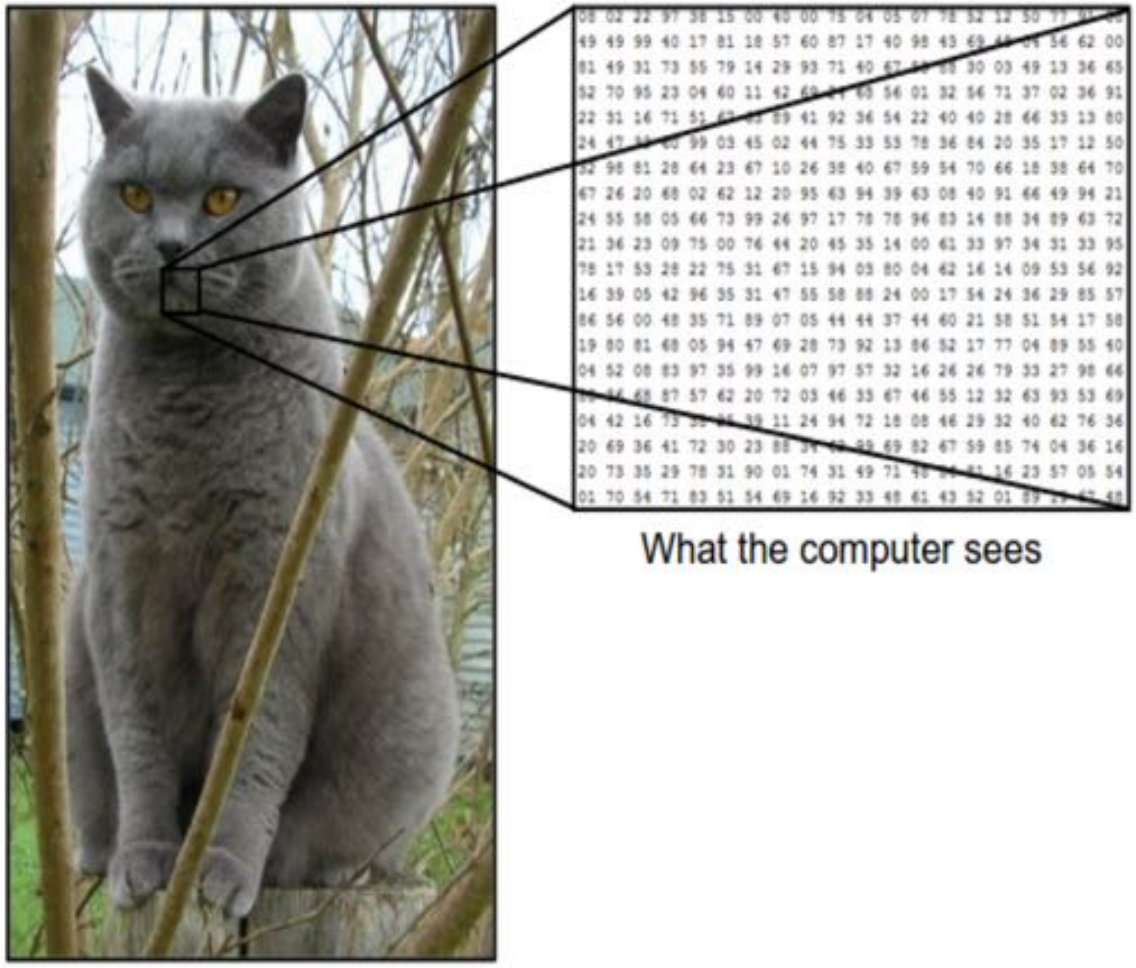
The problem: *semantic gap*

TECHNISCHE UNIVERSITÄT DARMSTADT

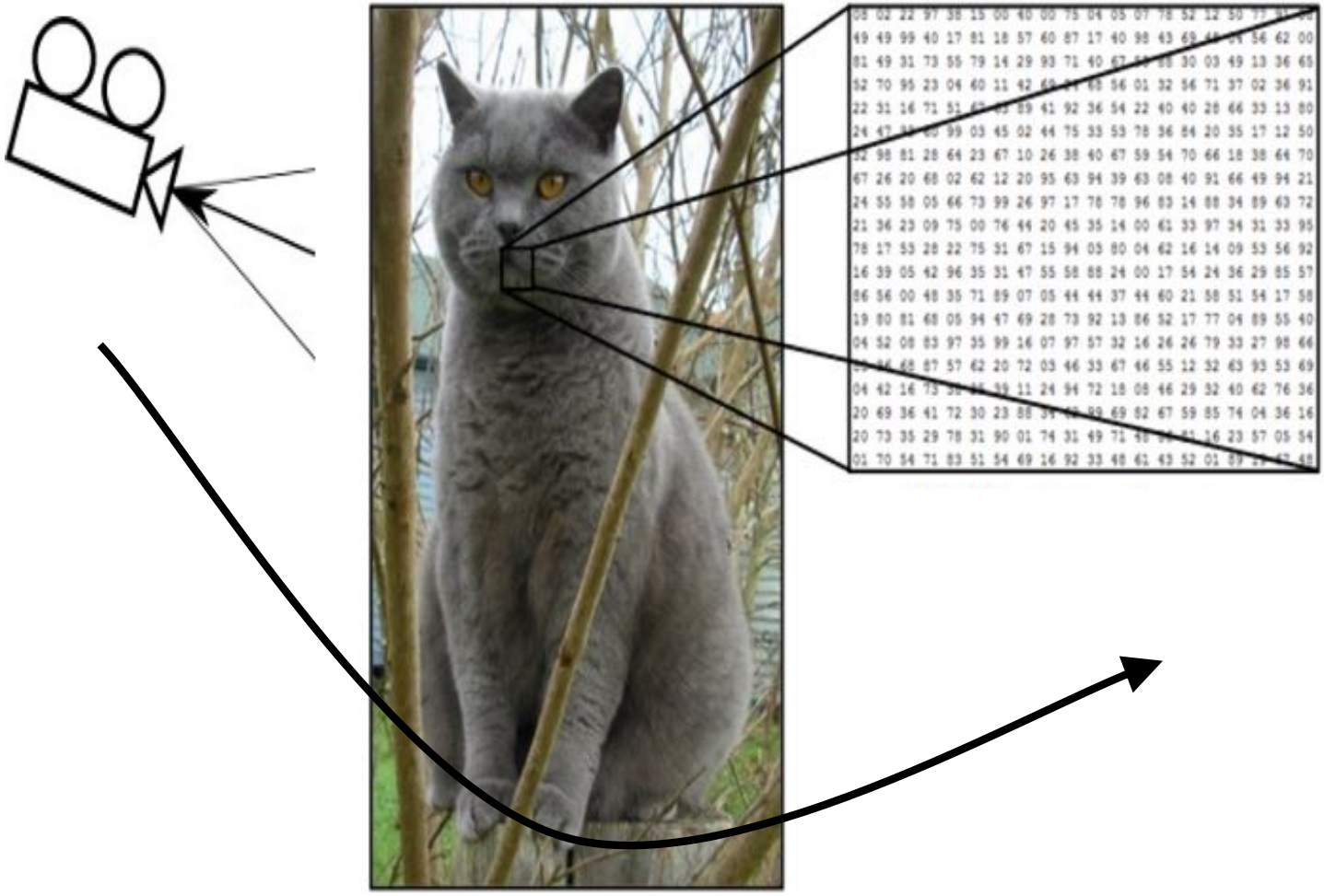
Images are represented as 3D arrays of numbers, with integers between [0, 255].

E.g.
300 x 100 x 3

(3 for 3 color channels RGB)



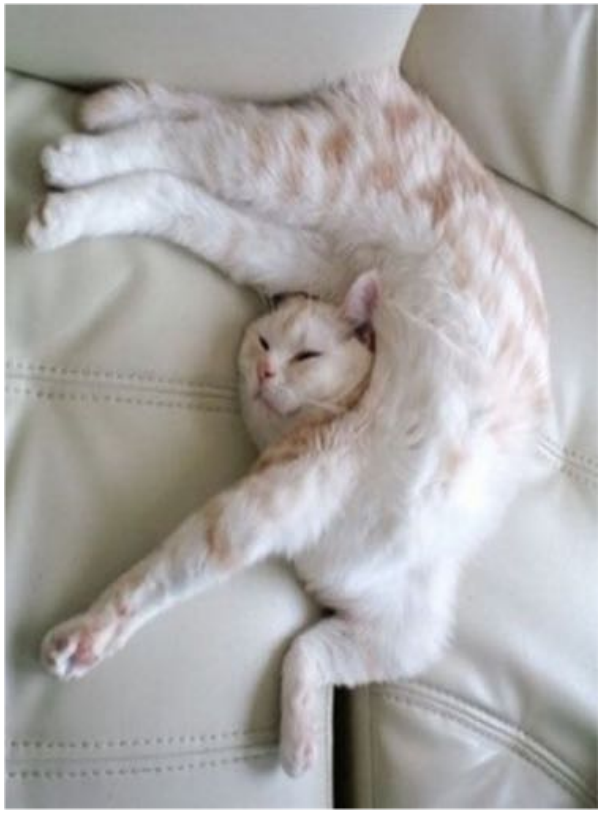
Challenges: Viewpoint Variation

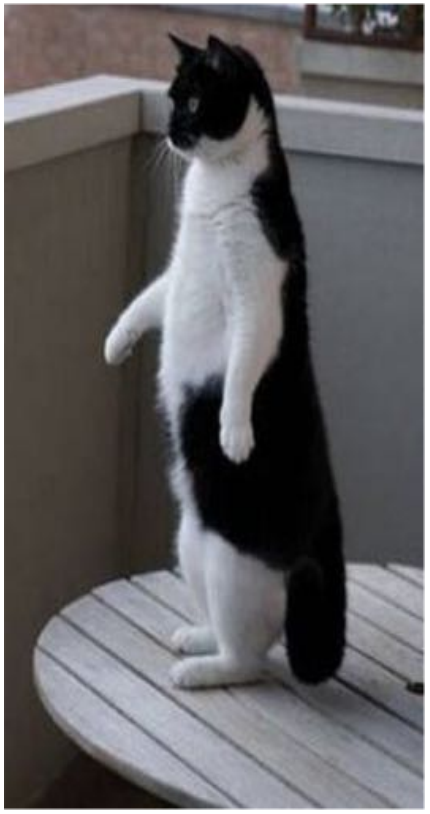


Challenges: Illumination

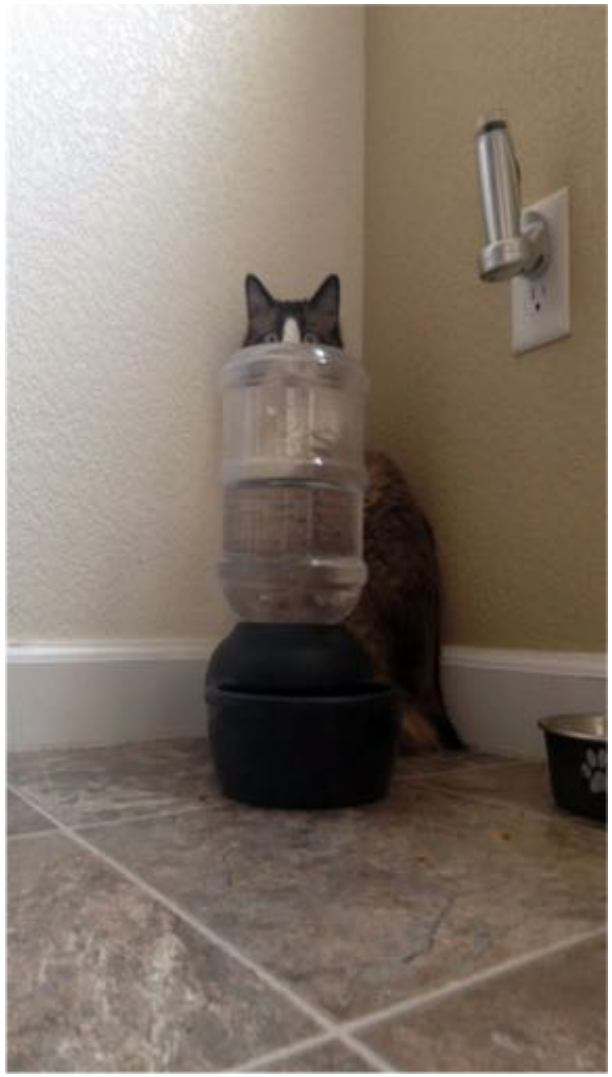
TECHNISCHE UNIVERSITÄT DARMSTADT

Challenges: Deformation



Challenges: Occlusion



Challenges: Background clutter

TECHNISCHE UNIVERSITÄT DARMSTADT

Challenges: Intraclass variation

TECHNISCHE UNIVERSITÄT DARMSTADT

An image classifier

TECHNISCHE UNIVERSITÄT DARMSTADT

```
def predict(image):
    # ????
    return class_label
```

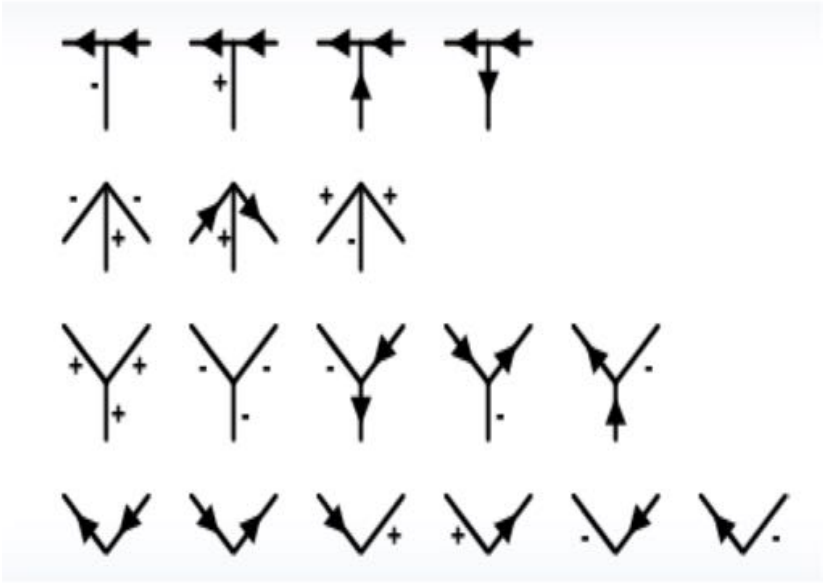
Unlike e.g. sorting a list of numbers,

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

Classifying using constraints? ?

TECHNISCHE UNIVERSITÄT DARMSTADT

???



Data-driven approach:

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Example training set

First classifier: Nearest Neighbor Classifier

TECHNISCHE UNIVERSITÄT DARMSTADT

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Remember all training images and their labels

Predict the label of the most similar training image

Example dataset: CIFAR-10

TECHNISCHE UNIVERSITÄT DARMSTADT

10 labels
50,000 training images, each image is tiny: 32x32
10,000 test images.

Example dataset: CIFAR-10

TECHNISCHE UNIVERSITÄT DARMSTADT

10 labels
50,000 training images
10,000 test images.

For every test image (first column), examples of nearest neighbors in rows

How do we compare the images? What is the distance metric?

TECHNISCHE UNIVERSITÄT DARMSTADT

L1 distance:

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

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

Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```


Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```

remember the 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 every test image:

- find nearest train image with L1 distance
- predict the label of nearest training 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: how does the classification speed depend on the size of the 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

Q: how does the classification speed depend on the size of the training data? **linearly :(**

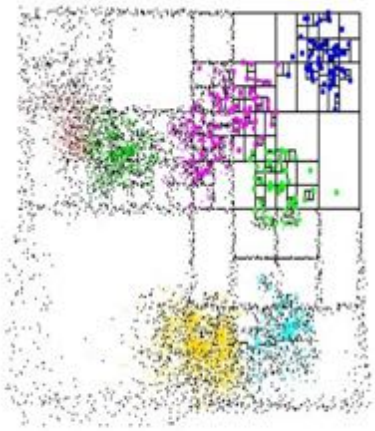
This is **backwards**:
- test time performance is usually much more important in practice.
- CNNs flip this: expensive training, cheap test evaluation

Aside: Approximate Nearest Neighbor find approximate nearest neighbors quickly

(and you can even learn binary hash codes

ANN: A Library for Approximate Nearest Neighbor Searching

David M. Mount and Sunil Arya
Version 1.1.2
Release Date: Jan 27, 2010



What is ANN?

ANN is a library written in C++, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in arbitrarily high dimensions.

In the nearest neighbor problem a set of data points in d-dimensional space is given. These points are preprocessed into a data structure, so that given any query point q, the nearest or generally k nearest points of P to q can be reported efficiently. The distance between two points can be defined in many ways. ANN assumes that distances are measured using any class of distance functions called Minkowski metrics. These include the well known Euclidean distance, Manhattan distance, and max distance.

Based on our own experience, ANN performs quite efficiently for point sets ranging in size from thousands to hundreds of thousands, and in dimensions as high as 20. (For applications in significantly higher dimensions, the results are rather spotty, but you might try it anyway.)

The library implements a number of different data structures, based on kd-trees and box-decomposition trees, and employs a couple of different search strategies.

The library also comes with test programs for measuring the quality of performance of ANN on any particular data sets, as well as programs for visualizing the structure of the geometric data structures.

FLANN - Fast Library for Approximate Nearest Neighbors

- Home
- News
- Publications
- Download
- Changelog
- Repository

What is FLANN?

FLANN is a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms we found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.

FLANN is written in C++ and contains bindings for the following languages: C, MATLAB and Python.

News

- (14 December 2012) Version 1.8.0 is out bringing incremental addition/removal of points to/from indexes
- (20 December 2011) Version 1.7.0 is out bringing two new index types and several other improvements.
- You can find binary installers for FLANN on the Point Cloud Library project page. Thanks to the PCL developers!
- Mac OS X users can install flann though MacPorts (thanks to Mark Moll for maintaining the Portfile)
- New release introducing an easier way to use custom distances, kd-tree implementation optimized for low dimensionality search and experimental MPI support
- New release introducing new C++ templated API, thread-safe search, save/load of indexes and more.
- The FLANN license was changed from LGPL to BSD.

How fast is it?

In our experiments we have found FLANN to be about one order of magnitude faster on many datasets (in query time), than previously available approximate nearest neighbor search software.

Publications

More information and experimental results can be found in the following papers:

- Marius Muja and David G. Lowe: **"Scalable Nearest Neighbor Algorithms for High Dimensional Data"**. Pattern Analysis and Machine Intelligence (PAMI), Vol. 36, 2014. [PDF] [BibTeX]
- Marius Muja and David G. Lowe: **"Fast Matching of Binary Features"**. Conference on Computer and Robot Vision (CRV) 2012. [PDF] [BibTeX]
- Marius Muja and David G. Lowe, **"Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration"**, in International Conference on Computer Vision Theory and Applications (VISAPP'09), 2009 [PDF] [BibTeX]

The choice of distance is a hyperparameter, common choices:

TECHNISCHE UNIVERSITÄT DARMSTADT

L1 (Manhattan) distance

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

L2 (Euclidean) distance

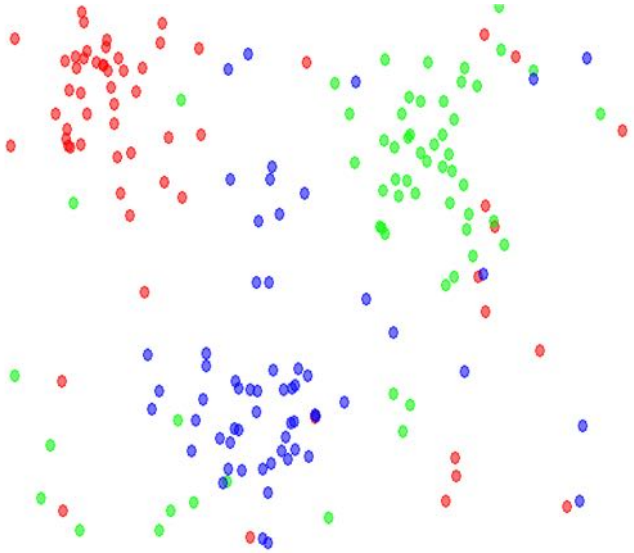
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

k-Nearest Neighbor

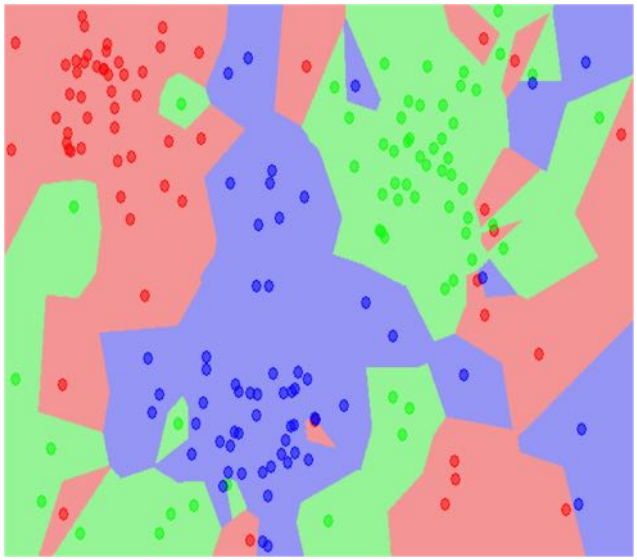
find the k nearest images, have them vote on the label

TECHNISCHE UNIVERSITÄT DARMSTADT

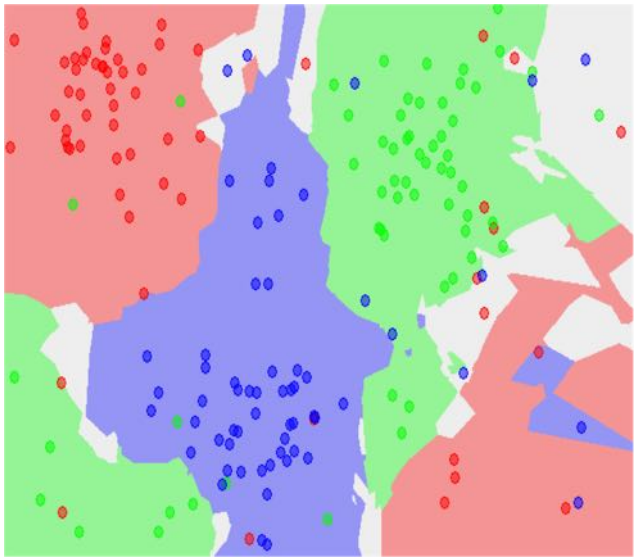
the data



NN classifier



5-NN classifier



http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

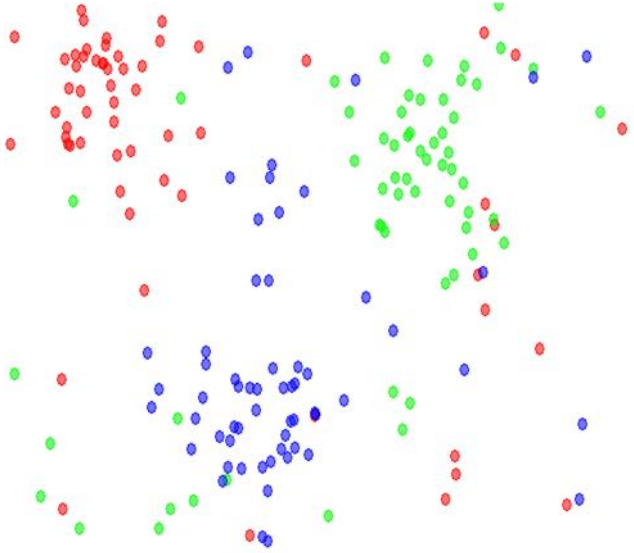
Example dataset: CIFAR-10

TECHNISCHE UNIVERSITÄT DARMSTADT

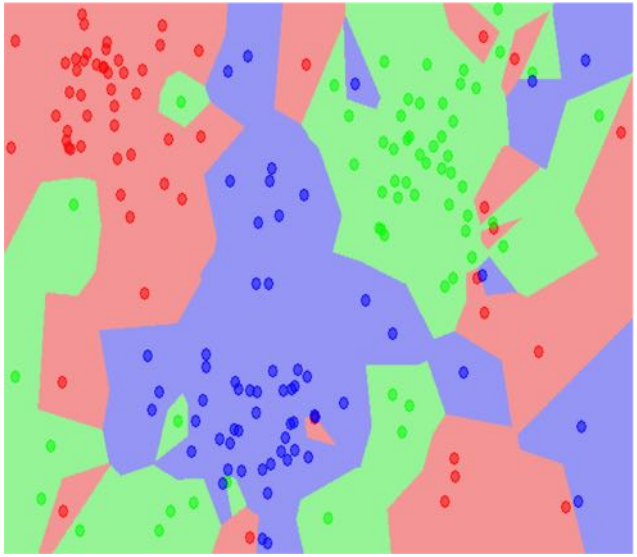
10 labels
50,000 training images
10,000 test images.

For every test image (first column), examples of nearest neighbors in rows

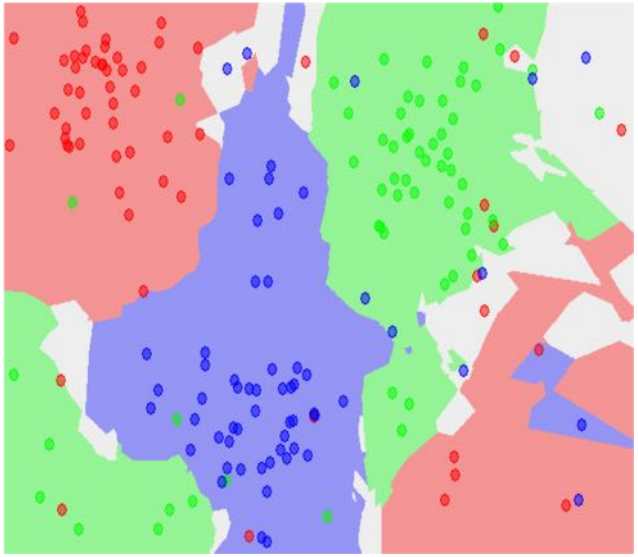
the data



NN classifier

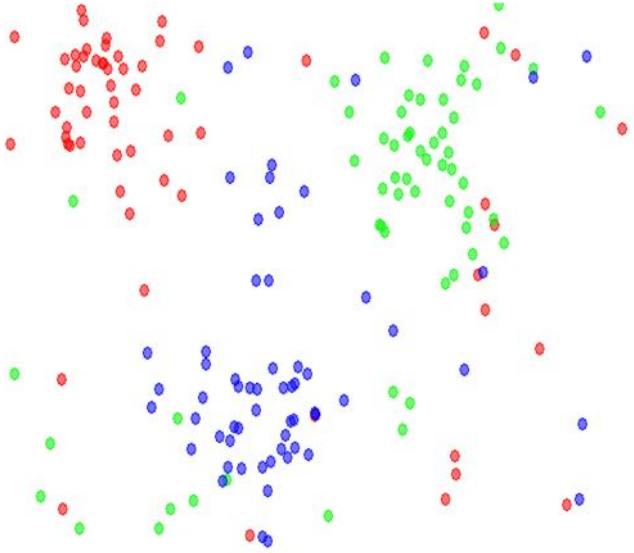


5-NN classifier

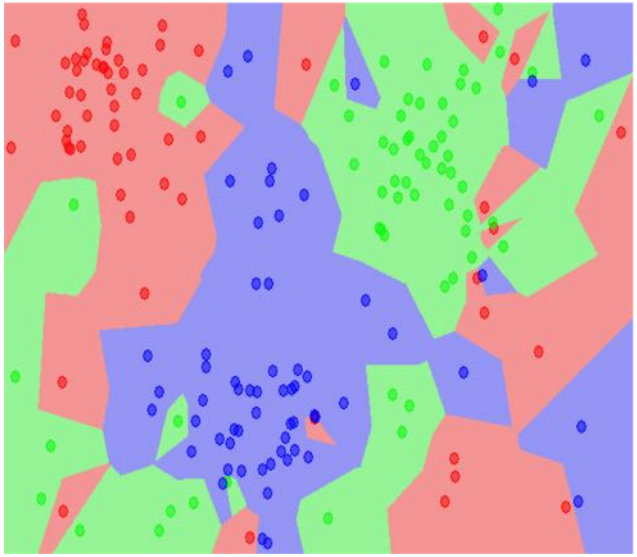


Q: what is the accuracy of the nearest neighbor classifier on the training data, when using the Euclidean distance?

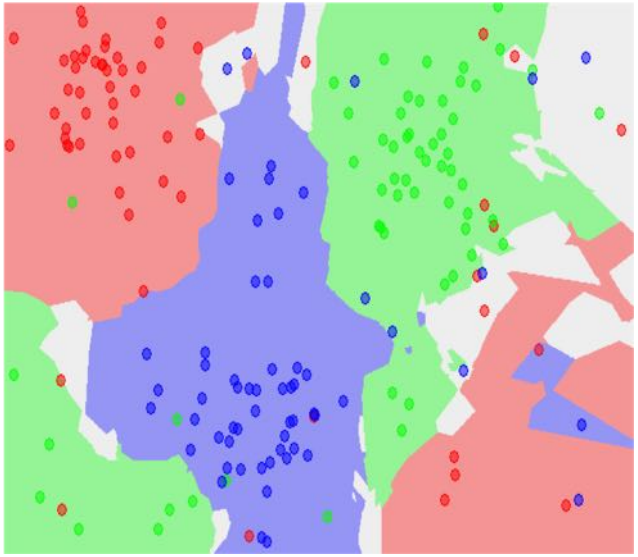
the data



NN classifier



5-NN classifier



Q2: what is the accuracy of the **k**-nearest neighbor classifier on the training data?

Hyperparameter tuning:

TECHNISCHE UNIVERSITÄT DARMSTADT

What is the best **distance** to use?
What is the best value of **k** to use?

i.e. how do we set the **hyperparameters**?

Bias and Variance

TECHNISCHE UNIVERSITÄT DARMSTADT

Recall from statistics (or ML), given a true function y=f(x) (mapping an input x to a label y), a machine learning algorithm is a statistical estimator $g_D(x)$. Here D is a data sample.

Estimators have:

- **Bias:** $g_D(x)$ is systematically different from f(x), i.e. $E_D(g_D(x)) \neq f(x)$
- **Variance:** $VAR_D(g_D(x))$. Even if no bias, $g_D(x)$ may be far from f(x) at most points x.

Bias and Variance

TECHNISCHE UNIVERSITÄT DARMSTADT

Increasing k in the kNN algorithm should have what effect on:

Bias: ?

Variance: ?

Bias and Variance

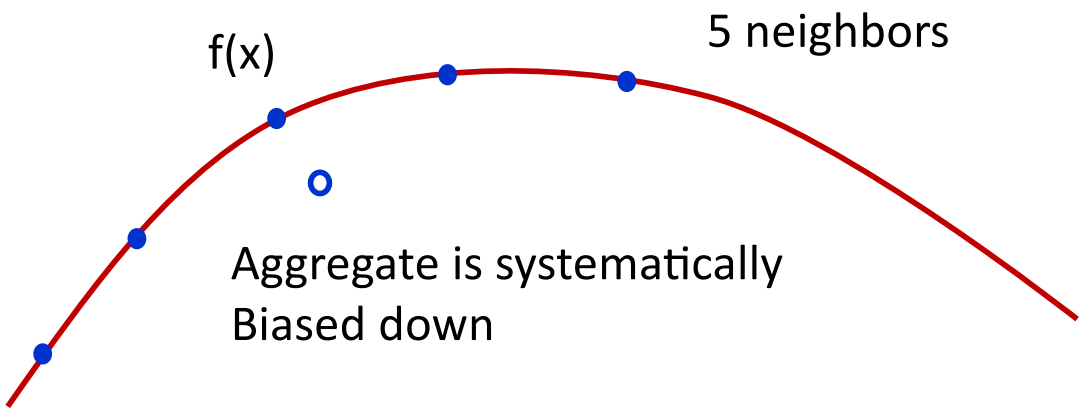
TECHNISCHE UNIVERSITÄT DARMSTADT

Increasing k in the kNN algorithm should have what effect on:

Bias: Should **increase**. The large k is, the further (on average) are the points being aggregated from x. i.e. the value depends on f(x') for x' further from x.

Variance: ?

Bias



Bias and Variance

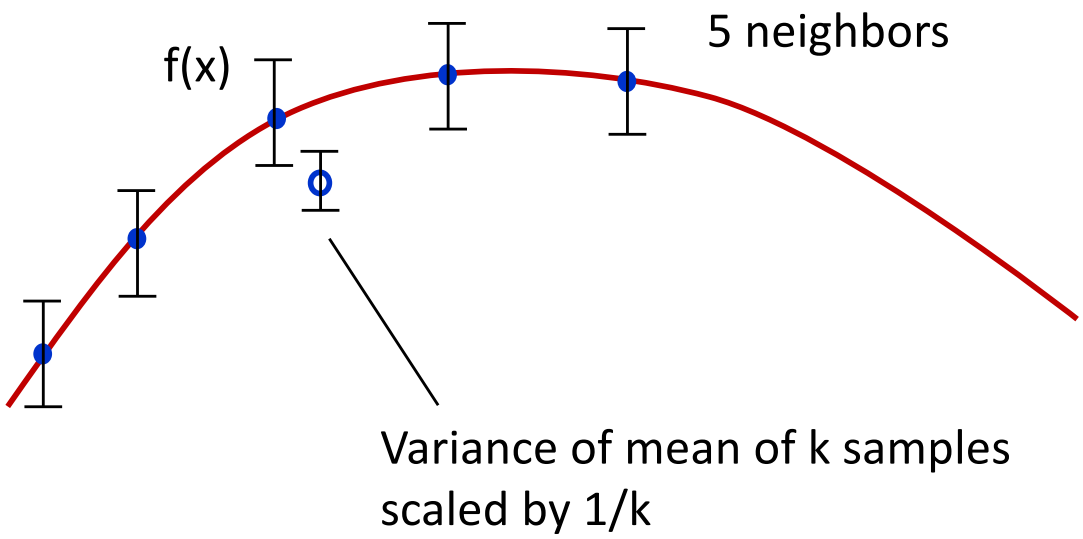
TECHNISCHE UNIVERSITÄT DARMSTADT

Increasing k in the kNN algorithm should have what effect on:

Bias: Should increase. The large k is, the further (on average) are the points being aggregated from x. i.e. the value depends on f(x') for x' further from x.

Variance: Should **decrease**. The average or majority vote of a set of equal-variance values has lower variance than each value.

Variance



Bias and Variance Rule of Thumb

TECHNISCHE UNIVERSITÄT DARMSTADT

Compared to simpler (fewer parameters), complex models have what kind of Bias and Variance?:

Bias: ?

Variance:?

Bias and Variance Rule of Thumb

TECHNISCHE UNIVERSITÄT DARMSTADT

Compared to simpler (fewer parameters), complex models have what kind of Bias and Variance?:

Bias: ? Lower, because complex models can better model local variation in f(x).

Variance:?

Bias and Variance Rule of Thumb

TECHNISCHE UNIVERSITÄT DARMSTADT

Compared to simpler (fewer parameters), complex models have what kind of Bias and Variance?:

Bias: ? Lower, because complex models can better model local variation in f(x).

Variance:? **Higher**, because the parameters are generally more sensitive to few data values.

Bias and Variance Rule of Thumb

TECHNISCHE UNIVERSITÄT DARMSTADT

Compared to simpler (fewer parameters), complex models have what kind of Bias and Variance?:

Bias: ? Lower, because complex models can better model local variation in f(x).

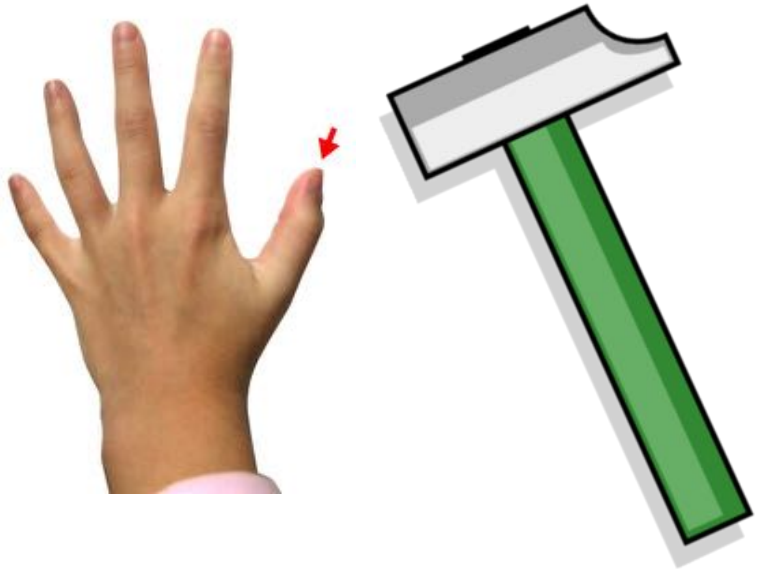
Variance:? Higher, because the parameters are generally more sensitive to few data values.

Complex models are prone to overfitting. They require/benefit from large training datasets in order to reduce variance. This includes most **Deep Networks**.

Bias and Variance Rule of Thumb

TECHNISCHE UNIVERSITÄT DARMSTADT

This rule of thumb is important to keep in mind, but it is only a **rule of thumb**, not a theorem. It applies to typical ML algorithms and deep networks, but not all.



A mathematical theorem is like a hammer. A rule of thumb is no match for it…

Hyperparameter tuning:

TECHNISCHE UNIVERSITÄT DARMSTADT

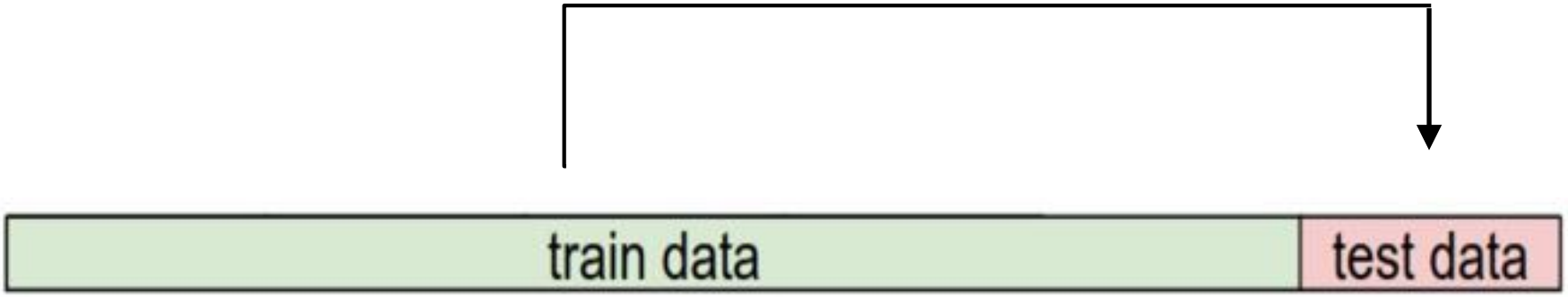
What is the best **distance** to use?
What is the best value of **k** to use?

i.e. how do we set the **hyperparameters**?

Very problem-dependent.
Must try them all out and see what works best.

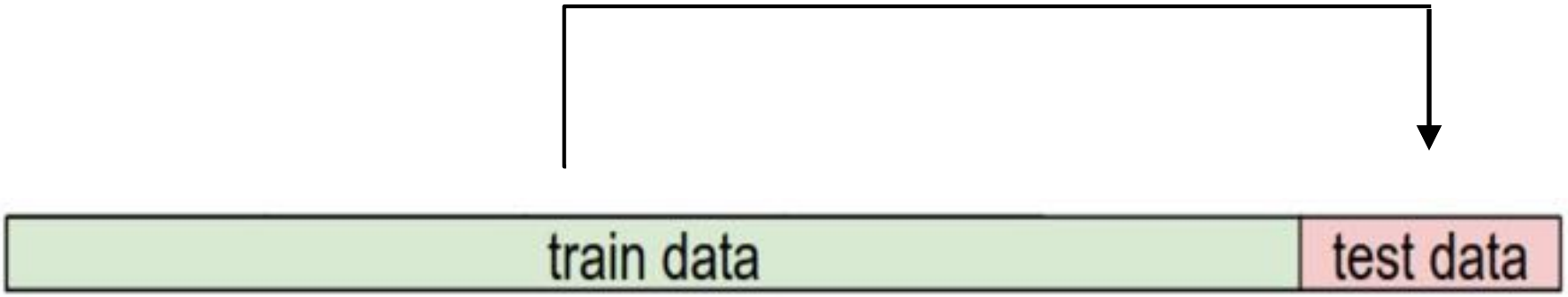
Try out what hyperparameters work best on test set.

TECHNISCHE UNIVERSITÄT DARMSTADT



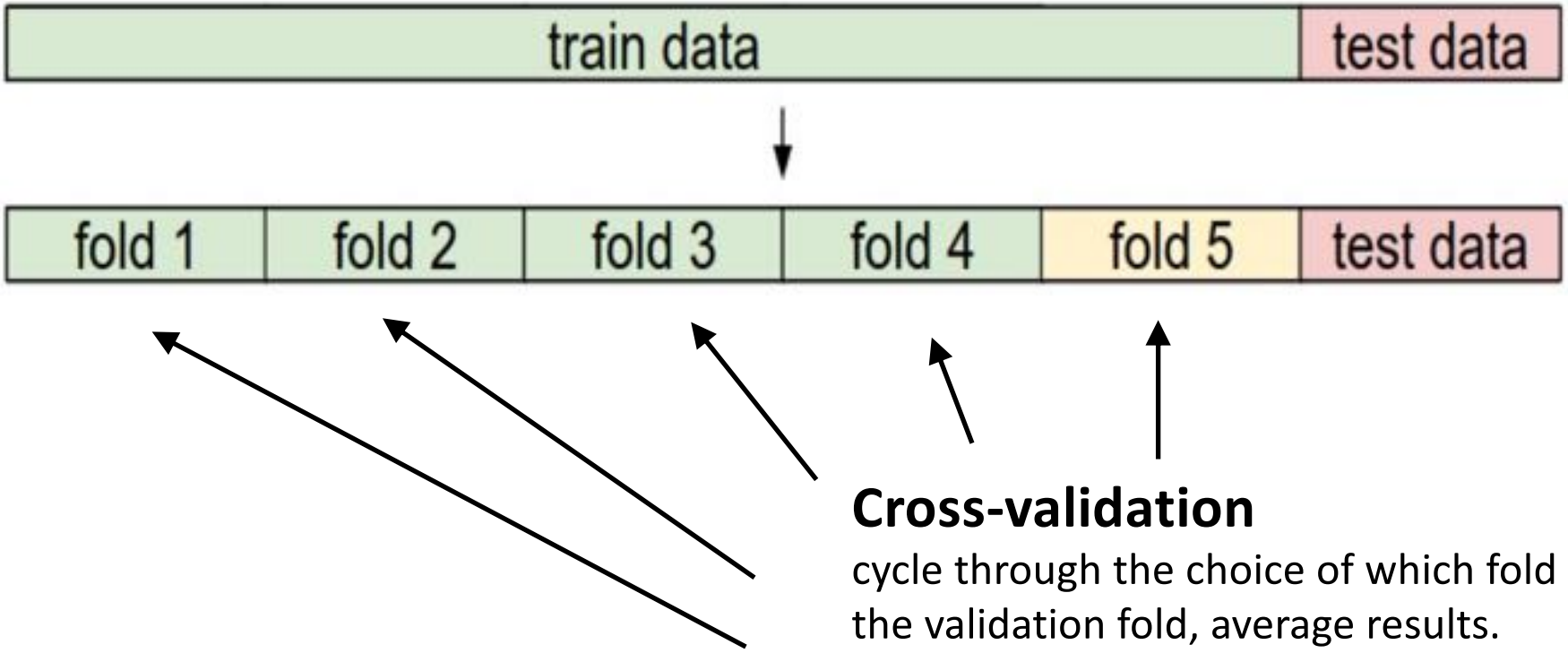
Trying out what hyperparameters work best on test set:

Very bad idea. The test set is a proxy for the generalization performance!
Use only **VERY SPARINGLY,** at the end.



Validation Set:

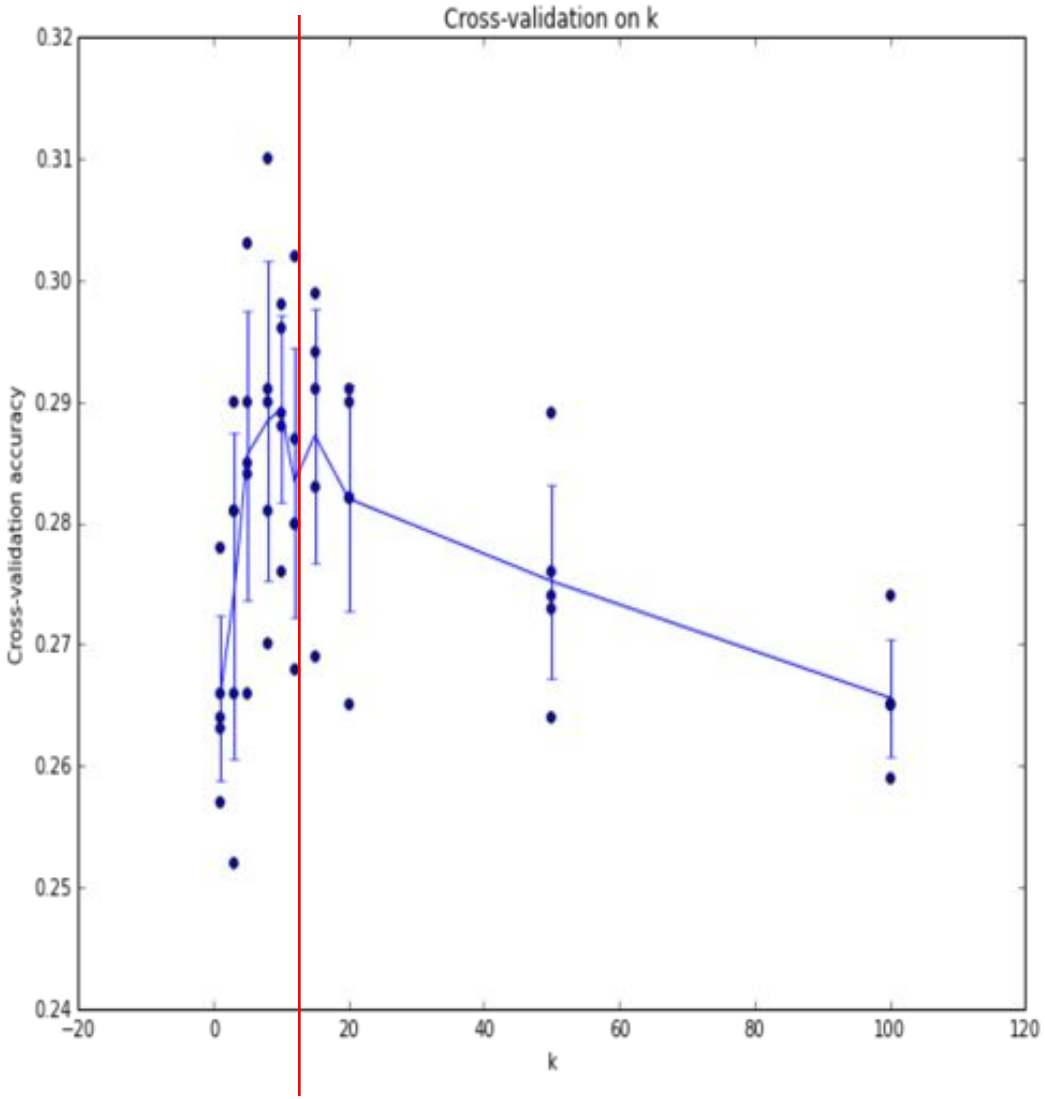
Validation Set:



Cross-validation

cycle through the choice of which fold is the validation fold, average results.

Hyperparameter tuning:



Example of
5-fold cross-validation
for the value of **k.**

Each point: single
outcome.

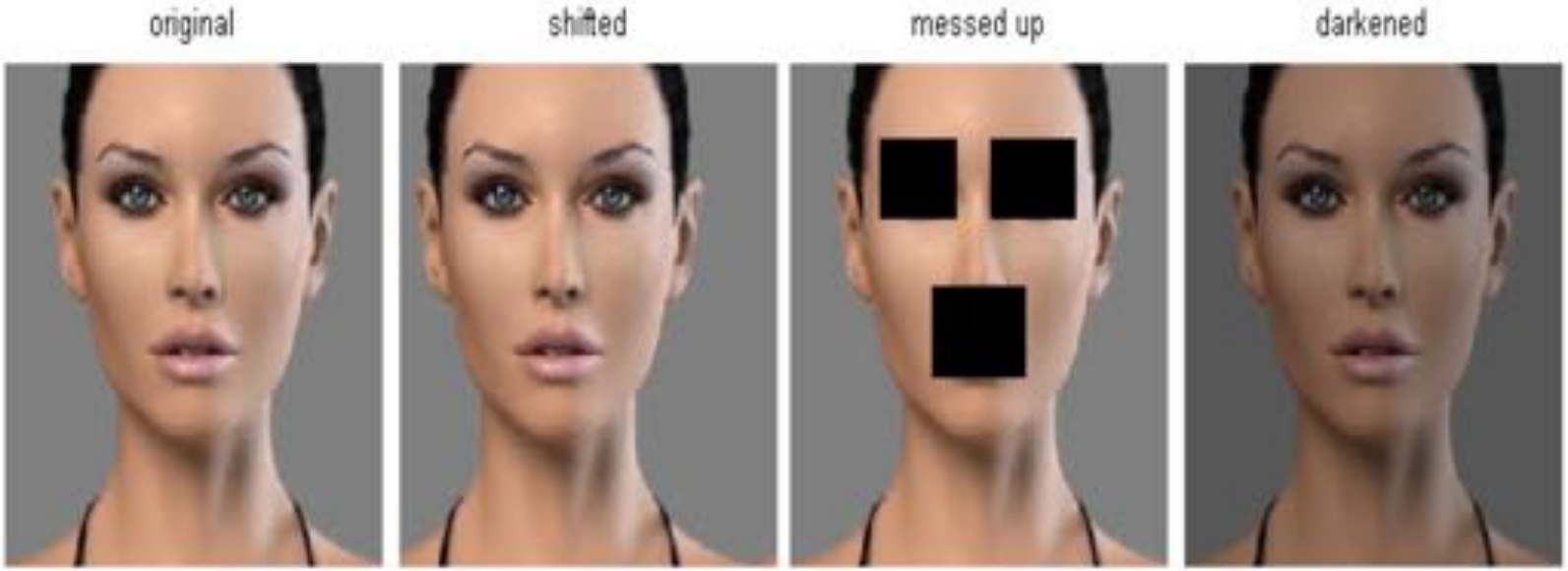
The line goes
through the mean, bars
indicated standard
deviation

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

k-Nearest Neighbor on images **never used.**

TECHNISCHE UNIVERSITÄT DARMSTADT

- terrible performance at test time
- distance metrics on level of whole images can be very unintuitive



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

What have you learnt?

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Image Classification:** given a **Training Set** of labeled images, predict labels on **Test Set**. Common to report the **Accuracy** of predictions (fraction of correct predictions)
- We introduced the **k-Nearest Neighbor Classifier**, which predicts labels based on nearest images in the training set
- We saw that the choice of distance and the value of k are **hyperparameters** that are tuned using a **validation set**, or through **cross-validation** if the size of the data is small.
- Once the best set of hyperparameters is chosen, the classifier is evaluated once on the test set, and reported as the performance of kNN on that data.