COMPUTER VISION 2018 - 2019

> TRAINING, CLASSIFICATION AND DETECTION

UTRECHT UNIVERSITY

RONALD POPPE

OUTLINE

Common vision tasks

- Image classification
- Object detection

Image description

- Global
- Local

Classification pipeline

- Training
- Testing

COMMON VISION TASKS

COMMON VISION TASKS

Output is a label per image, region or pixel

- We focus on classification (recognition) and object detection

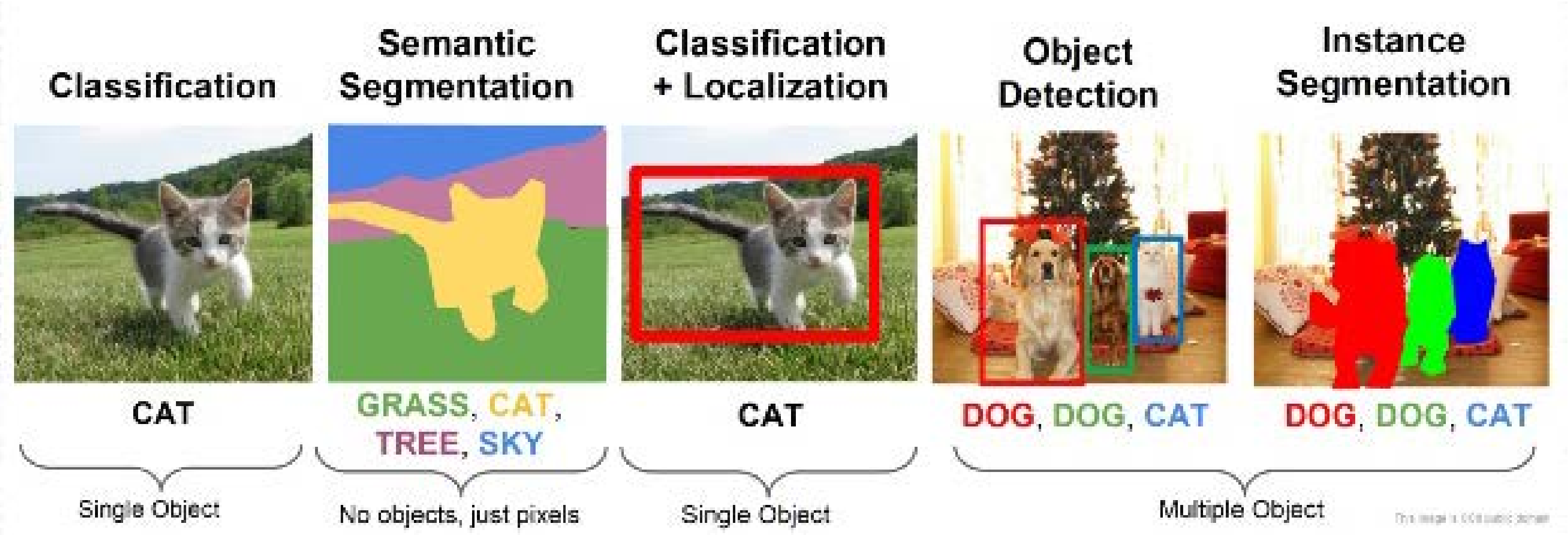


IMAGE CLASSIFICATION

Image classification: given an image, say what it depicts

- Labels correspond to classes (“koala”, “snake”, “cow”, …)

Image classification → “Koala”

IMAGE CLASSIFICATION²

Information is in the pixels

But there are all the variations (nuisance factors) that affect these pixels:

- Lighting
- Viewpoint
- Resolution
- Etc.

IMAGE CLASSIFICATION[3]

Instead of classifying the image directly, we can first extract image descriptors and then classify these

- Rely on invariancy of image descriptor

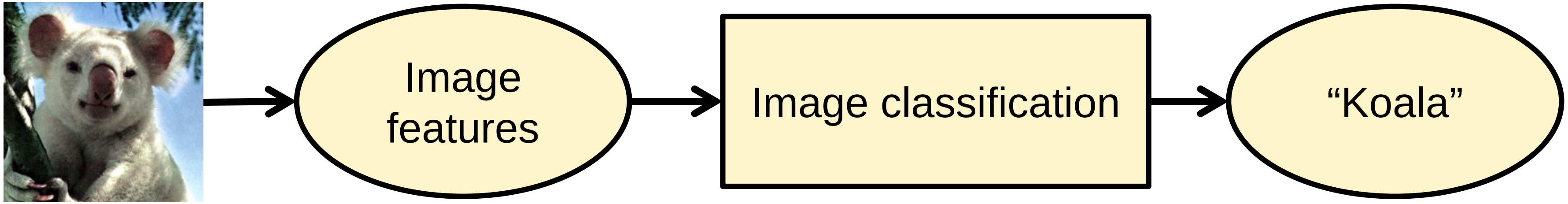


IMAGE CLASSIFICATION[4]

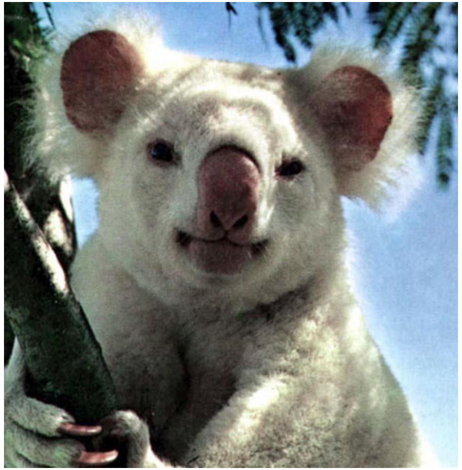
Image descriptors are either global or local

Global descriptors say something about the whole image

- All pixels are taken into account

Examples:

- Color histogram of whole image
- Voxel model
- HOG descriptor



OBJECT DETECTION

OBJECT DETECTION

Object detection requires information about only a part of an image

- Local image descriptors

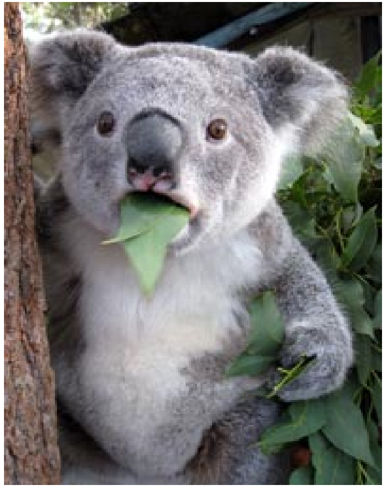
Examples:

- A region of pixels
- A SIFT descriptor

OBJECT DETECTION[2]

Object detection is the process of, given an image, finding regions that correspond to a specific object

- Output is not only class label but also bounding boxes



OBJECT DETECTION[3]

Instead of the whole image, we consider regions in the image

Typical approach:

- Divide the image into regions of a fixed size
- Check for each region whether it depicts the object or not (binary)

Regions can have different scales

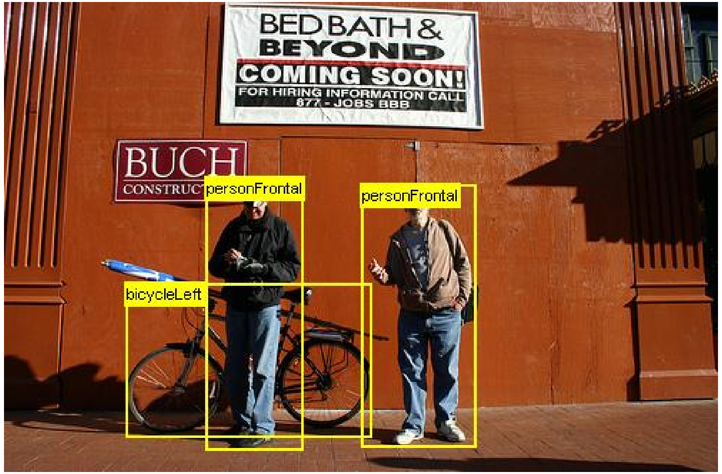
Regions can be overlapping

OBJECT DETECTION[4]

Regions are typically rectangular:

- Ease of computation
- Many objects are more-or-less rectangular

The rectangle in which an object appears is termed a bounding box



OBJECT DETECTION[5]

Regions typically have a fixed height-width ratio:

- Ratio is typically fixed for a specific object class
- E.g. people, faces, horses

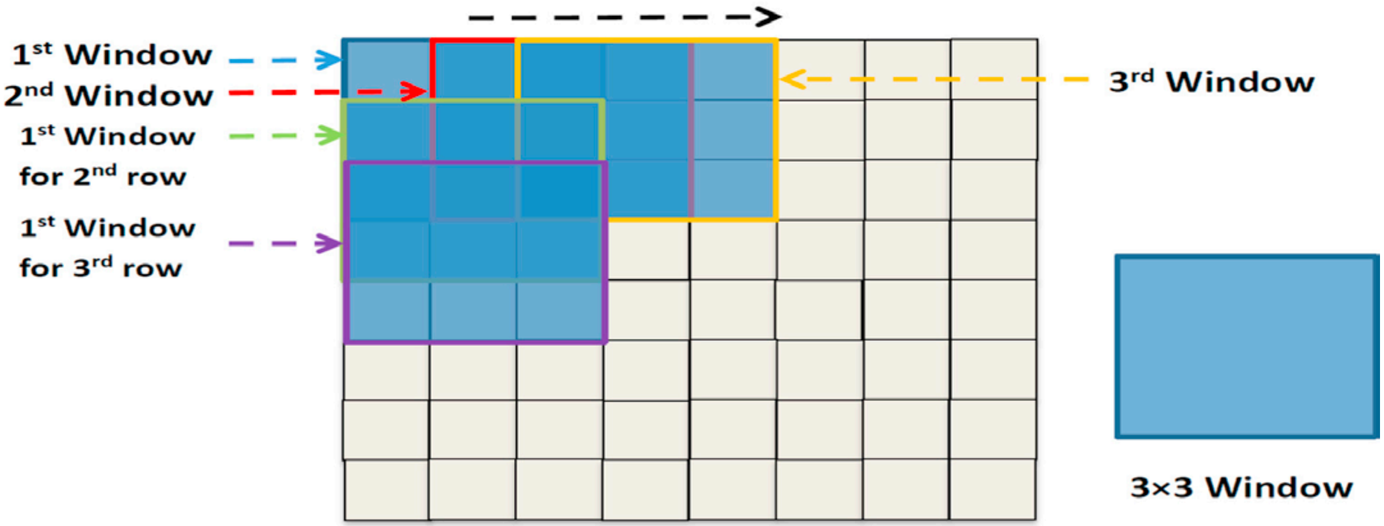
Again, saves computation time:

- Fewer different regions to evaluate

OBJECT DETECTION[6]

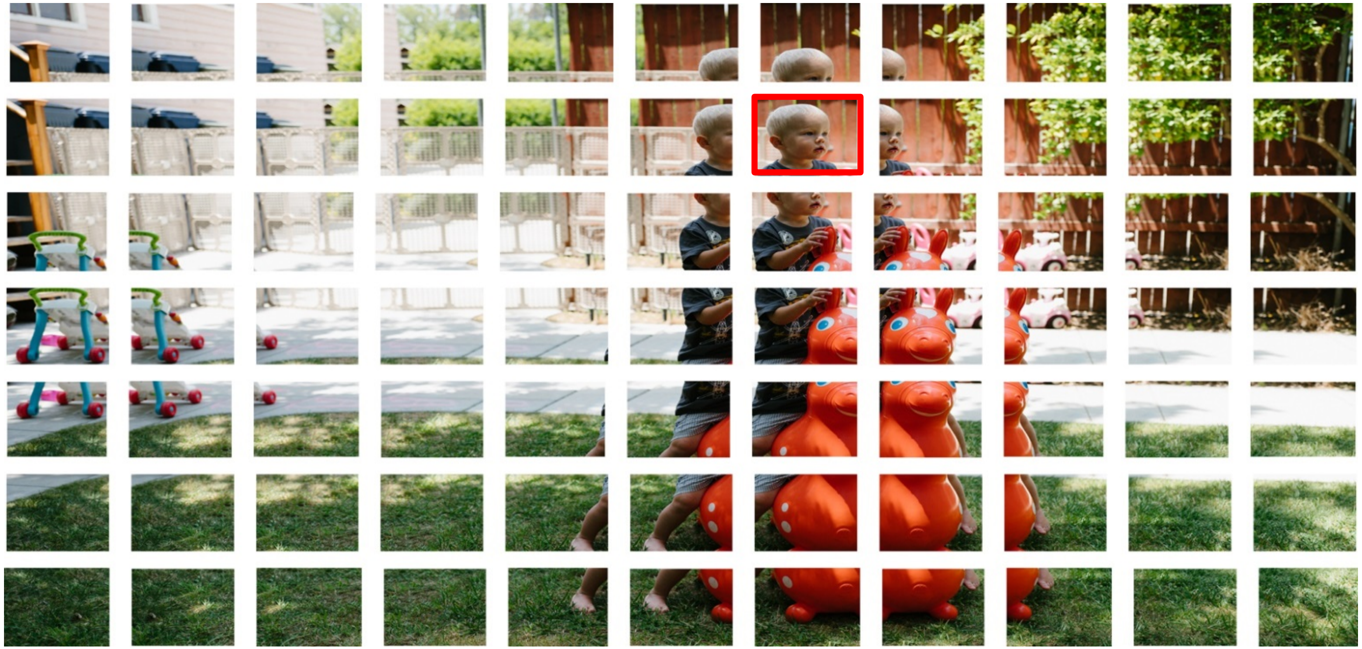
Sliding window approach converts object detection to a repeated image classification task

- Consider all possible windows of a particular size in the image



OBJECT DETECTION[7]

Example: sliding window with a fixed-size window



OBJECT DETECTION[8]

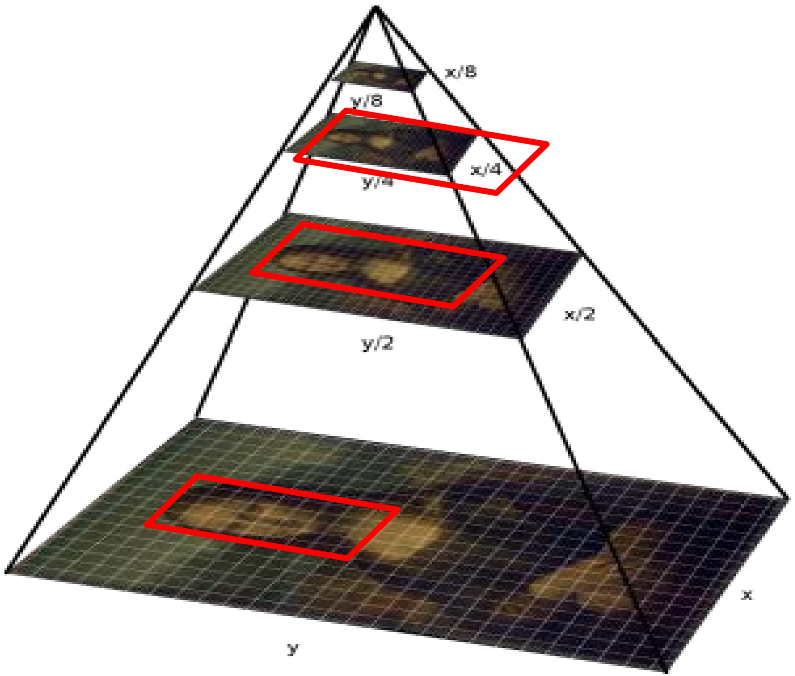
To deal with different scales, we can use an image pyramid:

- Each layer has a lower resolution
- Layers together form a "pyramid"

Pyramid image of higher level obtained by:

- Smoothing the current layer
- Sampling the current layer

With a fixed-size window, the object has a different size in each layer



OBJECT DETECTION[9]

A typical recipe for object detection with a sliding window:

Loop over possible scales (*s*)

  Loop over possible starting positions (*x*,*y*)

    Consider the region starting at (*x*,*y*) with scale s

    - Calculate image descriptors

    - Evaluate classifier → classifier score

    - Store position (*x*,*y*), scale *s* and classifier score

  End

End

OBJECT DETECTION[10]

Alternative is to use convolution

- Classifier is then a filter that outputs high values when the region is similar to the object we're looking for (similar to edge detection)
- Can be done very efficiently
- OpenCV supports many filters (e.g. based on pixel-values or HOG)

Core technique in Convolutional Neural Networks (CNNs)

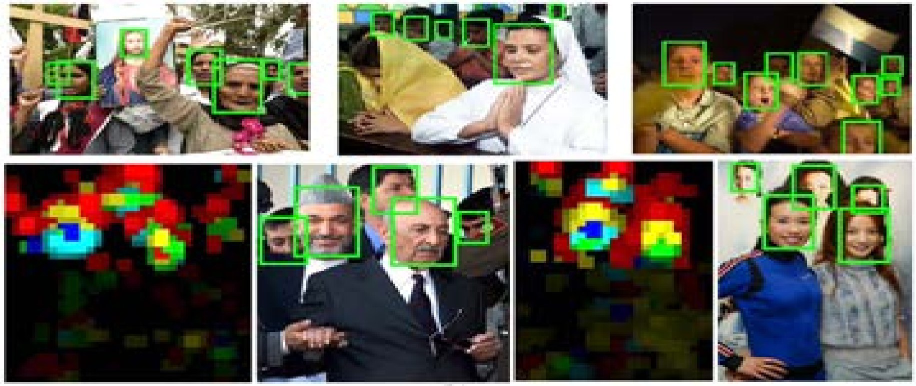
OBJECT DETECTION[11]

The output of convolution is a heat map or activation function:

- High values correspond to regions similar to the object class

We can threshold the detection scores in each layer to find candidate regions of different scales

- Layer determines size



OBJECT DETECTION[12]

Using a sliding window approach, many windows contain irrelevant parts of the image

- Even regions
- Background
- Noise

Recall that interesting objects typically stand out from the background in color or edges

OBJECT DETECTION[13]

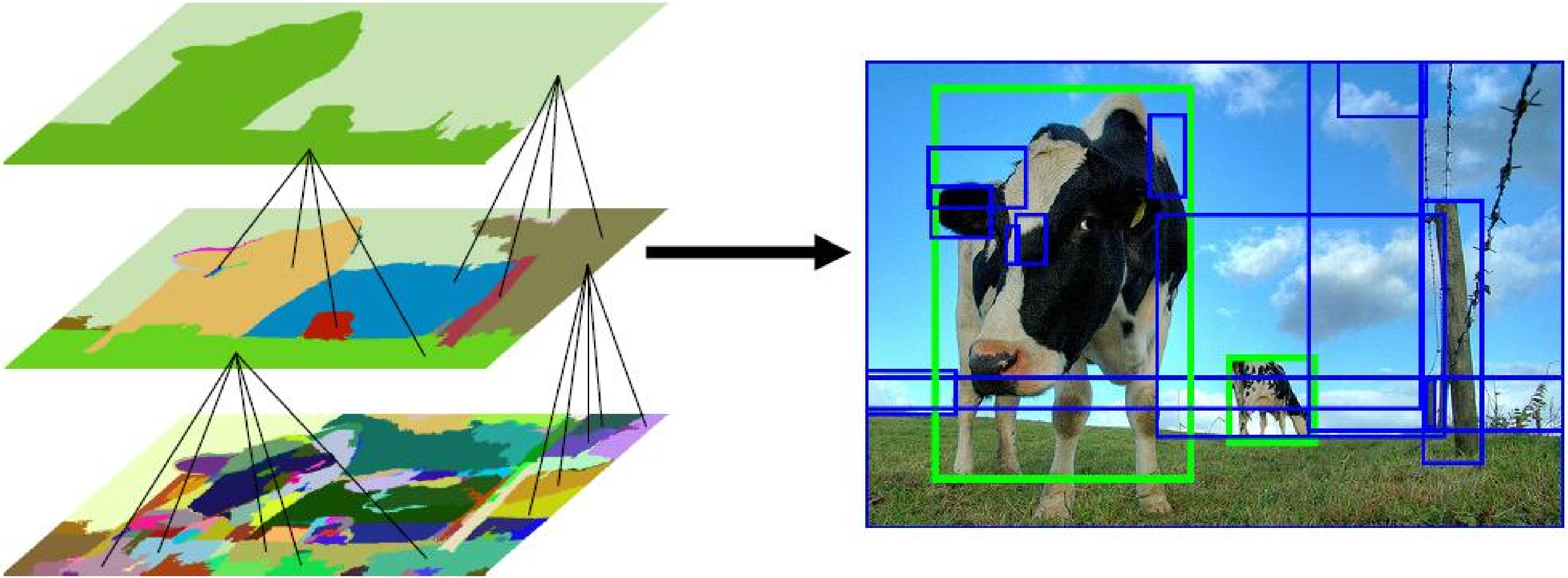
Selective search (Uijlings et al., 2013) is a technique to find promising regions in a bottom-up fashion

Iteratively merge clusters of pixels with similar characteristics

- Typically based on color or texture

OBJECT DETECTION[14]

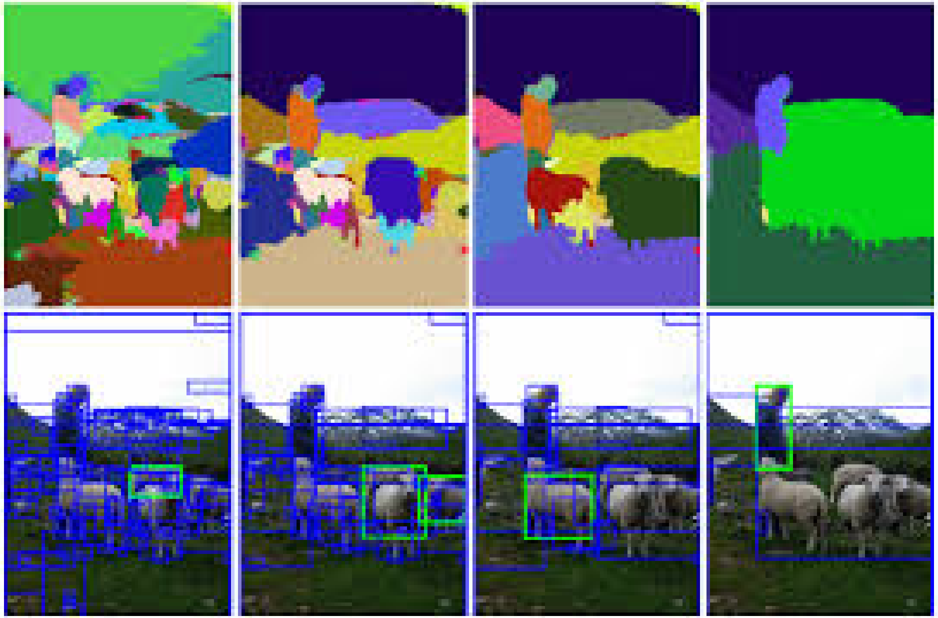
Evaluate bounding box around each cluster



OBJECT DETECTION[15]

Criterion on size determines when to stop merging regions

- Parameter that needs to be tuned



GLOBAL IMAGE DESCRIPTORS

GLOBAL IMAGE DESCRIPTORS

We start by looking at global image descriptors

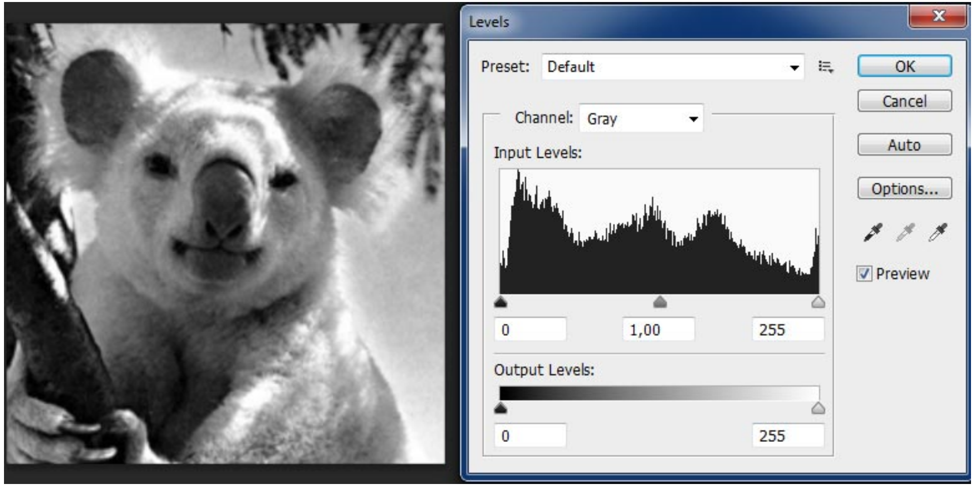
Whole image described with a single descriptor *x*

- $\boldsymbol{x}$ is the image descriptor (feature vector)
- n is the length of the feature vector
- $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x} = (x_1 \dots x_n)$

GLOBAL IMAGE DESCRIPTORS[2]

For grayscale images, we could have a histogram

- *n* is number of bins (e.g. 256)
- ***x*** is vector of percentages of occurrences of certain value



GLOBAL IMAGE DESCRIPTORS[3]

For color images, we could first cluster the colors, e.g., using K-means

- K is number of clusters
- For each cluster, we can count the pixels that belong to it
- Feature vector $\boldsymbol{x}$ is the percentage of pixels per cluster: $n = K$

GLOBAL IMAGE DESCRIPTORS[4]

When the image size changes:

- Number of pixels changes
- Number of clusters does not change!
- Percentage of pixels per cluster also doesn’t change (apart from rounding)
- Feature vector length remains the same
- Feature vector remains the same (apart from rounding)

GLOBAL IMAGE DESCRIPTORS[5]

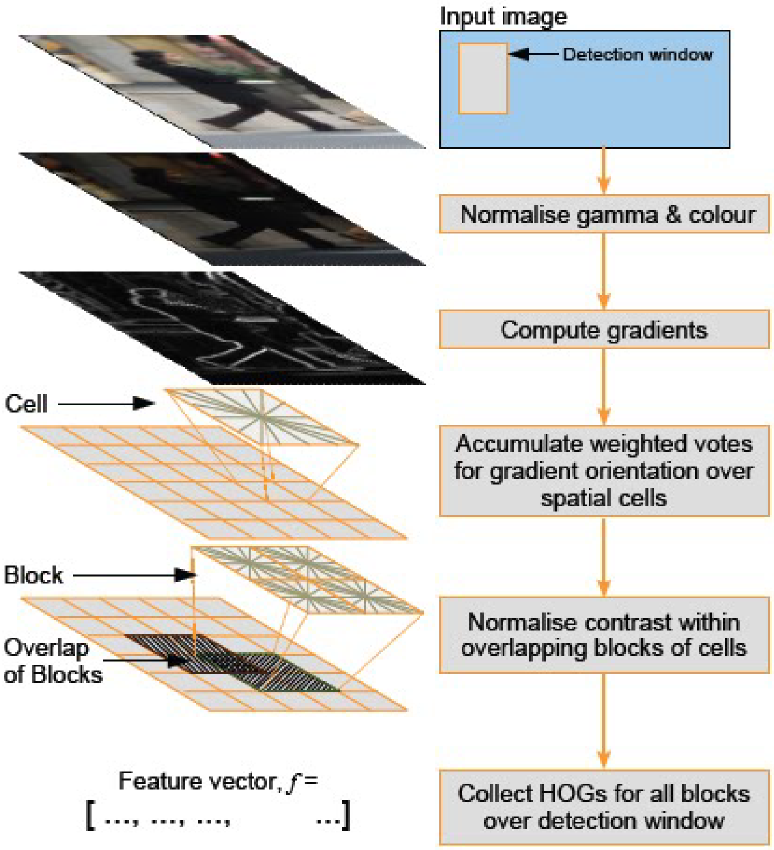
Histogram of oriented gradients (HOG) recap

Typically:

- 5 x 6 cells
- 8 orientations
- Blocks of 2 x 2 cells

Total number of blocks:

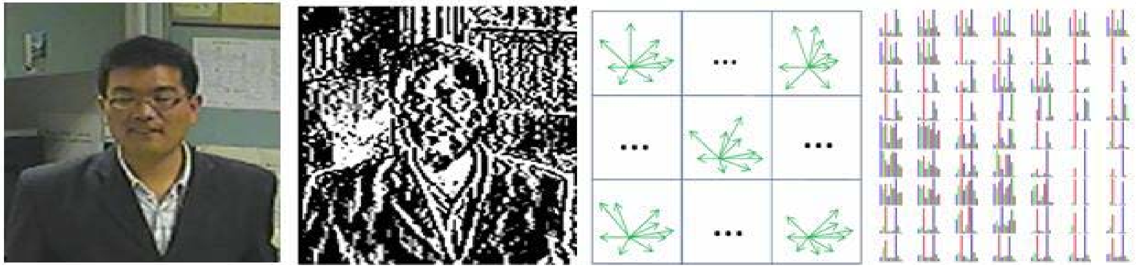
- 4 x 5 = 20



GLOBAL IMAGE DESCRIPTORS[6]

For a HOG feature vector, we have:

- 20 blocks, each contains 4 cells of 8 values (4 x a histogram):
- 640 values in total
- Feature vector $\boldsymbol{x} \in \mathbb{R}^{640}$



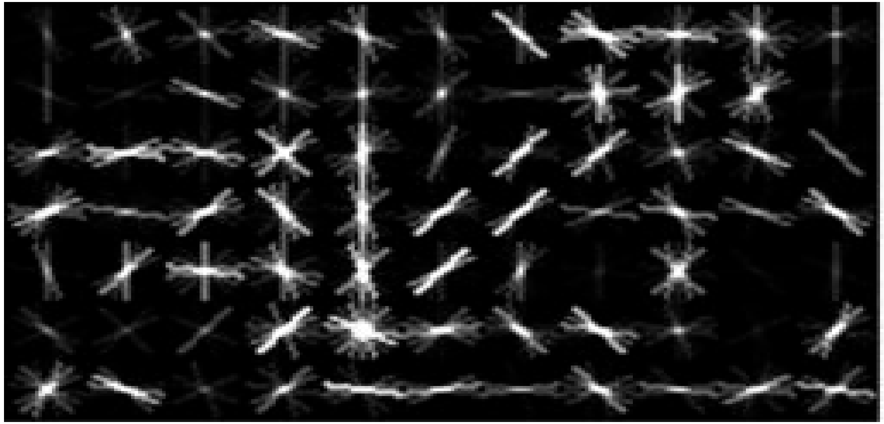
For a bigger image:

- More pixels per cell
- Number of cells, orientations and blocks remain the same
- Feature vector (length) remains the same

GLOBAL IMAGE DESCRIPTORS[7]

Global image descriptors can be used for image recognition

- They have the same descriptor length
- Can be compared with some distance/similarity function



LOCAL IMAGE DESCRIPTORS

LOCAL IMAGE DESCRIPTORS

So far, we have looked at global image descriptors:

- Single feature vector describes the whole image
- Vector lengths are the same

In contrast, local descriptors only say something about a small part of an image

- To describe a larger image, we need many local descriptors

LOCAL IMAGE DESCRIPTORS[2]

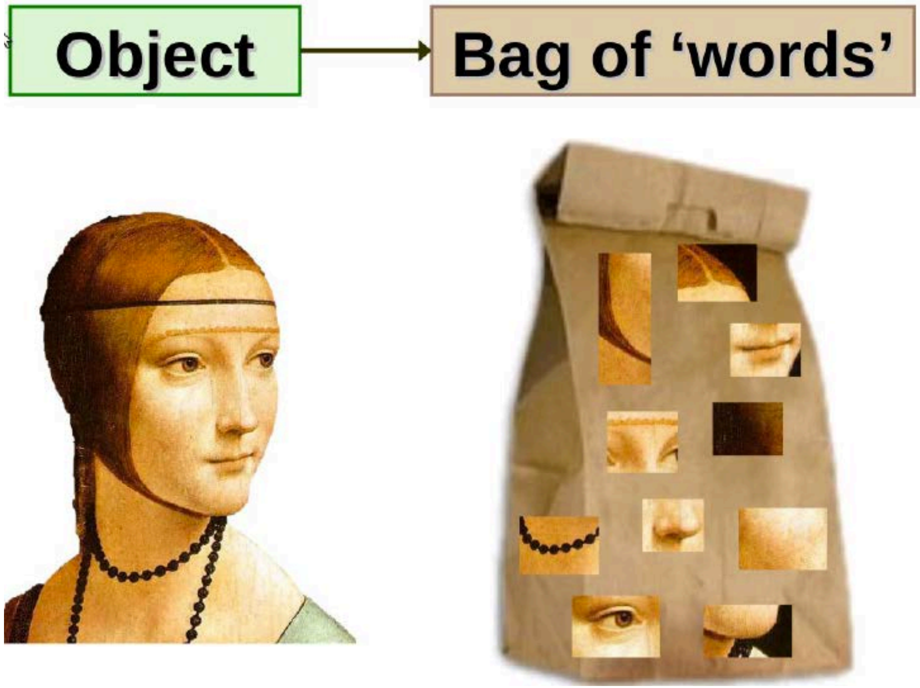
In general, we can extract any local feature (color, edge) at suitable locations in the image

- This leaves us with a **variable** number of image descriptors
- Comparison is not straightforward

For SIFT: more texture → more SIFT points

Solution: bag-of-words

LOCAL IMAGE DESCRIPTORS[3]



LOCAL IMAGE DESCRIPTORS[4]

A “word” is a local descriptor

- It is supposed to be distinctive, say something about an image (class)

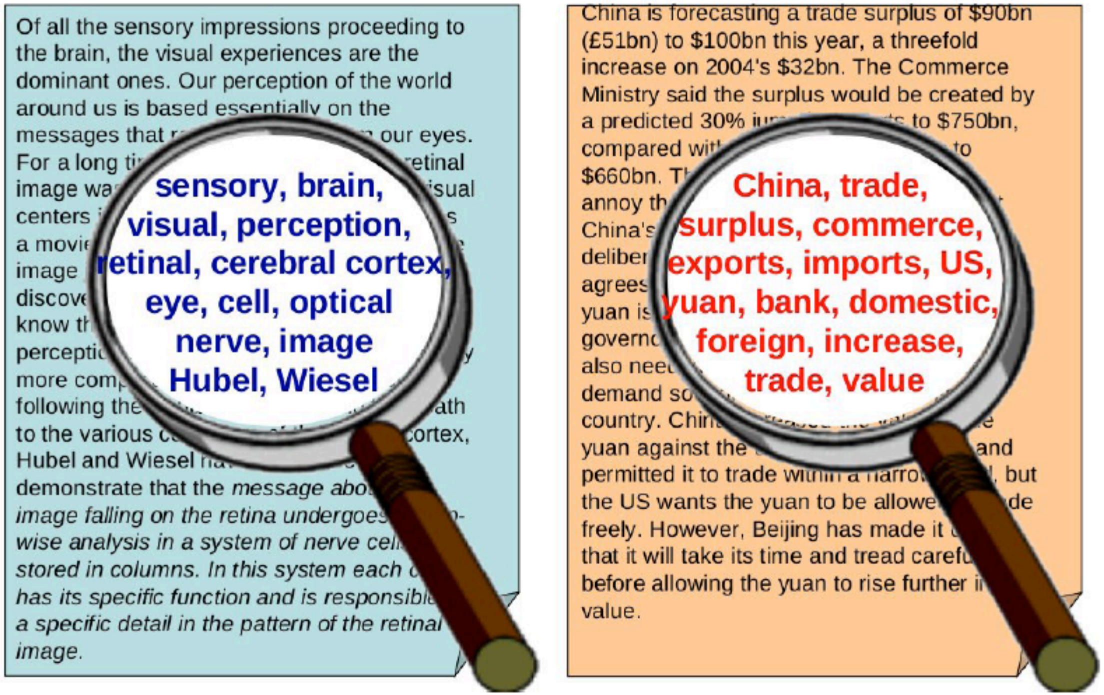
By “putting them in a bag”, we ignore spatial relations between words

- Conceptually not the most clever idea
- Computationally very efficient
- In practice works quite well
- Even helps in achieving invariance to translation and local variations

LOCAL IMAGE DESCRIPTORS[5]

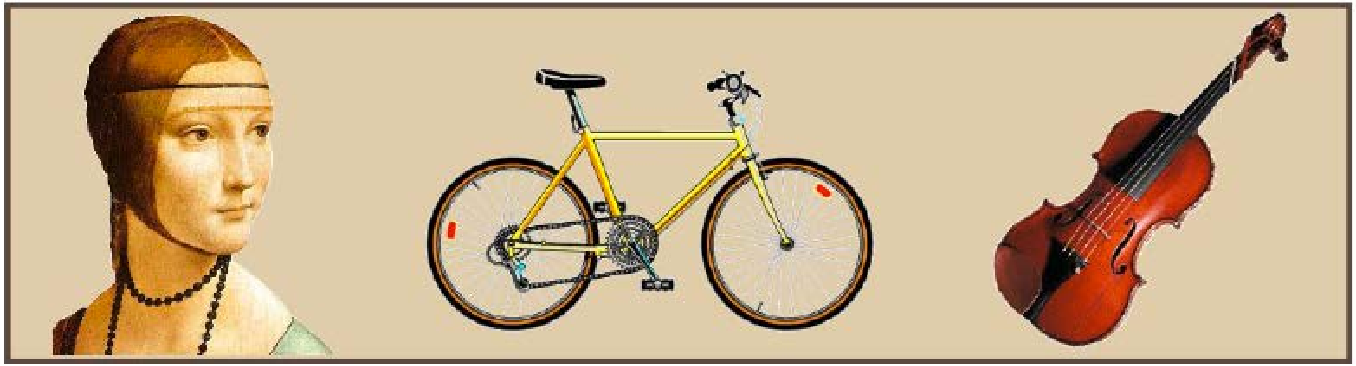
Analogy:

- Words can be distinctive for a certain type of text
- Some words appear more often than others
- The order of words in the text can often be ignored



LOCAL IMAGE DESCRIPTORS[6]

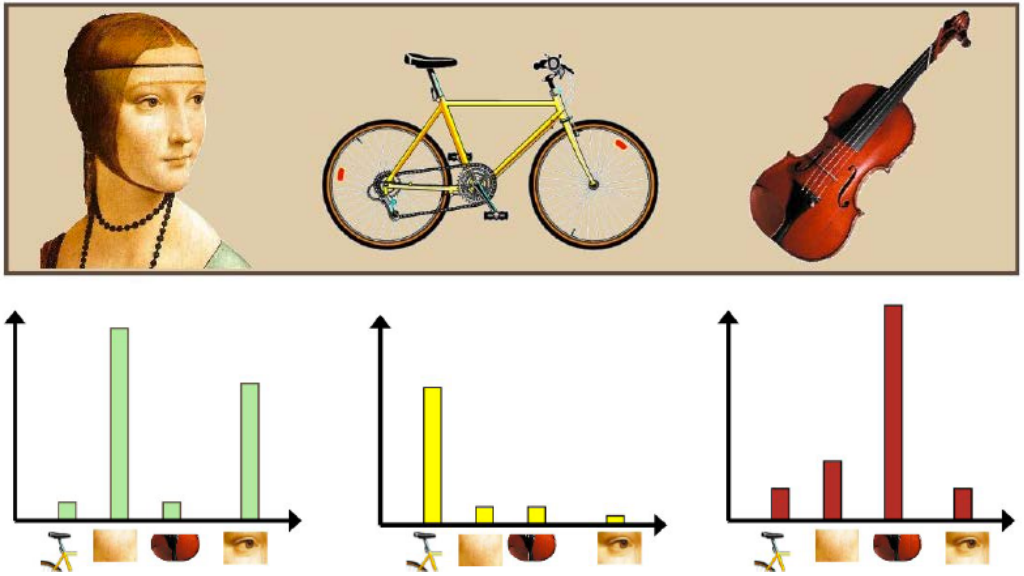
What if we would make a list of all possible words, across documents?



LOCAL IMAGE DESCRIPTORS[7]

We expect different word counts for images of different classes

- Images can be characterized by the counts of certain words



LOCAL IMAGE DESCRIPTORS[8]

A list of word frequencies is a histogram

- We can describe an image as such a histogram
- This is a feature vector!

Length of the vector is equal to the number of words:

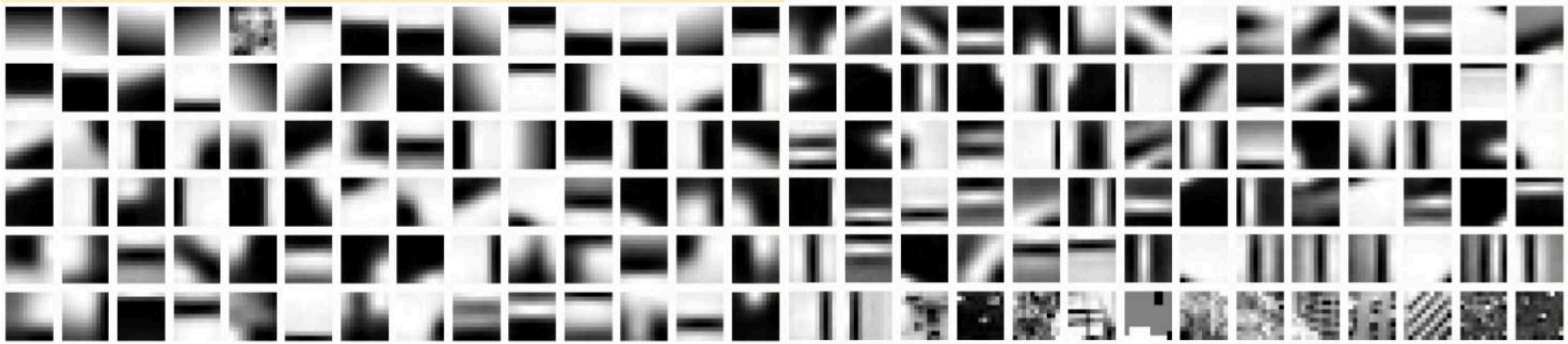
- All feature vectors have the same length
- Many possible words… and what IS a word?

LOCAL IMAGE DESCRIPTORS[9]

We need to find a subset of words to use in our histogram:

- Can be done using clustering (e.g. K-means)
- Each cluster is a “codeword” or “visual word”

Codeword is therefore a “concept” (similar words appear in the same cluster)



LOCAL IMAGE DESCRIPTORS[10]

Selecting the codewords is important:

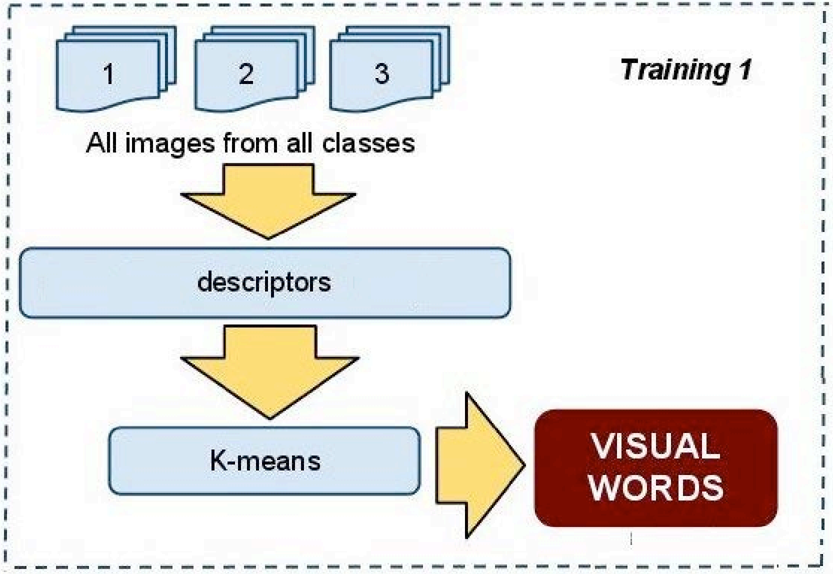
- To few might not be enough to distinguish between classes
- Too many might be too specific, leading to sparse histogram
- Codewords might be too common to distinguish classes (e.g. “the”, “a”)

Selection of discriminative codewords is beyond the scope of the lecture

LOCAL IMAGE DESCRIPTORS[11]

We learn the codewords from a training set *X*

- Local descriptors (e.g. SIFT) are obtained from all images in ***X***
- All words are clustered (e.g. K-means)
- A cluster center is a codeword



LOCAL IMAGE DESCRIPTORS[12]

Number of clusters is number of codewords

- This is the length of the feature vectors

We can describe an image as a histogram of codeword frequencies:

- We call this a bag-of-words (BoW) descriptor

LOCAL IMAGE DESCRIPTORS[13]

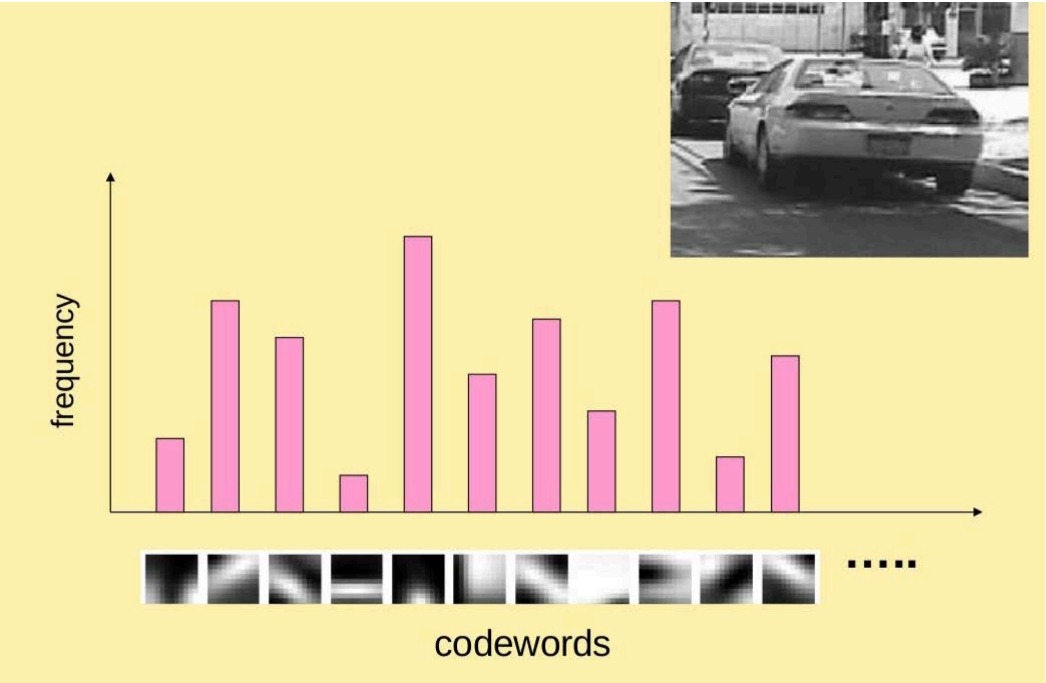
Obtaining a BoW descriptor for an image:

- Find keypoints
- Calculate the local feature descriptors (e.g., SIFT)
- Map all local descriptors onto codewords by checking which codebook cluster center is closest
- Each keypoint/local descriptor adds 1 to one bin of the BoW

To deal with arbitrary numbers of local descriptors:

- Normalize the BoW (histogram) to unit length

LOCAL IMAGE DESCRIPTORS[14]



QUESTIONS?

CLASSIFICATION

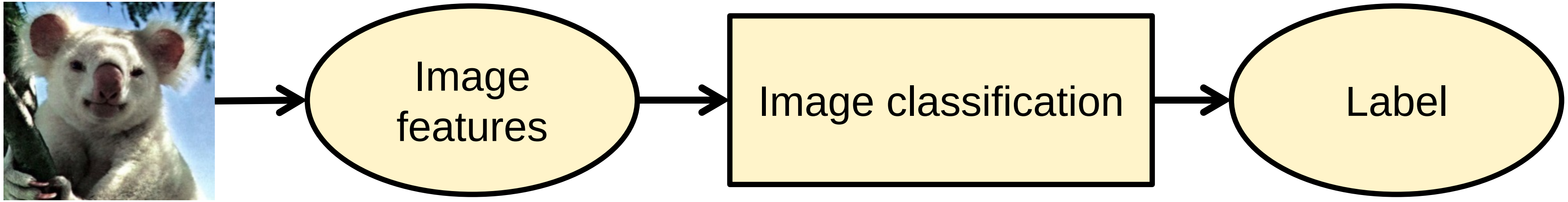
CLASSIFICATION

Classification is the process of assigning labels *y* to input vectors *x*

- Input vectors ***x*** are image descriptors (color histogram, HOG)
- Labels *y* are image classes ("koala", "car", etc.)

We usually use a (machine learning) classifier

- Classifiers come in many variations



CLASSIFICATION²

Classifiers can be "trained" to distinguish between classes

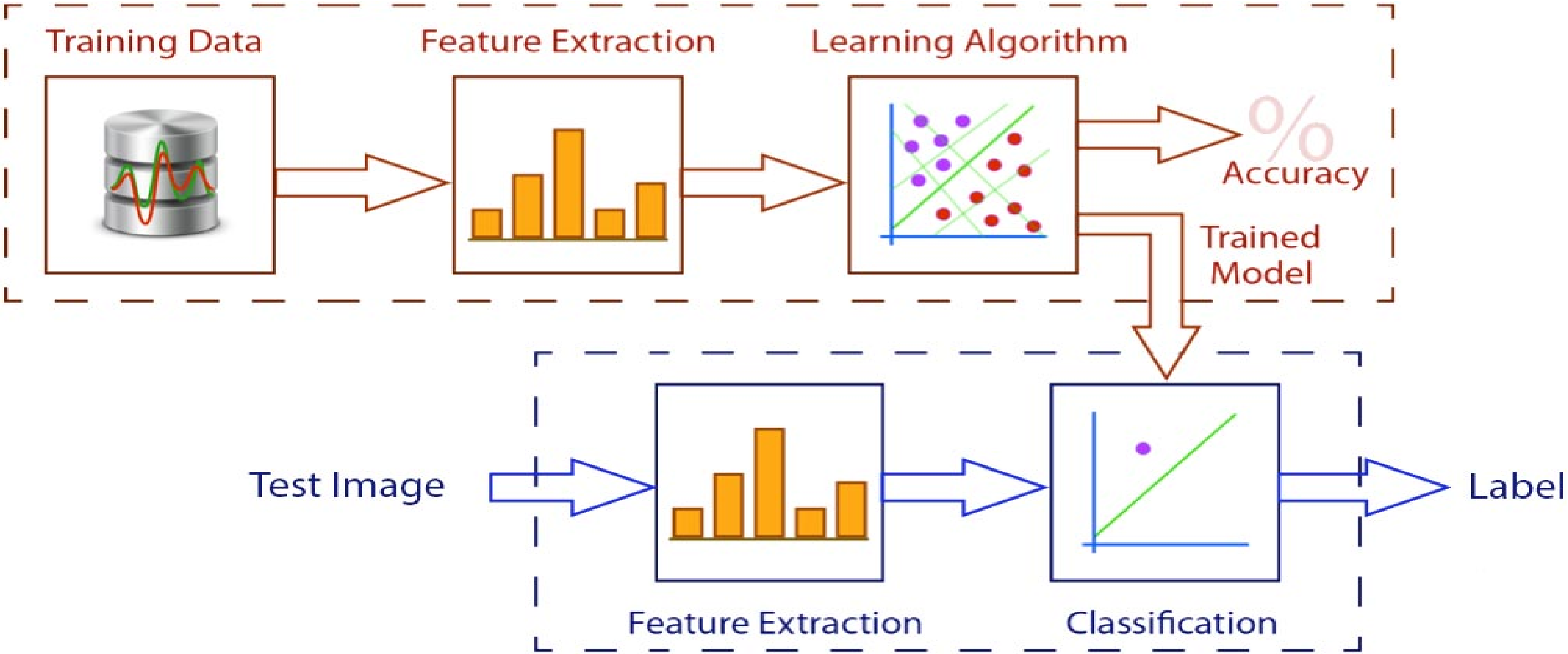
- Requires training data
- Requires a learning algorithm
- Output is a "trained classifier" (or "trained model")

Once we have a trained classifier, we can "test" a new image

- Estimate the label of the image, given its feature vector
- Process is called "inference" (or "classification")

CLASSIFICATION[3]

Training and testing:



CLASSIFICATION[4]

There are different ways of training a classifier:

- Supervised: provide ***x***-*y* pairs (labeled data)
- Unsupervised: only provide ***x*** (unlabeled data)
- Semi-supervised: provide some ***x***-*y* pairs and some unlabeled data ***x***

We'll focus on supervised classification

- Most common and is easiest to train

CLASSIFICATION[5]

For supervised learning, we provide a learning algorithm with labeled data *X*:

- Set (**x**, y) of image descriptor ***x*** with label *y* (E.g. (1, 1, 2, 5, 1, …) → "koala")
- The number of pairs in the dataset ***X*** is *m*
- The dimensionality of the feature vectors is *n*

The task of the learning algorithm is to train a model that can predict the labels for new data

- The trained classifier should generalize to unseen data
- Therefore, we need to provide it with a sufficient amount of relevant data

CLASSIFICATION[6]

Supervised learning algorithms can have many forms:

- Binary: one class vs. the rest (koala vs. no koala)
- Multi-class: models each class (koala vs. cat vs. cow)

In both cases, the algorithm minimizes a "loss function":

- Training samples ***x*** that are wrongly classified introduce a penalty term

The result of training is a trained classifier

- All parameters have been determined
- Model with minimum loss on the training data

CLASSIFICATION[7]

The idea is to separate one class from one or all others based on the image descriptors (feature vectors)

Difficult due to intra-class variation

- Not all images of the same class are similar
- Sometimes a lot of variation

Difficult due to inter-class similarities

- Images of one class could be very similar to images of another

CLASSIFICATION[8]

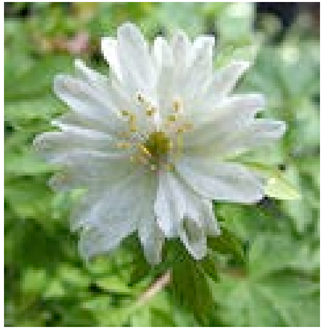
These are daisies:

CLASSIFICATION[9]

And so are these:

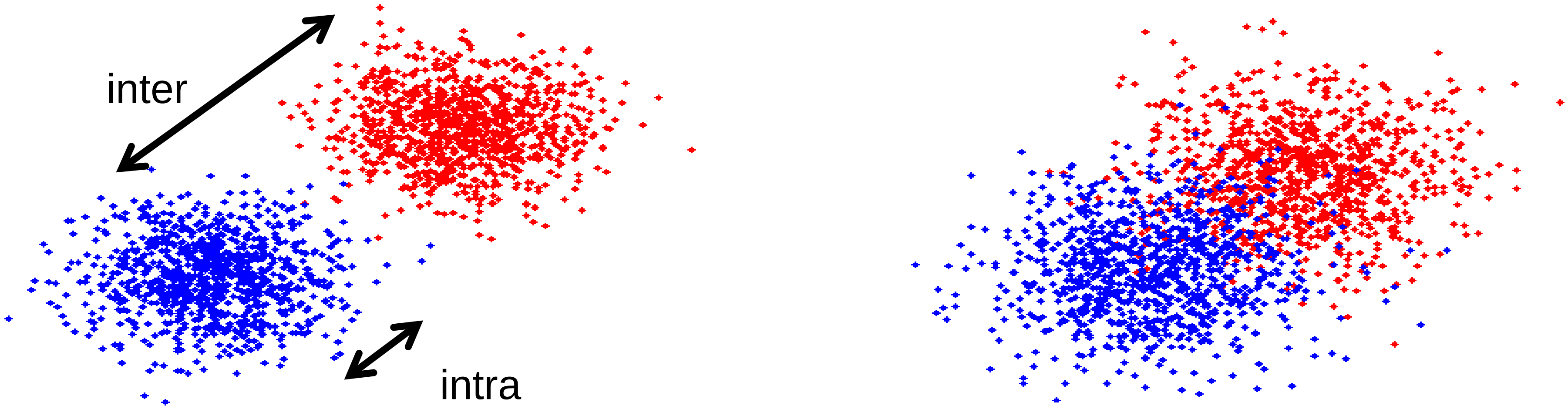
CLASSIFICATION[10]

These are windflowers:

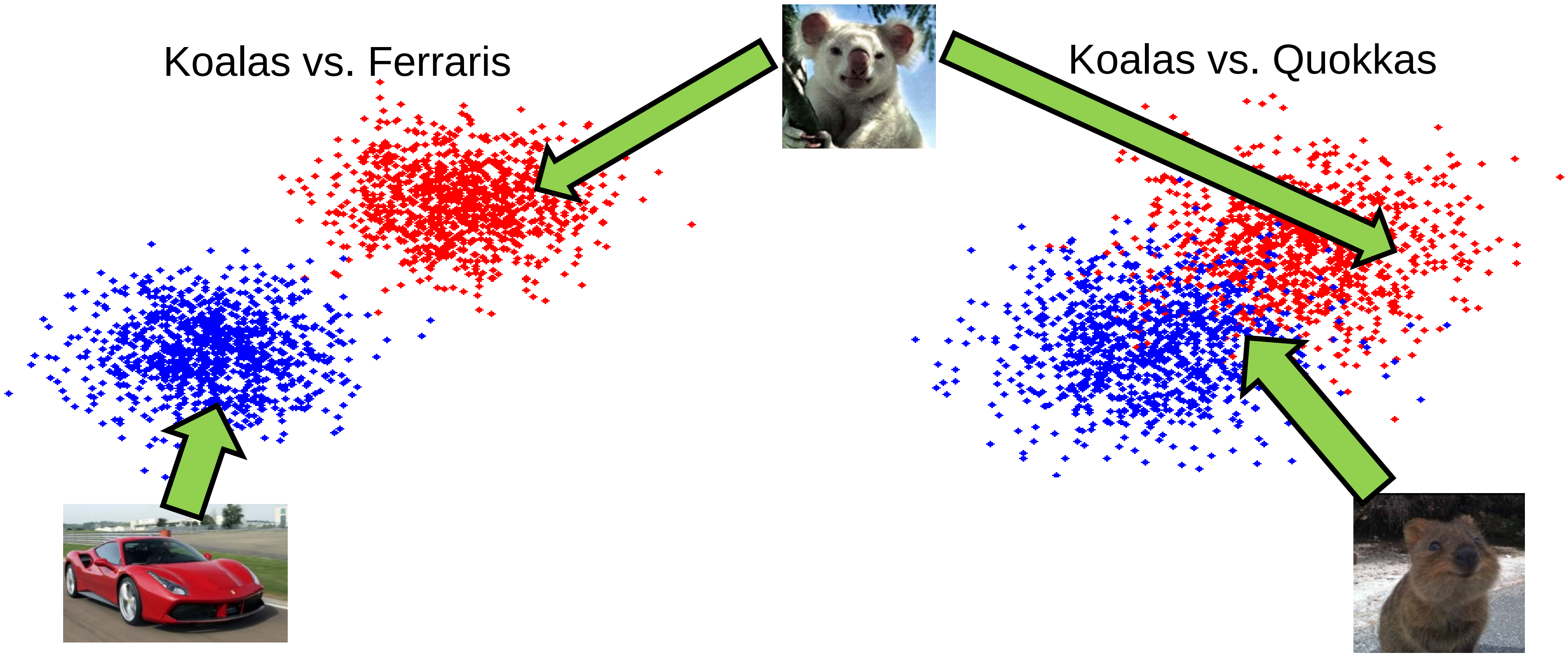


But these are daisies:

CLASSIFICATION[11]



CLASSIFICATION[12]



CLASSIFICATION[13]

To distinguish between classes, we can use characteristics:

- Means per class
- Intra-class variance
- Inter-class variance
- Number of samples per class
- Etc.

Typically, we extract this information from the training set *X*

CLASSIFICATION[14]

In this lecture, we will treat the classifier as a black box

- There are many options (SVM, Random Forrest, Naïve Bayes, etc.)
- Despite huge differences, many issues regarding training and testing are universal

QUESTIONS?

TRAINING AND TESTING

TRAINING

To train a classifier, we need data for all classes

- When we focus on a single class (binary), we need an “other” class

Data divided into positive (class) and negative (class) data

- Positive data can typically be obtained easily
- What kind of data should be considered as negative data?

TRAINING[2]

Negative examples should be taken from the same domain

- If we consider face detection from surveillance video, negative samples should also come from surveillance video
- If we detect them in cartoons, the negative samples should come from that domain

One option is to consider (parts of) images in which the class does not appear as negative samples

- Samples have same characteristics in terms of lighting etc.

TRAINING[3]

Example of positive and negative examples for face detection

TRAINING[4]

When training a classifier, we want to obtain a model that generalizes to unseen images

- We do not only want to classify the images in the training set, but also images similar to those in the training set
- The training set should be representative of our domain

Nuisance factors that we want to overcome should be part of the training set

- E.g. variation in lighting, viewpoint, etc.

TRAINING[5]

There are typically parameters in the classifier that affect the performance:

- E.g. a threshold, the number of clusters
- For the CNNs in later lectures: weights and biases

How do we know when we have trained a good classifier?

TRAINING[6]

Option 1: look at the performance on the training set

- Performance on the training set is the percentage of correctly classified data points
- Remember: a classifier does not always distinguish 100% correctly on a training set (why?)

However, now we're not testing the generalization ability

- We still do not know the performance on images we have never seen

TRAINING[7]

Option 2: divide dataset of (image, label) pairs into a training set and a validation set

- We use part of it for training the classifier (training data)
- And part of it to validate the classifier (validation set)
- The validation set is unseen during training (it is “clean”)

Typically more data for training

- E.g. ratio training/validation set size 80% / 20%

TRAINING[8]

The performance on the validation set should ideally be similar to that on the training set:

- This would indicate that the model generalizes to unseen data

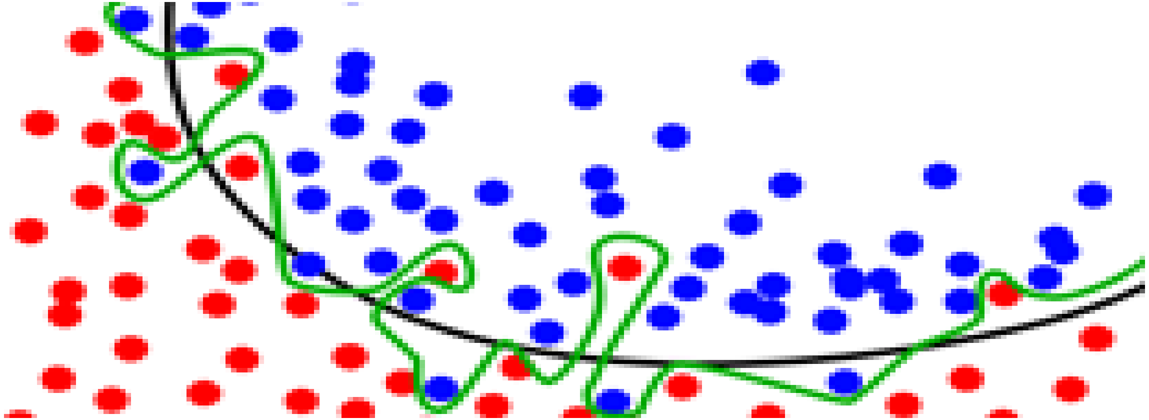
If it is lower, we might have overfitted on the training set:

- Instead of learning object classes in a more general sense, we have learned to recognize the specific images
- The trained model does not generalize well
- This is a challenge, especially with more complex image concepts

TRAINING[9]

For Support Vector Machines (SVMs):

- “Complex hyperplane” (green): better score on validation set
- More likely hyperplane (black): conceptually better distinction

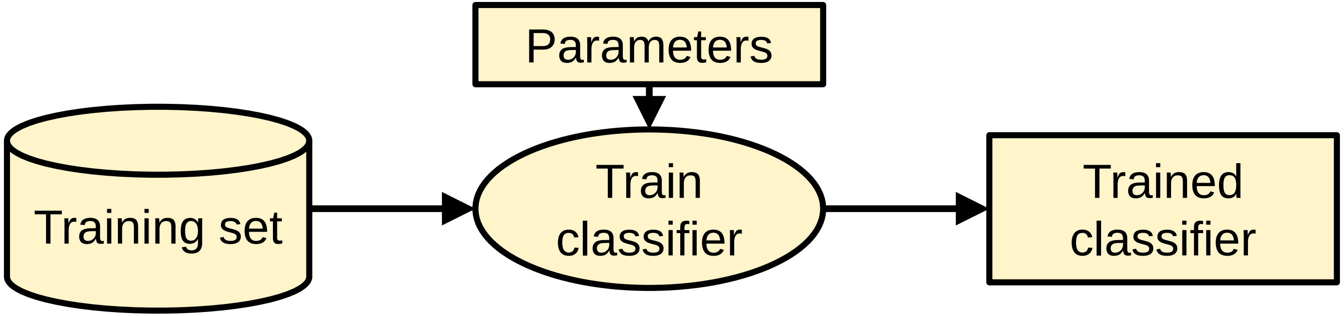


TRAINING[10]

When training a classifier, there are typically some parameters involved

How do we determine these parameters so that our classifier is optimal?

- Again, we aim at generalization



TRAINING[11]

Solution: use your validation set to test each setting

Recipe:

Loop over parameter instances

  Train classifier on training data

  Test classifier on validation data

End

Select parameters with best score on validation data

TRAINING[12]

We have assumed that our validation set is "similar" to the training data

- This is not always the case: biases might occur
- There are always "easy" and "hard" images
- The performance on each validation set can then vary

We want to minimize the chance that our split into training and validation set is biased

TRAINING[13]

Solution: cross-validation

- Divide your dataset into several “folds”
- Each fold contains approximately the same amount of data
- Each fold has approximately the same class distribution

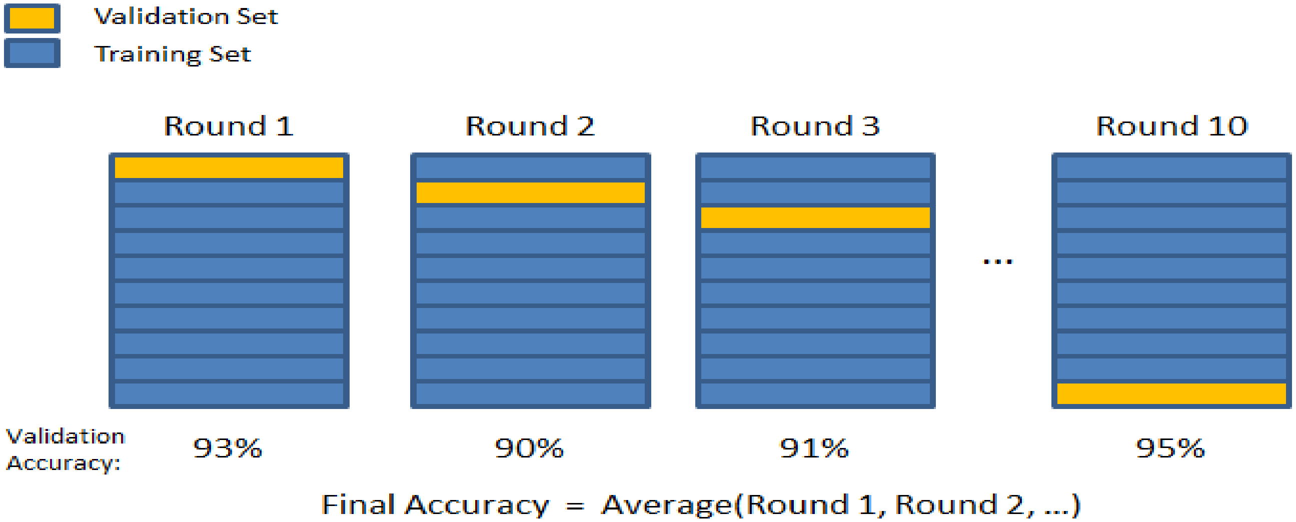
During training:

- Loop over these folds
- Every iteration, a different fold is the validation set. All others together are training set
- Multiple evaluations of same parameter setting, but with different data selections

TRAINING[14]

Best parameter setting is based on average performance over all folds

Example: 10-fold cross-validation



TRAINING[15]

As we measure the performance on the validation set, the number of data points per class is important

- Bias in class distribution affects performance of the classifier

Example: what happens when only one out of 100 images depicts a koala?

- We could always get 99% recognition if we would always “guess” an image does not depict a koala
- But then we would never recognize one…
- Many false negatives (more about this next lecture)

TRAINING[16]

Two options:

1. **We balance to make sure the number of training samples per class reflects that of the real world**
   - Not suitable when class distribution is very biased

2. **We balance the number of training samples per class and later multiply the probability of the class with its prior**
   - Again, can be tricky for very skewed class distributions

Later, we will see other ways by choosing a proper loss function

TRAINING[17]

Recap:

- To train a classifier, we need positive and negative training examples
- Performance of classifier can be determined on validation set
- Using cross-validation is more robust

Different parameter settings can be evaluated on validation set (once, or using cross-validation)

Dealing with skewed class distributions is a point of attention

TESTING

Once we have a trained classifier, testing is straightforward:

- Extract image descriptor
- Evaluate the trained classifier
- Retrieve the label

In many experiments, we use a separate test set that is withheld from the dataset

- The test set should not have been "seen" at all during cross-validation
- It is only used once, when the parameters are determined

QUESTIONS?

NEXT LECTURE

NEXT LECTURE

Deadline for Assignment 3:

- This Sunday March 10, 23:00

Assignment 4 will be online soon!

Performance measures

- Tuesday March 12, 13:15-15:00, BESTUURS-LIEREGG