Generic Object Recognition

10 Bag of Visual Words and Support Vector Machine

Kimiaki Shirahama, D.E.

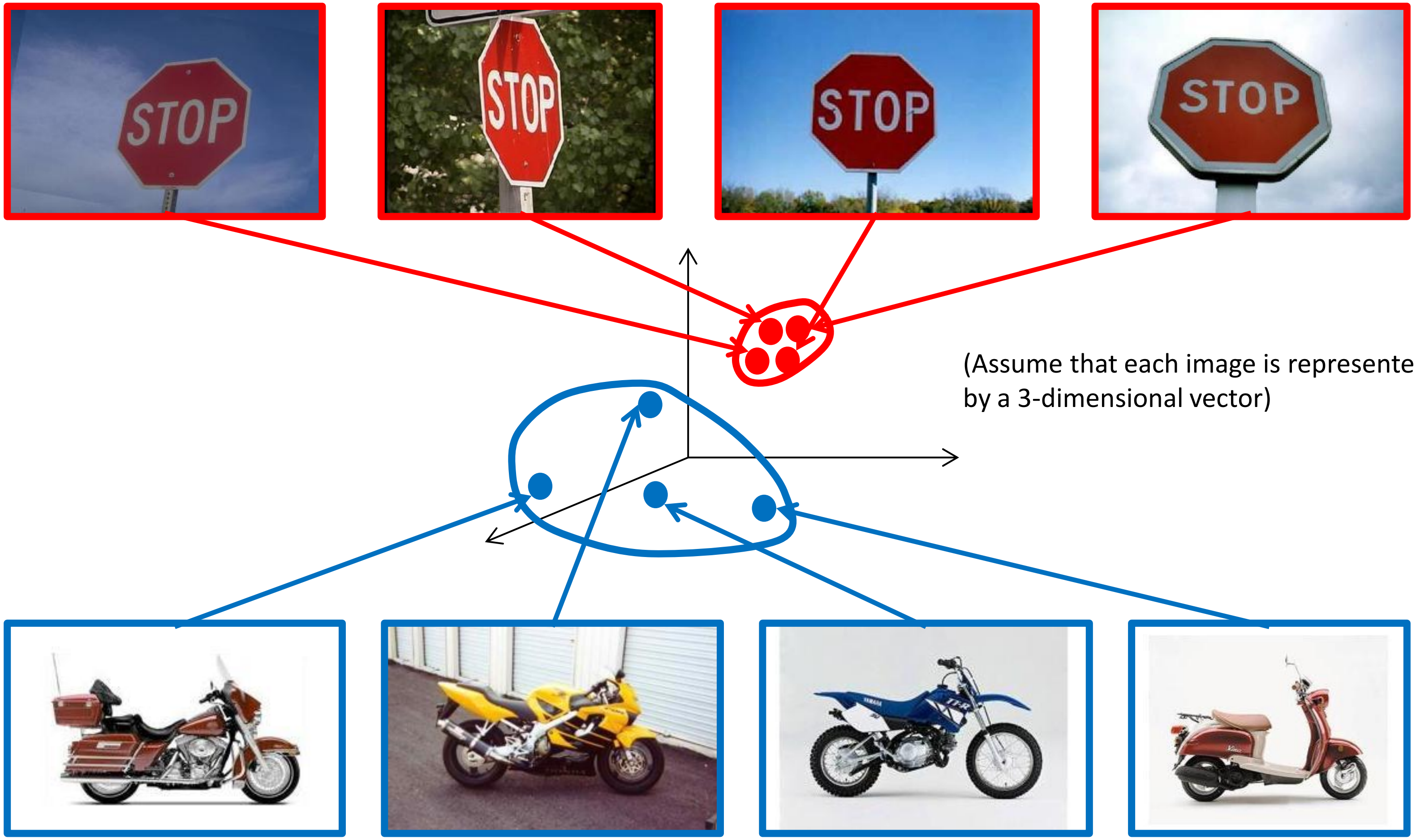
Research Group for Pattern Recognition
Institute for Vision and Graphics
University of Siegen, Germany

Overview of Today’s Lesson

- Generic object recognition
- Generic object recognition using Bag of Visual Words (BoVW)
- Short break
- Procedures for generic object recognition using BoVW
  - ❑ Visual word extraction
  - ❑ BoVW representation
  - ❑ Classification
- Implementing visual word extract by OpenCV

Specific Object Recognition vs. Generic Object Recognition

Specific object recognition: Detect instances of the same object in different images



(Assume that each image is represented by a 3-dimensional vector)

Generic object recognition: Identify the class (category) of objects in images

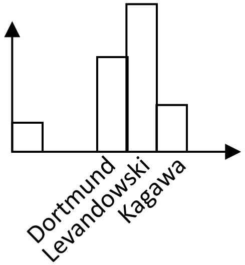
⇨ *Diverse visual appearances of objects in the same class (Inner-class variance)*

Generic Object Recognition Using Bag of Visual Words (BoVW) (1/2)

Analogy with document retrieval

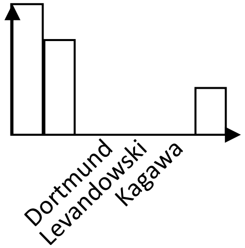
(Retrieving documents about Football)

… Dortmund … Lewandowski
… Kagawa … Dortmund …
….



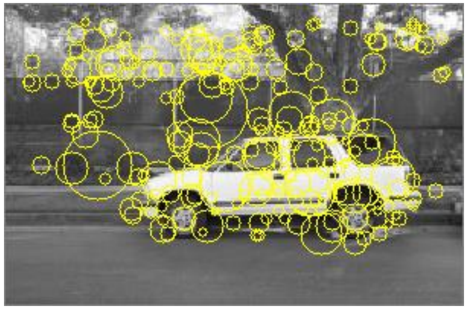
Retrieved because this contains a log of words related to Football

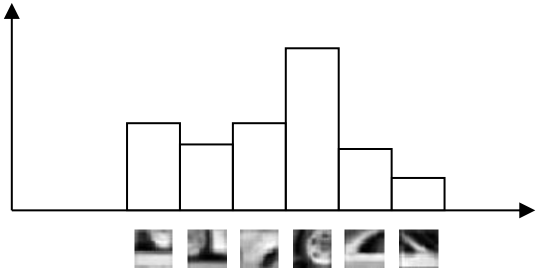
… Baseball … Major …
Yankees … New York …
….



Not retrieved because this does not contain words related to Football

What are words in images? *Characteristic (typical) local features*





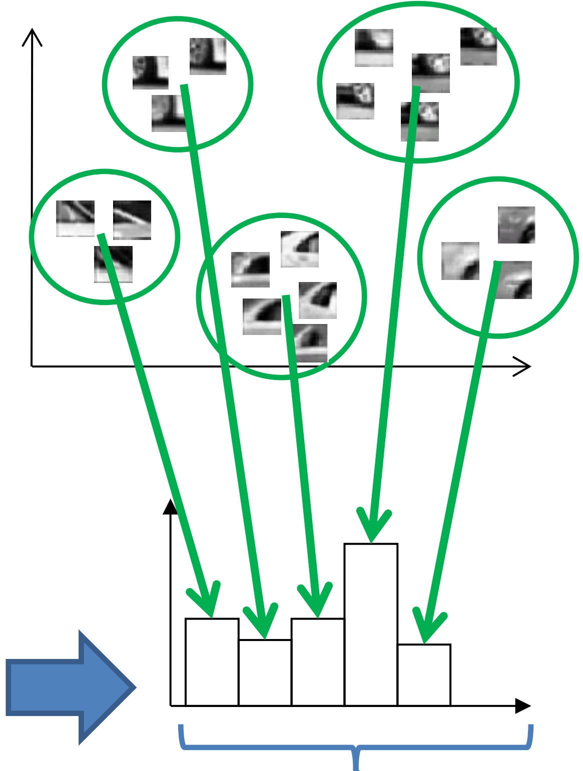
Bag of Visual Words (BoVW) is the representation of an image using a histogram where each bin represents the frequency of a certain characteristic local feature

Generic Object Recognition Using Bag of Visual Words (BoVW) (2/2)

1. Visual word extraction: Organise local features into groups of similar features

➡ *The center of each group becomes a visual word*

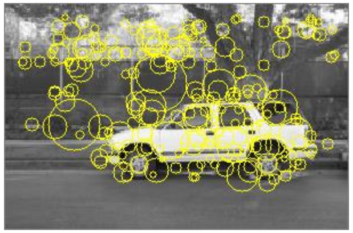
More than 100,000 local features are organised into more than 1,000 groups. In other words, more than 1,000 visual words are extracted



2. BoVW representation: Assign local features in each image to the most similar visual words

➡ For each visual words, the number of assigned local features becomes the value of a bin

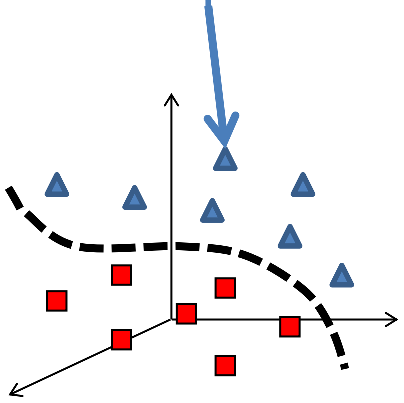
The number of dimensions of a histogram (vector) is more than 1,000.



3. Classification: Extract the boundary between images where a certain object is shown and images where it is absent

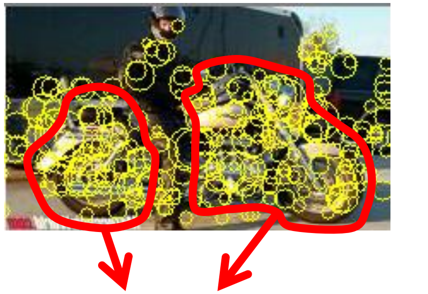
➡ One image represented by BoVW is represented as a point in the high-dimensional vector space

Because of the high-dimensionality, simple similarity measures (e.g., Euclidian distance and cosine distance) do not work. Support Vector Machine (SVM) or other effective classifiers for high-dimensional data must be used.
I spent more than one year to find this point ☹

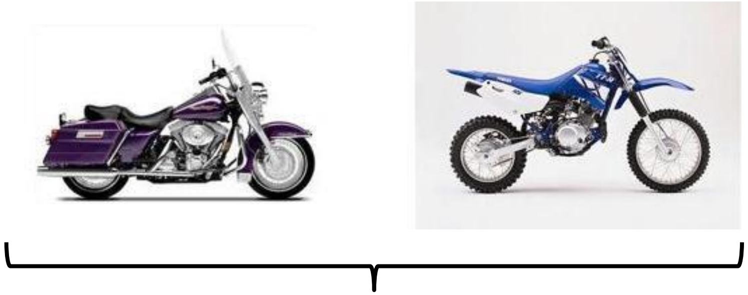


Concepts of BoVW

1. Robust to occlusion and deformation: Although a part of an object is occluded or deformed, as long as some visual features that characterise the object are contained in an image, appropriate recognition can be performed.



Although a part of the bike is invisible due to the person, visual words characterising bikes should be extracted from these regions.



Even if shapes of these bikes are significantly different, visual words characterinsig an engine, wheels and handle should be extracted from both of them.

Although visual words are extracted from the background of an image, most of their effects can be removed using *a large number of visual words*. In other words, as long as the background does not contain many local features that are very very similar to some parts of an object, recognition works correctly.

2. BoVW does not consider the overall shape of an object: In my experience, this kind of precise approach does not work for real-world images with cluttered backgrounds. Rather than this, a simple approach works much better.

3. Local features (SIFT or SURF) are already robust to scaling, rotation, viewpoint changes and illumination

Short Break (My Personal Perspective)

Machine learning is the discipline to classify examples

NOTE: Each example is represented as a feature
In this context, one image is an example, a feature is the BoVW representation of the image (Don’t confuse local features with BoVW)

1. Similarity search: Search the most similar example in the database to a given example

2. Unsupervised learning (Clustering): Organise examples into groups each of which contains similar examples

⇨ Visual word extraction

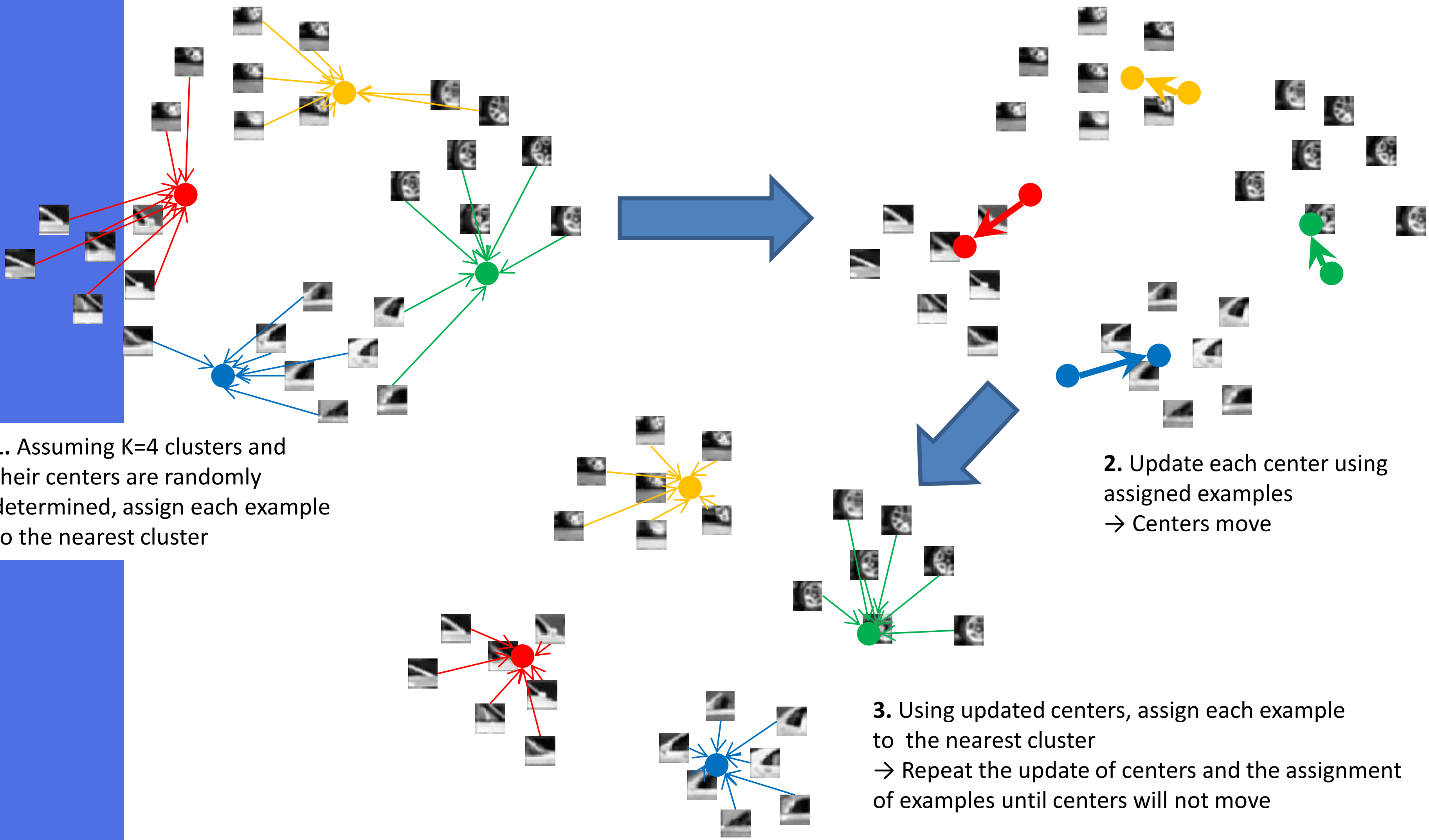
3. Supervised learning (Classification): Given examples with labels, extract a model which can distinguish examples with different labels

Simply speaking, supervised learning finds values in a feature, where examples with a certain label have these values, and examples with other labels do not have them

⇨ Image classification

Clustering of SURF Features

K-means clustering: One of the most famous and practical clustering method
Alternatively repeat updating means of K clusters, and assigning each example (**in this case, SURF feature**) to the nearest cluster among K clusters



1. Assuming K=4 clusters and their centers are randomly determined, assign each example to the nearest cluster

2. Update each center using assigned examples
→ Centers move

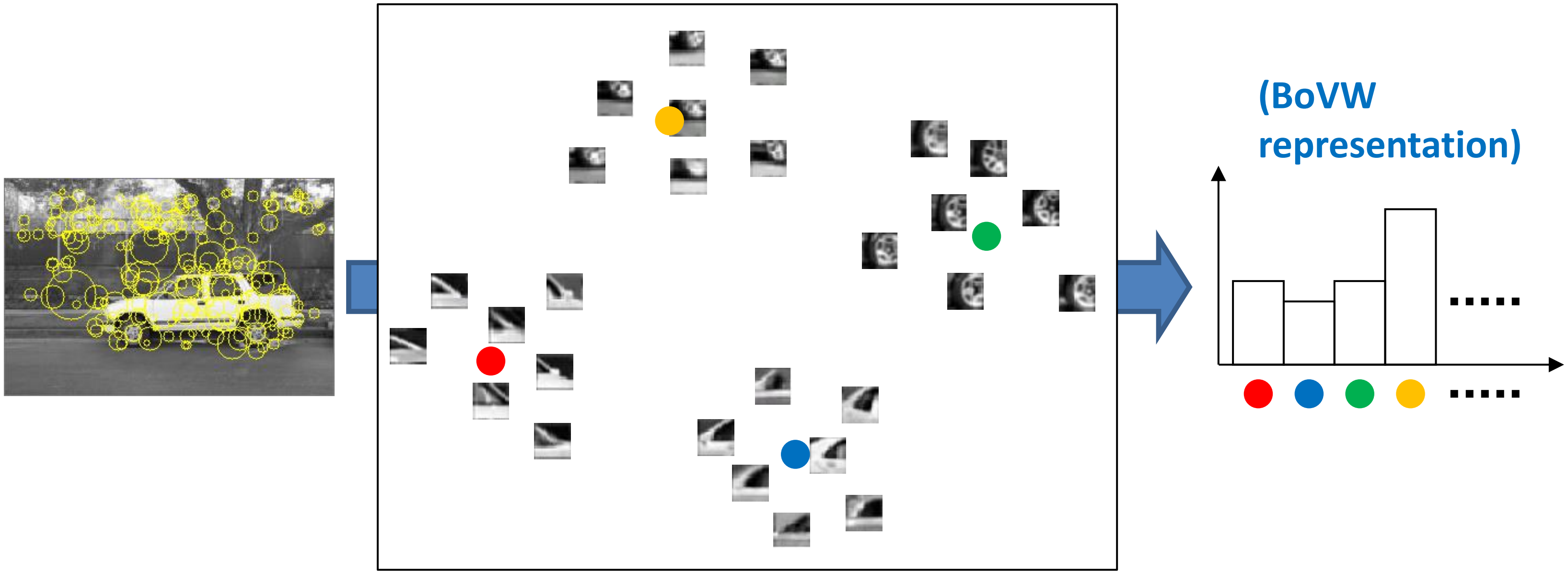
3. Using updated centers, assign each example to the nearest cluster
→ Repeat the update of centers and the assignment of examples until centers will not move

BoVW Extraction

1. Assign each local features in an image to the nearest cluster (i.e., **visual word**)

2. For each visual word, count the number of local features

I think there is no problem in this process



Support Vector Machine on BoVWs (1/2)
- My Personal Interpretation of SVM -

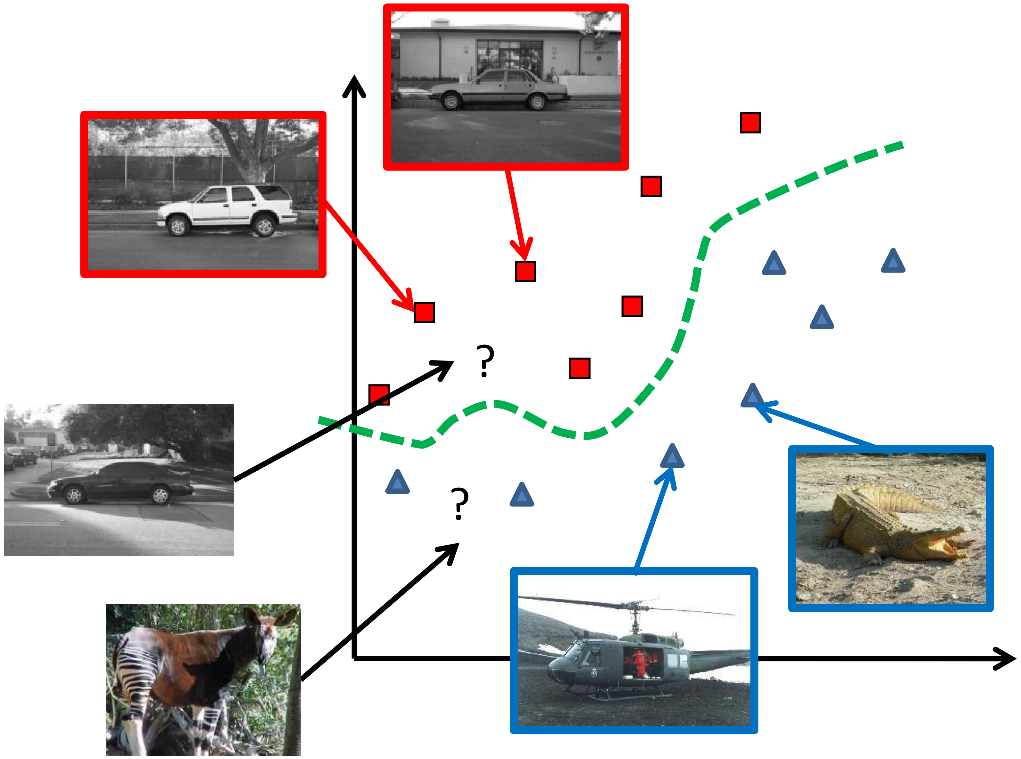
Problem: Given training examples, predict the label of a test example

In our case, training examples consists of

- **Positive examples** where a certain object (e.g., Car) is shown
- **Negative examples** where other objects (e.g., objects other than Car) is shown

For test examples, we don't know if the object is shown or not

NOTE: Examples mean images represented by BoVW



Support Vector Machine on BoVWs (2/2)
- My Personal Interpretation of SVM -

SVM extracts a useful similarity measure which combines weighted similarities of a test example to training examples

(Decision function of SVM)

$$h(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i) - b$$

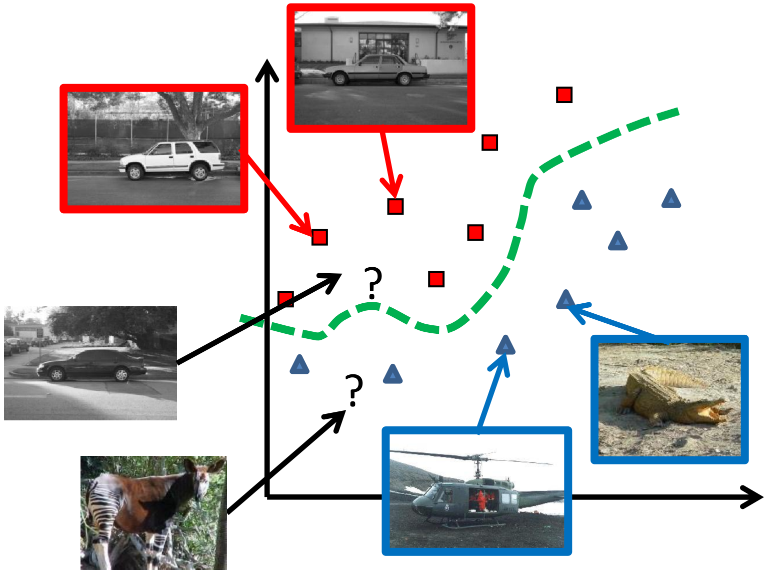
Weight

The optimal set of weight is computed using training examples

Kernel function (similarity between *x**i* and *x*)

$$K(x, x_i) = \exp\left(- \frac{\| x - x_i \|^2}{\gamma} \right)$$

Euclidian distance



■ If *x* is similar to positive examples, $+\alpha_i K(x_i,x)$

□ If *x* is similar to negative examples, $-\alpha_i K(x_i,x)$

If the combined similarity is larger than 0, the object (Car) is regarded to be shown in *x*,
otherwise, it is not shown in *x*

K-means clustering of SURF Features by OpenCV

Apply K-means clustering to about 300,000 SURF features, stored in “surf_features.txt” (created in the 8-th lesson). Group them into 1,000 clusters (visual words), and output obtained visual words into a text file.

```
double kmeans(const Mat& examples, int clusterCount, Mat& labels,
                    TermCriteria termcrit, int attempts, int flags, Mat* centers)
```

- examples: Floating-point matrix of examples, one row per example **(i.e., an example is a SURF feature)**
- clusterCount: The number of clusters **(i.e., number of visual words)**
- labels: Cluster indices of examples (we don’t use this, just feed an empty Mat)
- Termcrit: Specifies maximum number of iterations and/or distance that centers can move between subsequent iterations) **(I only set the maximum number of iterations to “1,000”)**
- Attempts: How many times the algorithm is executed using different initial labelings **(I set it to “1”)**
- Flags: How to set initial cluster centers **(I used KMEANS_PP_CENTERS)**
- Centers: The output matrix of cluster centers, one row per each cluster center **(Output the content of this matrix into a text file) NOTE: I think “Mat* centers” is wrong, “Mat& centers” is right**

For more detail, please refer to http://opencv.jp/opencv-2svn_org/cpp/core_clustering.html?highlight=kmeans

cv::Mat: Structure (class) for dealing with a matrix in OpenCV

- Initialisation: cv::Mat mat = cv::Mat(int rows, int cols, int type) **(type should be “CV_32F”)**
- To obtain the number of rows or columns, you can use “mat.row” or “mat.col”
- To obtain the pointer to the i-th row, you can use “float* ptr = mat.ptr<float>(i);” (after that, use “ptr” just like a normal array)
- To access the element at (i,j) position, you can use “vws.at<float>(i,j)”

For more detail, please refer to http://opencv.jp/opencv-2svn_org/cpp/core_basic_structures.html#Mat

Example Code

```
Mat surfs;
loadSURFFeatures(SURF_FILE, surfs); // Implement this function by yourself

int cluster_num = 1000;
Mat cluster_ids;
TermCriteria term_cri = TermCriteria(TermCriteria::MAX_ITER, 1000, 0);
int rep_num = 1;
Mat centers;

double tt = (double)cvGetTickCount();
cout << "### k-means clustering for extracting visual words ###" << endl;
kmeans(surfs, cluster_num, cluster_ids, term_cri, rep_num, KMEANS_PP_CENTERS, centers);
tt = (double)cvGetTickCount() - tt;
cout << ">> Time required for k-means clustering = " << ( tt / (cvGetTickFrequency()*1000.0*1000.0) ) << " sec" << endl;

outputVisualWords(VW_FILE, centers, cluster_num);  // Implement this function by yourself
```

In my case, clustering 297,660 SURF features into 1,000 clusters, took 2,014 seconds. The CPU of my laptop is i7. If your laptop's CPU is i3 or i5, much more time may be needed. I think the best way is to implement a clustering code in this lesson and run the code before you will sleep ☺