Object recognition and computer vision 2015

Ivan Laptev, Jean Ponce, Cordelia Schmid and Josef Sivic

Assignment 2: Image classification

(adapted from A. Zisserman and A. Vedaldi)

Goal

In image classification, an image is classified according to its visual content. For example, does it contain an airplane or not. An important application is *image retrieval* - searching through an image dataset to obtain (or retrieve) those images with particular visual content.

The goal of this exercise is to get basic practical experience with image classification. It includes: (i) training a visual classifier for five different image classes (*aeroplanes, motorbikes, people, horses* and *cars*); (ii) assessing the performance of the classifier by computing a precision-recall curve; (iii) varying the visual representation used for the feature vector, and the feature map used for the classifier; and (iv) obtaining training data for new classifiers using Google / Bing image search.

Getting started

- Download the **code and data**~914MB (**code only** ~12MB**, data only** ~902MB).
- Unpack the code archive. This will make a directory called **practical-category-recognition-2015a**.
- Unpack the data archive into **practical-category-recognition-2015a/data/**. This should create several sub-directories in the data directory such as **practical-category-recognition-2015a/data/images/**

- Start your MATLAB in the directory **practical-category-recognition-2015a**.
- Try running **setup.m** command (type **setup** without the .m suffix). If all goes well, you should obtain a greeting message. Note: this exercise is using several functions from the open source Matlab package vlfeat by A. Vedaldi.

As you progress in the exercises you can use MATLAB **help** command to display the help of the MATLAB functions that you need to use. For example, try typing **help setup**.

Exercise description

The image-classification directory contains Matlab scripts **exercise1.m** and **exercise2.m,** which contain a commented skeleton of the code. You will need to (1) complete these scripts by writing your code, and (2) produce a written report containing answers to questions given below (**shown in red**). Your answers should include results, graphs or images as specified in each question. Start by opening the script **excercise1.m.** You can modify this script and add your code to it. Follow the steps below:

Part 1: Training and testing an Image Classifier

Stage A: Data preparation and feature extraction

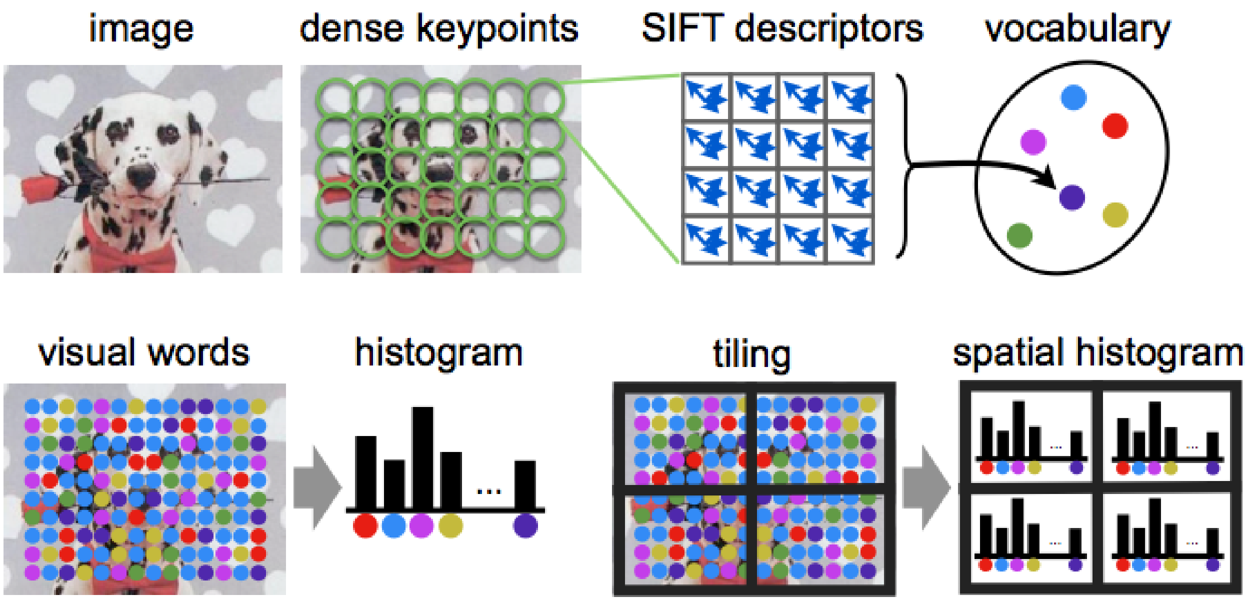
The data provided in the directory data consists of images and pre-computed descriptors for each image. The JPEG images are contained in **data/images**. The data consists of three image classes (containing *aeroplanes*, *motorbikes* or *persons*) and`background' images (i.e. images that do not contain these three classes). This data is divided as:

	aeroplane	motorbike	person	background
Training	112	120	1025	1019
Test	126	125	983	1077
Total	238	245	2008	2096

The list of **training images** for each class is given in text file “data/*_train.txt”, where * is the class name. The list of **testing images** for each class is given in text file “data/*_val.txt”, where * is the class name.

A descriptor for an image is obtained as a statistics of local image features, which in turn capture the appearance of a large number of local image regions. Mapping local features to a single descriptor vector is often regarded as an encoding step, and the resulting descriptor is sometimes called a code. Compared to sets of local features, a main benefit of working with codes is that codes can be compared by simple vectorial metrics such as the Euclidean distance. For the same reason, they are also much easier to use in learning an image classifier.

In this part we will use the Bag of Visual Words (BoVW) encoding. The process of constructing a BoVW descriptor starting from an image is summarized next:



First, SIFT features are computed on a regular grid across the image (”dense SIFT”) and vector quantized into visual words. The frequency of each visual word is then recorded in a histogram for each tile of a spatial tiling as shown in the figure above. The final feature vector for the image is a concatenation of these histograms.

QA1: Why is the spatial tiling used in the histogram image representation?

Stage B: Train a classifier for images containing aeroplanes

We will start by training a classifier for images that contain aeroplanes. The files **data/aeroplane_train.txt** and **data/aeroplane_val.txt** list images that contain aeroplanes. Look through example images of the aeroplane class and the background images by browsing the image files in the **data/images** directory.

The aeroplane training images will be used as the positives, and the background images as the negatives. The classifier is a linear Support Vector Machine (SVM). Train the classifier using the function **trainLinearSVM** by following the steps in **excercise1.m**.

We will first assess qualitatively how well the classifier works by using it to rank all the **training** images. View the ranked list using the provided function **displayRankedImageList** as shown in **excercise1.m**.

QB1: Show the ranked images in your report.

Use function **displayRelevantVisualWords** to display the image patches that correspond to the visual words which the classifier thinks are most related to the class (see the example embedded in **excercise1.m**).

QB2: In your report, show relevant patches for the three most relevant visual words (in three separate figures) for the top ranked image. Are the most relevant visual words on the airplane or also appear on background?

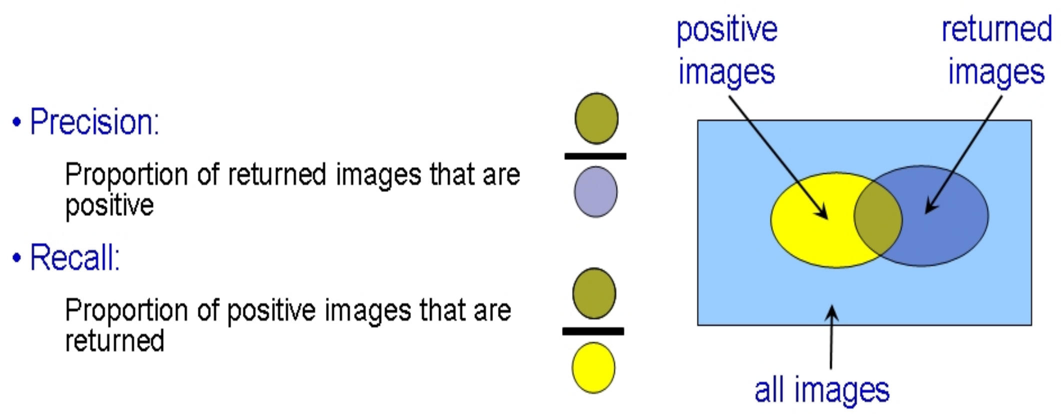
Stage C: Classify the test images and assess the performance

Now apply the learnt classifier to the test images. Again, you can look at the qualitative performance by using the classifier score to rank all the test images. Note the bias term is not needed for this ranking, only the classification vector **w.**

QC1: Why is the bias term not needed for the image ranking?

Now we will measure the retrieval performance quantitatively by computing a

Precision-Recall curve. Recall the definitions of Precision and Recall:


• Precision:
Proportion of returned images that are positive
• Recall:
Proportion of positive images that are returned
positive images
returned images
all images

The Precision-Recall curve is computed by varying the threshold on the classifier (from high to low) and plotting the values of precision against recall for each threshold value. In order to assess the retrieval performance by a single number (rather than a curve), the Average Precision (AP, the area under the curve) is often computed. Make sure you understand how the precision values in the Precision-Recall curve correspond to the ranking of the positives and negatives in the retrieved results

Stage D: Learn a classifier for the other classes and assess its performance

Now repeat stages (**B**) and (**C**) for each of the other two classes: motorbikes and persons. Re-use your code after changing the dataset loaded at the beginning of stage (**B**). Remember to change both the training and test data.

QD1: In your report, show the top ranked images, precision-recall curves and APs for the test data of all the three classes (aeroplanes, motorbikes, and persons). Does the AP performance for the different classes match your expectations based on the variation of the class images?

QD2: For the motorbike class, give the rank of the first false positive image. What point on the precision-recall curve corresponds to this first false positive image? Give in your report the value of precision and recall for that point on the precision-recall curve.

Stage E: Vary the image representation

Benefits of spatial tiling. Up to this point, the image feature vector has used spatial tiling. Now, we are going to`turn this off' and see how the performance changes. In this part, the image will simply be represented by a single histogram recording the frequency of visual words (but not taking any account of their image position). This is a **bag-of-visual-words** representation.

A spatial histogram can be converted back to a simple histogram by merging the tiles. To achieve that use the provided function **removeSpatialInformation**. Then evaluate the classifier performance on the test images. Make sure you re-train the classifier after removing the spatial information.

QE1: Include in your report precision recall-curves and APs, and compare the test performance to the spatially tiled representation in stage D. How is the performance changing? Why?

Benefits of descriptor normalization. An important practical aspect of image descriptors is the one of their normalization. For example, BoVW descriptors are histograms, and regarding them as discrete probability distributions it would seem natural that their elements should sum to 1. This is the same as normalizing the BoVW descriptor vectors in L1 norm. However, in

exercise1.m L2 normalization (sum of squares) is used instead.

QE2: Modify exercise1.m to use L1 normalization and no normalization and measure the performance change.

A linear SVM can be thought of as using a linear kernel

$$K(\mathbf{h},\mathbf{h}') = \sum_{i=1}^{d} h_i h_i'$$

to measure the similarity between pair of objects h and h′ (in this case pairs of BoVW descriptors).

QE3: What can you say about the self-similarity, K(h,h), of a BoVW histogram h that is L2 normalized? **Hint:** Compare K(h,h) to the similarity, K(h,h′), of two different L2 normalized BoVW histograms h and h′.
Can you say the same for unnormalized or L1 normalized histograms?

QE4: Do you see a relation between the classification performance and L2 normalization?

A useful rule of thumb is that better performance is obtained if the vectors that are ultimately fed to a linear SVM (after any intermediate processing) are L2 normalized.

Stage F: Vary the classifier

Up to this point we have used a linear SVM, treating the histograms representing each image as vectors normalized to a unit Euclidean norm. Now we will use a Hellinger kernel classifier but instead of computing kernel values we will explicitly compute the feature map, so that the classifier remains linear (in the new feature space). The definition of the Hellinger kernel (also known as the Bhattacharyya coefficient) is

$$k(\mathbf{h},\mathbf{h}') = \sum_i \sqrt{h(i)h'(i)} \text{ (compared to a linear kernel } \sum_i h(i)h'(i))$$

where h and h' are normalized histograms. Compare this with the expression of the linear kernel given above: all that is involved in computing the feature map is taking the square root of the histogram values.

QF1: Based on the rule of thumb introduced above, how should the BoVW histograms h and h′ be normalized? Should you apply this normalization before or after taking the square root?

- Edit **exercise1.m** so that the square root of the histograms are used for the feature vectors. Make sure that the proper normalization is used. **Hint:** In practice this involves writing one line of MATLAB code for the training and one for the test histograms. Use the Matlab function **sqrt,** which takes an element wise square root of a vector/matrix.

- Retrain the classifier for the aeroplane class, and measure its performance on the test data.
- Make sure you understand why this procedure is equivalent to using the Hellinger kernel.

QF2: Why is this procedure equivalent to using the Hellinger kernel in the SVM classifier?

QF3: Why is it an advantage to keep the classifier linear, rather than using a non-linear kernel?

QF2: Try the other histogram normalization options and check that your choice yields optimal performance. Summarize your finding in the report (include only mAP results, no need to include the full precision-recall curves).

Note: when learning the SVM, to save training time we are not changing the C parameter. This parameter influences the generalization error and should be learnt on a validation set when the kernel is changed. However, in this case the influence of C is small as can be verified experimentally.

Stage G: Vary the number of training images

Up to this point you have used all the available training images. Now train the classifier with only 10% and 50% of the training data. To achieve this, you can use the **fraction** variable in the provided example code in **excercise1.m**

QG1: Report and compare performance you get with the linear kernel and with the Hellinger kernel for the different classes and proportions of training images (10%, 50% and 100%). You don't have to report the precision-recall curves, just APs are sufficient. Plot the APs for one class into a graph, with AP on the y-axis and the proportion of training images on the x-axis. You can use the matlab function **plot** Plot three curves (one curve for each class) into one figure. Produce two figures, one for the linear kernel and one for the Hellinger kernel. Make sure to properly label axis (use functions **xlabel** and **ylabel**), show each curve in a different color, and have a legend (function **legend**) in each figure. Show the two figures in your report.

QG2: By analyzing the two figures, do you think the performance has `saturated' if all the training images are used, or would adding more training images give an improvement?

Part 2: Training an Image Classifier for Retrieval using Internet image search.

In Part 1 of this practical the training data was provided and all the feature vectors pre-computed. The goal of this second part is to choose the training data yourself in order to optimize the classifier performance. The task is the following: you are given a large corpus of images and asked to retrieve images of a certain class, e.g. those containing a bicycle. You then need to obtain training images, e.g. using Bing Image Search, in order to train a classifier for images containing bicycles and optimize its retrieval performance.

The MATLAB code exercise2.m provides the following functionality: it uses the images in the directory data/myImages and the default negative list data/background_train.txt to train a classifier and rank the test images. To get started, we will train a classifier for horses:

- Use Google/Bing image search with "horses" as the text query (you can also set the photo option on).
- Pick 5 images and drag and drop (save) them into the directory data/myImages. These will provide the positive training examples.
- Run the code **exercise2.m** and view the ranked list of images. Note, since feature vectors must be computed for all the training images, this may take a few moments.
- Now, add in 5 more images and retrain the classifier.

The test data set contains 148 images with horses. Your goal is to train a classifier that can retrieve as many of these as possible in a high ranked position. You can measure your success by counting how many appear in the first 36 images (this performance measure is `precision at rank-36') by visually inspecting the top ranked results. Here are some ways to improve the classifier:

- Add more positive training images.
- Add more positive training images, but choose these to be varied from those you already have

Note: all images are automatically normalized to a standard size, and descriptors are saved for each new image added in the **data/cache** directory.

The test data also contains the category **car**. Train classifiers for it and compare the difficulty of this and the horse class.

QP2.1: For the horse class, report the precision at rank-36 for 5 and 10 training images. Show the training images you used. Did the performance of the classifier improve when 10 images were used?

QP2.2: What is the best performance (measured by precision at rank-36) you were able to achieve for the horse and the car class? How many training images did you use? For each of the two classes, show examples of your training images, show the top ranked 36 images, and report the precision at rank-36. Compare the difficulty of of retrieving horses and cars.

Part 2: Advanced Encoding Methods

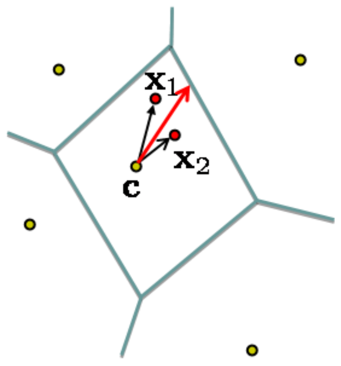
Up to this point we have used a BoVW representation of the original dense SIFT description of the image, together with spatial information coded by a spatial tiling of the image. The BoVW representation is simply a histogram of the number of SIFT vectors assigned to each visual word. The histogram has K bins, where K is the size of the vocabulary created by K-means, and each bin corresponds to a count of the SIFT vectors lying in the Voronoi cell associated with that visual word.

The BoVW is an encoding of the dense SIFTs into a feature vector. One way to think about the encoding is that it aims to best represent the distribution of SIFT vectors of that image (in a manner that is useful for discriminative classification). BoVW is a hard-assignment of SIFTs to visual words, and alternative soft-assignment methods are possible where the count for a SIFT vector is distributed across several histogram bins.

In this section we investigate a different encoding that records more than a simple count as a representation of the distribution of SIFT vectors. In particular we consider two encodings: one that records first moment information (such as the mean) of the distribution assigned to each Voronoi cell, and a second that records both the first and second moment (e.g. the mean and covariance of the distribution). The MATLAB code for computing these encodings is in the file **exercise1.m**.

Stage H: First order methods

The vector of locally aggregated descriptors (VLAD) is an example of a first order encoding. It records the residuals of vectors within each Voronoi cell (i.e. the difference between the SIFT vectors and the cluster centre (from the k-means) as illustrated below:



Suppose the SIFT descriptors have dimension D, then the total size of the VLAD vector is K×D (since a D-dimensional residual is recorded for each Voronoi cell). Typically, K is between 16 and 512, and D is 128 or less.

QH1: Compare the dimension of VLAD and BoVW vectors for a given value of K. What should be the relation of the K in VLAD to the K in BoVW in order to obtain descriptors of the same dimension?

QH2: Replace the encoding used in exercise1 with the VLAD encoding, and repeat the classification experiments for the three classes of Part I (Both linear and Hellinger kernel). How do the results compare to the BoVW encoding? Report mAP results in a table. No need to report all precision-recall curves.

Note, the performance improvement obtained simply by changing the feature encoding.

Stage I: Second order methods

The Fisher Vector (FV) is an example of second order encoding. It records both the residuals (as in VLAD) and also the covariance of the SIFTs assigned to each Voronoi cell. Its implementation uses a Gaussian Mixture Model (GMM) (instead of k-means) and consequently SIFTs are softly assigned to mixture components (rather than a hard assignment as in BoVW). Suppose there are k mixture components and the covariance is restricted to a diagonal matrix (i.e. only the variance of each component is recorded) then the total size of the Fisher vector is 2K×d.

Look through the computation of the FV.

QI1: Replace the encoding used in exercise1 with the FV encoding, and repeat the classification experiments for the three classes of Part I. Report the results in the same table as QH2 so that you can see the performance of the three encoding methods side by side.

Note, the performance improvement obtained (over BoVW and VLAD).

QI2: What are the advantages or disadvantages of FV compared to VLAD in terms of computation time and storage/memory footprint - especially for a large number (hundreds of millions) of images.

Further work (optional):

If there is a significant difference between the training and test performance, then that indicates overfitting. The difference can often be reduced, and the test performance (generalization) improved by changing the SVM C parameter. In Part I, vary the C parameter in the range 0.1 to 1000 (the default is C=100), and plot the AP on the training and test data as C varies for the linear and Hellinger kernels.

Additional Links:

- The code for this assignment is written using the software package **VLFeat.** This is a software library written in MATLAB and C, and is freely available as source code and binary, see http://www.vlfeat.org/.
- The images for this practical are taken from the **PASCAL VOC 2007** benchmark, see http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2007/
- For a tutorial on large scale image classification and references to the literature, see here.

What to hand-in:

Hand-in a written report containing brief answers to the above questions (**shown in red**). Include the label of the question, e.g. "QA1" at the beginning of each answer. Your answers should include results, graphs or images as requested in each question.

Instructions for formatting and handing-in assignments:

● At the top of the first page of your report include (i) your name, (ii) date, (iii) the assignment number and (iv) the assignment title.

● The report should be a single pdf file and should be named using the following format: A#_lastname_firstname.pdf, where you replace # with the assignment number and "firstname" and "lastname" with your name, e.g. **A2_SIVIC_Josef.pdf**.

● Zip your code and any additional files (e.g. additional images) into a single zip file using the same naming convention as above, e.g. **A2_SIVIC_Josef.zip**. We do not intend to run your code, but we may look at it and try to run it if your results look wrong.

Send your report (a single pdf file) and the zipped code (excluding the data) in two separate files to **Josef Sivic <Josef.Sivic@ens.fr>**.