Lab 8

Basic Image Processing Algorithms
Fall 2016

Lab 8: Classifying CIFAR images on the basis of HOG-descriptors, with nonlinear SVM

- deadline (if you do not finish until the end of the lab): **23:59, 16/11/2016**;
  please send it to: bipa2016fall@gmail.com
  with the subject: LAB8
  please **send only your script and your function**, nothing else

Support Vector Machines

- Cost function of SVM:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\text{cost}_1\left(\theta^T x^{(i)}\right) + \left(1-y^{(i)}\right)\text{cost}_0\left(\theta^T x^{(i)}\right)\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$J(\theta) = C\sum_{i=1}^{m}\left(y^{(i)}\text{cost}_1\left(\theta^T x^{(i)}\right) + \left(1-y^{(i)}\right)\text{cost}_0\left(\theta^T x^{(i)}\right)\right) + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

Rest of the notation:
x: input data / feature vector
y: labels
h: hypothesis function, it maps x to y
Θ: mapping parameters of h
J: cost function (how big is the difference between the mapped data and the real label)
cost_0 and **cost_1:** cost terms, they give "prices" to the different output-scenarios of the hypothesis function

- Hypothesis function of SVM:

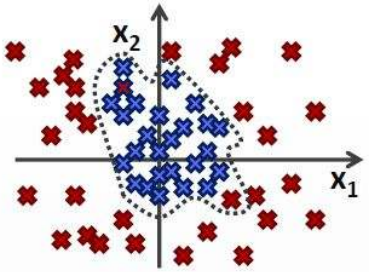
$$h_\theta(x) = \begin{cases} 1, & \text{if } \theta^T x^{(i)} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Although the decision threshold is at 0, at training we want our positive samples to be >1, and our negative samples to be <-1.

- The **C** parameter controls the trade-off between two goals:
  - Fitting the data well (high value for C)
  - Keeping the parameters low, to avoid overfitting (low value for C)

Support Vector Machines

- Using Kernels:
  - One way to define a complex non-linear decision boundary is by the use of high order terms:



$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2^{\ 3} + \ldots$$

  - If we change the notation a bit:

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \ldots$$

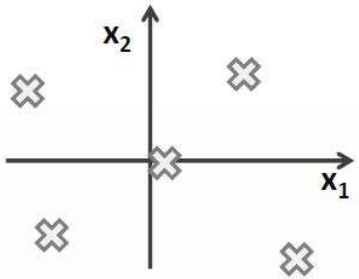
where

$$f_1 = x_1$$
$$f_2 = x_2$$
$$f_3 = x_1 x_2^3$$
$$\ldots$$

  - What else can we use as *f*?

Support Vector Machines

- Using Kernels:
  - What else can we use as *f*?
  - Distance from landmark points: $l^{(1)}$, $l^{(2)}$, $l^{(3)}$,...



$$f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$f_2 = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$$

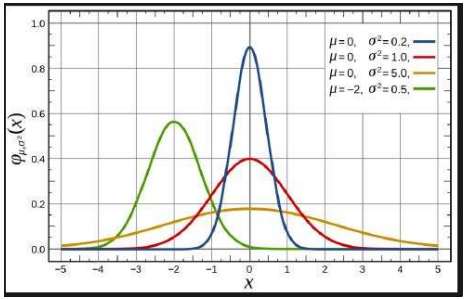
$$f_3 = \exp\left(-\frac{\|x - l^{(3)}\|^2}{2\sigma^2}\right)$$

...

Gaussian kernel

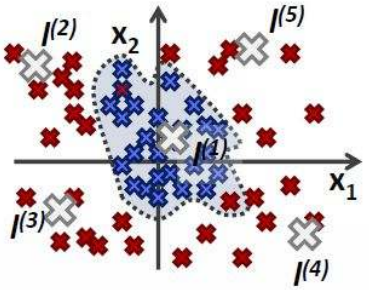
  - Where these *f* kernel functions measure the similarity between the point *x* and the landmark $l^{(i)}$:
    - If *x* is far from $l^{(i)}$ then $f_i \approx 0$.
    - If *x* is close to $l^{(i)}$ ($x \approx l^{(i)}$) then $f_i \approx 1$.

- For the Gaussian kernel the bandwidth parameter **σ**:
  - High σ, results a slowly changing Gaussian, which can cause high bias.
  - Low σ, results a more rapidly changing Gaussian, which can cause high variance.



Support Vector Machines

- Using Kernels:



Hypothesis function:

$$h_\theta(x) = \begin{cases} 1, \text{ if } \theta^T f = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \theta_4 f_4 + \theta_5 f_5 \geq 0 \\ 0, \text{ otherwise} \end{cases}$$

With parameters:

$$\theta_0 = -0.5, \theta_1 = 2, \theta_2 = -0.5,$$

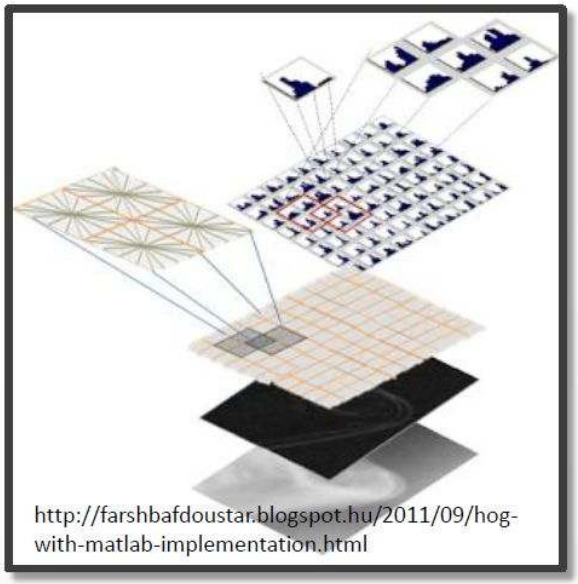
$$\theta_3 = -1.5, \theta_4 = -1, \theta_5 = -0.5$$

- How do we get the landmarks?
  - We place a landmark at the position of each training example.
  - The decision is made based on how close/similar the test samples are to the positive and negative training samples.
  - The feature vector ***x*** which represented a sample is now replaced a new feature vector ***f***, which contains the similarities to the training samples.

HOG: Histogram of Oriented Gradients

- Originally developed for pedestrian detection by N. Dalal, B. Triggs in 2005.

- Steps of the Algorithm:
  1. Gradient Computation
  2. Orientation Binning
  3. Block Description
  4. Block Normalization
  5. Classification



http://farshbafdoustar.blogspot.hu/2011/09/hog-with-matlab-implementation.html

N. Dalal,B. Triggs, *„Histograms of Oriented Gradients for Human Detection"* In Proceedings of IEEE Conference *Computer Vision and Pattern Recognition*, San Diego, USA, pages 886-893, June 2005.

HOG: Histogram of Oriented Gradients

- I. Gradient Computation:
  - Many gradient detector was tested (Sobel, Prewitt, ..)
  - The simple [-1 0 1] and $[-1\ 0\ 1]^T$ gradient detectors gave the best result.

- II. Orientation Binning for a cell:
  - A ***cell*** is rectangular (or circular) shaped, 8x8 window.
  - Histogram of gradient orientations is calculated over the cell.
  - Each pixel votes based on its magnitude on the gradient image.
  - 9 bin histogram: 0-180°

N. Dalal,B. Triggs, *„Histograms of Oriented Gradients for Human Detection"* In Proceedings of IEEE Conference *Computer Vision and Pattern Recognition*, San Diego, USA, pages 886-893, June 2005.

HOG: Histogram of Oriented Gradients

- Rectangular HOG is similar to SIFT, with a few differences:
  - there is no dominant orientation alignment
  - single scale
  - spatial position is coded

◉ III. Block description:

- A block contains 2x2 cells
- Pixels in the block are weighted by a Gaussian window.

◉ IV. Block Normalization:

- The blocks are overlapping, every cell is used 4 times in 4 different blocks. Also there are different versions of the normalization:

L1-norm: $v \rightarrow \frac{v}{\|v\|_1 + \varepsilon}$

L1-sqrt: $v \rightarrow \sqrt{\frac{v}{\|v\|_1 + \varepsilon}}$

L2-norm: $v \rightarrow \frac{v}{\sqrt{\|v\|_2^2 + \varepsilon^2}}$

L2-Hys: max value of v is limited to 0.2

HOG: Histogram of Oriented Gradients

- V. Classification
  - Linear SVM
  - For pedestrian detection it was trained on 64x128 sized samples (positive and negative images).
- HOG inspired methods
  - A fast version of HOG with..
    - variable block size,
    - AdaBoost for feature selection,
    - cascade of rejectors and integral image representation.
    - 70x faster than the original
  - Motion Flow based HOG

Zhu, Q., Yeh, M., Cheng, K., and Avidan, S., „***Fast Human Detection Using a Cascade of Histograms of Oriented Gradients***". In *Proceedings of the 2006 IEEE* Computer Society, CVPR, Washington, DC, 1491-1498.
N. Dalal, B. Triggs, C. Schmid, „***Human Detection Using Oriented Histograms of Flow and Appearance***", In Proceedings of the *European Conference on Computer Vision*, Graz, Austria, May 2006.

Overview

two labs ago: on the basis of the images → training of the SVM model
→ testing of the test set

today: on the basis of the images → features/descriptors → training of the SVM model
→ testing of the test set

Links

CIFAR-10 database:
http://www.cs.utoronto.ca/~kriz/cifar.html
classes: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'

for your convenience, please download this:
http://users.itk.ppke.hu/~kolmi/bipa/cifar10.zip

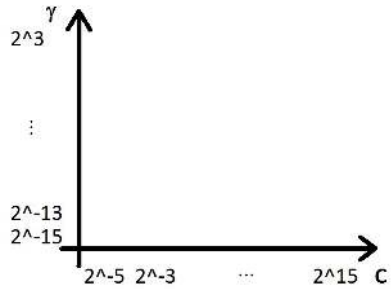
LIBSVM: 2 labs ago you have already downloaded

(http://www.csie.ntu.edu.tw/~cjlin/libsvm/ , if you use MATLAB and Windows, you can use the precompiled MEX-files **`svmtrain.mexw64`** and **`svmpredict.mexw64`** from the subfolder **`libsvm-3.21/windows/`**)

VLFeat library: last time you have already downloaded

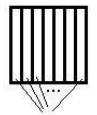
(original source: http://www.vlfeat.org/ but please use this: http://users.itk.ppke.hu/~kolmi/bipa/vlfeat-0.9.20.zip
to add vlfeat to the path: call from command window 'run ./vlfeat-0.9.20/toolbox/vl_setup.m' then save the path in Ribbon->Home->SetPath->SAVE
the order of the subdirs is important, that's why we are not going to add the vlfeat lib manually)

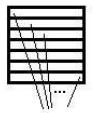
Exercise - create a function and a script



The **function**: a grid-search function

- now we will use RBF kernel, so we have *two parameters* (C, σ) to search for (in LibSVM, symbol σ is referred as γ);
- please do it parallelly with **`parfor`**
  - without parallelism, you would do this double-search with and embedded **`for`**-loop;
  - one way to solve parameter-representation:
    - N.B.: in MATLAB you can index a 2D matrix with one index -- this way it will read your array in column wise order;
    - replace your C and γ one-dimensional vectors with 2D arrays, one of them stacking horizontally its original vector, the other one stacking vertically its original vector (see bottom);
    - the results of every **`svmtrain`** with different C- and γ-value have to be saved in an array, where the index should be the same as the index of the specific C- and γ-value in their arrays;
    - after the **`parfor`**-cycle, you have to search for the max-value in your result-array, leading to the index which will show you the appropriate C- and γ-value.


original C-vectors


original gamma-vectors

Exercise - create a function and a script

The **script**:

1. loads in the database, adds the svm- and vlfeat-libraries to the path
2. trains an SVM model on the basis of images (pixel intensities)
3. tests the previous SVM model on the test images
4. extracts the HOG-features from the train images
5. trains another SVM model on the basis of the prev. extracted HOG-features
6. extracts the HOG-features from the test images
7. tests the the latter SVM model on the features of the test images

These are the steps you have to realize, but it is not mandatory to have the implementation-steps strictly in this order.

Comments to script step #1

- load the databases (**`load`**):
  - *./cifar10/batches.meta.mat:* contains the names of the labels
  - *./cifar10/data_batch_1.mat:* contains the training **`labels`** and samples (**`data`**)
  - *./cifar10/test_batch.mat:* contains the test **`labels`** and samples (**`data`**)
- as an example, the first image of the train-set:

```
tr_set = load('./cifar10/data_batch_1.mat');
img = imrotate(reshape(tr_set.data(1, :),32,32,3), -90);
figure;
imshow(img);
% just to have the text-labels as well, not important:
tr_meta = load('./cifar10/batches.meta.mat');
title(tr_meta.label_names(tr_set.labels(1)+1));
% the label-database starts from index 0, but
% matlab starts indexing from 1, of course
```

Comments to script step #2 & #3

- you have to convert your images to grayscale
- select randomly 650 pixels of the 1024: for this purpose, create an index-vector with **`randperm`**, and apply this “pixel-selecting vector” for every train&test images
- the type of the labels and samples should be **`double`** when you feed your SVM

I took every 10th image from the train set (only batch1), the predicted accuracy on the test set was around 10%.

Comments to script step #4 & #6

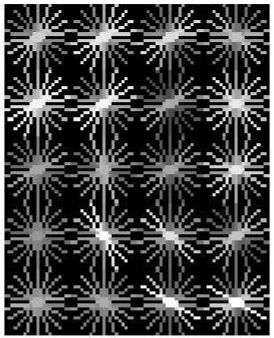
- to extract the HOG-features, you should use **`vl_hog`**
  the manual can be found here: http://www.vlfeat.org/matlab/vl_hog.html
- your images should be single
- compute the HOG-features with different cell-sizes (32, 16, 8), vectorize them and concatenate after each-other: in this way every image should be represented as a vector (with a length of approx. 651)
  - if you are interested in the specific size of the output of **`vl_hog`**, you can find more details on these links: http://www.vlfeat.org/api/hog.html and https://www.cs.berkeley.edu/~rbg/papers/Object-Detection-with-Discriminatively-Trained-Part-Based-Models--Felzenszwalb-Girshick-McAllester-Ramanan.pdf (see Section 6)
  - you can visualize the iconic representation of the HOG-features with the following commands:

```
f1 = vl_hog(im2single(img), 128);
f1_im = vl_hog('render', f1);
```

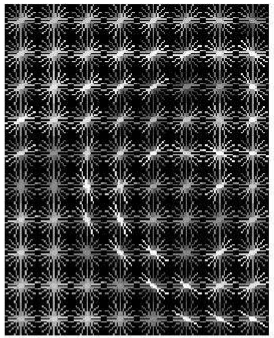
Comments to script step #4 & #6

original image (wxh=524x664)

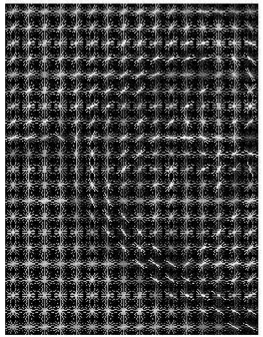
HoG features, cellsize: 128



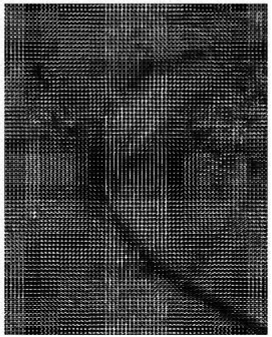
HoG features, cellsize: 64



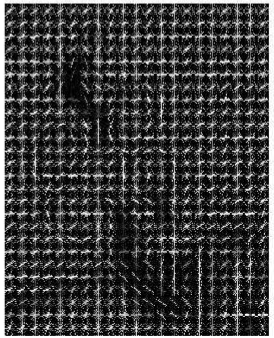
HoG features, cellsize: 32



HoG features, cellsize: 8



cellsize: 8, the guy with the bicycle enlarged



Comments to script step #5 & #7

- the type of the labels and samples should be **`double`** when you feed your SVM

I took every 10th vector from the train set (only batch1), the predicted accuracy on the test set's vector was a bit above 50%.

General steps to use LibSVM
A) Preprocessing

1) preprocessing of categories: *onehot encoding*
   instead of using integers for category-description, use n-tuples of 0s and 1s:

   1, (0, 0, 0, 1)
   2, ---> (0, 0, 1, 0)
   3, (0, 1, 0, 0)
   4 (1, 0, 0, 0)

2) scaling of the attributes: linearly, to the range of [-1, 1] (or to [0, 1])
   and do it consequently (if the first attributes of training data is mapped from region [-10, 10] to [-1, 1], then the first attributes of test data eg. from region [-11, 8] should be mapped to [-1.1, 0.8])

General steps to use LibSVM
B) Model selection

1) choose a kernel:
   a kernel is a similarity measure of learned instances (training examples) and test data
   RBF is the best first try in a new problem
2) cross validation + grid search:
   searching for the necessary parameter **C** (and for the further parameters of the applied kernel, like γ in case of RBF (σ in lecture slides))

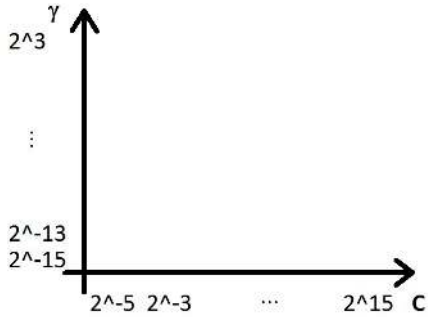
- linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.
- polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i{}^T \mathbf{x}_j + r)^d$, $\gamma > 0$.
- radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$.
- sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i{}^T \mathbf{x}_j + r)$.

General steps to use LibSVM
B) Model selection

v-fold cross validation: helps to prevent overfitting

- divide the learning set into v subsets of equal size,
- sequentially one subset is tested using the classifier trained on the remaining v-1 subsets

grid search: exhaustive search for the necessary parameter(s), preferably on exp-scale



- at every point apply cross-validation (it can be parallelized)
- first on a coarser scale, then on a finer grid
- it we have the best params, train the model again with whole training set

General steps to use LibSVM
C) Test

Apply the final model on your test set :)