# Part I
# *VLFeat*
## An Open and Portable Library of Computer Vision Algorithms

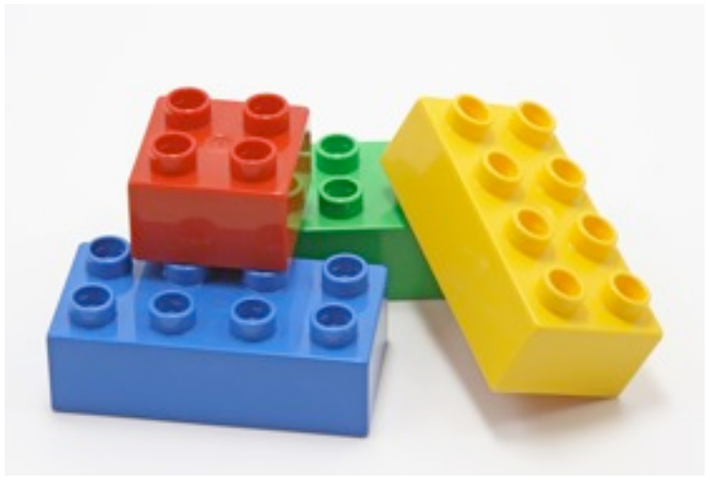
Andrea Vedaldi
*Visual Geometry Group*
Oxford

Brian Fulkerson
*VisonLab*
UCLA



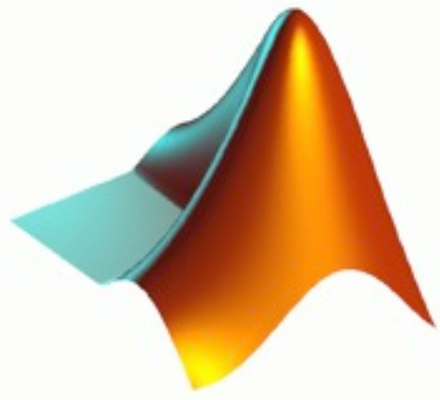
# Plan

- **The VLFeat library**
  - SIFT example (`vl_sift`)
- **Caltech-101 running example**
- **Visual descriptors**
  - PHOW feature (fast dense SIFT, `vl_phow`)
  - Vector Quantization (Elkan, `vl_kmeans, vl_kdtreebuild, vl_kdtreequery`)
  - Spatial histograms (`vl_binsum`, `vl_binsearch`)
- **Learning and classification**
  - Fast linear SVMs
  - PEGASOS (`vl_pegasos`)
  - Fast non-linear SVMs
  - Homogeneous kernel maps (`vl_homkermap`)
- **Other VLFeat features**



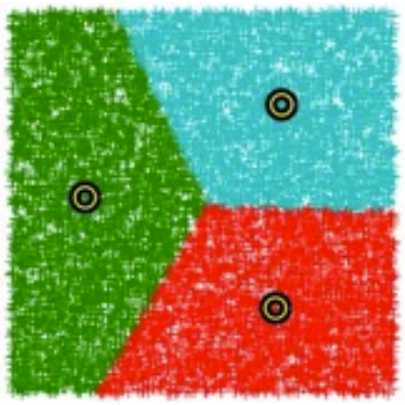
computer vision
building blocks


GPL

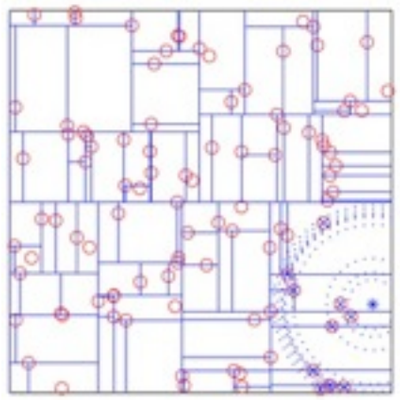


MATLAB

features ,



clustering ,



matching • • •

# Running VLFeat

Mac

Linux

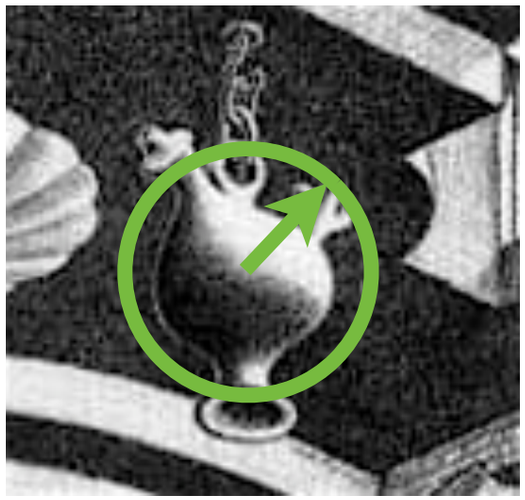
Windows

- VLFeat.org
- Quick start
  1. Get **binary package**
     ```
     wget http://www.vlfeat.org/download/vlfeat-0.9.9-bin.tar.gz
     ```
  2. Unpack
     ```
     tar xzf vlfeat-0.9.9-bin.tar.gz
     ```
  3. Setup MATLAB
     ```
     run('VLROOT/toolbox/vl_setup')
     ```

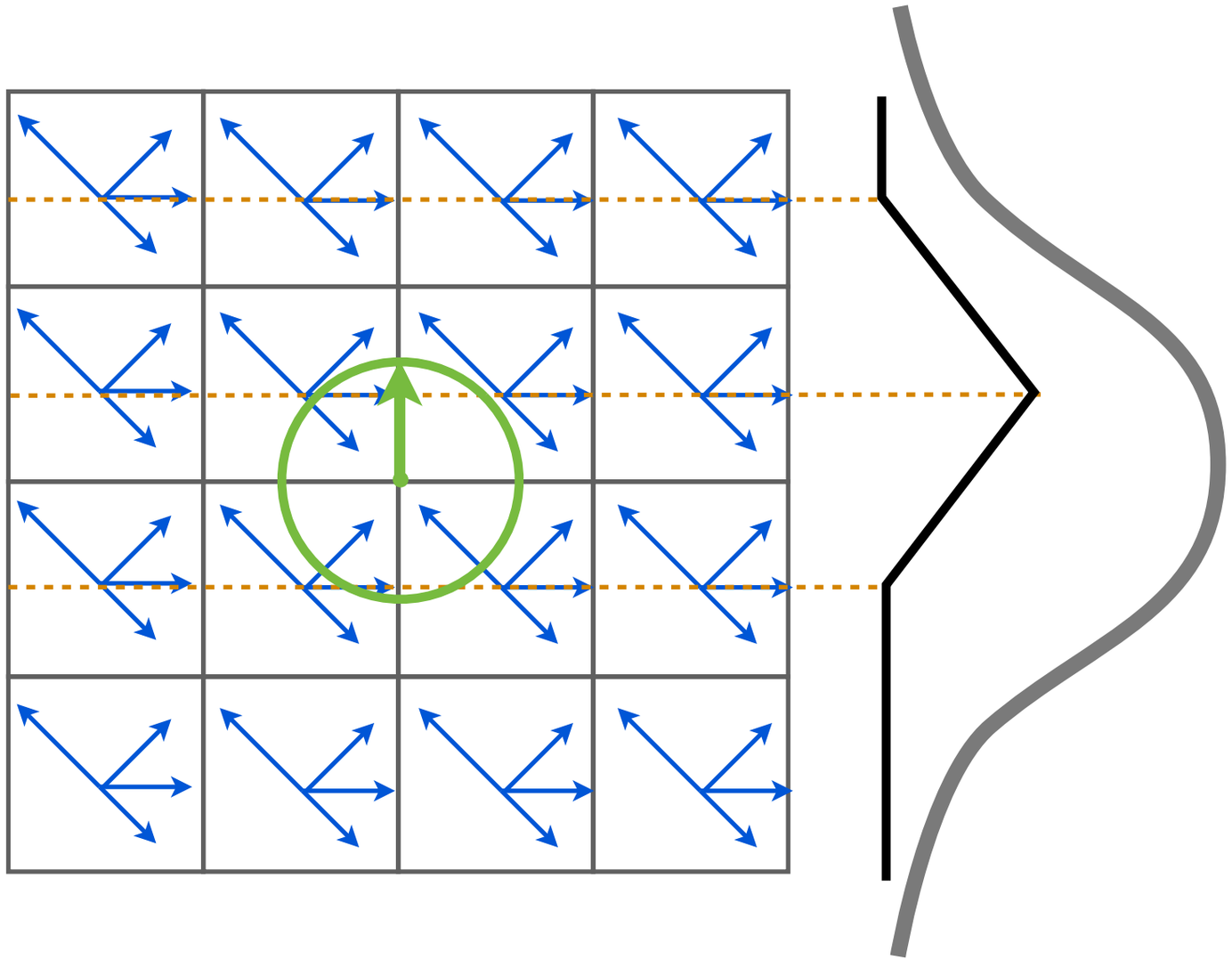
# Demo: SIFT features



- SIFT **keypoint** [Lowe 2004]
  - extrema of the DoG scale space
  - blob-like image structure
  - oriented



- SIFT **descriptor**
  - 4 x 4 spatial histogram of gradient orientations
  - linear interpolation
  - Gaussian weighting



# Demo: Computing SIFT features



- Output-equivalent to D. Lowe's implementation

- Example

1. **load an image**

```
imPath = fullfile(vl_root, 'data', 'a.jpg') ;
im = imread(imPath) ;
```

2. **convert into float array**

```
im = im2single(rgb2gray(im)) ;
```

3. **run SIFT**

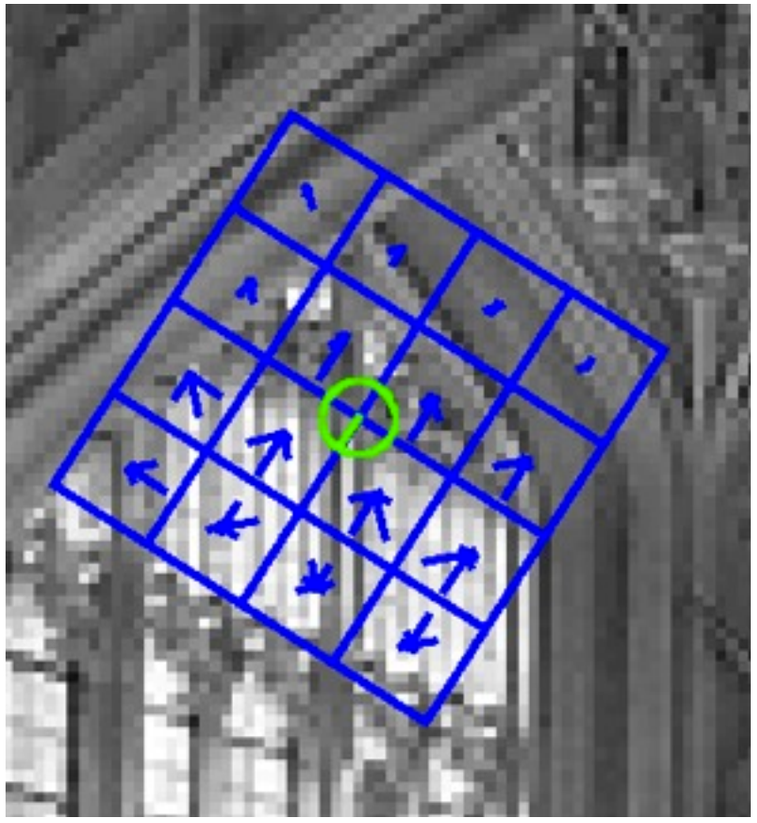
```
[frames, descrs] = vl_sift(im) ;
```

4. **visualize keypoints**

```
imagesc(im) ; colormap gray; hold on ;
vl_plotframe(frames) ;
```

5. **visualize descriptors**

```
vl_plotsiftdescriptor(...
  descrs(:,432), frames(:,432)) ;
```



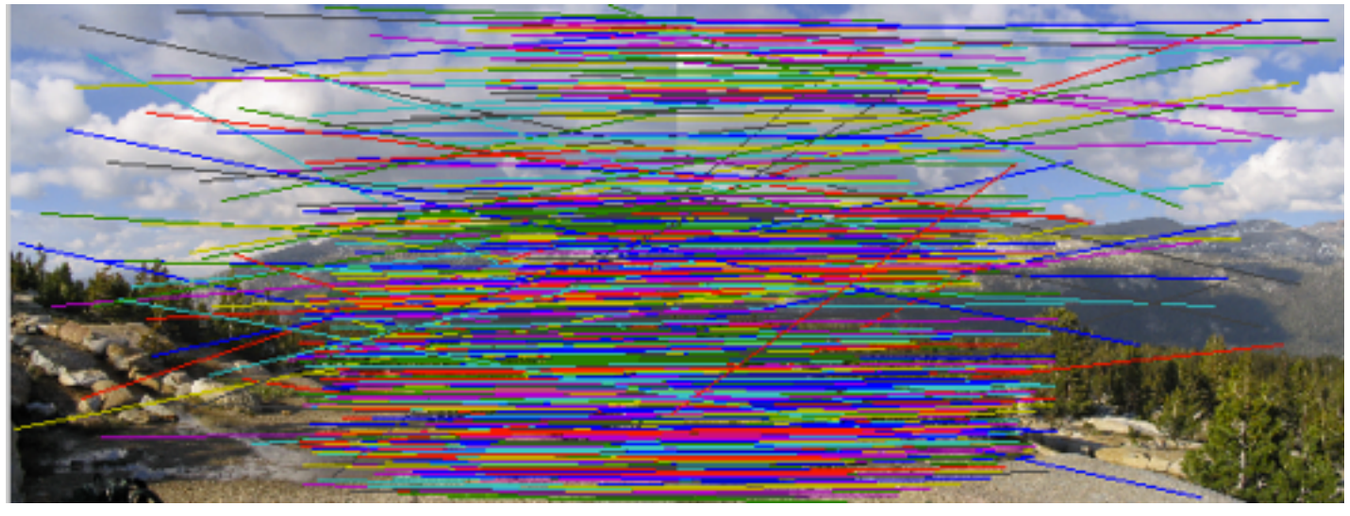
# Demo: Matching & Stitching



- Matching code

```
[f1,d1] = vl_sift(im2single(rgb2gray(im1))) ;
[f2,d2] = vl_sift(im2single(rgb2gray(im2))) ;
[matches, scores] = vl_ubcmatch(d1,d2) ;
```

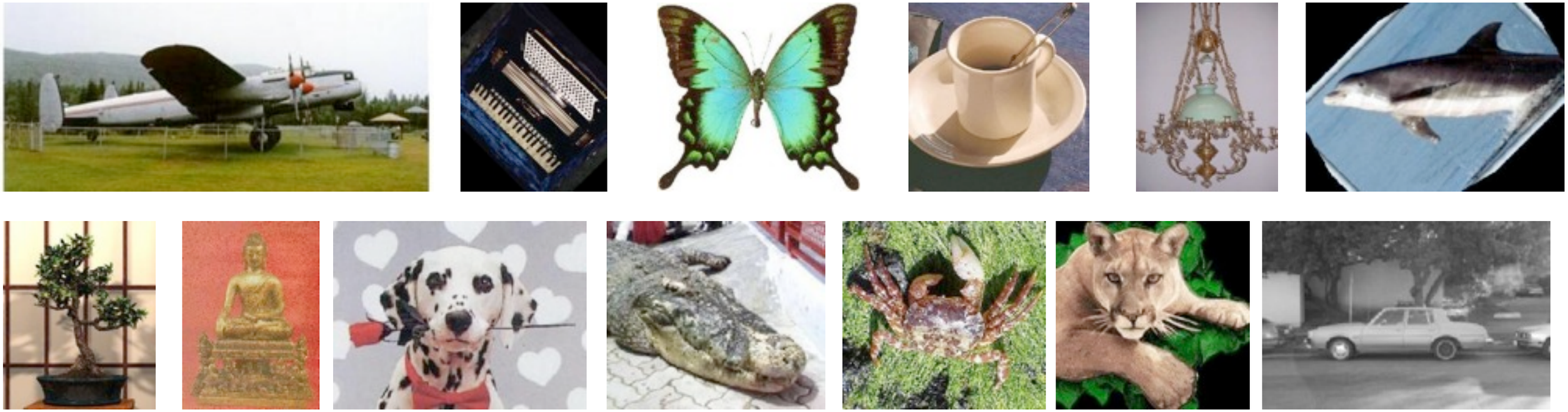
684 tentative matches



# Running example: Caltech-101



102 semantic labels



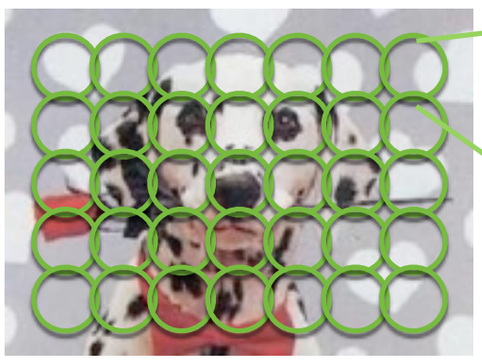
[Fei-Fei et al. 2003]

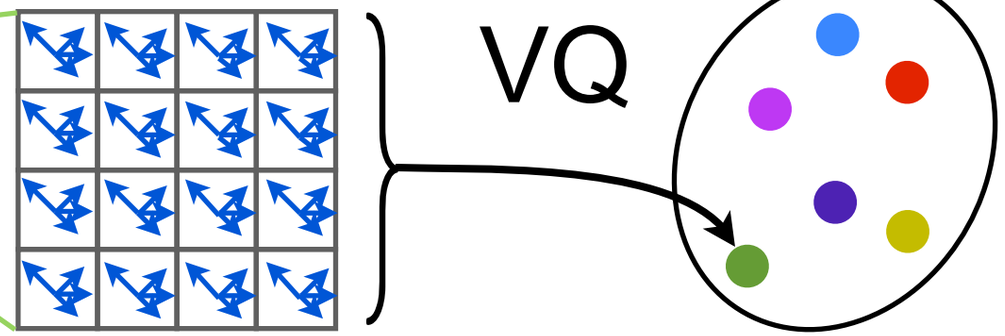
computer vision

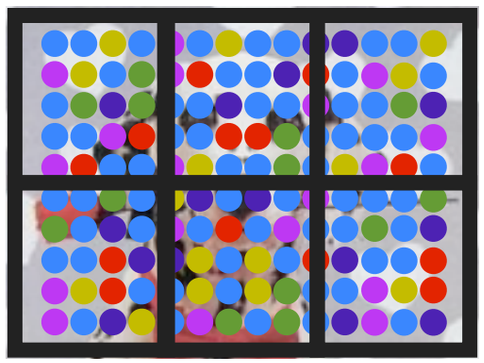
Dalmatian

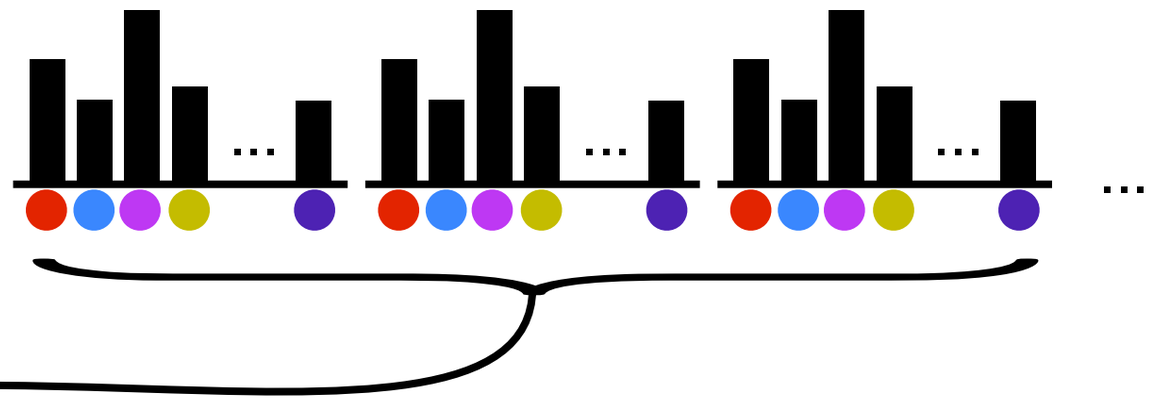
# Running example: System

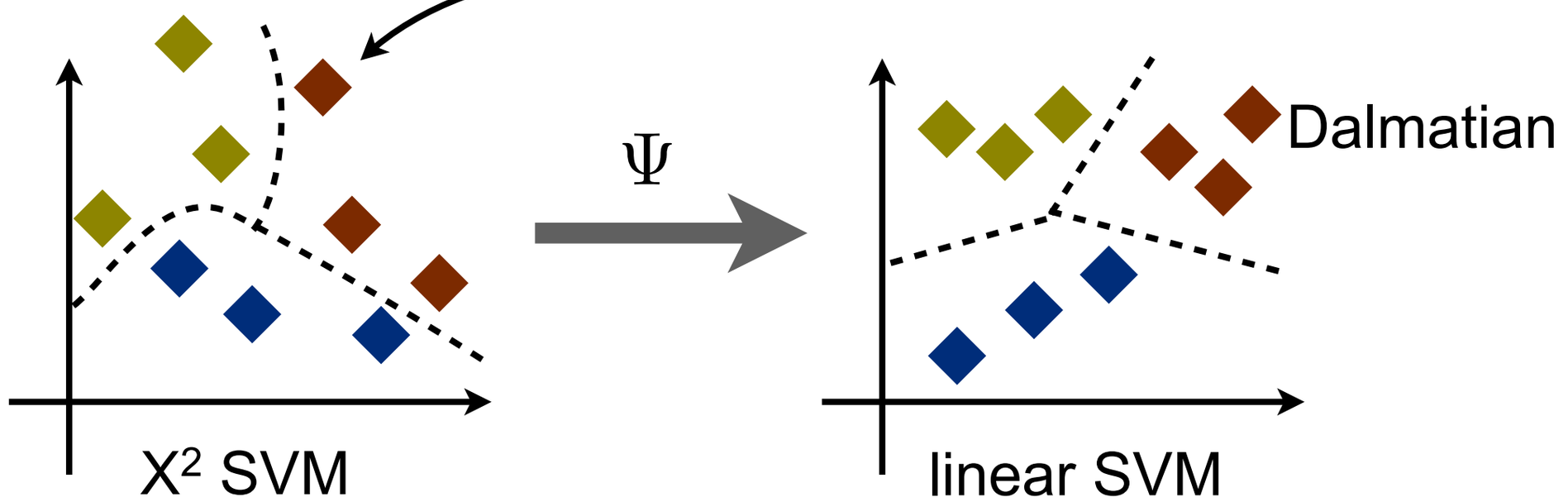






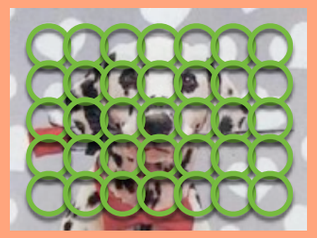


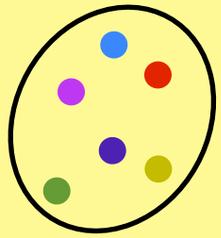
# Running example: Components



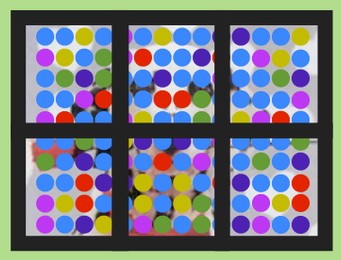
PHOW
(dense SIFT)



Visual Words



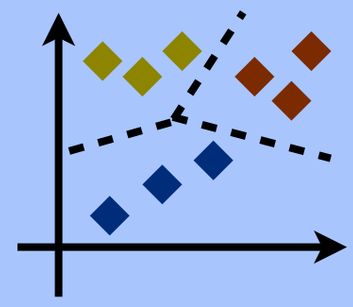
Spatial Histograms



Kernel Map

$\Psi$

SVM



- Complete source code available
  - `phow_caltech101.m`
  - http://www.vlfeat.org/applications/apps.html#apps.caltech-101

# Running example: Complete code



```
function phow_caltech101
% PHOW_CALTECH101 Image classification in the Caltech-101 dataset
%   A quick test should run out of the box by running PHOW_CALTECH101
%   from MATLAB (provied that VLFeat is correctly installed).
%
%   The program automatically downloads the Caltech-101 data from the
%   interned and decompresses it in CONF.CALDIR, which defaults to
%   'data/caltech-101'. Change this path to the desidred location, for
%   instance to point to an existing copy of the data.
%
%   Intermediate files are stored in the directory CONF.DATADIR.
%
%   To run on the entiere dataset, change CONF.TINYPROBLEM to
%   FALSE.
%
%   To run with a different number of training/testing images, change
%   CONF.NUMTRAIN and CONF.NUMTEST. By default 15 training images are
%   used, which should result in about 65% performance (this is quite
%   remarkable for a single descriptor).

% AUTORIGHTS

conf.calDir = 'data/caltech-101' ;
conf.dataDir = 'data/' ;
conf.autoDownloadData = true ;
conf.numTrain = 15 ;
conf.numTest = 15 ;
conf.numClasses = 102 ;
conf.numWords = 800 ;
conf.numSpatialX = 4 ;
conf.numSpatialY = 4 ;
conf.svm.C = 10 ;
conf.svm.solver = 'pegasos' ;
conf.phowOpts = {} ;
conf.clobber = false ;
conf.tinyProblem = false ;
conf.prefix = 'baseline' ;
conf.randSeed = 1 ;

if conf.tinyProblem
  conf.prefix = 'tiny' ;
  conf.numClasses = 5 ;
  conf.numSpatialX = 2 ;
  conf.numSpatialY = 2 ;
  conf.numWords = 300 ;
  conf.phowOpts = {'Verbose', 2, 'Sizes', 7, 'Step', 3, 'Co
true} ;
end

conf.vocabPath = fullfile(conf.dataDir, [conf.prefix '-voca
conf.histPath = fullfile(conf.dataDir, [conf.prefix '-hists
conf.modelPath = fullfile(conf.dataDir, [conf.prefix '-mode
conf.resultPath = fullfile(conf.dataDir, [conf.prefix '-
result.mat']) ;

randn('state',conf.randSeed) ;
rand('state',conf.randSeed) ;
vl_twister('state',conf.randSeed) ;

% --------------------------------------------------------------------
%                                            Download Caltech-101 data
% --------------------------------------------------------------------

if ~exist(conf.calDir, 'dir') || ...
   (~exist(fullfile(conf.calDir, 'airplanes'),'dir') && ...
    ~exist(fullfile(conf.calDir, '101_ObjectCategories',
'airplanes')))
  if ~conf.autoDownloadData
    error(...
      ['Caltech-101 data not found. ' ...
       'Set conf.autoDownloadData=true to download the requried
data.']) ;
  end
  vl_xmkdir(conf.calDir) ;
  calUrl = ['http://www.vision.caltech.edu/Image_Datasets/' ...
    'Caltech101/101_ObjectCategories.tar.gz'] ;
  fprintf('Downloading Caltech-101 data to ''%s''. This will take a
while.', conf.calDir) ;
  untar(calUrl, conf.calDir) ;
end

if ~exist(fullfile(conf.calDir, 'airplanes'),'dir')
  conf.calDir = fullfile(conf.calDir, '101_ObjectCategories') ;
end

% --------------------------------------------------------------------
```

```
%                                                        Setup data
% --------------------------------------------------------------------
classes = dir(conf.calDir) ;
classes = classes([classes.isdir]) ;
classes = {classes(3:conf.numClasses+2-1).name} ;

images = {} ;
imageClass = {} ;
for ci = 1:length(classes)
  ims = dir(fullfile(conf.calDir, classes{ci}, '*.jpg'))' ;
  ims = vl_colsubset(ims, conf.numTrain + conf.numTest) ;
  ims = cellfun(@(x)fullfile(classes{ci},x),
{ims.name},'UniformOutput',false) ;
  images = {images{:}, ims{:}} ;
  imageClass{end+1} = ci * ones(1,length(ims)) ;
end
selTrain = find(mod(0:length(images)-1, conf.numTrain+conf.numTest) <
conf.numTrain) ;
selTest = setdiff(1:length(images), selTrain) ;
imageClass = cat(2, imageClass{:}) ;

% --------------------------------------------------------------------
%                                                  Train vocabulary
% --------------------------------------------------------------------

if ~exist(conf.vocabPath) || conf.clobber

  % Get some PHOW descriptos to train the dictionary
```

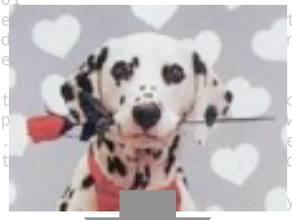
```
    im = imread(fullfile(conf.calDir, images{ii})) ;
    im = imageStandarize(im) ;
    [frames, descrs] = vl_phow(im, conf.phowOpts{:}) ;

    % quantize appearance
    [drop, binsa] = min(vl_alldist(words, single(descrs)), [], 1) ;

    % quantize location
    width = size(im, 2) ;
    height = size(im, 1) ;
    binsx = vl_binsearch(linspace(1,width,conf.numSpatialX+1), frames
(1,:)) ;
    binsy = vl_binsearch(linspace(1,height,conf.numSpatialY+1), frames
(2,:)) ;

    % combined quantization
    bins = sub2ind([conf.numSpatialY, conf.numSpatialX,
conf.numWords], ...
                   binsy,binsx,binsa) ;
    hist = zeros(conf.numSpatialY * conf.numSpatialX * conf.numWords,
1) ;
    hist = vl_binsum(hist, ones(size(bins)), bins) ;
    hist = single(hist / sum(hist)) ;

    hists{ii} = hist ;
  end

  hists = cat(2, hists{:}) ;
```

```
  save(conf.histPath, 'hists') ;
else
  load(conf.histPath) ;
end

% --------------------------------------------------------------------
%                                                  Compute feature map
% --------------------------------------------------------------------

psix = vl_homkermap(hists, 1, .7, 'kchi2') ;

% --------------------------------------------------------------------
%                                                        Train SVM
% --------------------------------------------------------------------

biasMultiplier = 1 ;

if ~exist(conf.modelPath) || conf.clobber
  switch conf.svm.solver
    case 'pegasos'
      lambda = 1 / (conf.svm.C *  length(selTrain)) ;
      models = [] ;
      for ci = 1:length(classes)
        perm = randperm(length(selTrain)) ;
        fprintf('Training model for class %s\n', classes{ci}) ;
        2 * (imageClass(selTrain) == ci) - 1 ;
els(:,ci) = vl_pegasos(psix(:,selTrain(perm)), ...
                       int8(y(perm)), lambda, ...
                       'NumIterations', 20/lambda, ...
                       'BiasMultiplier', biasMultiplier) ;

iblinear'
 = train(imageClass(selTrain)', ...
        sparse(double(psix(:,selTrain))),  ...
        sprintf(' -s 3 -B 1 -c %f', conf.svm.C), ...
        'col') ;
s = model.w' ;

.modelPath, 'models') ;

.modelPath) ;


% --------------------------------------------------------------------
                                                  Test SVM and evaluate
% --------------------------------------------------------------------

the class of the test images
 ;
length(classes)
, :) = models(1:end-1,ci)' * psix + models(end,ci) *
ier ;

eEstClass] = max(scores, [], 1) ;

% Compute the confusion matrix
idx = sub2ind([length(classes), length(classes)], ...
              imageClass(selTest), imageEstClass(selTest)) ;
confus = zeros(length(classes)) ;
confus = vl_binsum(confus, ones(size(idx)), idx) ;

% Plots
figure(1) ; clf;
subplot(1,2,1) ;
imagesc(scores(:,[selTrain selTest])) ; title('Scores') ;
set(gca, 'ytick', 1:length(classes), 'yticklabel', classes) ;
subplot(1,2,2) ;
imagesc(confus) ;
title(sprintf('Confusion matrix (%.2f %% accuracy)', ...
              100 * mean(diag(confus)/conf.numTest) )) ;
print('-depsc2', [conf.resultPath '.ps']) ;
save([conf.resultPath '.mat'], 'confus', 'conf') ;


% --------------------------------------------------------------------
function im = imageStandarize(im)
% --------------------------------------------------------------------

im = im2single(im) ;
if size(im,1) > 480, im = imresize(im, [480 NaN]) ; end
```

- single file
- ~ 240 lines
- all inclusive

# Running example: Complete code





PHOW
(dense SIFT)

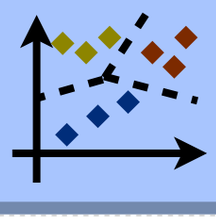
Visual Words

Spatial
Histograms

Kernel Map

$\Psi$

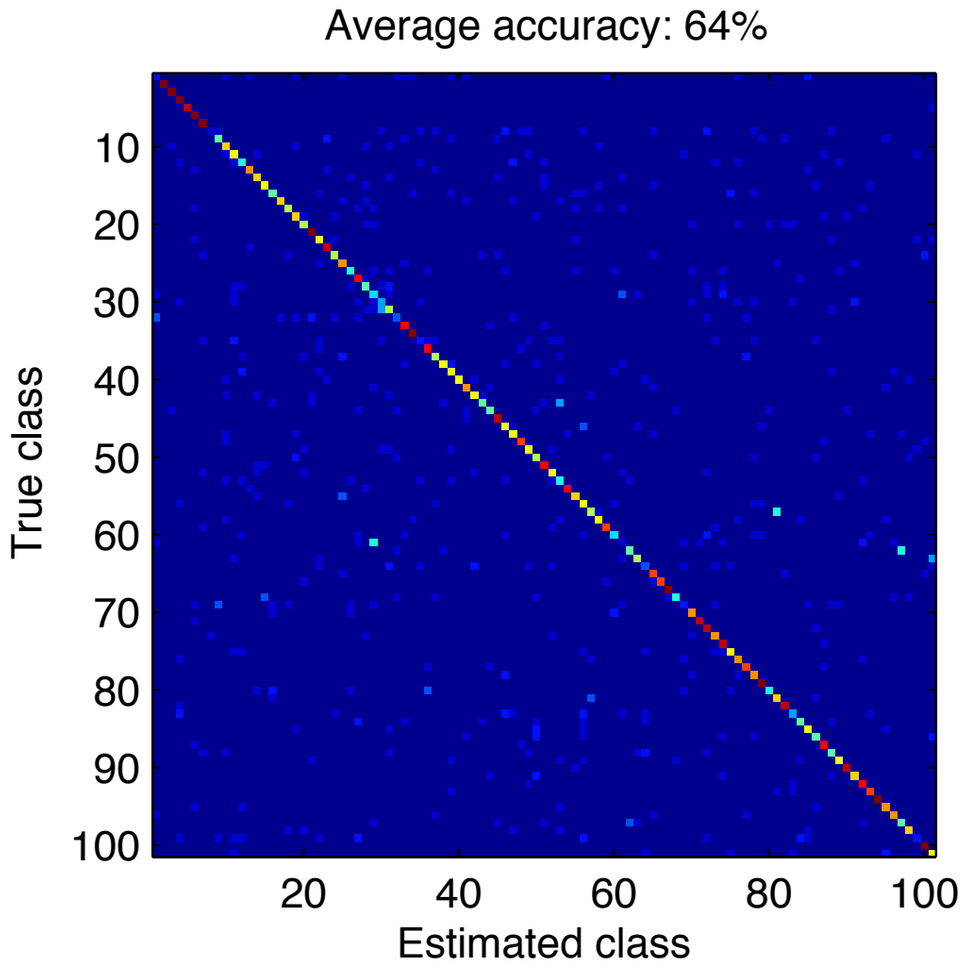
SVM

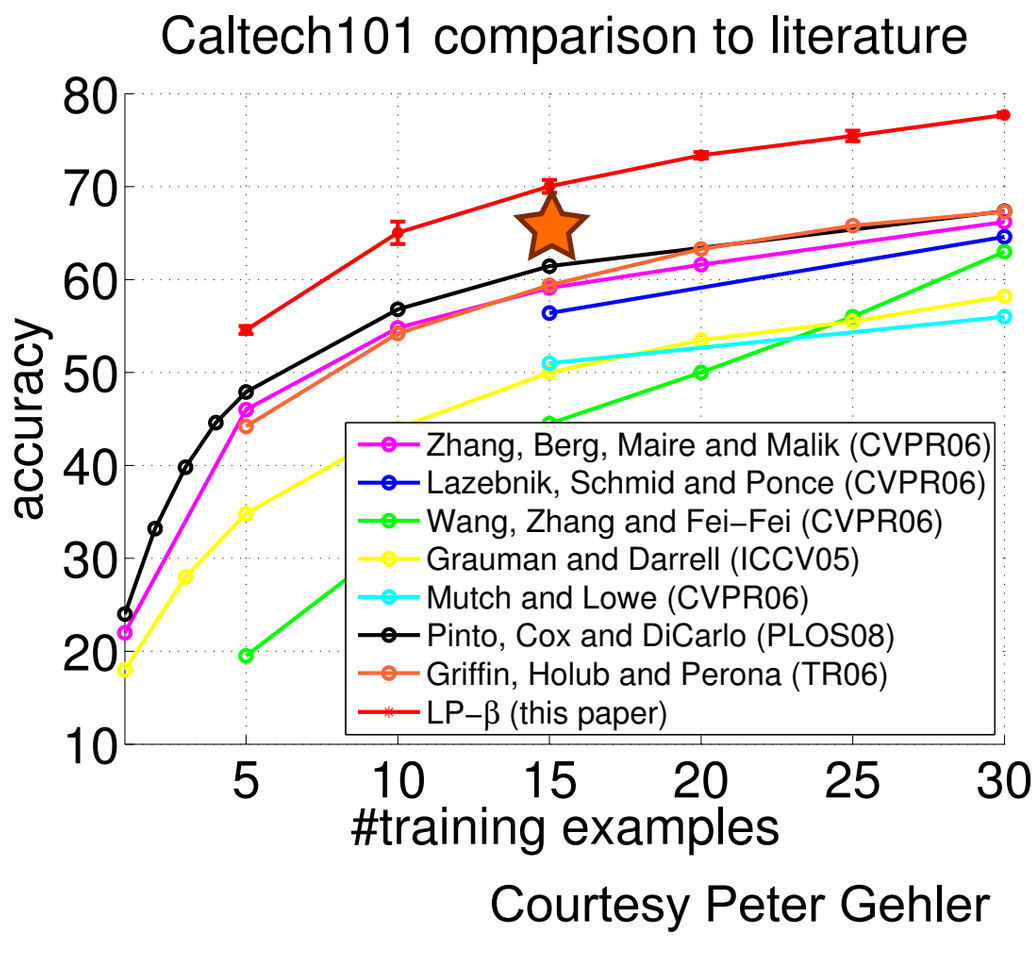


```
function phow_caltech101
% PHOW_CALTECH101 Image                                  tech-101 dataset
%   A quick test should                                  ning PHOW_CALTECH101
%   from MATLAB (provie                                   installed).
%
%   The program automat                                  ch-101 data from the
%   interned and decomp                                  which defaults to
%   'data/caltech-101'.                                  esidred location, for
%   instance to point t                                  data.
%
%   Intermediate files                                   y CONF.DATADIR.
%
%   To run on the entiere datase                         e CONF.TINYPROBLEM to
%   FALSE.
%
%   To run with a                                        g images, change
%   CONF.NUMTRAIN a                                      ining images are
%   used, which sho                                      e (this is quite
%   remarkable for

% AUTORIGHTS

conf.calDir = 'data
conf.dataDir = 'dat
conf.autoDownloadDa
conf.numTrain = 15
conf.numTest = 15 ;
conf.numClasses = 102
conf.numWords = 800
conf.numSpatialX =
conf.numSpatialY =
conf.svm.C = 10 ;
conf.svm.solver = '
conf.phowOpts = {}
conf.clobber = fals
conf.tinyProblem =
conf.prefix = 'base
conf.randSeed = 1 ;

if conf.tinyProble
  conf.prefix = 'ti
  conf.numClasses =
  conf.numSpatialX
  conf.numSpatialY
  conf.numWords =
  conf.phowOpts =                                        3, 'Color', true} ;
end

conf.vocabPath = fu                                      '-vocab.mat']) ;
conf.histPath = ful                                      -hists.mat']) ;
conf.modelPath = fu                                      '-model.mat']) ;
conf.resultPath = f                                      '-result.mat']) ;

randn('state',conf
rand('state',conf.r
vl_twister('state',c

% -------------------                                    ----------------
%                                                        Caltech-101 data
% -------------------                                    ----------------

if ~exist(conf.calD
   (~exist(fullfile                                      && ...
    ~exist(fullfile                                      s', 'airplanes')))
  if ~conf.autoDown
    error(...
      ['Caltech-101
       'Set conf.au                                      e requried
data.']) ;
  end
  vl_xmkdir(conf.calD
  calUrl = ['http://                                     sets/' ...
    'Caltech101/10
  fprintf('Downloa                                       is will take a
while.', conf.calD
  untar(calUrl, co
end

if ~exist(fullfile
  conf.calDir = fu                                       gories') ;
end

% -------------------                                    ----------------
%                                                                Setup data
% -------------------                                    ----------------
classes = dir(conf.calDir) ;
classes = classes([classes.isdir]) ;
classes = {classes(3:conf.numClasses+2-1).name} ;

images = {} ;
imageClass = {} ;
for ci = 1:length(classes)
  ims = dir(fullfile(conf.calDir, classes{ci}, '*.jpg'))' ;
  ims = vl_colsubset(ims, conf.numTrain + conf.numTest) ;
  ims = cellfun(@(x)fullfile(classes{ci},x),
{ims.name},'UniformOutput',false) ;
  images = {images{:}, ims{:}} ;
  imageClass{end+1} = ci * ones(1,length(ims)) ;
end
selTrain = find(mod(0:length(images)-1, conf.numTrain+conf.numTest) <
conf.numTrain) ;
selTest = setdiff(1:length(images), selTrain) ;
imageClass = cat(2, imageClass{:}) ;

% -------------------------------------------------------------------
%                                                     Train vocabulary
% -------------------------------------------------------------------

if ~exist(conf.vocabPath) || conf.clobber

  % Get some PHOW descriptos to train the dictionary
  selTrainFeats = vl_colsubset(selTrain, 30) ;
  descrs = {} ;
  for ii = 1:length(selTrainFeats)
    im = imread(fullfile(conf.calDir, images{ii})) ;
    im = imageStandarize(im) ;
    [drop, descrs{ii}] = vl_phow(im, conf.phowOpts{:}) ;
  end

  descrs = vl_colsubset(cat(2, descrs{:}), 10e4) ;
  descrs = single(descrs) ;

  % Quantize the descriptors to get the visual words
  words = vl_kmeans(descrs, conf.numWords, 'verbose', 'algorithm',
'elkan') ;
  save(conf.vocabPath, 'words') ;
else
  load(conf.vocabPath) ;
end

% -------------------------------------------------------------------
%                                           Compute spatial histograms
% -------------------------------------------------------------------

if ~exist(conf.histPath) || conf.clobber
  hists = {} ;
  % par
  for ii = 1:length(images)
    fprintf('Processing %s (%.2f %%)\n', images{ii}, 100 * ii / length
(images)) ;
    im = imread(fullfile(conf.calDir, images{ii})) ;
    im = imageStandarize(im) ;
    [frames, descrs] = vl_phow(im, conf.phowOpts{:}) ;

    % quantize appearance
    [drop, binsa] = min(vl_alldist(words, single(descrs)), [], 1) ;

    % quantize location
    width = size(im, 2) ;
    height = size(im, 1) ;
    binsx = vl_binsearch(linspace(1,width,conf.numSpatialX+1), frames
(1,:)) ;
    binsy = vl_binsearch(linspace(1,height,conf.numSpatialY+1), frames
(2,:)) ;

    % combined quantization
    bins = sub2ind([conf.numSpatialY, conf.numSpatialX,
conf.numWords], ...
                   binsy,binsx,binsa) ;
    hist = zeros(conf.numSpatialY * conf.numSpatialX * conf.numWords,
1) ;
    hist = vl_binsum(hist, ones(size(bins)), bins) ;
    hist = single(hist / sum(hist)) ;

    hists{ii} = hist ;
  end

  hists = cat(2, hists{:}) ;
  save(conf.histPath, 'hists') ;
else
  load(conf.histPath) ;
end

% -------------------------------------------------------------------
%                                                  Compute feature map
% -------------------------------------------------------------------

psix = vl_homkermap(hists, 1, .7, 'kchi2') ;

% -------------------------------------------------------------------
%                                                        Train SVM
% -------------------------------------------------------------------

biasMultiplier = 1 ;

if ~exist(conf.modelPath) || conf.clobber
  switch conf.svm.solver
    case 'pegasos'
      lambda = 1 / (conf.svm.C *  length(selTrain)) ;
      models = [] ;
      for ci = 1:length(classes)
        perm = randperm(length(selTrain)) ;
        fprintf('Training model for class %s\n', classes{ci}) ;
        y = 2 * (imageClass(selTrain) == ci) - 1 ;
        models(:,ci) = vl_pegasos(psix(:,selTrain(perm)), ...
                                  int8(y(perm)), lambda, ...
                                  'NumIterations', 20/lambda, ...
                                  'BiasMultiplier', biasMultiplier) ;
      end
    case 'liblinear'
      model = train(imageClass(selTrain)', ...
                    sparse(double(psix(:,selTrain))),  ...
                    sprintf(' -s 3 -B 1 -c %f', conf.svm.C), ...
                    'col') ;
      models = model.w' ;
  end
  save(conf.modelPath, 'models') ;
else
  load(conf.modelPath) ;
end

% -------------------------------------------------------------------
%                                                    Test SVM and evaluate
% -------------------------------------------------------------------

% Estimate the class of the test images
scores = [] ;
for ci = 1:length(classes)
  scores(ci, :) = models(1:end-1,ci)' * psix + models(end,ci) *
biasMultiplier ;
end
[drop, imageEstClass] = max(scores, [], 1) ;

% Compute the confusion matrix
idx = sub2ind([length(classes), length(classes)], ...
              imageClass(selTest), imageEstClass(selTest)) ;
confus = zeros(length(classes)) ;
confus = vl_binsum(confus, ones(size(idx)), idx) ;

% Plots
figure(1) ; clf;
subplot(1,2,1) ;
imagesc(scores(:,[selTrain selTest])) ; title('Scores') ;
set(gca, 'ytick', 1:length(classes), 'yticklabel', classes) ;
subplot(1,2,2) ;
imagesc(confus) ;
title(sprintf('Confusion matrix (%.2f %% accuracy)', ...
              100 * mean(diag(confus)/conf.numTest) )) ;
print('-depsc2', [conf.resultPath '.ps']) ;
save([conf.resultPath '.mat'], 'confus', 'conf') ;


% -------------------------------------------------------------------
function im = imageStandarize(im)
% -------------------------------------------------------------------

im = im2single(im) ;
if size(im,1) > 480, im = imresize(im, [480 NaN]) ; end
```

# Running example: Results

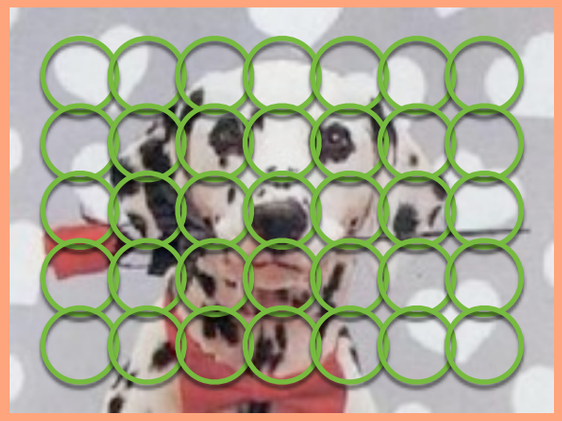
- Among the best single-feature methods on Caltech-101
  - PHOW (dense SIFT)
  - 15 training - 15 testing images
  - 64-66% average accuracy
- Fast training and testing



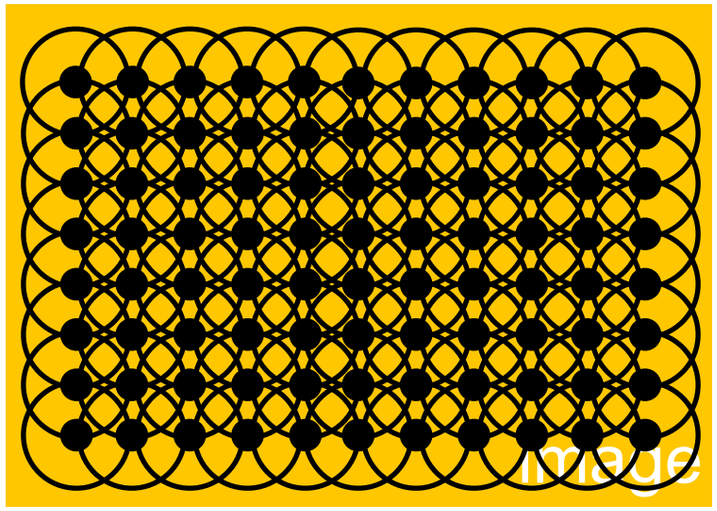

Courtesy Peter Gehler

PHOW
(dense SIFT)



# Dense SIFT: PHOW features

- **PHOW features**
  - [Bosch et al. 2006], [Bosch et al. 2007]
  - dense multiscale SIFT
  - uniform spacing (e.g. 5 pixels)
  - four scales (e.g. 5, 7, 10, 12 pixels)

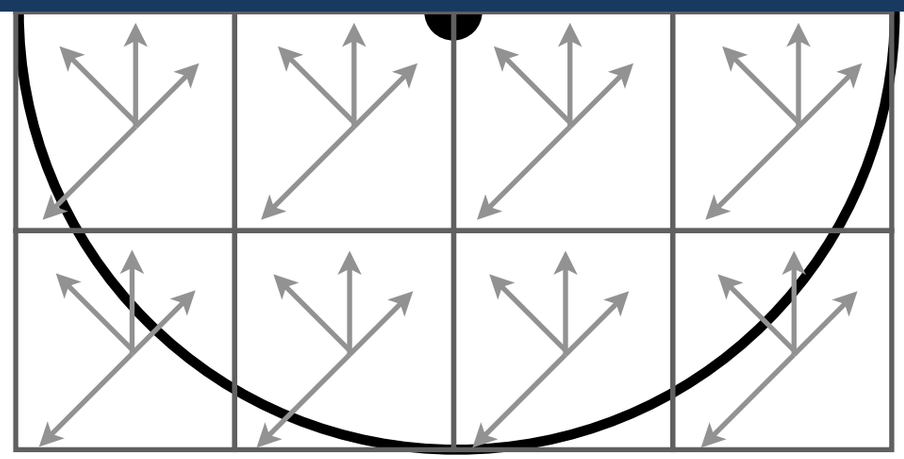



- **Direct implementation**
  - for each scale
    - create a list of keypoints
    - call **vl_sift**

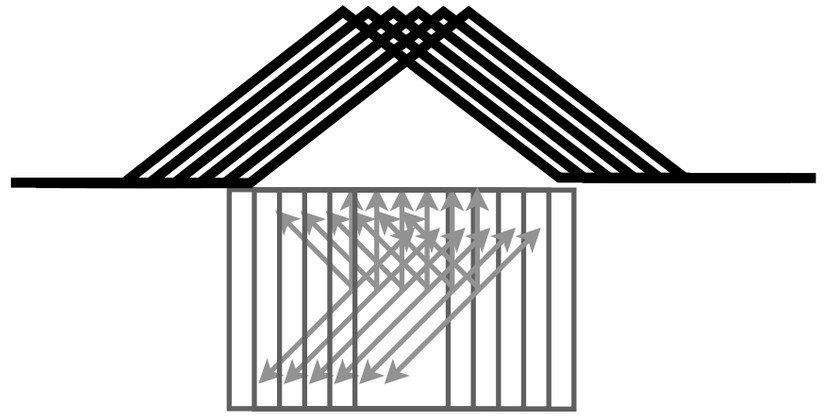

```
step = 5 ;
for s = [5, 7, 10, 12]
  [x, y] = meshgrid(1:step:width, 1:step:height) ;
  frames = [x(:)'; y(:)'] ;
  frames(3,:) = s / 3 ;
  frames(4,:) = 0 ;
  [frames, descrs] = vl_sift(im, 'Frames', frames) ;
end
```

# Accelerating dense SIFT

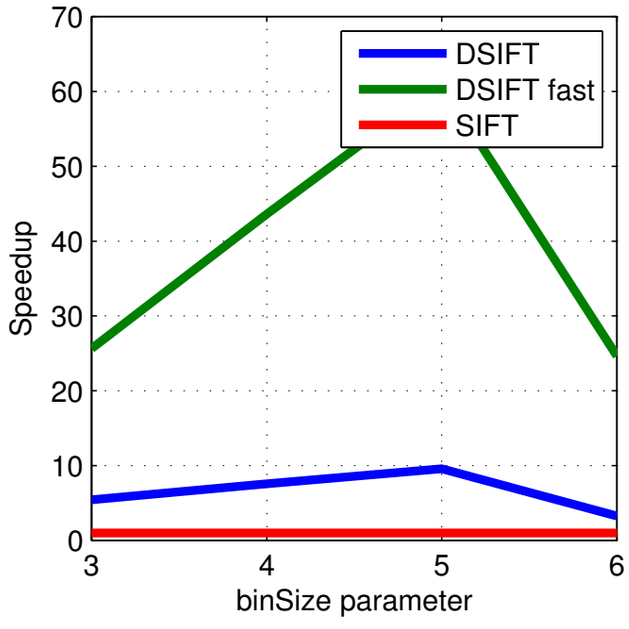
- Large **speedup** by exploiting
  - same scale and orientation
  - uniform sampling



- **Dense SIFT**
  - linear interpolation = convolution
  - convolution by integral images
  - piecewise approximation of Gaussian window

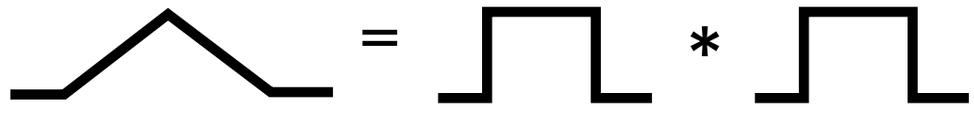


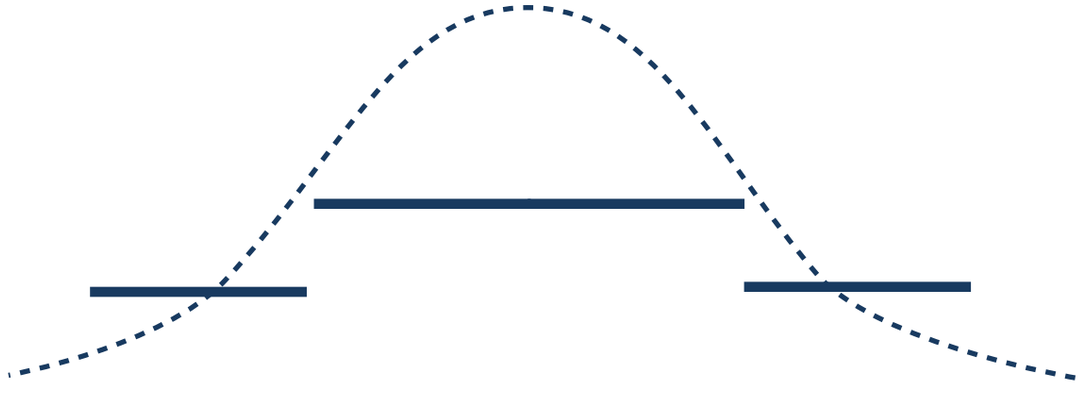
- Implemented by **`vl_dsift`**




60 x speedup!

30 x speedup!





# Dense SIFT: The fast version

- PHOW = Fast dense SIFT at multiple scales

Use `vl_imsmooth` compute the scale space.

```
for i = 1:4
  ims = vl_imsmooth(im, scales(i) / 3) ;
  [frames{s}, descrs{s}] = ...
      vl_dsift(ims, 'Fast', ...
               'Step', step, 'Size', scales(i)) ;
end
```

- **Remark**
  - for demonstration only!
  - in practice use the provided wrapper

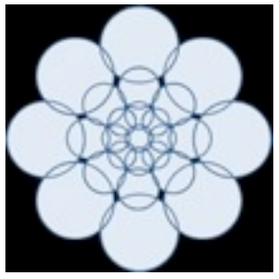
```
[frames, descrs] = vl_phow(im) ;
```

# Accelerating dense SIFT: related methods



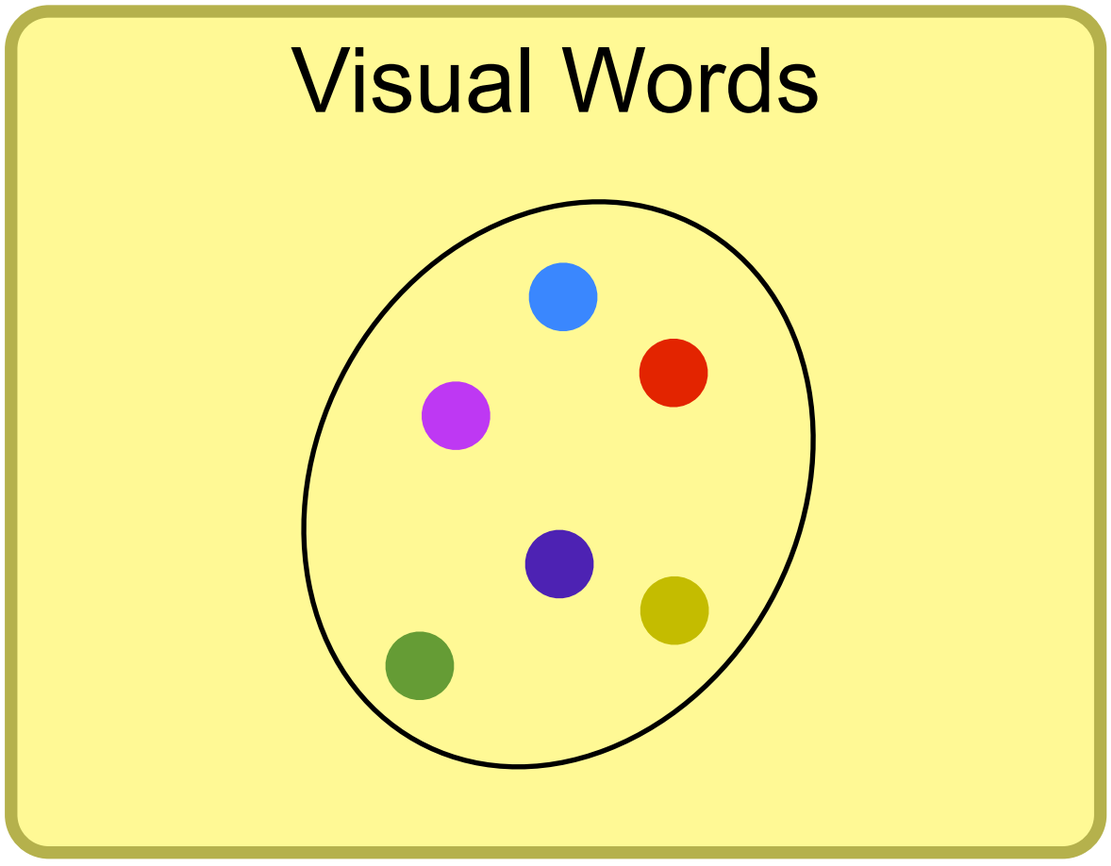
- **SURF**
  - [Bay et al. 2008]
  - Original closed source
    http://www.vision.ee.ethz.ch/~surf/index.html
  - OpenSURF
    http://www.chrisevansdev.com/computer-vision-opensurf.html

- **Daisy**
  - [Tola et al. 2010]
  - http://cvlab.epfl.ch/~tola/daisy.html



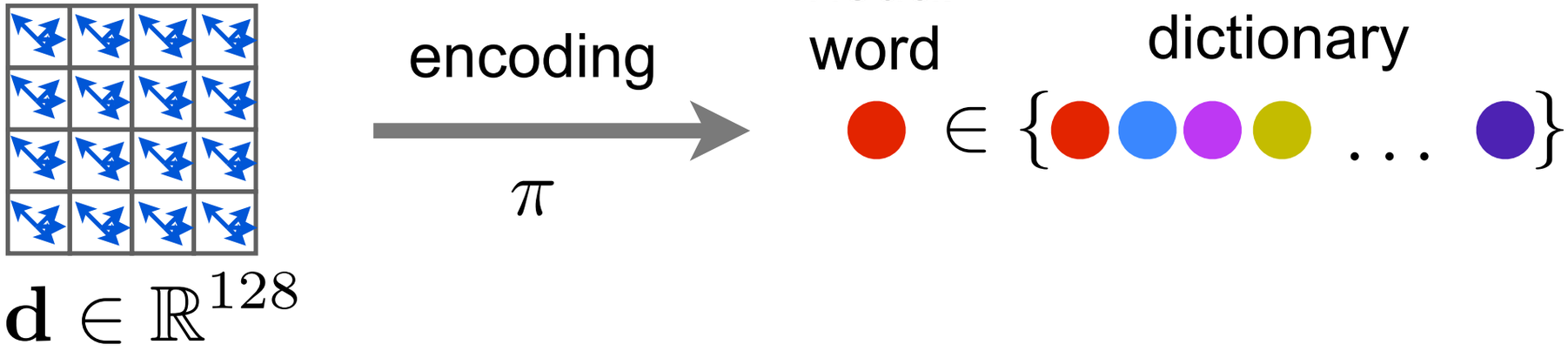
- **GPU-SIFT**
  - [S. N. Sinha et al. 2010]
  - http://cs.unc.edu/~ssinha/Research/GPU_KLT/

- **SiftGPU**
  - [C. Wu 2010]
  - http://www.cs.unc.edu/~ccwu/siftgpu/


Visual Words

# Visual words

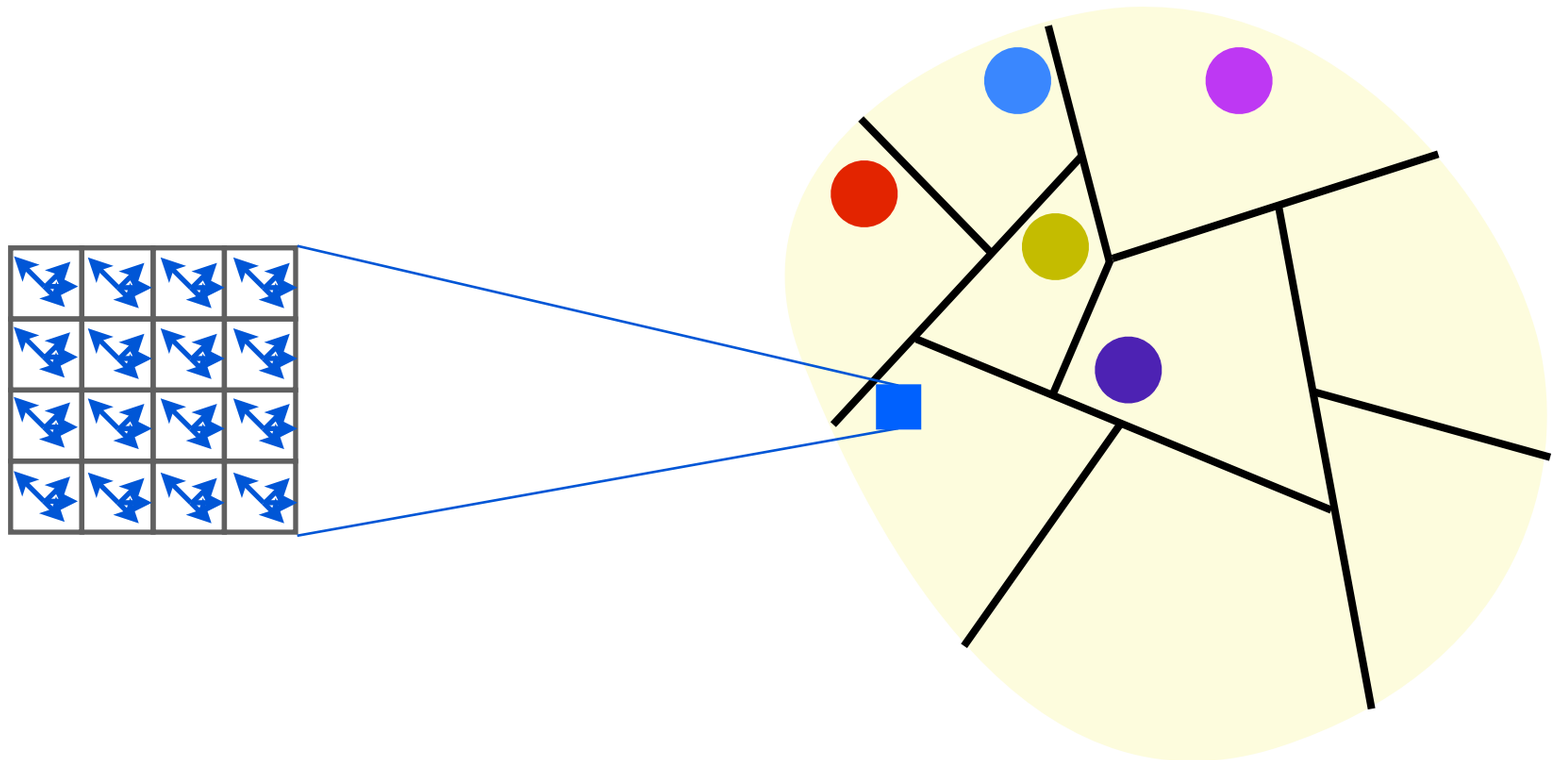
- Visual words



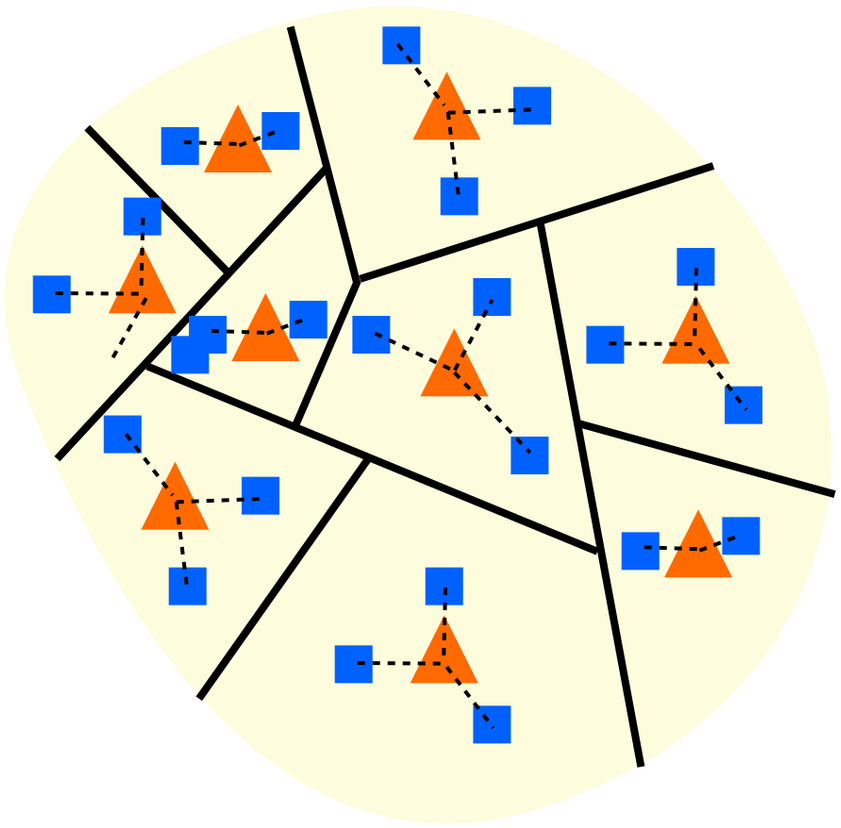

- Encoding = clustering

[Sivic and Zisserman 2003]

  - vector quantization (k-means) [Lloyd 1982]
  - agglomerative clustering [Leibe et al. 2006]
  - affinity propagation [Frey and Dueck 2007]
  - ...

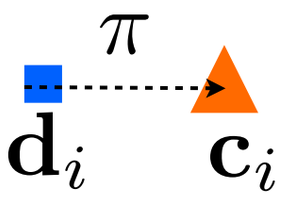


- Optimize **reconstruction cost**

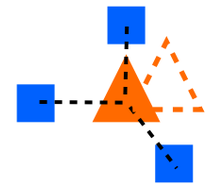


$$E(\pi, \mathbf{c}_1, \dots, \mathbf{c}_K) = \sum_{i=1}^{M} \|\mathbf{d}_i - \mathbf{c}_{\pi(\mathbf{d}_i)}\|^2$$

1. reassign points to closest centers



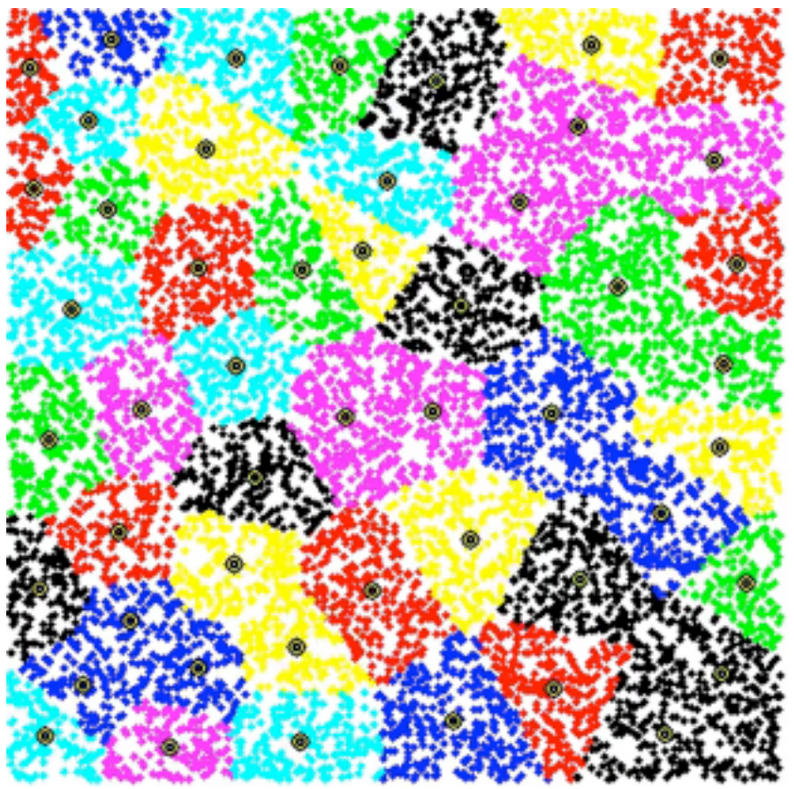

2. fit centers to assigned points



- As simple as you expect

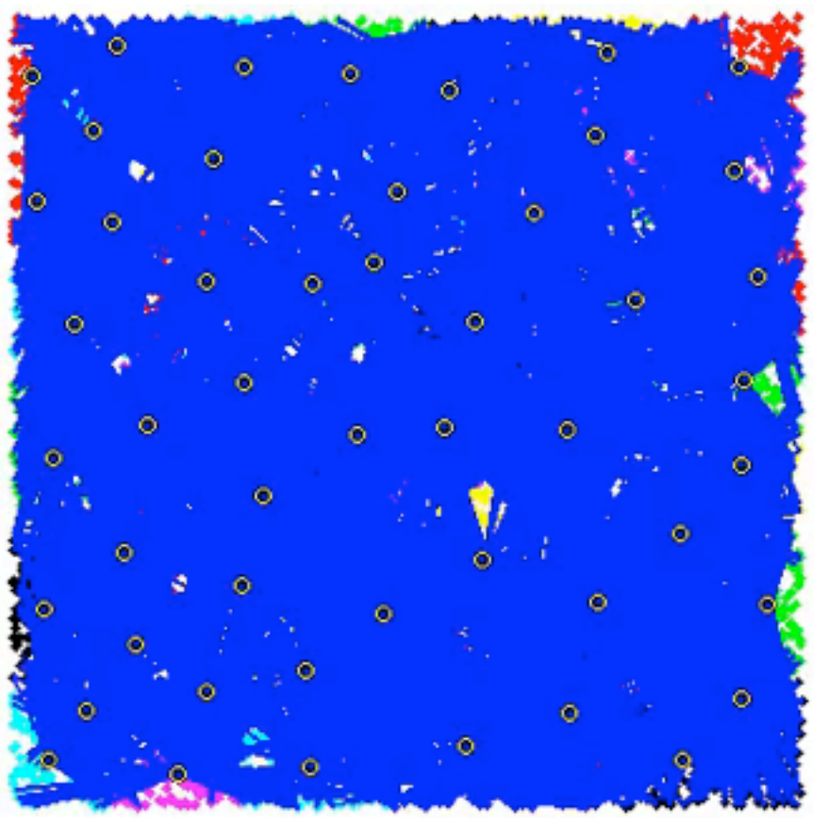
```
centers = vl_kmeans(descrs, K);
```

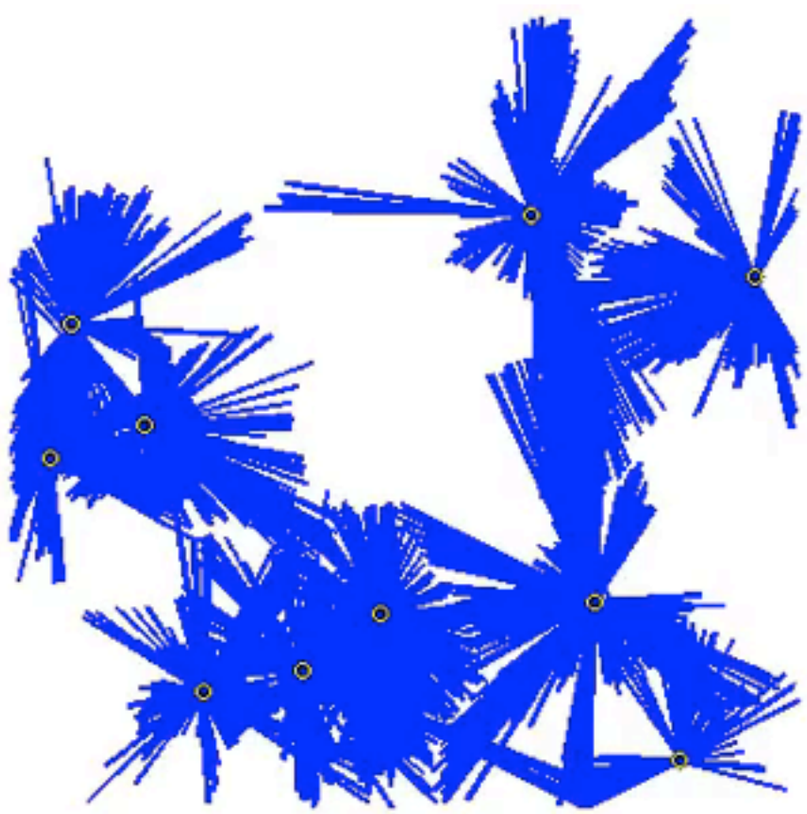
- **Much** faster than MATLAB builtin



# Faster k-means

- **Bottleneck**
  - assign point to closest center
  - needs to compare each point to each center
- **Speedup**
  - if centers change little, then most assignment don’t change
  - can keep track with triangle inequality
  - [Elkan 2003]
  - `centers = vl_kmeans(descrs, K, ‘Algorithm’, ‘Elkan’);`

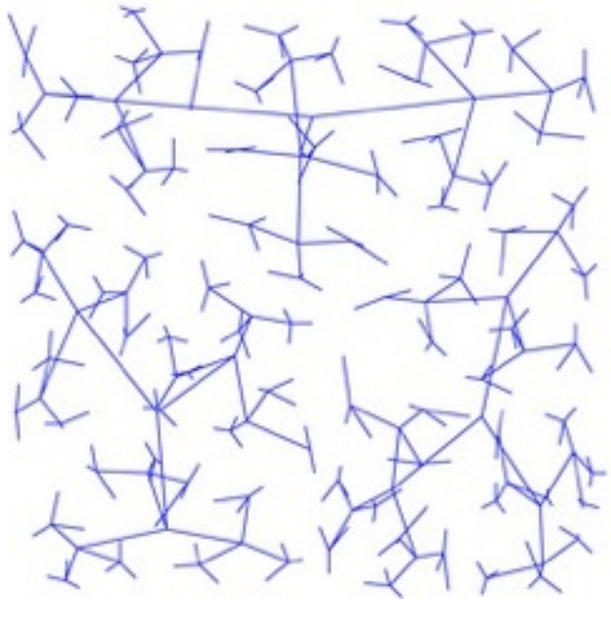


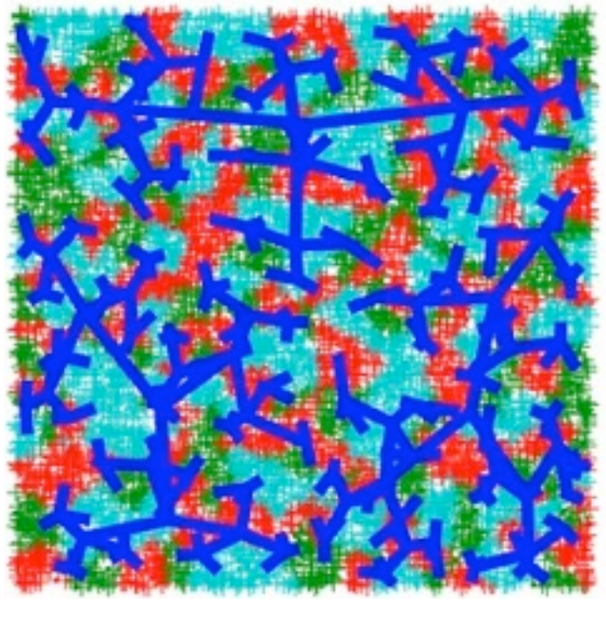


Vector-to-center comparison

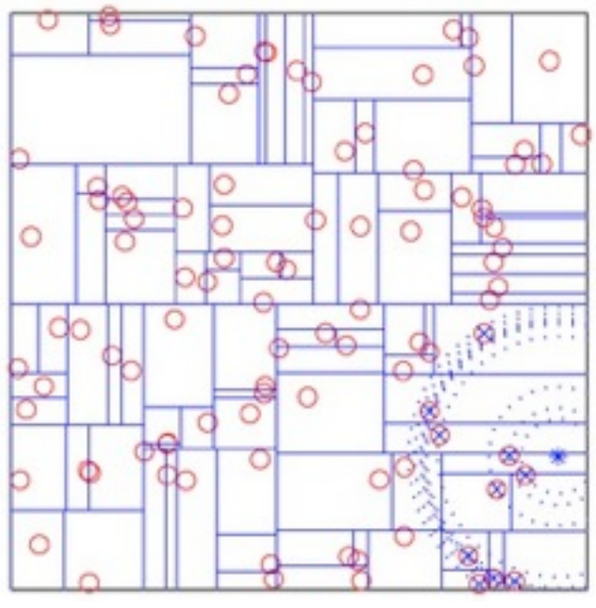
# Speeding-up encoding

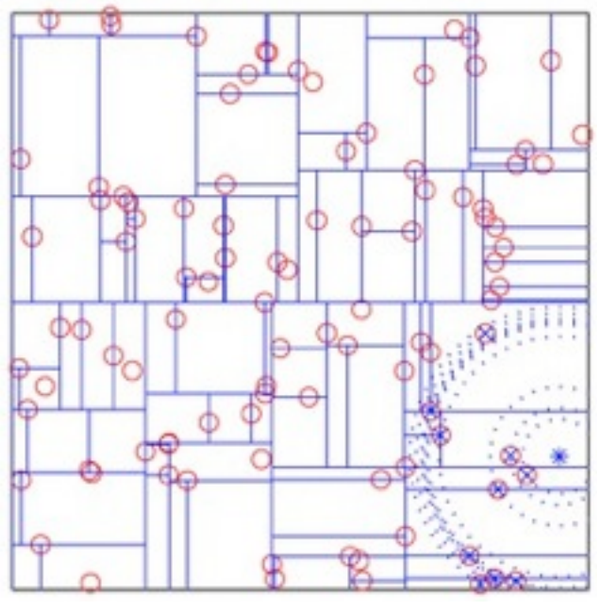
- **Hierarchical k-means clustering**
  - [Nistér and Stewénius 2006] [Hastie et al. 2001]
  - apply k-means recursively
  - logarithmic encoding
  - `vl_hikmeans`





- **kd-trees**
  - multi-dimensional logarithmic search [Friedman et. al 1977]
  - best-bin search and random forests [Muja and Lowe 2009]





# Randomized kd-tree forests

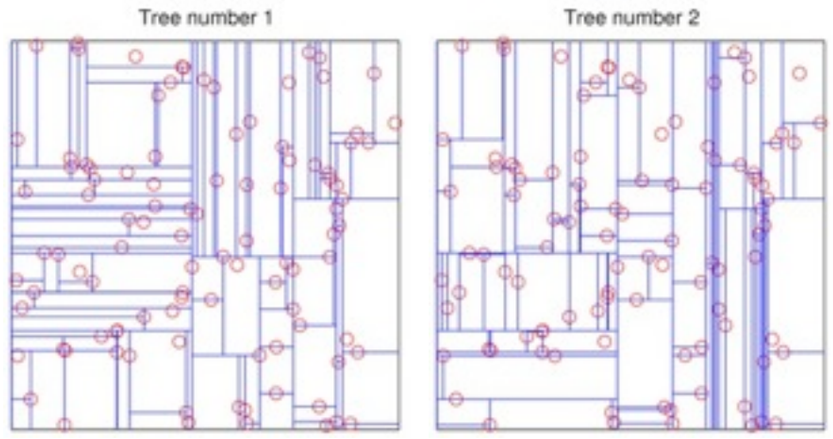


- **Build the tree**

```
kdtree = vl_kdtreebuild(X) ;
```

  - random forests

```
kdtree = vl_kdtreebuild(X, 'NumTrees', 2) ;
```



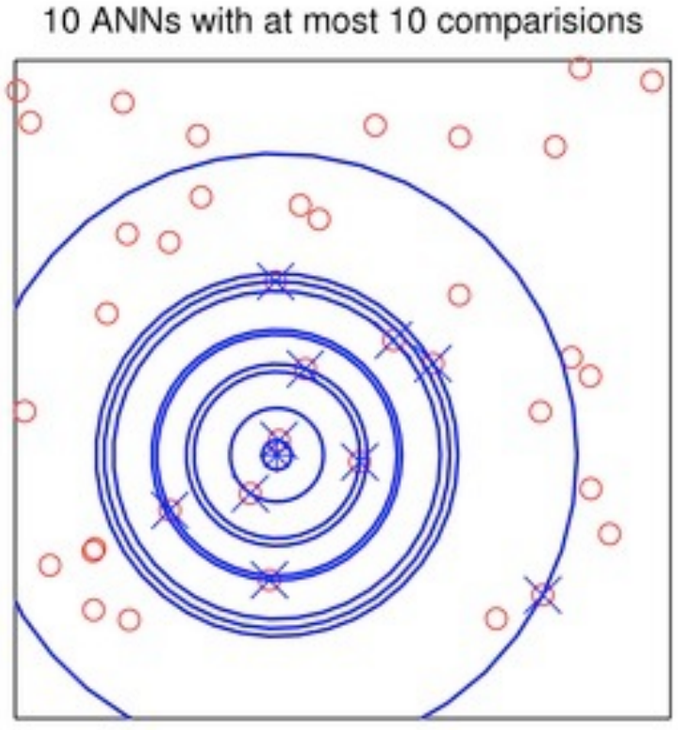

- **Query the tree**

  - Exact nearest neighbors

```
[ind, dist] = vl_kdtreequery(kdtree, X, Q, ...
                             'NumNeighbors' 10) ;
```

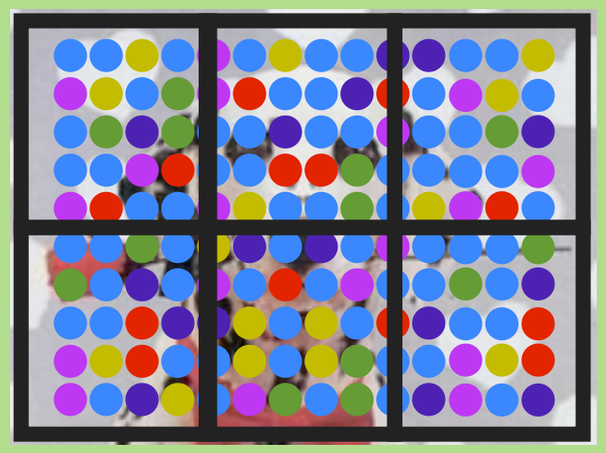
  - Approximate nearest neighbors

```
[ind, dist] = vl_kdtreequery(kdtree, X, Q, ...
                             'NumNeighbors' 10, ...
                             'MaxComparisons', 10)
```

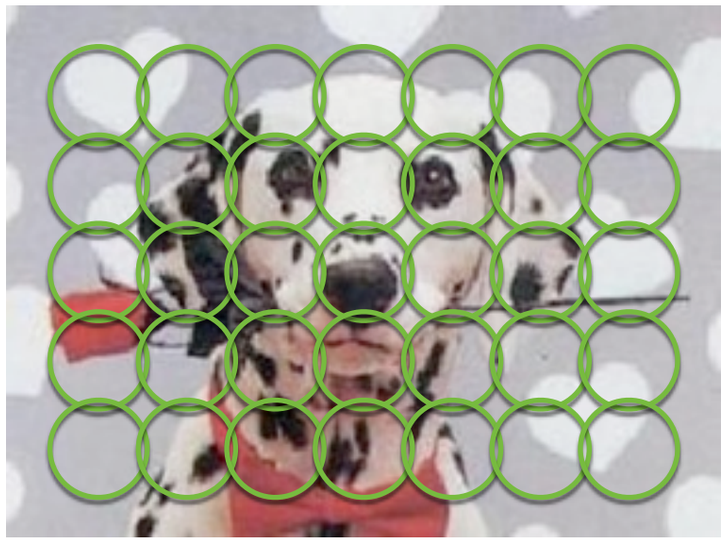


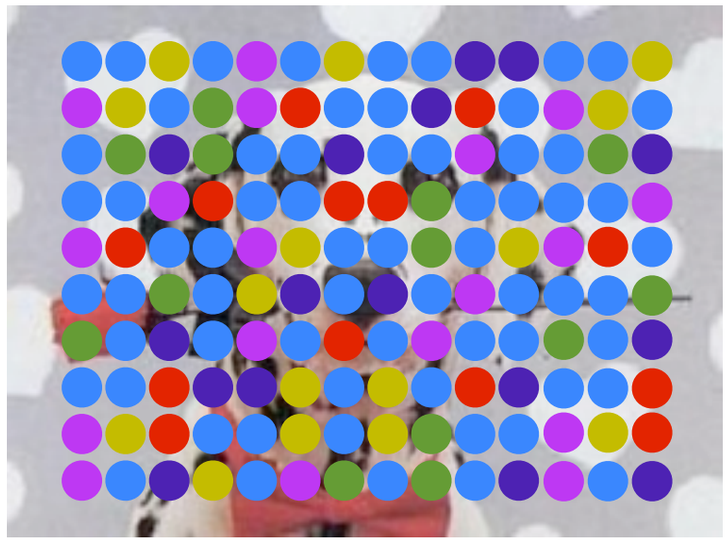

- Random kd-tree forest implementation
  - Equivalent to FLANN
  - [Muja and Lowe 2009]
  - http://www.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN

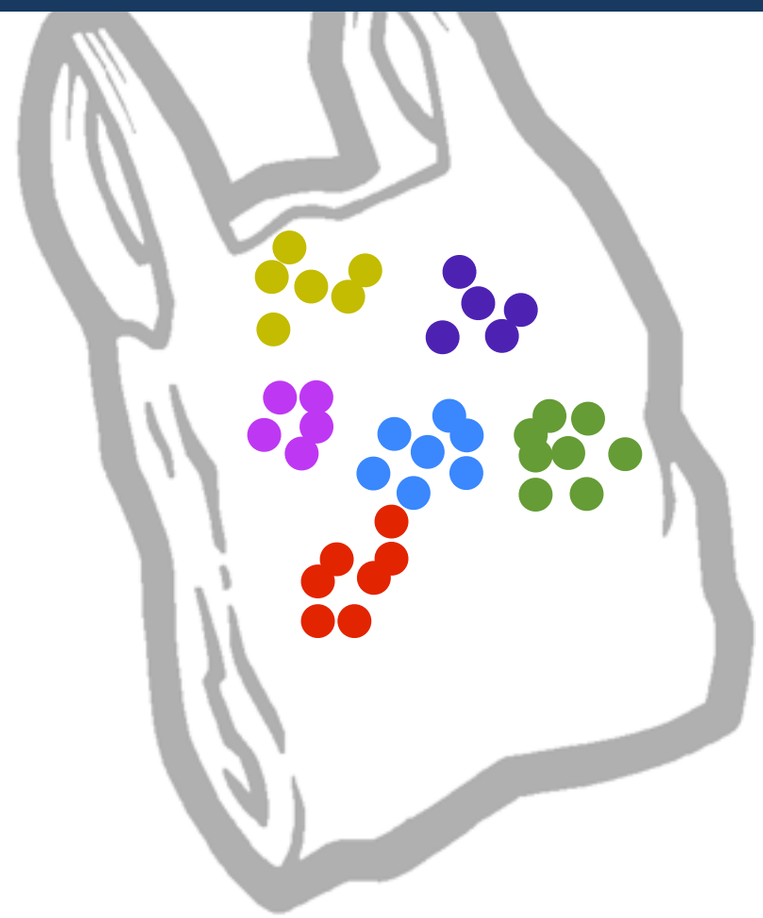
# Spatial Histograms

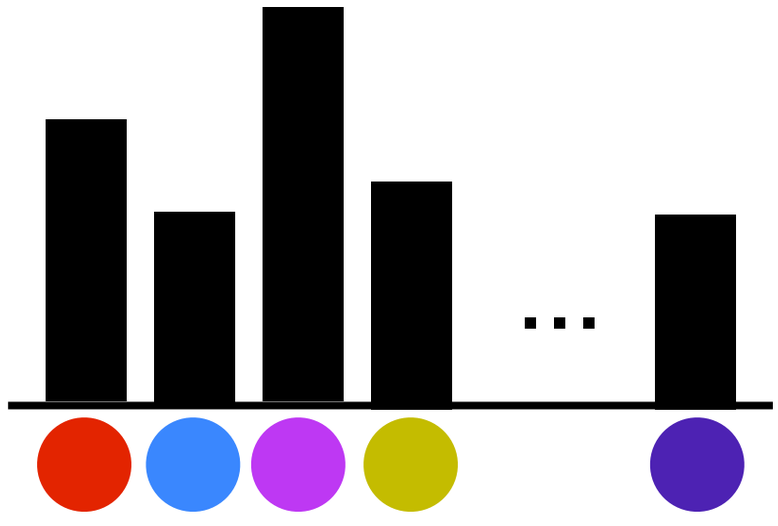


# Bag-of-words









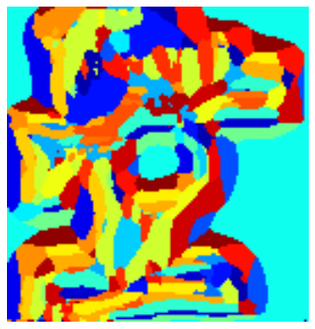
histogram (bag) of visual words

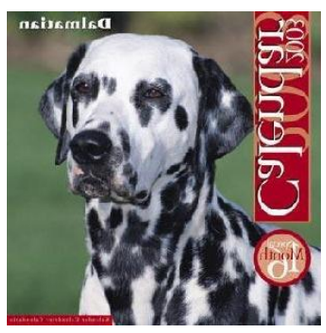
[Csurka et al. 2004]

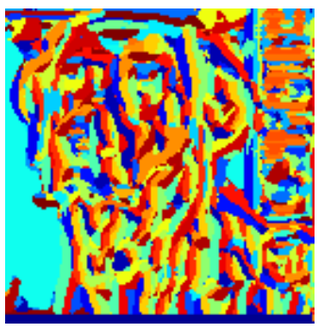
# Example encodings

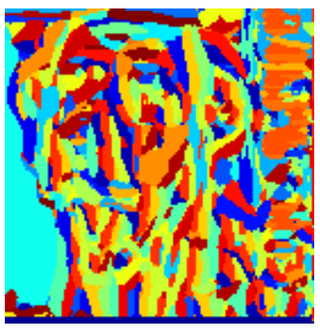


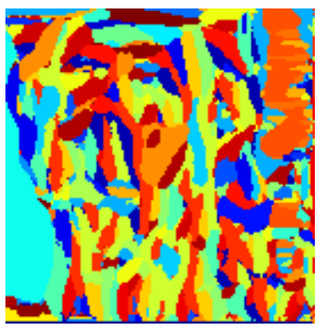
SIFT descriptor scale

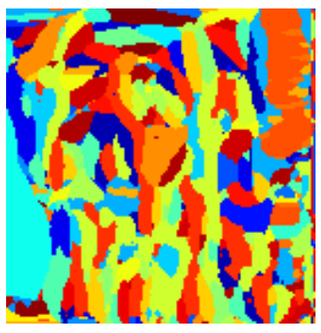


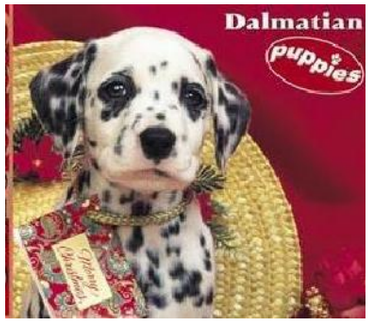



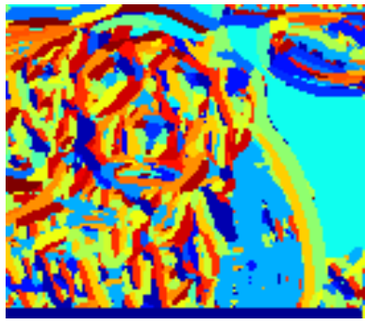



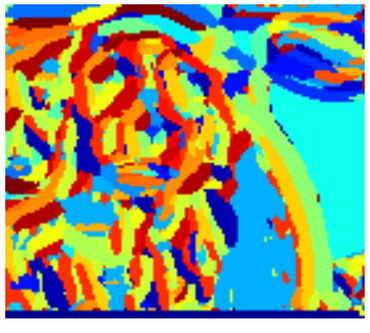


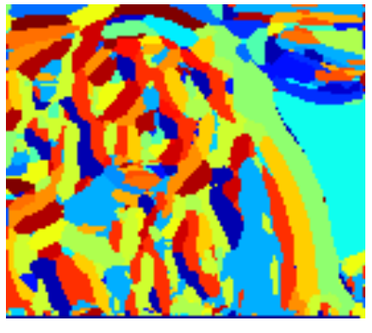


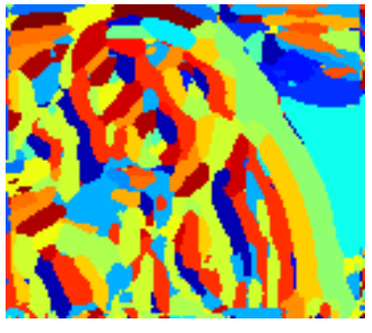




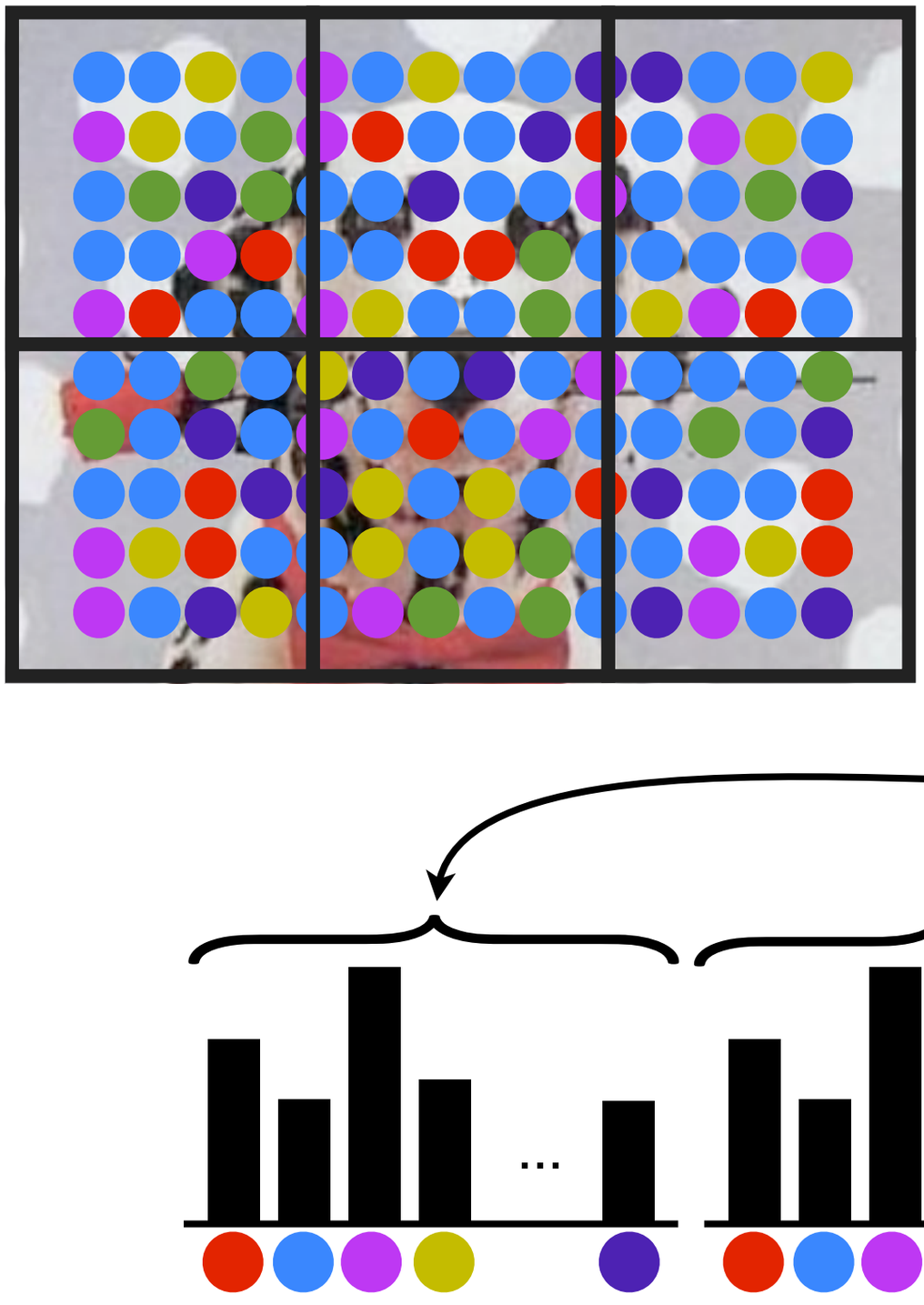


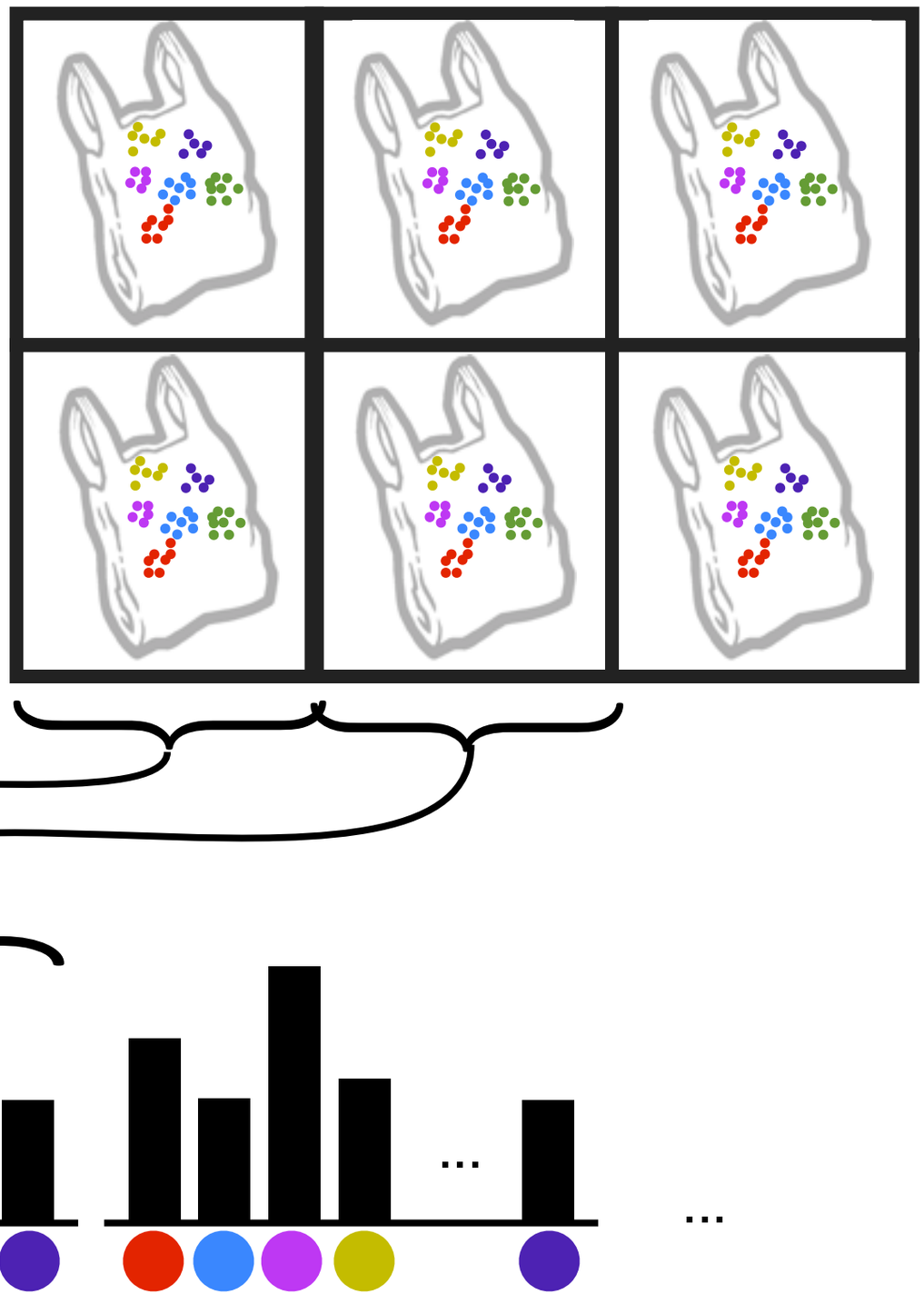




# Spatial histograms







spatial histogram

[Lazebnik et al. 2004]

# VLFeat utility functions



- MEX-accelerated primitives
- Speed-up MATLAB in key operations
- Examples:
  - **Binning**

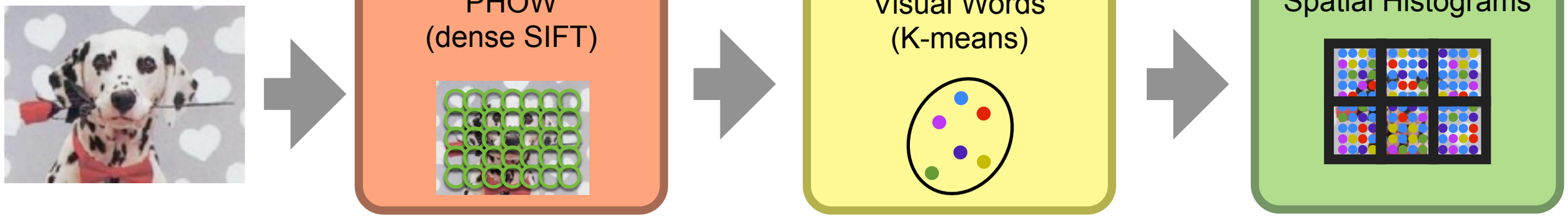
```
% quantize location
binsx = vl_binsearch(linspace(1,width,conf.numSpatialX+1), frames(1,:)) ;
binsy = vl_binsearch(linspace(1,height,conf.numSpatialY+1), frames(2,:)) ;
```

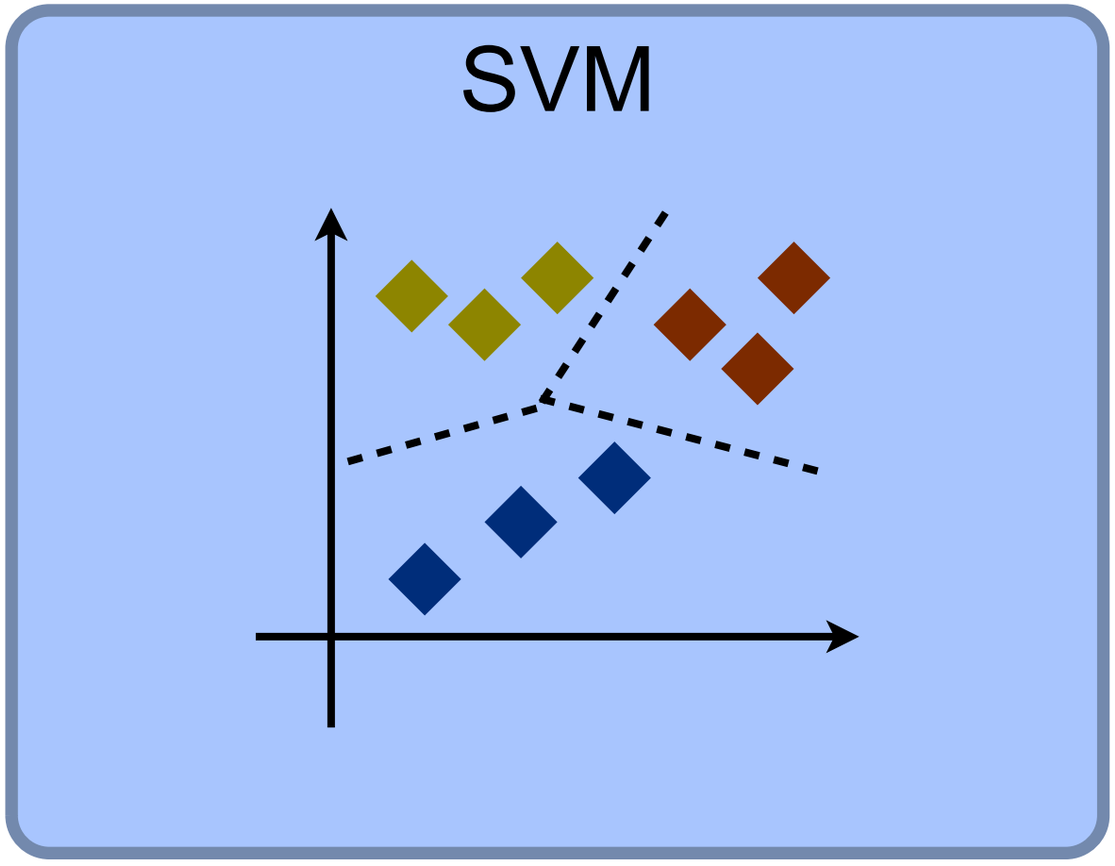
  - **Binned summations**

```
% histogram computation
bins = sub2ind([conf.numSpatialY, conf.numSpatialX, conf.numWords], ...
               binsy,binsx,binsa) ;
hist = zeros(conf.numSpatialY * conf.numSpatialX * conf.numWords, 1) ;
hist = vl_binsum(hist, ones(size(bins)), bins) ;
```

# Summary: image descriptors






SVM

Fisher discriminant

LDA

Dirichelet processes

kernel density estimation

separating hyperplane

kernel methods

logistic regression

nearest neighbors

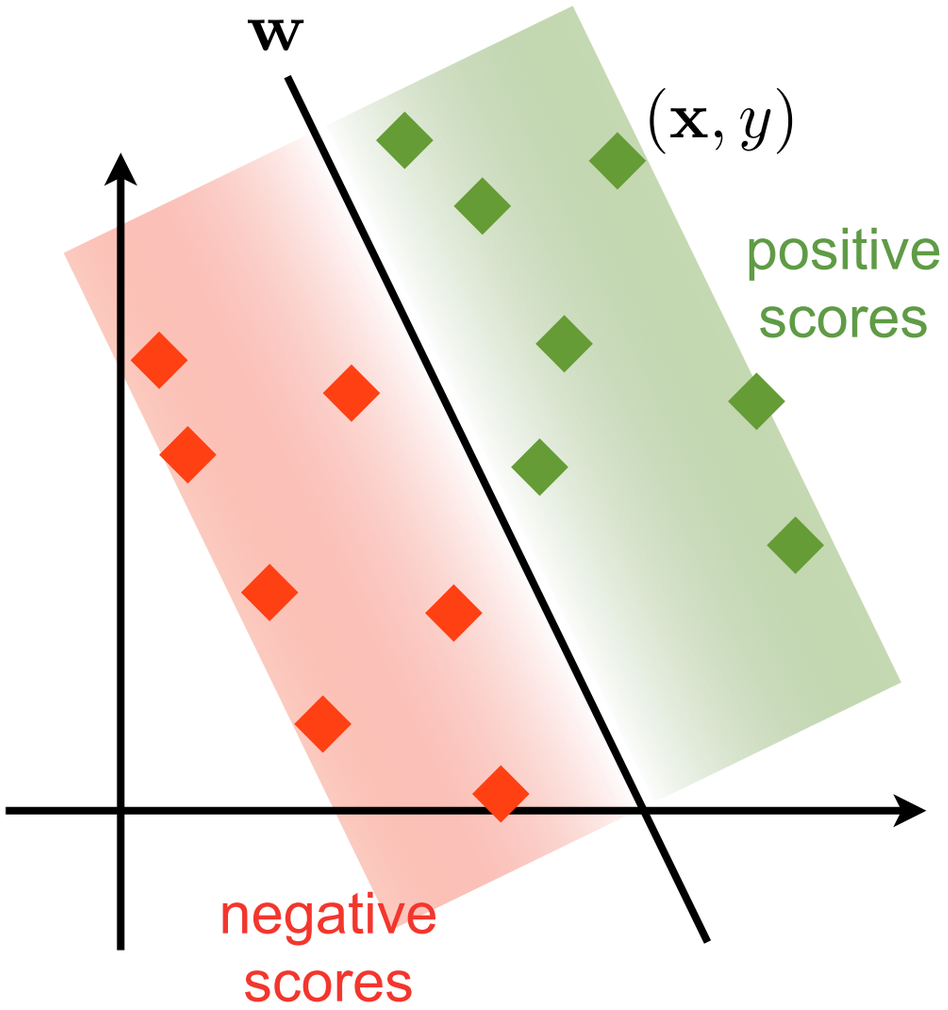
topic models

AdaBoost


random forests

- **SVMs**
  - often *state-of-the-art*
  - *simple*
    to use and tune
  - *efficient*
    with the latest improvements

[Shawe-Taylor and Cristianini 2000]
[Schölkopf and Smola 2002]
[Hastie et al. 2003]




Discriminant score

$$f(\mathbf{x}; \mathbf{w}) = \langle \mathbf{w}, \mathbf{x} \rangle$$

$$\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{w}; (\mathbf{x}_i, y_i))$$

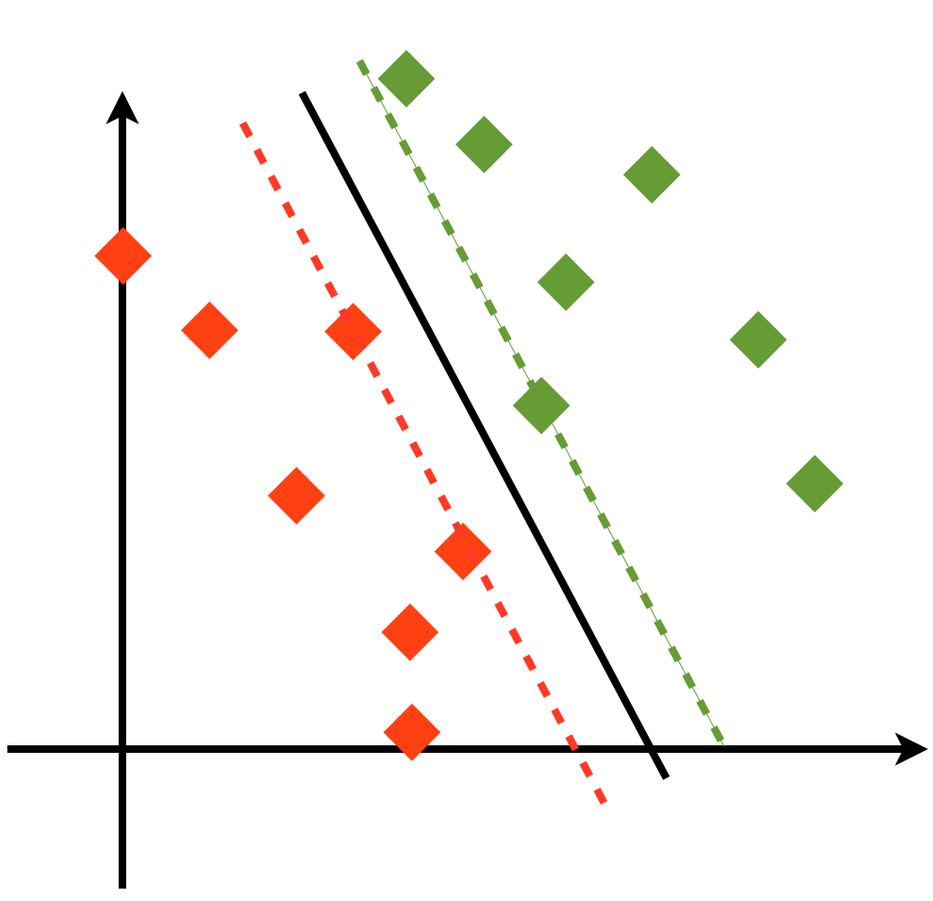
$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{w}; (\mathbf{x}_i, y_i))$$

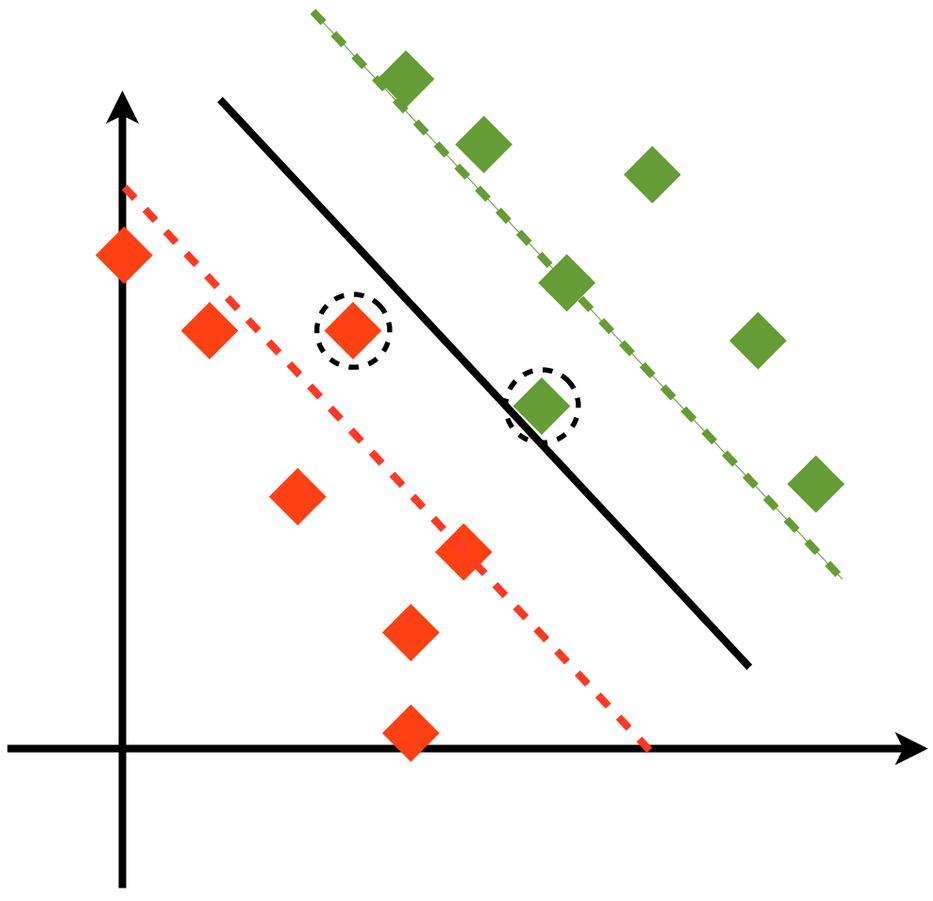
**hard loss**

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \begin{cases} 0, & y\langle \mathbf{w}, \mathbf{x}\rangle > 1, \\ +\infty, & \text{otherwise.} \end{cases}$$

**hinge loss**

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \begin{cases} 0, & y\langle \mathbf{w}, \mathbf{x}\rangle > 1, \\ 1 - y\langle \mathbf{w}, \mathbf{x}\rangle, & \text{otherwise.} \end{cases}$$





# Linear SVM: Fast algorithms



- **SVM*perf***
  - [Joachims 2006]
  - http://www.cs.cornell.edu/People/tj/svm_light/svm_perf.html
  - cutting plane + one slack formulation
  - **from superlinear to linear complexity**
- **LIBLINEAR**
  - [R.-E. Fan et al. 2008]
  - http://www.csie.ntu.edu.tw/~cjlin/liblinear/

**l1 loss**

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}$$

**l2 loss**

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = (\max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\})^2$$

**logistic loss**

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \log(1 + \exp(-y\langle \mathbf{w}, \mathbf{x} \rangle))$$

**l² regularizer**

$$\frac{\lambda}{2} \sum_{i=1}^{D} w_i^2$$

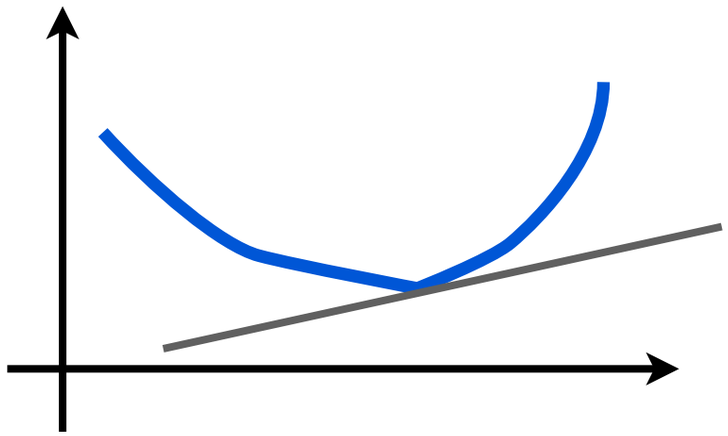
**l¹ regularizer**

$$\frac{\lambda}{2} \sum_{i=1}^{D} |w_i|$$

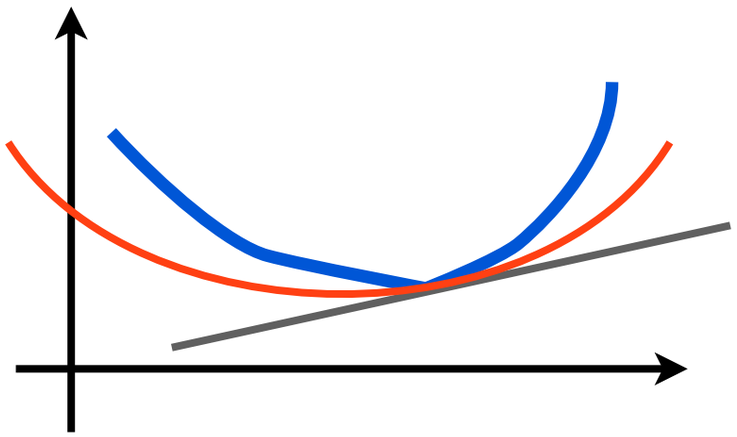
# Linear SVM: PEGASOS

- [S. Shalev-Shwartz et al. 2010]
- http://www.cs.huji.ac.il/~shais/code/index.html

1. Pick a random data point $(\mathbf{x}_i, y_i)$
2. $$\nabla E_t(\mathbf{w}) = \begin{cases} \lambda \mathbf{w}_t, & y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 1, \\ \lambda \mathbf{w}_t - y_i \mathbf{x}_i, & \text{otherwise.} \end{cases}$$
3. $$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{\lambda t} \nabla E_t(\mathbf{w})$$



non differentiable objective
⇒ **sub-gradient**



strongly-convex objective
⇒ **optimal schedule**

# VLFeat PEGASOS

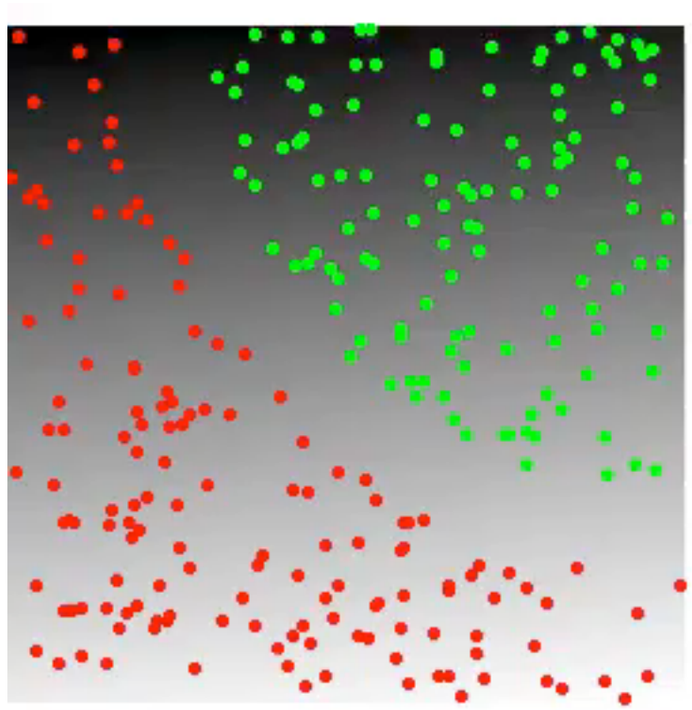
- **training**

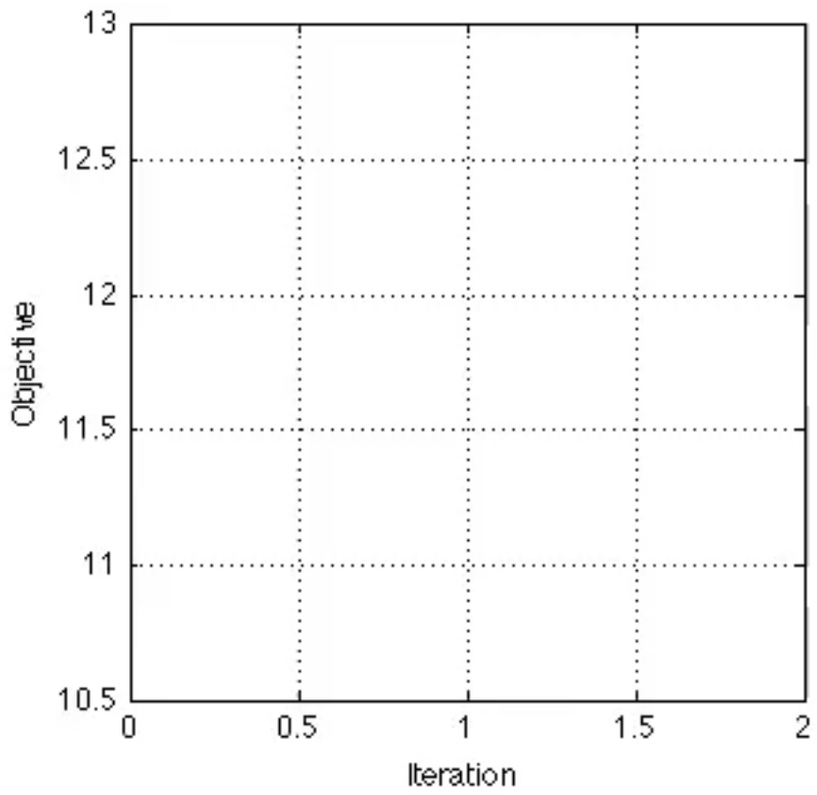
```
w = vl_pegasos(x, y, lambda)
```

- **testing**

```
w' * x
```

- Observations:
  - gets a rough solution very quickly
  - many solutions almost as good
  - testing dominated by generalization error



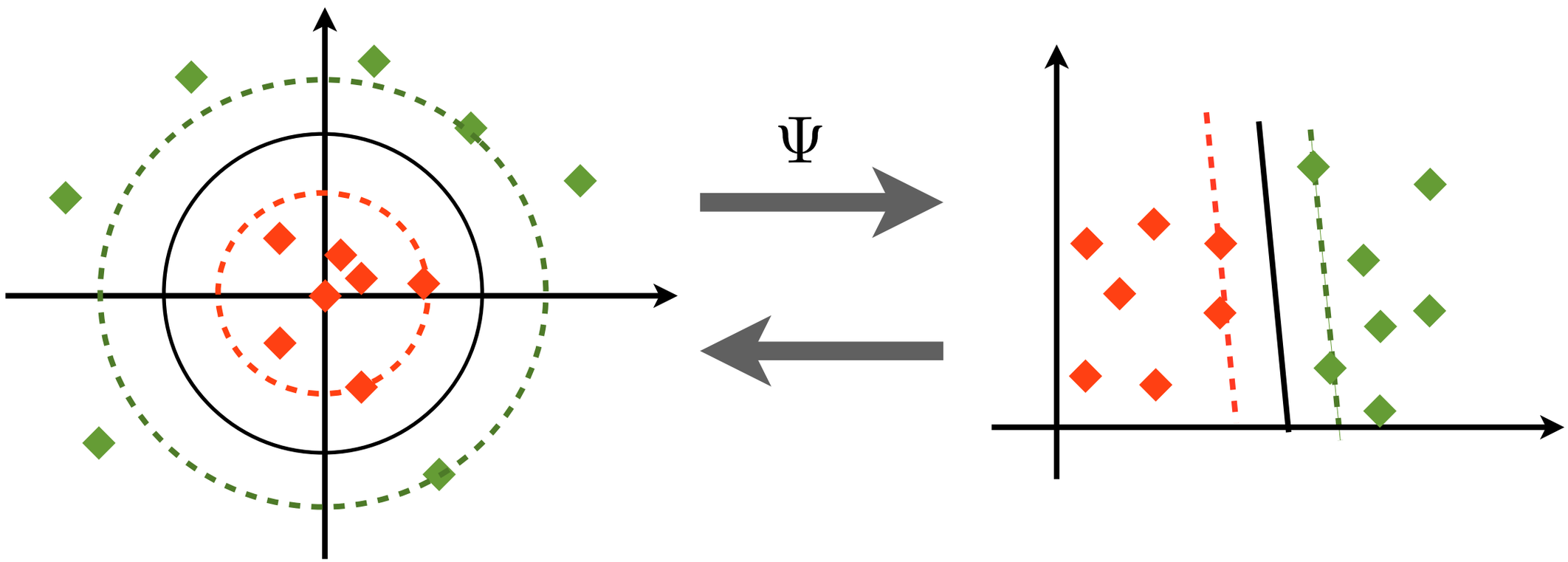


Kernel Map

$$\Psi$$

# Non-linear SVM

- Generalize SVM through a *feature map*




- The feature map may be defined implicitly by a **kernel function**

$$\mathbf{w} = \sum_{i=1}^{N} \beta_i \Psi(\mathbf{x}_i)$$

$$\langle \mathbf{w}, \Psi(\mathbf{x}) \rangle = \sum_{i=1}^{N} \beta_i \langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}) \rangle$$

$$K(\mathbf{x}_i, \mathbf{x}) = \langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}) \rangle$$

faster

more discriminative

**linear**

$$K(\mathbf{x}, \mathbf{y}) = \sum_{l=1}^{D} \mathbf{x}_l \mathbf{y}_l$$

[Zhang et al. 2007]
[Vedaldi et. al 2010]

Fast kernel and distance matrices with `vl_alldist`

**additive**

$$K(\mathbf{x}, \mathbf{y}) = \sum_{l=1}^{D} k(\mathbf{x}_l, \mathbf{y}_l)$$

**additive RBF**

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{1}{2\sigma^2}\sum_{l=1}^{D} d^2(\mathbf{x}_l, \mathbf{y}_l)\right)$$

faster

more discriminative

$$l=1$$

**additive**

$$K(\mathbf{x},\mathbf{y}) = \sum_{l=1}^{D} k(\mathbf{x}_l,\mathbf{y}_l)$$

**additive RBF**

$$K(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{1}{2\sigma^2}\sum_{l=1}^{D} d^2(\mathbf{x}_l,\mathbf{y}_l)\right)$$

**Hellinger's**

$$k(x,y) = \sqrt{xy} \qquad d^2(x,y) = (\sqrt{x}-\sqrt{y})^2$$

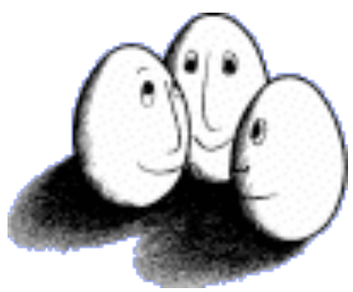
**$\chi^2$**

$$k(x,y) = \frac{2xy}{x+y} \qquad d^2(x,y) = \frac{(x-y)^2}{x+y}$$

**intersection**

$$k(x,y) = \min\{x,y\} \qquad d^2(x,y) = |x-y|$$

- **LIBSVM**
  - [C.-C. Chang and C.-J. Lin 2001]
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- **SVM*light***
  - [T. Joachims 1999]
  - http://svmlight.joachims.org



- Many other open source implementations ...

**bottleneck**
support vector expansion

# Speeding-up non-linear SVMs

- Avoid support vector expansion
- Actually compute the feature map $\Psi(\mathbf{x})$ $\quad K(\mathbf{x}, \mathbf{y}) = \langle \Psi(\mathbf{x}), \Psi(\mathbf{y}) \rangle$
- Exact feature map is usually:

**hard to compute** **high dimensional**

- Seek for approximations $\widehat{\Psi}(\mathbf{x})$ $\quad K(\mathbf{x}, \mathbf{y}) \approx \langle \widehat{\Psi}(\mathbf{x}), \widehat{\Psi}(\mathbf{y}) \rangle$
  - low dimensional
  - efficient computation
  - good approximation quality

**reduced expansions**

**direct approximations**

[C. K. I. Williams and M. Seeger 2001]
[Bach and Jordan 2006]
[Bo and Sminchiescu 2009]

# VLFeat homogeneous kernel map



- [Vedaldi and Zisserman 2010]
- Closed feature maps by Fourier transform in log domain

**Hellinger's**

$$k(x,y) = \sqrt{xy} \qquad \Psi(x) = \sqrt{x}$$

**Intersection**

$$k(x,y) = \min\{x,y\} \qquad \Psi(x) = \sqrt{\frac{2x}{\pi(1+4\omega^2)}} e^{i\omega \log x}$$

**$X^2$**

$$k(x,y) = \frac{2xy}{x+y} \qquad \Psi(x) = \sqrt{x\,\mathrm{sech}(\pi\omega)} e^{i\omega \log x}$$

- Approximate by uniform sampling and truncation

$X^2$ 3× approximation

$$\widehat{\Psi}(x) = \begin{bmatrix} 0.8\sqrt{x} \\ 0.6\sqrt{x}\cos(0.6\,\log x) \\ 0.6\sqrt{x}\sin(0.6\,\log x) \end{bmatrix}$$

# VLFeat homogeneous kernel map



- `x` data (along columns)

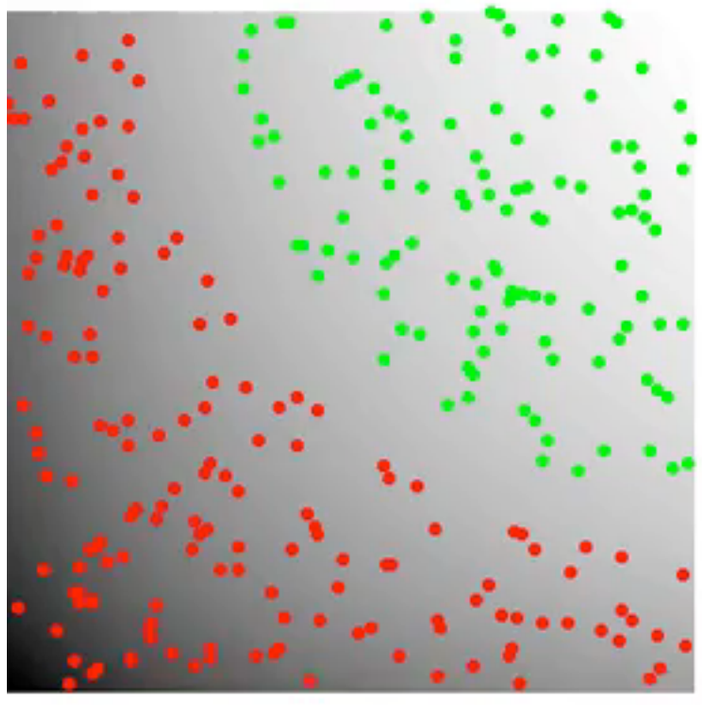
- **Linear SVM**

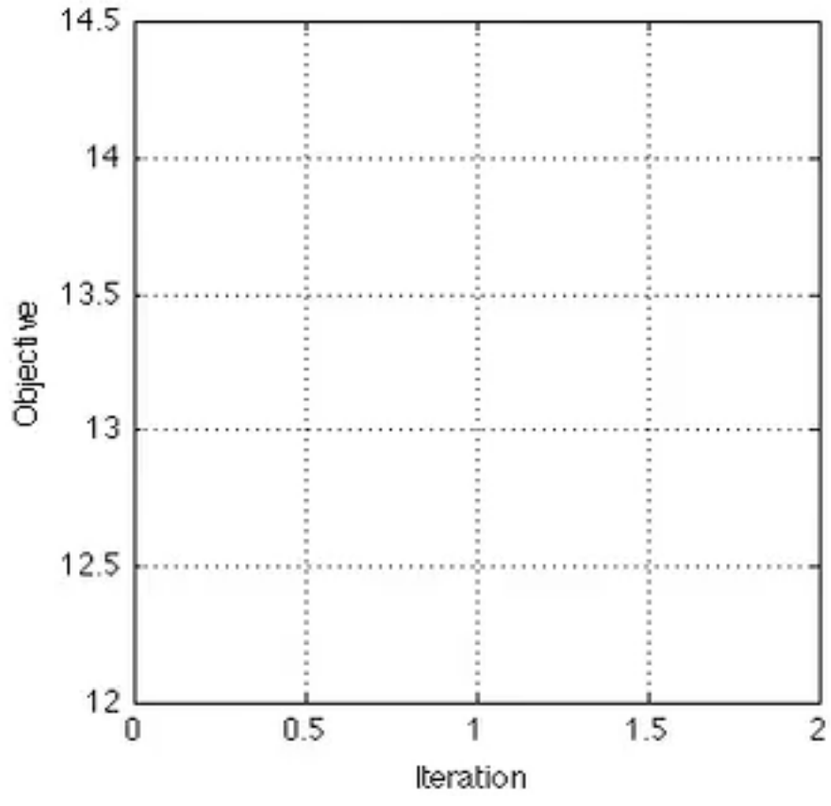
```
w = vl_pegasos(x, y, lambda)
```

- **$X^2$ SVM**

```
psix = vl_homkermap(x, 1, .6, 'kchi2') ;
w = vl_pegasos(psix, y, lambda) ;
```

- Advantages
  - universal (no training)
  - fast computation
  - high accuracy
  - on-line training and HOG like detectors





# Other direct feature maps



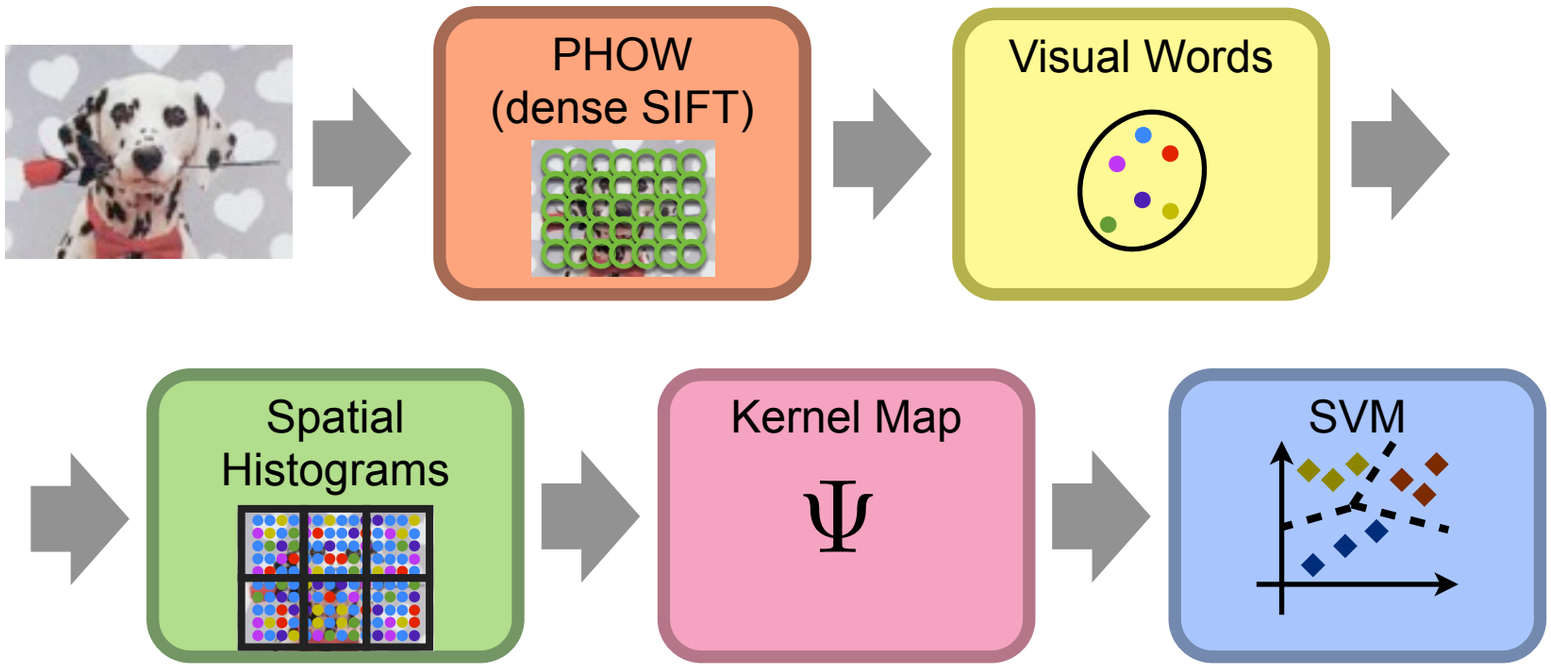
- **Explicit map for intersection kernel**
  - [Maji et al. 2008] and [Maji and Berg 2009]
  - http://www.cs.berkeley.edu/~smaji/projects/ped-detector/
  - http://www.cs.berkeley.edu/~smaji/projects/add-models/

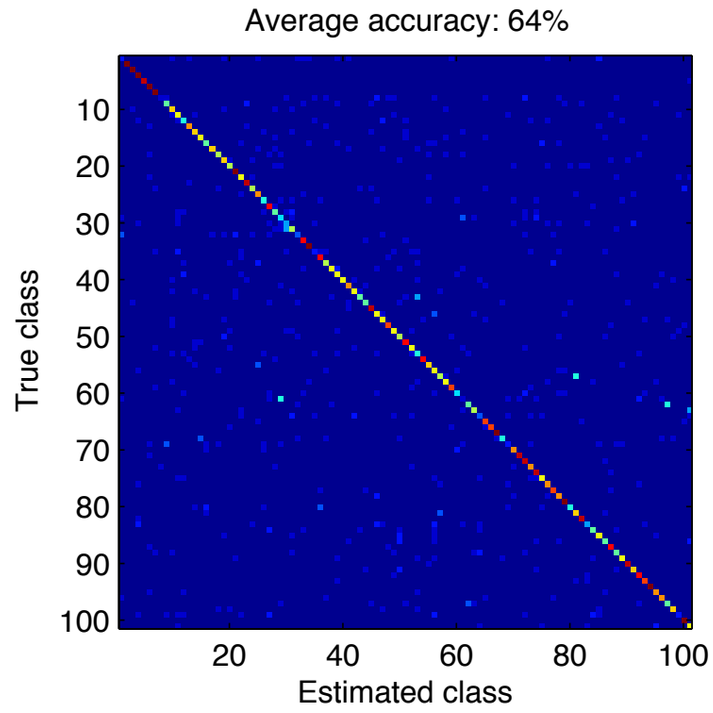
- **Additive kernel-PCA** for additive kernels
  - [Perronnin et al. 2010]

- **Random Fourier features** for RBF kernels
  - [Rahimi and Recht 2007]
  - http://people.csail.mit.edu/rahimi/random-features/

# Caltech-101 summary





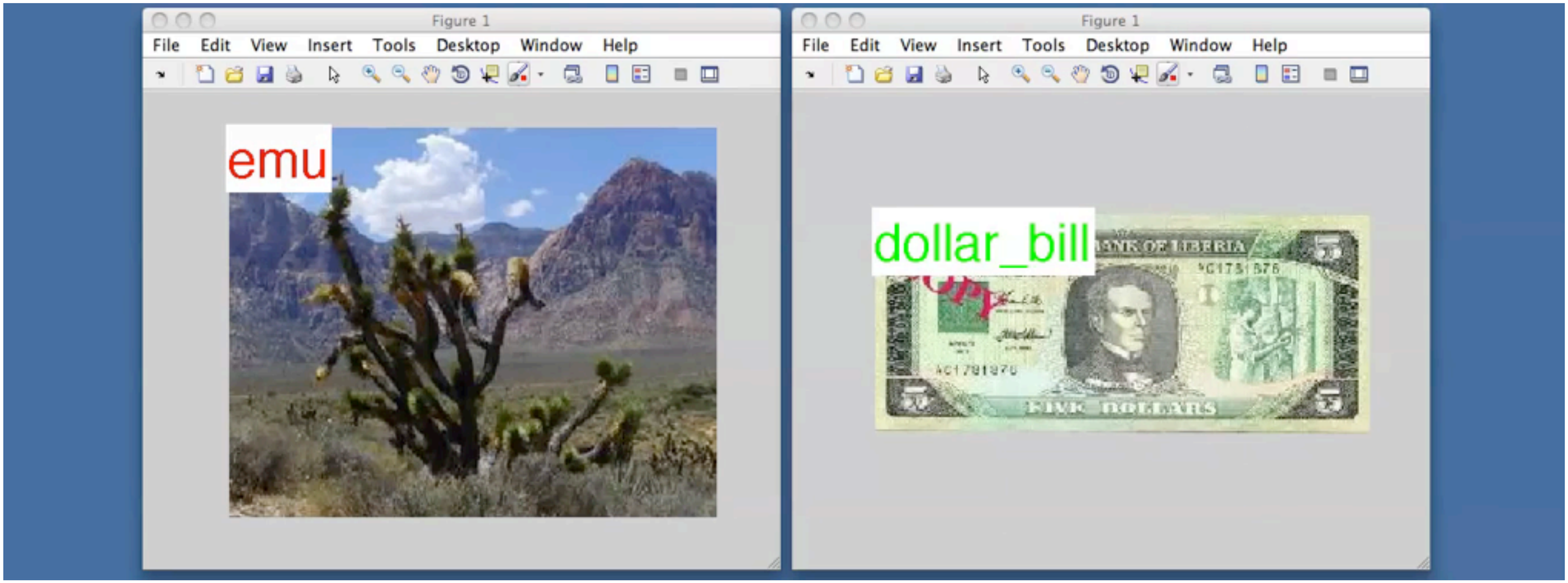




- **PHOW features**
  - Fast dense SIFT (`vl_dsift`)
- **Visual Words**
  - Elkan k-means (`vl_kmeans`)
- **Spatial Histograms**
  - Convenience functions (`vl_binsum`, `vl_binsearch`)
- **Linear SVM**
  - PEGASOS (`vl_pegasos`)
- **$X^2$ SVM**
  - Homogeneous kernel map (`vl_homkermap`)

# Demo

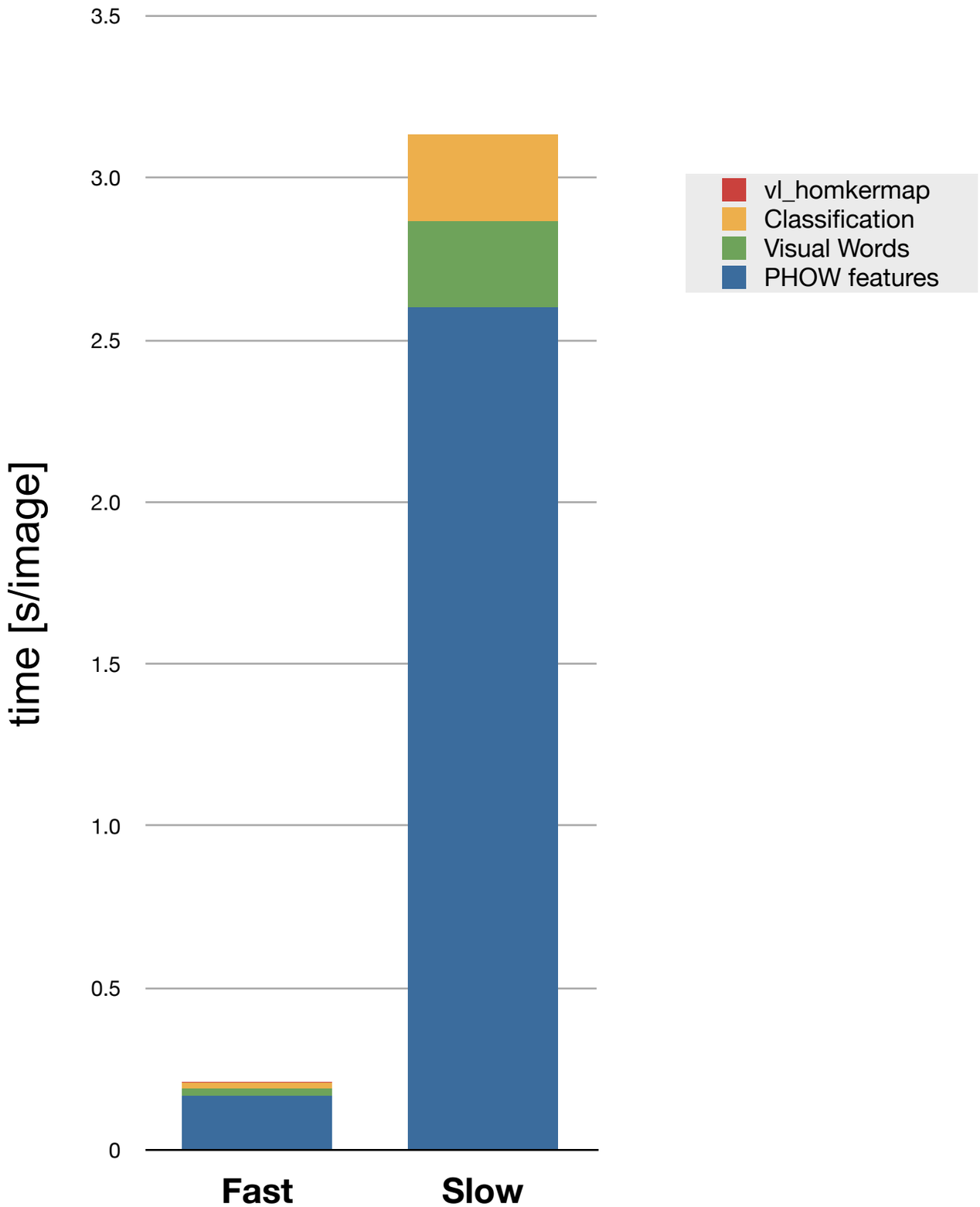


- Real time classification, 15 training images, 102 classes, 64% accuracy
- **Left**
  standard dense SIFT, kernelized SVM
- **Right**
  fast dense SIFT, kdtree, homogeneous kernel map SVM

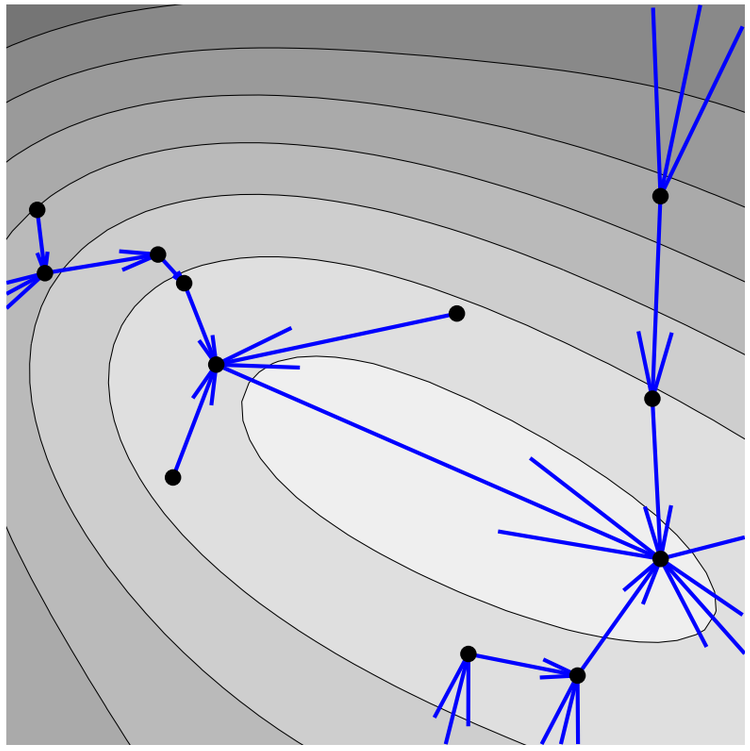


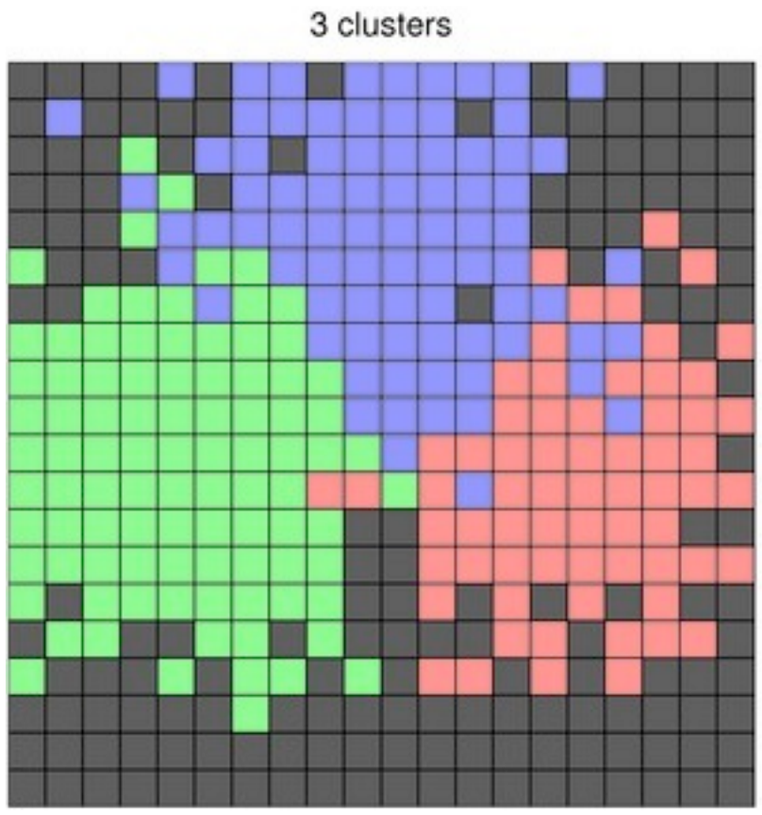
# Demo: Speedups

- Feature extraction: 15x
- Feature quantization: 12x
- Classification: 13x
- Overall: 15x



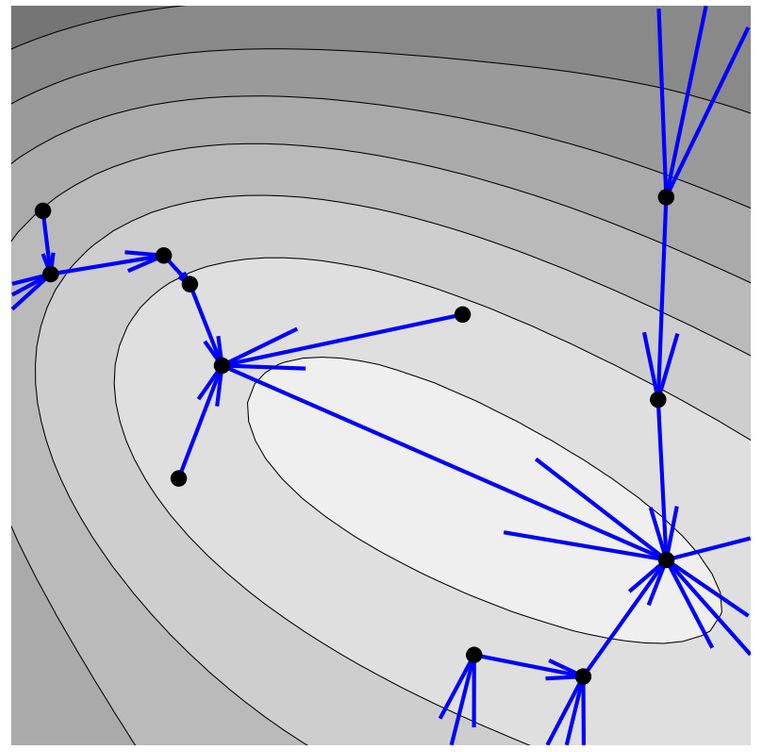
# Other features



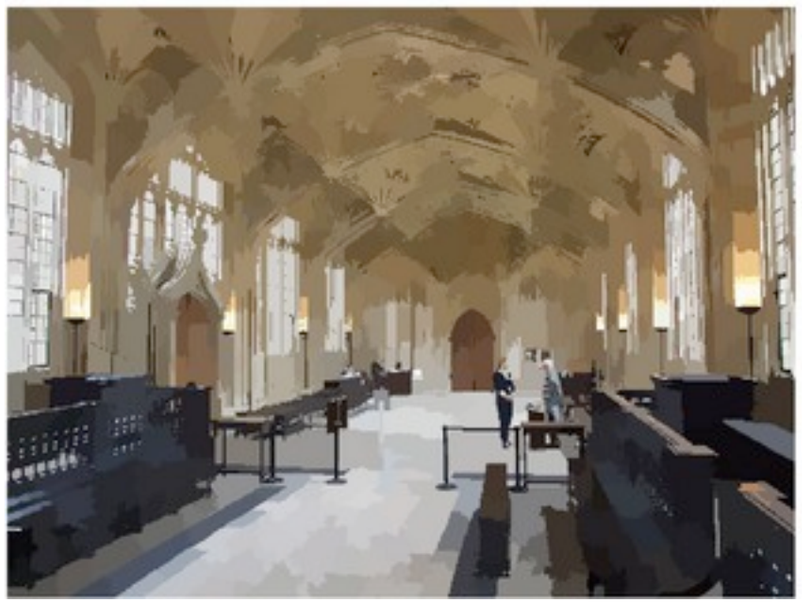

3 clusters

# Quick shift

- Mode seeking algorithm (like mean shift)
  - [Comaniciu and Meer 2002]
    [Vedaldi et. al 2008]
  - Connect to the nearest point with higher energy.
  - Break links > `maxdist` to form clusters.



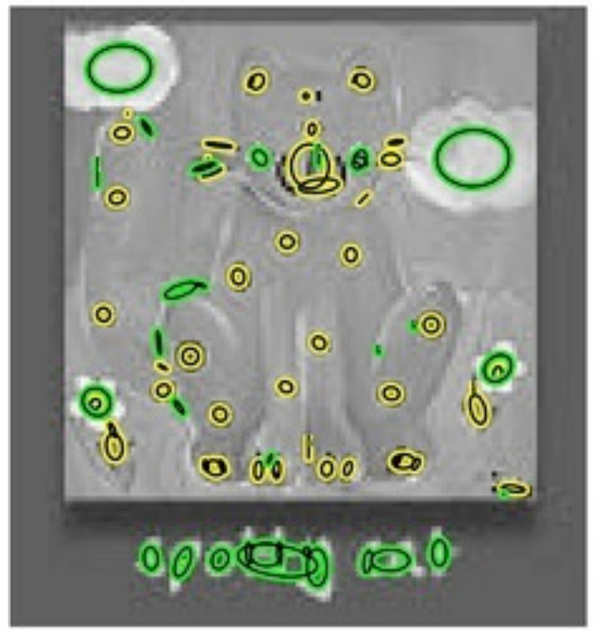
```
ratio = 0.5;
kernelsize = 2;
Iseg = vl_quickseg(I, ratio, kernelsize, maxdist) ;
```



maxdist = 10;

maxdist = 20;

# Maximally Stable Extremal Regions (MSER)

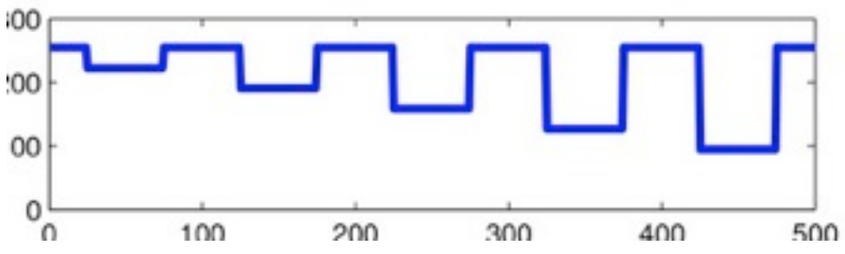
- Extracts MSER regions `regions` and fitted ellipses `frames`:

```
Ig = uint8(rgb2gray(I)) ;
[regions, frames] = vl_mser(Ig) ;
```

[Matas et al. 2002]

- Control region stability:

```
[regions, frames] = vl_mser(Ig, ...
                    'MinDiversity', 0.7, ...
                    'MaxVariation', 0.2, ...
                    'Delta', 10) ;
```



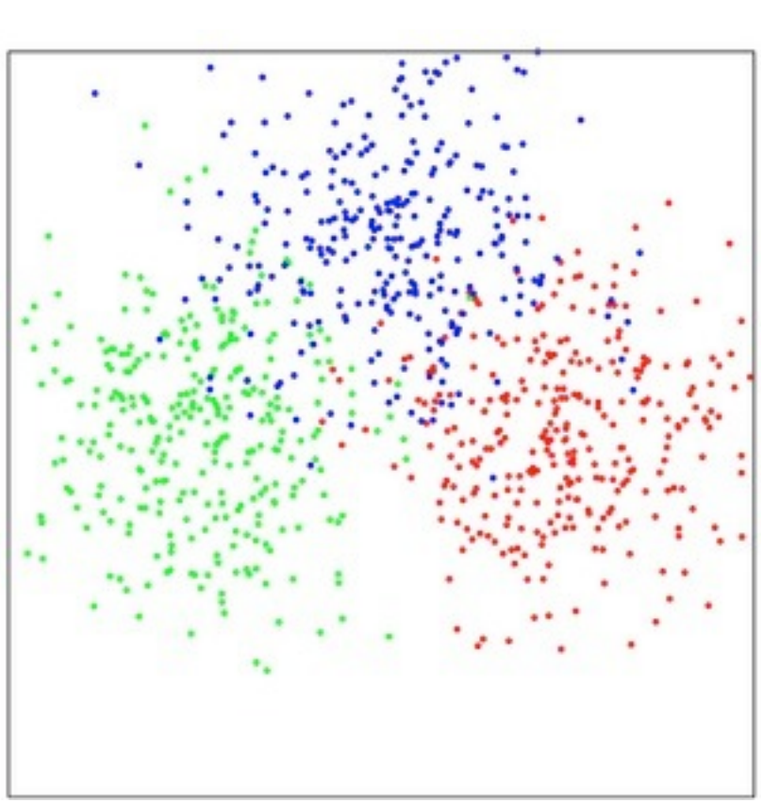

# Agglomerative Information Bottleneck (AIB)

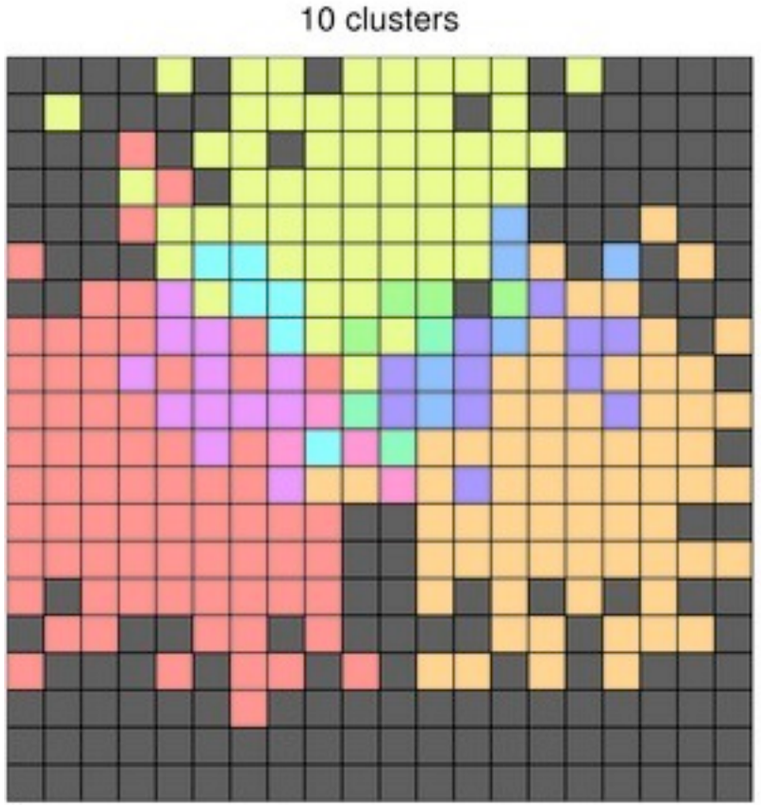
- Merge (visual) words while preserving information
  - given word-class co-occurrence matrix `Pcx`

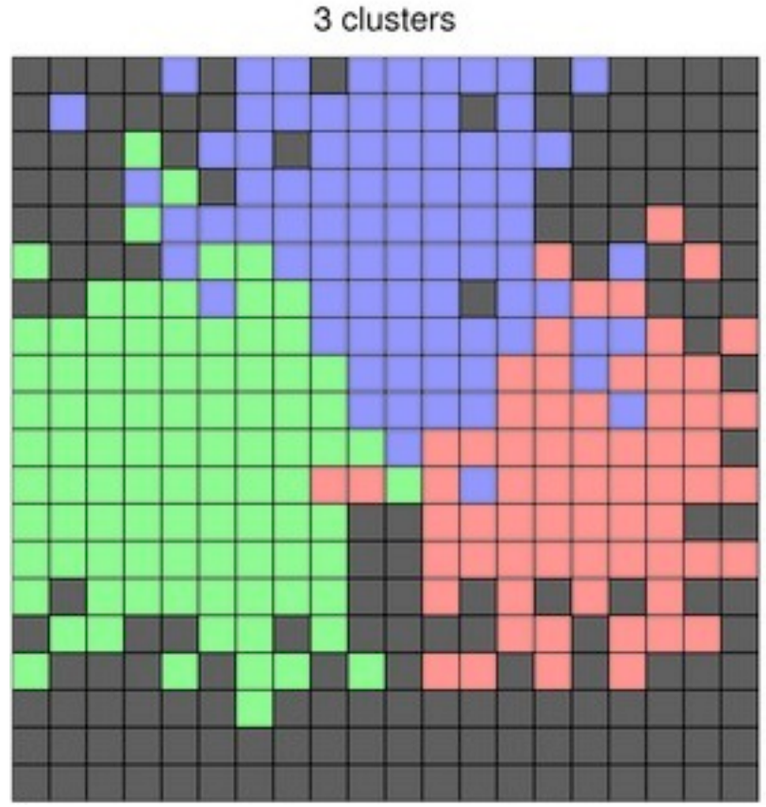
```
[parents, cost] = vl_aib(Pcx) ;
```

  - produces a tree of merged words
  - tree cut
    = partition of words
    = simplified dictionary

[Slonim and Tishby 1999]
[Fulkerson et al. 2008]





# Fast image distance transform



- [Felzenszwalb and Huttenlocher 2004]

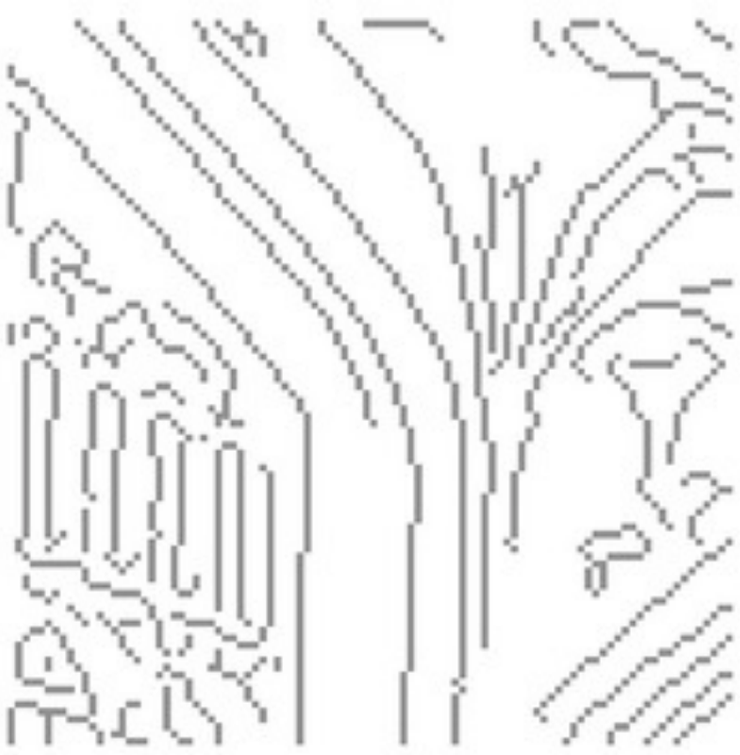
- Compute feature responses (e.g. edges):

```
edges = zeros(imsize) + inf ;
edges(edge(rgb2gray(im), 'canny')) = 0 ;
```

- For each point, find `nearest` edge and its `distance`:

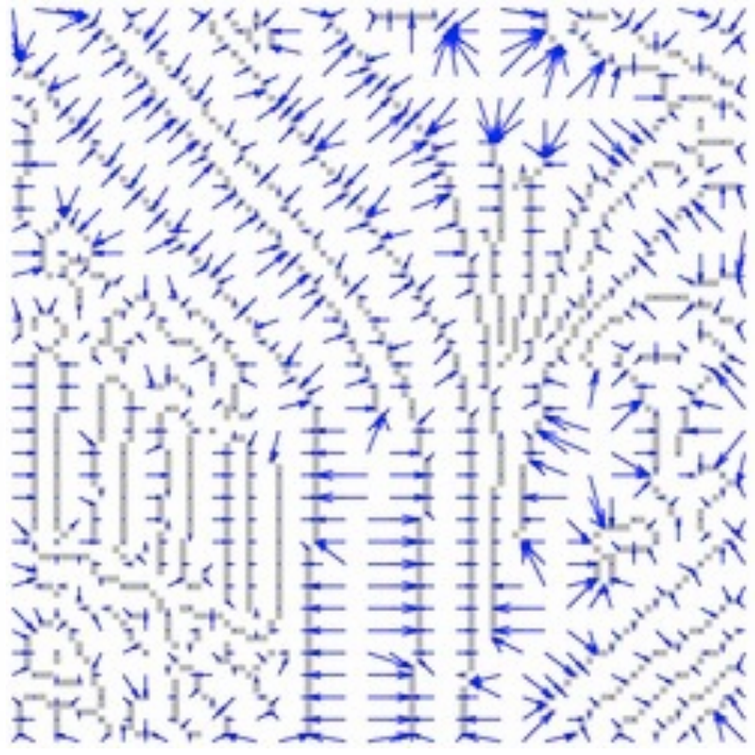
```
[distance, nearest] = vl_imdisttf(single(edges));
```

edges



distance

nearest



- Built-in MATLAB help

```
help vl_sift
```

```
VL_SIFT  Scale-Invariant Feature Transform
  F = VL_SIFT(I) computes the SIFT frames [1] (keypoints) F of the
  image I. I is a gray-scale image in single precision. Each column
  of F is a feature frame and has the format [X;Y;S;TH], where X,Y
  is the (fractional) center of the frame, S is the scale and TH is
  the orientation (in radians).

  [F,D] = VL_SIFT(I) computes the SIFT descriptors [1] as well. Each
  column of D is the descriptor of the corresponding frame in F. A
  descriptor is a 128-dimensional vector of class UINT8.

  VL_SIFT() accepts the following options:

  Octaves:: [maximum possible]
    Set the number of octave of the DoG scale space.

  Levels:: [3]
    Set the number of levels per octave of the DoG scale space.

  FirstOctave:: [0]
    Set the index of the first octave of the DoG scale space.

  PeakThresh:: [0]
    Set the peak selection threshold.

  EdgeThresh:: [10]
    Set the non-edge selection threshold.

  NormThresh:: [-inf]
    Set the minimum l2-norm of the descriptors before
    normalization. Descriptors below the threshold are set to zero.

  Magnif:: [3]
    Set the descriptor magnification factor. The scale of the
    keypoint is multiplied by this factor to obtain the width (in
    pixels) of the spatial bins. For instance, if there are there
    are 4 spatial bins along each spatial direction, the
    ``side'' of the descriptor is approximatively 4 * MAGNIF.

  WindowSize:: [2]
    Set the variance of the Gaussian window that determines the
    descriptor support. It is expressend in units of spatial
    bins.
```

- Also available as HTML

http://www/vlfeat.org/doc/mdoc/VL_SIFT.html

Index | Prev | Next

F = VL_SIFT(I) computes the SIFT frames [1] (keypoints) F of the image I. I is a gray-scale image in single precision. Each column of F is a feature frame and has the format [X;Y;S;TH], where X,Y is the (fractional) center of the frame, S is the scale and TH is the orientation (in radians).

[F,D] = VL_SIFT(I) computes the SIFT descriptors [1] as well. Each column of D is the descriptor of the corresponding frame in F. A descriptor is a 128-dimensional vector of class UINT8.

VL_SIFT() accepts the following options:

**Octaves** [maximum possible]
Set the number of octave of the DoG scale space.

**Levels** [3]
Set the number of levels per octave of the DoG scale space.

**FirstOctave** [0]
Set the index of the first octave of the DoG scale space.

**PeakThresh** [0]
Set the peak selection threshold.

**EdgeThresh** [10]
Set the non-edge selection threshold.

**NormThresh** [-inf]
Set the minimum l2-norm of the descriptors before normalization. Descriptors below the threshold are set to zero.

**Magnif** [3]
Set the descriptor magnification factor. The scale of the keypoint is multiplied by this factor to obtain the width (in pixels) of the spatial bins. For instance, if there are there are 4 spatial bins along each spatial direction, the ``side'' of the descriptor is approximatively 4 * MAGNIF.

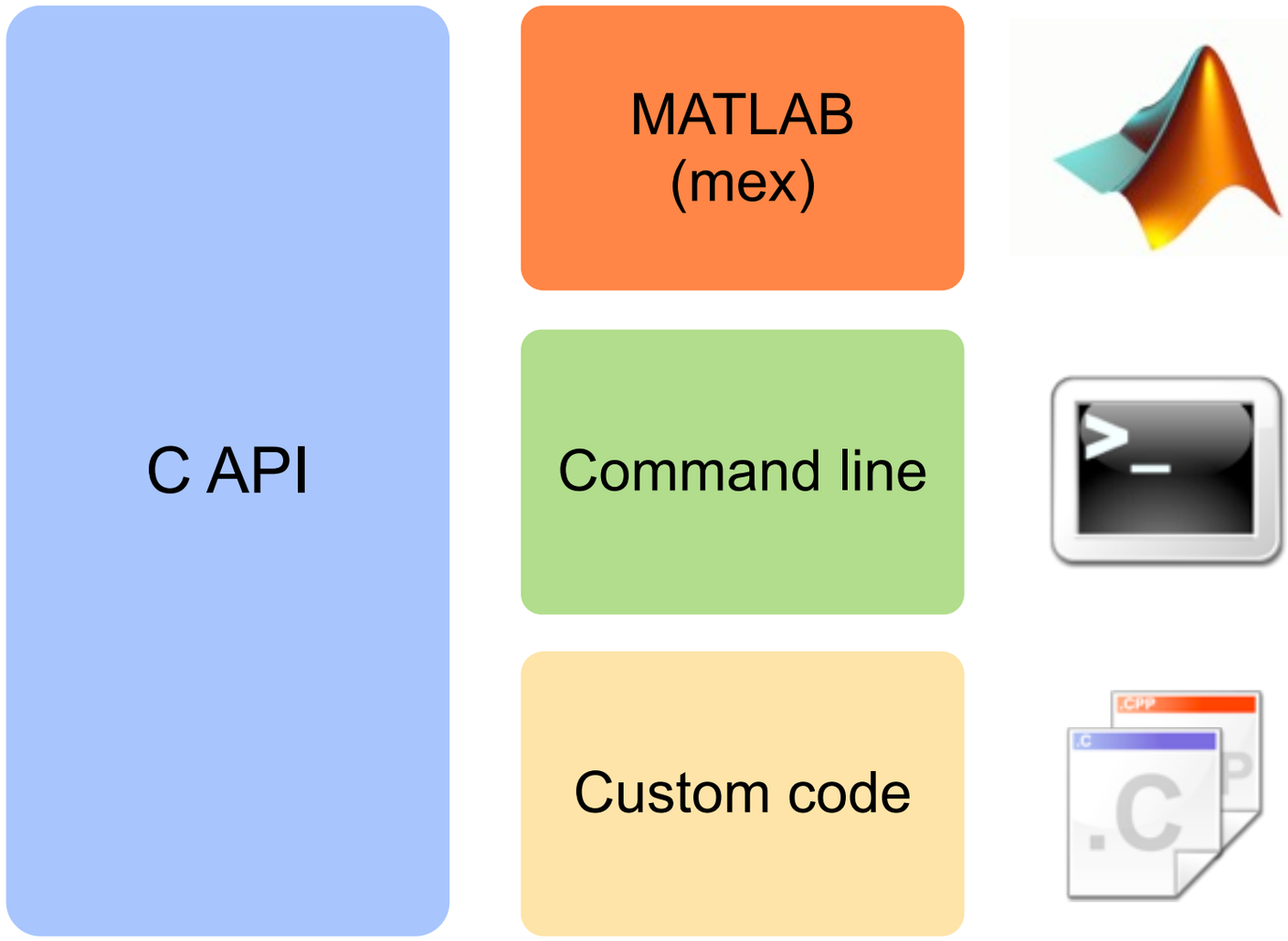
**WindowSize** [2]
Set the variance of the Gaussian window that determines the descriptor support. It is expressend in units of spatial bins.

**Frames** [not specified]
If specified, set the frames to use (bypass the detector). If frames are not passed in order of increasing scale, they are re-orderded.

**Orientations**

# VLFeat architecture

- Components




- No dependencies
- C implementation

- Full algorithm details
  - embedded in source code
  - Doxygen format **http://www.doxygen.org**

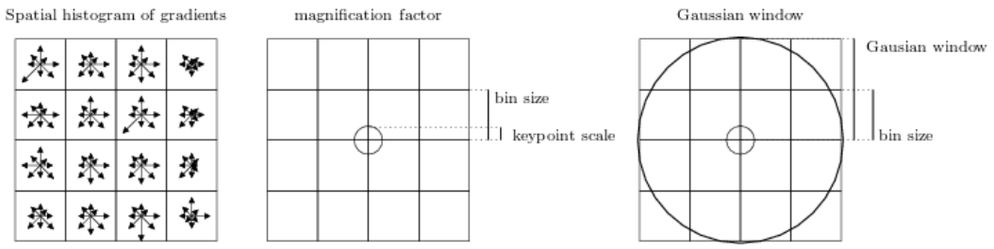
| | | | |
|---|---|---|---|
| first octave index | detector | vl_sift_new | features |
| number of scale levels per octave | SIFT detector | vl_sift_new | can affect the number of extracted keypoints |
| edge threshold | SIFT detector | vl_sift_set_edge_thresh | decrease to eliminate more keypoints |
| peak threshold | SIFT detector | vl_sift_set_peak_thresh | increase to eliminate more keypoints |

Summary of the parameters influencing the SIFT detector.

### SIFT Descriptor

**See also:**
Descriptor technical details

A SIFT descriptor is a 3-D spatial histogram of the image gradients in characterizing the appearance of a keypoint. The gradient at each pixel is regarded as a sample of a three-dimensional elementary feature vector, formed by the pixel location and the gradient orientation. Samples are weighed by the gradient norm and accumulated in a 3-D histogram *h*, which (up to normalization and clamping) forms the SIFT descriptor of the region. An additional Gaussian weighting function is applied to give less importance to gradients farther away from the keypoint center. Orientations are quantized into eight bins and the spatial coordinates into four each, as follows:

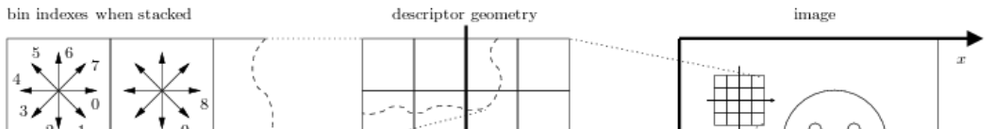



**The SIFT descriptor is a spatial histogram of the image gradient.**

SIFT descriptors are computed by either calling vl_sift_calc_keypoint_descriptor or vl_sift_cal_keypoint_descriptor_raw. They accept as input a keypoint frame, which specifies the descriptor center, its size, and its orientation on the image plane. The following parameters influence the descriptor calculation:

- **magnification factor**. The descriptor size is determined by multiplying the keypoint scale by this factor. It is set by vl_sift_set_magnif.
- **Gaussian window size**. The descriptor support is determined by a Gaussian window, which discounts gradient contributions farther away from the descriptor center. The standard deviation of this window is set by vl_sift_set_window_size and expressed in unit of bins.

VLFeat SIFT descriptor uses the following convention. The *y* axis points downwards and angles are measured clockwise (to be consistent with the standard image convention). The 3-D histogram (consisting of $8 \times 4 \times 4 = 128$ bins) is stacked as a single 128-dimensional vector, where the fastest varying dimension is the orientation and the slowest the *y* spatial coordinate. This is illustrated by the following figure.




vl_kmeans_get_centers to obtain the `numCluster` cluster centers. Use vl_kmeans_push to quantize new data points.

### Initialization algorithms

kmeans.h supports the following cluster initialization algorithms:

- **Random data points** (vl_kmeans_init_centers_with_rand_data) initialize the centers from a random selection of the training data.
- **k-means++** (vl_kmeans_init_centers_plus_plus) initialize the centers from a random selection of the training data while attempting to obtain a good coverage of the dataset. This is the strategy from [1].

### Optimization algorithms

kmeans.h supports the following optimization algorithms:

- **Lloyd** [2] (VlKMeansLloyd). This is the standard k-means algorithm, alternating the estimation of the point-to-cluster memebrship and of the cluster centers (means in the Euclidean case). Estimating membership requires computing the distance of each point to all cluster centers, which can be extremely slow.
- **Elkan** [3] (VlKMeansElkan). This is a variation of [2] that uses the triangular inequality to avoid many distance calculations when assigning points to clusters and is typically much faster than [2]. However, it uses storage proportional to the square of the number of clusters, which makes it unpractical for a very large number of clusters.

## Technical details

Given data points $x_1, \ldots, x_n \in R^d$, k-means searches for $k$ vectors $c_1, \ldots, c_n \in R^d$ (cluster centers) and a function $\pi : \{1, \ldots, n\} \to \{1, \ldots, k\}$ (cluster memberships) that minimize the objective:

$$E(c_1, \ldots, c_n, \pi) = \sum_{i=1}^{n} d^2(x_i, c_{\pi(i)})$$

A simple procedure due to Lloyd [2] to locally optimize this objective alternates estimating the cluster centers and the membeship function. Specifically, given the membership function $\pi$, the objective can be minimized independently for eac $c_k$ by minimizing

$$\sum_{i:\pi(i)=k} d^2(x_i, c_k)$$

For the Euclidean distance, the minimizer is simply the mean of the points assigned to that cluster. For other distances, the minimizer is a generalized average. For instance, for the $l^1$ distance, this is the median. Assuming that computing the average is linear in the number of points and the data dimension, this step requires $O(nd)$ operations.

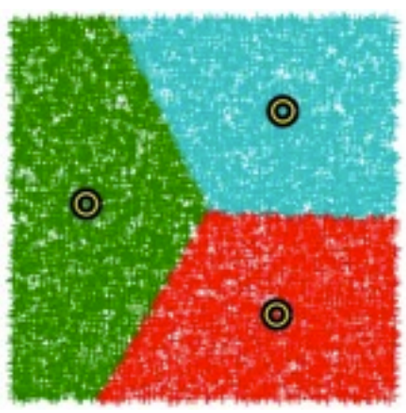
# Summary

open source

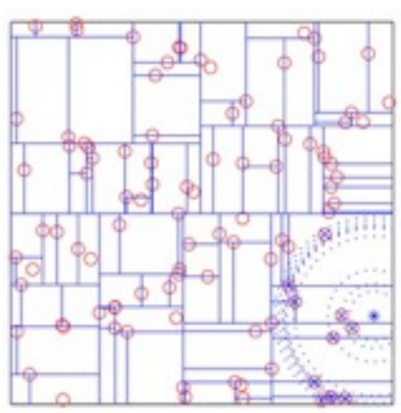
VLFeat.org



,



,



...

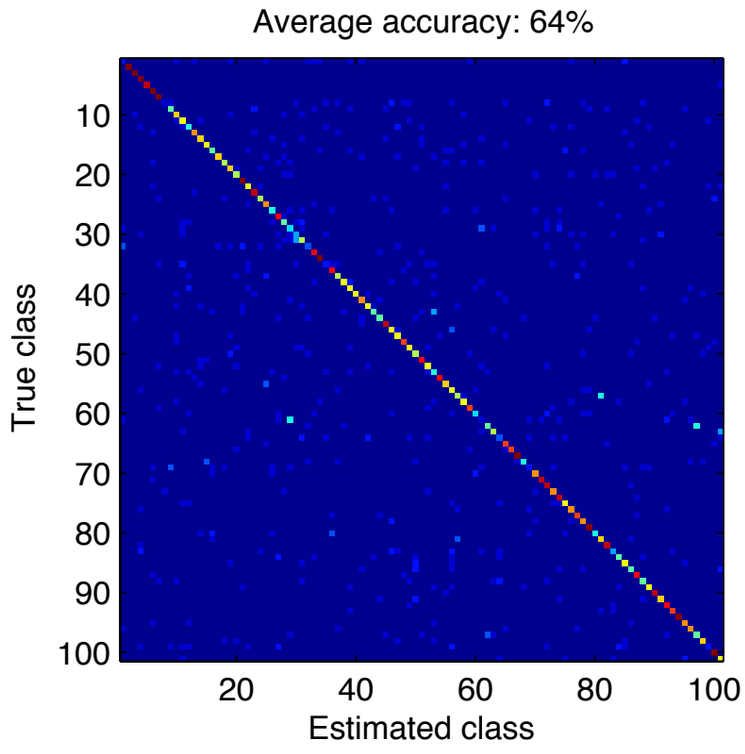
features clustering matching

C API

MATLAB (mex)

Command line

Custom code



# References




F. R. Bach and M. I. Jordan. Predictive low-rank decomposition for kernel methods. In *ICML*, 2005.

H. Bay, A. Ess, T. Tuytelaars, and L. van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 2008.

L. Bo and C. Sminchisescu. Efficient match kernels between sets of features for visual recognition. In *Proc. NIPS*, 2009.

A. Bosch, A. Zisserman, and X. Muñoz. Scene classification via pLSA. In *Proc. ECCV*, 2006.

A. Bosch, A. Zisserman, and X. Muñoz. Image classification using random forests and ferns. In *Proc. ICCV*, 2007.

C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *PAMI*, 24(5), 2002.

G. Csurka, C. R. Dance, L. Dan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Proc. ECCV Workshop on Stat. Learn. in Comp. Vision*, 2004.

C. Elkan. Using the triangle inequality to accelerate *k*-means. In *Proc. ICML*, 2003.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9, 2008.

L. Fei-Fei, R. Fergus, and P. Perona. A Bayesian approach to unsupervised one-shot learning of object categories. In *Proc. ICCV*, 2003.

P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell University, 2004.

B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315, 2007.

J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 1977.

B. Fulkerson, A. Vedaldi, and S. Soatto. Localizing objects with smart dictionaries. In *Proc. ECCV*, 2008.

T. Hastie. Support vector machines, kernel logistic regression, and boosting. Lecture Slides, 2003.

T. Joachims. Making large-scale support vector machine learning practical. In *Advances in kernel methods: support vector learning*, pages 169–184. MIT Press, Cambridge, MA, USA, 1999.

T. Joachims. Training linear SVMs in linear time. In *Proc. KDD*, 2006.

S. Lazebnik, C. Schmid, and J. Ponce. Beyond bag of features: Spatial pyramid matching for recognizing natural scene categories. In *Proc. CVPR*, 2006.

B. Leibe, K. Micolajckzyk, and B. Schiele. Efficient clustering and matching for object class recognition. In *Proc. BMVC*, 2006.

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2(60):91–110, 2004.

S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In *Proc. ICCV*, 2009.

D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *Proc. CVPR*, 2006.

F. Perronnin, J. Sánchez, and Y. Liu. Large-scale image categorization with explicit data embedding. In *Proc. CVPR*, 2010.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Proc. NIPS*, 2007.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.

S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. *MBP*, 2010.

J. Shawe-Taylor and N. Cristianini. *Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Gen. Gpu-based video feature tracking and matching. In *Workshop on Edge Computing Using New Commodity Architectures*, 2006.

J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proc. ICCV*, 2003.

N. Slonim and N. Tishby. Agglomerative information bottleneck. In *Proc. NIPS*, 1999.

E. Tola, V. Lepetit, and P. Fua. DAISY: An efficient dense descriptor applied to wide-baseline stereo. *PAMI*, 2010.

A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *Proc. ECCV*, 2008.

C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Proc. NIPS*, 2001.