KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculteit
Ingenieurswetenschappen

Object detection using the chains model

Eduardo Pérez Pellitero

Thesis submitted for the degree
of Master of Engineering:
Electrical Engineering

Thesis supervisor:
Prof. Dr. Ir. Luc Van Gool

Assessor:
Dr. Ir. Rodrigo Benenson

Academic year 2011 – 2012

Contents

Abstract

This thesis explores the use of the chains model for object detection in still images. The method presents a new way of detecting objects: The chains model do not capture the geometric relations between features through a semi-rigid model nor through an articulated model fitted to the most frequent shape of the training set. It does it through a new non-parametric probabilistic model which can handle complex non-rigid configurations. It builds an ensemble of feature chains and detects the object by marginalizing over them. The chains test for connectivity within the object to detect, having each of the chains a voting weight based on both the local appearance and the local context of the patch. This is achieved by building the chains for the testing image out of sub-chains extracted from the training examples, allowing to generate complex chains, even that ones which include uncommon poses or deformations.

The chains model was previously described as a part detector starting from a previously known reference location, nevertheless in this thesis we study and implement the case of object detection without any reference part.

List of Figures

List of Tables

List of Abbreviations

CART	Classification and Regression Trees
CVPR	Computer Vision and Pattern Recognition
EER	Equal Error Rate
FANN	Fast Approximate Nearest Neighbor
FLANN	Fast Library for Approximate Nearest Neighbors
HOG	Histogram Oriented Gradients
IJCV	International Journal of Computer Vision
LoG	Laplacian-of-Gaussian
MAP	Maximum a posteriori
ML	Maximum Likelihood
NBNN	Naive-Bayes Nearest Neighbor
NN	Nearest Neighbor
SIFT	Scale Invariant Feature Transform

Chapter 1

Introduction

Object detection is a topic within Computer Vision which deals with recognizing and localizing instances of determined classes of objects (e.g. humans, cars, horses) in images or video. Although for the human visual system this is a natural process, it remains a challenging task for computers.

This thesis focuses on part-based object detection using as a source of information not only the appearance of the parts themselves, but also the context supplied by other surrounding parts (either inside or outside the object to detect), in opposition to the most used approach of recognizing the object using as a source of information only the appearance of the object itself. This approach, known as *The chains model for detecting parts by their context* [15] present several advantages, some of the most remarkable ones are allowing to recognize objects that present large deformations, or making possible object detection in low resolution images.

1.1 Challenges in object detection

Automatic object detection is a difficult task. Schneiderman made a list of the main important challenges in object detection [27] :

1. **Intra-class variations:** Object classes have some characterizing set of features shared among all the objects belonging to one class. Nevertheless, within the very same class there can be differing features that can make the appearance of the object vary from object to object, yet belonging to the same class (e.g. a car can vary in color, shape, size, small details such as front lights, number of doors or tires thickness).

2. **Object orientation and distance:** The object can be at different distances and orientations from the camera. That might change significantly the scale and appearance of the object in the 2D representation (image), i.e. an object might look different from the front or from the sides.

3. **Geometric ambiguity:** Some information is lost in the translation from the 3D world to the 2D world.

4. **Object pose:** Some classes can present different shapes, i.e. they are articulated; like animals or humans. They present an extra difficulty in detection since the variations of the object shape can be vast and notable (e.g. a human sited does not look like a human standing).

5. **Occlusion:** The object to detect can be partially occluded by some other object.

6. **Illumination:** Pixels intensity depends on many factors in the environment. It depends on the light source: its locations, color, intensity. It also depends on the surrounding objects, since some of them may cast a shadow on the object or reflect some light on to it.

7. **Scalability:** When doing object detection, the quantity of information the detector has to deal with might be huge, so some problems might appear for multiclass detectors or for time constrained detection.

1.2 Motivation

As we have seen, there are several challenges in object detection. Existing methods provide solutions to some of them: *Histogram Oriented Gradients (HOG) template based* detectors, as the one proposed by Dalal et al. [7] are, roughly, a template matching technique which only works for object classes which can be represented by a template. If the variability of the appearance of the class is high (i.e. intra-class, orientation, deformation, distance and pose variations, occlusions) is complicate to use successfully a HOG template. Another approach would be the part-based models, which rely in detecting objects by having repetitive specific structures (parts) with an spatial consistent configuration. The simplest part-based methods do not necessarily model geometry: the *bag of features* approach [4] retrieves just a cluster of important features, giving a coarse estimation of the object location. More complex part-based models like the *constellation model* [11] or the *star model* [10] include not only the parts, but their connectivity between each other. Constellation model parts are fully connected with each other, so when number of parts increase computational cost can be prohibitive. Star model considers a reference part which the rest of the parts are connected to. These two models are particularly suitable for semi-rigid objects and they are not designed to achieve part detection. *Articulated models* [2] methods need a previously known model of the object deformation which means that they may come across problems when dealing with complex deformations non covered with the training examples.

The approach presented in this thesis, the *chains model detector*, differs from previous approaches in several aspects. First, the chains model do not capture the geometric relations between features through a semi-rigid model (such as the star or constellation model) nor an articulated model fitted to the most frequent shape of the training set. It does it through a new non-parametric probabilistic model (the chains model), which can handle complex non-rigid configurations. This is achieved by

using probabilistic feature chains leading from some arbitrary source location to the detected location, achieving in some sort of way a parsing of the object to detect (e.g. a possible chain to detect a hand could be face-elbow-forearm-hand). This has clear advantages, since a unique model can handle both rigid and articulated parts and objects. Second, all pair of pair-wise spatial configurations of the relevant features extracted from the training images are retained and used to build the geometric model in the testing, so even rare poses or deformations are going to be represented in it. Third, it can function as a full object detector (the approach of the present thesis) or as a part detector. Since it explicitly uses the context information, the appearance of the object or part can present large deformations or even work correctly in low-resolution images.

1.3 Contributions

The chains model detector is a new recent model which has several advantages regarding other existing methods and which is highly versatile. Even though, to the best of our knowledge, since Karlinsky publication in 2010 [15] no other follow-up work has been done. Karlinsky et al. proposed in their article a formulation of the chains model specifically adapted to part based detection from a reference part (from face to hand detection), and just mentioned briefly the generic full object detection approach. This might have mislead some readers. In this thesis we will implement a chains model object detector, providing explicit step-by-step mathematical formulation for the general object chains model detector without a reference part, something not previously done in [15], and we will also try to clarify with a more extended review the functioning of the whole method and put light in performance and implementation details.

1.4 Outline

This work is structured as follows: Chapter 2 gives the basis of the SIFT features and some different ways to detect and sample them. Chapter 3 state the foundations of nearest neighbor search, which has an important paper in the chains model, making a brief review about kd-trees and randomized trees. Chapter 4 exposes how can a nearest neighbor search be used for classification purposes, presenting two bag of features methods [4, 21]. Chapter 5 presents the main chapter of the thesis: The chains model, where the details of the method and the current implementation are explained. Finally we conclude in chapter 7. Appendix 6 contains the results of the chains model and the NBNN classifier.

Chapter 2

Feature extraction and description

2.1 Introduction

Many problems in computer vision require efficient image matching techniques. One approach for the image matching is the use of the so called feature detectors and descriptors. Roughly, this feature detectors find important keypoints of the image, while the feature descriptors are nothing more than compact representations that summarize the contents of an image region . When matching images, this descriptors give a far more distinctive information than the one contained in the non-transformed pixels.

This descriptors can use several information: gradients, phase, scale, etc. which will define different types of features. The one used by [4, 21, 15] in their works are the SIFT features [19], which are going to be explained in more detail.

2.2 SIFT Detector

The acronym SIFT stands for Scale Invariant Feature Transform. In its definition it is included a detector to select important regions of the image or keypoints according to a determined criteria. This detectors keeps also the associated scale and orientation of the resulting interest points.

The procedure to detect keypoints is based on cascade filters, in such a way that the most computational expensive operations are done at the lasts filters, assuring this costly operations to be only done in the selected keypoints.

1. **Scale-space extrema detection.**
2. **Keypoint localization.**
3. **Orientation assignment.**

Starting with the original image, we filter it with a difference of Gaussian Kernel at a series of increasingly coarse scales.

The filtered images are used to compose a 3D volume of dimensions I x J x K, being I and J the vertical and horizontal size of the image and K the increasing in coarse scales.

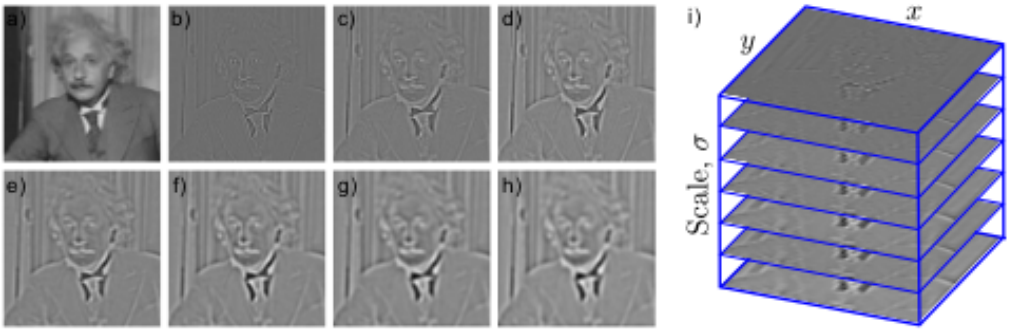


Figure 2.1: The SIFT detector. a) Original image. b-h) The image is filtered with difference of Gaussian kernels at a range of increasing scales. i) The resulting images are stacked to create a 3D volume. Points that are local extrema in the filtered image volume are considered to be candidates for interest points. *Image and footer extracted from* [26], p.340.

In this step Extrema are identified: We can define Extrema as the positions where all the voxel neighbors are either all greater or all less than the current value. This first Extrema positions will be the candidates for the keypoints, which will be refined in every step of the algorithm.

After that, in the next step we should ensure that the the selected points do not have low contrast (which are sensitive to noise). In order to do that, a local quadratic approximation is applied and the value of the peak is obtained. Lowe [19] approach uses the Taylor expansion (up to the second power term) of the scale-space function, $D(x, y, \sigma)$, shifted so that the origin is at the sample point:

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{2.1}$$

Where D and its derivatives are evaluated at the sample point and $x = (x, y, \sigma)^T$ is the offset from this point.

The function value at the extrema $D(\hat{x})$ will give information about the stability of itself. In [19], every extrema with value of $D(\hat{x})$ under 0.03 (considering pixel information from [0,1]) was discarded.

$$D(\hat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x} \tag{2.2}$$

The difference-of-gaussian function has a strong response along the edges, even if the features located there are susceptible to noise. To improve stability, we should also reject the keypoints poorly located on the edges. After this, we would have a

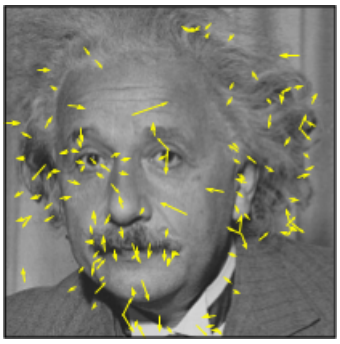


Figure 2.2: Result of SIFT detector. *Image extracted from* [26], p.341.

set of interest points with sub-pixel accuracy and with a particular scale. At this point, one orientation is computed for each interest point. To achieve that, the amplitude and orientation of the local gradients are calculated in an area of the image surrounding the interest keypoint location and whose size is proportional to the scale of that keypoint. An orientation histogram is then computed over this region with 36 bins covering all 360° of orientation. The contribution to the histogram depends on the gradient amplitude and is weighted by a Gaussian profile centered at the location of the interest point, so that nearby regions contribute more. The orientation of the interest point is assigned to be the peak of this histogram. If there is a second peak within 80% of the maximum, we may choose to compute descriptors at two orientations at this point. The final detected points are hence associated with a particular orientation and scale.

2.3 Dense Feature Sampling

The dense feature sampling is characterized by sampling in a regular way along the whole image, creating in this way a constant number of features per image region. There is no algorithm to reject or to select features, so even the features located in less-contrast regions will contribute in the same way. The idea behind the dense feature sampling is to have an overall coverage of all the image, because although features are not located in a before hand interest point (which is aim to be computed to maximize the distinctiveness of the feature), the content of a feature in a dense feature sampling can contain valuable information in order to interpret the whole image. In addition, it allows to extract a great number of features of a single image (which is also explicitly fixed and controlled by the user), which can be difficult when using feature detectors: For instance with a SIFT detector, the number of extracted features will vary a lot from image to image, depending on the content of the image itself. As an example, images with low contrast may have few interest points detected.

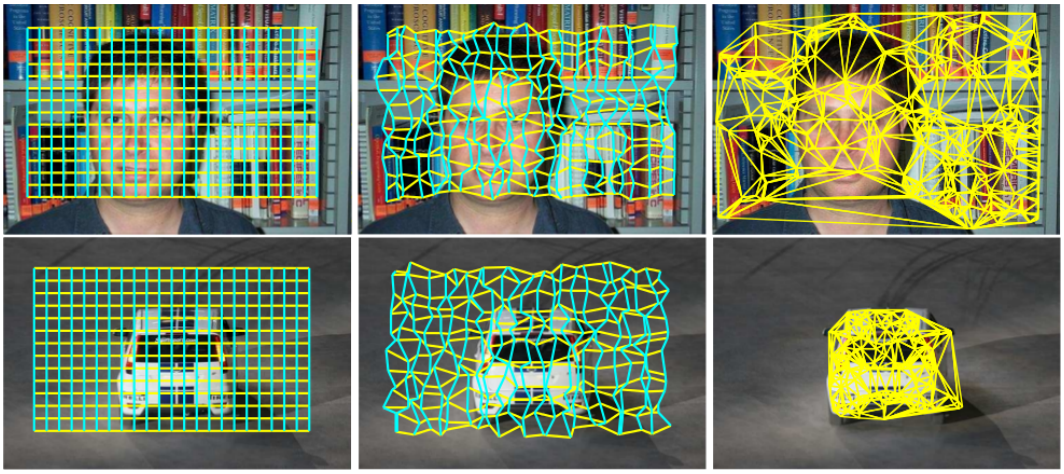


Figure 2.3: Different feature detection approaches. *Image extracted from* [30].

2.4 Dense Interest Points

The Dense Interest Points approach was first proposed by [30]. It is a hybrid scheme of dense feature sampling and interest points. It starts from dense feature sampling patches, and from that, it optimizes the localization and scale of the keypoints within a search area. In this way, the whole image is covered, and still we get benefits of the detecting algorithms. The results obtained in [30] show an improvement over dense feature sampling technique. In figure 2.3 we can see the different approaches explained.

2.5 SIFT Descriptor

The SIFT descriptor characterizes the content of an image around a given point. It is usually used with the SIFT detector, although it can work with any points provided (for instance the techniques previously mentioned in sections 2.3 and 2.4). The interest points, if computed with a SIFT detector, will include scale and rotation, which will define the square region of the image where the descriptor is going to be extracted. The main goal of the SIFT descriptor is to extract features which are distinctive, but also resilient to intensity variation, contrast changes and geometric deformations.

To extract a descriptor, the image gradient magnitudes and orientation are sampled around the keypoint location, using the scale of the keypoint to select the Gaussian blurness for the image. In order to improve the scale invariance, the coordinates of the descriptor and the gradient orientations are rotated following the keypoint orientation.

After this, a Gaussian weighting function with σ equal to one half the width of the descriptor window is used to weight each sample point. Applying this Gaussian

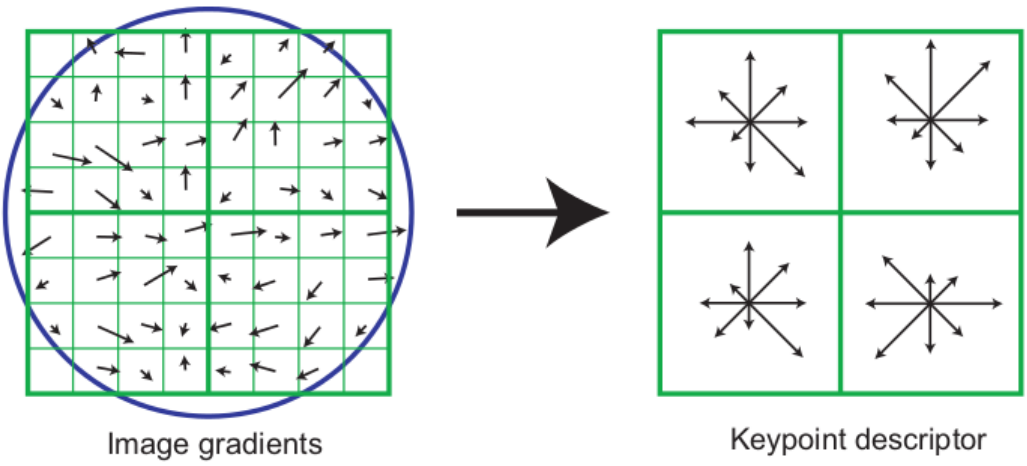


Figure 2.4: Sift Descriptor. A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over 4x4 sub-regions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 2x2 descriptor array computed from an 8x8 set of samples, whereas the experiments in this paper use 4x4 descriptors computed from a 16x16 sample array. *Image and footer extracted from* [19], p.15

function respond to several criteria: Avoiding the abrupt changes in the descriptor with small changes in the position of the windows, and also giving less importance to gradients far away from the actual center of the descriptor keypoint. The samples of the image gradients are accumulated into orientation histograms, as can be seen on the right in figure 2.4. This figure shows eight directions for each orientation histogram, with the length of each arrow representing the magnitude of that histogram entry. A gradient sample on the left can shift up to four sample positions and still contribute to the same histogram on the right, allowing thus for larger local positions shifts.

The descriptor is formed by a vector containing the values of all the orientation histogram entries. Although figure 2.4 shows a 2x2 array of orientation histograms, empirical experiments of [19] were made with 4x4 arrays and 8 possible directions for each of the histogram (4x4x8 = 128 element vector). Finally, the vector is normalized to unit length in order to improve resilience to illumination changes, since this will cancel a change in image contrast (modeled as each pixel value multiplied by a constant). For the particular case of a brightness (modeled as a constant added to each image) change will not affect the gradients since they are computed out of pixel differences.

2.6 Conclusion

SIFT descriptors are a representation of an image region. This representation is formed by a high-dimensional vector, which makes it distinctive and therefore useful for matching descriptors within a database of keypoints. The descriptors are resilient to image rotation, scale variance and have some mechanisms against added noise or changes in illumination. SIFT keypoints can be located in the image using a SIFT detector, which looks for interesting points in the image; using dense sampling, which samples regularly along all the image or using dense interest points, an hybrid technique which samples regularly but at the same time allows for some search area to optimize the location of the keypoint, among other methods. Using this last two methods allows to extract large number of keypoints, which lead to more accuracy when matching objects or regions.

SIFT features can be used for object classification and recognition methods, combined with techniques such as approximate nearest-neighbor look-ups, some of them mentioned in the chapter. Other potential applications include view matching for 3D reconstruction, motion tracking and segmentation, robot localization, image panorama assembly, epipolar calibration, and any others that require identification of matching locations between images. [19]

Chapter 3

Nearest Neighbor search

3.1 Introduction

Nearest Neighbor (NN) search is a fundamental problem in computer vision. It could be defined as an optimization problem for finding closest points in metric spaces. This problem can be formulated as:

$$X_* = \arg\min_{X \in D} \rho(X, Q) \tag{3.1}$$

Where $D = \{X_1, \ldots, X_n\} \subset \mathbb{R}$ is a dataset, Q is a query and ρ a distance measure.

Most simple and naive approach will be to compute the distance from every element inside the dataset to the query, and retrieve the nearest one. This method is called linear search and it ensures that the Nearest Neighbor retrieved is the exact one, not an approximation. The computational cost of finding the exact nearest neighbor is $O(nd)$ being n the size of the dataset and d the dimensionality of each element. This computational cost can be prohibitive for large or high dimensional datasets, so other methods to the NN finding have been developed through the years. In this chapter we are going to review briefly some of this methods, starting with an introduction to one of the most used data structures for approximate Nearest Neighbor search: the kd-trees. Furthermore we will review in a deeper way the Fast Approximate Nearest Neighbors method, which has been used for this thesis implementation of the chains model.

3.2 Decision trees

Decision trees are tree-like models which use the hierarchical structure in order to make decisions and evaluate their possible consequences. They have been around for more than twenty years, since Breiman et al. stated their basis in his CART book [5]. They are experiencing nowadays a revival due to the recent discovery that a group of slightly different trees have a much higher accuracy on previously unseen data, a phenomenon called generalization.

Although the most common applications of decision trees have been classification and regression, many other problems can be solved with decision trees, such as density estimation, manifold learning, semi-supervised learning and active learning. [6]

3.2.1 Decision trees basis

A decision tree is composed by nodes and edges. We can distinguish between internal nodes (or splits) and the terminal nodes (or leaves). Each node has just one edge coming in and, for binary trees, just two outgoing edges per internal node (external nodes do not have outgoing edges), as shown in figure 3.1 (a).

Every internal node represents a test or a question, which allows to split down a complex problem into several simpler ones. This tests or questions determine a path in the tree, going hierarchically through as many test/questions as levels in the tree until reaching a leave. The selected leave itself is an output describing a unique path within the tree.

As an example, we could think of an image introduced into the root of the tree. Every split does a testing, e.g. "Is the top part of the image blue?". Every test done give information about the content of the image, and at the end, we will be able to classify whereas it is an indoor image or an outdoor image, figure 3.1 (b).

One of the key points of building a useful decision tree is then selecting correctly the testing functions in every internal node. If the functions selected do not separate as expected the data, the whole tree will be malfunctioning. For simple cases this functions can be selected manually, but for complicated problems such as Computer Vision, the tree structure, the functions and the parameters are learnt automatically from a training stage with training datasets.

3.2.2 Tree training and testing

As said previously, for complex problems, decisions trees are usually trained in order to obtain all their parameters. A division in two stages is, then, created: the off-line phase (training) and the on-line phase (testing). A. Criminisi et al. made in [6] a brief explanation of both this steps, which is going to be exposed here.

Tree training

The off-line phase of tree training is responsible of the optimizing of all the parameters of the split functions associated with the internal nodes, as well as the leaf predictors. In order to analyze the way the training works, it is useful to divide the training data into subsets of training points associated with different tree branches. We will name S_1 the subset of training points reaching node 1, and S_1^R and S_1^L the subset going to the right and to the left child respectively, if it is a binary tree.

Starting with training set S_0 of data points $\{v\}$ and the associated ground truth labels, the tree parameters are selected in a way that minimize a chosen energy

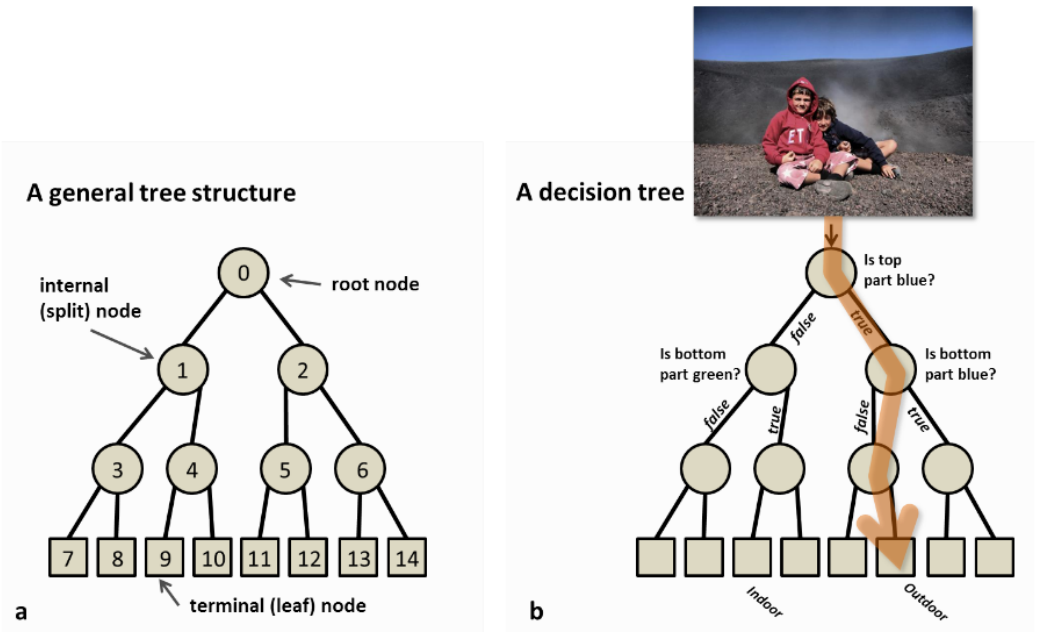
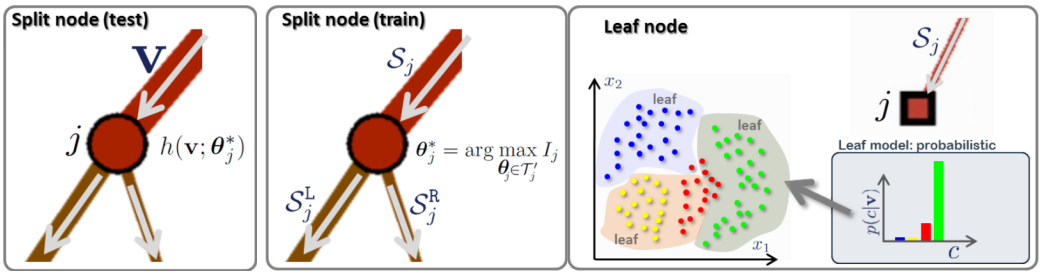


Figure 3.1: Decision tree. (a) A tree is a set of nodes and edges organized in a hierarchical fashion. In contrast to a graph, in a tree there are no loops. Internal nodes are denoted with circles and terminal nodes with squares. (b) A decision tree is a tree where each split node stores a test function to be applied to the incoming data. Each leaf stores the final answer (predictor). This figure shows an illustrative decision tree used to figure out whether a photo represents and indoor or outdoor scene. *Image and footer extracted from* [6], p.6.



(a) (b) (c)

Figure 3.2: Split and leaf nodes (a) Split node (testing). A split node is associated with a weak learner (or split function, or test function). (b) Split node (training). Training the parameters θ_j of node j involves optimizing a chosen objective function (maximizing the information gain I_j in this example). (c) A leaf node is associated with a predictor model.

function. Some predefined stopping criteria has to be fixed to decide exactly when to stop growing the tree or trees branches. For forest, i.e. systems with more than one tree, the process is usually repeated in each tree independently.

During the training, the optimal parameters of θ_j^* of the j^{th} split node. This is done by maximizing an information gain objective function.

$$\theta_j^* = \arg\max I_j \tag{3.2}$$

$$I_j = I(S_j, S_j^R, S_j^L, \theta_j) \tag{3.3}$$

The symbols S_j, S_j^R, S_j^L express the training sets before and after the split. Equation 3.3 is of an abstract form here, since the concrete definition depends on the task which the tree is going to perform (e.g. supervised or not, continuous or discrete output).

The maximization operation in equation (3.2) can be achieved simply as an exhaustive search operation. Often, finding the optimal values of τ the thresholds may be obtained efficiently by means of integral histograms.

Tree testing

Given a previously unseen data point v a decision tree hierarchically applies a number of predefined tests (see figure 3.3 (c)). Starting at the root, each split node applies its associated split function to v. Depending on the result of the binary test the data is sent to the right or left child. This process is repeated until the data point reaches a leaf node.

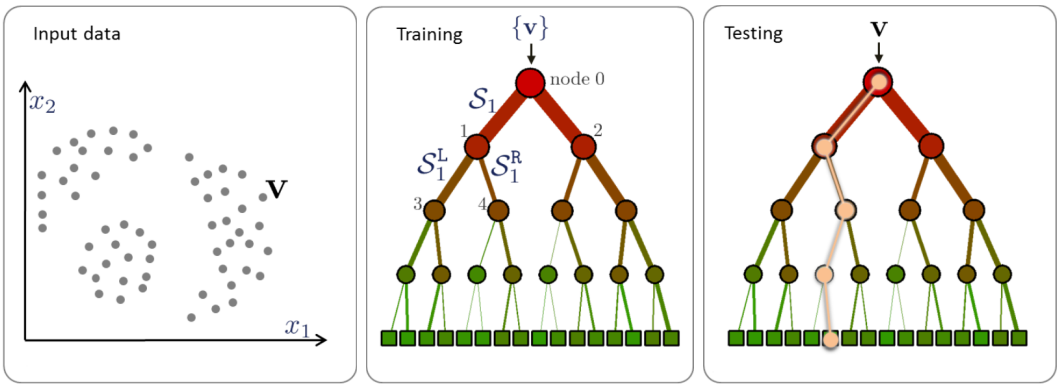
3.2.3 Randomized decision forest

A randomized decision forest is an ensemble of randomly trained trees. The randomized decision forests are characterized by some aspects: the split functions (also called weak learners), the leaf predictors, the number of trees in the forest, and the randomness model. In this section we are going to focus on this last aspect to understand better the way randomized decision forest work.

A key aspect of randomized decision forests is that all the component trees are slightly different from each other. This creates some de-correlation between the individual tree predictions and improved generalization, achieving also more robustness when dealing with noisy data. This randomness is injected to the trees during training stage: One popular way of doing it is the randomized node optimization [13], which enables us to train each tree on the totality of training data.

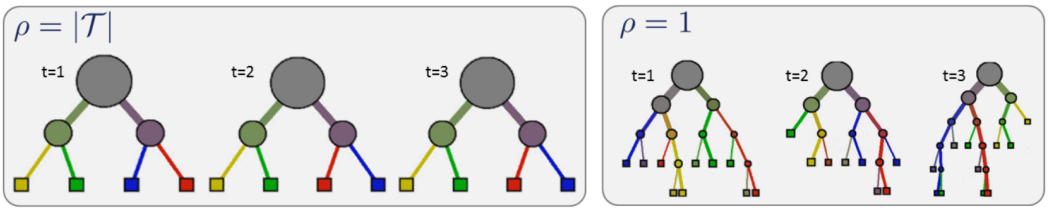
Being τ all the possible parameters θ, when training the node j-th we only make available a part of this whole set of possible parameters named $\tau_j \subset \tau$. The randomized node optimization will then optimize each j-th node as follows:

$$\theta_j^* = \arg\max_{\theta_j \in \tau_j} I_j \tag{3.4}$$



(a) (b) (c)

Figure 3.3: (a) Input data is represented as a collection of points in the d-dimensional space defined by their feature responses (2D in this example). (b) A decision tree is a hierarchical structure of connected nodes. During testing, a split (internal) node applies a test to the input data v and sends it to the appropriate child. The process is repeated until a leaf (terminal) node is reached (beige path). (c) Training a decision tree involves sending all training data {v} into the tree and optimizing the parameters of the split nodes so as to optimize a chosen energy function. *Image and footer extracted from* [6].



(a) (b)

Figure 3.4: Controlling the amount of randomness and tree correlation. (a) Large values of ρ correspond to little randomness and thus large tree correlation. In this case the forest behaves very much as if it was made of a single tree. (b) Small values of ρ correspond to large randomness in the training process. Thus the forest component trees are all very different from one another. *Image and footer extracted from* [6].

Being I_j the information gain. The amount of randomness is controlled by the ratio τ_j/τ. A parameter p should be used as $p = |\tau_j|$, and its value can be $p = 1, \ldots, |\tau|$; having identical trees (no randomness) when $p = |\tau|$ and maximum uncorrelated trees (maximum randomness) when $p = 1$ (see figure 3.4).

3.3 Approximate Nearest Neighbors

Approximate Nearest Neighbors search include all the methods which do not necessarily retrieve the exact nearest neighbors, but give in exchange some improvement in terms of time or computational cost. The phenomenon of concentration of distance would suggest that approximate nearest neighbor searching is meaningless. Fortunately, most applications present distributions which tend to be clustered in lower dimensional subspaces [20]. The idea of ANN is to take advantage of this low-dimensional clustering by using practical data structure for the nearest neighbor searching.

Mount et al. implemented an ANN library [22] which has been used by [15] and [4] in their respective detector and classifier implementations. The ANN library is based on the use of efficient approximate-r-nearest-neighbors algorithm and KD- tree implementation, whose expected time for an ANN search is logarithmic in the number of elements stored in the kd-tree. As shown previously, an important issue in the construction of tree data structure is the splitting method used, which will determine the dimension and splitting plane at each stage of decomposition. Although a deep study of split methods is beyond the scope of this chapter, some different methods have been proposed: The optimized kd-tree splitting method, the sliding-midpoint method or the minimum-ambiguity, among others.

It is also important to mention that for high dimensional queries, tree based ANN techniques are not scalable and recently hashing techniques have been studied to provide efficient solutions for such high-dimensional problems.

3.4 Fast Approximate Nearest Neighbors

The Fast Approximate Nearest Neighbors is the method used in this thesis. It was proposed by Marius Muja et al. [23] and the Fast Library for Approximate Nearest Neighbors (FLANN) is available through internet.

The Fast Approximate Nearest Neighbors uses two algorithms to retrieve approximate Nearest Neighbors: The randomized kd trees and the hierarchical k-means trees, with an automatic selection algorithm available if desired.

Randomized kd trees algorithm

Classical kd-tree algorithm (Freidman et al., 1977) is known to work efficiently for low-dimensional problems, but the performance degrades fast if it is used for high dimensional ones. Silpa et al. published in 2008 [29] an approach to ANN search using multiple randomized kd-trees, specifically designed to query SIFT features or similar. The original kd-tree algorithm splits the data in half at each level of the tree in the dimension for which the data manifested the biggest variance. The randomized trees are built choosing the split dimension randomly from the first D[1] dimensions on which data shows greatest variance. When searching the trees, a single

[1] In their experiments, they used $D = 5$.

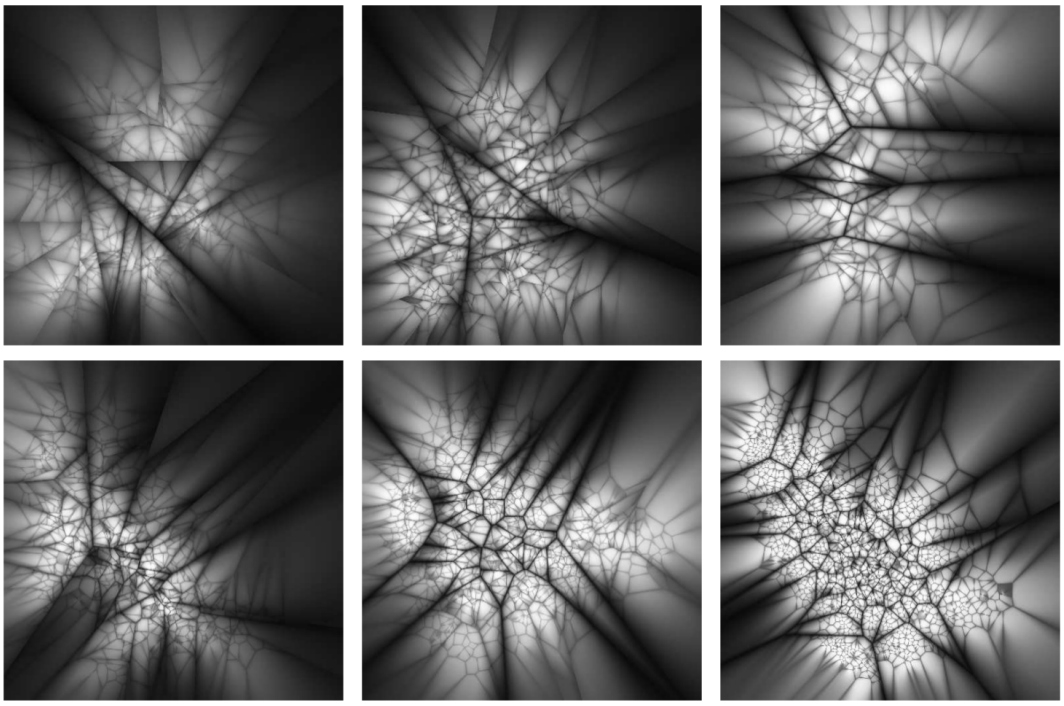


Figure 3.5: Projections of hierarchical k-means trees constructed using the same 100K SIFT features dataset with different branching factors: 2, 4, 8, 16, 32, 128. The projections are constructed using the same technique as in (Schindler et al., 2007). The gray values indicate the ratio between the distances to the nearest and the second-nearest cluster center at each tree level, so that the darkest values (ratio≈1) fall near the boundaries between k-means regions. *Image and footer extracted from* [23].

priority queue is maintained in all randomized trees so that search can be ordered by increasing distance to each bin boundary. The degree of approximation is determined by the number of checks done within the whole number of leaf nodes. After this fixed number of leaf examinations is done, the best k candidates are returned as the k ANN. In the implementation of FLANN, the user is able to specify the desired search precision which, during the training, will automatically fix the number of leaf nodes that will be checked to achieve the requested accuracy.

Hierarchical k-means trees algorithm

The hierarchical k-means tree is constructed splitting the data into K different regions using a k-means clustering at each level. In k-means clustering, we are given a set of n data points in d-dimensional space $\mathbb{R}^d$ and an integer p and the problem is to determine a set of p points in d, called centers, so as to minimize the mean squared distance from each data point to its nearest center [14]. The algorithm is performed recursively in every point in every region. The recursion is stopped when the number

of points in a region is smaller than K.

The algorithm developed in FANN explores the hierarchical k-means tree in a best-bin-first approach, in a similar way to the exploration of the kd-tree. The algorithm initially performs a single traversal through the tree and adds to a priority queue all unexaminated branches in each node along the path. Then, it obtains from the queue the branch that has the closest center to the query point and restarts the tree traversal from that branch. In each iteration the unexplored branches along the path are added to the priority queue. The degree of approximation works exactly as it does for the kd-trees algorithm: It checks a fixed number of leaf nodes, which can be set from a search precision user specified parameter.

Automatic selection of the optimal algorithm and parameters

This method, as has been seen, presents two different algorithms which do not work together. The selection of one of them will change performance depending on the dataset used. The selecting criteria has then to be highly dependent on several factors related with the dataset (i.e. the structure of the dataset, correlation) and the search precision required. Additionally we have seen that each of the previous algorithms have parameters in their own (e.g. the number of randomized trees to use in the case of kd- trees or the branching factor and the number of iterations in the case of the hierarchical k-means tree).

The choosing of either one or the other algorithm is considered as an additional parameter, which reduces the problem of algorithm selection to an optimization problem in the parameter space. The cost taken into consideration includes and combines search time, build time, and memory overhead. The relative importance of each of these factors is controlled by some weights (w_b and w_m) into the overall cost computation:

$$cost = \frac{s + w_b b}{(s + w_b b)_{opt}} + w_m m \tag{3.5}$$

Where s represents the search time for the number of vectors in the sample dataset, b represents the tree build time, and $m = m_t/m_d$ represents the ratio of memory used for the tree to the memory used to store the data. First step to get the parameter optimization is to sample the parameter space at multiple points and choose those values that minimize the cost in equation 3.5. After that Nelder-Mead downhill simplex method is used to further locally explore the parameter space. The optimization can be run in the whole dataset or in a small portion of it.

3.5 Conclusion

Nearest Neighbor search is a basic tool in Computer Vision which deals with finding closest points in metric spaces. To obtain exact Nearest Neighbors a linear search based method should be used, with a computational cost of $O(nd)$ being n the size of the dataset and d the dimensionality of each element. This makes unfeasible to apply exact NN for high dimensional or huge datasets. ANN are a solution to this

problem, retrieving approximate Nearest Neighbors in exchange of more efficiency in computation cost. Several approaches have been proposed to retrieve ANN, most of them based on kd-tree data structures and recently also hashing techniques for high dimensional datasets. As an example of one of this kd-tree based ANN method, FANN is a method which uses either randomized kd trees (which improves time and performance with respect original kd-tree algorithm) or hierarchical k-means trees, having an automatic selection of the optimal algorithm in its own implementation.

Chapter 4

Nearest-Neighbor Based Image Classification

4.1 Introduction

Image classification can be divided in two generic broad approaches: parametric classifiers and non-parametric classifiers. The first ones rely their classification in some parameters which usually require a learning or training stage (e.g. parameters of SVM [31], Boosting [25], parametric generative models [9]). The second ones base their classification directly on the data (although in some cases minor parameters can be also tuned). The most common non-parametric methods use nearest-neighbor (NN). NN based image classifier is believed to be one of the simplest multiclass classifier existing. The fact that it does not require training, it is a non-parametric model and its ease of implementation make NN favorable enough to be a fast-implementing baseline method.

In 2008, Boitman et al. defended in [4] that the Nearest Neighbor Based classifiers had been underestimated and misused due to finding nearest distances from images to images, instead of from images to classes, which could improve their experiments performance more than 17%. NBNN classifier results ranked among its contemporary classifiers, e.g. Varma, Bosch Trees, SPM, Bosch SVM, which implementation was way more complicated.

In this chapter we will expose the basis of the Naive-Bayes Nearest-Neighbor method and the recently published Local Naive-Bayes Nearest-Neighbor.

4.2 Naive-Bayes Nearest-Neighbor

The Naive-Bayes Nearest-Neighbor classifier is a very simple NN-based image classifier which main goal is to find the class C of a given query image Q. The algorithm has been implemented following the indications in [4] and [21], using dense SIFT features sampling.

4.2.1 Feature extraction

A dense feature sampling is computed for each image, allowing the option of enabling one or four spatial scales in case some scale invariance is needed (the selection of this option will depend on each specific dataset). In Boiman et al. method each SIFT descriptor d is augmented with its location l in the image in order to further use spatial position: $d = (d, \alpha l)$. The resulting L_2 distance between descriptors $\|d_1 - d_2\| = \|d_1 - d_2\|^2 + \alpha^2 \|l_1 - l_2\|^2$ having in this way a combined descriptor and spatial distance. The variable α is to be selected manually depending on the dataset.

In the thesis implementation we used OpenCV library SIFT features and specific tools for dense sampling and feature extraction [1]. The spatial locations of the feature were not implemented in this thesis method.

4.2.2 Probabilistic formulation

The NBNN classifier is an approximation based on the optimal Naive-Bayes image classifier. We have a test image query Q which class C we want to find. When the class prior $p(C)$ is uniform, the maximum a posteriori (MAP) classifier reduces to the Maximum Likelihood (ML) classifier:

$$\hat{C} = \arg\max_C p(C \mid Q) = \arg\max_C p(Q \mid C) \tag{4.1}$$

If we assume the simplest probabilistic model, which is the Naive-Bayes assumption (i.e. that descriptor $d_1, \ldots, d_n$ of query image Q are independent and identically distributed in class C :

$$p(Q \mid C) = \arg\max_C p(d_1, \ldots, d_n \mid C) = \prod_{i=1}^{n} p(d_i \mid C) \tag{4.2}$$

If we take the log ML probability:

$$\hat{C} = \arg\max_C \log(p(C \mid Q)) = \arg\max_C \frac{1}{n} \sum_{i=1}^{n} \log(p(d_i \mid C)) \tag{4.3}$$

Equation 4.3 presents the simplest classifier under the Naive-Bayes assumption. This optimal classifier needs to compute the probability density $p(d \mid C)$ of descriptor d in class C. The number of descriptors per image can be about the order of pixels if they are computed in a dense way, so a Parzen density estimation provides an accurate enough non-parametric approximation of the continuous descriptor probability density. If $d_1^C, \ldots, d_L^C$ are all the descriptors belonging to class C, we can express Parzen likelihood probability estimation as:

$$\hat{p}(d \mid C) = \frac{1}{L} \sum_{j=1}^{L} K(d - d_j^C) \tag{4.4}$$

$$K(d - d_j^C) = \exp(-\frac{1}{2\sigma^2} \parallel d - d_j^C \parallel^2) \tag{4.5}$$

Original NBNN (Q)

Require: A nearest neighbor index for each C, queried using $NN_C()$.
Require: A query image Q, with descriptors d_i .

for all descriptors $d_i \in Q$ **do**
  for all classes C **do**
    $totals[C] = totals[C] + d_i{-}NN_C(d_i)$
  end for
end for
return $\arg\min_C(totals[C])$

Table 4.1: Original NBNN (Q) Algorithm

Where $K()$ is the Parzen kernel function (equation 4.5) . To obtain best accuracy, all descriptors should be used in this computation. For computational reasons, a NN search is done in order to approximate this Parzen estimator.

4.2.3 NBNN Algorithm

The final NBNN algorithm deals with L_2 distances in the descriptor space. Usually the descriptors are relatively isolated from each other, so that translates in big distances within descriptors in the database. As a consequence, mainly all terms of the summation in equation 4.4 except an small group will be negligible. Therefore, we can safely approximate equation 4.4 just including in the summation the few r largest elements. These large elements can be obtained using a r-Nearest Neighbor search of a descriptor $d \in Q$ within the class descriptors $d_1^C, \ldots, d_L^C \in C$:

$$p_{NN}(d \mid C) = \frac{1}{L} \sum_{d=1}^{L} K(d - d_{NN_j}^C) \tag{4.6}$$

Figure 4.1 shows the accuracy of NN approximation of $p(D \mid C)$ taking 20 NN and 1 NN. Even selecting $r = 1$ the approximation is still acceptable. Boeman et al. claim that results from setting r from 1 to 1000 showed very small differences, and for $r = 1$ the algorithm becomes very simple and there is a speed up since there are fewer NN queries.

The final algorithm will be then:

$$\begin{aligned} logP(Q \mid C) &\propto -\sum_{i=1}^{n} \parallel d_i - NN_C(d_i) \parallel^2 \\ \hat{C} &= \arg\min \sum_{i=1}^{n} \parallel d_i - NN_C(d_i) \parallel^2 \end{aligned} \tag{4.7}$$

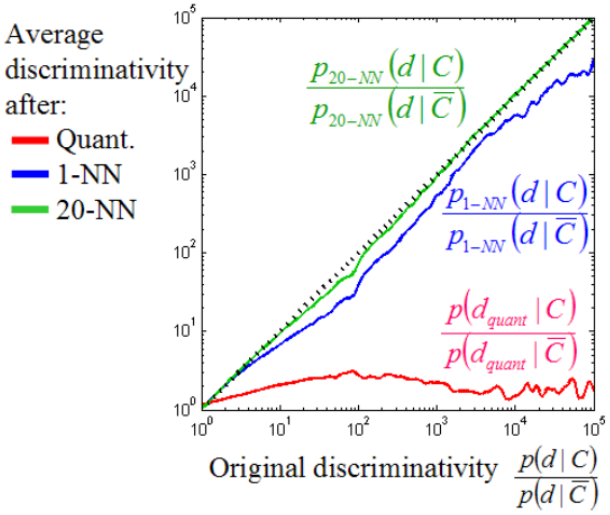


Figure 4.1: NN descriptor approximation. *Image extracted from* [21].

Local NBNN (Q, k)

Require: A nearest neighbor index comprising all descriptors, queried using NN(descriptor, #neighbors).
Require: A class look-up, Class(descriptor) that returns the class of a descriptor.

for all descriptors $di \in Q$ **do**
  $\{p_1, p_2, \ldots, p_{k+1}\} \leftarrow NN(d_i, k+1)$
  $dist_B \leftarrow \|d_i - p_{k+1}\|^2$
  for all categories C found in the k nearest neighbors **do**
    $dist_C = \min_{\{p_j | Class(p_j)=C\}} \|d_i - p_j\|^2$
    $totals[C] \leftarrow totals[C] + dist_C - dist_B$
  end for
end for
return $\arg\min_C totals[C]$

Table 4.2: Local NBNN Algorithm (Q, k)

The dependence on the K() has been removed and this simple form of the Naive-Bayes classifier has been implemented and tested. A clearer representation with a pseudo-code language is shown in table 4.1.

4.3 Local Naive-Bayes Nearest-Neighbor

In 2012 McCann et al. introduced one modification of the already known NBNN classifier [21]. In their work they claim to improve the NBNN image classifier in

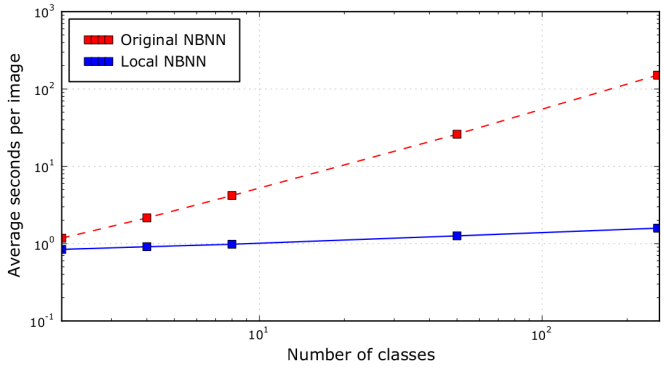


Figure 4.2: Time complexity of NBNN and Local NBNN. *Image extracted from* [21].

accuracy and in time speed when scaling to large number of object classes (scalability). The key point is that only classes which are local neighbors of a descriptor contribute in a secure way to their posterior probability estimates. In order to achieve that, just one search structure is used (instead of one search structure per class), enabling the possibility of finding descriptor's local neighborhood.

The original NBNN algorithm is represented in table 4.1, the local NBNN Algorithm is represented in table 4.2. Instead of searching for a query descriptor's nearest neighbor in each of the classes, we search for only the nearest few neighbors in a single dataset which includes all the features from all training data of all classes. An ANN k nearest neighbor search in this index is way faster than doing one ANN search for every class index, as it was done in the original NBNN algorithm. This allows the Local NBNN algorithm to scale up in a better way than the original algorithm to manage more classes.

Complexity of the original algorithm is $O(c\,N_D N_C \log(N_T N_D))$ and complexity of the Local NBNN algorithm is $O(c\,N_D \log(N_C N_T N_D))$, being c the number of checks done in the FLANN query, N_D the number of descriptors, N_C the number of classes and N_T the number of training images per class. As it can be observed, the number of classes is contributing linearly to the complexity in the NBNN original algorithm and logaritmically in the Local NBNN algorithm (figure 4.2).

To handle the distances of the rest of the descriptors, McCann et al. propose to estimate them as a upper bound on the density of background features: The estimation fix their distance arbitrarily to be the distance to the $k+1$ nearest neighbor. Instead of adjusting the distance totals to every class, they propose only to adjust the distances for the relatively few classes that were found in the k nearest neighbors, but discount those adjustments by the distance to background classes (the $k+1$ nearest neighbor).

The improvement in accuracy (figure 4.3) with respect the NBNN can be somehow unexpected, since Local NBNN is an approximation and original NBNN does indeed a more exhaustive search. When dealing with this issue, the author was contacted and he provided some more information about this phenomena. One possible explanation

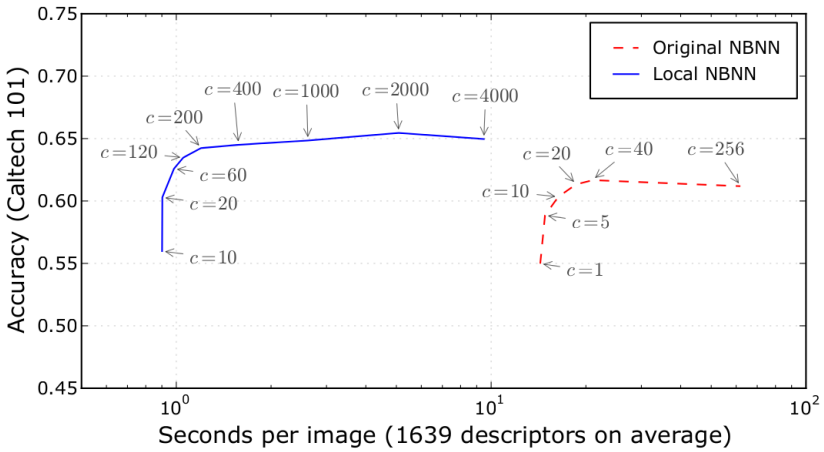


Figure 4.3: Accuracy of Local NBNN and NBNN. *Image extracted from* [21].

could be that visual features reside on lower dimensional manifolds in descriptor space, making Euclidean distances only meaningful within a small local region. *"Out of this region, two local features measured close by the Euclidean distance may actually be far from each other"* [18].

4.4 Conclusion

The NBNN classifier is a non-parametric classifier which base their classification directly on the data, and do not require learning/training of parameters. It is based on the optimal Naive Bayes classifier, approximated using a Parzen density estimator with Approximate Nearest Neighbors search. The NBNN classifier retrieves the class which accumulated distance between each descriptor d in an test image query Q and its ANN within class C is smallest, so in order to do that we need a NN search index for each class, and to do this queries for every class.

Local NBNN classifier modifies the previous algorithm in a way that only the classes represented in the local neighborhood of a descriptor contribute to their posterior probability estimates. That means that a merged global ANN search index is kept with all the descriptors from all the classes. The results in [21] show an improvement both in accuracy and in scalability when handling a large quantity of classes.

Chapter 5

The Chains Model

5.1 Feature extraction

For each image I_N dense SIFT feature sampling is computed, keeping the location of each feature for posterior use. It is advisable to previously obtain some scale measurement from the images in order to do all computations relative to it, eliminating this way part of the dependence on the scale of the object to detect. The method to obtain this measurement will depend on the object itself, and can be non trivial (e.g. Karlinsky et al. uses the horizontal size of humans head for detecting human hands). For single-scale datasets this measurement can be omitted, although is still interesting to have it in order to adapt some of the parameters (e.g. the SIFT sampling frequency).

We will denote this scale measurement S_n and the set of extracted features $F_n = \{f_n^j = \langle SIFT_n^j, X_n^j \rangle \mid j = 1, 2, \ldots, k_n\}$, where k_n is the total number of features extracted from image I_n. The $SIFT_n^j$ is the SIFT descriptor (128D row-vector [19]) of feature j from image n. If S_n has been extracted, dense features sampling frequency should be of $\min(0.2 \cdot S_n, 4)$ pixels and the size of the SIFT feature $S_n \times S_n$ centered at X_n^j. If there is no scale measurement, this values should be selected manually and multi-scale support could be achieved by testing each image at different scales .

For training images only, the detection location is marked as $l_n^d = (x_n^d, y_n^d)$.

5.2 Feature selection

Karlinsky et al. proposes a feature selection for the features extracted during the training stage according to a predefined *likelihood ratio*, fixing the number of selected features to 100 per each training image. To the best of our knowledge, this criteria only responds to computational time, memory constrains and scalability, therefore the number of selected features can be a parameter (i.e. the number of selected features can be increased, or even the whole selection stage removed) as far as the computational time remains feasible. We denoted this parameters as $N_{selected}$.

For every positive training image, features are selected following a log likelihood ratio defined as:

$$\Lambda_n^j = \frac{P(O = true, f_n^j)}{P(O = false, f_n^j)} \tag{5.1}$$

The likelihood in equation gives the quotient between the probability of a certain feature being in a positive image and the probability of a certain feature being in a negative background image. After this ratio is computed for F_n, the best $N_{selected}$ features are retained.

5.3 The chains model for object detection

In this section we are going to analyze the mathematical generative probabilistic chains model. General ideas about the mathematical development are extracted from [15], although they do not give self-explanatory information about the chains model for object detection without a reference part, focusing their explanations just in the specific case of the part detection with a reference part (face to hand detection). The main goal of this section is, therefore, to clarify the mathematical development and give an step-by-step formulation for the object detection without a reference part.

First of all, we will name every test image I_N. The observed variables of the model are the extracted features $F_N = \left\{F_j = F_N^j = \left\langle SIFT_N^j, X_j^N \right\rangle\right\}$. The index j of the features indicates some arbitrary order of the feature extraction which has no impact on the computation.

The unobserved variables are the target location L^d, an arbitrary length M and an ordered list of feature indices denoted by $T = \langle T(1), \ldots, T(M)\rangle$, where $1 \leq T(i) \leq k_N$, being k_N the number of features extracted from an image I_N. The list T describes a simple chain of features without repetitions.

The joint distribution of these variables would then be:

$$\begin{aligned} P_{CH}(M, T, L^d, \left\{f_N^j\right\}) = P(M, T) \cdot P(L^d \mid f_N^{T(M)}) \cdot \\ \prod_{m=1}^{M-1} P(f_N^{T(m+1)} \mid f_N^{T(m)}) \cdot P(f_N^{T(1)}) \cdot \prod_{j \notin T} P_W(f_N^j) \end{aligned} \tag{5.2}$$

Conditional probability is used along the chain, meaning *probability of each feature given the previous feature in the chain,* and we take independent product of their "world probabilities" for the the features outside the chain. The distribution of $P(M, T)$ is taken to be uniform for simple chains of up to $M = 4$ which distance between consecutive features in the chain are within a given threshold, fixed by Karlinsky et al. [15] in $1 \leq m \leq M : \frac{1}{2}s_N \leq \| X_N^{T(m+1)} - X_N^{T(m)} \| \leq \frac{3}{4}s_N$.

As we can see here, $P_W(f_N^j)$ has to be computed for every feature extracted from every testing image. In order to avoid that computation, we can take advantage of the fact that $\prod_{f_j^N \in F_N} P_W(f_N^j)$ is fixed for a given image I_N, so dividing the equation (5.2) by this term, we will have in stead of an equality, a proportionality; as shown in equation (5.3). Since we are dividing the product of all the $P_W(f_N^j)$

outside the chain by the product of all the $P_W(f_N^j)$ in a testing image I_N, the final result will be just the product of all $\frac{1}{P_W(f_N^j)}$ inside the chain, equation (5.4).

$$P_{CH}(M,T,L^d,\left\{f_N^j\right\}) \propto \frac{P_{CH}(M,T,L^d,\left\{f_N^j\right\})}{\prod\limits_{f_j^N \in F_N} P_W(f_N^j)} \tag{5.3}$$

$$\frac{\prod\limits_{j\notin T} P_W(f_N^j)}{\prod\limits_{f_j^N \in F_N} P_W(f_N^j)} = \frac{1}{\prod\limits_{j\in T} P_W(f_N^j)} = \frac{1}{P_W(f_N^{T(1)})} \cdot \frac{1}{\prod\limits_{m=1}^{M-1} P_W(f_N^{T(m+1)})} \tag{5.4}$$

If we substitute equation (5.4) in equation (5.3) and we convert conditional probabilities into joint probabilities we get:

$$\begin{aligned} P_{CH}(M,T,L^d,\left\{f_N^j\right\}) \propto P(M,T) \cdot \frac{P(f_N^{T(1)})}{P_W(f_N^{T(1)})} \cdot \\ \prod_{m=1}^{M-1} \frac{P(f_N^{T(m+1)},f_N^{T(m)})}{P_W(f_N^{T(m+1)})P(f_N^{T(m)})} \cdot P(L^d \mid f_N^{T(M)}) \end{aligned} \tag{5.5}$$

Equation 5.5 can be explained as follows: First factor is the distribution $P(M,T)$ (as said before, uniform over simple chain up to $M=4$) . Second factor is the first feature of the chain T of length M: it is the *"probability of the feature being similar to the training objects"* divided by the *"probability of having seen a similar feature in the whole training set"*. Third factor (multiplication from $m=1$ to $M-1$) contains a ratio similar to the 'suspicious coincidence' [3] of the pairs composing chain T, i.e. a multiplication of the mentioned ratio of each pair of features composing the chain, meaning this ratio *"probability of having seen this pair link on the training"* divided by *"probability of having seen a similar feature in the whole training set"* (for the second feature in the pair) and "*probability of the feature being similar to the training objects"* (for the first feature in the pair). Fourth factor is the voting probability from last feature in the chain to target detection.

The inference of the chains model is to obtain the posterior distribution over the target detection location.

$$P_{CH}(L^d \mid \left\{f_N^j\right\}) \propto \sum_{M,T} P_{CH}(M,T,L^d,f_N^j) \tag{5.6}$$

To compute this posterior, we will define S, C and D matrices based on the factors previously explained of equation 5.5 (second, third and fourth factors respectively). The marginal over chains of up to M length can be efficiently computed as a matrix multiplication:

$$P_{CH}(L^d \mid \left\{f_N^j\right\}) \propto \sum_{M,T} P_{CH}(M,T,L^d,f_N^j) = D \cdot C^* \cdot S \approx D \cdot C \cdot S \tag{5.7}$$

To compute C matrix we first will need an adjacency matrix A_N which entry jk is equal to w_{jk} if feature f_N^j and f_N^k are neighbors (i.e. they fulfill $\frac{1}{2}S_N \leq \left\|X_N^k - X_N^j\right\| \leq \frac{3}{2}S_N$) and 0 if they are not. The weight on the directed edge from f_N^k to f_N^j is $w_{jk} = \frac{P(f_N^j, f_N^k)}{P_W(f_N^j)P(f_N^k)}$.

$$\begin{aligned} C &= \sum_{m=0}^{M-1} (A_N)^m \\ D &= \left[P(L^d \mid f_N^1), \ldots, P(L^d \mid f_N^{k_N})\right] \\ S &= \left[\frac{P(f_N^1)}{P_W(f_N^1)}, \ldots, \frac{P(f_N^{k_N})}{P_W(f_N^{k_N})}\right] \end{aligned} \tag{5.8}$$

S is the vector containing the weights of the features to be the start of the chain, it indicates the probability of each feature to be inside the object to detect (similar to "*a good starting chain point*"). C^* is the matrix containing the chains, i.e. where entry jk is the sum of products of weights on all simple paths going from node k to node j. Matrix C is an easy to compute approximation of C^*, summing not only over simple paths but over all paths of length $\leq M$. D is the vector containing all the weights between the detection location and its neighbors, and has to be computed for every possible detection position (normally every pixel). Finding the D matrix which maximizes the matrix multiplication $D \cdot C \cdot S$ will give the detection location L^d. This is equivalent to a max-marginal inference over the posterior of the chains model.

It would also be possible to compute the MAP inference for the chains models, raising each entrance of D, A_N and S to a sufficiently large r power, obtaining D_r, A_N^r and S_r and maximizing $\sqrt[r]{D_r C_r S_r}$. Karlinsky et al. state that experimentally a hybrid technique between MAP and max-marginal inference attains the best performance: $D\sqrt[r]{C_r S_r}$, saving in this way also the computation of raising each element in D (which is computed once per pixel) to the r-th power.

5.3.1 Example

To clarify the step by step formulation, an example of a simple chain map is showed in figure 5.1. The schematic represents four features and their neighborhood relations (represented with black arrows). Out of this simple feature map we will compute and show the content of the matrix A_N (table 5.1) used to compute the C matrix (table 5.2) and also vectors S (table 5.4) and D (table 5.3).

5.4 Probability estimation using FLANN and KDE

The computation of probability estimations are done using Kernel Density Estimation from the training data. Given a set of samples from a distribution of interest

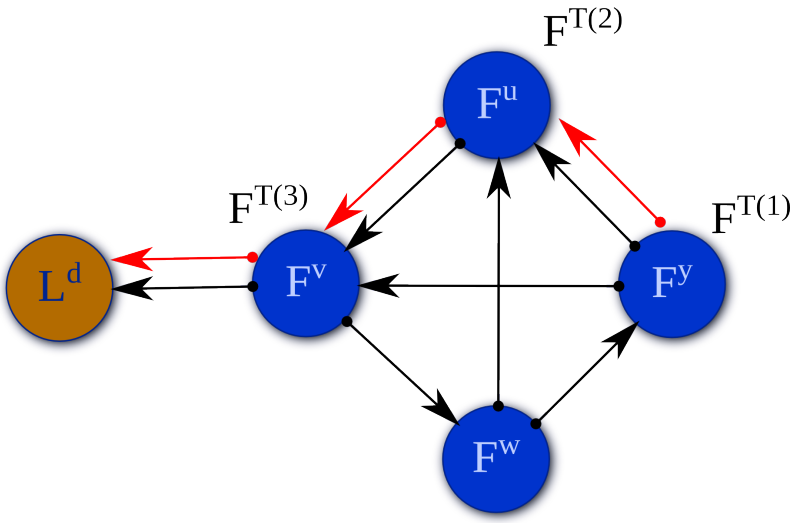


Figure 5.1: Chains schematic example. In it we can see a map composed by four features, linked within them by the black arrows. The red arrows show one possible chain.

	F^u	F^v	F^w	F^y
F^u	0	w_{uv}	0	0
F^v	0	0	w_{vw}	0
F^w	w_{uw}	0	0	w_{uy}
F^y	w_{yw}	w_{yv}	0	0

Table 5.1: A_N Matrix. We can see each weight of entry jk. If they are neighbors, they have a value w_{jk} , otherwise the value is zero.

$\{Y_1, \ldots, Y_R\}$, a Gaussian KDE is able to estimate $P(Y)$ as the probability of a new sample Y:

$$\begin{aligned} P(Y) &\propto \frac{1}{R}\sum_{r=1}^{R}\exp(-\left\|Y-Y_r\right\|^2/2\sigma^2) \\ &\approx \frac{1}{R}\sum_{Y_r\in NN}\exp(-\left\|Y-Y_r\right\|^2/2\sigma^2) \end{aligned} \tag{5.9}$$

Being NN the set of nearest neighbors of sample Y. This nearest neighbors are computed using FLANN explained in section 3.4, which is reported to be faster than the ANN technique [22] used originally by Karlinsky et al.

For the computation of the different probabilities we have seen in sections 5.2 and 5.3 we will build different FLANN search indices, do the respective query into them and apply a KDE to the nearest-neighbors obtained. The indices which need to be build are illustrated in table 5.5.

	F^u	F^v	F^w	F^y
F^u	null	w_{uv}	$w_{uv} \cdot w_{vw}$	$w_{uv} \cdot w_{vw} \cdot w_{wy}$
F^v	$w_{vw} \cdot w_{wu}$ $+w_{vw} \cdot w_{wy} \cdot w_{yu}$	null	w_{vw}	$w_{vw} \cdot w_{wy}$
F^w	w_{wu} $+w_{wy} \cdot w_{yu}$	$w_{wu} \cdot w_{uv}$ $+w_{wy} \cdot w_{yu} \cdot w_{uv}$ $+w_{wy} \cdot w_{yv}$	null	w_{uy}
F^y	w_{yu} $+w_{yv} \cdot w_{vw} \cdot w_{wu}$	w_{yv} $+w_{yu} \cdot w_{uv}$	$w_{yu} \cdot w_{uv} \cdot w_{vw}$ $+w_{yv} \cdot w_{vw}$	null

Table 5.2: C Matrix is the matrix containing all the chains. We can see all possible chains and their weights (i.e. their relevance into the L^d location). Each cell of the matrix contains the summation over all possible paths of length smaller than M . The chains represented in red in figure 5.1 would be the chain F^y to F^v .

$P(L^d \mid f^u)$	$P(L^d \mid f^v)$	$P(L^d \mid f^w)$	$P(L^d \mid f^y)$

Table 5.3: D vector. This vector expresses the edge weights from every feature to the target part.

$\frac{P(f^u)}{P_W(f^u)}$	$\frac{P(f^v)}{P_W(f^v)}$	$\frac{P(f^w)}{P_W(f^w)}$	$\frac{P(f^y)}{P_W(f^y)}$

Table 5.4: S vector. This vector contains the appearance likelihoods of each feature. If we were dealing with horse detection, the translation could be “How probable is that this feature looks like a horse feature?” divided by “Have I ever seen this feature before?”.

5.5 Memory requirements

The chains model requires to create at least six Nearest Neighbors search indices containing training data. The fact that Karlinsky et al. used a feature selection process give us some clues that the whole data structure could be difficult to handle either in computational cost (if the data indices are large, searching times will also grow) or either in memory. In this section we will realize an study about the memory needed for the chains models, linking it to parameters such as the total number of training images or the $N_{selected}$. The study will be done assuming this is not an optimal implementation, so each index is kept separately even though there is information redundancy.

We will denote ξ the size in Bytes of a SIFT OpenCV descriptor in our implementation, which uses 128 row vectors of 32 Bytes floats. $\{M_{a,}, M_b, \ldots, M_e\}$ will denote the memory usage of each of the indices, following the sub-index notation of table 5.5. k_n is the approximate average number of features per image, $R = R_{pos} + R_{neg}$ are the number of images, $N_{selected}$ is the number of selected features per positive training image and $N_{neighbors}$ is the number of m and l pairs of selected features,

Prob (KDE(V,S))	*Query vector (V)*	*Queried set (S)*
a. $P_W(f_N^j)$	$SIFT_N^j$	$\left\{SIFT_{t_i}^k \mid 1 \le i \le R,\ all\ k\right\}^{1,2,6}$
b. $P(f_N^j)$	$SIFT_N^j$	$\left\{SIFT_{t_i}^k \mid 1 \le i \le R,\ selected\ k\right\}^{1,2,6}$
c. $P(O, f_n^j)$	$SIFT_N^j$	$\left\{SIFT_{t_i}^k \mid 1 \le i \le R,\ all\ k,\ O_{t_i} = O\right\}^{1,3,6}$
d. $P(f_N^k, f_N^j)$	$\left[SIFT_N^k, SIFT_N^j, \alpha_N(X_N^k - X_N^j)\right]$	$\left\{[SIFT_{t_i}^m, SIFT_{t_i}^l, \alpha_{t_i} \cdot (X_{t_i}^m - X_{t_i}^l)] \mid 1 \le i \le R,\ selected\ m\ and\ l\right\}^{1,4,6}$
e. $P(L^h, f_N^j)$	$\left[SIFT_N^j, \alpha_N(L^d - X_N^j)\right]$	$\left\{SIFT_{t_i}^k, \alpha_{t_i} \cdot (l_{t_i}^d - X_{t_i}^k), \mid 1 \le i \le R,\ selected\ k\right\}^{1,4,5,6}$

[1] 'all k' - all of the features extracted from the training images; 'selected k' - only the selected features; 'selected m and l' - pairs of selected features.
[2] For both $P_W(f_N^j)$ and $P(f_N^j)$, the location component of the feature, i.e. X_N^j, is assumed to be distributed uniformly.
[3] O is binary and$O_{t_i} = true$ means that I_{t_i} is a positive training image.
[4] α_* is a weighting factor for the location component.
[5] To limit the set of features that can vote for the target part, we set $P(L^d|f_N^j) = 0$ when $\alpha_N \cdot (X_N^j - l_N^f) > 4$.
[6] $\{I_{t_1}, \ldots, I_{t_R}\}$ denotes the set of training images.

Table 5.5: Computations of all the required model probabilities using FLANN queries. A query for a vector V in a set S returns a set of k neighbors, from which the estimated probability is computed by Gaussian Density Estimation - $KDE(V, S)$.

which we can experimentally upper bound by $N_{neighbors} \le R_{pos} \cdot N_{selected}$.

$$\xi = 128 \cdot 32\, Bytes \tag{5.10}$$

We approximate the cases where the queried set is a 130 and a 258 row vector in the following way:

$$\begin{aligned} \xi' &= 130 \cdot 32 \approx \xi \\ \xi'' &= 258 \cdot 32 \approx 2\xi \end{aligned} \tag{5.11}$$

a,c. $P_W(f_N^j)$, $P(O, f_n^j)$

$$M_a = M_c = \xi \cdot k_n \cdot R \tag{5.12}$$

b,e. $P(f_N^j)$, $P(L^h, f_N^j)$

$$M_b = M_e = \xi \cdot N_{selected} \cdot R_{pos} \tag{5.13}$$

d. $P(f_N^k, f_N^j)$

$$M_d = 2\xi \cdot N_{neighbors} = 2\xi \cdot R_{pos} \cdot N_{selected} \tag{5.14}$$

Total memory usage will be then M_{Total}:

$$M_{Total} = \sum M_i = \xi(2k_nR + 4N_{selected}R_{pos}) \tag{5.15}$$

After the training we can remove index c used to obtain $P(O, f_n^j)$, since it is not used during the training.

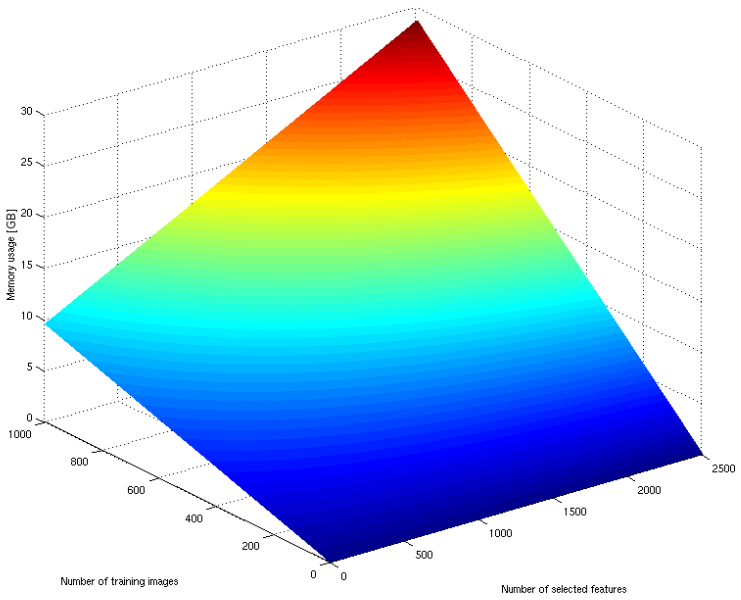


Figure 5.2: Memory usage in GB for different values of $N_{selected}$ and R, with a fixed $k_n = 2500$. Maximum value for $N_{selected} = k_n$ (no selection process) and $R = 1000$ images is $M_{Testing} = 28.61$ GB.

$$M_{Testing} = \xi(k_n R + 4N_{selected}R_{pos}) \tag{5.16}$$

As we can see the memory usage grows linearly with R (if we assume a symmetric dataset, $R = 2R_{pos}$), k_n and $N_{selected}$.

One reasonable example of memory usage could be: $N_{selected} = 100$, $R_{pos} = 328$ (Weizmann horses dataset, assuming same amount of negative background images) and $k_n = 2500$, the memory usage during testing will be of $M_{Testing} = 6.76$ GB. In figure 5.2 some other values are shown.

5.6 Implementation details

The implementation of the chains model object detector has been done using C++ as the programming language. The main libraries used are OpenCV [1], Boost and Eigen. The multithread support was achieved by using OpenMP. Additionally, some Python scripts have been developed in order to create the detection annotations in the training images.

The implementation can be divided in two different stages: The training (not an *actual training*, the detector needs to have examples of the data in order to estimate the probabilities in the chains model) and the testing. The two stages are clearly

separated and can be performed independently, so we are going to study them as different blocks (figure 5.3).

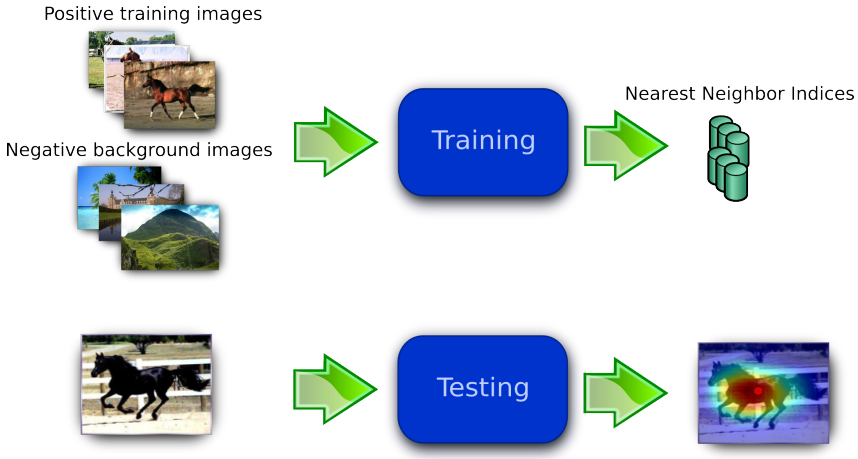


Figure 5.3: Training and testing. We can clearly divide the implementation in two stages: training and testing. The training has as inputs the training images (positive images and negative background images), and it builds the Nearest Neighbor indices used to estimate the probabilities. The testing has images input images, and the detection of the object as outputs.

5.6.1 Training

In figure 5.4 we can see the flow diagram of the training stage in the chains model. In this subsection we will discuss the implementation of each of the blocks in the diagram.

Feature extraction

The feature extraction is done following the guidelines from section 5.1. We decided to implement this using two different classes: PathToFeatures and ImageToFeatures. The first one only cares about how to recursively iterate through the filesystem

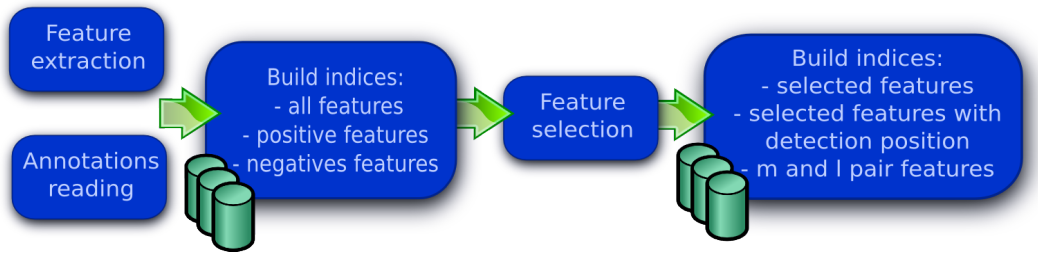


Figure 5.4: Training flow diagram.

folders and read images, whereas the second one contains the process of extracting the features and their locations. This way of constructing the classes allow easily to modify the way we extract features (e.g. type of feature, detector, extractor) or to do any other transformation to the image just providing another object.

The approach to memory efficiently keep the features and spatial position of these features is the following: The PathToFeatures extract first all the positive features in a matrix, keeping in a separate vector to which image belongs every feature. After that, all the negative background images are extracted and stored in the very same matrix, having in a merged matrix (named *all_features*) positive and negative features. This will allow in the future to build three indices using the same matrix, saving $\xi \cdot k_n \cdot R$ memory (section 5.5). The coordinates (normalized to one) of each feature are kept in a separate matrix.

Annotations reading

Every positive training image has an annotation indicating where is the center of the object. This is marked as a position in rows and columns in the pixel of the matrix of the images. We read them from a file and we save them in a matrix, so each entry j in that matrix will be the annotation to image I_j.

Feature selection

In order to do the feature selection, we have to compute the likelihood ratio of equation 5.1 for every positive image feature. After that we keep the $N_{selected}$ with a higher likelihood ratio. When we select them, we copy them to a new matrix which will contain all the selected features (named *selected_features*). We keep also the information of which image each feature belongs to and the location coordinates of each selected feature.

Indices building

In order to realize FLANN search queries, we have to build an index first. This index will contain a reference to the data, which in our case will be the different matrices with the different sets of features.

The first index to build is the one referencing all the features extracted, and its implementation is straight forward. For the positive index building, we first create a matrix header (reference) *positive_features* to the range of all_features which contains positive features, and the same for the negative features (which matrix name will be *negative_features*).

For the selected features, the first index containing just the SIFT descriptors is trivial since we have the data already in the matrix *selected_features*. Another index has to be build with the same data, but concatenating at the end the coordinate difference between the annotated detection and the location of the feature. FLANN class structure does not allow to perform queries in non continuous matrices, so we need to copy the information of the descriptors in to another matrix and add there the two columns of the x and y coordinates difference.

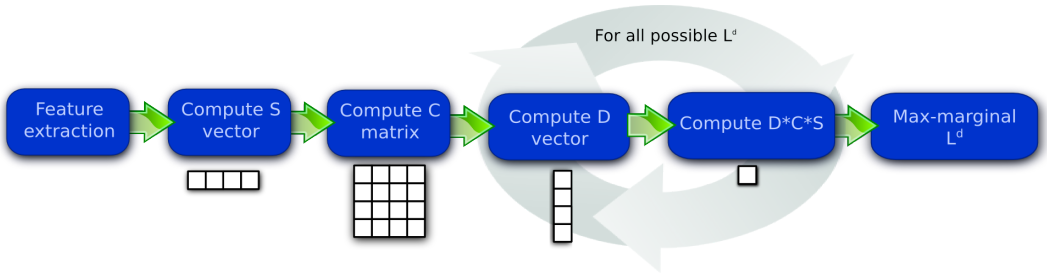


Figure 5.5: Testing flow diagram.

The last index needed to be build is the one containing the l and m pairs. In order to build it, we iterate over all selected features and add to a new matrix all the feature pairs which fulfill the neighborhood criteria. This new matrix will contain both SIFT descriptors and the difference of coordinates between them.

5.6.2 Testing

The testing starts with all the FLANN indices already created with the example data provided during the training. Basically, the testing will take an image $I_{testing}$, and then compute the matrices D, C and S and find L^d which maximizes the matrix multiplication $D \cdot C \cdot S$. The complete diagram flow is shown in figure 5.5.

Feature extraction

The feature extraction uses the same tools as the training (PathToFeature and ImageToFeature) and there is not any difference.

Matrices and vectors arithmetic

In order to efficiently obtain the posterior of equation 5.6 the computation is piecewised in matrices and vector multiplications. We will compute one S vector and C matrix per image, and as many D vector as L^d possible candidates we want to take into account for the max-marginalization (in our implementation, one per pixel).

First we will compute S vector (which is the simplest to compute) and after that, C matrix (out of an A_n image). Once we have this two elements, we will multiply them (one $C \cdot S$ multiplication per image).

S will be a vector of opposite dimensions to D (i.e. S should be a $1 \times N_{features}$ vector, D has to be a $N_{features} \times 1$ vector, since we want $D \cdot C \cdot S$ to be a 1×1 matrix, so $1 \times N_{features} \cdot N_{features} \times N_{features} \cdot N_{features} \times 1$), and it requires two FLANN queries for each element in the vector (the number of elements of the vector is the number of features extracted from the testing image).

For the computation of C matrix, we first need to compute A_N. In order to do that, we need to iterate over all features and compute all possible neighbors of each feature (this means $N_{features} \times N_{features}$ neighborhood criteria checks if no further

optimization is done), and for each pair of neighbor features we have to realize three FLANN queries. The size of A_N matrix is $N_{features} \times N_{features}$, filling the entries jk which are not neighbors with a 0. The use of sparse matrix for this matrix improves substantially the memory efficiency.

Once A_N is computed, C comes straight forward following equation 5.8. At this point, the matrix multiplication $C \cdot S$ is computed, retrieving a vector of $1 \times N_{features}$.

The following step is to create all possible D matrices in the testing image. As said previously, in our implementation we compute one D matrix per pixel. For each D computation we need to perform two FLANN queries, and the number of D matrices computed will be equal to $I_{rows} \times I_{cols}$. For each D matrix computed we have to do the matrix multiplication $D \cdot C \cdot S$. Although it could seem more efficient to just keep the highest result of the matrix multiplication, we decided to keep all the values of all matrices multiplication and retrieve the position of the highest value as the detected location, enabling in this way multithreading and eventually plotting the values of $D \cdot C \cdot S$.

The most costy part of the testing was indeed the computation of all possible D matrices per testing image. In order to deal with it in a more efficient way, we enabled multithread support for it, assigning to each of the cores available the computation of a D matrix. The second more costy operation was the creation of the A_N matrix containing all the possible weights. The same approach was followed: Multithread was enabled in such a way that each of the available cores could search the neighbors of different features.

Chapter 6

Results

6.1 Chains model object detector

In this section we will show the performance and some results of the chains model object detector. The subsections 6.1.1 and 6.1.2 show experiments done with the implementation of the thesis. This experiments have been done with different sets of data coming from the Weizmann horse dataset, in their single scale version. All the testing have been done using dense features sampling with a 4 px separation, with SIFT descriptors of 10 px of diameter and a criteria of neighborhood with respect the width of the image of: minimum 0.07 and maximum 0.2. We keep $N_{selected} = 100$ features per positive image.

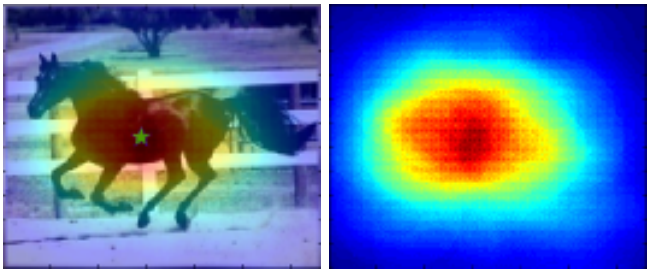
Subsection 6.1.3 shows the experiments obtained by Karlinsky et al. [15].

6.1.1 Set 1

For this test we have used image number 147 of the Weizmann Horse Dataset. The training has been done with 10 positive images and 10 background images provided from that dataset, assuring the testing image is not between them. In figure 6.2 we show the 40 and 100 most significant chains (i.e. with a higher w_{jk}), in figure 6.1 we show the detection marked with a green pentagram star and the different values of $D \cdot C \cdot S$ for every pixel. To end with, we show in figure6.3 the output of the detector for a background image, plotting it with the same colormap scale than figure 6.1 to state that the chains detector is indeed recognizing the horse since the values for a background image are substantially lower.

6.1.2 Set 2

For this test we have used image number 46 of the Weizmann Horse Dataset. The parameters are the same as in *Set 1*.



(a) (b)

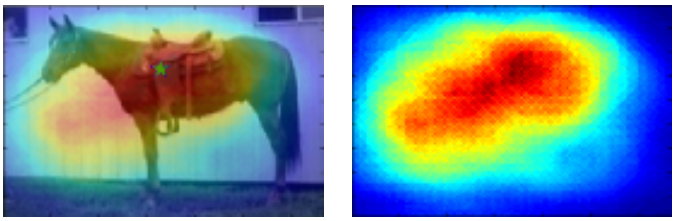
Figure 6.1: Horse detection. (a) Superposed to the testing image we can see the detection point marked with a green pentagram star and the $D \cdot C \cdot S$ values for all possible $L^d$with a colormap from blue (lower values) to red (higher values) with 40% of transparency. (b) $D \cdot C \cdot S$ values for all possible L^d.

(a) (b)

Figure 6.2: Chains plotted with a colormap from yellow (lower values) to red (higher values). (a) 40 most significant chains (i.e. with a higher w_{jk}). (b) 100 most significant chains.

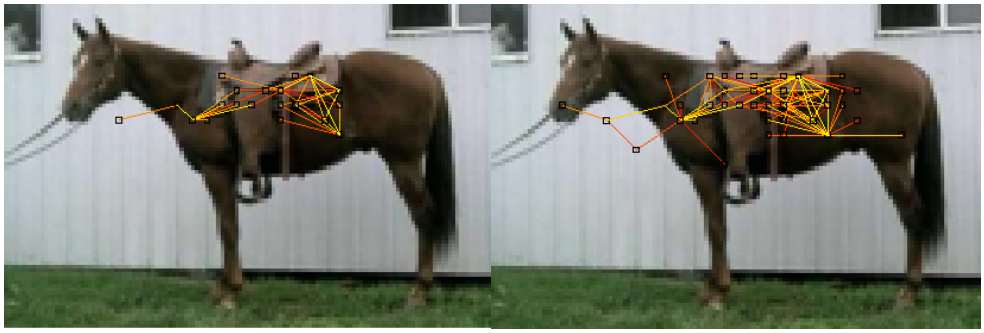
(a) (b)

Figure 6.3: Negative background image detection. (a) Superposed to the background image we can see the $D \cdot C \cdot S$ values using the same colormap and scale as for figure 6.1. The values of $D \cdot C \cdot S$ are substantially lower than for horse detection, which enables the possibility of fixing a threshold for accepting or rejecting the max-marginal. (b) $D \cdot C \cdot S$ values for all possible L^d.



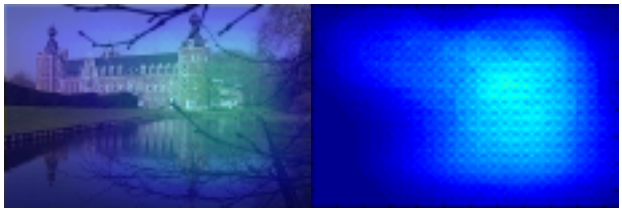
(a) (b)

Figure 6.4: Horse detection. (a) Superposed to the testing image we can see the detection point marked with a green pentagram star and the $D \cdot C \cdot S$ values for all possible $L^d$with a colormap from blue (lower values) to red (higher values) with 40% of transparency. (b) $D \cdot C \cdot S$ values for all possible L^d.



(a) (b)

Figure 6.5: Chains plotted with a colormap from yellow (lower values) to red (higher values). (a) 40 most significant chains (i.e. with a higher w_{jk}). (b) 100 most significant chains. We can observe that two of the chain points are outside the horse, although they are leading the chains inside the horse. As reflected in figure 6.4, this affects the $D \cdot C \cdot S$ matrix distribution, but the overall detection is still correct (global maximum is inside the horse).



(a) (b)

Figure 6.6: Negative background image detection. (a) Superposed to the background image we can see the $D \cdot C \cdot S$ values using the same colormap and scale as for figure 6.4. (b) $D \cdot C \cdot S$ values for all possible L^d.

Method	[24]	[17]	[12, 16]	Chains Model[15]	Star Model
Result	99.9%	97.5%	98.5%	99.5%/93.4%	98%/89.2%

(a)

Method	[28]	[12]	Chains Model[15]	Star Model
Result	95.7%	93.9%	97.5%	87.8%

(b)

Table 6.1: Comparison between different methods by precision-recall at EER. (a) UIUC car testset; (b) Weizmann Horses dataset, the larger advantage of the chains may be related to fact that horses are somewhat more flexible than rigid 3-D objects, such as cars. *Image and footer extracted from* [15].

6.1.3 Paper results

Karlinsky et al. provides full testing for object detection in the following datasets: The Weizmann Horse dataset and the UIUC car testset.

Car Detection. Using 550 positive and 500 negative training images and 200 cars in 170 test images of the UIUC single scale dataset, the chains model achieved 99.5% at precision-recall Equal Error Rate (ERR). To test cross-dataset generalizations Karlinsky et al. replaced the training set with Caltech-101 cars, achieving 93.4%.

Horse Detection. Using 323 side-view horse images of the Weizmann horses single scale dataset and 500 negative images, the chains model achieved $97.5 \pm 1\%$ at EER.

A comparison with other methods is provided in table 6.1.

6.2 Naive-Bayes Nearest Neighbor image classificator

6.2.1 Goal and results

In this thesis we have implemented also a Naive-Bayes Nearest Neighbor image classificator. Implementing and studying this classifier was not the main goal of the thesis, neither was doing a full testing of it; but just was planned as an intermediate step to implement and understand correctly the chains model detector, enabling the possibility of testing some features (such as filesystem iterations, feature extraction, nearest neighbor search, etc.) with a way simpler algorithm in order to do a refactoring of the same code for the chains model.

For this reason, the algorithm of the NBNN classifier is not exhaustively tuned (since that was not the goal) and the results are below the ones shown by [4, 21]. To illustrate about the results obtained with an NBNN classifier correctly tuned, we will show in subsection 6.2.2 the results provided by them.

The testing realized for the NBNN implementation consist in a testing of 15 classes extracted from the Caltech-101 dataset. For each class, 30 images were used

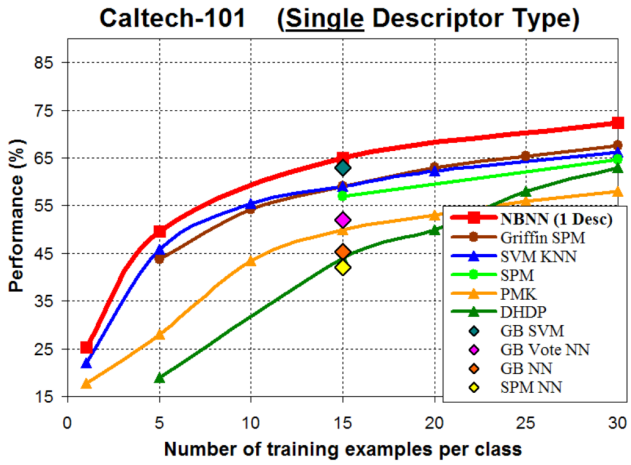


Figure 6.7: Performance comparison of NBNN Classifier on Caltech 101 with single type SIFT descriptors. *Image extracted from* [4]

as training examples and 15 as testing images (avoiding overlapping between training and testing), reporting a ratio of correct classified images of 68.48%.

6.2.2 Paper results

Boiman et al. in [4] present several results of the NBNN tested in different datasets and with different configurations (single descriptor type, multi descriptor type, etc.). In this subsection we are going to show the results of the NBNN classifier for the Caltech 101 with single type SIFT descriptors (figure 6.7). As it is possible to see, the results of the NBNN classifier outperformed some of their contemporary methods, which in some cases had a more complex algorithm.

Chapter 7

Conclusions

In this thesis we present the chains model object detector that can efficiently detect and locate specific objects in images. First of all, we have provided a step-by-step formulating of the chains model for object detection without a previously known reference part. Moreover, a detailed explanation of the functioning of the chains model has been given and an study of the memory consumption of the method have been developed.

On top of that, a chains model object detector has been implemented, having two clearly separated stages: The training and the testing. During the training we use positive and negative training images, the features are extracted and selected, and the Nearest Neighbor search indices which will be needed for the estimation of the chains probabilities are built.

During testing, for each test image we compute C and S matrix, and D matrix for every pixel in every image. For the computation of these matrices we do FLANN queries and KDE probability estimates. The generated chains will have a voting weight based on the appearance of the patch itself but also in its context. After this, a max marginalization is done to obtain the most likely object detection L^d.

The chains model detector has been tested with horses from the Weizmann horse dataset and background images, showing the weighted chains of the horses, the detection location and the chains posterior for the whole image. For the background images we show that the chains posteriors are considerably lower than in the horse testing images, which allow us to establish a threshold to accept or reject the max-marginalization of L^d.

7.1 Future work

The chains model detection is still a young detection method which has space for improvement. Some ideas which may improve the overall performance of the method are:

1. **The use of more distinctive features:** Nowadays there are features proven to be more distinctive than SIFT descriptors. One possible option to start

with could be the use of Integral Channel Features [8] although a deeper study of the state-of-the-art descriptors should be done.

2. **Better feature selection:** The feature selection done by Karlinsky et al. seems to be related to computational and memory limitations (as shown in section 5.5). The feature selection is by the moment quite simple and straight forward, but do not ensure that all the features selected are of interest for the training. Some pre-processing could be done to the training images in order to improve this. One possible approach could be to previously apply a bounding box to the positive training images in order to ensure there are not selected features from non informative regions.

3. **Nearest Neighbor search and KDE:** The Nearest Neighbors search and KDE probability estimates are computationally heavy. Other alternatives could be studied.

4. **Extend the chains model to video:** It could be interesting to extend the chains model to deal with video by introducing not only space-based chains, but spatio-temporal chains, taking advantage of the information redundancy.

Bibliography

[1] Open source computer vision library 2.4.1. http://code.opencv.org.

[2] M. Andriluka, S. Roth, and B. Schiele. Pictorial structures revisited: People detection and articulated pose estimation. *CVPR*, 2009.

[3] H. B. Barlow. *Unsupervised learning*, volume 1, pages 295–311. MIT, 1989.

[4] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. *IJCV*, 2004.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[6] A. Criminisi, J. Shotton, and E. Konukoglu. *Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning*. Microsoft Research technical report, 2011.

[7] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. *CVPR*, 2005.

[8] P. DollÃ¡r, Z. Tu, P. Perona, and S. Belongie. Integral channel features. *BMVC*, 2009.

[9] R. Fei-Fei, L. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. *CVPR*, 2004.

[10] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. *CVPR*, 2008.

[11] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. *CVPR*, 2003.

[12] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. *CVPR*, 2009.

[13] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1998.

[14] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 881–892, 2002.

[15] L. Karlinsky, M. Dinerstein, D. Harari, and S. Ullman. The chains model for detecting parts by their context. *CVPR*, 2010.

[16] C. Lampert, M. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. *CVPR*, 2008.

[17] B. Leibe, A. Leonardis, and B. Schiele. Robust object detection with interleaved categorization and segmentation. *IJCV*, 2008.

[18] L. Liu, L. Wang, and X. Liu. In defense of soft-assignment coding. *ICCV*.

[19] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[20] S. Maneewongvatana and D. Mount. The analysis of a probabilistic approach to nearest neighbor searching. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures*, WADS '01, pages 276–286, 2001.

[21] S. McCann and D. G. Lowe. Local naive bayes nearest neighbor for image classification. *To appear in CVPR*, 2012.

[22] D. Mount and S. Arya. Ann: A library for approximate nearest neighbor searching. CGC 2nd Annual Workshop on Comp. Geometry, 1997.

[23] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[24] J. Mutch and D. Lowe. Multiclass object recognition with sparse, localized features. *CVPR*, 2006.

[25] A. Opelt, M. Fussenegger, A. Pinz, and P. Auer. Weak hypotheses and boosting for generic object detection and recognition. *ECCV*, 2004.

[26] S. Prince. *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012.

[27] H. Schneiderman. *A Statistical Approach to 3D Object Detection Applied to Faces and Cars*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2000.

[28] J. Shotton, A. Blake, and R. Cipolla. Multiscale categorical object recognition using contour fragments. *PAMI*, 2008.

[29] C. Silpa-Anan and R. Hartley. Optimised KD-trees for fast image descriptor matching. *CVPR*, 2008.

[30] T. Tuytelaars. Dense interest points. *CVPR*, 2010.

[31] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *IJCV*, 2007.