CHALMERS

An Imaging System for Object Detection

Master's Thesis in Engineering Mathematics and Computational Science

Simon Pfreundschuh

Department of Mathematical Science
Chalmers University of Technology
Gothenburg, Sweden 2014

Abstract

This thesis was conducted at the *Fraunhofer Chalmers Centre for Industrial Mathematics* in collaboration with the *Fraunhofer-Institut für Techno- und Wirtschaftsmathematik*. The aim of this thesis is to develop an imaging system for the automated detection of holes in images of supermarket shelves. The proposed approach uses an unsupervised segmentation method to presegment the image into homogeneous regions. Each of those image regions is then classified separately using a support vector machine. Finally, suitable bounding boxes are found for image regions that are likely to represent holes. Apart from the SVM classifier also an AdaBoost classifier and a structural classifier based on conditional random fields are implemented and tested. This thesis describes the implementation and performance characteristics of the resulting imaging system, which is implemented using the ToolIP graphical image processing framework and C++.

Acknowledgements

I would like to thank Fraunhofer Chalmers Centre for giving me the possibility of working on this thesis. I am especially grateful for the help and guidance provided by my supervisor at Fraunhofer Chalmers Centre, Johan Karlsson. I highly appreciate his help, inspiring discussions and feedback during my time at the centre. I also want to thank Mikael Wallman for many interesting discussions and the implementation of the agent-based texture descriptor. Finally, I want to thank Tobias Gebäck, my supervisor and examiner at Chalmers University of Technology, for the feedback and guidance.

Simon Pfreundschuh, Gothenburg May, 2014

Contents

Introduction

In this work, an imaging system for the detection of holes in images of supermarket shelves is presented. The work is embedded in a project conducted by the *Fraunhofer ITWM* and the *Fraunhofer-Chalmers Center for Industrial Mathematics* in collaboration with an industrial partner. The aim of the project is to automate the analysis of supermarket shelves. Shelf appearance contains valuable information for the manufacturer and, so far, has to be evaluated manually. This makes the process slow and costly and an automated analysis is therefore of great interest.

Background

Recent advances in technology have made digital imaging widely available. Today, taking an image is probably the fastest and cheapest way to transfer information to another person. Unfortunately, this information is not directly available for computers. For a computer an image is nothing but gridded spectral data recorded by the imaging sensor. The field of image processing describes methods of how high-level information, for example what kind of objects contained in an image, can be extracted from image data. Such methods can be applied in almost every scientific field as well as in industry. The field of image processing is still relatively young and vivid research throughout the last 30 years has produced many powerful methods. However, there is no general solution to the problem. Each solution has to be tailored to the application context, often requiring the combination of different methods and the integration of domain specific knowledge.

Aim

The aim of this thesis is to develop an imaging system for the detection of holes in shelf images. A sample image is displayed in figure 1. A hole is defined to be an image region that

- contains no products,

- goes from the front to the back of the shelf,
- is at least as wide as the adjacent products.

Figure 1: A sample image of a supermarket shelf containing a hole. The region marked in red satisfies the definition of a hole. It is an image region, that contains no products, goes from the front of the shelf to the back and is at least as wide as the adjacent products.

The system is implemented using the ToolIP image processing framework[1] developed by the Fraunhofer ITWM. This document introduces the fundamentals of the applied methods from the fields of image processing and machine learning and presents the implementation of the system. Finally, the different performance aspects of the resulting system are evaluated.

Methods

The approach taken here is to combine an unsupervised segmentation algorithm and an SVM classifier to classify the resulting image regions. Presegmenting images in order to obtain a more expressive image representation is a popular approach which has been successfully applied to other object classification problems, for example in [2]. The resulting image regions are then classified using an SVM classifier. Besides the SVM classifier also an AdaBoost classifier and a structural classifier based on conditional random fields are implemented and tested. The structural classifier is realized using a two-level approach by using the results from the SVM and AdaBoost classifier as inputs

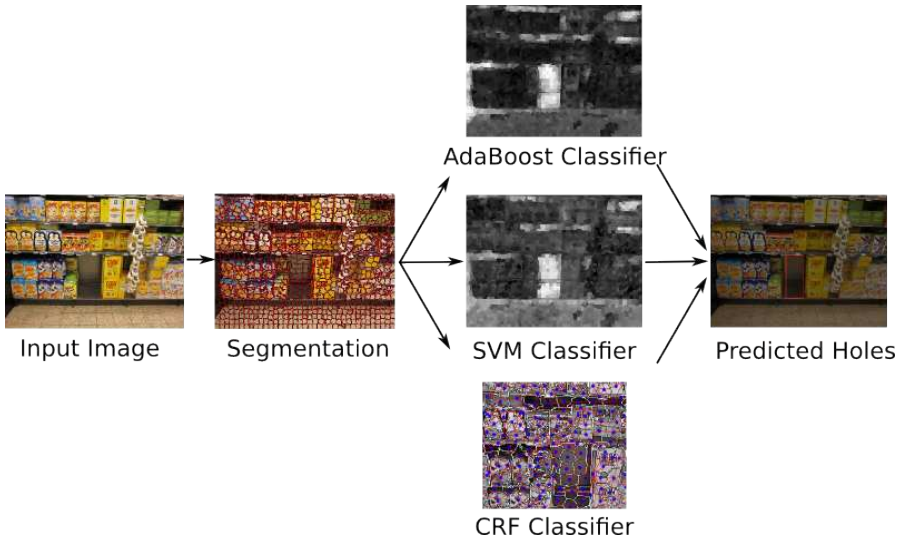


Figure 2: Flow diagram of the proposed method. The input image is segmented using an unsupervised segmentation method. The resulting image regions are then classified using an SVM classifier. Finally, suitable bounding boxes for the holes in the image are found.

to the conditional random field classifier. Similar approaches to image segmentation and classification have also been taken in [3, 4, 5]. In figure 2 the proposed method is displayed as a flow diagram.

Outline

The structure of this document is as follows. The first two chapters introduce the theoretical foundations from the field of image processing (chapter 1) and the field of machine learning (chapter 2). The third chapter describes the implementation of the imaging system and presents intermediate results, that motivate concrete implementation choices. The fourth chapter contains a detailed evaluation of the imaging system. Finally, the results and difficulties encountered throughout the implementation are discussed in chapter 5.

1

Image Processing

In this chapter the foundations of digital color imaging and image processing are introduced. An image describes a given scene using information about the spectral composition of the light refracted by the objects in that scene. Mathematically, an image is given by an image function I of two local variables x,y. The function I may be scalar- or vector-valued, depending on whether the function represents a black-and-white or a color image. Here, only color images will be considered, that can be described by a three-dimensional image function. It is common to consider each component of a vector-valued image function separately. Thus, a color image may be given by three grayscale images, each of them representing one dimension of the color information. The three resulting images are referred to as *color channels*. Processing an image on a computer requires the domain and the values of I to be digitized. A digital image is thus given by a matrix $I = (I_{i,j})$. The elements I are called pixels. The value $I_{i,j}$ of the pixel at position i,j is referred to as its gray value.

1.1 Color Imaging

In medium- and high-light situations human color vision is trichromatic. This means that the human eye possesses three different kind of color sensing cells with different responsivity spectra. As a result, almost every visible color can be reproduced by three different monochromatic light sources. A natural description of color would thus be given by the intensities of those three light sources. In some contexts, however, different descriptions of color may be more suitable. Color information is usually given with respect to a certain color space. A color space is a mathematical model that is used to represent colors using tuples of numbers. The following section contains a description of the color spaces that will be relevant for the implementation of the imaging system.

1.1.1 CIE 1931 XYZ

The CIE 1931 XYZ color space is the scientific basis of objective color representation [6]. The general idea is to describe each color with respect to three monochromatic reference colors, so called primaries. A color is then described by the respective intensities of each primary that is needed to reproduce the color by additive mixing. Those values are called tristimulus values. Given the power spectrum of a light signal $E(\lambda)$ with respect to the wavelength λ, the relation between $E(\lambda)$ and the tristimulus values is described by the three color matching functions $\bar{x}, \bar{y}, \bar{z}$. Figure 1.1 **(a)** displays the color matching functions for the CIE 1931 XYZ color space. The XYZ representation of a light signal is then given by

$$X = \int \bar{x}(\lambda)E(\lambda)\, d\lambda \tag{1.1}$$

$$Y = \int \bar{y}(\lambda)E(\lambda)\, d\lambda \tag{1.2}$$

$$X = \int \bar{z}(\lambda)E(\lambda)\, d\lambda \tag{1.3}$$

From (1.1),(1.2),(1.3), it is clear that the energy spectrum of the light refracted by an object depends on the energy spectrum of the illuminating light source. Human color vision, however, is able to adapt to varying illumination conditions. A white object, for example, appears white in the light of an incandescent bulb as well as in daylight. Their spectra however are different. In order to truthfully reproduce a color it is hence necessary to specify a reference white point. Such a reference white point is also of importance for the conversion into other color spaces. The XYZ representation of a color depends on the brightness of the color. This dependence can be eliminated by normalizing the X,Y,Z by their sum. This yields two independent values x,y, called the chromaticity coordinates.

$$x = \frac{X}{X+Y+Z} \tag{1.4}$$

$$y = \frac{Y}{X+Y+Z} \tag{1.5}$$

The *chromaticity* of a color denotes the brightness independent part of the color description. Projecting all visible colors onto the chromaticity plane yield in the CIE x,y chromaticity diagram shown in figure 1.1 **(b)**. The CIE xy chromaticity coordinates form a common basis for the conversion between different color spaces. One of the disadvantages of the CIELAB XYZ space is that it is defined using *imaginary* primaries. This means that they do not correspond to any human-perceivable color, but rather are derived from experiments conducted with real colors. It is thus not suitable for the use in devices that reproduce colors.

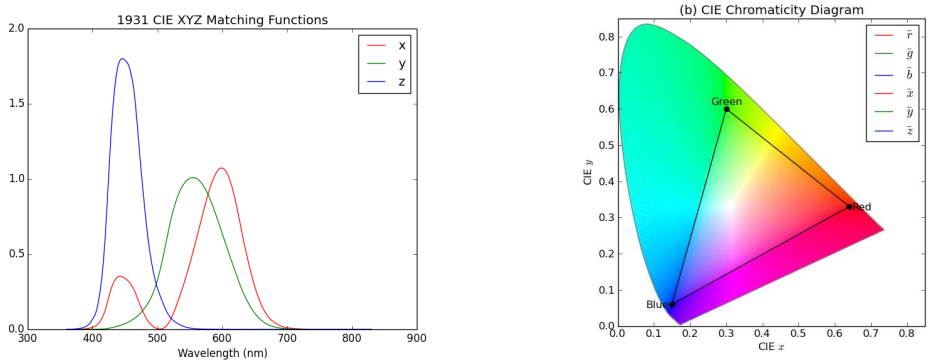


Figure 1.1: **(a)** The CIE-color-matching functions define how the XYZ representation of a color can be obtained from its spectral power spectrum. Each curve corresponds to one of the tristimulus values.
(b) The CIE chromaticity diagram displays the colors that can be perceived by humans in the CIE xy-plane. The triangle marks the colors that can be represented using the sRGB color space.

1.1.2 RGB color spaces

RGB color spaces are based on real, monochromatic primaries in the red, green and blue regions of the visible spectrum. Due to their connection to color reproduction they are often used in computer applications. The most commonly used RGB space is the sRGB space. Its RGB coordinates are defined with respect to the XYZ tristimulus values via

$$\begin{pmatrix} R_S \\ G_S \\ B_S \end{pmatrix} = \begin{pmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{1.6}$$

and

$$R = f(R_S) \tag{1.7}$$
$$G = f(G_S) \tag{1.8}$$
$$B = f(B_S) \tag{1.9}$$
$$f(x) = \begin{cases} 12.92x & x \leq 0.0031308 \\ 1.055x^{1/2.4} - 0.055 & \text{otherwise} \end{cases} \tag{1.10}$$
$$\tag{1.11}$$

The disadvantage of RGB spaces is that they do not correspond to human color vision in the sense that differences in this space do not correspond to the perceptual differences between colors.

1.1.3 CIE LAB

The CIE LAB color space is a color space derived from the CIE 1931 XYZ space with the aim of providing a perceptually uniform color space. A color space is said to be perceptually uniform if two pairs of colors in the color space that are equally far away from each other also appear equally different. The CIE LAB space has three dimensions or channels l,a,b. The l channel represents lightness, i.e. how bright an object appears. The a and b channels represent the two independent color components, namely red/green and yellow/blue. The l,a,b components can be obtained from the XYZ tristimulus values as follows:

$$l = 116\, f\big(\frac{Y}{Y_W}\big) - 16 \tag{1.12}$$

$$a = 500\big(f(\frac{X}{X_W}) - f(\frac{Y}{Y_W})\big) \tag{1.13}$$

$$b = 200\big(f(\frac{Y}{Y_W}) - f(\frac{Z}{Z_W})\big) \tag{1.14}$$

$$f(t) = \begin{cases} t^{\frac{1}{3}}, & \text{if } t > (\frac{6}{29})^3 \\ \frac{1}{3}(\frac{29}{6})^2 t + \frac{4}{29}, & \text{otherwise} \end{cases} \tag{1.15}$$

The values X_W,Y_W,Z_W are the XYZ tristimulus values of a reference white point. The a,b values can be used to derive another color feature, the *chroma*. The chroma $C_{a,b}$ describes the colorfulness of a color and is defined as the euclidean distance of a color to zero in the a,b-subspace:

$$C_{a,b} = \sqrt{a^2 + b^2} \tag{1.16}$$

Although the CIE LAB space achieves perceptual homogeneity it is still not very intuitive. A more intuitive way of describing colors is given by HSV color space.

1.1.4 HSV Color Space

Similar to the CIE LAB space, the HSV space separates the concept of brightness of a color from its chromaticity. Nevertheless, compared to CIE LAB, the color description is more intuitive and similar to an artists reasoning. A color in this space is specified by its hue, its saturation and its value. The hue of a color describes the dominant wave length of the spectral power distribution. It represents what is informally described by color names such as red, blue, purple and yellow. It is given by a value between 0 and 360 and can be interpreted as an angle along a color disc. The HSV color disc is displayed in figure . The center of the disc corresponds to neutral gray. The distance of a color from the center of a disc corresponds to its saturation. The saturation (S) describes how pure a given color is. The V channel, called *value*, describes how bright a color appears.

The HSV space is defined with respect to the RGB space. In order to convert the RGB values of a given color into the HSV space, let

$$R' = \frac{R}{255} \tag{1.17}$$
$$G' = \frac{G}{255} \tag{1.18}$$
$$B' = \frac{B}{255} \tag{1.19}$$
$$C_{max} = max(R',G',B') \tag{1.20}$$
$$C_{min} = min(R',G',B') \tag{1.21}$$
$$\Delta = C_{max} - C_{min}. \tag{1.22}$$

The corresponding HSV values are then given by

$$H = \begin{cases} 60°(\frac{G'-B'}{\Delta} \bmod 6), & C_{max} = R' \\ 60°(\frac{B'-R'}{\Delta} + 2), & C_{max} = G' \\ 60°(\frac{R'-G'}{\Delta} + 4), & C_{max} = B' \end{cases} \tag{1.23}$$
$$S = \frac{\Delta}{C_{max}} \tag{1.24}$$
$$V = C_{max}. \tag{1.25}$$

1.2 Image Segmentation

Digital images today usually contain several millions of pixels. An image given by a grid of pixels is what is usually referred to as low-level information: It only specifies the color of given patches of a scene and their relative position. The aim of computer vision is to extract high-level information, i.e. information about objects present in the scene, from the given low-level information. Since it is usually not possible to infer much about a scene from only one pixel, it is often desirable to group pixels into larger sets. This task is called image segmentation. The approach taken in this work builds on a technique called superpixels.

1.2.1 Superpixels

Superpixels are locally coherent patches of pixels of similar colors. The idea behind this approach is that pixels that are close locally as well as in a certain color space are likely to belong to the same object in the original scene and can thus be used to describe this object. Apart from containing more relevant information, superpixels also facilitate further analysis by reducing the computational complexity. An efficient and powerful superpixel algorithm is presented by Achanta et al. in [7]. The *Simple linear iterative*

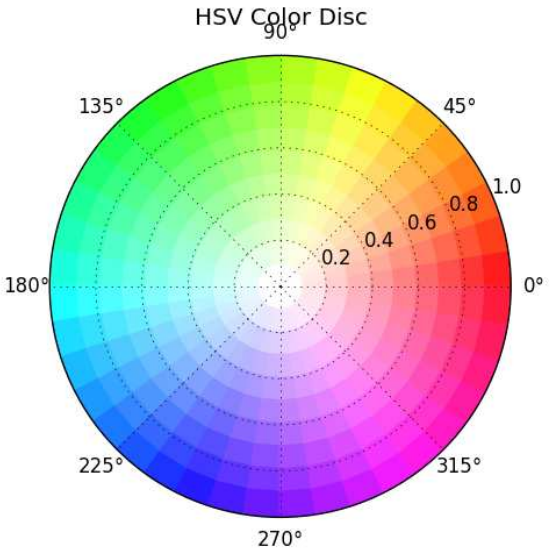


Figure 1.2: The HSV Color Disc. The plot shows the colors of the HSV color space for $V = 1$. This results in a color disc with pure colors on its edge and the line of gray at its center.

clustering algorithm (SLIC) generates clusters in the five-dimensional space consisting of the two spatial dimensions x,y and the the color dimensions of the CIE LAB space. The algorithm starts out with a square grid of superpixels centers. Then in each iteration, the algorithm assigns to each center pixels that are close spatially as well as with respect to color. This is repeated until convergence. In a final step stray pixels are eliminated by assigning them to the nearest neighboring cluster. Closeness in the combined space is defined using the distance measure D_S defined by

$$d_{lab}(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}_l - \mathbf{y}_l)^2 + (\mathbf{x}_a - \mathbf{y}_a)^2 + (\mathbf{x}_b - \mathbf{y}_b)^2} \tag{1.26}$$

$$d_{xy}(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}_x - \mathbf{y}_x)^2 + (\mathbf{x}_y - \mathbf{y}_y)^2} \tag{1.27}$$

$$D_S = d_{lab} + \frac{k}{S} d_{xy} \tag{1.28}$$

Here, d_{lab} is the euclidean distance in the CIE LAB space and d_{xy} the euclidean distance of the pixel coordinates. S is the interval length of the initial grid. The parameter k of the algorithm determines how fast the color distance is outweighed by the local distance. In this way it controls the compactness of the resulting superpixels. High values of k lead to more compact superpixels at the price of reduced homogeneity. On the other hand, small values of k result in more uniform superpixels with less regular boundaries.The algorithm is quite similar to the k-means clustering algorithm but by exploiting spatial information it achieves better runtime.

1.3 Texture Descriptors

A large amount of the information in an image is contained in its structure. The local structure of an image region is referred to as its texture. A texture descriptor consists of one or more values describing this structure. They are useful for comparing different regions in images, because they summarize the information contained in them and sometimes even provide independence of certain imaging conditions.

1.3.1 Local Binary Patterns

A powerful texture descriptor is the *local binary pattern* (LBP) operator [8]. It describes the local texture pattern in the neighborhood of a pixel in a grayscale image. Figure 1.3 displays how the LBP operator for a given pixel is computed. The 8-connected neighbors of a given pixel are thresholded using the pixels gray value. The binary values of the neighbors are then interpreted as a bit sequence. This bit sequence can then be interpreted as an unsigned integer assigning a value between 0 and 255 to all possible neighborhood configurations. Through the thresholding the LBP operator achieves independence of illumination conditions, which is a useful property when the operator is used to compare textures. A given bit string is said to be *uniform*, if less than three transitions from 1 to 0 or 0 to 1 occur in the string. There are 58 uniform binary patterns: 00000000,00000001,00000011,00000111,00001111,00011111,00111111,0111111,11111111 and their circular permutations. The texture of a given image region can be described by computing a histogram over the values of the LBP operator for each pixel in the region. This results in a histogram with $n = 256$ bins. As shown in [9], the expressiveness of the texture descriptor can be increased by accumulating all occurrences of non-uniform patterns in only one bin. This results in a histogram with 59 bins: 58 bins for uniform patterns and 1 for the non-uniform patterns. This technique can be extended to color images by considering each color channel separately.

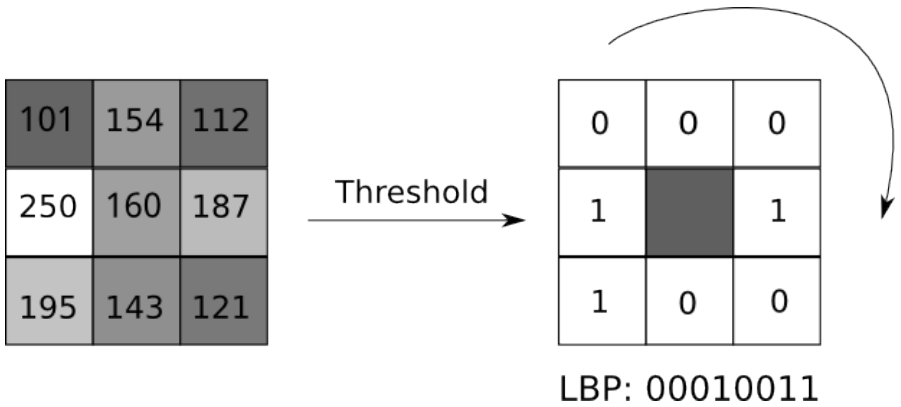


Figure 1.3: The local binary pattern operator describes the local texture in the neighborhood of a given pixel. The neighboring pixels are thresholded by the pixels grayvalue. The resulting binary values are then combined into a binary string describing the configuration.

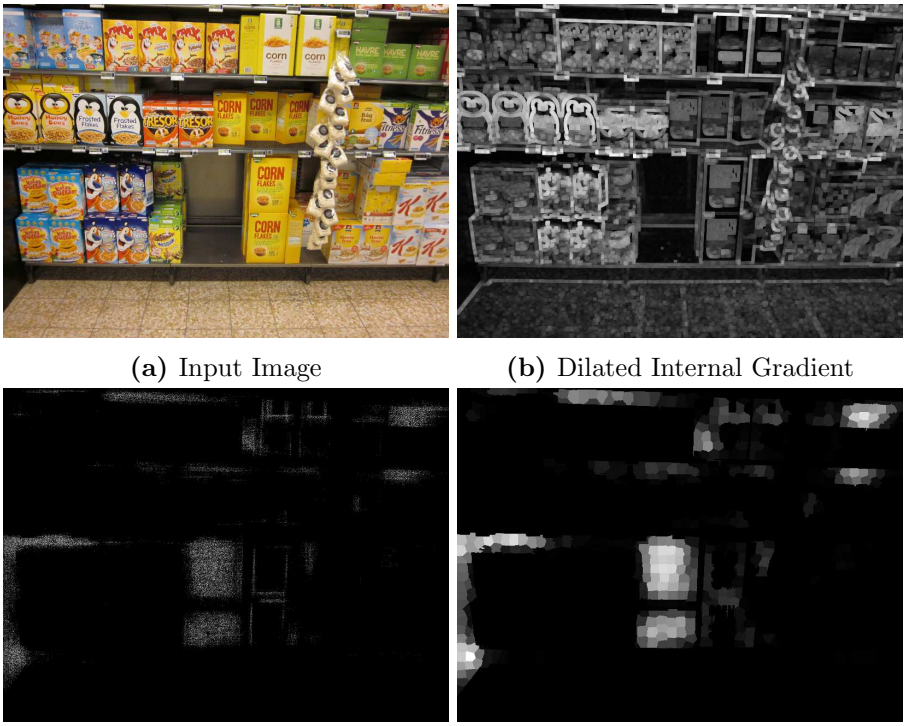
1.3.2 Agent Based Texture Descriptors

The idea of agent-based methods, is to simulate a population of agents in the image, whose dynamics are influenced by the local image properties. The density of the agents in a given region can then be used as a descriptor of the local structure of the image. Agent based techniques have been successfully applied to image processing problem, for example in [10].

The method used here is based on agents that move around in the image at a constant speed. Each agent has a current position and moving direction. In each time step an agents moves one pixel into its current direction. Moreover, an agent may give birth to another agent, die or switch direction. The chance that an agent gives birth to another agent is defined by the birth rate r_b, which is given as a parameter of the method. If a new agent is born it moves in the opposite direction than its ancestor. There are three ways an agent can die. Besides dying when an agent leaves the image, an agent also dies when the number of agents at a given pixel position is higher than a given threshold t_c. Finally, at each step an agent has a certain chance of dying that depends on the gray value of the pixel at its current position. The death probability is specified by the parameter t_D. If the gray value of the corresponding pixel is larger than t_D, the agent dies with a probability of 1. Otherwise, the death rate is proportional to the ratio of the gray value of the current pixel and t_D. Apart from this, at each step an agent has a certain chance of changing direction, this is given by the parameter r_{dir}.

The initial population consists of a given number N_A of agents, whose positions and moving directions are chosen uniformly at random. Then T time steps are performed. The results of the method are the position of the agents after T time steps.

The underlying image that will be used here is a dilated version of the internal gradient. The internal gradient of an image is given by the difference of an image and an eroded version of it. This will lead to high death rates in regions with distinct structure and lower death rates in more homogeneous regions. Figure 1.4 displays the results of the agent based texture descriptor applied to a sample image.



(a) Input Image **(b)** Dilated Internal Gradient

(c) Agent Distribution **(d)** Number of Agents per Superpixel

Figure 1.4: The agent based texture descriptor applied to the sample image. The agent method is applied to a dilated version of the internal gradients of the input image, as displayed in **(b)**. The resulting distribution of the agents over the image is displayed in **(c)**. The number of agents per superpixel are displayed in **(d)**.

2

Machine Learning

The field of machine learning is concerned with the problem of developing automatic methods to identify the internal structure inherent to data. In the context of image processing machine learning is usually applied in order to find ways of extracting high-level information from images. For example in the hole-detection setting, the problem is to design an algorithm to find holes in shelf images. This requires the definition of conditions that a set of pixels has to satisfy to be classified as a hole. Here, machine learning can be used to learn those relations from a given set of test data. For a broader and more detailed treatment of the topic the reader is referred to [3, 11, 12].

2.1 Supervised Learning

The type of learning that will be considered throughout this work is called *supervised learning*. The supervised learning problem is to find relations between a set of input variables and a set of output variables from observations of both sets of variables. The hole detection problem may be viewed as an instance of this problem: The input are a set of features extracted from a given region of an image and the output is yes or no depending on whether the region is part of a hole. The observations to learn from are a set of image regions displaying holes and non-holes. Formally, this may be expressed as follows. Assume there are p scalar input variables $X_1, \ldots, X_p$. The set of input variables is denoted by X. For now, let Y be the only output variable, that takes values in $\{-1,1\}$. This is the setting if only a single image region is considered at a time. Y represents the class of the image region with 1 corresponding to the region being a hole and -1 to the region not being a hole. Later, also the case when Y is a set $Y = (Y_1, \ldots, Y_n)S$ of binary output variables will be considered. A concrete sample from X or Y is written as a vector $\mathbf{x}$ or $\mathbf{y}$ respectively. If there is only one output variable then a sample is denoted by y. The terms *input* and *sample* will be used to refer to concrete instantiations of X. Yet, the word sample will be used for an element with known corresponding output,

whereas input will be used more generally for a value of X. The term *features* is used to denote the input variables X with *feature* or *input space* referring to the space of all values X can take on. The relation between the input X and the output Y is modeled using a function $f : X \to Y$ called *decision rule* or *hypothesis*. The predicted outcome of an input $\mathbf{x}$ is given by $f(\mathbf{x})$. For binary classification f is often of the form

$$f(\mathbf{x}) = sign(g(\mathbf{x})) \tag{2.1}$$

with $g : X \to \mathbf{R}$ a function that maps an input $\mathbf{x}$ into $\mathbf{R}$. The data that is available to learn from is called the *training set* S. It consists of N observations of X and Y:

$$S = \{(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_N,\mathbf{y}_N)\} \tag{2.2}$$

The observations in S are assumed to be i.i.d. samples from the joint distribution $\mathcal{D}_{X,Y}$ of X,Y.

2.1.1 Statistical Decision Theory

Statistical decision theory is the framework that is used for the development of methods for machine learning. Assume, the hypothesis f is used to predict the outcome of an input $\mathbf{x}$. Now, define a *loss function* $L : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$. The role of the loss function $L(\mathbf{y},f(\mathbf{x}))$ is to quantify the quality of the prediction $f(\mathbf{x})$ compared to the correct output $\mathbf{y}$. The loss function can be used as a criterion for choosing f. Generally, the aim is to find the decision function f that minimizes the *expected prediction risk*:

$$\mathcal{R}(f) = \mathbf{E}_{X,Y}(L(Y,f(X))) \tag{2.3}$$

The expectation here is taken with respect to the joint distribution $\mathcal{D}_{X,Y}$ of X and Y. Of course, $\mathcal{D}_{X,Y}$ is generally unknown. The expected prediction risk may be estimated by the *empirical prediction risk* with respect to a given training set S:

$$\mathcal{R}_S(f) = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{y},\mathbf{x}_i) \tag{2.4}$$

For example, a common choice for the loss function L in a classification setting is the so called 0/1-loss assigning cost 1 to a misclassification:

$$L(\mathbf{y},f(\mathbf{x})) = \begin{cases} 1 & ,\mathbf{y} \neq f(\mathbf{x}) \\ 0 & ,\text{otherwise} \end{cases} \tag{2.5}$$

By parametrizing f, a decision rule can now be found by minimizing the empirical risk. However, the approach of minimizing the empirical risk with respect to a certain loss function may turn out to be too short-sighted. Finding a sufficiently flexible function that yields zero empirical risk over the training set is always possible. But does that mean that the resulting hypothesis performs well on unseen samples? The answer is

no. A problem that is likely to occur when the empirical risk is required to be very low is *overfitting*. Overfitting occurs when the classifier adapts too closely to the training data. This may lead to a decrease in performance on new data either due to noise in the training set or because the training set fails to convey the true structure of the underlying relation. A general technique to avoid overfitting in the learning phase is to introduce a regularization term into the risk leading to the so called *regularized risk*:

$$\mathcal{R} = \mathbf{E}_{X,Y}(L(Y,f(Y))) + J(f) \tag{2.6}$$

Here, J is a functional that penalizes high frequencies in f. The approach of formulating the learning problem as a minimization problem of some loss-function turns out to be very powerful. A lot of machine learning techniques can be shown to minimize a certain loss-function although they were motivated in a different way.

2.1.2 Generalization Theory

The aim of generalization theory is to examine the capability of learning methods to generalize to indepedent test data. Measuring the generalization performance of a machine learning technique is important to compare different models and to assess the overall quality one can expect from predictions on independent inputs. The *generalization error* is defined as

$$\mathrm{err}_S = \mathbf{E}_{X,Y}(\mathbf{I}[f(X) \neq Y]). \tag{2.7}$$

Here, $\mathbf{I}$ is the indicator function that is 1 if $f(\mathbf{x})$ and $\mathbf{y}$ are not equal and zero otherwise. The expectation here is taken over X and Y, while the training set S is held fixed. Averaging the generalization error over all possible training sets S yields the expected error:

$$\mathrm{err} = \mathbf{E}_{X,Y,S}(\mathbf{I}[f(X) \neq Y]) \tag{2.8}$$

Nevertheless, the value that best describes the performance of a given classifier is the generalization error for a given training set S, err_S. The sample analogue of the generalization error over the training set is called the *training error*, denoted by $\overline{\mathrm{err}}_S$. The training error, however, is not a suitable estimator for the generalization error since it tends to underestimate it. The training error constantly decreases with increasing complexity of the model. The generalization error, on the contrary, often increases when the model complexity becomes too high. In order to estimate the performance of a model, some of the available data is held back during the training process. If the amount of available data permits, the general approach is to define a *validation* and a *test* set. The validation set is used to find suitable meta parameters for the model and to compare different models. Finally, the test set is used to estimate the overall performance of the system.

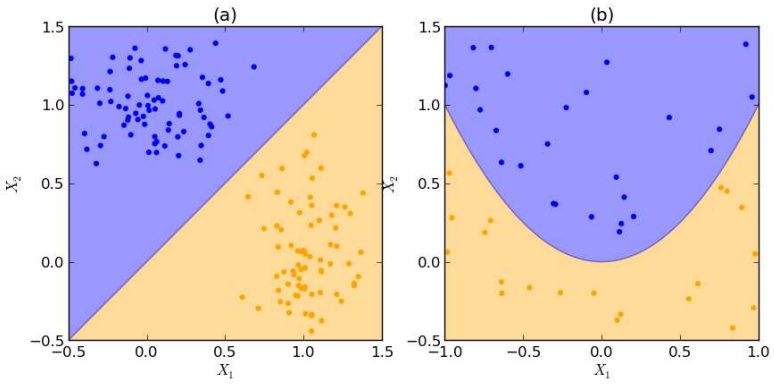


Figure 2.1: Two two-dimensional toy problems displaying the concept of linear decision boundaries in the input space. Learning here amounts to using the training data to find a suitable splitting of the input space. The example in plot **(a)** can easily be separated using a linear decision boundary whereas the example in **(b)** cannot.

2.2 Support Vector Machines

Support vector classifiers and support vector machines are motivated from a geometric view on the data. A sample $\mathbf{x}$ from the input space X can be viewed as a point in a p-dimensional space. In the binary classification setting, a hypothesis f assigns a label to every point in the input space and thus defines a binary partition on X. A simple way to partition a given space into two regions is by defining a *separating hyperplane*, also called a *linear decision boundary*. This is the concept behind support vector classifiers.

For the case of two input variables, i.e. a two-dimensional input space, the problem can be visualized in a plot. Consider the examples displayed in figure 2.1. Given are data points belonging to two different classes, here marked by their color. In the first figure, the two classes are samples from two scaled bivariate normal distributions. One of them is centered around (0,1) and the other around (1,0). A linear decision boundary can easily be found by inspection, for example $f(x_1,x_2) = x_2 - x_1$. Of course, this problem is particularly simple: The two classes are well separated and the dimensionality is low. As shown in the second figure in 2.1, the linear decision boundaries fail in the case of a nonlinear relation between X_1 and X_2.

In the next section a formalized approach to finding separating hyperplanes in the input space is presented.

2.2.1 Linear Decision Boundaries

Assume a given training set $S = \{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_N,y_N)\}$ with $\mathbf{x}_i \in \mathbf{R}^p$ and $y_i \in \{-1,1\}$ for $i = 1,\ldots,N$. A hyperplane is a set of points $\{\mathbf{x} : \boldsymbol{\theta}^T\mathbf{x} + \theta_0 \ = 0\}$ satisfying the equation

$$\boldsymbol{\theta}^T\mathbf{x} + \theta_0 = 0. \tag{2.9}$$

Here, $\boldsymbol{\theta}^T\mathbf{x}$ is the dot product of $\boldsymbol{\theta}$ and $\mathbf{x}$. $\boldsymbol{\theta}$ is the normal vector to the hyperplane. Setting $f(\mathbf{x}) = sign(\boldsymbol{\theta}^T\mathbf{x} + \theta_0)$, the resulting hypothesis assigns label -1 to the points lying on one side of the plane and label 1 to all others. The objective is thus to find the parameters $\{\boldsymbol{\theta},\theta_0\}$ that represent an optimal separating hyperplane. But what are the characteristics of such an optimal hyperplane? If the classes are separable, a natural choice for the optimal separating hyperplane is the plane that maximizes the minimal distance between the plane and the points from each class. This distance is also called the *margin*. If the classes are not separable one may still try to minimize the distance between the points from each class and the hyperplane but allow for some of them to lie on the wrong side of the margin. Formally, this may be expressed as the following minimization problem

$$\begin{aligned} \min_{\boldsymbol{\theta},\theta_0,\boldsymbol{\xi}} \quad & ||\boldsymbol{\theta}|| \\ \text{subject to} \quad & y_i(\boldsymbol{\theta}^T\mathbf{x}_i + \theta_0) \geq 1 - \xi_i, \quad i = 1,\ldots,N \\ & \xi_i \geq 0, \qquad\qquad\qquad\quad i = 1,\ldots,N \\ & \textstyle\sum_{i=1}^N \xi_i \leq const \end{aligned} \tag{2.10}$$

The rationale behind this formulation is as follows. Let d_i denote the distance of sample $\mathbf{x}_i$ to the decision boundary. Define d_i to have positive sign if $\mathbf{x}_i$ lies on the correct side of the boundary and negative sign otherwise. The expression $y_i(\mathbf{x}^T\boldsymbol{\theta} + \theta_0)$ then corresponds to the product of d_i and the length of the normal vector $||\boldsymbol{\theta}||$:

$$|y_i(\mathbf{x}^T\boldsymbol{\theta} + \theta_0)| = d_i||\boldsymbol{\theta}|| \tag{2.11}$$

The condition $y_i(\mathbf{x}^T\boldsymbol{\theta} + \theta_0) \geq 1$ thus ensures that $||\theta||$ is larger than the inverse distance of each point from the hyperplane. Hence minimizing $||\theta||$ is the same as maximizing the minimum distance of the points from the separating hyperplane. In order to allow for some points to lie on the wrong side of the margin the slack variables ξ_i are introduced. The value of ξ_i represents how far relative to the margin M the point x_i lies on the wrong side of the margin. Figure 2.2 illustrates the situation in a two-dimensional feature space. A general formulation of the support vector classifier reads

$$\begin{aligned} \min_{\boldsymbol{\theta},\theta_0,\boldsymbol{\xi}} \quad & \tfrac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} + C\textstyle\sum_{i=1}^N \xi_i \\ \text{subject to} \quad & \xi_i \geq 0, \qquad\qquad\qquad\quad i = 1,\ldots,N \\ & y_i(\boldsymbol{\theta}^T\mathbf{x}_i + \theta_0) \geq 1 - \xi_i, \quad i = 1,\ldots,N \end{aligned} \tag{2.12}$$

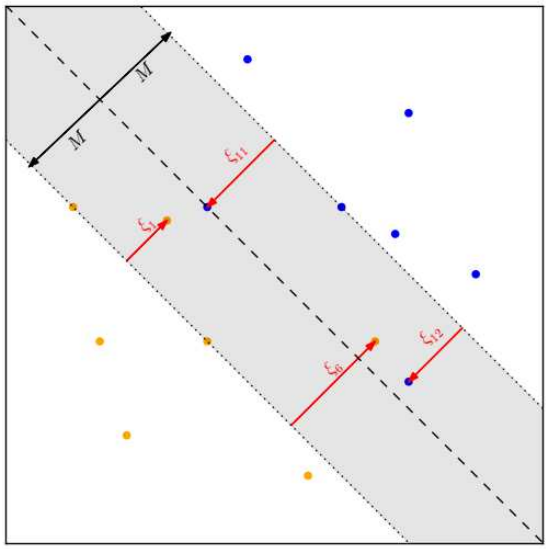


Figure 2.2: A separating hyperplane in $\mathbf{R}^2$. If the two classes are separable, the optimal separating hyperplane is defined as the one that maximizes the margin M, i.e. the minimum distance of the samples to the plane. If the two classes are not separable, the slack variables ξ_i are introduced to allow sample $\mathbf{x}_i$ to lie on the wrong side of the margin. The value of ξ_i corresponds to how far sample $\mathbf{x}_i$ lies on the wrong side of the margin relative to the size of the margin.

which is equivalent to (2.10) with C replacing the constant limiting the sum of the slack variables ξ_i. Investigating this optimization problem further will reveal valuable information about the structure of the support vector classifier.

2.2.2 Optimization Theory

The aim of optimization theory is to provide methods for the solution and characterization of optimization problems. It turns out that applying Lagrangian theory to the optimization problem (2.12) reveals an interesting structure of the solution. Based on this structure it is possible to extend the method to non-linear decision boundaries by mapping the input space into higher- or even infinite-dimensional spaces.

The Lagrangian function corresponding to the support vector optimization problem (2.12) is given by

$$L(\boldsymbol{\theta},\theta_0,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\Big(\mathbf{y}_i(\boldsymbol{\theta}^T\mathbf{x}_i + \theta_0) - (1-\xi_i)\Big) - \sum_{i=1}^{N}\mu_i\xi_i. \tag{2.13}$$

The corresponding dual function may be obtained by differentiating with respect to β,β_0 and ξ_i for $i = 1 \ldots, N$ and setting the respective derivatives to zero. This gives

$$\sum_{i=1}^{N} y_i \alpha_i \mathbf{x}_i = \boldsymbol{\theta} \tag{2.14}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.15}$$

$$C - \mu_i - \alpha_i = 0 \tag{2.16}$$

The significance of (2.14) is that the normal vector of the separating hyperplane is given by a weighted sum of the samples in the training set. Inserting the above equation into (2.13) the objective function of the dual problem is obtained.

$$L_D(\alpha_1, \ldots, \alpha_n) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{2.17}$$

The dual problem is thus given by

$$\begin{aligned} \underset{\alpha_1, \ldots, \alpha_N}{max} \quad & L_D(\alpha_1, \ldots, \alpha_N) \\ \text{subject to} \quad & \alpha_i \geq 0, \qquad i = 1, \ldots, N \\ & \alpha_i \leq C, \qquad i = 1, \ldots, N \\ & \textstyle\sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{2.18}$$

The primal problem is convex with affine constraints and domain $\mathbf{R}^N$ and thus the duality gap is zero. Consequently, the maximization problem (2.18) is equivalent to the original minimization problem. What is interesting here is that the evaluation of the dual function involves the input only through a weighted sum of the vector products over the training set. This property will prove to be very useful for realizing non-linear decision boundaries. Finally, the Karush-Kuhn-Tucker complementarity conditions yield

$$\alpha_i \big(y_i (\boldsymbol{\theta}^T \mathbf{x}_i + \theta_0) - (1 - \xi_i) \big) = 0 \tag{2.19}$$

$$\mu_i \xi_i = 0. \tag{2.20}$$

This together with (2.14) implies that the normal θ is a linear combination of the input vectors with non-zero α_i only for those points that satisfy

$$y_i (\boldsymbol{\theta}^T \mathbf{x}_i + b_0) = 1 - \xi_i. \tag{2.21}$$

That means that the SV classifier depends only on a subset of the samples in the training set. The vectors $\mathbf{x}_i$ with nonzero α_i are called the *support vectors*. The dual perspective also clarifies the influence of the parameter C on the support vector classifier. Condition (2.20) together with (2.16) and the positivity constraint on α_i for $i = 1, \ldots, N$ guarantees

that $\alpha_i \leq C$ holds. Since the value of the dual variable reflects the influence of the corresponding constraint on the objective function, the constant C limits the influence a point from the training set can have on the solution. In this way C limits the influence of outliers in the training set, which would otherwise have large values in the corresponding dual variables [13].

2.2.3 Overcoming Linearity

It is not hard to imagine problems where linear decision boundaries are insufficient. Already a rather simple problem as the one depicted in figure 2.1 **(b)** cannot be solved satisfactory using a linear decision boundary. Linearity may be overcome by mapping the input space into another, possibly higher- or even infinite-dimensional, space using a mapping h. Such methods are known as *kernel methods*. For the example given in figure 2.1, one may choose $h : (x_1,x_2) \mapsto (x_1^2,x_2)$. The resulting space is plotted in figure 2.3 **(a)**. As can be seen from the plot, the two classes are linearly separable in the resulting space $h(X)$. The resulting decision boundary in the original space is displayed in figure 2.3 **(b)**. More generally, non-linear decision boundaries can be obtained by mapping the input space into a transformed feature space and computing the decision boundary there. The idea of kernel methods is to exploit the fact that the input samples enter the optimization problem only through an inner product. In some cases the inner product in the resulting feature space can be computed efficiently. In this way, computations in higher- or infinite-dimensional feature spaces can be avoided and such computations become tractable.

The main concept of kernel methods is the *kernel function*. A kernel function may be viewed as a generalized version of the dot product. In this way it implicitly defines another feature space. Consider a kernel function $K : \mathbf{R}^p \times \mathbf{R}^p \to \mathbf{R}$ with possibly finite eigen-expansion

$$K(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x})\phi_i(\mathbf{y}). \tag{2.22}$$

Note the structure similar to the dot product $\phi_x^T \phi_y$ of two vectors with $\phi_x = (\phi_1(\mathbf{x}),\ldots,\phi_n(\mathbf{x}),\ldots)$ and $\phi_y = (\phi_1(\mathbf{y}),\ldots,\phi_n(\mathbf{y}),\ldots)$. Replacing $\mathbf{x}^T\mathbf{y}$ with $K(\mathbf{x},\mathbf{y})$ thus amounts to the inner product of the input vectors mapped into a different features space given by the eigenvectors of K:

$$X = (X_1,\ldots,X_p) \mapsto \phi(X) = (\sqrt{\lambda_1}\phi_1(X),\ldots,\sqrt{\lambda_n}\phi_n(X),\ldots) \tag{2.23}$$

In literature this space is called the *implicit feature space*, because the space appears only implicitly through the use of K. This leads to an optimization problem corresponding to (2.12):

$$\begin{aligned} \underset{\boldsymbol{\theta},\theta_0,\boldsymbol{\xi}}{min} \quad & \tfrac{1}{2}K(\boldsymbol{\theta},\boldsymbol{\theta}) + C\textstyle\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & \xi_i \geq 0, \qquad\qquad i = 1,\ldots,N \\ & y_i(\mathbf{x}_i^T\boldsymbol{\theta} + \theta_0) = 1 - \xi_i, \quad i = 1,\ldots N \end{aligned} \tag{2.24}$$

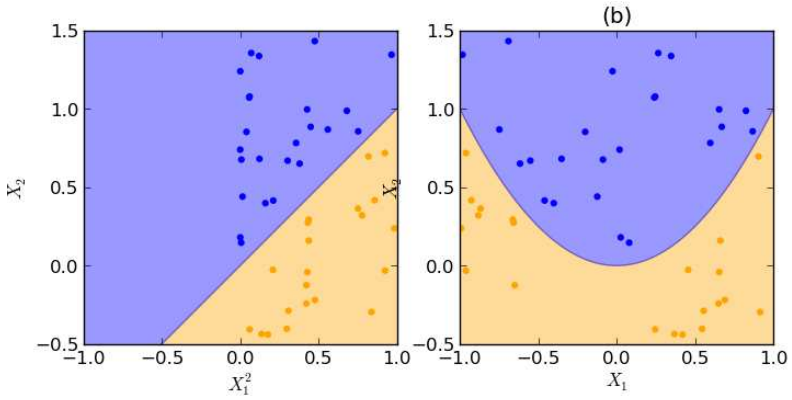


Figure 2.3: Plots of the second example problem from figure 2.1. **(a)** displays the samples in the augmented space $h(X)$. h simply maps the first input variable X_1 to its square and leaves the second input variable X_2 unchanged. The two classes are linearly separable in the augmented space. In **(b)**, the decision boundary in the original space is displayed.

The solution to the problem corresponds to a hyperplane in the implicit feature space. The decision function in the original space is given by

$$f(\mathbf{x}) = K(\mathbf{x},\theta) - \theta_0 \tag{2.25}$$

$$= \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i) - \theta_0 \tag{2.26}$$

and thus requires at most N evaluations of K. The function f lies in the space $\mathcal{H} = \{K(\cdot,\mathbf{x}) : \mathbf{x} \in \mathbf{R}^P\}$ spanned by the eigenvectors of K. The norm on $\mathcal{H}$ is the norm defined by:

$$||f||^2_{\mathcal{H}} = ||\sum_{i=1}^{\infty} c_i \phi_i(x)||^2_{\mathcal{H}} = \sum_{i=1}^{\infty} \frac{c_i^2}{\lambda_i} \tag{2.27}$$

Such a space $\mathcal{H}$ is called *reproducing kernel Hilbert space* (RKHS). Common examples of kernel functions are

dth-Degree polynomial: $K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T\mathbf{y})^d$

Radial basis functions: $K(\mathbf{x}, \mathbf{y}) = exp(-\gamma||\mathbf{x} - \mathbf{y}||^2)$

Neural network: $K(\mathbf{x}, \mathbf{y}) = tanh(\kappa_1 \mathbf{x}^T \mathbf{y} + \kappa_2)$.

Figure 2.4 displays the resulting decision boundary of a SVM based on a radial basis function kernel (RBF SVM) applied to the toy problem from figure 2.1 **(b)**. The parameter γ determines the smoothness of the decision boundary. For small values of γ the boundary becomes very smooth, whereas large values of γ lead to a very irregular decision boundary.

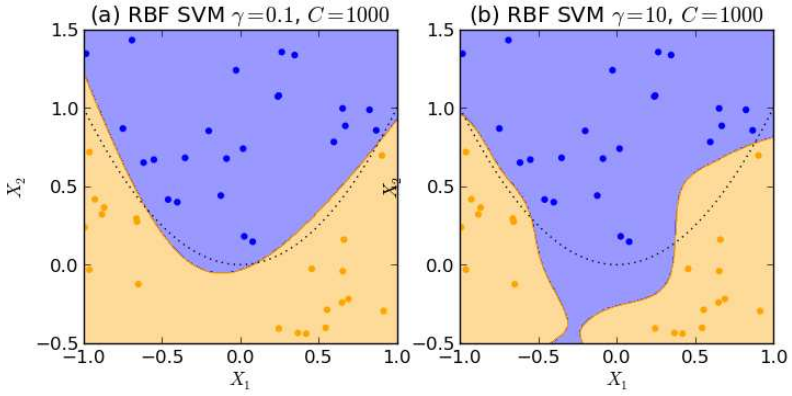


Figure 2.4: Radial basis function kernel. The decision boundaries of an RBF SVM applied to the toy problem from figure 2.1 **(b)**. The parameter γ determines the smoothness of the decision boundary. $\gamma = 0.1$ leads to a smooth decision boundary, whereas $\gamma = 10$ leads to a very irregular decision boundary. The true decision boundary is represented by the dotted line.

The support vector classifier that is used in the implementation uses a radial basis function (RBF) kernel. The parameter γ of the RBF kernel together with the parameter C from the formulation of the SVM optimization problem in (2.12) are meta parameters of the method. This means that they cannot be learned from data, but have to be determined by trying out different values and choosing those that yield the best performance.

2.3 Boosting

The term boosting designates a class of machine learning techniques, that achieve high classification performance by combining a large number of classifiers with low

performance, so called *weak classifiers* or *weak learners*. A weak classifier is a machine learning technique that, given a set of training samples, issues a decision rule that yields only slightly better results than guessing. The key idea of boosting is to train each of the weak classifiers on a modified training set, forcing them to concentrate on different aspects of the data. In the following the foundations of Boosting are presented together with a concrete machine learning technique, the AdaBoost algorithm. For a more detailed treatment the reader is referred to [11] and [14].

2.3.1 Strong and Weak learnability

A formal definition of boosting is given in terms of the concepts of *strong* and *weak learnability*. The concept of strong learnability, or *probably approximately correct* (PAC) *learnability*, is a formal definition of what is meant by learning a relation of an input X and an output Y from data. Strong learnability assumes the existence of a *target function* g, that defines the relation between X and Y. Since nothing can be learned from the training set if the relation between X and Y is purely random, g is assumed to be deterministic. Moreover it is assumed, that g comes from a class C of target functions, that summarizes prior knowledge about the target function. Then a class C is strongly PAC learnable if for every $g \in C$, every distribution $\mathcal{D}_X$ over the input space X and any positive ϵ and δ, there exists a learning algorithm A and a number N of training samples, such that A, trained on a training set S consisting of N training samples, outputs a hypothesis f that satisfies

$$Pr(\text{err}_S \geq \epsilon) \leq \delta. \tag{2.28}$$

The probability here is taken with respect to any selection of training samples. In other words, strong PAC learnability means that the target function f can with very high probability be learned almost correctly given a sufficient number of training samples. Weak learnability is a relaxation of the demanding strong learnability model. A target class C is *weakly* PAC *learnable* if for some $\gamma > 0$ there exists a learning algorithm A that for every distribution $\mathcal{D}_X$ over the input space and for every $\delta > 0$ takes as input N training samples and outputs a hypothesis $f(x)$ such that

$$Pr(\text{err}_S \geq \frac{1}{2} - \gamma) \leq \delta. \tag{2.29}$$

Thus, weak PAC learnability formalizes that the relation between X and Y can with high probability be learned with an accuracy just slightly better than guessing. It can be shown that the weak and strong learnability are actually equivalent. This means that there exists a technique for transforming a method for weak learning into a method for strong learning. This is exactly the defining property of boosting methods. They describe how a given target class can be learned in the strong sense, given an algorithm that realizes weak learnability. Boosting methods thus rely on the assumption that the given target class is weakly learnable. This assumption is called the *weak learnability assumption*.

2.3.2 AdaBoost

In this section the general AdaBoost algorithm as given in [14] is presented. The algorithm assumes the existence of a weak learner, an algorithm that implements weak learnability for the given learning problem. The AdaBoost algorithm describes how this algorithm can be used to learn the target function in the strong sense. The weak classifier algorithm is treated as a black box, assuming only that it takes a set of training samples and outputs a hypothesis $f(x)$.

Algorithm 2: AdaBoost

Input: Training set $(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_N,y_N)$

Initialize: $p_1(i) = \frac{1}{N}$ for all $i = 1,\ldots,N$,i.e. the uniform probability distribution function over the training set

For $t = 1,\ldots,T$ **do**

- Train weak learner using the distribution corresponding to p_t to obtain hypothesis f_t
- Compute the weighted training error ϵ_t:

$$\epsilon_t = \sum_{i=1}^{N} \mathbf{I}(f_t(\mathbf{x}_i) \neq y_i) p_t(i) \tag{2.30}$$

- Compute the weight α_t:

$$\alpha_t = \log\left(\frac{1-\epsilon_t}{2\epsilon_t}\right) \tag{2.31}$$

- Update p_{t+1}:

$$p_{t+1}(i) = \frac{p_t(i)\exp(-\alpha_t y_i f_t(\mathbf{x}_i))}{Z_t} \tag{2.32}$$

where $Z_t = \sum_{i=1}^{N} p_t(i)\exp(-\alpha_t y_i f_t(\mathbf{x}_i))$ is a normalization factor

Output: The final hypothesis $F(x) = sign\left(\sum_{t=1}^{T} \alpha_t f_t(\mathbf{x})\right)$

The algorithm presented above runs for T iterations, producing T weak classifiers. In each iteration $t = 1,\ldots,T$ the classifier f_t is trained on a distribution over the training set given by p_t. How this is realized depends on the type of weak classifiers used. Some methods, such as decision trees, allow to integrate the weights given by p_t directly into the training process, for others a new training set has to be sampled from the original

training set using the distribution given by p_t. Moreover, in each iteration a weight α_t is computed in a way that minimizes the error over the training set. The weight α_t is defined in (2.31) to be $\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$. To see why this minimizes the expected error consider the following upper bound for the training error $\overline{\text{err}}_S$:

$$\overline{\text{err}}_S = \frac{1}{N}\sum_{i=1}^{N} \mathbf{I}[f(\mathbf{x}_i) \neq y_i] \tag{2.33}$$

$$\leq \sum_{i=1}^{N} p_1(i) \exp\big(-y_i F(\mathbf{x}_i)\big) \tag{2.34}$$

which is obtained using that $\mathbf{I}[F(\mathbf{x}_i) \neq y_i] \leq \exp\big(-y_i F(\mathbf{x}_i)\big)$, since the labels y are encoded using -1 and 1. Then, unraveling the recursion in (2.32), the distribution function p_t may be written as

$$p_{t+1}(i) = p_1(i) \times \frac{\exp\big(-\alpha_1 y_i f_1(\mathbf{x}_i)\big)}{Z_1} \times \ldots \times \frac{\exp\big(-\alpha_t y_i f_t(\mathbf{x}_i)\big)}{Z_t} \tag{2.35}$$

$$= p_1(i) \frac{\exp\big(-y_i \sum_{j=1}^{t} \alpha_j f_j(\mathbf{x}_i)\big)}{\prod_{j=1}^{t} Z_t}. \tag{2.36}$$

This expression may in turn be used to derive the following upper bound on the training error.

$$\overline{\text{err}}_S \leq \sum_{i=1}^{N} p_1(i) \exp\big(-y_i F(\mathbf{x}_i)\big) \tag{2.37}$$

$$= \sum_{i=1}^{N} p_{T+1}(i) \prod_{t=1}^{T} Z_t \tag{2.38}$$

$$= \prod_{t=1}^{T} Z_t. \tag{2.39}$$

Finally, expanding the term for Z_t using the definition of ϵ_t gives

$$Z_t = \sum_{i=1}^{N} \mathcal{D}_t(i) \exp\big(-\alpha_t y_i f_t(\mathbf{x}_i)\big) \tag{2.40}$$

$$= \exp(-\alpha_t)(1-\epsilon_t) + \exp(\alpha_t)\epsilon_t. \tag{2.41}$$

Using basic calculus it can be seen that this expression is minimized by $\alpha_t = \log\left(\frac{1-\epsilon_t}{2\epsilon_t}\right)$. Thus, in every step AdaBoost minimizes Z_t, which then appears as a factor in the upper bound on the training error. AdaBoost can therefore be viewed as a greedy method to minimize the expected training error. Apart from the weak learnability assumption, the AdaBoost algorithm makes no assumptions about the provided weak

classifier. This means that potentially every machine learning technique can be used as a weak learner. It should be noted however, that using a relatively strong weak classifier might result in a greater test error, even though the training error is reduced. This is because more accurate weak learners with higher complexity also increase the complexity of the combined classifier and may thus lead to overfitting. A common solution is to use decision trees with a low depth or stumps. This is also the approach that is taken here.

2.3.3 Decision Trees

The general idea of decision trees is to recursively partition the input space into different regions and then assign a class to each of the resulting regions. Each internal node of a tree splits the feature space in two by specifying a threshold with respect to a certain input variable. Classifying an input using a decision tree amounts to moving along the tree from its root to one of the leafs. If the value of the input variable lies below the threshold specified at the current node one proceeds to the left and otherwise to the right. Arriving at a leaf, the input is classified as the class represented by the leaf. The learning task for decision trees amounts to growing a tree that suitably partitions the input space. Since finding an optimal partition of a high dimensional space is computationally infeasible, a greedy method is applied for growing the tree. At each step of growing the tree, the problem is how to find the variable l and the threshold p at which the splitting should occur. To this let

$$\hat{p}_{m,k} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} \mathbf{I}(y_i = k) \tag{2.42}$$

be the fraction of samples $\mathbf{x}_i$ with class $k \in \{-1,1\}$ in the Region R_m corresponding to node m. Now define the following *node impurity measure* for a given node m, known as the *gini index*.

$$\sum_{k \in \{-1,1\}} \hat{p}_{m,k}(1 - \hat{p}_{m,k}) \tag{2.43}$$

Then at each node the splitting variable l and split point p are found by minimizing the sum of the impurity measures in each of the newly created nodes m_1,m_2 weighted by the number of samples in those regions.

$$N_{m_1} \sum_{k \in \{-1,1\}} \hat{p}_{m_1,k}(1 - \hat{p}_{m_1,k}) + N_{m_2} \sum_{k \in \{-1,1\}} \hat{p}_{m_2,k}(1 - \hat{p}_{m_2,k}) \tag{2.44}$$

The splitting is repeated until a given maximum depth is attained. Since trees grown in this way are vulnerable to overfitting, the resulting tree is usually pruned using a technique called *cost-complexity pruning*. But, since for Boosting the used trees are generally small, this is not necessary here. In order to use a decision tree as a weak classifier for Boosting, the sample weights given by the distribution function $p(i)$ have to be considered in the learning process. When using trees they can be incorporated directly as weights into formula (2.42).

2.3.4 Variants and Generalizations

The AdaBoost algorithm assumes binary outputs from the weak classifiers. A natural extension is allowing real valued output for the weak classifiers. This variant is known as *Real AdaBoost*. Instead of weighting f_t by a factor α_t, the value $f_t(\mathbf{x})$ is chosen to be half the logit transform of the conditional probability of $Y = 1$ given $X = \mathbf{x}$.

$$f_t(x) = \frac{1}{2} \log \left(\frac{p(Y = 1|x = X)}{p(Y = -1|X = x)} \right) \tag{2.45}$$

If decision trees are used as weak classifiers this value can be approximated using the fraction of samples from class $Y = 1$ in a given final node. It can be shown[15], that Discrete and Real AdaBoost fit a stagewise additive model to the training data that minimizes the expected loss with respect to an exponential loss function $L(y,f(\mathbf{x}) = \exp(-yf(\mathbf{x}))$. However, the logit-transform in (2.45) may lead to numerical instabilities. *Gentle AdaBoost* replaces (2.45) by the difference of the conditional probabilities:

$$f_t(x) = p(Y = 1|X = x) - p(Y = -1|X = x) \tag{2.46}$$

This leads to more stable results and Gentle AdaBoost often outperforms Real and Discrete AdaBoost.

2.3.5 Regularization

The meta parameters of the presented Boosting methods are the size J of the trees used as weak classifiers and the number M of weak classifiers. Although Boosting usually generalizes well, the performance decreases if J or M are chosen too large. An interesting property of Boosting is that its performance may increase even after the training error has been driven down to zero. This is due to the margin increasing property of boosting [14]. This means that with increasing M the classification results of the samples from every class move away further from zero, yielding more confident results. In order to find suitable parameters J and M, it is therefore necessary to monitor the performance using a separate validation set. The usual approach is fixing a value for J and then to iteratively increase M until the validation error starts to increase. This method is known as *early stopping*.

2.3.6 Interpretation

A major advantage of decision trees is their interpretability. Not only can the classifier be easily visualized, it is also possible to define a relative importance measure for the input variables. Each node in the tree corresponds to a binary split of a region. Let $\hat{i}^2$ be the improvement in the empirical squared error that is obtained by fitting a piece-wise constant decision function over the two regions compared to fitting a constant function over the hole region. The estimated relative importance $\hat{\mathcal{I}}_l^2$ of variable l is then the sum of the improvement over all nodes n that split the variable $l = var(n)$ in tree T.

$$\mathcal{I}_l^2(T) = \sum_{n \in T} \hat{i}_n^2 \mathbf{I}(var(n) = l) \tag{2.47}$$

By averaging over the trees this measure can be extended to AdaBoost classifiers based on trees:

$$\mathcal{I}_l^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_l^2(T_m) \tag{2.48}$$

2.4 Conditional Random Fields

The drawback of the presented SVM and AdaBoost machine learning methods is that they are unsuitable to model dependencies between the output variables Y. Especially in the field of image processing a lot of important information is contained in those dependencies. Such structural dependencies can to some extent be modeled using *conditional random fields* (CRF). The CRF approach to classification is essentially different to the one taken by non-structural methods such as SVMs and Boosting. Instead of considering the classification of each region independently, the whole image is considered, represented by a set of output variables $Y = (Y_1, \ldots, Y_n)$. A *Conditional random field* (CRF) models the conditional joint probability of a set of output variables Y given a set of input variables X. Conditional random fields can be conveniently described using graphical models. Nevertheless, the richer structure of CRF models makes learning and inference in those models more complicated. In this section graphical models for the description of CRFs and the foundations of learning and inference in conditional random fields will be presented.

2.4.1 Graphical Models

A *factor graph* $\mathcal{G}$ is given by a triple $(\mathcal{V}, \mathcal{F}, \mathcal{E})$. The graph consists of a set $\mathcal{V}$ of vertices and a set $\mathcal{F}$ of factors. The set $\mathcal{E} \subset \mathcal{G} \times \mathcal{F}$ of edges connects vertices with factors. A sample graph consisting of three variables and three factors is displayed in figure 2.5. Factor graphs can be used to represent joint probabilities over a set of random variables. Each variable is represented by a node in the graph. Here, the nodes are the input variables $X = (X_1, \ldots, X_p)$ and the output variables $Y = (Y_1, \ldots, Y_q)$. The probability distribution modeled by a factor graph is defined to be of the form

$$p(\mathbf{x},\mathbf{y}) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(\mathbf{x}_F,\mathbf{y}_F). \tag{2.49}$$

The shorthand $\mathbf{x}_F$,$\mathbf{y}_F$ is used to denote values of the variables that are adjacent to the factor $F \in \mathcal{F}$, also called the *scope* of the factor F. Thus, the function ψ_F is a function of the variables adjacent to the factor F. The functions ψ_F for $F \in \mathcal{F}$ are called *potentials*.

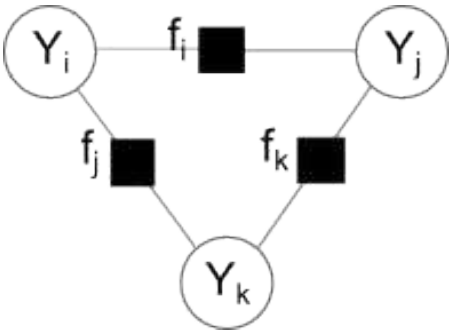


Figure 2.5: A sample graph representing a distribution over three variables(circles). The factors are displayed by the black square. They are used to model interaction between the variables.

Z is a normalization constant, called the *partition function*. A common reformulation is to instead of the potentials ψ_F use the energies

$$E_F(\mathbf{x}_F,\mathbf{y}_F) = -\log(\psi_F(\mathbf{x},\mathbf{y})). \tag{2.50}$$

The probability distribution $p(\mathbf{x},\mathbf{y})$ can then be given in terms of the energies.

$$p(\mathbf{x},\mathbf{y}) = \frac{1}{Z}\prod_{F\in\mathcal{F}} \exp\big(-E_F(\mathbf{x}_F,\mathbf{y}_F)\big) \tag{2.51}$$

$$= \frac{1}{Z}\exp\big(-\sum_{F\in\mathcal{F}} E_F(\mathbf{x}_F,\mathbf{y}_F)\big) \tag{2.52}$$

Using the energy formulation makes it possible to reformulate the problem of probability maximization as a minimization problem:

$$\underset{\mathbf{x},\mathbf{y}}{\operatorname{argmax}}\ p(\mathbf{x},\mathbf{y}) = \underset{\mathbf{x},\mathbf{y}}{\operatorname{argmax}}\ \exp\big(-\sum_{F\in\mathcal{F}} E_F(\mathbf{x}_F,\mathbf{y}_F)\big) \tag{2.53}$$

$$= \underset{\mathbf{x},\mathbf{y}}{\operatorname{argmin}} \sum_{F\in\mathcal{F}} E_F(\mathbf{x}_F,\mathbf{y}_F) \tag{2.54}$$

It is convenient to define the total energy of the model as $E = \sum_{F\in\mathcal{F}} E_F(\mathbf{y}_F,\mathbf{x}_F)$. The energy minimization approach often offers more efficient solution methods than the maximization of the probabilities [3]. Since the aim is to predict Y given the values of X, one generally considers the conditional distribution of Y given X. Since conditioning on X only changes the value of the normalization constant $Z(\mathbf{x})$, that now depends on the value of X, the same graph is used to model the probability distribution. A factor graph describing a conditional probability distribution is called a conditional random field. The conditional probability of $Y = \mathbf{y}$ given $X = \mathbf{x}$ is then given by

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\prod_{F\in\mathcal{F}} \psi_F(\mathbf{x}_F,\mathbf{y}_F) \tag{2.55}$$

2.4.2 The Image Model

So how can CRFs be used to model an image? A presegmented image containing n image regions can be modeled using n variables $Y = (Y_1, \ldots, Y_n)$. For the hole detection problem it is sufficient to assume the Y_i to be discrete, binary variables. For each region the input variables $(X_{i,1}, \ldots, X_{i,p})$ are given. Two simplifying assumptions made here are that there are only pairwise dependencies in the image and that those dependencies are limited to neighboring image regions. Each of these dependencies is modeled by a potential function. The dependency of variable Y_i on the corresponding inputs $X_{i,1}, \ldots, X_{i,p}$ is modeled by the *unary potential* $\psi_i(\mathbf{x}_i)$. The dependencies of Y_i on one of the neighboring regions Y_j is modeled using the *pairwise potential* $\psi_{i,j}(\mathbf{y}_i, \mathbf{y}_j)$. Figure 2.6 shows the model displayed by a factor graph. Each node represents one input or output variable. Each potential corresponds to one factor in the graph. The corresponding conditional probability distribution and energy are then given by

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{i \in 1,\ldots,n} \psi_i(\mathbf{x}_i, \mathbf{y}_i) \prod_{\substack{i \in 1,\ldots,n \\ j \in \mathcal{N}(i)}} \psi_{i,j}(\mathbf{x}, \mathbf{y}_i, \mathbf{y}_j) \tag{2.56}$$

$$E(\mathbf{x}, \mathbf{y}) = \sum_{i \in 1,\ldots,n} E_i(\mathbf{x}_i, \mathbf{y}_i) + \sum_{\substack{i \in 1,\ldots,n \\ j \in \mathcal{N}(i)}} E_{i,j}(\mathbf{x}_i, \mathbf{y}_j) \tag{2.57}$$

Here, the notation $\mathcal{N}(i)$ was used to denote the indices of the variables Y_j that are adjacent to variable Y_i. What remains to be specified is the form of the unary and pairwise potentials $\psi_i, \psi_{i,j}$ or their negative log-potentials $E_i, E_{i,j}$. For the unary energies E_i the form

$$E_i(\mathbf{y}_i) = \boldsymbol{\theta}_u(\mathbf{y}_i)^T \mathbf{x}_i \tag{2.58}$$

is assumed. Thus, for binary classification the unary energies are given by a weighted sum of the input variables $\mathbf{x}$ with the weights given by the parameters $\boldsymbol{\theta}(0)_U$ and $\boldsymbol{\theta}_U(1)$. Note that here the two classes are represented using $k \in \{0,1\}$ instead of $k \in \{-1,1\}$. In order to model horizontal and vertical interactions, the pairwise energy of two variables Y_i, Y_j is assumed to be of the form

$$E_{i,j}(\mathbf{y}_i, \mathbf{y}_j) = \begin{cases} \theta_H(\mathbf{y}_i, \mathbf{y}_j) + \theta_H^C(\mathbf{y}_i, \mathbf{y}_j) \exp(-\gamma_C J_{i,j}) & \text{for a horizontal edge} \\ \theta_V(\mathbf{y}_i, \mathbf{y}_j) + \theta_V^C(\mathbf{y}_i, \mathbf{y}_j) \exp(-\gamma_C J_{i,j}) & \text{for a vertical edge} \end{cases} \tag{2.59}$$

Here, $\exp(-\gamma_C J_{i,j})$ is a similarity measure for the two image regions corresponding to the variables Y_i, Y_j. $J_{i,j}$ can for example be the distance between the two regions in a certain color space. Since the value $J_{i,j}$ is neither a feature of the superpixel corresponding to Y_i nor of the superpixel corresponding to Y_j, features of this type are called *edge features*, because they are associated to an edge in the graph. The interaction between to adjacent image regions is thus assumed to consist of a constant part, $\theta_H(\mathbf{y}_i, \mathbf{y}_j)$ or $\theta_V(\mathbf{y}_i, \mathbf{y}_j)$, and a contrast sensitive part, $\theta_V^C(\mathbf{y}_i, \mathbf{y}_j) \exp(-\gamma_C J_{i,j})$ or

$\theta_H^C(\mathbf{y}_i,\mathbf{y}_j)\exp(-\gamma_C J_{i,j})$. Since there are two classes in the model, each of those parts is specified by four values. All in all, the model contains 20 parameters that have to be learned from the data. The additional parameter γ_C determines the form of the contrast function between superpixels. It is chosen by measuring the performance of the classifier on the validation set. How the parameters defining the unary and pairwise energies can be learned from the data is presented in the following section.

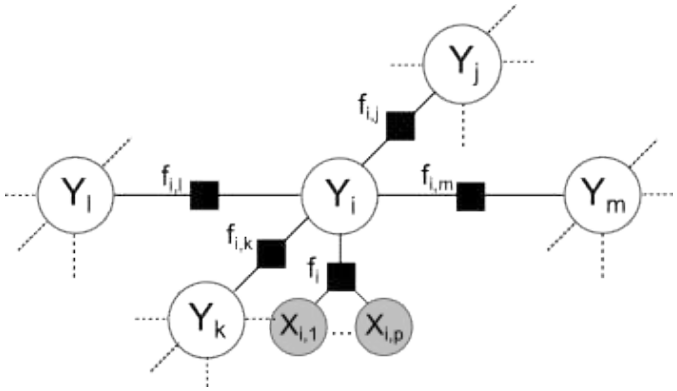


Figure 2.6: Graphical model of an image. Each output variable Y_i (white discs) corresponds to one image region. The dependencies between the regions are assumed to be pairwise and limited to neighboring regions. The dependency of a given region on its features, is represented by the corresponding input variables X_i (gray discs). Each of those dependencies is modeled by one factor in the graph.

2.4.3 Learning in CRFs

In the previous section, a model for structural dependencies of image regions was presented. In order to use this model for image classification, its parameters have to be learned from training data. For now, assume the parameters to be given by a vector $\boldsymbol{\theta}$. Let X and Y be random variables with joint distribution $\mathcal{D}_{X,Y}$. The given training data $S = \{(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_N,\mathbf{y}_N)\}$ is assumed to consists of i.i.d. samples from this distribution. Moreover, it is assumed that, given the parameters $\boldsymbol{\theta}$, a prediction rule is given of the form $f(X)$. Generally, there are two different approaches to learning the parameters $\boldsymbol{\theta}$ from the data. In *probabilistic parameter learning*, $\boldsymbol{\theta}$ is chosen so that the models conditional probability distribution $p(\mathbf{y}|\mathbf{x})$ is closest to the distribution $\mathcal{D}_{Y|X}$. Close here means in the sense of the *Kullback-Leibler divergence* between the two probability distributions. The other approach to parameter learning is called *Loss-Minimizing Parameter Learning*. Given a loss-function L, the task of loss-minimizing parameter learning is finding the value of $\boldsymbol{\theta}$ such that the expected prediction risk

$$\mathcal{R} = \mathbf{E}_{X,Y}(L(Y,f(X)) \tag{2.60}$$

is minimized. In the following, a method will be presented that realizes loss-minimizing parameter learning. Assume that the prediction given an input $\mathbf{x}$ is chosen to be the $\mathbf{y}$

that maximizes the conditional probability $p(\mathbf{y}|\mathbf{x})$. This approach may be generalized by defining a *discriminant function* $g : X \times Y \to \mathbf{R}$ from which the prediction $f(\mathbf{x})$ is obtained by maximizing $g(\mathbf{x},\cdot)$, i.e.

$$f(\mathbf{x}) = \underset{\mathbf{y}\in Y}{\operatorname{argmax}}\ g(\mathbf{x},\mathbf{y}). \tag{2.61}$$

Note that the maximum here is taken over all possible values $\mathbf{y}$ can take on. Since the aim is learning the discriminant function from the data, it is assumed that g has the form

$$g(\mathbf{x},\mathbf{y}) = \boldsymbol{\theta}^T \Psi(\mathbf{x},\mathbf{y}). \tag{2.62}$$

Here, $\Psi(\mathbf{x},\mathbf{y})$ is the joint feature vector containing the input and output variables, whereas $\boldsymbol{\theta}$ is a weight vector defining a linear function in $\mathbf{x}$ and $\mathbf{y}$. In the case where edge features are used they also have to be included in the joint feature vector. The discriminant function then is a linear function of the joint feature vector. For a suitably chosen joint feature vector $\Psi(\mathbf{x},\mathbf{y})$, this is exactly the form of the energy function of the image model. Nevertheless, attention has to be paid to the sign. Maximizing the probability corresponds to minimizing the energy, but here the aim is to maximize g. Therefore instead of the negative log potentials only the log potentials $\hat{\psi}(\mathbf{x},\mathbf{y}) = \log \psi(\mathbf{x},\mathbf{y})$ will be considered. Assume now that for a given training set there is a linear g such that all samples are classified correctly. This is the case if for all inputs $\mathbf{x}_1,\ldots,\mathbf{x}_N$, $g(\mathbf{x},\cdot)$ attains its maximum at $\mathbf{y}_i$. This may be equivalently expressed using a set of linear constraints:

$$\boldsymbol{\theta}^T(\boldsymbol{\Psi}(\mathbf{x}_i,\mathbf{y}_i) - \boldsymbol{\Psi}(\mathbf{x}_i,\mathbf{y})) \geq 0, \mathbf{y} \in Y, \mathbf{y} \neq \mathbf{y}_i, i = 1,\ldots,N \tag{2.63}$$

Here Y is used to denote the space of all possible configuration $\mathbf{y}$ can take on. In order to identify a unique solution to the problem, the value of $\boldsymbol{\theta}$ is chosen that yields the largest margin. This leads to a minimization problem similar to the formulation of the SVM classifier.

$$\begin{aligned} \min_{\theta} \quad & \tfrac{1}{2}||\theta||^2 \\ \text{subject to} \quad & \boldsymbol{\theta}^T(\boldsymbol{\Psi}(\mathbf{x}_i,\mathbf{y}_i) - \boldsymbol{\Psi}(\mathbf{x}_i,\mathbf{y})) \quad , \mathbf{y} \in Y, \mathbf{y} \neq \mathbf{y}_i, i = 1,\ldots,N \end{aligned} \tag{2.64}$$

As with the basic SVM formulation, this can be extended to allow training errors by introducing slack variables. For a more detailed treatment of the different formulations the reader is referred to [16]. The problem with the formulation (2.64) is that the number of linear constraints $N|Y| - N$ is usually very large. For graphs for example, the size of the output space $|Y|$ grows exponentially in the number of variables. It can however be shown, that the solution to this minimization problem can be found using only a small subset of the constraints. This observation leads to the *cutting plane training method* which is described in [16]. The cutting plane training method iterates solving a relaxed minimization problem using only a subset of the constraints. This subset is successively updated until the solution is found.

2.4.4 Inference

A fully specified CRF model describes a probability distribution. The task of using this distribution to draw conclusions about the output variables Y is called inference. Again, the concept of a Loss-function can be used to motivate different ways of conducting inference in CRFs. Consider the two following loss-functions:

0/1 - Loss: The 0/1-loss is zero when the prediction is correct and 1 when it deviates from the correct output, i.e.

$$L(\mathbf{y},f(\mathbf{x})) = \begin{cases} 1, & \text{if } y_i = f_i(\mathbf{x}) \text{ for } i = 1\ldots,n \\ 0, & \text{otherwise} \end{cases} \tag{2.65}$$

Hamming - Loss: As opposed to the 0/1-loss the Hamming-loss is proportional to the number of misclassified variables:

$$L(\mathbf{y},f(\mathbf{x})) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{I}(\mathbf{y}_i \neq f_i(\mathbf{x})) \tag{2.66}$$

The choice of the loss function defines which property of the distribution given by p is used to make the prediction $f(X)$. In either case, the aim is to minimize the risk. Choosing the 0/1-loss, the risk is given by

$$\begin{aligned} \mathcal{R}(f) &= \mathbf{E}_Y(L(Y,f(\mathbf{x}))|X = \mathbf{x}) && (2.67)\\ &= \sum_{Y} p(\mathbf{y}|\mathbf{x})L(\mathbf{y},f(\mathbf{x})) && (2.68)\\ &= 1 - p(f(\mathbf{x})|\mathbf{x}) && (2.69) \end{aligned}$$

Thus, minimizing the expected prediction error under 0/1-loss amounts to choosing $f(\mathbf{x})$ so that $p(f(\mathbf{x})|\mathbf{x})$ is maximized. This type of inference is called *maximum aposteriori* (MAP) inference. On the other hand, choosing the hamming loss results in the following expression for the risk.

$$\begin{aligned} \mathcal{R}(f) &= \mathbf{E}_Y(L(Y,f(\mathbf{x}))|X = \mathbf{x}) && (2.70)\\ &= \sum_{y\in Y} p(\mathbf{y}|\mathbf{x})L(\mathbf{y},f(\mathbf{x})) && (2.71)\\ &= \frac{1}{n}\sum_{i=1}^{n}(1 - p_i(f(\mathbf{x})_i|\mathbf{x})) && (2.72) \end{aligned}$$

Here, p_i was used to denote the marginal probability mass function of Y_i. The expression in (2.72) is minimized by the prediction

$$f(\mathbf{x})_i = \underset{y\in Y_n}{\operatorname{argmax}}\ p_i(y|\mathbf{x}), \tag{2.73}$$

i.e. the value of Y_n that maximizes the marginal distribution of Y_n given X. Computing the marginals and the partition function Z of a CRF is known as *probabilistic inference*. The advantage of probabilistic inference is that it yields a more differentiated result than MAP inference. This is because the marginals of each variable y_i provide useful information that can be used in consecutive processing steps. A commonly used method for probabilistic inference in graphs is *belief propagation*, which will be presented in the following section.

Belief Propagation

Belief propagation is a dynamic programming method to perform probabilistic inference in tree-structured graphs. For a cyclic graph the method provides an approximate solution. The approximate version is also called *loopy belief propagation*. Even though it lacks a theoretical justification, the methods performs very well in practice [17]. If the method converges, the approximation is normally very good. Nevertheless, it can happen that messages start circulating in the graph leading to an oscillation between states. The core concept of belief propagation is to pass messages in the graph from factors to variables and vice versa. The message from variable Y_i to factor F_j is denoted by $q_{Y_i \to F_j}$, the message from factor F_j to variable Y_i by $r_{F_j \to Y_i}$. For tree structured graphs, they are defined as follows.

$$q_{Y_i \to F_j}(y_i) = \sum_{\substack{k \in \mathcal{N}(Y_i) \\ k \neq j}} r_{F_k \to Y_i}(y_i) \tag{2.74}$$

$$r_{F_j \to Y_i}(y_i) = \log \Bigg(\sum_{\substack{\mathbf{y}'_F \in Y_{F_j} \\ y'_i = y_i}} \exp \big(- E_{F_j}(y'_{F_j}) + \sum_{k \in \mathcal{N}(F_j) \setminus \{i\}} q_{Y_k \to F_j}(y'_k) \big) \Bigg) \tag{2.75}$$

Here the notation $\mathcal{N}(F_j)$ was used to denote the set of indices of the variables Y_i adjacent to the factor F_j. Those formulas can be derived by computing marginal distribution in a tree-structured graph, as shown in [12]. Since the messages depend on each other, only the messages from factors or variables with only one adjacent variable or factor are known initially. In graphs with loops this is not the case. Instead, messages are initialized to zero or some random value. The message updates are then performed iteratively or at random. In order to conduct inference in graphs with loops those messages have to be modified slightly. The detailed formulas can be found in the appendix B.

3

Implementation

In this chapter, the implementation of the proposed hole detection system is described. Since this work is part of a larger software project, the general structure of the hole detection has to be adapted to the requirements of this project. The embedding of the hole detection is displayed in the flow chart in figure 3.1. The hole detection is required to be performed in two parts. The first part consists of an independent hole detection, that uses only the raw information contained in the images. The second part refines the results from the first step using high-level information from other modules for the detection of shelves and products.

3.1 Software

The hole detection is implemented using the ToolIP [1] graphical framework, developed by the ITWM Fraunhofer institute. ToolIP provides a library of different image processing algorithms in the form of plugins, that can be combined as nodes in a graph. Images are passed along edges in this graph and in that way complex image processing solutions can be constructed in a convenient manner. ToolIP also provides an interface for the development of new plugins in `C++`. For the hole detection 8 new plugins were developed. The parameter learning of the machine learning techniques and other data processing tasks were performed using Python 2.7.6. Furthermore, the following external libraries were used to implement the hole detection system

OpenCV: The OpenCV 2.4.7 library[18] was used to implement the SVM and the AdaBoost classifier.

OpenGM: Probabilistic inference in graphs was implemented using the OpenGM 2.0.2 library[19].

Dlib: Learning the parameters of the CRF image model was performed using the structural SVM implementation of the Dlib library[20]

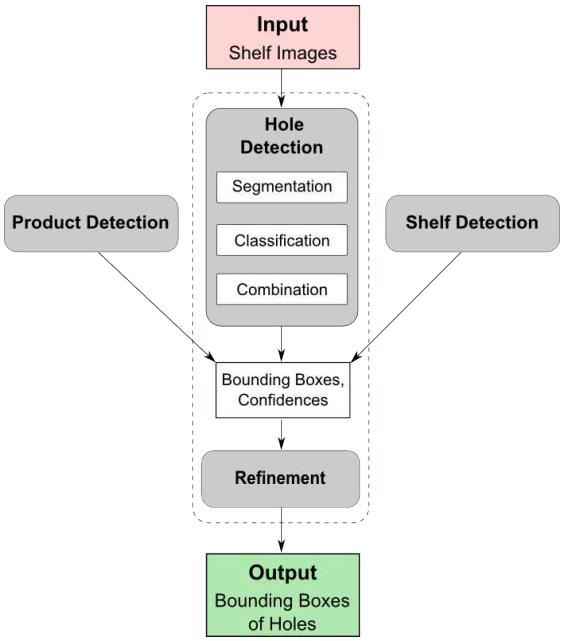


Figure 3.1: The structure of the hole detection system. The first part of the hole detection is encapsulated in an independent hole detection module that uses only the raw information from the shelf images. This preliminary hole detection consists itself of three steps: segmentation of the image into superpixels, classification of the superpixels and combination of the classified superpixels to obtain bounding boxes for potential holes. The predictions are then refined using the information from the shelf and product detection modules.

SLIC: The implementation of the SLIC superpixel algorithm was taken from [7].

3.2 The Proposed Method

In this section the proposed approach to the hole detection is described. The hole detection makes two kind of predictions: An unrefined prediction using only the raw information from the image and a refined prediction, that was obtained using information from the other detection modules. The approach can be divided into the following, consecutive steps:

Segmentation: First, the input image is segmented into superpixels using the SLIC superpixels algorithm presented in section 1.2.1.

Classification: Each superpixel is classified using an SVM classifier as presented in section 2.2.

Combination: In this step suitable bounding boxes for the tentative holes are found and a confidence measure for each bounding box is computed. Then the unrefined

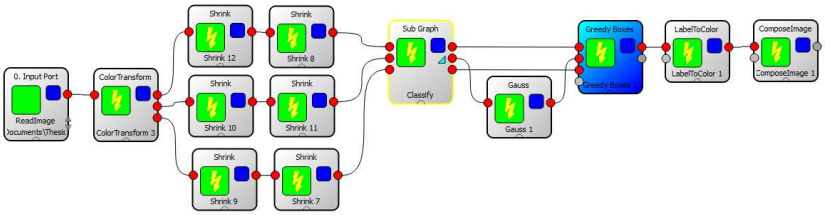


Figure 3.2: Example of a ToolIP graph. A graph consists of plugins and subgraphs. Each node processes the image that is given as input and passes it on to the next nodes. In this way complete image processing solutions can be built from the available algorithms.

prediction is made using the bounding boxes and the confidence values.

Refinement: Using contextual information from the product and shelf detection, a refined prediction for the holes in the image are made.

Apart from the SVM classifier, two other classification methods are implemented in order to determine which one performs best. The two other classifiers are an AdaBoost classifier and a CRF classifier, that uses the combined results from the SVM and the AdaBoost classifier. The detailed structure of the proposed hole detection is displayed in figure 3.3.

3.3 Image Segmentation

To segment the images, the SLIC algorithm described in chapter 1.2.1 is used. The algorithm outputs an image containing labeled regions with labels $1, \ldots, N_{\text{sp}}$, where N_{sp} is the number of superpixels in the image. There are two parameters that influence the results of the superpixel segmentation: The number N_{sp} of superpixels and the compactness k. The compactness value used here is $k = 10$, as recommended in [7]. The choice of the parameter N_{sp} is a tradeoff between the information content of a single superpixel and the ability to adapt to the structure of the image. The situation is further complicated, because the shelf images come in different scales and may contain products of varying sizes. The number of superpixels also has an impact on the computation time because it determines the number of classifications that have to be performed. Those may be relatively costly depending on the chosen classification method. The value used for almost all computations in this work is $N_{\text{sp}} = 2000$. The influence of N_{sp} on the image segmentation is displayed in figure 3.4. As can be seen from the figure, a value

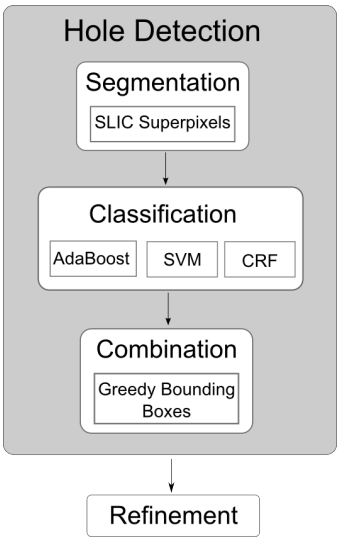


Figure 3.3: The proposed hole detection method. The detection is performed in four steps. First, the image is segmented into superpixels using the SLIC superpixels algorithm. The superpixels are then classified an SVM classifier. The classification results for the superpixels are then combined to find bounding boxes of tentative holes. Finally, the predictions are refined using information from the other detection modules. Apart from the SVM classifier, two other classification methods, an AdaBoost and a CRF classifier, are implemented and tested.

(a) Input Image **(b)** Segmented Image

Figure 3.4: **(a)** Input image. An example image containing a hole and several products. **(b)** Superpixel segmentation of **(a)**. Displayed are the borders of the superpixels for different numbers of superpixels in the hole image. Red corresponds to $N_{\text{sp}} = 500$, green to $N_{\text{sp}} = 1000$, purple to $N_{\text{sp}} = 2000$ and yellow to $N_{\text{sp}} = 4000$.

value of N_{sp} that is too low fails to capture the structure of the image which may lead to problems in the classification step.

3.4 Classification

For the classification step three different methods are presented in this section. The classification may be viewed as the main part of the hole detection, because it determines which image regions are considered as potential holes. Therefore, three different classifiers are implemented and tested in order to find the method that performs best. The first two methods, the AdaBoost and the SVM classifier, can be considered as local, because they only use features of one superpixel and its neighbors and don't consider the whole picture. The CRF classifier on the other hand takes into account the whole structure of the image. Since training and inference in a CRF model is very demanding, the CRF classifier uses the results from the SVM and AdaBoost classifier as input features. It may thus also be viewed as a two-level classifier, building on the results of the AdaBoost and CRF classifier. In this section the details of the implementation and training of the classification methods are described.

3.4.1 Features

The local classifiers use a total of 219 features, that are extracted from each superpixel. Apart from relatively common features, such as color and texture descriptors, some of the features were designed specifically for the application context.

HSV Color Space: For each superpixel the mean, variance and maximum and minimum value of each channel h, s and v are computed giving $3 \times 4 = 12$ features.

CIE LAB Color Space: Similarly, the mean, variance, maximum and minimum values of each channel of the CIE LAB space are computed for each superpixel. This yields another 12 features.

Chroma: For each pixel the chroma c is computed from the means of the a and b channels of the CIE LAB color space. The features extracted from it are mean, variance, maximum and minimum, yielding four more features.

Merged Superpixels: Over the whole image superpixels are merged depending on the distance of their means in the CIE LAB color space. The size, maximum width and maximum height are used as features. The motivation is that holes are usually large homogeneous regions that yield large values, whereas products are inhomogeneous and yield smaller values. A figure illustrating the merging of superpixels and the corresponding features is included in the appendix in figure A.1.

Further Variance Measures: The homogeneity of holes can be exploited using variance measures. However, the expressiveness of such measures on superpixel-level is deteriorated by the segmentation, which aims to give homogeneous regions. To overcome this, the variance in a bounding box of the superpixel rather than only over the superpixel is computed. This is done for all three channels of the CIE LAB space.

Compactness Measures: The compactness of the superpixels depends on the structure of the corresponding image region. Products often exhibit salient structures, which result in superpixels that are not very compact. Holes on the other hand are relatively homogeneous, which leads to compact superpixels. The compactness measures used are given by

$$\text{comp}_1 = \frac{A_{\text{sp}}}{A_{\text{bb}}} \tag{3.1}$$

$$\text{comp}_2 = \frac{4\pi A_{\text{sp}}}{S_{\text{sp}}^2} \tag{3.2}$$

where A_{sp} is the area of the superpixel, A_{bb} the area of its bounding box and S_{sp} its border length. A figure illustrating the compactness features can be found in the appendix, figure A.2.

Local Binary Patterns: Two different histograms for the local binary patterns are computed. The basic LBP histogram on three color channels would yield 3×256 features. Since very high numbers of features may affect the performance of the classifier and also make prediction more costly, compressed versions of the basic and uniform LBP histograms are used. The standard LBP histogram with 256 bins is reduced to 8 bins. This is done by grouping the histogram into groups of 32 adjacent bins and merging them. For the uniform histogram, bins corresponding to cyclic permutations of a given uniform pattern are merged together. As presented in 1.3, this yields 9 bins plus one for the non-uniform patters. Thus, the resulting histogram contains 10 bins. Computing the histograms for the three channels of the CIE LAB space yields $3 \times (8 + 10) = 72$ features.

Color Histograms: To describe the color characteristics of the superpixels color histograms are used. A common approach taken here is to consider chromaticity information and brightness information independently. To describe the brightness of a superpixel a histogram with 10 bins of the l channel of the CIE LAB space is used. The histogram is centered around the mean, because the brightness range in a single superpixel is usually narrow. The chromaticity information is summarized using a 10×10 histogram over the a and b channels. Again, the histograms are centered around the mean of the superpixel. This yields another 110 features.

Agents An agent based structure descriptor, as described in section 1.4, is also used as a feature for the superpixels. The algorithm is run on the dilated internal gradient image and the number of agents in each superpixel is used as a feature. This is done for the three channels l,a,b of the CIE LAB space.

Redrik For each row in a superpixel a maximum and a minimum to the left and another minimum to the right of the maximum are computed. The differences are then averaged over all rows in the superpixel. The resulting average values, one for the minima on the left and one for the minima on the right, are then used as features

for the superpixel. They are computed for all three channels of the CIE LAB space. An illustration of the Redrik features can be found in figure **??** in the appendix.

Neighboring Superpixels: For the four neighboring superpixels that have the longest common border with the current superpixel, the means of the l, a and b channel of the CIE LAB space are used as features. If a superpixels has less than four neighbors, the missing values are replaced by the means of the superpixel itself. The exact neighborhood relation used here is described in section 3.5. This gives another 12 features.

3.4.2 Training Data

The AdaBoost and the SVM classifier are both trained on the same training set consisting of 25 images. The training images were annotated using the MAOI software tool, that is included in the ToolIP package. One of the requirements of the project is to avoid to falsely classify a product as a hole. On the other hand, false positives in other image regions are tolerable. The classifiers were thus primarily trained to distinguish between products and holes. This is achieved by explicitly labeling image regions corresponding to holes or products and using the superpixels in those regions to train the classifiers. Another problem with the training images is that the superpixels corresponding to products usually outnumber the superpixels corresponding to holes. In order to avoid complications due to the imbalance of the training set, the product instances in the training set are randomly sampled without replacement from the total set of superpixels classified as products. In order to evaluate the performance of different versions of the classifiers and find suitable values for the meta-parameters a validation set consisting of another 25 images was used. The validation data was preprocessed in the same way as the training data.

3.4.3 The AdaBoost Classifier

In order to explore the capabilities of the AdaBoost classifier, different variants of it with different weak classifiers were trained on the training set. To evaluate their performance, the error on the validation set is computed. The results are displayed in figure 3.5. The tested variants of the AdaBoost classifier are Real AdaBoost and Gentle AdaBoost. For each variant weak classifiers with different complexities are tested. The weak classifiers used are stumps, i.e. decision trees of depth 1, decision trees of maximum depth 5 and decision trees of depth 10. For stumps the performance of Real AdaBoost and Gentle AdaBoost on the validation set is equal. For deeper trees the Gentle AdaBoost method outperformed Real AdaBoost. The best performance on the validation set is obtained by the Gentle AdaBoost method using decision trees with a maximum depth of 5. Nevertheless, the advantage over Gentle AdaBoost with a decision tree depth of 10 is only marginal. Since the depth of the weak classifiers used affects the time needed to evaluate the classifier, the Gentle AdaBoost method with decision trees of maximum depth 5 is chosen. Since the gain in performance for more than 150 weak classifiers

is minimal, a number of 150 weak classifiers is chosen for the AdaBoost classifier. As

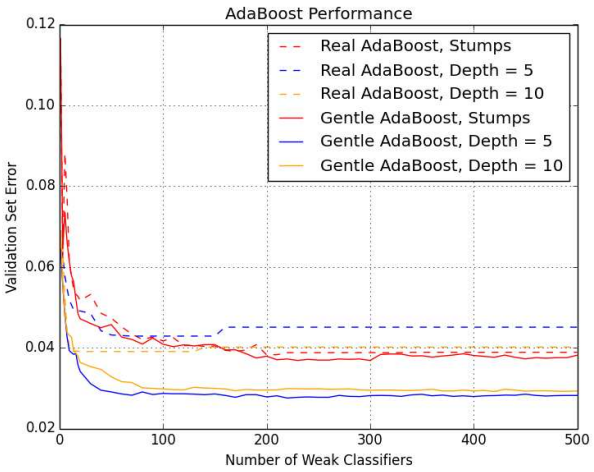


Figure 3.5: Performance of the AdaBoost classifier. The graph displays the error on the validation set with respect to the number of weak classifiers used. For each variant different depths of the weak classifiers are tested. It can be seen that, except for the classifiers using stumps, Gentle AdaBoost outperforms Real AdaBoost. The best performance yields the Gentle AdaBoost method with a maximum depth of 5 or 10.

described in section 2.3.6, the AdaBoost method can be used to compute a relative importance measure for the features used. For this purpose, a classifier is trained using the Gentle AdaBoost method and the relative importances were computed. The 50 highest rated features are displayed in figure A.4 in the appendix. The most important feature is the agent feature on the l-channel, followed by the Redrik Feature for the l channel and the size of the merged superpixels. All of these features are in some way connected to the homogeneity of the image, which goes along with the intuition that holes are characterized by relative homogeneity compared to products. Since the number of features has a strong impact on the computational cost of the classification and sometimes also its performance, it is interesting to investigate the classification performance with respect to the number of features. Figure 3.6 displays the performance of the Gentle AdaBoost classifier based on decision trees with a maximum depth of 5 for different numbers of features. The performance is measured for all as well as the 100, 50 and 25 features with the highest importance. The graph shows that up to a number of 250 weak classifiers the performance is almost equal for 50, 100 and all features. For 25 features the performance is slightly worse. The performance of the classifier using the 50 best features even improves slightly, when the number of weak classifiers is increased further.

Based on the performance results and computational considerations, the Gentle AdaBoost classifier with 150 decision trees of maximum depth 5 trained on the 50 features with the highest relative importances is chosen for the implementation. In figure 3.7 the classification results of the AdaBoost classifier on a sample image are

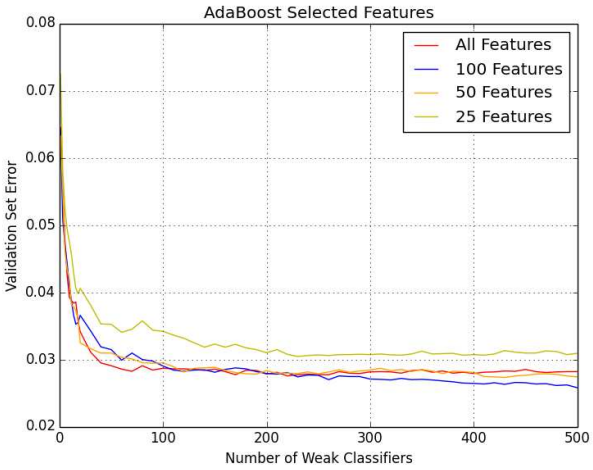


Figure 3.6: AdaBoost performance for selected features. A classifier was trained on different numbers of samples using the Gentle AdaBoost method with a maximum-depth of 5 and 200 weak classifiers. The graph displays the error on the validation set for different numbers of features. The performance for all features and the 100 and 50 features with the highest importance is essentially equal. The performance of the classifier using the 50 best features even improves slightly when the number of weak classifiers is increased over 200. Using only the 25 features with the highest importances results in a slight performance loss.

(a) Input Image **(b)** Classification Results

Figure 3.7: Results of the AdaBoost classifier. The results of the classification are the weighted votes of the AdaBoost classifier, which can be interpreted as confidences. White regions are where the weighted votes of the weak classifier are positive indicating a hole. In the dark regions the weighted votes are negative, which indicates products.

displayed. The values displayed are the weighted votes of the weak classifiers output by the AdaBoost classifier. Those values can, to some extent, be interpreted as a confidence value. That means large, positive values (white) indicate that the classifier is sure about the corresponding region being a hole, whereas values that are much smaller than zero (black) indicate that the classifier is sure about the region not being a hole.

3.4.4 The SVM Classifier

The SVM classifier used here is based on radial basis functions (RBFs). SVMs based on RBF are commonly used in image analysis because of their classification performance and computational efficiency. Since SVM classifiers are sensitive to differently scaled features, all features are normalized to have mean 0 and standard deviation 1. The RBF SVM classifier has two meta-parameters. The parameter C appears as a factor of the sums of the slack-variables in the formulation of the SVM, see (2.12). It determines how big the influence of misclassified training samples is. A too large C leads to overfitting of the classifier and makes it vulnerable to noise in the training data. The parameter γ determines the smoothness of the decision boundary in the feature space. Again, it is a trade-off between the ability of the classifier to adapt to the data and overfitting on the other hand. In order to determine suitable values for the parameters a log-scale grid search was performed. This was done for different subsets of the features presented in the previous section. Again, the different subsets used were all features as well as the 100, 50 and 25 features with the highest relative importance values. The result are displayed in figure 3.8. In the plots, the evaluation points are marked by gray dots. The coloring was obtained using linear interpolation. The general structure of the results is the same for all sets of features that were used. Also the optimal values obtained and their position were essentially equal. The best performance on the training set was obtained for the feature set containing the 50 highest rated features at values $\gamma = 0.0251$ and $C = 1$. This is the classifier that was chosen for the implementation. The classification results of the SVM classifier are the distances of each input sample to the decision boundary. Similar to the AdaBoost classifier, large values greater than zero indicate holes and negative values indicate non-holes. The results of the classifier on the sample image are displayed in figure 3.9

3.5 The CRF Classifier

The CRF classifier models structural dependencies in the image using pairwise interactions between neighboring superpixels. In order to model an image as a graph, it is necessary to define a neighborhood relation on superpixels. Here, a generalized version of the four connected neighborhood is used, in the sense that for every superpixel the neighbors in a given direction are the superpixels that have the most border pixels in this direction. However, since a neighborhood relation is symmetric this may result in a superpixel with more than four neighbors. Formally, the neighborhood relation is defined as follows. For each pair i,j of superpixels, for each direction north, east, south, west, it is counted how many pixels of i have a neighbor in j in the given direction. The north, east, south and west neighbors of i are then the superpixels with the most neighbors in the respective direction, that are not already a neighbor in another direction. In order to obtain a symmetric relation, all superpixels j that have i as a north, east, south or west neighbor are also defined to be neighbors of i. The relation between neighboring superpixels is modeled by an edge in the corresponding factor graph. Using the above

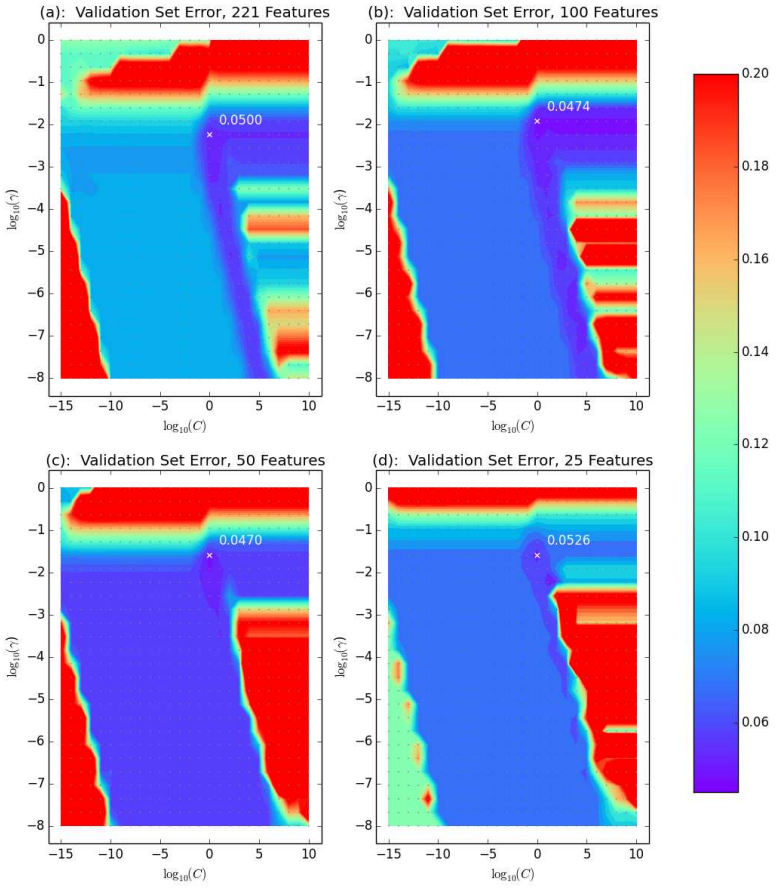


Figure 3.8: Optimizing the SVM classifier. In order to find suitable meta-parameters C and γ of the RBF SVM classifier a grid search using the error on the validation set was performed for the different sets of features. The coloring was obtained using linear interpolation. Evaluation points are marked using Gray dots. The features used were all features as well as the 100, 50 and 25 best features as determined in the previous section. The optimal values for each set of features are essentially equal. Moreover, the general structure of the results looks similar.

(a) Input Image **(b)** Classification Results

Figure 3.9: Results of the SVM classifier. The results of the classification are the distances to the decision boundary, which can be interpreted as confidences. Large, positive results (white) indicate a confident hole, whereas large, negative values (black) indicate confidence that the corresponding superpixel is not a hole.

definition of the neighborhood relation, it is possible to distinguish horizontal and vertical edges. The resulting graph is displayed in figure 3.5. Vertical edges in the graph are displayed in blue and horizontal edges in red.

3.5.1 Training

The CRF model presented in section 2.4 has 20 parameters specifying the unary and pairwise potentials. Since the CRF classifier is based on the SVM and AdaBoost classifier, a different training set is used. The data used to trained the classifier consists of 15 images from the validation set. The remaining 10 images in the validation set are used to determine the parameter γ_C of the contrast function and the parameter C of the structural SVM. Since the training of the CRF model is computationally demanding, an extensive grid search as in the SVM case was not possible. 30 different combinations of the parameters were tested and their errors on the validation set computed. The best values are obtained for $C = 1$ and $\gamma_C = 0.45$. The corresponding unary energies of node i in the graph are given by

$$E_i(y_i,\mathbf{x}_i) = \begin{cases} -1.2x_{i,1} - 0.4x_{i,2} & , y_i = 0 \\ 1.2x_{i,1} + 0.4x_{i,2} & , y_i = 1 \end{cases}. \tag{3.3}$$

Here, $x_{i,1}$,$x_{i,2}$ are the classification results of the corresponding superpixel from the SVM classifier and the AdaBoost classifier, respectively. An interesting result here is, that the results from the SVM classifier are weighted heavier than the results from the AdaBoost classifier. The values of parameters of the pairwise potentials $E_{i,j}$ are given as 2×2 matrices in figure 3.11. The rows correspond to the classes of Y_i and the columns to the class of Y_j. They can be interpreted as follows. Negative values on the diagonal as well as positive values on the off-diagonal favor neighboring superpixels to have the same class and thus have a smoothing effect on the image. The model has thus learned

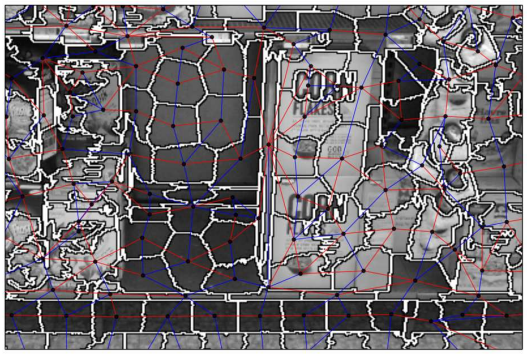


Figure 3.10: The CRF model. The CRF model uses a graph to model the interactions between image regions. The graph corresponding to an example image is displayed in the figure. Vertical edges are displayed in blue and horizontal edges in red.

that the results can be smoothed in vertical direction rather than in horizontal direction. However, for the non-hole class ($Y_i = 1$) the soothing is restricted to similar superpixels.

3.5.2 Inference

In order to classify the superpixels in an image the marginal distributions of each node are computed. The used method is loopy belief propagation. The method yields two values for each superpixel, namely the energy corresponding to the marginal probability of the superpixel being a hole and for the superpixel being a non-hole. In order to obtain the corresponding probabilities, it would be necessary to approximate the partition function $Z(\mathbf{x})$. The approach chosen here is to compute the negative difference between the two energies which amounts to computing a logarithmic likelihood ratio. Hence, the class of a superpixel is given by the sign of the log-likelihood ratio. The results of the CRF classifier on the sample image are displayed in figure 3.12.

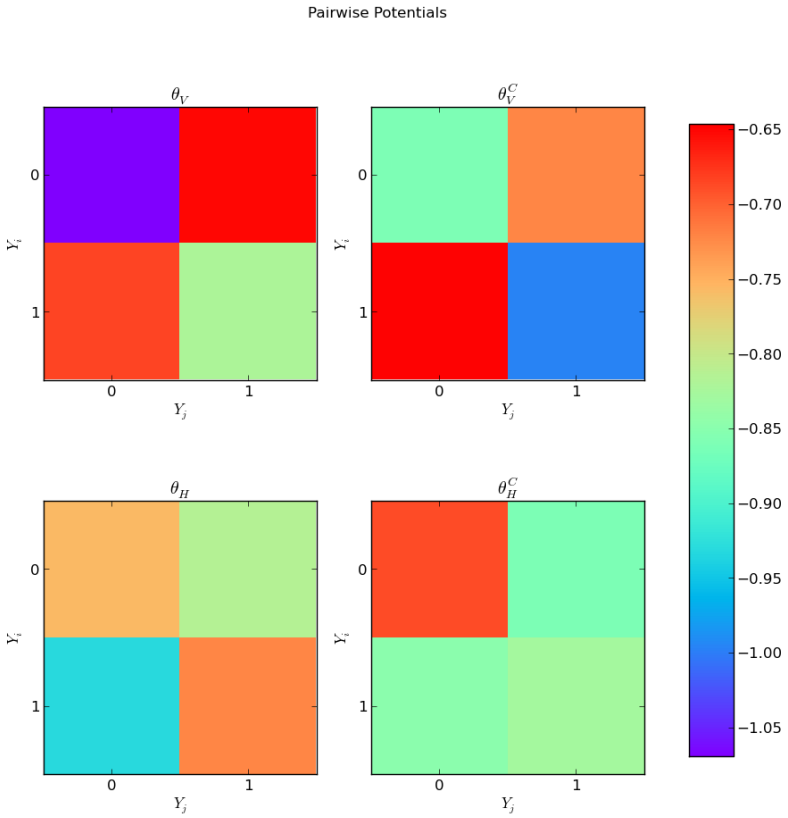


Figure 3.11: Pairwise Potentials. The plots display the pairwise potentials that were learned from the training data. The plots in the top row display the parameters for vertical edges. Here, negative values on the diagonal indicate that the result is smoothed in this direction. The plots in the bottom row display the parameters for horizontal edges. Here, no strong smoothing effect occurs.

3.6 Combination

Since the results of the hole detection are required to be bounding boxes together with confidence values, it is necessary to find suitable bounding boxes for the image regions that were classified as holes. Moreover, this step is also used to refine the results from the previous steps and eliminate unlikely candidates. For the combination step it is assumed that the classification results are in or close to the range $[-1,1]$ and in some way represent confidences whether a given superpixel is a hole (positive) or not (negative). In this way it is possible to recover from small classification errors and obtain a more precise detection by allowing small negative region in the detected holes.

(a) Input Image **(b)** Classification Results

Figure 3.12: Results of the CRF classifier. The displayed values correspond to the logarithm of the likelihood ratio of a superpixel being a hole. Thus, positive regions (white) imply confidence about the superpixel being a hole and negative regions (black) confidence about the superpixel not being a hole.

3.6.1 Greedy Bounding Boxes

At the heart of the combination step is a greedy algorithm for finding bounding boxes for potential holes. The algorithm starts out with the superpixel that obtained the highest result in the classification and grows a bounding box around this superpixel. All superpixels that are contained within this box are then removed from the seeds. This is repeated until no seeds are left. A pseudo-code formulation of the algorithm follows.

Algorithm 3: Greedy Bounding Boxes

Input: Image I containing classification results, corresponding superpixels $s_1, \ldots, s_n$

Initialize: Seed list $l_S = (s_1, \ldots, s_n)$, bounding box list $l_B = \varnothing$

- Compute classification results $c_{s_1}, \ldots, c_{s_n}$ of each superpixel $s_1, \ldots, s_n$.
- Remove superpixels s_i with $c_{s_i} < 0$ from l_S
- Sort l_S with respect to the classification results in descending order

For $s_i \in l_S$ **do**

- Compute bounding box b_i of s_i
- Grow bounding box b_i
- Add bounding box b_i to l_b
- Remove all superpixels s_i from the seed list l_S that are contained in b_i

Output: The list of bounding boxes l_b

An important part of this algorithm is the growing of the boxes. The idea here is to grow the boxes in vertical direction primarily, since holes per definition have to span one full row of the shelf. Beginning with the north and south direction, the boxes are grown pixel-wise while the sum over the corresponding edge in the classification results is positive and the box does not collide with any other box. In the west and east direction, the image is only grown one step, in order to prioritize vertical growth. This is repeated until the box could not be grown further in any direction.

Algorithm 4: Grow Bounding Box

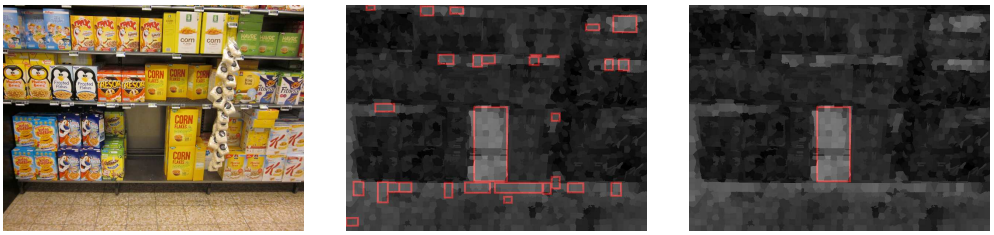
Input: Image I containing classification results, bounding box $b = (x_1,y_1,x_2,y_2)$
while *b has grown* **do**
  while *no collision* **and** $sum_N(I) > 0$ **do**
    grow box to the north
  end
  while *no collision* **and** $sum_S(I) > 0$ **do**
    grow box to the south
  end
  if *no collision* **and** $sum_W(I) > 0$ **then**
    grow box to the west
  end
  if *no collision* **and** $sum_E(I) > 0$ **then**
    grow box to the east
  end

Output: The list of bounding boxes l_b

Here, $\text{sum}_N(I)$ is the sum of the pixel-wise classification results over the northern edge of the box. The sums over the other edges are denoted accordingly. Using the sum over one of the edges as growth condition may turn out to be too gentle and lead to too large boxes. To prevent this, a non-linear scaling is applied to negative values that puts increasing weight on large, negative values. In a final step, bounding boxes that are directly above each other are merged. This is done to ensure that holes are detected in their full height, which is important for the elimination of false positives.

3.6.2 Confidences

Apart from the bounding box, a predicted hole is required to have an associated confidence value between 0 and 1 that reflects how certain the predictions is. The approach taken here is as follows. For each bounding box, the average of the classification results over a larger box containing this box is computed. The larger box is required to have the same width as the bounding box and to be at least as high as a given fraction of the highest bounding box in the image. The height here is chosen to be $0.9 \cdot h_{\max}$, where $h_{\max}$ is the height of the highest bounding box. This larger box is found by choosing the box of the required height that contains the original box and yields the highest average over the classification results. The approach is based on the assumption that holes in an image should have roughly the same height. Forcing the boxes to have a given minimum height avoids being overly confident about small boxes. In order to obtain values between zero and one, the validation set was used to fit a scaled and shifted version of the tanh



(a) Input Image **(b)** Detected Bounding Boxes **(c)** Thresholded Bounding Boxes

Figure 3.13: Discarding unlikely bounding boxes. Growing bounding boxes around every superpixel that was classified as a hole leads to many false positives (b). Requiring the boxes to have a certain minimum height and width relative to the height of the highest box in the image improves the results (c).

function to the averages. The curve was parametrized to have form

$$g(x) = 0.5(1 + \tanh(bx + a)). \tag{3.4}$$

(3.5)

The obtained parameters are given by

$$a = 0.76 \tag{3.6}$$

$$b = 2.86 \tag{3.7}$$

3.6.3 Discarding Holes

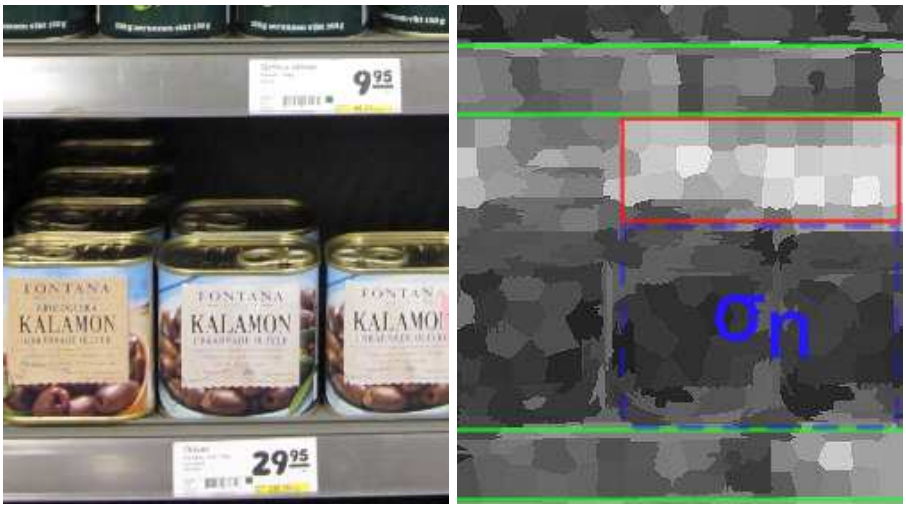
The results from the previous steps yield bounding boxes around every superpixel that was classified to be a hole. Those results still contain too many false positives, that might resemble a hole on a local scale but do not satisfy the exact definition of it. This is for example the case when a product does not reach up to the next shelf edge. It is possible to eliminate some of those false positives by discarding all holes with a height less than a certain fraction of the height of the highest bounding box found in the image. In the following h_{rel} will be used to denoted the relative height of a detected hole compared to the height of the highest hole. Also holes that are very narrow are unlikely to be actual holes, because they have to be at least as wide as adjacent products. The detected holes can thus also be thresholded on the their width relative to the height of the highest hole in the image. The effect of this is displayed in figure 3.13 for a classification result from the SVM classifier.

3.7 Refinement

In this step the predicted holes are further refined using the high-level information provided from the other detection modules. Since a hole is required to go from the lower shelf edge to the upper shelf edge and to be at least as wide as the adjacent

products, the information from the other detection modules is very valuable to further eliminate false positives. Here, the focus will lie on using information from the shelf detection to eliminate false positives.

For the refinement it is assumed that the shelves are given by a set of bounding boxes corresponding to the shelf edges. First of all, detected holes whose centers do not lie over a shelf are discarded. This is done to get rid of detections in regions where there is no shelf present and detections that are beneath the shelf. Then, for each detected hole the number of pixels with a negative classification result in the rectangle between the bounding box of the detected hole and the upper and lower shelf edge denoted by is computed. This value is then divided by the width of the tentative hole and the distance between the two shelve edges. This value is denoted by σ_n. The purpose of this is to obtain a measure of how probable it is that there is a product between the predicted hole and the shelf edges, which would mean that the detection is false. This value can then be used as a threshold to eliminate false positives in the results. The principle is illustrated in figure 3.14. The figure displays a region of a shelf image, where a false hole was detected (red box). The value σ_n is computed over the region indicated by the blue rectangle.



(a) Input Image **(b)** Classification Results

Figure 3.14: Refinement of predicted holes. The predictions for holes are refined by thresholding on the value σ_n, which is the sum of negatively classified pixels in the rectangles between the detected hole (red) and the upper and lower shelf edge (green) divided by its width and the minimum distance between two shelves in the image.

4

Results

In this chapter the hole detection system is evaluated with respect to its performance. The proposed solution based on the SVM classifier is compared to the AdaBoost and the CRF classifier. This is followed by a detailed analysis of the performance characteristics of the hole detection based on the SVM classifier.

4.1 Classifier Performance

The detection performance of the hole detection is measured by applying it to a set of test images and comparing the results to a ground truth. A true hole is counted as detected if a predicted hole box covers at least 25% of the true hole. Moreover, a predicted hole is counted as correct prediction if at least 50% of it covers a true hole. Based on this, *precision* and *recall* of the hole detection are computed. The precision of a detection method is the ratio of correct predictions and the total number of predictions made. The recall of a detection method is the ratio of true holes that were detected and the total number of true holes contained in the test images. Moreover, the computational performance of the method is evaluated by computing the average runtime on the test set.

4.1.1 Unrefined Predictions

In figure 4.1, the detection performance of the unrefined hole detection using the SVM classifier and the two other methods is displayed. As described in section 3.6.3, detected holes with a relative height with respect to the highest box in the image h_{rel} are discarded if h_{rel} is below a certain threshold. The values of recall and precision for different values of this threshold are given by the recall-precision curves in figure 4.1. Moreover, holes that were narrower than 0.1 of the height of the highest hole in the image are discarded. The SVM classifier performs better than the other two methods for all but one value. The AdaBoost method performs worse than the SVM method, but approaches the curve

of the SVM classifier for high recall values. The CRF classifier performs almost as good as the AdaBoost classifier. The values of h_{rel} used as threshold corresponding to the recall-precision curves are given in table C.1 in the appendix. All methods can reach a relatively high recall values, but at insufficient precision rates. As can be seen for example from the sample results in the appendix, the high number of false positives come from regions where the shelf is visible above products for example or regions outside of the shelf. At this level, however, it is very hard to discard those because only very little is known about the structure of the image.

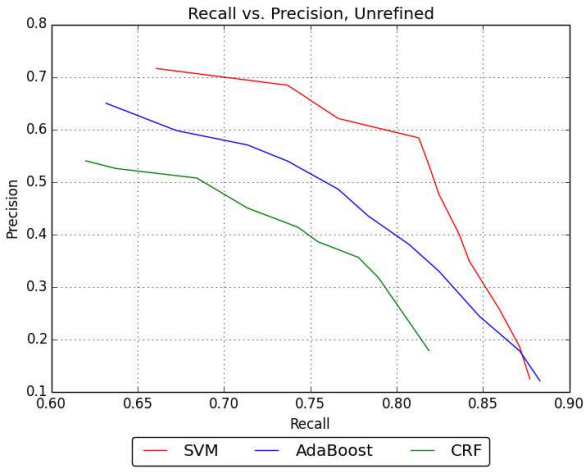


Figure 4.1: Detection Performance of the unrefined hole detection. The hole detection based on the SVM classifier yields the best performance on the test set. The performance of the AdaBoost and the CRF classifier is very much alike. A problem inherent to all classifiers are the high false positive rates.

4.1.2 Refined Predictions

In figure 4.2, the detection performance for the hole detection using refinement is displayed. Since the refinement is only used to eliminate detections, no bounding boxes are discarded with respect to their relative height compared to the highest detected box in the image h_{rel}. Instead, the normalized sum σ_n of the negative classification results in the rectangle between detected hole and upper and lower shelf is computed (see section 3.7). This value is then used as a threshold to discard detections. Moreover, bounding boxes are discarded that are smaller than one third of the minimum distance between two shelves in the image. The resulting recall-precision curve is displayed in figure 4.2. The refinement significantly improves the performance of all three methods. Through the refinement much higher precision can be obtained at similar recall levels. From the three classifier, it is still the SVM that performs best. The values of σ_n corresponding to the recall-precision curves in figure 4.2 are given in table C.2 in the appendix.

Figure 4.2: Performance of the hole detection using refinement. Using refinement, the performance of the hole detection can be increased drastically. The SVM classifier still yields the best performance. Moreover, the CRF classifier now performs nearly equally well as the AdaBoost classifier.

4.1.3 Computational Performance

The computational performance of the system for the three classification methods is compared by computing the average runtime on the test set. The size of the test images is 812×608 pixels and the computations were performed on an intel core i-5 2500 processor with 8 gb memory. Since the contribution of the refinement to the overall runtime is small and the same for all classification methods, this is only done for the unrefined hole detection. The results are displayed in figure 4.3. The SVM classifier clearly outperforms the other two methods with respect to the runtime. Its average runtime is 2.16 s, which is more than half a second faster than the AdaBoost method and one second faster than the CRF classifier.

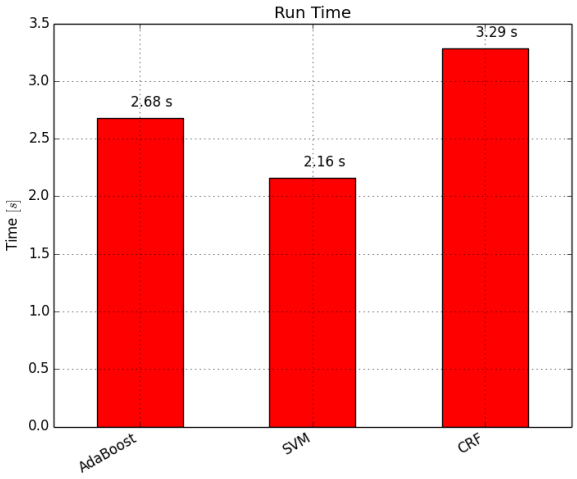


Figure 4.3: Average runtime of the hole detection on the test set. The SVM classifier clearly outperforms the other two methods with respect to the runtime. It is half a second faster than the AdaBoost method and more than one second faster than the CRF classifier.

4.1.4 Sample Images

The results of the hole detection for the three different methods applied to sample images are displayed in figures A.5, A.6, A.7 in appendix A. For each method the input image, the classification results, the unrefined results and the results using refinement are displayed. On the sample image the results obtained from the SVM classifier and the AdaBoost classifier are very much alike. The images also illustrate the problem of

false positives. Regions without products are likely to be falsely detected as holes. Also detections outside of shelves occur. The classification results of the CRF classifier show the smoothing effect the CRF classifier has on the results. In some cases this helps to reduce false positives but also leads to the loss of detail in other regions and spreading of holes over its actual boundaries.

4.2 Detailed Analysis

In this section a detailed analysis of the performance of the hole detection system based on the SVM classifier is presented. The influence of the number of superpixels used to segment the image is examined. Moreover, the computational performance of the different parts of the system are analyzed. Finally, conditions under which the system performs badly are discussed.

4.2.1 The Number of Superpixels

As mentioned in section 1.2.1, the number of superpixels N_{sp} used to segment an image is a tradeoff between the information content in one superpixel and the conservation of object borders in the image. Moreover, the value of N_{sp} also influences the computational performance of the system. In this concrete case, if a too small number of superpixels is chosen, small holes may be missed because the superpixel either contains a part of the product or the variance in its vicinity is large. In order to determine a suitable number N_{sp} of superpixels, the detection performance of the hole detection is evaluated for different values of N_{sp}. The resulting recall-precision curves are displayed in figure 4.4. The system performs best using $N_{\text{sp}} = 2000$ for most parts of the curve, even though for large recall values $N_{\text{sp}} = 1000$ yields slightly better performance. However, due to the very low precision, this part of the curve is not really relevant and the chosen number of superpixels is thus $N_{\text{sp}} = 2000$.

4.2.2 Confidences

Apart from bounding boxes, for each detected hole a confidence is computed, which is supposed to represent a measure of how certain the hole detection is about a given prediction. The method used to compute the confidences is presented in 3.6.2. In order to validate the significance of the confidence values, their distribution for true and false predictions on the test set is computed. The results are displayed in figure 4.5. The plot displays an overlay of the histograms of the confidence values of the true positive and false positive predictions on the test set. The predictions used are the results from the unrefined hole detection using a threshold $h_{\text{rel}} = 0.7$. and the refined hole detection using $\sigma_n = 0.25$ as threshold. For the unrefined predictions one can see that, even though there is overlap between the two distributions, the values still display an obvious correlation between low confidence values and a detection being a false positive. For the refined predictions, the distribution of the true positives stays nearly the same whereas many

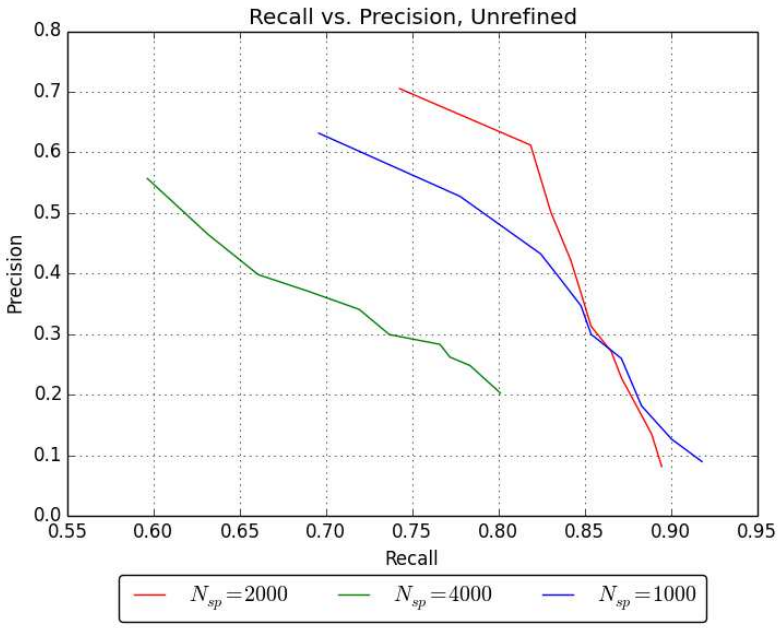


Figure 4.4: Detection performance for different numbers of superpixels. Three SVM classifiers were trained and tested using different numbers of superpixels and the classification performance of the unrefined hole detection over the test set was measured. Using $N_{\text{sp}} = 2000$ yields a clear advantage over the two other values tested, $N_{\text{sp}} = 4000$ and $N_{\text{sp}} = 1000$.

of the false positives disappear. However, the remaining false positives do not show any clear correlation to low confidence values.

4.2.3 Computational Performance

The average runtime of the hole detection on the test set is given in figure 4.3. The proposed method has an average runtime of $2.16s$ on the test set. Since this is quite much, a more detailed analysis of the computational performance is given here. In table 4.1 the runtime for the different steps of the method, segmentation, classification, combination and refinement are given. The time measurement was performed using the timing function in ToolIP on a sample image.

As can be seen from the table, the classification step takes the most time in the process. It should however be noted, that this step also includes the feature extraction, which takes about 1506 ms. The current implementation of the feature extraction extracts all 219 features and not only the 50 used features. This is due to the implementation in ToolIP in which algorithms can only be used in their entity. Since each of the feature extraction algorithms runs separately, their overheads sum up leading to a high runtime. Compared to the overall runtime the contribution of the combination

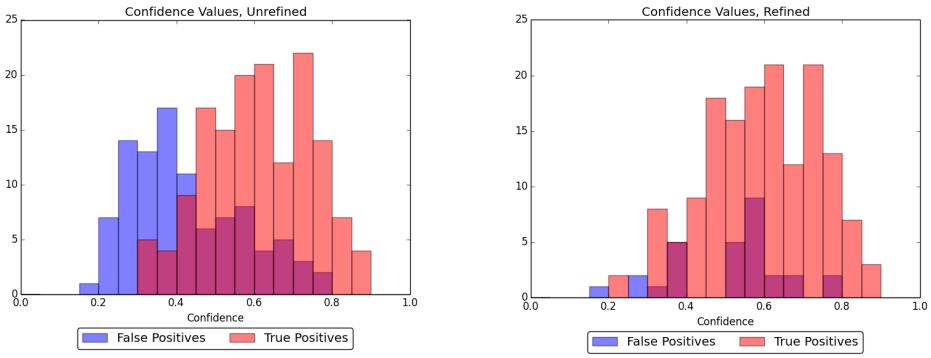


Figure 4.5: Distribution of the confidence values for the true and false positive predictions over the test set. The plots display an overlay of the two histograms of the confidence values of true and false positive predictions. **(a)** displays the values for the unrefined prediction using $h_{\text{rel}} = 0.7$ as threshold on the relative height of the holes. The results clearly show a correlation between false positives and low confidence values. **(b)** displays the results for the refined predictions using $\sigma_n = 0.25$. The distribution shows no clear correlation between the confidence values and false positive predictions.

Step	Time $[ms]$
Segmentation	322
Classification	1607
Combination	21
Refinement	30
Average Total	2160 ms

Table 4.1: Approximate runtime of the different steps of the hole detection

and refinement steps is very small.

4.2.4 Problems

Under certain conditions the detection performance of the hole detection system may deteriorate significantly. The classification of superpixels is based on local color and variance properties. One the one hand, this means that if an image contains products that resemble a shelf in its appearance, i.e. without salient structures or strong colors, those products are likely to be falsely classified as a hole. On the other hand, this also means that shelves with distinct structures or strong colors may not be detected. The hole detection also struggles to detect holes in the top rows of shelves. This is due to the different illumination conditions compared to other holes as well as the fact that in the background of those holes often roof or other products appear. Another problem, that is

specific to the unrefined hole detection, is that in images that do not contain true holes many false positives are detected. This is because the unrefined hole detection uses the height of the highest hole in the image to estimate the distance between two shelves. This approach obviously fails if there is no true hole in the image. In figure 4.6, the hole detection is applied to images displaying such problematic conditions.

Input Image | Unrefined Predictions | Refined Predictions

(a)

Input Image | Unrefined Predictions | Refined Predictions

(b)

Figure 4.6: Difficult conditions for the hole detection. In **(a)**, the unrefined version yields false positive detections where the products to not reach up to the upper shelf edge. Moreover, the hole is not completely detected due to the structures in the back of the shelves. In **(b)** the hole in the top row is not detected because of the salient structures in the background.

5

Discussion and Conclusions

In this work an imaging system for the detection of holes in images of supermarket shelves was designed, implemented and tested. The unrefined predictions manage to detect holes quite well, but at the price of a relatively low precision. This however, is in accordance with the requirements to the system, because only very few of the false positive detections are actual products. The detection performance can be significantly improved using the refined method, which requires the detection of shelves in the image.

5.1 Main Difficulties

Compared to other detection problems, the main difficulty of this project is that, rather than detecting objects, the task of the hole detection is to detect the absence of objects. Moreover, already the definition of a hole requires knowledge about the shelf and the products contained in the image. This has made the task quite difficult to fulfill using only low-level information. Another difficulty here was that the images come from an uncontrolled environment, i.e. they may contain shelves at different scales and also images taken from different angles. Thus, only very few assumptions about the structure of the images could be made.

The approach taken here is to segment the image into small, homogeneous regions and to classify each of those regions independently. Then bounding boxes for the predictions are found in the image containing the classification results. The main problem of this approach is its locality. Each image region is classified independently using only the information from the region itself and its neighbors. For most of the images those regions are much smaller than the actual holes and hence a lot of valuable information is left unconsidered. In order to overcome the locality of the classification step, also a structural classifier was implemented and tested. The results however showed, that instead of improving performance the classifier actually performed worse. One reason for this may be that the model used was too simple to fully reflect the structural dependencies in the

image. Another reason may be that the images contain holes at many different scales. However, there was not sufficient time to investigate this further. Another difficulty of the chosen approach was to find suitable confidence values for the bounding boxes. The approach taken here was chosen rather heuristically and might not perform optimal.

5.2 Future Improvements

Even though the method presented here is fully functional, there are still ways to improve the system. One improvement would be to include the classification results in the threshold used to discard unlikely predictions in the unrefined version of the hole detection. The value currently used is the relative height of the bounding box, whereas the confidence value computed also considers the classification results. It would be more natural to use only one value for both purposes. Using the classification results to discard false positives might also lead to a better detection performance. Further improvement of the detection results may be obtained by considering the results from the product detection in the refinement step. So far, the products are left unconsidered in the detection step and incorporating this information may lead to an improved precision. One obvious improvement concerning the runtime of the system would be to implement the feature extraction in only one plugin. Also instead of all features only the 50 features that are actually used could be computed. Since the feature extraction contributes most to the overall runtime of the system this should lead to a significant improvement in computational performance.

5.3 Conclusions

The imaging system developed within this thesis is fully functional and yields satisfactory results when refinement is used. Without refinement however, the system suffers from a low precision. The reason is that, due to the restriction of the unrefined hole detection not to use other high-level information, it is not possible to decide with certainty whether a given prediction is a true hole. One drawback of the proposed solution is that the classification is performed on the whole image and then bounding boxes are found in the classification results. The finding of suitable bounding boxes and the computation of the confidence are heuristics and are hard to optimize in their performance. Moreover, the locality of the classification also ignores important information. Using a more integral approach, such as for example a sliding window as it is commonly used for face detection, might sidestep such problems. Nevertheless, even if another approach should prove to be more suitable, this work still contains valuable information for the realization of such an approach.

Bibliography

[1] F. ITWM, Toolip - tool for image processing (2014).
URL http://www.itwm.fraunhofer.de/abteilungen/bildverarbeitung/oberflaecheninspektion/toolip.html

[2] S. Gould, J. Rodgers, D. Cohen, G. Elidan, D. Koller, Multi-class segmentation with relative location prior, Int. J. Comput. Vision 80 (3) (2008) 300–316.
URL http://dx.doi.org/10.1007/s11263-008-0140-x

[3] S. Nowozin, C. H. Lampert, Structured learning and prediction in computer vision, Found. Trends. Comput. Graph. Vis. 6 (3–4) (2011) 185–365.
URL http://dx.doi.org/10.1561/0600000033

[4] . L. Ladický, P. Sturgess, K. Alahari, C. Russell, P. Torr, What, where and how many? combining object detectors and crfs, in: K. Daniilidis, P. Maragos, N. Paragios (Eds.), Computer Vision – ECCV 2010, Vol. 6314 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 424–437.
URL http://dx.doi.org/10.1007/978-3-642-15561-1_31

[5] P. Kohli, L. Ladický, P. H. Torr, Robust higher order potentials for enforcing label consistency, Int. J. Comput. Vision 82 (3) (2009) 302–324.
URL http://dx.doi.org/10.1007/s11263-008-0202-0

[6] T. Gevers, A. Gijsenij, J. van de Weijer, J.-M. Geusebroek, Color Imaging, John Wiley & Sons, Inc., 2012, pp. 26–45.
URL http://dx.doi.org/10.1002/9781118350089.ch3

[7] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, Slic superpixels, EPFL Technical Report 149300.

[8] T. Ojala, M. Pietikäinen, D. Harwood, A comparative study of texture measures with classification based on featured distributions, Pattern Recognition 29 (1) (1996) 51 – 59.
URL
http://www.sciencedirect.com/science/article/pii/0031320395000674

[9] T. Mäenpää, T. Ojala, M. Pietikäinen, M. Soriano, Robust texture classification by subsets of local binary patterns, in: ICPR, 2000, pp. 3947–3950.

[10] S. Movaghati, F. Samadzadegan, A. Azizi, An Agent-Based approach for regional forest fire detection using MODIS data: A preliminary study in iran, Journal of the Indian Society of Remote Sensing (2012) 1–13.
URL `http://dx.doi.org/10.1007/s12524-012-0200-0`

[11] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer Series in Statistics, Springer New York Inc., New York, NY, USA, 2001.

[12] D. Barber, Bayesian Reasoning and Machine Learning, Cambridge University Press, New York, NY, USA, 2012.

[13] N. Christianini, J. Shawe-Taylor, Support Vector Machines and Other kernel-based Learning Methods, Cambridge University Press, 2000.

[14] R. E. Schapire, Y. Freund, Boosting: Foundations and Algorithms, The MIT Press, 2012.

[15] J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting, Annals of Statistics 28 (1998) 2000.

[16] I. Tsochantaridis, T. Joachims, T. Hofmann, Y. Altun, Large margin methods for structured and interdependent output variables, J. Mach. Learn. Res. 6 (2005) 1453–1484.
URL `http://dl.acm.org/citation.cfm?id=1046920.1088722`

[17] K. P. Murphy, Y. Weiss, M. I. Jordan, Loopy belief propagation for approximate inference: An empirical study, in: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, UAI'99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 467–475.

[18] G. Bradski, Opencv, Dr. Dobb's Journal of Software Tools.

[19] B. Andres, B. T., J. H. Kappes, OpenGM: A C++ library for discrete graphical models, ArXiv e-prints.
URL `http://arxiv.org/abs/1206.0111`

[20] D. E. King, Dlib-ml: A machine learning toolkit, Journal of Machine Learning Research 10 (2009) 1755–1758.

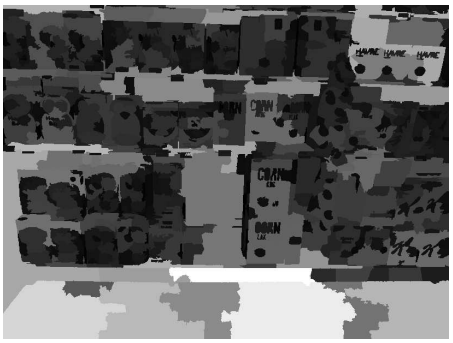
A

Appendix A

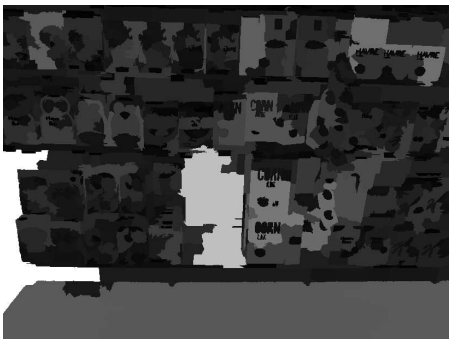
(a) Superpixel Segmented Images

(b) Merged Superpixels

(c) Size of the Merged Superpixel

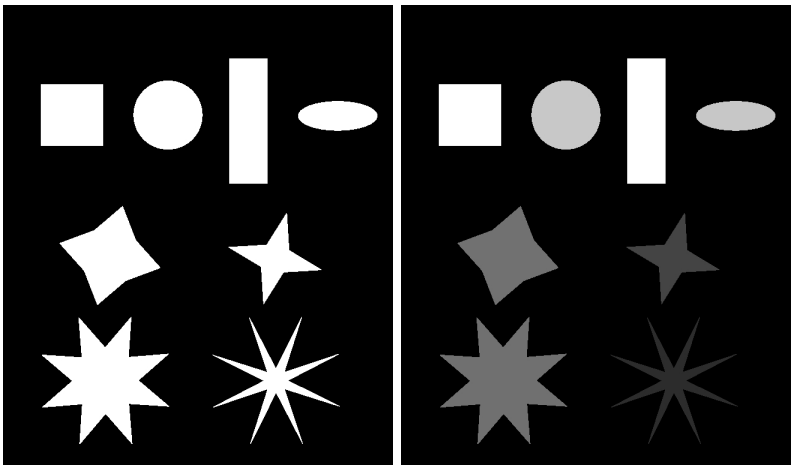


(d) Height of the Merged Superpixel

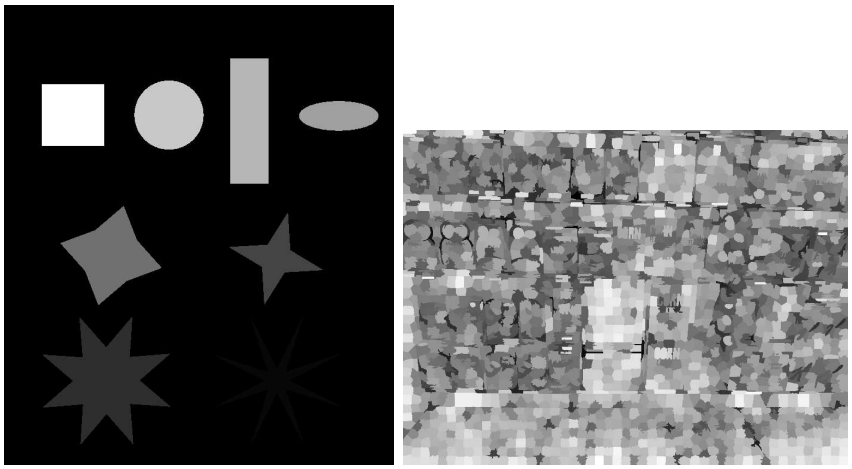


(e) Width of the Merged Superpixel

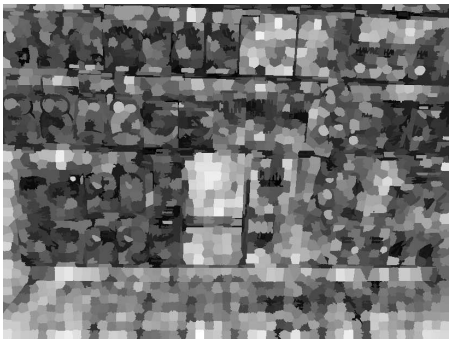
Figure A.1: Features derived from merged superpixels. (a) Displays the input image that was segmented using N_{sp} superpixels. The superpixels are merged depending on the euclidean distance of the superpixels in CIE LAB space (b). Figures (c),(d),(e) display the relative size, width and height of each of the merged superpixels.



(a) Test Shapes **(b)** comp_1 Compactness Measure



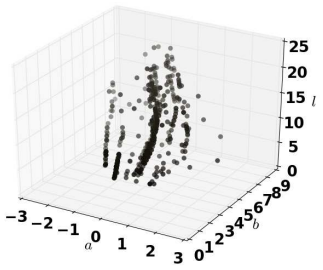
(c) comp_2 Compactness Measure **(d)** comp_1 Compactness Measure



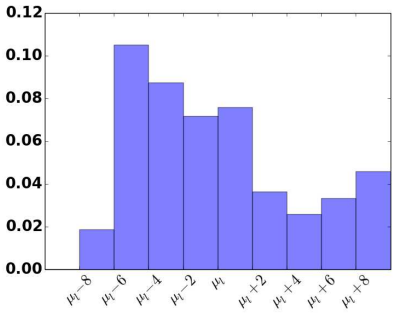
(e) comp_2 Compactness Measure

Figure A.2: Compactness measures. The compactness values of the shapes in figure (a) are computed to illustrate the compactness measures defined in section 3.4.1. Figure (b) and (c) display the values of the compacteness measures of the shapes in figure (a). Figure (d) and (e) the compactness values of the superpixels in a shelf image. As can be seen from figure (d) and (e), the superpixels in products tend to be less compact than the superpixels in hole regions.

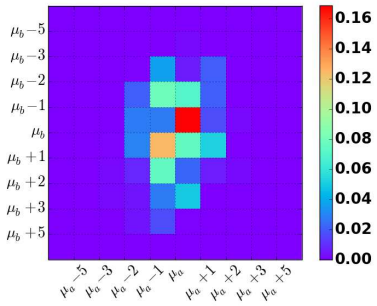
(a) Input Image



(b) Pixel Color Distribution in CIE LAB Space



(c) l-Channel Histogram



(d) a,b-Channel Histogram

Figure A.3: Color Histograms. **(b)** displays the distribution of the pixel colors in the CIE LAB color space of the superpixel highlighted in **(a)**. The histogram of the l channel is displayed in **(c)**, the 2d histogram of the a and b channel is displayed in **(d)**. The histograms are centered around the means of the distributions. Values lying outside of the histogram are handled by applying a cutoff of the highest and lowest bin edges respectively.

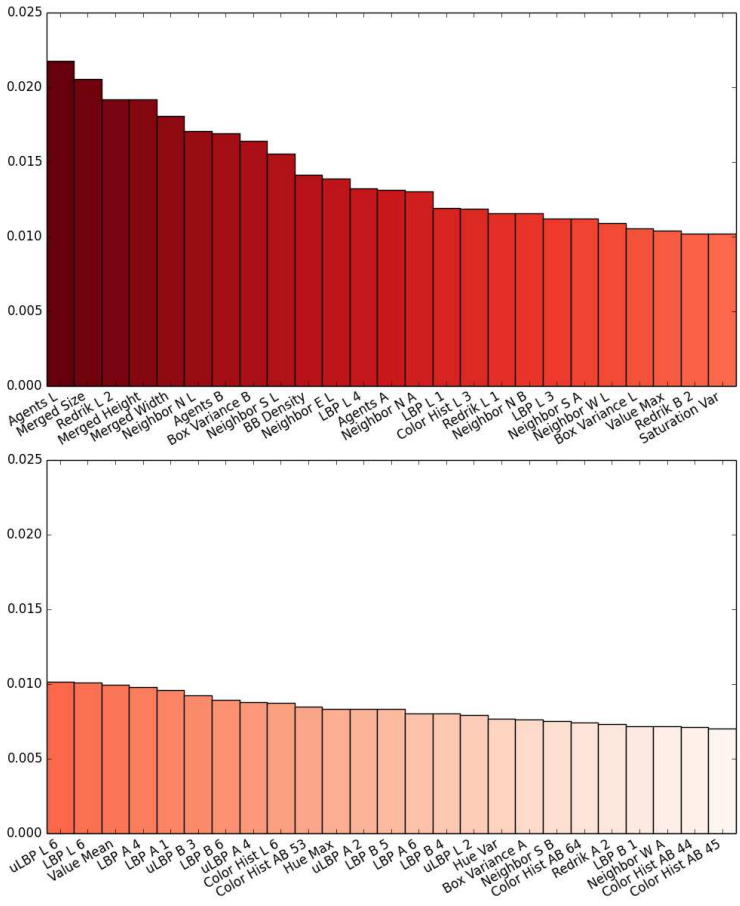


Figure A.4: Relative feature importance. An AdaBoost classifier was trained on the training set and used to computed the relative importances of the features. The figure displays the fifty features with the highest importances. Those are the features that are used in the implementation.

Figure A.5: Hole detection using the SVM classifier. In the second row false holes are detected beneath the shelf. Otherwise, even the unrefined predictions work quite well, due to the fact that all images contain holes.

Figure A.6: Hole detection using the AdaBoost classifier. The results are quite similar to the ones obtained using the SVM classifier. Nevertheless, as can be seen in the first row, for example, the AdaBoost classifiers has more probles with false positive detections.

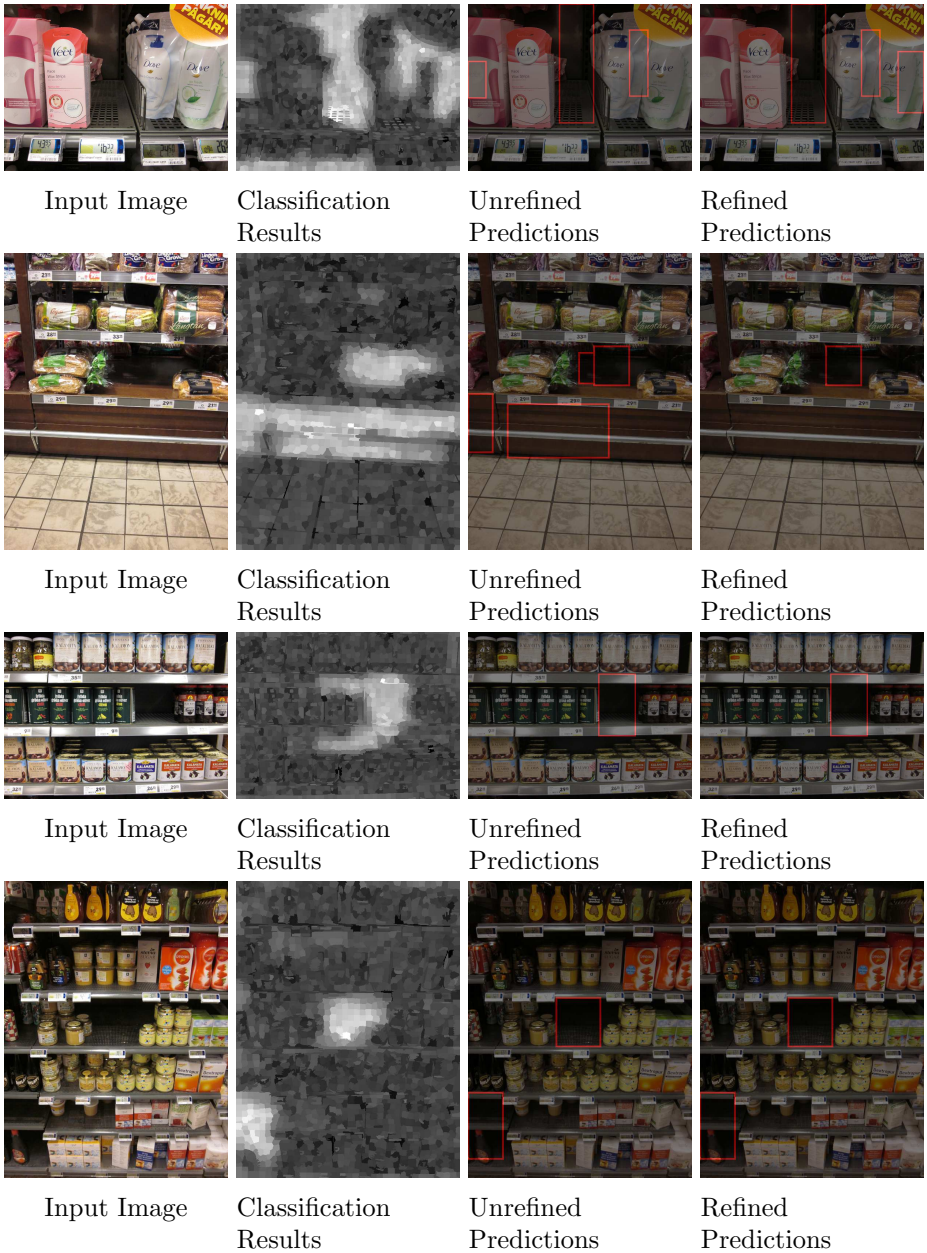


Figure A.7: Hole detection using the CRF classifier. Comparing to the other classifiers one can clearly see the smoothing effect of the CRF classifier. In some cases this helps to eliminate false positives. In other this leads to loss of detail and sometimes even false detections in other regions.

B

Loopy Belief Propagation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$ be a factor graph modeling the conditional probability of a set of output variables $Y_1, \ldots, Y_n$. Belief propagation is a method of conducting probabilistic inference in such graphs. This is done by iteratively passing messages from factors to variables and vice versa until convergence of the marginals. The messages for tree-structured graphs are given by

$$q_{Y_i \to F}(y_i) = \sum_{j \in \mathcal{N}(Y_i)} r_{F_i \to Y_i}(y_i) \tag{B.1}$$

$$r_{F \to Y_i}(y_i) = \log \Bigg(\sum_{\substack{y'_F \in \mathcal{Y}_F \\ y'_i = y_i}} \exp \big(- E_F(y'_F) + \sum_{j \in \mathcal{N}(F) \setminus \{i\}} q_{Y_j \to F}(y'_j) \big) \Bigg). \tag{B.2}$$

The method can also be applied to cyclic graphs, even though it may not converge. The messages for the method for cyclic graphs, called loopy belief propagation, change slightly:

$$\bar{q}_{Y_i \to F_j}(y_i) = \sum_{j \in \mathcal{N}(Y_i) \setminus j} r_{F_j \to Y_i}(y_i) \tag{B.3}$$

$$\delta = \log \sum_{y_i \in Y_i} \exp(\bar{q}_{Y_i \to F}(y_i)) \tag{B.4}$$

$$q_{Y_i \to F}(y_i) = \bar{q}_{Y_i \to F}(y_i) - \delta \tag{B.5}$$

From this the variable marginals can be computed using

$$\bar{\mu}_{Y_i}(y_i) = \sum_{j \in \mathcal{N}(Y_i)} r_{F_j \to Y_i}(y_i) \tag{B.6}$$

$$z_{Y_i} = \log \sum_{y_i \in Y_i} \exp(\bar{\mu_{Y_i}}(y_i)) \tag{B.7}$$

$$\mu_{Y_i}(\mathbf{y}_{Y_i}) = \exp(\bar{\mu}_{Y_i}(\mathbf{y}_i) - z_{Y_i}). \tag{B.8}$$

Finally, also the factor marginals $\mu_{F_i}(\mathbf{y}_{F_i})$ can be computed as follows:

$$\bar{\mu}_{F_i}(\mathbf{y}_{F_i}) = -E_{F_i}(\mathbf{y}_{F_i}) + \sum_{j \in \mathcal{N}(F)} q_{Y_j \to F_i}(\mathbf{y}_{F_j}) \tag{B.9}$$

$$z_{F_i} = \log \sum_{\mathbf{y}_{F_i} \in Y_{F_i}} \exp(\bar{\mu}_{F_i}(\mathbf{y}_{F_i})) \tag{B.10}$$

$$\mu_{F_i}(\mathbf{y}_{F_i}) = \exp(\bar{\mu}_{F_i}(\mathbf{y}_{F_i}) - z_{F_i}). \tag{B.11}$$

C

Appendix C

The following tables contain the values of h_{rel} and σ_n corresponding to the recall precision curves that were used to discard false positives in the unrefined and the refined predictions, respectively.

Table C.1: Threshold values h_{rel}. Values of the threshold used to discard detected holes and the corresponding recall and precision values on the test set.

h_{rel}	SVM		AdaBoost		CRF	
	Recall	Precision	Recall	Precision	Recall	Precision
0.20	0.85	0.31	0.91	0.14	0.83	0.23
0.30	0.85	0.31	0.90	0.2	0.82	0.27
0.35	0.85	0.31	0.89	0.24	0.82	0.3
0.40	0.85	0.31	0.89	0.28	0.82	0.34
0.45	0.85	0.31	0.87	0.32	0.81	0.37
0.50	0.85	0.37	0.85	0.36	0.80	0.42
0.55	0.84	0.42	0.84	0.41	0.80	0.45
0.60	0.83	0.5	0.81	0.44	0.78	0.49
0.70	0.82	0.61	0.77	0.53	0.74	0.58
0.80	0.74	0.71	0.72	0.63	0.66	0.66
0.90	0.64	0.75	0.63	0.69	0.60	0.69

Table C.2: Values of σ_n used as a threshold to discard unlikely holes and the corresponding recall and precision values on the test set.

σ_n	SVM		AdaBoost		CRF	
	Recall	Precision	Recall	Precision	Recall	Precision
0.00	0.14	0.96	0.12	0.88	0.20	0.89
0.10	0.65	0.95	0.57	0.88	0.55	0.91
0.15	0.74	0.93	0.68	0.86	0.63	0.89
0.20	0.78	0.89	0.74	0.82	0.70	0.85
0.23	0.80	0.85	0.75	0.79	0.72	0.84
0.25	0.81	0.84	0.77	0.79	0.74	0.82
0.28	0.82	0.81	0.78	0.78	0.74	0.81
0.30	0.82	0.79	0.78	0.77	0.74	0.79
0.35	0.82	0.75	0.80	0.75	0.75	0.77
0.40	0.82	0.72	0.80	0.72	0.75	0.73
0.45	0.83	0.65	0.81	0.66	0.77	0.67
0.50	0.83	0.62	0.81	0.63	0.79	0.63
0.60	0.83	0.58	0.81	0.57	0.79	0.57