University of Groningen
Dept. Artificial Intelligence

Master's Thesis

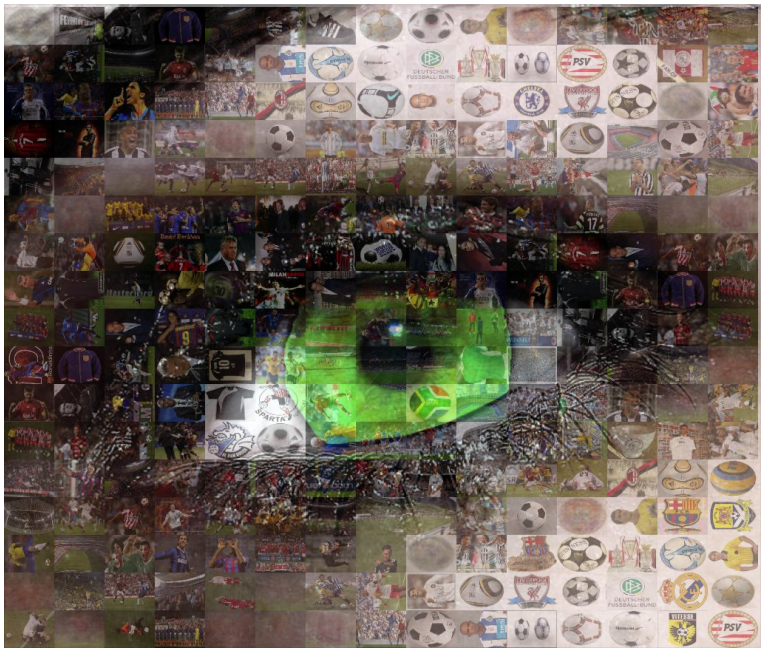
Improving Online Image Queries with Computer Vision

Author:
Maurice Mulder
S1565575

July 25, 2011

Internal Supervisor:
Dr. M.A. Wiering
Artificial Intelligence
University of Groningen

External Supervisor:
M. Homminga
CTO at Kalooga.com



Abstract

Kalooga is a small internet company specialized in providing photo galleries relevant to search queries. This relevance is completely deduced from web-text surrounding the image, giving the image certain labels. One source of information about the image is left untapped: the image content itself. This information could also be used to label images, which is the essential goal of this project: to improve image labeling by exploiting image-content. Content-based image labeling, however, is a difficult task. Automatically applying every possible label to every possible image is far too much to ask. To find realistic and usable results, instead this project focuses on solving problems that Kalooga's current system cannot solve. Such problems are classifying more manageable subdomains of images and detecting or filtering certain types of images.

To this end, an image classification tool was developed, encompassing an entire image classification pipeline allowing quick experimentation and testing on any dataset. The tool implements different classification algorithms, the most successful of which is a Support Vector Machine, and several types of image descriptors like the Histogram of Gradients (HoG). These methods are tested on the Caltech101 dataset as well as a new dataset created from Kalooga's data on soccer. The system performs well, but does not improve on scientific benchmarks.

An interesting alteration was introduced to the HoG, shifting the orientation bins of its histograms. This improves the descriptor's performance and allows it to remain descriptive when greatly decreasing its length. This reduction is useful to classification algorithms, storage and descriptor extraction speed. Simply increasing the number of bins also improves performance when using the HoG descriptor. Thus, when extracting HoGs, not only the number of histograms should be considered but also the binning method.

Comparing results on both datasets shows HoG to work well on Caltech101, but poorly on Kalooga data. This is because the Kalooga dataset's images have much higher inner-class variability. To achieve better results, the problem was scaled down to a detection task instead of a multi-class classification task. Detecting action scenes in the Kalooga soccer dataset, a task useful to Kalooga's presentation, the system performs well and is promising for use on real data.

For this task, combining color and texture descriptors also proves to be just as useful as variations of the HoG descriptor. So when deciding which descriptor to use, the target image domain should be well tested.

Acknowledgements

I would like to thank my external supervisor, Mathijs Homminga, and the other friendly people over at Kalooga. Not only for accommodating me throughout this project, but also for the help, interesting conversations, kick offs in the parking lot and generally including me in Kalooga's everyday life.

I'd like to thank my internal supervisor, Marco Wiering, to whom I'm indebted to for his knowledge, motivation and even more inspiration.

Lastly, I'd like thank my family and friends for their unconditional support and my girlfriend, Sanne, whose relentless faith became daunting enough to finally make me finish this.

"By the way: thank you. Without you I'd just be a weirdo, talking to myself." - *Ze Frank*

Contents

1 Introduction **3**
- 1.1 Related Research . . . 4
- 1.2 Research Question . . . 5
- 1.3 Approach and Pipeline . . . 6
- 1.4 Outline . . . 8

2 Machine Learning **9**
- 2.1 K-Nearest Neighbors . . . 10
- 2.2 Support Vector Machine . . . 11
  - 2.2.1 Linear Classification . . . 11
  - 2.2.2 Maximum Margin . . . 13
  - 2.2.3 Support Vectors . . . 14
  - 2.2.4 Dual Form . . . 14
  - 2.2.5 Soft Margin . . . 15
  - 2.2.6 Kernels . . . 16
  - 2.2.7 Multi-class SVM . . . 16
  - 2.2.8 Parameter Selection . . . 16
- 2.3 Bags of Visual Keywords . . . 18
  - 2.3.1 K-Means Clustering . . . 18
  - 2.3.2 Sliding Window . . . 18
  - 2.3.3 Hard Bag of Features . . . 20
  - 2.3.4 Minimum Distance map . . . 20
- 2.4 Feature Combination . . . 20
- 2.5 Information Retrieval . . . 21

3 Image Descriptors **23**
- 3.1 Color Histogram . . . 24
- 3.2 Histograms of Gradients . . . 25
  - 3.2.1 Original Sift Descriptor . . . 25
  - 3.2.2 Changes to Sift Descriptor . . . 26
- 3.3 Mpeg7 Features . . . 28
  - 3.3.1 Edge Histograms . . . 28
  - 3.3.2 Scalable Color . . . 28
  - 3.3.3 Color Layout . . . 28
  - 3.3.4 Dominant Color . . . 29

4 Implementation **30**
4.1 Programming Language and Approach 30
4.2 Libraries 30
4.3 Computer Vision Tool 31
4.3.1 Graphical User Interface 32
4.3.2 Datasets 32
4.3.3 Parameters and Descriptors 32
4.3.4 Using the Vision Tool 34
4.3.5 Discussion 35

5 Experiments and Results **36**
5.1 Datasets 36
5.1.1 Caltech101 36
5.1.2 Kalooga Dataset 37
5.2 Parameter Optimization 39
5.3 Experimental Setup 39
5.4 Color Histogram and MPEG7 Descriptor Experiments 40
5.4.1 Discussion 40
5.5 HoGs experiments 40
5.5.1 Basic Performance 40
5.5.2 Phase Shift and Orientation Bins 42
5.5.3 Use Of Gradient Diagonals 43
5.6 Influence of Training-Set Size 44
5.6.1 Discussion 44
5.7 Visual Keyword Descriptors 45
5.7.1 Discussion 45
5.8 Descriptor Combination 46
5.8.1 Discussion 46
5.9 Kalooga Dataset 47
5.9.1 12 Classes 47
5.9.2 Action vs All 48
5.9.3 Information Retrieval Approach: Detecting Action Images 49

6 Conclusion and Future Work **52**
6.1 Conclusion 52
6.2 Future Work 53

Chapter 1

Introduction

A picture is worth more than a thousand words. Visual information grabs our attention: the images are often the first thing we look at when opening a newspaper. Visual information also helps us understand concepts more quickly. Directions to a particular location remain cryptic and confusing until even the simplest map is drawn besides it. Visual information evokes emotions. Images of young children and creatures make many of us smile involuntarily, while graphic scenes of the human drama, war or disaster may just as easily tear at our hearts. Visual information is able to give a better scope of raw facts and information. We often seem to make estimates and summaries from visual information all the time. At any rate, it is clear that the visual modality is very important to human thinking.

As a result of this, our perceptual environment has become flooded with imagery. Every day we are bombarded with art, diagrams, maps, photography and brightly colored advertisements depicting everything from sports-cars to sweating beer-glasses. We find them on massive billboards, the sides of buses and practically every product we buy.

The same has become true for the internet. With the development and popularization of digital photography, the web and online photo sharing, a large and ever growing amount of digital imagery has now become instantly accessible. This has also created a need for a way to organize these images and retrieve them for a given query.

Kalooga is an internet company that aims to provide such a service and specializes in gallery based image retrieval. Image galleries are different to regular images randomly found on the web. In image galleries, the images themselves are the primary concern. They tend not to be simple illustrations or auxiliary elements to a story like non-gallery images found on the web do. These images are generally of higher quality, both in fidelity and esthetic value. When crawling important gallery sources on a regular basis, it is easier to find recently created galleries with great news value. Galleries are also more likely to be relevant to a certain topic or event and are generally more newsworthy. This separates Kalooga from other online image services, along with the specialized services Kalooga offers to publishers.

Kalooga offers widgets, small software applications that can be executed inside a webpage, to publishers. These widget automatically illustrate the articles they are placed in. This is done by composing an image query from the article's

title and body and automatically selecting the most appropriate images from this query for display in the widgets, though sometimes these need to be filtered by hand. These widgets are meant to enrich the user experience with quality images relevant to the article.

Currently, Kalooga achieves its search results by labeling images and galleries using textual information such as the image's URL, its tags, and the text surrounding the image. This works sufficiently well, however, no visual content is exploited towards their cause. Kalooga and the University of Groningen have decided to work together to research possibilities in exploiting image content to improve Kalooga's search results. This master's project has aimed to improve the results of image database queries by applying content-based image classification.

The field of computer vision has come a long way, and may help harness the information present in the images themselves. There is no definitive solution to the general problem of image classification yet, but the field has produced and applied many methods that are effective in solving more specific image-related problems. The following section will explore these methods.

1.1 Related Research

The problem of finding labels or categories to go with images based on the image's content and other related tasks have received a lot of attention in the last decade. The research topic has received several names over the years; to name a few: image annotation, image retrieval, image classification, image categorization, object recognition and scene recognition. While the purpose, emphasis and approach of these different studies addressing this problem vary, they all come down to the same fundamental problem: the identification of some semantic properties of an image given only the content of an image.

This project's main goal is to create a system that can be used to improve online image query results. This is a somewhat overwhelming problem because the number of labels applicable to web imagery is very large and grows constantly. Labels can describe image features ranging from low level image features, objects present in the image or scene identifications. Furthermore the images to be annotated are extremely variable. Because of this some simplifications need to be made to the problem to make it more manageable.

Many previous works do this by limiting the labels assigned to images. Some make matters even simpler by avoiding object names and only use general descriptions based on color and texture such as *sky*, *people* and *grass*. This kind of simplification is mainly due to what is called the *semantic gap* (Datta et al., 2008). The semantic gap is the gap between the information extractable from an image and the information as interpreted by a person viewing the image. This is a deep problem in image classification and annotation that has not yet been solved.

Another simplification is to limit the domain of an annotation system, to either a very limited amount of classes or to a set of images that share many visual properties. This not only limits the number of labels to be used but also the variability of the images.

Once the domain and complexity of the task have been decided on, one can focus on the task itself. One important aspect of this task is extracting an image

signature from either the entire image or one or more segments of the image. This image signature is a numerical description of the image composed of one or more features extracted from the image describing its color, texture, shape or key-points. These signatures are then used to learn from a training set and later to classify new images using machine learning techniques.

Many features describe an object class more efficiently when extracted from the image area containing the object. Extracting features from the image at different scales and segments can help in extracting more descriptive features. Such segments can be selected regularly, randomly, using saliency maps (Itti et al., 1998) or by segmenting the image. Some approaches cluster segmentations and describe a single image by which clusters are represented in them ("bag of words" description) (Harchaoui, 2007). This approach is referred to as *region-based image retrieval* (RBIR) and has been shown to be generally effective at classifying scenes (Liu et al., 2007).

A shared requirement of content-based image annotation systems is a numerical description of images. Such a description can be obtained by extracting a range of useful features from the image. These features either exploit the image's color composition, textural properties, its shapes or the salient points within the image (Datta et al., 2008). Deselaers et al. (2008) give an overview and comparison of popular features used for content-based image retrieval. Popular features are *color histograms*, various implementations of *SIFT* features, *MPEG7* features, features obtained from the *wavelet transform* and the *Gabor histogram*.

The Scale Invariant Feature Transform (Lowe, 1999, 2004) (SIFT) is a popular algorithm that is able to detect visual patterns regardless of size, scale and rotation. It detects important keypoints in an image and is able to match them to similar keypoints in other images based on a keypoint descriptor. When extracted in a grid, instead of from a selection of keypoints, this keypoint descriptor has been shown to be effective at object recognition and image classification.

The image signature, i.e. the extracted image description, can be used as the input to a classifier for training and classification. There is a wide variety of classification methods that have been used and could be used as classifier. Support Vector Machines (SVMs) (Cortes and Vapnik, 1995), have been shown to be effective in image recognition (Zhang et al., 2001; Datta et al., 2008). Their performance can be further improved using ensembles like stacking techniques (Abdullah et al., 2009a) and other ensemble methods (Abdullah et al., 2009b).

1.2 Research Question

This brings us to the central question of this research: how can computer vision methods aid in improving online image query results?

Finding all applicable labels to an image using only its visual content is simply not feasible with computer vision as it is today. Firstly, a single image can be labeled by an overwhelming number of words, due to the versatile and replaceable nature of language. On top of that, many of these labels are abstract and represent a non-visual aspect of the image, such as proper names, locations, action etc.

This requires the scope of the problem to be reduced to a more manageable size. Instead, this research focusses on creating a system that may solve

problems a company like Kalooga might come across. Such problems could include the need to detect images with a certain visual property that makes them preferable over other images. Inversely, Kalooga may wish to filter out unwanted images from a certain group of images based on visual information. Furthermore, as Kalooga mostly deals with serving query results of some way or form, it will also be interesting to sort images according to some visual quality. That way users can be presented with the most relevent, high quality images first.

To achieve this, a general image classification pipeline has been developed with a full range of visual descriptors, described in the next chapter. The use of this entire pipeline will be supported by a user friendly tool, allowing quick composition and testing of image classification pipelines on any user dataset. Furthermore, a dataset has been assembled out of Kalooga's data composed completely of soccer related images. This dataset poses a far greater challenge to the implemented methods and gives a more realistic view of their performance on a real problem.

1.3 Approach and Pipeline

The approach to the problem of this project will be that of image classification. The image classification pipeline as it was implemented in this project has been illustrated in figure 1.1. To train a model on a database of categorized images, the image description is first extracted using a certain visual descriptor algorithm. These visual descriptions are then used in conjunction with the image classes to train a classification model using a certain machine learning algorithm. When new images are to be classified, their image descriptions are also extracted in the same fashion. These can then be used by the classifier in conjunction with the trained model to predict the class to which the image belongs.

In figure 1.1 one can see that *machine learning algorithms* and *visual descriptors* are central to the system. Much of this thesis is about these methods and their effectiveness on different datasets. These functions will be fulfilled by a number of machine learning and computer vision methods.

For the machine learning algorithm, there are two classification algorithms that have been applied. The first is the K-Nearest Neighbor algorithm which makes up a baseline classifier for the system. This algorithm was chosen as it complements the nature of the problem. The second are Support Vector Machines, which are expected to achieve a higher performance. This method was chosen as other studies have managed to achieve good results in image classification using SVMs.

For the visual descriptor element, a range of descriptors will be applied to contain at least one of each type: color, texture and shape. The first to be implemented was a color histogram, which is a simple baseline descriptor. Secondly, the Histogram of Gradients (or HoG) was created, which is based largely on the SIFT keypoint descriptor. This descriptor will also be altered in two ways. Lastly, four descriptors defined in the MPEG7 standard are used, namely the Edge Histogram, Color Layout, Dominant Color and Scalable Color descriptors.

There are two main exceptions to this pipeline, namely: *visual keyword*

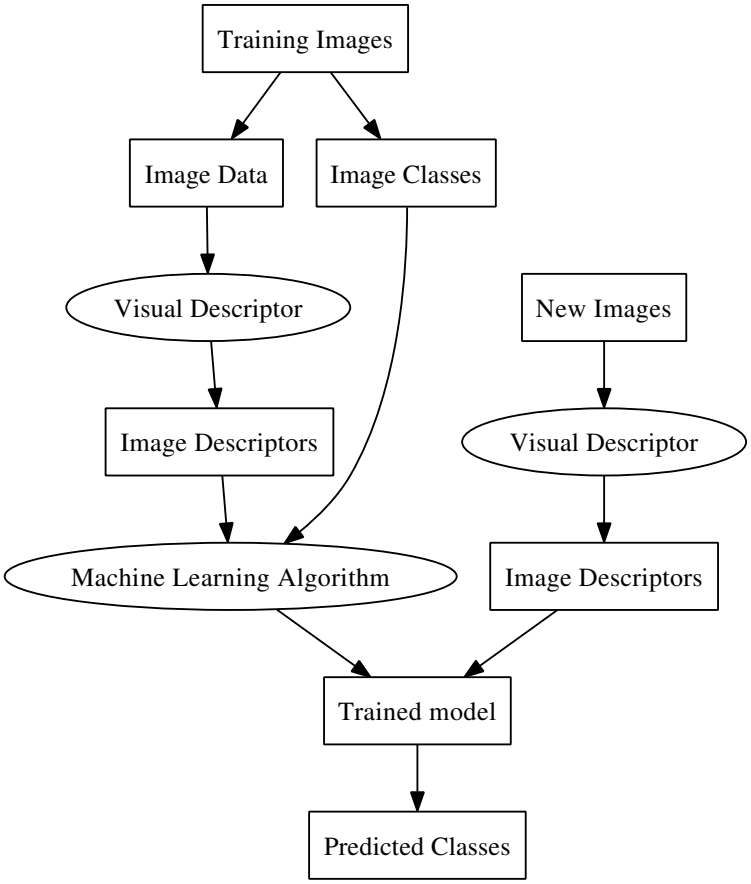


Figure 1.1: Image Classification Pipeline.

descriptors and *feature combination methods*. A visual keyword approach is used in which a codebook is first made, by clustering patches of images using k-means clustering. The images are then expressed in terms of this codebook, by creating visual keyword descriptors.

Image descriptors will also be combined using two feature combination methods. The first is a simple naive approach, concatenating visual descriptions. The second is an SVM classifier pyramid approach, training SVMs for each feature and using their posterior probability outputs as input for a final SVM.

1.4 Outline

In the next chapter, chapter 2, the machine learning methods used in this project are described. This includes k Nearest Neighbor Classifiers, the use of Support Vector Machines as well as K-means clustering and classifier stacking techniques. After that the input for these methods, the visual descriptors are discussed in chapter 3. This includes the color histogram, MPEG7 features and the Histogram of Gradients and the alterations made on this descriptor. These machine learning and computer vision methods have been combined into a user friendly tool for quick development and testing of image classification pipelines. This tool is discussed in chapter 4. The system has been tested on the Caltech dataset (Fei-Fei et al., 2004) and the Kalooga dataset. These datasets and the results of the system on them are discussed in chapter 5. Finally a discussion and conclusion is given in chapter 6.

Chapter 2

Machine Learning

In this chapter we discuss the machine learning algorithms implemented in this project.

The problem this project intends to solve is the classification of images. Such classification problems are addressed by a number of fields such as pattern recognition and machine learning. In this project, several machine learning algorithms have been implemented and applied. Machine Learning covers algorithms that allow machines to generate or learn useful behaviors from data. This data may be anything from a live recorded stream of sensory input, to a prerecorded database of information. The behaviors that are generated are forms of automated decision making that can range from classifying a set of inputs, to controlling a complex interface. In our case, the data are simple digital images and the desired behavior is assigning these images an appropriate label.

As machine learning algorithms attempt to learn they often rely on human knowledge of some form to provide them a measure for their success. The manner in which any human knowledge is presented to an algorithm allows us to distinguish three different types of Machine Learning: *Supervised Learning*, *Reinforcement Learning* and *Unsupervised Learning*.

In Supervised Learning, each problem is presented as inputs paired with the corresponding correct response, forming examples. A learning algorithm is supervised by providing it with the correct answer to each problem it tries to solve. This is a popular approach to the problem, image classification, as image-label pairs are readily available as datasets and can be easily generated.

Reinforcement Learning is mostly applied to problems for which it is difficult to produce observation-decision pairs. This is often because there are long term effects for short term decisions. When for instance navigating a maze, a short term decision will not get you to your target directly but may be essential to your eventual success. Describing such problems with simple observation-decision pairs is often inappropriate and ineffective. Instead of the observation-decision pairs used in supervised learning, the reinforcement learning algorithm requires a cumulative reward. This cumulative reward summarizes the performance of multiple decisions taken by the algorithm, each decision being made with its own local observations. No use will be made of such methods in this project.

Unsupervised Learning is applicable when the desirable behavior to be exhibited by the algorithm can neither be described by labeling examples on a case by case basis nor by summarizing it in a final cumulative reward value.

Unsupervised learning, instead, extracts information from only the input data itself, for instance by clustering. In this project one clustering methods will be used: *k-means clustering*.

The focus of this project, image classification, encourages the use of Supervised Learning methods, because it is very easy to create training sets. Images are readily available and it is not hard to create groups of images with similar content to function as training data. Within supervised learning algorithms, a distinction that can be made is in their pro-activeness. On one hand a learning algorithm may actively use the data it has been given to learn with, creating a model that generalizes these examples in some way. These algorithms have a time and resource intensive training phase in which the example data are processed. However, due to their generalizing nature they often are resource friendly when used on new inputs. Thus, the 'work' done by the algorithm, learning a pattern, is done in its training phase. Such learning algorithms are called *Eager Learning* algorithms, as they are proactive.

On the other hand a learning algorithm may not really have a training phase and only store the given examples. These algorithms will use the examples at classification time, without previously creating an explicit model. Such algorithms are called *Lazy Learning* algorithms, as they are not proactive but are 'lazy' and wait until a classification is required.

The goal of classification is identifying the class an object belongs to by the object's characteristics. These characteristics can be a full description of the object itself or a summary of certain properties of its content. Either way a finite, numeric description of the object is required called the object's features, or its description. The objects to be classified in this project are images. Features extracted from images summarize certain properties of images and help by reducing the description size for machine learning algorithms. The features extracted from the images are explained in chapter 3.

2.1 K-Nearest Neighbors

A simple solution to the problem of classifying new datapoints using a set of examples is as follows: one simply finds the labeled example closest to that datapoint, its neighbor, and uses that example's label for the target. A generalization of this technique is called the *k-nearest neighbor* algorithm (KNN) (Cover and Hart, 1967).

KNN is a supervised learning algorithm as it expects a clear input-decision pair: in this case a description and a label. The KNN algorithm processes each training example when classifying a new datapoint a. For each training example b it has stored it calculates a certain distance measure d between b and a, which are N-dimensional. Any distance measure can be used, of which the most popular is the Euclidian distance shown in equation 2.1. This is also the distance measure used throughout this project.

$$d(a,b) = \sqrt{\sum_{i=1}^{N} (a_i - b_i)^2} \tag{2.1}$$

The algorithm selects the k datapoints that are closest to a. These k examples are then used to find a label for a by letting them vote for their own

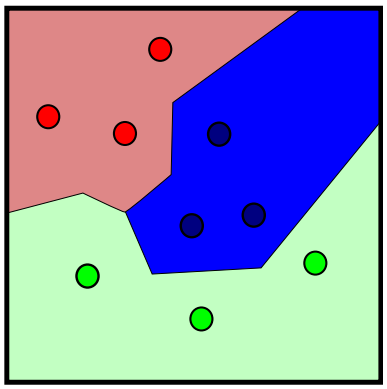


Figure 2.1: Given a k of 1, each training example in the KNN will produce its own space around it in which datapoints will be classifier as that class. This creates a Voronoi diagram. When k increases it acts as a smoothing factor on the decision boundaries.

label. In this project the votes are weighted by dividing them by the distance to a because of the great number of categories to distinguish. Thus k nearest neighbors are used to determine a label for unlabeled data.

When k is set to 1, a new input will be assigned the same label as its closest neighbor. Each input can be considered a point in a space that has as many dimensions as the input has values. The KNN defines decision boundaries separating this space for each of its examples. Inputs within the area around a datapoint will be classified as belonging to that datapoint's class. In this way these boundaries define the implicit, instance-based model described by the KNN. This model in a two-dimensional feature-space is visualized in figure 2.1, showing a Voronoi diagram for $k = 1$. Increasing the k value, allowing more neighbors to vote on the input's label, will cause the decision boundaries to be smoothed.

The algorithm is 'lazy' as it uses its training data at classification time, directly using the instances provided as training set without training a model. Though simple, this algorithm is very effective given enough data.

2.2 Support Vector Machine

Support Vector Machines (SVM) (Cortes and Vapnik, 1995) are eager supervised learning methods for classification. The Support Vector Machine is a *linear classifier*. To explain how an SVM works, first linear classification and support vector machines in the case of a two-class classification problem will be discussed. Next the problem will be expanded to the classification of datasets having an arbitrary number of classes.

2.2.1 Linear Classification

A two-class linear classifier classifies new datapoints given a set of example datapoints belonging to either of two classes. Each datapoint is described with

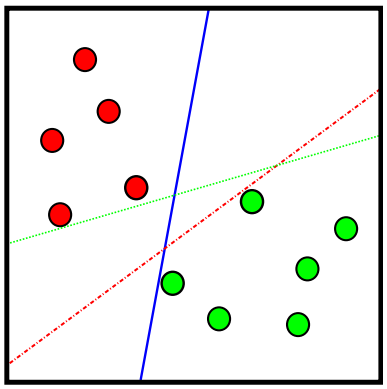


Figure 2.2: Each circle represents a datapoint within a 2d feature space. The lines represents hyperplanes, that all separate the datapoints.

an N-dimensional feature vector, and so each datapoint can be represented as a point in an N-dimensional *feature space*.

The task of the linear classifier is to find a suitable hyperplane, a $N-1$ dimensional space, that separates the example datapoints. Given a 2-dimensional feature space for the datapoints, the feature space could be represented by a surface. The separating hyperplane would in turn be represented by a 1-dimensional space, i.e. a line, in this feature space. Such a feature space is depicted in figure 2.2 along with a set of datapoints and a set of possible hyperplanes separating the two classes.

A linear classifier uses a linear combination of its input values to make a classification decision by defining an output value $g(\vec{x})$ for a new input $\vec{x}$. $g(\vec{x})$ includes a threshold parameters b to offset the hyperplane from the origin of the feature space. In a two-class problem, datapoint $\vec{x}$ will belong to one class when $g(\vec{x})$ is greater than zero and to the other class when smaller than zero. Any linear combination of input values can be realized by a dot product with a chosen weight vector $\vec{w}$ shown in equation 2.2 .

$$g(x) = \vec{x} \cdot \vec{w} + b \tag{2.2}$$

Given this output value, the hyperplane it creates to separate the feature space is defined in equation 2.3.

$$g(x) = \vec{x} \cdot \vec{w} + b = 0 \tag{2.3}$$

For this hyperplane to separate the datapoints in two classes it needs to fulfill two requirements for each training example x_i with label y_i:

$$g(\vec{x}_i) = \vec{w} \cdot \vec{x}_i + b \geq 1 \qquad \textbf{if } y_i = 1; \tag{2.4}$$

$$g(\vec{x}_i) = \vec{w} \cdot \vec{x}_i + b \leq -1 \qquad \textbf{if } y_i = -1; \tag{2.5}$$

or, in a single equation:

$$g(\vec{x}_i) = y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \tag{2.6}$$

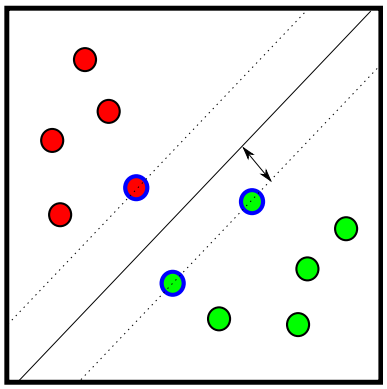


Figure 2.3: An SVM tries to maximize the margin of its decision boundary. In this figure, non-zero alpha valued datapoints, the support vectors, have been given a larger, blue outline. The line represents the separating hyperplane, while the dotted lines are the same line with different offsets going through the support vectors. The arrows point out one half of the maximum margin.

This only leaves a weight vector $\vec{w}$ and offset b to be constructed that fulfill equation 2.6 for each training example. The type of linear classification algorithm used defines how $\vec{w}$ is constructed.

2.2.2 Maximum Margin

In general there are two main approaches to find b and $\vec{w}$: generative models and discriminative models. Generative models are probabilistic models that specify a probability density function and are able to generate random samples within feature space. Discriminative models are not required to be probabilistic but cannot generate random samples within feature space. Discriminative models generally have a greater performance on classification tasks while generative models are often more flexible.

An SVM is a discriminative non-probabilistic linear classifier. The SVM tries to find the hyperplane that separates the datapoints with the largest possible margin. Given a hyperplane separating the dataset, we can define the hyperplanes that lie on the border of its margin:

$$\vec{w} \cdot \vec{x} + b = 1 \tag{2.7}$$

$$\vec{w} \cdot \vec{x} + b = -1 \tag{2.8}$$

The optimal hyperplane can then be defined as the hyperplane that maximizes the distance between these two hyperplanes. A representation of the maximum margin decision boundary in a two-dimensional feature space and the two border hyperplanes is given in figure 2.3. This distance between these hyperplanes is $\frac{2}{||\vec{w}||}$, which means to maximize the margin we need to minimize $||\vec{w}||$.

2.2.3 Support Vectors

However the $\vec{w}$ vector still needs to be constructed. The SVM algorithm does this by choosing its set of *support vectors*. These define the weight vector $\vec{w}$ and with it the hyperplane $g(\vec{x}) = 0$. Support vectors are found by evaluating each datapoint d_x in dataset D for their relevance α_x to the eventual decision boundary. The datapoints most important to the decision boundary should become support vectors while others should receive a zero value for α_x. Each non-zero value is considered a support vector. The α value is a real positive value that denotes the relevance to the decision boundary of a training example. Using the α values, the weight vector $\vec{w}$ can be constructed:

$$\vec{w} = \sum_{i=1}^{l} \alpha_i y_i \vec{x}_i \tag{2.9}$$

Where l is the number of support vectors, $\vec{x}_i$ is support vector i and y_i is the label $\{-1, 1\}$ of datapoint $\vec{x}_i$.

As shown before, to find the maximum margin hyperplane, $||\vec{w}||$ should be minimized while w still fulfilling 2.6. As calculating the norm $||\vec{w}||$ requires a square root, we can change this to $\frac{1}{2}||\vec{w}||^2$ for convenience. This brings us to:

$$\textbf{minimize} \quad \frac{1}{2}||\vec{w}||^2 \quad \textbf{subject to} \quad y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \tag{2.10}$$

This representation is the quadratic programming problem known as the primal form (Nocedal and Wright, 2006).

2.2.4 Dual Form

We can find our $\vec{w}$ using the optimization method of Lagrange multipliers (Rockafellar, 1993). A specific optimization problem can be defined as follows:

$$\textbf{minimize} \quad f(x, y) \quad \textbf{subject to} \quad g(x, y) >= c \tag{2.11}$$

where $f(x, y)$ is the function to be optimized, and $g(x, y)$ and c define the constraints. Given this optimization problem and its definition in equation 2.11, the Lagrange function is defined as:

$$\Lambda(x, y, \alpha) = f(x, y) - \alpha(g(x, y) - c) \tag{2.12}$$

$$min_{\vec{w},b} max_{\alpha} \Lambda(x, y, \alpha) \quad \textbf{subject to} \quad \alpha \geq 0 \tag{2.13}$$

The Lagrange function introduces a new variable, the Lagrange multiplier α.

Our optimization problem was defined in equation 2.10. Applying the Lagrange function to this problem gives us the following equation:

$$min_{\vec{w},b} max_{\alpha} \quad \frac{1}{2}||\vec{w}||^2 - \sum_i \alpha_i (y_i(\vec{w} \cdot \vec{x}_i + b) - 1) \quad \textbf{subject to} \quad \alpha_i \geq 0 \tag{2.14}$$

The Lagrange multipliers introduced in this equation will eventually become our support vectors, the α described in equation 2.9.

α should maximize Λ where $\vec{w}$ and b should minimize it. That leaves us to find the saddle points for $\vec{w}$ and b, which can be found by deriving. Minimizing b leads to the following constraint:

$$\frac{\delta\Lambda}{b} = 0 = \sum_i \alpha_i y_i \tag{2.15}$$

$$\frac{\delta\Lambda}{\vec{w}} = 0 = \vec{w} - \sum_i \alpha_i y_i \vec{x}_i \tag{2.16}$$

Which leads to:

$$\vec{w} = \sum_i \alpha_i y_i \vec{x}_i \tag{2.17}$$

which corresponds to equation 2.9. By substituting this in the maximization problem, the following is found:

$$max_\alpha \quad \tfrac{1}{2} \textstyle\sum_i \alpha_i y_i \vec{x}_i \sum_i \alpha_i y_i \vec{x}_i - \sum_i \alpha_i y_i \vec{x}_i \sum_i \alpha_i y_i \vec{x}_i + \sum_i \alpha_i$$

$$\textbf{subject to} \quad \sum_i \alpha_i y_i = 0, \alpha_i \geq 0 \tag{2.18}$$

Or, shorter:

$$max_\alpha \quad \sum_i^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\vec{x}_i \cdot \vec{x}_j) \quad \textbf{subject to} \quad \sum_i \alpha_i y_i = 0, \alpha_i \geq 0 \tag{2.19}$$

This is known as the dual form.

2.2.5 Soft Margin

The training data may not be linearly separable because of noise or an inherent property of the class. One would want the SVM to still find a hyperplane that separates the datapoints as cleanly as possible. The cleanest cut would be the cut having the fewest possible errors which themselves are as close to the margin as possible. To achieve this, the Soft Margin SVM was introduced by Cortes and Vapnik (1995). The soft margin SVM introduces an error cost term to the optimization objective. This term is defined using the distance of each datapoint to its target class, ϵ_i. When the datapoint is already classified correctly under the current $\vec{w}$ then this distance will be 0. Thus $\epsilon_i \geq 0$. This changes equation 2.10 to:

$$\textbf{minimize} \quad \frac{1}{2}||\vec{w}||^2 + C \sum_i \epsilon_i \quad \textbf{subject to} \quad y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \epsilon_i, \quad \epsilon_i \geq 0 \tag{2.20}$$

2.2.6 Kernels

The size of the feature vectors provided as training examples defines a feature space in which these feature vectors are points. However, linear separability of given classes in this feature space is quite unlikely to occur in standard feature space. More likely the datasets used will be very complex. To solve this problem the feature vectors are projected into a higher dimension using a *kernel function* $K(x, y)$ (Boser et al., 1992). Intuitively, a kernel adds another dimension for every point. This dimension is constructed from the values already available in the feature vector. This new dimension then allows the separation of subsets of points that were previously not linearly separable.

The kernel used by the SVMs is a gaussian *Radial Basis Function* as described in equation 2.21.

$$K(x_i, x_j) = e^{-\gamma||x_i - x_j||} \tag{2.21}$$

2.2.7 Multi-class SVM

Up until now the problem discussed was limited to that of two-class classification. It is also possible to distinguishes more than two classes in the feature space. This is done by defining the multi-class classification problem as a combination of two-class classification problems. There are two main approaches to this:

In the *one-versus-all* approach, the problem is divided into a binary classification problem for each available class. Each binary classifier distinguish the proper class against all other classes. The class belonging to the classifier giving the highest output value will then be assigned to a new datapoint.

In the *one-versus-one* approach, a binary classifier is built for each unique combination of classes. The ultimate classification in this approach is achieved by letting each classifier vote toward one of two classes. The absolute value of each classifier's output is added to the vote of the favored class of that classifier. The class with the greatest vote becomes the classification of the instance.

In this project the one-versus-one approach is employed.

2.2.8 Parameter Selection

The performance of an SVM is dependent on certain parameters used. In the case of the current setup these include the γ parameter, which is used by the kernel and the C parameter, which defines the soft margin. To find values for these variables that suit the system best, i.e. result in the highest performance, a grid search is employed. The grid search iterates a certain domain of values for each of n variables it optimizes, forming an n-dimensional grid. For each set of variable values the system is tested, storing the best results and the belonging variables. The system is tested the same way it is eventually evaluated: using randomly selected datapoints.

This fairly standard grid search has been altered using statistical methods in two ways to increase efficiency without significantly increasing the chance of missing performance peaks. These alterations are original to this project. The first assures a low pooled variance of each sample made in the grid search. The

second aborts low expectation experiments prematurely. Each is described in its own section below.

Dynamic Estimation of Required Sample Size

Each experiment needs to be repeated a sufficient number of runs depending on the standard deviation to assure that parameters are not selected or discarded based on biased data. This is done by requiring the standard deviance of the sampled mean, σ_{mean}, to be below a certain threshold, c. Whenever a set of parameters are being tested in the grid search, runs are added to minimize the standard variance of the mean until the following is true:

$$\sigma_{mean} = \frac{\sigma}{\sqrt{N}} < c \tag{2.22}$$

Where N is the number of samples, i.e. runs for this particular parameter setup so far an σ is the standard deviation of the current sample aggregated in the N runs. This equation can be simplified to avoid divisions:

$$\sigma < c \cdot \sqrt{N} \tag{2.23}$$

Which can be further simplified to prevent having to find a square root, which is a costly operation.

$$\sigma^2 < c^2 \cdot N \tag{2.24}$$

This prevents us from calculating square roots for both the variance σ^2, and the number of runs in the sample N. In the optimization done for this project, c has been set to: 0.004.

Experiment Abortion by Hypothesis Testing

In practice, the grid search quickly became too time consuming when using the visual keyword method or stacked SVMs. To save on time, runs which very likely will not improve the system's maximum improvement can be prematurely aborted. This can be safely done by estimating the chance that the new sampled mean given its variance will exceed the current best performance. There are already a number of statistical methods readily available for this purpose. The one used in the grid search was Student's t-test.

Per each run added to the sample mean, Student's t-test (Fisher, 1925) is performed for unequal sample sizes and unequal variance. The t-value $t_{c,b}$ for the current sample c and the best sample so far b is as follows:

$$t_{c,b} = \frac{\overline{X}_c - \overline{X}_b}{S_{\overline{X}_c\overline{X}_b} \cdot \sqrt{\frac{1}{n_c} + \frac{1}{n_b}}} \tag{2.25}$$

Where the pooled variance $S_{\overline{X}_c\overline{X}_b}$ is defined as:

$$S_{\overline{X}_c\overline{X}_b} = \sqrt{\frac{(n_c - 1)S^2_{X_c} + (n_b - 1)S^2_{X_b}}{n_c + n_b - 2}} \tag{2.26}$$

The degrees of freedom can be defined using the Welch-Satterthwaite equation (Satterthwaite, 1946) as follows:

$$DOF = \frac{(s_c^2/n_c + s_b^2/n_b)^2}{(s_c^2/n_c)^2/(n_c-1) + (s_b^2/n_b)^2/(n_b-1)} \tag{2.27}$$

The chance of the current sample to exceed the best previous sample $P(\overline{X}_c > \overline{X_2})$ is $P(T(DOF) < t_{c,b})$. In the grid search, finding the performance of a current parameter setup is abandoned when this chance falls below 0.25.

A flaw of this approach is that the standard deviation σ is in itself also a stochastic variable. For this reason, the sample is first filled up with at least 5 examples to assure some accuracy of the standard deviation.

This improvement in optimization time is promising. Optimizations that had earlier taken 42.3 minutes to complete with the standard grid search can now be completed in 12.4 minutes.

2.3 Bags of Visual Keywords

The bags of Visual Keywords approach is widely popular and has been tested to work with impressive performance on image categorization tasks (Csurka et al., 2004). It works as follows: Patches of a certain constant size are first extracted from an image database. Local image features, like any of the features to be described in chapter 3, are extracted from these patches. These patches are then clustered by a clustering algorithm, creating a codebook. In the case of this project, this will be done using the *k-means clustering algorithm*. This codebook will then be used to describe the images in the dataset, using bags of visual keywords. There are many ways to express a given image in terms of this codebook (Nowak et al., 2006). In this project the Hard Bag of Features is described and the Minimum Distance map by Abdullah et al. (2010) is used.

2.3.1 K-Means Clustering

The *k-means clustering* method partitions a set of datapoints into clusters. The number of clusters k needs to be defined up front. These clusters are represented by cluster centers μ, vectors with the same size as a datapoint x. Initially each cluster center μ_i is set to the value of a randomly selected datapoint. After initializing the cluster centers, two steps are alternated: the *assignment step* and the *update step*.

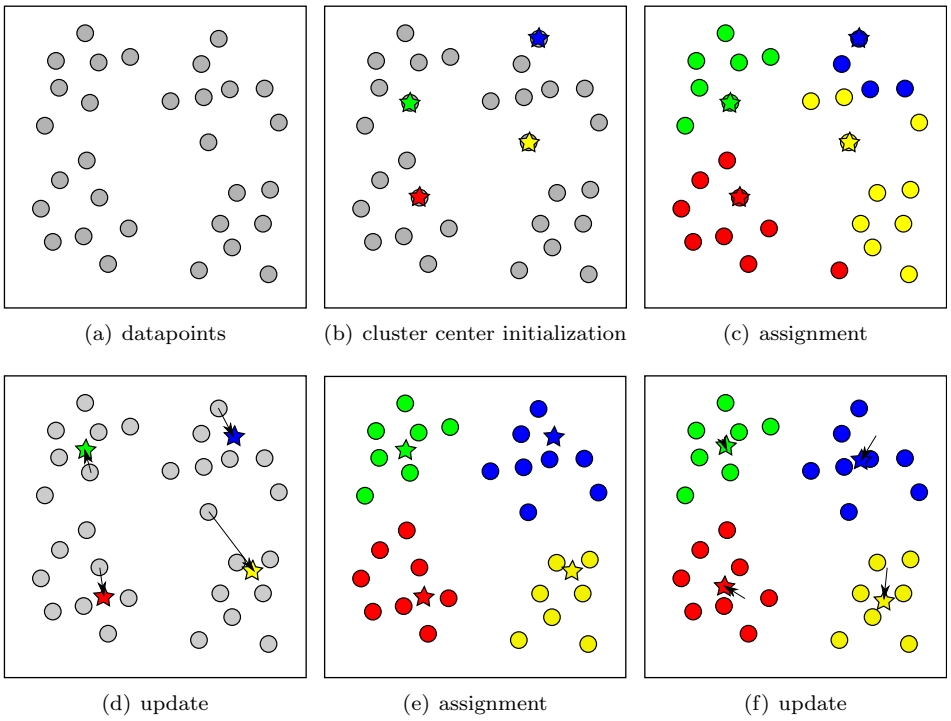
In the assignment step each datapoint is assigned to its closest cluster center, dividing our dataset into k clusters. Which cluster center is closest is defined by some distance measure, which in our case is the Euclidean distance measure, described in equation 2.1.

The cluster centers are then recalculated in the update step. The new cluster centers are defined by the mean of all the points assigned to that cluster center in the assignment step, hence the name k-means.

These two steps are repeated until the cluster centers no longer change. The entire process is visualized for a 2d feature-space in figure 2.4.

2.3.2 Sliding Window

To create a visual keyword histogram of an image, a sliding window is passed over that image. This is simply a region that is moved in constant increments



(a) datapoints (b) cluster center initialization (c) assignment

(d) update (e) assignment (f) update

Figure 2.4: The *k-means* algorithm clusters a set of datapoints (a) in a certain feature space, in these figures a 2d feature-space. k cluster centers are chosen from the datapoints randomly (b), represented by stars. Each datapoint is then assigned to its closest cluster center in an assignment step (c and e). The means of the datapoints assigned to a cluster center then define the new cluster centers in the update step (d and f). The assignment and update steps are repeated until they converge.

through the whole body of the image. Each increment, the local region selected by the window is analysed by a certain visual keyword descriptor scheme. This scheme influences the visual keyword histogram for each region the window passes, summarizing the image in terms of the codebook. This histogram is the eventual descriptor vector and its size is the same size as the codebook. It has a value for each word in the codebook. In section 2.3.3 and 2.3.4 the two visual keyword descriptor schemes used in this project are discussed.

2.3.3 Hard Bag of Features

The Hard Bag of Features visual keyword scheme (Nowak et al., 2006) is a simple winner-takes-all scheme and is widely used. A given patch is assigned to the single closest mean in the codebook. The histogram value of this codebook entry is thus incremented by one for each image patch that finds itself closest to that center. This is shown more explicitly in equation 2.28, where $HBOF(w)$ is the value of the bag of words descriptor for word w, c is the cluster-center in the codebook that corresponds with word w, n is the number of patches extracted from the image and r_i is the local feature vector of the ith patch.

$$HBOF(w) = \sum_{i=1}^{n} \begin{cases} 1 & \textbf{if} \quad w = argmin_c(dist(c, r_i)) \\ 0 & otherwise \end{cases} \tag{2.28}$$

This scheme has been implemented, but is not used in the results section, because of time restraints. The Minimum Distance map scheme provided significantly better results and was preferred over the Hard Bag of Features Scheme.

2.3.4 Minimum Distance map

The minimum distance map, introduced by Abdullah et al. (2010), is also a visual keyword descriptor scheme. The value for each word c in vocabulary V, is defined by the minimum distance of a patch in the image to that given word, as shown in equation 2.29.

$$MIN(w) = min_r(dist(c, r)) \tag{2.29}$$

where $MIN(w)$ is the minimum distance map value for word w, c is the cluster-center corresponding with word w and r is a description of a given patch.

2.4 Feature Combination

As described in previous sections, our images are described by feature vectors. These feature vectors are used to classify the datapoints. How these feature vectors are constructed and what they describe is further discussed in section 3. There are different features that describe different properties with each their own expertise and performance. It has been shown that combining several features for a single classification can significantly increase performance. This can be done in different ways, two of which will be implemented in this project:

Firstly there is the *naive approach*. In this approach features vectors are simply combined into a single large feature vector with which to train. This

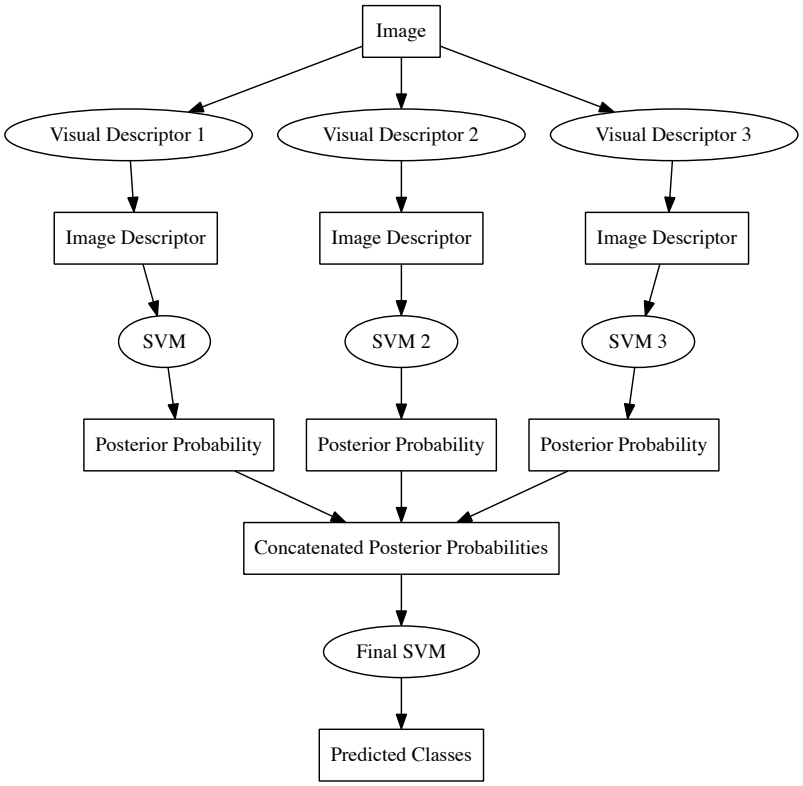


Figure 2.5: 3 visual descriptors are combined by stacking SVMs. An SVM is trained for each image description. The posterior probabilities of the SVMs are calculated and concatenated into a single vector. This vector is used to train the final SVM, which eventually provides class prediction.

method is effective, but it causes feature vectors to become very long, thus training classifiers becomes more time consuming.

Another approach is stacking SVMs. This is done by training an SVM on each feature. The predicted posterior probabilities for each class predicted by each SVM are then taken and combined into a single vector. This vector is then used by a final SVM, which is provided the eventual class prediction. A flowchart of this method is depicted in figure 2.5.

2.5 Information Retrieval

The methods described will not be strictly used in a traditional multiclass classification fashion. They may also be used for information retrieval, or in this case image retrieval. The distinction here is that not all datapoint are required to be classified. Kalooga might wish to extract a certain kind of image from a

given set of images, either to filter an unwanted image type out, or to select an image type that is particularly interesting to Kalooga. This is achieved in this project with SVMs by using the SVM's posterior probability output to decide whether to classify a datapoint or not. Given an SVM's posterior probability $P_j^P(x)$ for datapoint x as class j and a cutoff value c , it will only be classified as class j, when the following is true:

$$P_j^P(x) > c \tag{2.30}$$

Otherwise the value will be left unclassified. This way, classification can be expected to be more accurate while sacrificing the total number of images the system will classify.

When dealing with information retrieval problems, the performance is measured in *precision*, *recall* and what is called an *F-measure*, (Davis and Goadrich, 2006). Precision, is defined as:

$$precision = \frac{nr\ correctly\ retrieved\ relevant\ images}{all\ retrieved\ images} \tag{2.31}$$

While recall is defined as:

$$recall = \frac{nr\ correctly\ retrieved\ relevant\ images}{all\ relevant\ images} \tag{2.32}$$

The F-measure is a score combining precision and recall in a single value. The F_β-value does the same, but allowing changing the importance of precision and recall through β. When β is 1, the value of F_β will be equal to the regular F value. Values for β lower than 1 will place more value on the precision, for instance value of 0.5 for β will place twice the importance on precision. Values for β higher than 1 will place more value on the recall, for instance a value of 2 for the β will place twice the importance on recall. The F_β-measure is defined as:

$$F_\beta = \left(1 + \beta^2\right) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{2.33}$$

Chapter 3

Image Descriptors

In this chapter the algorithms used in this project that extract a visual description from images are discussed. To classify an image using any of the methods described in chapter 2, a description of the image is required. In the case of image classification, ideally, this is a visual descriptor that best distinguishes between the available image classes. The role of visual descriptors to the image classification system has also been represented in the pipeline, shown in figure 1.1 of page 6. A visual descriptor comprises all the information about the images the system is able to use in later processes and is crucial to the eventual performance of the system.

When considering the problem naively, one could consider to simply use the entire image. The pixels contained within the image themselves would function as the description. This, however, proves both infeasible and inefficient. The reasons for this illustrate the requirements of a good visual descriptor, so the requirements of a good visual descriptor will be explained alongside these reasons.

First of all, using the proposed method, images of different sizes would yield descriptions of different lengths. Larger images would contain more pixels, and thus longer descriptions, while smaller images would have shorter descriptions. However, many eligible machine learning algorithms compare datapoints by considering the descriptions to be points in a certain space: feature-space. The length of the extracted feature dictates the dimensionality of that feature-space. A variable feature length will thus disallow comparison of the datapoints, as each datapoint will be expressed in a space with a different dimensionality. Thus, a possible requirement of a visual descriptor is to define a description of any image within the same feature-space, ensuring the same length.

This is often achieved by creating histograms. Many histograms are made by dividing the range of a certain image property into a number of bins and summing the total intensity of that property within each subrange. This is the case for color histograms. Combining several histograms calculated from subregions of an image is a good way to incorporate the spatial aspect as well. This is the case for histograms of gradients (HoGs).

A descriptor including the color values of each single pixel would also become very large, causing two problems. Firstly, it would make any application using the descriptor very resource and time heavy. This may cause a descriptor to become unmanageable when using more computationally complex machine

learning algorithms such as SVMs.

Secondly, the higher dimensionality of a feature-space will make it harder for any machine learning algorithm to learn from a training set. A visual descriptor should summarize a visual property of an image in such a way that retains the discriminative power of the visual property it describes but minimizes the length and complexity of the description itself.

The qualities and features of images that a visual descriptor uses to distinguish images from each other can be almost anything. The final goal of an image feature, in the given case, is to provide a basis to make high level semantic distinctions between images. The image features that can be extracted from just the image information itself can only rely on the low-level information in the image. An image feature helps in bridging the gap between the low-level image information and high-level semantic information. Common image descriptors focus on describing an image's color, shape or texture (Liu et al., 2007). In this chapter the visual descriptors used in this project will be explained in terms of what they describe and how they describe it.

3.1 Color Histogram

The color histogram is a simple, widely used image descriptor. It describes the color distribution within the image. Color histograms are often used in scene recognition (Oliva and Torralba, 2006; Vailaya et al., 2001) and image indexing (Deselaers et al., 2008; Datta et al., 2008).

As with any histogram, a certain space, in this case the color space, is divided into a number of bins. Each pixel within the image is assigned to a bin by its color value, which then serves as a description. As this image descriptor was developed as a baseline and a placeholder in the pipeline, it has not been optimized for performance. The color space used is HSV (Hue, Saturation and Value), binning only for differences in hue and value. 10 bins are used for each of both components, and the HSV-space is divided linearly leading to $10 \times 10 = 100$ color bins, $c_1 \dots c_{100}$, in total, which is also the length of the feature vector. This allows us to define the histogram as follows:

$$H_I(c_i) = \frac{\sum_x \sum_y L(I(x,y), c_i)}{|I|} \tag{3.1}$$

where $H_I(c_i)$ is the histogram value of image I for the bin of color c_i defined by hue and value, $|I|$ is the number of pixels in image I, $I(x,y)$ is the color from c_1 to c_{100} that the pixel at x, y is closest to and $L(a,b)$ is a function which is 1 when $a = b$ and 0 if not.

As described here the color histogram is useful as it is invariant to any spatial transformation of the image. However, the distinguishing power of the color histogram in detecting objects and scenes is very limited. Often categories and scenes are too variable to separate them using color alone.

Novak and Shafer (1992) show however that modeling histograms may lead to useful description of the image representing surface roughness and geometry.

3.2 Histograms of Gradients

The *Scale Invariant Feature Transform*, or SIFT (Lowe, 1999, 2004), is a very successful and popular computer vision algorithm designed to detect and describe important features in an image. Points of interest within the image, which are called key-points, are located. These are points that often occur in high energy places within the images and are therefore more recognizable. After locating them, the keypoints are described by histograms of the orientation and magnitude. This is done in such a manner that found keypoints can be compared with those of other images.

In this project an image descriptor is used that is similar to the descriptor created by the SIFT algorithm to describe keypoints. The descriptor is not used to describe keypoints however, but patches from the image extracted in a regular grid. This way the image is described in a uniform way, encompassing the entire image and not just points of interest. More importantly, this ensures a constant length of the feature vector, which is required by the classifying algorithms described in the previous chapter. This promotes changing the descriptor in several ways that are of more use in this setup than when using it for keypoint comparison. The changes to the descriptor and the difference in use to SIFT are such that the name SIFT is no longer appropriate. For this reason this descriptor will be referred to as Histograms of Gradients (HOG).

3.2.1 Original Sift Descriptor

The original sift descriptor is extracted as follows. After a keypoint has been designated it is rotated to achieve rotation invariance of the descriptor. An area of 16×16 pixels around the keypoint is then selected. A gradient of this area is created by calculating the magnitude and orientation for each pixel. This is done by using the pixels adjacent to that particular pixel:

$$I_x(x, y) = I(x + 1, y) - I(x - 1, y) \tag{3.2}$$

$$I_y(x, y) = I(x, y + 1) - I(x, y - 1) \tag{3.3}$$

$$M(x, y) = \sqrt{I_x(x, y)^2 + I_y(x, y)^2} \tag{3.4}$$

$$\Theta(x, y) = tan^{-1} \frac{I_x(x, y)}{I_y(x, y)} \tag{3.5}$$

where $I_x(x, y)$ and $I_y(x, y)$ are the horizontal and vertical component of the gradient belonging to pixel $< x, y >$ in horizontal and vertical directions respectively and $M(x, y)$ and $\Theta(x, y)$ are the magnitude and orientation at that pixel. This gradient is visualized in figure 3.1(b).

These orientations and magnitudes are then used to create a set of 4 by 4 orientation histograms. Each histogram's bins the 16 orientations and magnitudes of the 4 by 4 patch into 8 equally distributed bins. The final description becomes the vector 8 bin values of each of the 16 histograms, making for a total of 128 values.

(a) original (b) gradient (c) HoG

Figure 3.1: A visualisation of an image (a), its gradient and the HoG description computed from that gradient. The image gradient visualisation (b) is white at high magnitude areas emphasizing edges and noisy areas. Along with these magnitudes, an orientation for each pixel is calculated, not visualized in this image but it is visualized indirectly in the HoG descriptor visualisation (c). To calculate the HoG description, the image gradient is divided regularly into a number of rectangular regions. In this example the image was divided into $4 \times 4 = 16$ areas for the sake of visibility, for the actual description however it is divided into $8 \times 8 = 64$ segments. In (c) these histograms are visualized by a set of vectors for each histogram. Every region has 16 vectors, each representing a bin of the histogram, the length of that bin representing the magnitude in that direction within the area.

3.2.2 Changes to Sift Descriptor

In this section the changes made to the original sift descriptor are described. There are two in total: a phase shift for the orientation bins and the use of diagonal gradients when calculating the orientation of a pixel.

The descriptor used in this project is different to the original SIFT descriptor in a number of ways. First of all there are no keypoints, instead the entire image is divided into a number of segments. The height and width of the image are evenly divided among the number of segments creating rectangular patches with no overlap. As the magnitude and orientation of almost each pixel will be used eventually, the gradient is precomputed. A histogram of each of these segments is then extracted and combined into the final description.

The number of bins is parameterized, any number of bin directions can be chosen instead of just the 8. Fewer may be used for a smaller descriptor size and more may be used to achieve a higher resolution. In image 3.1 a visualization is shown of the image gradient and the HoG feature extracted from an image.

Phase Shift

A more important change to the original sift keypoint descriptor is a phase shift of the bins distribution. The original sift descriptor has orientation bins that split the range of orientations at all of the orthogonal and diagonal angles. To alleviate this, the bins can be shifted half the length of a single bin as shown in figure 3.2.

Many shapes in nature, architecture and design place special emphasis on

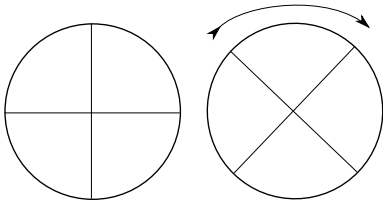


Figure 3.2: An illustration of the phase-shift on histogram bin alignment. The left image shows a 4 bin histogram in the standard alignment, each bin centered on a diagonal angle. On the right, the same bins are shown but shifted in phase such that the bins are now aligned to be centered around the horizontal and vertical angles.

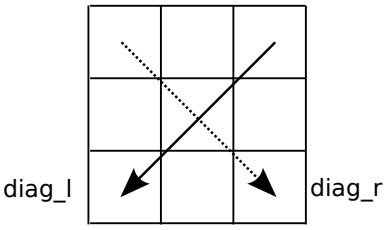


Figure 3.3: The two diagonal vectors added to infer pixel magnitude and orientation.

the 4 primary directions: up, down, left and right. Think of horizons, windows, roads and houses for example. Using phase shift, the orientation bins may more correctly match the these dominant directions. This could create a more accurate summary of the gradient and make the final feature vector more descriptive.

Use of Diagonals in Gradients

Another change made is the way in which the gradient is made. In the original descriptor the image's gradient is computed by deducing horizontal and vertical components. These horizontal and vertical components are then used to deduce the orientation. However it could be considered that the pixels diagonal to the center pixel are just as representative of the orientation and magnitude of the edge on that pixel.

In the modified SIFT descriptor, the diagonal components $I_{diag\_r}(x, y)$ and $I_{diag\_l}(x, y)$, as shown in figure 3.3 are calculated. These are derived from the pixel's diagonal to the center pixels as shown in equations 3.6 and 3.7. The value of the diagonals are then translated into the simple x and y components as shown in equations 3.8 and 3.9

$$I_{diag\_r}(x, y) = I(x + 1, y + 1) - I(x - 1, y - 1) \tag{3.6}$$
$$I_{diag\_l}(x, y) = I(x - 1, y + 1) - I(x + 1, y - 1) \tag{3.7}$$

$$I_x(x,y) = \frac{1}{2}I(x+1,y) - \frac{1}{2}I(x-1,y) + \frac{1}{4}I_{diag\_r}(x,y) - \frac{1}{4}I_{diag\_l}(x,y) \quad (3.8)$$

$$I_y(x,y) = \frac{1}{2}I(x,y+1) - \frac{1}{2}I(x,y-1) + \frac{1}{4}I_{diag\_r}(x,y) + \frac{1}{4}I_{diag\_l}(x,y) \quad (3.9)$$

This way, all the bordering pixels are accounted for without having to deal with angles.

3.3 Mpeg7 Features

MPEG-7 is a standard for the description of images (Manjunath, 2002). It defines a number of descriptions based on the actual content of the images itself. The standard was designed to facilitate searching within the content of the images. A set of four commonly used descriptors contained within the MPEG-7 standard are used in this project.

3.3.1 Edge Histograms

The Edge Histograms descriptor is a popular textural MPEG-7 feature.

To create this feature, the image is divided into 4x4 subimages (Won et al., 2002; Lux et al., 2003). Of these 16 subimages a histogram is created summarizing the occurrence of 5 types of edges in a histogram with a total of 5 bins. These edge-types can be vertical edges, horizontal edges, either of two opposite diagonal edge and a non-directional edge class. Each histogram is created by dividing the subimage into a number of yet smaller subimages. If an edge is present in the subimage, the edge is classified as one of the 5 edge types incrementing the corresponding bin value by one. If the subimage contains no edge, none of the bins is incremented. Finally the bins are normalized, leaving a compound histogram descriptor of $4 \times 4 \times 5 = 80$ values.

3.3.2 Scalable Color

The Scalable Color Descriptor (SCD) creates a description of the color content of an image extracted in the HSV color-space (hue, saturation and value) (Manjunath et al., 2001). The three-dimensional HSV space is uniformly divided into 256 bins, up out of 16 for the hue dimension, 4 for saturation and another 4 for value. This descriptor owes its name to the fact that it can equally well represent images of both high and low resolutions.

3.3.3 Color Layout

The Color Layout Descriptor (CLD) gives a more spatial representation of the color distribution of an image (Manjunath et al., 2001). This is done by first dividing the image into $8 \times 8 = 64$ parts. These parts are then described by the average color present in that part. The color-space in which this average is found is YC_bC_r. In this space Y, or luma, is a measure of the luminance. C_b and C_r are the blue and red differences from the chromatic center respectively.

3.3.4 Dominant Color

The Dominant Color Descriptor or (DCD) gives a distribution of the most salient colors found in an image, much like the Color Layout feature does. The dominant color descriptor creates a flexible feature, having a larger feature size as there are more dominant colors. For this project however, description length is required to be constant, and so up to 8 dominant colors are given. If the feature turns out to be smaller than 8, the remaining values are set to be 0.

Chapter 4

Implementation

In this chapter, the implementation of the methods described in the previous chapters is discussed. This includes the choice of programming language, the software libraries used and the way parameters, descriptors and datasets are managed. Furthermore, this chapter discusses a Computer Vision Tool that was made to facilitate the use of these methods and the motivation behind its creation.

4.1 Programming Language and Approach

The software that was written especially for this project was written in the STL C++ programming language (Musser and Saini, 1995) and compiled using the GCC 4.2.1. compiler (Gough and Stallman, 2004). This language was chosen as it would ease the use of the many libraries involved in image processing and feature extraction. It also provides a fast performance which is vital to the processing of experiments. The processing time required for these experiments can become very large when constantly recalculating descriptors or when optimizing SVM parameters. This is most important for the clustering and SVM stacking experiments, especially considering some experiments will have to be repeated for different parameter settings. The required processing time was also reduced by making use of threading while extracting features.

The program design follows an Object Oriented approach. Classifiers and Features are described generically and are easily accessible for customization of existing classifiers and features as well as the insertion of new objects. This not only facilitates adding new features or classifiers, but also using these new objects when clustering and stacking classifiers. The Object Oriented approach has also aided greatly in creating the vision tool discussed in section 4.3 and the graphical user interface it provides.

4.2 Libraries

A large range of libraries was used for image handling, the extraction of image features and classification. Each library that was used can be run and compiled on most popular platforms, making the application portable.

First of all the *ImageMagick* Library (Still, 2005) was used for most image conversion and handling. It provides access to most popular image formats, conversions to different color representations and access to low level pixel information. Its image writing functions were also used to create simple visualizations of image descriptors and clustering results, two of which are available in the tool.

For threading the Boost thread library was used (Demming and Duffy, 2010). Another boost library that was used is the boost filesystem library, giving universal access to filesystems of different operating systems.

The *OpenCV* Library (Bradski and Kaehler, 2008) was also used for this purpose, as well as some classification and face recognition provided by the library. To reduce memory usage, it has been made possible to access the same imagedata as ImageMagick dataobject and the OpenCV dataobject without the need to copy it.

For the Support Vector Machines, the *libsvm* library (Chang and Lin, 2011) was used. Java code for the MPEG7 features was at first provided by code from *Emir and Caliph* (Lux et al., 2003). This Java code had to be accessed externally by the main application whenever MPEG7 descriptors were required. Later however, it was replaced by a C++ code provided by Bastan et al. (2010).

4.3 Computer Vision Tool

As discussed in the introduction, the task of finding high-level semantic images from low-level pixel data is a hard problem to solve. There is currently no complete solution for it yet. This is especially true for the broad range of pictures Kalooga deals with, and the labels that are popular in queries. First of all, there is a very large number of different labels. The greater the number of distinctions to be made, the harder the task becomes. Secondly, the most interesting and useful of these labels are often high-level and semantic in nature. Labels may refer to the picture's abstract qualities or the names of people and locations. Some labels are even hard for humans to discern from the image alone and often require some knowledge of the domain.

These limitations make it unfeasible to find every correct label for any image automatically. However, methods that use only low-level data are not necessarily without merit for parties like Kalooga. The problem becomes easier to solve by making the number of labels the system concerns itself with smaller and more manageable. This way, the problem becomes more of a categorization task, rather than an identification task. Groups of images can be subdivided into a relatively small number of meaningful groups of which the semantic distinction is still useful.

Another way to make use of these methods is as tools to solve more specific problems. Information within the image could for instance be used to filter out unwanted images: product images, banners and logos. Given a particular subset of images sharing a property one can also subdivide the images further. This way existing categories could be subdivided into more visual categories: out of a set of tourism photos, photos containing landscape, nature, culture and architecture could be grouped.

To supply a way to quickly arrange and test an image classification pipeline, a Vision Tool was written. The goal of this tool is to be able to quickly make an

estimation of the performance of a feature and classifier on any dataset. This way one could easily find out to what extent the visual information of an image can be used to solve a problem.

4.3.1 Graphical User Interface

The vision tool has a graphical user interface (GUI) which allows for easy access to its functionality. *OpenGL* (Shreiner et al., 2005) was used to display the images in conjunction with the *GLUI* (Rademacher, 2006) graphical user interface library for the buttons. The layout of the interface is shown in figure 4.1.

The GUI allows users to quickly load any custom dataset or images. The dataset may be explored, showing each image on screen on its own or several at once. After the images are loaded, categories may be selected at will to be included while training and testing. The user may select and modify which descriptors to extract and use for classification. Each of the parameters used when extracting the descriptor may be altered.

The user may also view an image through one of the visualizations available. These include the image gradient used to create a sift descriptor or a visualization of the sift descriptor itself.

The GUI allows its user to extract features, train a KNN classifier or an SVM on them and test the results. K-fold cross validation may also be used to test the system for a more accurate prediction of performance. The results are displayed as statistics, giving the performance and a breakdown of how many instances of which class were classified correctly. The user can view the classification results, showing the images per class, when classifying a test set or when performing cross validation.

The GUI also allows the user control over clustering. The user may create a clustering descriptor by selecting any descriptor setup and the preferred clustering parameters. After clustering is completed, the corresponding bag-of-words histogram may be calculated using the found cluster centers. This means of describing an image then becomes available for use like any of the other features. The cluster centers may also be saved and loaded for later use. A visualization is also available for the cluster descriptor, showing for each patch within an image to which cluster it belongs by number and color code.

4.3.2 Datasets

The Vision Tool provides an environment in which any dataset can be loaded. A dataset can be any directory which contains one or more directories containing images of any popular image format (png, jpg, gif). The images can be browsed visually, and each category may be turned on or off for use in training and testing.

4.3.3 Parameters and Descriptors

Parameters are stored using xml files. Within these files, all parameters that are needed for feature extraction and classification are stored.

When using a descriptor a hashtag is made for that particular kind of descriptor. This hash tag is based on every setting relevant to the descriptor itself. When calculated, the descriptor outputs are stored in binary files labeled for

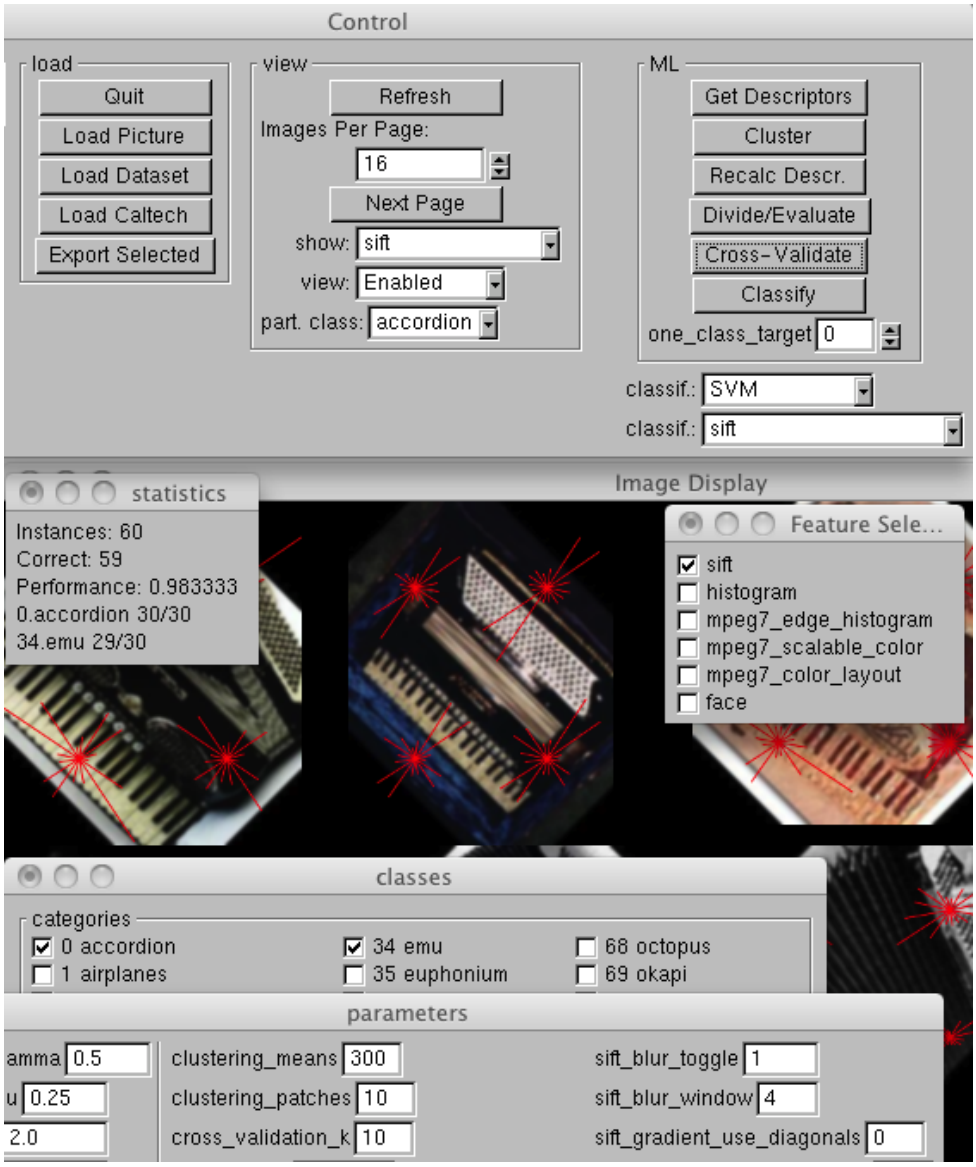


Figure 4.1: The Computer Vision Tool allows the user to load his own single images, image datasets or scientific image datasets. The loaded images are then displayed in the image-window, and can be selected in a class selection window. The user may freely browse the database, extract any of the available low-level visual features, display available visualizations of these features, specify classification methods to be used when training, change any available parameter in the parameter-window, train on, classify or cross-validate on sets of classes, directly evaluate test results statistics and create a hard bag of words feature extractor using the k-means clustering algorithm which in turn can be used as a low-level image feature.

filename and descriptor hash. The next time a descriptor needs to be calculated with the same filename and descriptor hash, the stored descriptor output is used to save processing time.

Combining features is defined by selecting a combination method and the parameter files of the features to be combined. As of yet feature-combination through naive concatenation or SVM stacking is not supported by the GUI. However, these methods can be accessed through the use of arguments before starting the application.

4.3.4 Using the Vision Tool

There are six basic commands to give the application as arguments which correspond to the three basic modes of the application: the GUI, performing an experiment, creating a Visual Keyword Histogram, and classifying or estimating the classes of new images. The syntax of these is explained in the following:

```
Start Graphical user interface:
  ./machineVisionTool gui

Start an exeriment:
  ./machineVisionTool experiment {nn,svm} DATASET NR_REPETITIONS

Create a Bag of Words feature by kmeans clustering:
  ./machineVisionTool cluster DATASET OUTPUT_FILE

Optimize SVM parameters:
  ./machineVisionTool optimize [OUTPUT_XML]

Classify new datapoints:
  ./machineVisionTool classify NEW_IMAGES

Estimate class probability
  ./machineVisionTool estimate NEW_IMAGES

DATASET can be any directory containing subdirectories with images
to be used as dataset, each subdirectory constituting a catagory and
the images within belonging to that catagory.
You can also choose one of the following predefined datasets:

  abdullah2010:         Dataset used by Abdullah et. al. (2010)
  kalooga:              A Dataset composed of images from the web
                        crawled by Kalooga about soccor.
  actionkalooga:        The same Kalooga dataset, only reorganised
                        to be a 2-class problem: Action vs. Rest

Additional Datasets may be added by editing datasets/index.xml
```

Along with the command a number of options may be given. Different sets of parameters may be used or a previously created visual keyword histogram feature can be included for use as feature. The syntax of these is as follows:

```
-p PARAMETER_FILE       ; use parameter file
-c CLUSTER_ROOT         ; include cluster histogram feature
                          defined by CLUSTER_ROOT.xml and
                          CLUSTER_ROOT.clustercenters
-C CLUSTER_ROOT         ; same as -c but also activates the feature
-d DATASET              ; use DATASET as the preferred dataset in
                          every task
-naive FEATURES_ROOT    ; concatenate the parameters files contained
                          in the folder FEATURES_ROOT to create a new
                          feature.
-svmstack FEATURES_ROOT ; Combine the parameter files contained
                          in the folder FEATURES_ROOT to create an
                          svm_stack ensemble.
```

4.3.5 Discussion

The vision tool described in this chapter provides an entire image classification pipeline. It allows this pipeline to be quickly modified and tested. It gives access to a range of image descriptors, describing color and texture as well as shape.

This makes it easy to predict the efficiency of a certain classification pipeline on that particular image domain. There are user friendly applications that provide easy access to machine learning methods or computer vision methods. However no applications are known to us that focus on image categorization and image features and provide an entire modifiable pipeline. The tool can currently run on Linux and Mac systems and may easily be ported to a Windows operating system. It provides a good environment to be used in the development of image descriptors.

During the development of the tool, the use of libraries quickly became tedious. Both the ImageMagick library and the OpenCV library have been used for the handling and manipulation of images. The former was used initially, and the latter was added later. However, both libraries are not necessarily needed. In future work the ImageMagick Library could be fully replaced by the OpenCV library, which would reduce the number of dependencies and the tool's portability.

The inherent architecture of the system also imposes restrictions on the algorithms that may be implemented. The system can be expanded with features and classifiers that adhere to a classical input-output layout. A feature may only use an image's visual information, and a classifier simply receives labeled feature vectors. Any more involved methods, making a complex and specialized use of the information available would be incompatible with the current system.

Chapter 5

Experiments and Results

This chapter describes the experiments that were designed to test the performance of the methods described in chapters 2 and 3 and their results. The experiments were designed for two purposes. Firstly, to test the performance of the implemented methods and the changes made to the HoG descriptor. These methods include the color histogram descriptor, the HoG descriptor, MPEG7 descriptors, visual keyword descriptors, classifier combination and stacking methods. Secondly, to test the system's performance on a specialized Kalooga dataset.

This chapter begins by discussing both datasets. After this, each experiment will be described along with its results and a relevant discussion.

5.1 Datasets

Two image-datasets were used for the experiments. The first is the *Caltech101* dataset, a dataset intended for research in Computer Vision. The second dataset, the *Kalooga Dataset* was compiled from web-images crawled by Kalooga.

5.1.1 Caltech101

The Caltech dataset (Fei-Fei et al., 2004) is well-known and often used in testing object recognition and categorization algorithms. It contains 101 categories, each category containing from 40 to 800 images. Each image is roughly 300×200 pixels large. There are annotations provided which will not be used, only the image category will be used in training.

To yield results comparable to other studies a category selection is used identical to the one used by Abdullah et al. (2010). In this setup 10 classes of the Caltech dataset are used, namely: airplane, camera, car side, cellphone, cup, helicopter, Motorbike, scissors, umbrella and watch, shown in figure 5.1. In most experimental setups that use this dataset, 30 images will be used per test: 15 training images and 15 test images. These 30 images are selected randomly for each test, from a larger pool of images with varying number of images for each category.

In figure 5.2 the Caltech images from the 10 named classes have been blended together per class. This gives insight into the nature of these images and any

Figure 5.1: 10 of the 101 classes from the caltech101 dataset used by Abdullah et al. (2010): airplane, camera, car side, cellphone, cup, helicopter, motorbike, scissors, umbrella and watch.

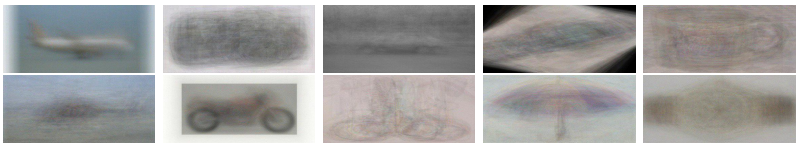


Figure 5.2: Composite images for each of the classes from the caltech101 used by Abdullah et al. (2010): airplane, camera, car side, cellphone, cup, helicopter, motorbike, scissors, umbrella and watch.

recurring shapes that are present in each class. For each class in the Caltech101 dataset a shape becomes clear that is particular to that class. Especially the classes airplanes and motorbikes show a distinctive shape. Some classes also give an indication of colors that are likely to occur for that class, such as the blue sky in the airplanes class. These images help somwhat in understanding the difficulty of classifying a dataset and which descriptors may be useful.

Caltech256

One experiment required classes with large datasets, however, Caltech101 only contains a few classes with a great enough number for this purpose. Caltech 256 (Griffin et al., 2007) has more images per class, and thus 2 classes from the Caltech256 dataset were used in that particular experiment.

5.1.2 Kalooga Dataset

Many of the publishers Kalooga serves its visualisation widgets to are specialized in soccer. Kalooga provides these soccer news sites with relevant images to go with the articles they post. This is done by automatically generating queries from the articles to search their own gallery database. A set of images is then chosen for display, sometimes automatically but often handpicked to assure quality. To get the most attention and the best impression, it is important for these images to be visually pleasing while still relevant to the subject at the same time.

Figure 5.3: 5 Images of every Category from the Kalooga-soccer dataset and a composite image for each class as given for the Caltech101 dataset in figure 5.2.

A way to classify or otherwise organize images which are already known to concern soccer could be very useful to Kalooga. Studying the performance of the system on this particular domain will gauge the practical applicability of the system. For this purpose a dataset was composed out of images from the Kalooga database concerning soccer. The images were collected from various soccer-related queries, and ordered into categories. The categories into which these images have been separated have not been chosen to assure a high performance. Rather, the categories are meant to be useful in selecting images for display in Kalooga's widgets.

Figure 5.3 shows 5 images for each of the classes in the dataset and a composite image as was given for the Caltech data.

This dataset will be used in two ways. One experiment will deal with the classes similarly to the Caltech101 dataset, training and classifying each datapoint with its named label. This experiment will be referred to as the *12 Kalooga classes* problem. In the second experiment it will be tested how well the system can extract the action class from the other classes by posing a 2 class problem. The first of these two classes is the action class as it is in the 12 class problem, while the second contains the images from all the other classes in the 12 class problem together. This use of the dataset will be called *ActionKalooga* henceforth.

5.2 Parameter Optimization

Each of the results shown in this chapter was obtained after optimizing the classifier's parameters for the particular feature vector being used. The SVM Classifiers were optimized for their γ and C parameters using crossvalidation. The parameter γ was found between 2^{-20} and $2^{-\frac{1}{10}}$ and C was found between 2^{-20} and 2^{20}. For the k nearest neighbors classifier, k was found between 1 and 30. The blur parameter for the HoG descriptor was also found for each particular kind of HoG descriptor.

5.3 Experimental Setup

Before training an SVM on a set of image descriptions or classifying a set of image descriptions, all descriptions are normalized to fall between -1 and 1. Doing this sets the collective minimum and maximum of the descriptor group to -1 and 1 as opposed to normalizing each descriptor separately.

Each result displayed in this chapter was obtained as a mean of 1000 runs unless otherwise stated. This may be the case as some clustering experiments become very time consuming. As each category contains at least 30 images, this allows for at least $\frac{30!}{(30-15)!} > 2 \cdot 10^{20}$ combinations of runs. Each of these runs is performed with a new random subset of the database, again randomly separated into equally large training and test sets. Thus, the surplus of images in the categories containing more images than others will still be used.

Each performance result will be reported as percentage alongside its standard deviation in percentage, in the style of '$\mu \pm \sigma$', in two decimal precision.

Visual Descriptor	SVM Perf.	KNN Perf.
Color Histogram	41.71 ± 2.88	35.28 ± 2.81
MPEG7 Dominant Color	42.82 ± 3.20	35.14 ± 2.85
MPEG7 Scalable Color	50.33 ± 3.01	39.97 ± 2.81
MPEG7 Color Layout	65.84 ± 3.17	57.71 ± 3.06
MPEG7 Edge Histogram	76.00 ± 3.05	67.49 ± 2.84

Table 5.1: Results of the Color Histogram and MPEG7 descriptors on the Caltech101 Dataset with 10 classes.

5.4 Color Histogram and MPEG7 Descriptor Experiments

In this section, the MPEG7 descriptors discussed in section 3.3 are tested as well as the color histogram described in section 3.1. Performance was tested using Support Vector Machines as well as the k-nearest neighbors algorithm, indicated as SVM Perf. and KNN Perf. respectively from here on. The results are shown in table 5.1.

5.4.1 Discussion

It seems that the color related methods have problems describing the images of the selected ten Caltech101 classes. This is to be expected as the classes are intentionally selected on shape, which can be seen in figure 5.2. Each of the classes exhibits a clear shape, but only some contain a defined color.

The best performing descriptor in this experiment was the MPEG7 Edge Histogram. This is a texture descriptor that classifies subregions of the image into 5 edge categories and stores it in a histogram. Such a descriptor is somewhat capable of describing the image's shape, which explains its higher performance compared to the other methods.

Each SVM experiment outperforms its KNN counterpart by a significant amount. This is the effect of the SVMs maximum margin, kernel, and soft margin properties over the KNN.

5.5 HoGs experiments

In these experiments the performance of the Histogram of Gradients feature, as described in section 3.2, is tested using the k-nearest neighbor algorithm and an SVM classifier. There are two changes to the descriptor: phase shift and the use of gradient diagonals, both described in section 3.2.2. The performance of the two changes to the descriptor are also tested in this section.

5.5.1 Basic Performance

The original SIFT keypoint descriptor can be extracted at different levels called *pyramid levels* (Lowe, 1999). These different SIFT pyramid levels can be achieved with the HoGs feature by creating more histograms per image. The number of

Level	Histograms	SVM Perf.	KNN Perf.
L_0	4×4	82.99 ± 2.50	78.00 ± 2.53
$L_{0.5}$	6×6	84.16 ± 2.45	79.67 ± 2.50
L_1	8×8	84.55 ± 2.49	79.04 ± 2.73
$L_{1.5}$	12×12	83.90 ± 2.67	75.53 ± 3.05
L_2	16×16	83.22 ± 3.16	74.00 ± 3.06

Table 5.2: Results of the HoG descriptor on different pyramid levels (indicated here as 'Level').

histograms per descriptor determines the detail in which an image is described and the eventual size of the descriptor, much like the SIFT pyramid level does. However it would be useful to compare the results of the HoG descriptor with the SIFT keypoint descriptors of other studies. To do so, an analogue is made for each of the SIFT descriptor pyramid levels that contains the same number of histograms and share the same descriptor length. The lowest pyramid level, referred to as L_0, contains 16 histograms, in a 4 by 4 grid, and thus a total of 128 feature values. A similar description can be made with HoGs by also extracting 16 histogram in a 4 by 4 grid. Each higher pyramid level squares the number of histograms, thus L_1 contains 64 histograms in a 8 by 8 grid and L_2 contains 256 histograms in a 16 by 16 grid.

Tests were also performed for histogram grids between these pyramid levels. These setups are 6×6 and 12×12, effectively becoming $L_{0.5}$ and $L_{1.5}$ respectively. The results of this experiment are shown in table 5.2.

Discussion

While Abdullah et al. (2010) managed to get a better classification performance on this dataset using his SIFT descriptors, these results aren't far behind and are satisfactory for the purposes of this project. This difference in performance may be due to a great number of aspects that influence the creation of the descriptors. It might be the differences between the HoG implemented in this project and the original SIFT descriptor. Gradient values in the original SIFT descriptor were weighted for their proximity to the center of the keypoint. It may also have something to do with the way in which images are blurred or descriptors are normalized.

The HoG's performance peaks at L_1, apparently giving the best level of detail for these images. At L_1 the system performs significantly better than any of the other setups, it's closest competitor being $L_{0.5}$: $P(T(1997) \leq -3.53051) = 0.00021$ This level will be used henceforth, when using just a single setup of the HoG descriptor.

A pattern that becomes clear in these results, is that the difference between KNN and SVM become smaller as the results become higher. This is due to the fact that, as the performances grow higher, increasing that performance even further becomes harder. This is intuitive, as increasing performance from 90% to 95% seems a lot more meaningful than increasing it from 60% to 65%.

In these results, the difference between SVM and KNN also becomes larger as the total number of histograms increases. This is because KNN will have a

Ori. Bins	Shift	SVM Perf.	KNN Perf.
4	no	74.50 ± 3.07	66.26 ± 2.87
4	yes	82.34 ± 3.05	72.77 ± 3.05
8	no	84.55 ± 2.49	79.04 ± 2.73
8	yes	83.85 ± 2.45	73.04 ± 3.07
16	no	86.87 ± 2.40	79.12 ± 2.90
16	yes	87.05 ± 2.25	74.44 ± 3.22

Table 5.3: Results of the HoG descriptor when altering the number of orientation bins (indicated as Ori. Bins) and applying phase shift. In all experiments, 8×8 histograms were extracted.

harder time dealing with a large feature size than an SVM will.

5.5.2 Phase Shift and Orientation Bins

In this experiment the phase shift alteration to the HoG descriptor, described in section 3.2.2, was tested. The total number of orientation bins, i.e. the number of directions a gradient is quantized into, is also varied to find out whether it influences the effect of the alteration. The number of orientation bins itself may also influence performance of the visual descriptor as a whole. The results of this experiment are shown in table 5.3.

SVM performance using phase shift with 4 orientations is clearly significantly better than when not shifting in phase. However using phase shift also causes a significantly lower performance using 8 orientations: $P\left(T(1997) \geq 6.3368\right) = 0.00$. Using 16 orientations with phase shift performs significantly higher than performance without phase shift again: $P(T(1989) \leq -1.73025) = 0.0419$, although not by a large margin. Furthermore the results show a record high for this system when using the HoG descriptor with 16 orientations while phase shifting. These results are also shown even more strongly in the KNN counterparts of these experiments.

Discussion

The most dramatic difference is found when only 4 orientation bins are used. Shifting phases in that case significantly improved the results. This is an interesting result, also because the performance remains relatively high while the description length is halved compared to the 8 histograms setup. This is probably because of the important role the angles up, down, left and right, play in photography and thus the image descriptor. Humans instinctively align cameras to assure the horizon is leveled and these orthogonal angles are found everywhere in nature, human architecture and design. For these reasons, it is very possible that a better expression of these angles in a descriptor may improve its performance.

Furthermore, when using 8 bins, the phase shift achieves a lower performance, while strangely outperforming non-phase shift again at 16 orientations. It is unclear why this is the case, but it is clear that phase shift is most beneficial to low orientation bin counts. This is to be expected as with a large number

Histograms	Shift	Desc. size	SVM Perf.	KNN Perf.
8×8	no	256	74.50 ± 3.07	66.26 ± 2.87
8×8	yes	256	82.34 ± 3.05	72.77 ± 3.05
4×4	no	64	73.80 ± 4.80	59.62 ± 3.02
4×4	yes	64	83.93 ± 4.08	75.60 ± 2.25
2×2	no	16	64.69 ± 5.01	54.22 ± 2.98
2×2	yes	16	76.09 ± 4.73	65.83 ± 2.95

Table 5.4: A followup experiment to expand on the result in table 5.3. The performance difference when using phase shift is tested for low histogram counts, from 8×8, down to 2×2, all using only 4 orientation bins.

of bins, the shift will only cause each bin to move a few radians. These experiments have proven that, especially when using few values to do it, the way the orientations are summarized is very important to the histogram of gradients.

Furthermore, using 16 orientation bins has achieved the highest performance by the system yet. This performance comes at a high price however: doubling the description length.

Small HoG Descriptors by Shifting Phase

A followup experiment was performed to find out how small descriptors could be made using 4 phase shifted orientation bins. The results are found in table 5.4. Using this alteration the performance of the descriptor remains relatively high as the number of histograms decreases. Even at 2 histograms it outperforms the unshifted descriptor at any other SIFT level.

It is also interesting to see that a description of only 16 values can still perform so well. A smaller description is useful in many ways: it can often be calculated and trained on more quickly. This may not directly be the case for the HoG algorithm, as it still has to pass every pixel in the image the same amount of times. But one could resize the image to a lower resolution and then extract the description without changing the eventual content of the descriptor too much. Small description size also helps classifying algorithms, as they will have less dimensions to deal with comparing descriptors. It also limits the storage space required, which makes storing the description for every image of an online image database less of an investment.

5.5.3 Use Of Gradient Diagonals

In this experiment the results of the HoGs descriptor were tested using the diagonal gradients as described in section 3.2.2. The results are shown in table 5.5. No significant differences were found in these experiments.

Discussion

It seems the use of gradient diagonals, does not affect the descriptor's performance. It is possible the use of gradient diagonals does improve the precision of each single value in the gradient that is later used to make the HoG descriptor.

Sift Level	Histograms	diagonals	SVM Perf.	KNN Perf.
L_0	4	no	82.99 ± 2.50	78.00 ± 2.53
L_0	4	yes	82.81 ± 2.60	78.51 ± 2.54
L_1	8	no	84.55 ± 2.49	79.04 ± 2.73
L_1	8	yes	84.69 ± 2.49	78.88 ± 2.79
L_2	16	no	83.30 ± 2.70	74.99 ± 3.01
L_2	16	yes	83.22 ± 2.74	74.87 ± 3.02

Table 5.5: Results of the experiments that test the performance difference when using diagonals when calculating the image gradient.

However, this increase in local precision either does not change: 1. the orientation bin each pixel is assigned to, or: 2. the eventual averages found for the orientation bins. Either way the description finally calculated comes out mostly the same. Thus, the alteration only serves to make the extraction process more complex without increasing performance.

5.6 Influence of Training-Set Size

In this experiment the importance of having a large amount of training data is illustrated. To do this, a dataset with more images per class was needed, as some of the Caltech101 classes only contained 39 images. Instead 2 classes from the Caltech256 were used in addition to 3 of the original Caltech101 classes, although these classes are also included in the Caltech256 dataset itself. The class names are: Binoculars, Motorbikes, T-shirt, Watch and Airplanes. The system was trained on training sets of varying sizes using the best performing HoG descriptor setup, which happened to be using 16 orientations, 8×8 histograms and phase shift. Performance was tested using the same number of test images as were used for training. The results have been plotted in figure 5.4.

5.6.1 Discussion

From the graph in figure 5.4 it is clear that dataset size is very important to the system's performance. The Caltech101 experiments were all performed on training sets of only 15 datapoints, so it is clear that the performance of the system may improve greatly with large training sets.

When applying these methods to real problems, one can expect to be able to use a great amount of datapoints per class. Especially the experiments performed in this project, which almost all use a training set size of a mere 15 datapoints. These can be improved upon by simply enlarging the amount of training data. This should always be kept in mind when evaluating the practical applicability of these methods.

Problems that Kalooga may wish to solve will deal with large amounts of online images. This abundance of information should allow for the training set to be continually enlarged by hand or perhaps even automatically. This is an important advantage, easily overlooked by solely looking at benchmarks on scientific data.

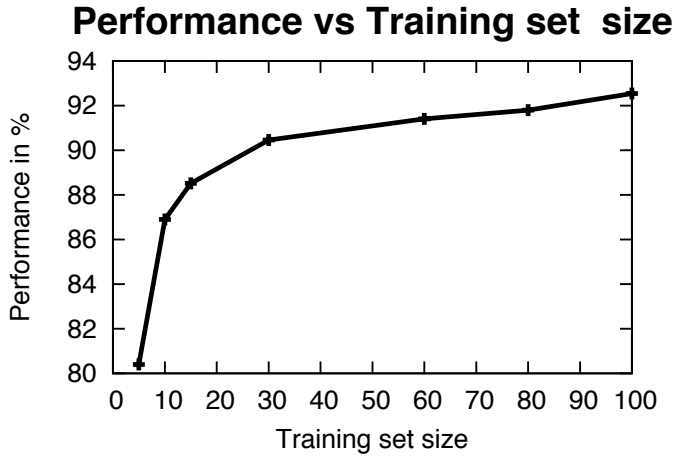


Figure 5.4: A plot of experiments ran to illustrate the importance of training set size. Each experiment was performed using 5 classes from both the Caltech101 dataset and the Caltech256 dataset, each containing over 200 datapoints. Test sets were used of identical size to those used when training.

5.7 Visual Keyword Descriptors

In this section the results that have been made using the clustering method and visual keyword descriptors described in section are described. The codebook for each test is created by converging a given number of means k using the k-means clustering algorithm. The patches used for this are either 32×32 pixels or 64×64 pixels, depending on the experiment, and are drawn randomly from the dataset itself. The eventual descriptions were found using a sliding window of the appropriate patch size, sliding 8 pixels at a time over the entire image. They were constructed using the Minimum Distance map scheme proposed by Abdullah et al. (2010). There were visual keyword descriptors created for a number of HoG setups and the best color and texture feature of the MPEG7 features: Color Layout and the Edge Histogram. The results of these experiments are shown in table 5.6.

5.7.1 Discussion

The results found in this experiment are not very competitive in nature. Abdullah et al. (2010) have achieved better results in comparable setups, going up so high as 90.7 percent. There are many reasons for this difference in performance due to the complex nature of these methods. However, optimizing parameters and testing the system in different setups has proven to cost a lot of time. Thus, finding optimal results for this particular method has become too hard to realise.

All of the experiments have also shown that the results using the visual keyword descriptors achieve poorer results than simply using the descriptors themselves. Clustering and creating visual keyword descriptors already is very time and computationally intesive, so this seems to exclude the method as it is

Patch desc.		Patch Size	Nr. Means	SVM Perf.
HoG	4×4	32	300	70.11 ± 3.17
nr. of hists:	4×4	32	600	75.12 ± 2.66
	4×4	32	700	75.61 ± 3.16
	4×4	64	700	76.52 ± 2.75
	8×8	32	700	77.16 ± 2.87
	8×8	64	700	74.31 ± 3.16
	16×16	64	700	76.51 ± 3.01
MPEG7	Edge Histogram	64	700	70.16 ± 3.23
type:	Color Layout	64	700	65.60 ± 5.06

Table 5.6: Performance results of visual keyword descriptors on the Caltech101 dataset.

combination type	SVM Perf.
Naive combination of HoG features$L_0 + L_1$	88.52 ± 3.52
Naive combination of HoG features $L_0 + L_1 + L_2$	86.43 ± 3.45
SVM stack of HoG features $L_0 + L_1$	88.64 ± 3.69
SVM Stack of HoG features $L_0 + L_1 + L_2$	89.03 ± 3.21
SVM Stack of all MPEG7 features	75.49 ± 4.42

Table 5.7: Results of the naive and SVM stacking experiments.

implemented here from practical use.

5.8 Descriptor Combination

In this section the feature combination experiments are described. These experiments are also performed on the Caltech101 dataset, with 15 training images per experiment. There are two feature combination methods to speak of: Naive combination, and SVM stacking described in section 2.4.

The former simply concatenates the image descriptions before training the classifier on them. The latter trains an SVM classifier on each of the features, and uses the probability values of each of those classifiers to train a new SVM. The results are shown in table 5.7.

5.8.1 Discussion

The SVM stack of all HoG features yields the highest performance on the Abdullah et al. (2010) dataset found in this project yet. In earlier research, higher values have been found, but these results indicate the efficiency of SVM stacking none the less and are quite good.

Naive stacking methods also yield high performance. When combining too many features, however, it seems to have problems with the descriptor size. When using all pyramid levels of the HoG descriptor for instance, it begins to perform worse where the SVM stack method would perform better.

classifier	SVM Perf.
SVM-Stack HoG $L_0 + L_1 + L_2$	53.63 ± 4.88
SVM-Stack HoG $L_0 + L_1$	53.73 ± 3.88
SVM-Stack of all MPEG7 features	46.45 ± 4.32
HoG L_1 16 orientations	53.53 ± 5.82
Visual Keyword Descriptor of HoG 4×4 phase shift	50.57 ± 4.27
Visual Keyword Descriptor of HoG 8×8 phase shift	48.68 ± 3.63
MPEG7 Edge Histogram	51.53 ± 5.81
MPEG7 Color Layout	46.17 ± 5.76
MPEG7 Scalable color	45.21 ± 5.55
MPEG7 Dominant color	29.10 ± 5.33

Table 5.8: Performance of the system as 12-class classifier on the Kalooga data.

Stacking MPEG7 features yielded a performance lower than one of its member features, the Edge Histogram. This is probably due to the low usability of the color descriptors in the classification problem, only functioning as load on the descriptor size.

5.9 Kalooga Dataset

The methods tested in the previous experiments are now tested on the Kalooga dataset, described in section 5.1.2 of this chapter. The Kalooga dataset was used in two different setups. One experimental setup resembles the use of the Caltech101 dataset: a multiclass SVM was trained to classify all datapoints as one of 12 classes. In the other setup, a 2-class SVM was trained to classify datapoints as belonging to either of two classes. The two classes are composed by combining the existing categories of the Kalooga dataset into two classes: One being the Action class as it is, the Other being the combination of all other classes. A selection of the methods tested in the previous section of this chapter were tested for their performance on these classes

Because of time restraints, not all setups have been tested for performance on the Kalooga dataset. Instead various setups were selected for testing based mostly on their performance on experiments on the Caltech101 Dataset. Some setups were also chosen as they were expected to perform differently on the Kalooga dataset than they would on the Caltech101 dataset.

5.9.1 12 Classes

In this experiment the system's performance was tested in a 12 class classification task on the Kalooga dataset. The experiments were made to resemble those on the Caltech101 dataset as training and test sets contain 15 images and experiments were repeated 1000 times. The results are displayed in table 5.8.

Discussion

Compared to the results of the 10 class subset of the Caltech101 dataset, the results are quite low, with 53.73 being the highest performance achieved. This considerable difference is probably due to the lack of inner class variability found in the 10 classes from the Caltech101 dataset used by Abdullah et al. (2010) and the great inner class variability in the Kalooga dataset.

The 10 Caltech classes share not only the same subject, but often also the scene composition, object size and orientation. This selection of images is ideal for shape based descriptors, and are easily classified by histograms of gradients.

The Kalooga dataset however is the result of many soccer related queries sorted into visually recognizable classes. Images within any of these classes often share compositions, angles, etc. but still vary greatly within their own class. This variation is mostly due to the fact that the subject of the images is not a single static object, but rather an abstract visual category.

This difference seems to make it impractical to use a multiclass classifier to classify each image in a query as a class in the Kalooga dataset similarly to what can be done with the Caltech101 dataset. This could be remedied by doing two things.

Firstly the number of classes could be reduced, by merging some of the existing classes. This would simply make the problem easier, but of course makes the results less useful. It's important to make every class into which the datapoints are sorted a useful one, as they each impose an incremental difficulty on the system.

Secondly the number of images available as training data could be increased. This is especially important to a dataset like this, as the inner class variability is particularly high. There is much gain to be expected in expanding the training set.

Doing these two things could ensure a certain acceptable level of performance, though perhaps this approach to the problem is in itself inappropriate. One would perhaps be better off using the same system for an information retrieval or sorting task.

5.9.2 Action vs All

In this experiment the system acts as a binary classifier and is tested on the ActionKalooga dataset. This dataset distinguishes between Action photos and photos of any other class. Classes now have more than just 15 training point: there are 65 training images per experiment. It was tested using the most successful methods applied in the 12 classes experiment. The results are shown in table 5.9.

Discussion

The results achieved on this dataset are higher than found in the 12 class experiment, which is to be expected as there are now only 2 classes instead of 2. Furthermore, there are now 65 images to train on, instead of just 15. Considering those advantages, the system does not perform exceedingly well. This is a testament to the diversity of the Action Class, and it shows just how challenging the ActionKalooga dataset is to classify compared to the Caltech101 dataset.

classifier	SVM Perf.
Visual Keyword Descriptor of HoG 8×8 phase shift	77.91 ± 3.51
Visual Keyword Descriptor of HoG 4×4 phase shift	75.57 ± 3.49
SVM-Stack of all MPEG7 features	76.16 ± 3.68
SVM-Stack HoG $L_0 + L_1$	75.80 ± 3.11
HoG L_1 16 orientations	75.14 ± 5.37

Table 5.9: Results of the system as 2-class classifier on a the ActionKalooga dataset, separating the Action class from all other classes.

It is interesting to see that on these images, the visual keyword descriptor does perform better than the original descriptors it is based on. The shape was found in the image as a whole, while in the ActionKalooga dataset, smaller segments of the image may be found in other images. This is probably what makes this feature more suitable to the ActionKalooga dataset than the Caltech101 dataset. There is much to these features that could be improved for performance on the ActionKalooga dataset, but has not been explored due to time constraints and the scope of this project.

The combined MPEG7 features also scored relatively well, compared to the other descriptors, which was previously never the case. This is probably due to the combination of having fewer classes than there were in the 12 class problem and the fact that the Kalooga dataset is more defined by color than the Caltech101 dataset. So it seems that descriptors that are useful to a multiclass shape based problem like the Caltech dataset, are not necessarily as useful to a 2 class scene based problem like the ActionKalooga dataset.

5.9.3 Information Retrieval Approach: Detecting Action Images

In this experiment the information retrieval approach described in section 2.5 was tested on the ActionKalooga dataset. The action class of the Kalooga dataset are spectacular highlights of a soccer game, capturing impressive headers, shots, struggles, etc. The main idea behind the ActionKalooga is that when presenting an end user with a soccer related widget, Kalooga would prefer showing the action images over any of the other categories. These action images are detected within a query, sacrificing the number of action photos found, for precision. This sacrifice can be suffered however, as such a widget would only need to be filled with 5 or so photos.

In this experiment, the accurate detection of such images is achieved by only classifying images that were given a posterior probability by the SVM that exceeds a certain cutoff value. The cutoff value was used that achieved the highest $F_{0.25}$-value. The $F_{0.25}$ was chosen as Kalooga would only require a small amount of correctly classified images to fill a single widget. The results are shown in table 5.10.

classifier	cutoff	precision	recall	$F_{0.25}$
SVM-Stack of all MPEG7 features	90.9	93.86	33.42	84.84
Vis. Keyword Desc. HoG 8×8 phase shift	83.0	92.58	34.20	84.13
HoG $L_0 + L_1 + L_1$	87.0	87.44	39.98	81.74
HoG $L_0 + L_1$	88.9	89.00	41.52	83.40

Table 5.10: Detection of the Action Category in the Kalooga dataset. Classes with SVM Probability outputs below the cutoff value were ignored by the system.

Discussion

The performance of the different classifier/descriptor setups used in this experiment are hard to compare. It generally depends on the importance that is placed on precision and recall. The $F_{0.25}$-measure is the safest guideline for these results in particular, as the cutoff value was optimized to maximize that value. However, some methods may perform better on another kind of f-measure, that may also be even more appropriate. It is safe to assume though, that all setups work well enough.

An interesting observation is that in this problem, as in the previous, the MPEG7 descriptors and visual keyword descriptors work just as well or better than the HoG descriptor, where HoG would always outperform them in the Caltech101 experiments. Apparently, color information can be more easily exploited in classification problems on the Kalooga dataset than classification problems on the Caltech101 dataset. This is an interesting point for companies such as Kalooga, as it seems descriptor choice is very dependent on the nature of the dataset and task, which seem to dictate what descriptors to eventually develop and extract. It seems that to find out what descriptors are most useful for a particular kind of problem, research on that problem, or a similar type of problem is required to make a wise decision. It may be that investing in a wide range of image descriptors is more useful than developing a descriptor that scores well on scientific benchmarks on a single type of problem.

The results found show that the system is very usable in a detection task. The precision and recall are relatively high and the system is able to detect a good number of action images while maintaining a high precision. One should also consider that the results can be sorted. If the purpose is to only find a few images, those that are classified with high posterior probabilities will more probably belong to their designated classes. This way, the precision will probably be even higher if the number of images used is smaller than the number of images classified.

It should be noted that the system still works generically, and can be used to detect any preferred class on any problem. The action class is probably one of the harder classes in the dataset to detect as the images in it can be very different from each other. It may well be that similar results on other classes are even higher.

The system may also be used inversely to this experiment, in a filtering task instead of a detection task. In such a task the recall would likely become more important that the precision value. The system would then be optimized to

maximize an f-measure like F_2. This way the system could be used to remove unwanted images from a group of images. In the Kalooga Dataset discussed in this thesis for instance, that would be the product category.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this project an entire generic image classification pipeline has been successfully assembled. It supports a number of machine learning algorithms: k-nearest neighbors (KNNs), k-means clustering, Support Vector Machines (SVMs) and SVM stacking. These operate by classifying data described by visual descriptors extracted by a number of computer vision algorithms, namely: MPEG7 descriptors, a color histogram, histograms of gradients (HoG) and visual keyword descriptors.

This system was also backed up by a user friendly graphical user interface (gui), and together they allow the quick creation of a pipeline with different visual descriptors and classifiers. This tool helps in quickly creating a classification setup and testing the system on any choice domain of images. The tool is useful, but somewhat bogged down for distribution and development purposes by the libraries it requires.

These implemented methods were then tested on two datasets: a scientific dataset, Caltech101, and a Kalooga dataset. The Kalooga dataset was especially created for this project, and focusses on soccer. It was arranged by sorting images found by Kalooga soccer queries into useful visual categories. This dataset serves as a more realistic approximation of the tasks a system like this would face when being used to classify online images.

From the experiment on the Caltech101 dataset the HoG descriptor arose as the most effective descriptor, as was expected. Its performance is satisfactory, although it is somewhat poorer than results found earlier by Abdullah et al. (2010).

Two changes were made to the HoG descriptor which were also tested. One of these changes was dubbed phase shifting and entails rotating the orientation bins so that the orthogonal directions (up, down, left, right) would also be represented when using four orientation bins. The performance of the HoG descriptor improved because of phase shifting, especially when only using four orientation bins. This change allowed the HoG descriptor to become very small, using as few as up to $4 \times 4 = 16$ total histograms, each with only four values,

while maintaining a respectable performance.

Furthermore, when using more than the standard 8 bins, the descriptor seemed to perform better while the orientation bins were shifted. When trying to make an HoG more descriptive, not only the number of histograms (otherwise known as its pyramid level) should be taken into account. The number and arrangement of the orientation bins should also be considered for this.

The second change to the HoG was the use of a pixel's diagonals when calculating an image's gradient. Using said diagonals proved to be unimportant to the system's performance, as no significant difference could be found. Thus using diagonals when calculating gradients only serves to increase the method's complexity, without improving its performance.

Results employing feature combination techniques show that combining features can improve performance, as different features are able to complement each other. Using SVM stacks for this is more effective than using naive combination, especially as the combined descriptor size becomes large.

After testing the system on the Caltech101 data, it was tested on the Kalooga data. An important conclusion of the experiments on Kalooga data are that the image domain is very important to an image classification system's performance. A dataset such as the Kalooga soccer dataset, is much harder to classify than a scientific dataset such as the Caltech101 data. This is due to the shape based classes in Caltech101 as well as the inner-class variability of the categories of the Kalooga dataset. This difference in domain and difficulty also influences the visual descriptors that are optimal for the system's performance. For instance, images from the Kalooga dataset were relatively more effectively classified using color descriptors and visual keyword descriptors than the Caltech101 dataset.

The best approach to use the methods described in this master's thesis to practical image classification problems is to scale the problem down to a manageable level. An instance of this is given, by creating the detection task of detecting Action images from the Kalooga dataset. Using the SVM's posterior probability output, classifications of which the system is unsure can be ignored to assure a high precision. These posterior probabilities can also be used to sort the images on likeliness of containing the class. This way machine learning methods and computer vision methods may be used to provide a useful function for improving online queries.

This finally leaves the main research question stated in the introduction to answered. This question was: how can computer vision methods aid in improving online image query results? Using just the methods described in this thesis, the problem of image retrieval through image labeling itself can't be entirely solved currently. Nor can every multi class classification problem be solvable by these methods for every given image domain. However, when the problem is scaled down to a manageable size, these methods can very much provide a meaningful contribution by solving specific image-classification problems.

6.2 Future Work

The results and the tool created in this project can be improved and expanded on in many ways. Firstly, some of the features and methods implemented do not perform as well as reported in other studies. These might be changed and improved to match these higher performances. There are also many combination

of classifiers, descriptors and parameters that have not been explored. These may readily be explored and tested. More features and classifiers could also be added to the system as it is. The system allows the easy extension of classes so that new elements may be added. The tool could be further improved by removing some of its dependencies. It may also be improved with better display of information, adding additional functionality, and making existing functionalities more easily accessible.

An approach that has not been attempted is using the system as a means to filter out unwanted images. This resembles the detection task, however, in this case, the recall is more important than the precision. This way images that are not of interest to Kalooga could be filtered from a query.

Other datasets could be created that focus on other realistic problems like the Kalooga soccer dataset created for this project. These problems should be created in such a way that solving them is useful to parties like Kalooga. This prevents the problem from becoming too easy and the system designed to solve it from becoming too far removed from practical use.

Adding more images to such a dataset would improve performance as training size becomes larger. Exploring this would help understand the importance and influence of training set size on domains such as the Kalooga dataset.

It would also be interesting to further explore the versatility of the HoG descriptor while using the phase shift alteration. A HoG descriptor could be created that's extracted much faster than the current one by first resizing the image and using the phase shift alteration to create a small descriptor while keeping the descriptor descriptive enough to perform well.

This project shows that there is a basis for the practical application of Computer Vision and Machine learning methods to the problem of image retrieval. This basis is then best incrementally expanded upon in future work, using any number of relevant, interesting new methods. Perhaps when the system's performance on more limited problems becomes more reliable, the system can start being used for more general purposes as well.

Bibliography

Abdullah, A., Veltkamp, R. C., and Wiering, M. A. (2009a). An Ensemble of Deep Support Vector Machines for Image Categorization. In *SOCPAR '09 Proceedings of the 2009 International Conference of Soft Computing and Pattern Recognition*, pages 301–306.

Abdullah, A., Veltkamp, R. C., and Wiering, M. A. (2009b). Spatial pyramids and two-layer stacking svm classifiers for image categorization: A comparative study. In *IEEE - INNS - ENNS International Joint Conference on Neural Networks*, pages 5–12.

Abdullah, A., Veltkamp, R. C., and Wiering, M. A. (2010). Ensembles of novel visual keywords descriptors for image categorization. In *ICARCV*, pages 1206–1211. IEEE.

Bastan, M., Cam, H., Gudukbay, U., and Ulusoy, O. (2010). Bilvideo-7: An mpeg-7- compatible video indexing and retrieval system. *IEEE Multimedia*, 17:62–73.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152, New York, NY, USA. ACM.

Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 1st edition.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. In *Machine Learning*, pages 273–297.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Csurka, G., Dance, C. R., Fan, L., Willamowski, J., and Bray, C. (2004). Visual categorization with bags of keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision, Prague, 2004.*

Datta, R., Joshi, D., Li, J., and Wang, J. Z. (2008). Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2):1–60.

Davis, J. and Goadrich, M. (2006). *The relationship between Precision-Recall and ROC curves*. ICML '06. ACM Press, New York, New York, USA.

Demming, R. and Duffy, D. J. (2010). *Introduction to the Boost C++ Libraries - Volume 1 - Foundations*. Datasim.

Deselaers, T., Keysers, D., and Ney, H. (2008). Features for image retrieval: an experimental comparison. *Inf. Retr.*, 11:77–107.

Fei-Fei, L., Fergus, R., and Perona, P. (2004). Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. In *IEEE. CVPR*.

Fisher, Ronald Aylmer, S. (1925). An approximate distribution of estimates of variance components. *Metron*, 5:90–104.

Gough, B. J. and Stallman, R. M. (2004). *An Introduction to GCC*. Network Theory Ltd.

Griffin, G., Holub, A., and Perona, P. (2007). Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology.

Harchaoui, Z. (2007). Image classification with segmentation graph kernels. In *Proc. CVPR*.

Itti, L., Koch, C., and Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:1254–1259.

Liu, Y., Ma, W., Zhang, D., and Lu, G. (2007). A survey of content-based image retrieval with high-level semantics. *Pattern Recognition*, 40(1):262–282.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA. IEEE Computer Society.

Lux, M., Becker, J., and Krottmaier, H. (2003). Caliph & Emir : Semantic Annotation and Retrieval in Personal Digital Photo Libraries. In *15th CAiSE*, pages 85–89.

Manjunath, B. S. (2002). *Introduction to MPEG-7, Multimedia Content Description Interface*. John Wiley and Sons, Ltd.

Manjunath, B. S., Ohm, J. R., Vasudevan, V. V., and Yamada, A. (2001). Color and texture descriptors. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6):703–715.

Musser, D. R. and Saini, A. (1995). *The STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer series in operations research. Springer.

Novak, C. L. and Shafer, S. A. (1992). Anatomy of a Color Histogram. In *Proc. of Computer Vision and Pattern Recognition*, pages 599–605.

Nowak, E., Jurie, F., and Triggs, B. (2006). Sampling strategies for bag-of-features image classification. In *European Conference on Computer Vision*. Springer.

Oliva, A. and Torralba, A. (2006). Building the gist of a scene: the role of global image features in recognition. *Progress in brain research*, 155:23–36.

Rademacher, P. (2006). Glui user interface library. Software available at http://www.cs.unc.edu/rademach/glui/.

Rockafellar, R. T. (1993). Lagrange multipliers and optimality. *SIAM Rev.*, 35:183–238.

Satterthwaite, F. E. (1946). An approximate distribution of estimates of variance components. *Biometrika*, 33:110–114.

Shreiner, D., Woo, M., Neider, J., Davis, T., and OpenGL (2005). *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional.

Still, M. (2005). *The definitive guide to ImageMagick*. Apress.

Vailaya, A., Figueiredo, M. A. T., Jain, A. K., and Zhang, H.-J. (2001). Image classification for content-based indexing. *IEEE Transactions on Image Processing*, 10(1):117–130.

Won, C. S. W., Park, D. K. P., and Park, S.-J. P. (2002). Efficient Use of MPEG-7 Edge Histogram Descriptor. *ETRI Journal*, 24(1):23–30.

Zhang, L., Lin, F., and Zhang, B. (2001). Support vector machine learning for image retrieval. *Proc. IEEE Int. Conf. on Image Processing*, 2:721–724.