Detecting Cartoons: a Case Study in Automatic Video-Genre Classification

VNIVERSITAT DE VALÈNCIA

Autor: *Tzvetanka Ianeva Ianeva*
Tutor: *Juan José Martínez Durá*

Ph. D. Research Work

Valencia, 2003

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 What is this research work about?

This research project presents a new approach for classifying individual video frames as being a 'cartoon'[1] or a 'photographic image'. It is an extension of results published in [11].

The original motivation for this work arose from experiments performed at the TREC-2002 video retrieval benchmark: 'cartoons' are returned unexpectedly at high ranks even if the query gave only 'photographic' image examples. The Text REtrival Conferences (TREC) is a series of workshops for large scale evaluation of information retrieval technology (e.g., see [31]) sponsored by the National Institute of Standards and Technology (NIST) with additional support from other U.S. government agencies. Distinguishing between the two genres has proved difficult because of their large intra-class variation. In addition to image descriptors used in prior cartoon-classification work, this work introduces novel descriptors based on the pattern spectrum of parabolic and disk size distributions derived from parabolic and disk granulometries and the complexity of the image signal approximated by its compression ratio. The effectiveness of the proposed feature set for classification (using Support Vector Machines) is evaluated on a large set of key-frames from the TREC-2002 video track collection. The system is compared with one that classifies Web images as photographs or graphics and its superior performance is evident. Finally, we incorporated the cartoon filter into the retrieval framework of [33] and measured the improvement of the search results.

[1] The class 'cartoon' is defined more precisely in Section 1.5.

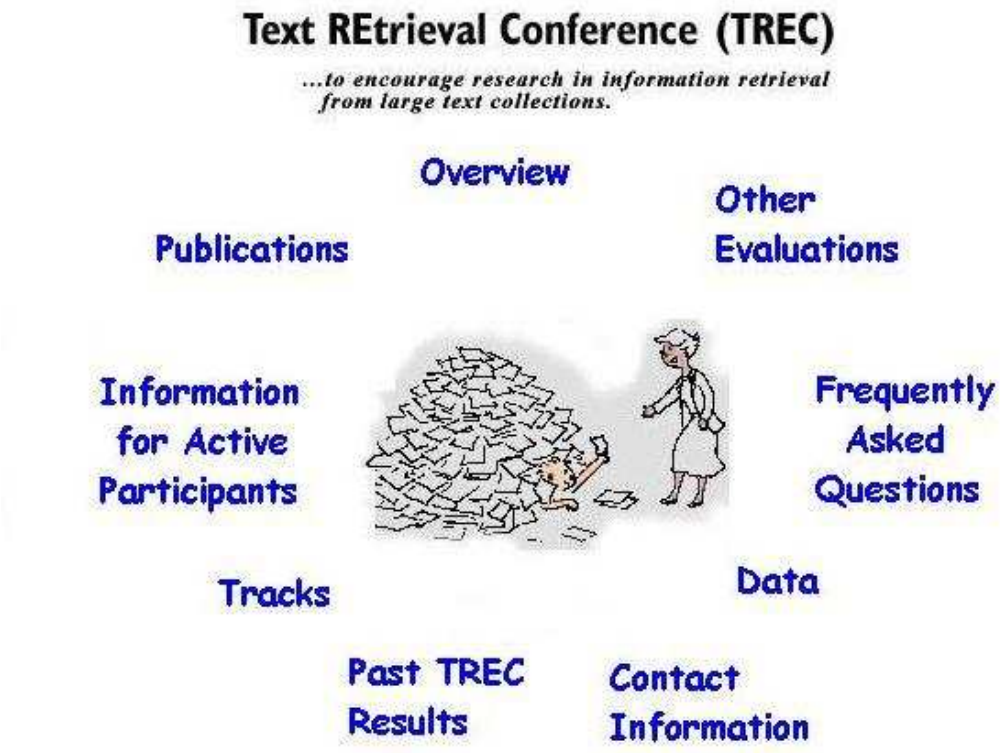


Figure 1.1: TREC

1.2 TREC Video Retrieval

The goal of the conference series is to encourage research in information retrieval (Figure 1.1) by testing retrieval technology on realistic test collection using uniform and appropriate scoring procedures. The general procedure is as follows:

- A set of M statements of information need (topic) is created[2];
- participants search the collection and return an ordered list of the top N results for each topic;

[2]see Appendix B for the TREC-2002 topic descriptions

- returned shots (i.e., sequences of frames generated during a continuous camera operation) are pooled and judged for relevance to the topic;
- systems are evaluated using the relevance judgments.

The measures used in evaluation are

$$\text{precision} := \frac{\text{number of relevant shots retrieved}}{\text{total number of shots retrieved}}$$

$$\text{recall} := \frac{\text{number of relevant shots retrieved}}{\text{total number of relevant shots in collection}}$$

In video retrieval, the total number of relevant shots in the collection is unknown because the collection is extremely large and not fully annotated. In this case, recall can only be approximated.

We can also compute the precision when 0, 10, ..., 100 shots have been retrieved. The average of these precision values is called the *average precision*; it is a more refined notion of precision in the sense that it penalizes more heavily errors that occur early in the retrieval.

1.2.1 TREC Tracks

A TREC workshop consists of a series of 'tracks', a set of tasks each focused on some facet of the retrieval problem. Examples of tracks include retrieval of speech documents, cross-language retrieval, retrieval of web documents, and question answering. The tracks support the retrieval research community by creating the infrastructure such as test collections necessary for task-specific research. The set of tracks that will be run in a given year of TREC is determined by the TREC program committee with a limit of at most eight tracks that can be run at one time. Some tracks that will be run in TREC 2003 are:

GENOME Track (GENOME is a new track for TREC 2003). The purpose of the track is to study the retrieval of genomic data, where genomic data is broadly interpreted to mean not just gene sequences but also supporting documentation such as research papers, lab reports, etc.

HARD Track (HARD is a new track for TREC 2003). The goal of HARD is to achieve High Accuracy Retrieval from Documents by leveraging additional information about the searcher and/or the search context, through techniques such as passage retrieval, and using very targeted interaction with the searcher.

NOVELTY Track A track to investigate systems' abilities to locate new (i.e., non-redundant) information.

QUESTION ANSWERING Track A track designed to take a step closer to information retrieval rather than document retrieval.

ROBUST RETRIEVAL Track A new track for TREC 2003. The task in the track will be a traditional ad hoc retrieval task, but with the focus on individual topic effectiveness rather than average effectiveness.

VIDEO Track A track designed to investigate content-based retrieval of digital video. [3]

WEB Track A track featuring search tasks on a document set that is a snapshot of the World Wide Web.

This research work is an attempt to improve the results on the search task of the TREC-2002 video track.

1.2.2 Video Track

TREC-2001 has introduced a video retrieval task, on a collection of (copyright free) videos produced between the 1930s and the 1970s (including advertising, educational, industrial, and amateur films) by corporations, nonprofit organizations, trade associations, community and interest groups, educational institutions, and individuals. The videos vary in their age, productional style, and quality. 16 teams representing 5 companies and 11 universities - 4 from Asia, 8 from Europe, and 4 from the US - participated in one or more of three tasks: shot boundary determination, feature extraction, and search (manual or interactive) [23].

The track defines three tasks: shot boundary detection, feature detection and general information search. The goal of the shot boundary task is to identify shot boundaries in a given video clip. In the feature detection task, one has to assign a set of predefined features to a shot, e.g., *indoor, outdoor, people* and *speech*. In the search task, the goal is to find relevant shots given a description of an information need, expressed by a multimedia topic. Both in the feature detection task and in the search task, a predefined set of video clips and images is to be used. [4]

[3] Beginning in 2003, this track will become an independent evaluation (TRECVID) with a 2-day workshop taking place each year just before TREC.

[4] More information about the track is available from the track website at `http://www-nlpir.nist.gov/projects/t01v/`

1.3 Motivation, Task and Contributions

Experiments at CWI (Centrum voor Wiskunde en Informatica) for the search task at the video track studied a generic probabilistic retrieval model that ranks shots based on the content of their keyframe image and speech transcript [33]. Evaluating the results, it was noticed that the model does not distinguish sufficiently between 'cartoons' and other keyframe images. Figures 1.2, 1.3, and 1.4 are examples of such behavior. Of course, one generally does not expect a 'cartoon' as query result unless explicitly asked for; consequently, returning these 'cartoons' by mistake results in a lower precision of the system.

The objective of this study is to implement a classifier that distinguishes 'cartoon' keyframes from 'non-cartoons'. The problem can be viewed as a case study of automatic video genre classification. The research work describes an approach which employs both grayscale and color image features. The output from various feature extractions is combined in a Support Vector Machines (SVM) training process to produce a classification model. The results demonstrate a small error rate on both the TREC-2002 video corpus and a collection of images gathered from the WWW.

The main contributions of this research are a rigorous analysis of the classification results on a large corpus, the use of image morphology in the feature set, as well as the good results achieved in spite of difficult, inhomogeneous data: indeed, our cartoon detector turns out to be useful for improving the TREC-2002 retrieval performance, not just for CWI's entry [33] but also on other contestants' entries.

1.4 Related work

Roach et al. published an approach for the classification of video fragments as cartoons using motion only [19]. Yet, their database consisted of only 8 cartoon and 20 non-cartoon sequences, so it is difficult to predict how it would perform on the TREC corpus, and their data set is not publicly available. Another recent effort addressed the classification of video into seven categories (including cartoons) [26]. Two of our features are similar to theirs, but our approach is different and the experiments are incomparable.

A closely related problem is the automatic classification of WWW images as photographs or graphics; examples are the WebSeek search engine [24] and the systems described in [20, 1]. Unfortunately, the most discriminative features used in these works take advantage of some characteristics of web images that do not exist in video keyframes, notably the aspect ratio

Figure 1.2: An example from CWI's entry to the TREC-2002 benchmark. The search algorithm is given the sample images and attempts to find similar images in the video database. The map of the US is considered an undesirable 'cartoon'.

Figure 1.3: Another example from CWI's entry to the TREC-2002 benchmark. Here a frame from an animated-picture cartoon is returned unexpectedly.

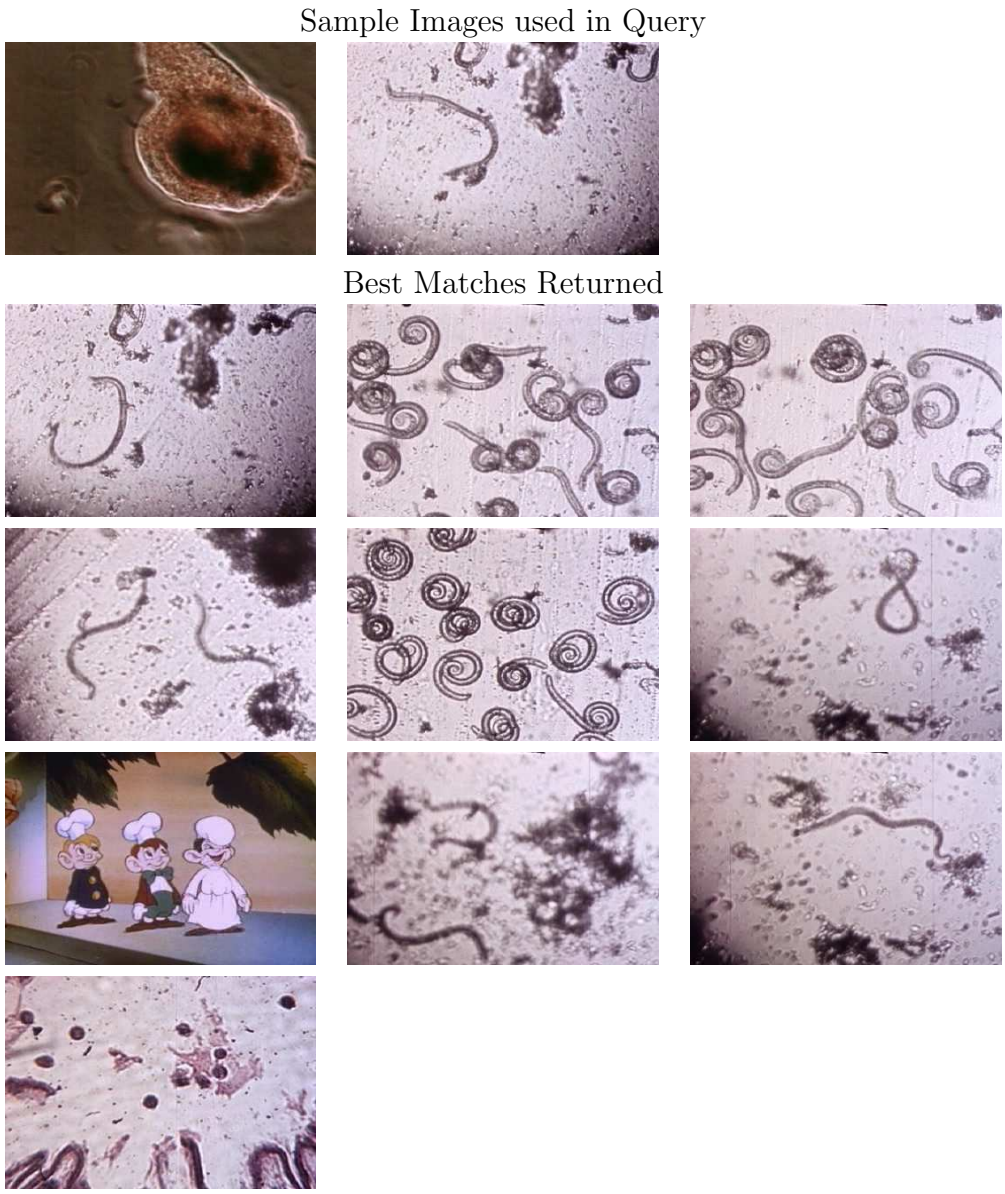


Figure 1.4: Another example from CWI's entry to the TREC-2002 benchmark. Here a frame from an animated-picture cartoon is returned unexpectedly.

and the word occurrence in the image URLs. We applied the farthest neighbor histogram descriptor suggested by [1] to our data collection, but this characteristic is expensive to compute without resulting in improved error rates.

The photo/graphics classifier of [1] had been previously implemented in INS1 (Data Mining and Knowledge Discovery) group as part of the Acoi system [34]. A decision-rule classifier (C4.5, [18]) has been trained on the features given in [1] on varying quantities of training data. However, as the results in Chapter 5 show, the features do not provide enough (or even no) discriminating power in the case of photo/cartoon classification on the TREC video collection. Conversely, the same implementation has a classification score of 0.9 on a data set of 14,040 photos and 9,512 graphics harvested from the WWW.

1.5 What is a Cartoon?

Recall that the objective of this project was to investigate the possibilities of filtering certain undesirable images from image lists generated by search operations in an image database. In particular, only scenes from the real world were of interest. This motivates the following definition:

> We call images or parts of images *photographic material* or *photos* if they have been obtained by means of a photographic camera.
>
> We call images *cartoons* if they do not contain any photographic material.

In this work we were only concerned with single frames. For this reason, our definition and the discussion of the properties of cartoons below do not take into account time-dependent properties of cartoons such as characteristic patterns of motion and rate of change.

Since they do not contain photographic material, cartoons are artificial images like animated cartoons for children (Figure 1.5), educational schematics, and movie credits (Figure 1.11). They are created by humans as pencil sketches, by drawing and painting, or with various technical tools for typesetting and computer animation. Some distinguishing features of cartoons are:

Few, simple, and strong colors: The abstraction in transforming a real-world scene into the cartoon world leads to a reduction of colors and exaggeration in saturation.

Patches of uniform color: Textures are often simplified to uniform color.

Strong black edges: The large patches of uniform color are often surrounded by strong black edges.

Text: Educational cartoons, charts, etc. often contain large text that is aligned horizontally and not distorted by a perspective transformation. Moreover, the fonts are chosen to be readable and the colors to give good contrast on TV.

Given this list, it may appear easy to separate cartoons from other keyframes. But in practice the problem is not as simple. Part of the problem lies in the low quality of the video streams in the TREC-2002 collection:

- the analog NTSC signal discards high-frequency components and therefore blurs sharp edges;
- the color fidelity is poor, with frames tending towards washed-out colors shifted towards blue or brown;
- photographic film transmitted over TV results in black speckles and alignment problems, e.g., the trapezoid black sections at the left and right of the image in Figure 1.5;
- the resolution of the digitized frames is low (352×240 pixels) and occasionally the frame grabber mixed two consecutive frames into a single digital image;
- the digitized frames were compressed using the lossy JPEG compression scheme, leading to artifacts like noise around sharp edges and rectangular areas of uniform color where the original image had smooth subtle color variations.

Another serious source of problems is that the possibly photographic origin of a picture often cannot be inferred with precision:

- Artificial scenes recorded by a photographic camera (Figure 1.13) or photography of abstract or technical objects (Figure 1.10) can have strong resemblance to a cartoon.
- 'Mixed' pictures such as Figure 1.9 or people in an theatrical scene mean that local and possibly very small features (see Figure 1.10) can reverse strong evidence in favor of a classification as cartoon.

- Postprocessing may hide or obfuscate any use of photographic material (e.g., Figure 1.12).

In fact, these problems often make it impossible even for humans to reach consensus in classifying the video frames from the training and testing collection. Consequently, the training sets are not very stable, which poses a great challenge to any automated procedure.

1.6 Outline

This research work is organized as follows in a bottom-up manner, proceeding from standard image descriptors to the construction of new descriptors, which are then fed into machine-learning algorithms for our experiments:

- Chapter 2 discusses image descriptors and presents the ones used previously in other research.
- Chapter 3 proposes new image descriptors based on granulometry and compression.
- Chapter 4 gives a short background about VC theory and kernel feature spaces and then proceeds to kernel-based learning used in the classification task.
- Chapter 5 reports on our experimental setup and results.
- Chapter 6 summarizes the contributions made with this research work and discusses directions for further research.
- The TREC-2002 topics are described in Appendix B.
- Source code of Matlab routines are provided in Appendix C.

Figure 1.5: A typical 'cartoon' from TREC-2002. Note the JPEG compression artifacts around the edges, the black cutoff at left and right, and the vertical stripes from the original film material.

Figure 1.6: This image from TREC-2002 clearly is not a cartoon, hence a 'photo'.

Figure 1.7: Another 'clear' cartoon image from TREC-2002, but the exaggerated light beams are the only distinguishing feature

Figure 1.8: From TREC-2002, a misleading photographic image of a car.

Figure 1.9: An illustration of the difficulty of mixed images in TREC-2002: small (human) features reverse an otherwise clear classification as cartoon

Figure 1.10: A photograph from TREC-2002 with all the cited properties of a cartoon; only the sparkle of the metallic parts and the shadows justify the classification as 'photograph.'

Figure 1.11: Movie credits count as a cartoon.

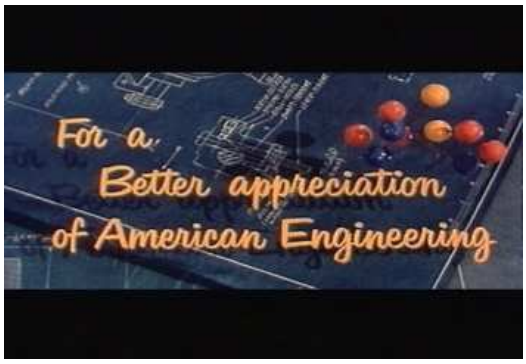


Figure 1.12: Postprocessing can obfuscate the use of photographic material.

Figure 1.13: Photography of artificial objects such as the architectural model here is hard to distinguish from 'cartoons'.

Chapter 2

FEATURE EXTRACTION

The system extracts a number of descriptors for each 'key-frame' image. In a probabilistic multimedia retrieval model is used straightforward key frame selection i.e., the middle frame from each shot is representative for the shot [33] and our full collection of images contains all these key-frames.

In this chapter we discuss our image descriptors similar to previously used in other classification systems. Table 2.1 presents an overview of all our image descriptors. Their individual 'naive' usefulness is given by the error on photos $\mathcal{E}(p)$, error on cartoons $\mathcal{E}(c)$, and total error $\mathcal{E}(t)$, when performing classification with machine learning (as outlined in Chapter 4) using the given image descriptor alone.

More of the extracted descriptors are based on a HSV color space and this is the reason why we start with a short presentation of the most popular color spaces, emphasizing on a HSV color space.

Image Descriptors	Dimension
average saturation	1
threshold brightness	1
color histogram	45
edge-direction histogram	40
compression ratio	1
multi-scale pat.spectrum	60

Table 2.1: Overview of Image Descriptors.

2.1 Color spaces

When looking at an image color is a visual feature which is immediately perceived. Models of color stimuli are used in retrieval by color similarity, such that distances in the color space correspond to human perceptual distances between colors. Ones of the most used color spaces in image analysis can be divided in:

- **Hardware-oriented** models. Here belong **RGB**, **CMY**, and **YIQ** color models. They are defined according to properties of optic devices used to reproduce colors, such as the TV monitor, the computer screen, the color printer, etc.

- **User-oriented** models. Here belong the **HIC**, **HCV**, **HSV**, **HSB**, and **MTM**. These are based on human perception (through hue, saturation, and brightness percepts) of colors. Hue describes the actual wavelength of the color percept. Saturation indicates the amount of white light which is present in a color. Highly saturated colors (pure colors) have no white light component. Brightness represents the intensity of a color.

2.1.1 Hardware-oriented models

RGB color space

RGB is the most commonly used hardware-oriented scheme for digital images based on the physiology of human retina. Colors in RGB are obtained as the addition of the three primaries Blue, Green and Red. The color space is visualized by a unit cube where each color (red, green, blue) is assigned to one of the three orthogonal coordinate axes in 3D space (Figure 2.1).

In this cube, as shown in (Figure 2.3):

- The RGB coordinates are related to *tristimulus*[1] values, XYZ, according to the following linear transformation [4]:

$$\begin{aligned} X &= 0.490R + 0.310G + 0.200B \\ Y &= 0.177R + 0.813G + 0.010B \\ Z &= 0.000R + 0.010G + 0.990B \end{aligned}$$

  White is represented as $X = 1$, $Y = 1$, $Z = 1$

[1] Tristimulus values X, Y, Z are the corresponding amounts of red, green, and blue necessary to form a particular color

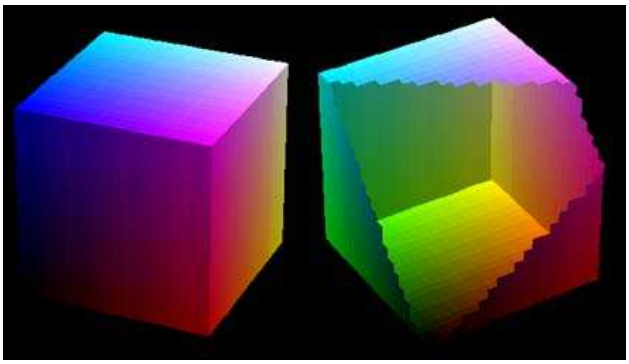


Figure 2.1: View of the shell of the color cube.

There are many measures that can be derived from the tristimulus values, these include chromaticity coordinates (x, y, z) and color spaces. Thus a color is specified by its chromaticity coefficients, defined as:

$$x = \frac{X}{X+Y+Z}$$
$$y = \frac{Y}{X+Y+Z}$$
$$z = \frac{Z}{X+Y+Z}$$

Chromaticity coordinates define color in relation to the *Chromaticity Diagram* [2] (Figure 2.2), which has a white point at its center with more saturated colors along the radii coming out from the center.

- Along each axis of the color cube, the colors range from no contribution of that component to a fully saturated color.
- Maximally saturated colors are placed at the corners of the cube, Red at (1,0,0), Green at (0,1,0), Blue at (0,1,1), Cyan at (0,1,1), Magenta at (1,0,0), and Yellow at (1,1,0).
- The color cube is solid, any point (color) within the cube is specified by three numbers, namely, an r,g,b triple.
- The diagonal line of the cube from black (0,0,0) to white (1,1,1) represents all the grays, that is, the red, green, and blue components have the same values.

[2]The chromaticity diagram is a tool for color definition but does not correspond to the operation of any hardware device, nor to the way in which human vision operates.

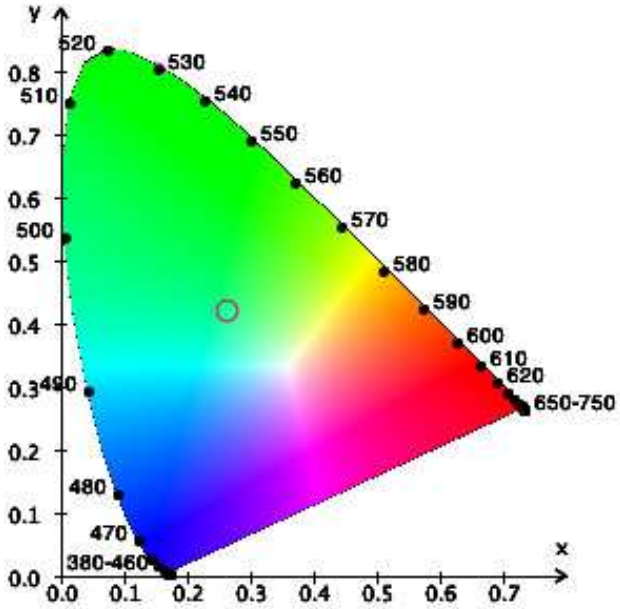


Figure 2.2: Chromaticity diagram. If three points of the diagram, corresponding to three primaries, are selected, all the colors obtainable by means of them are included in the triangle having these three points as vertexes.

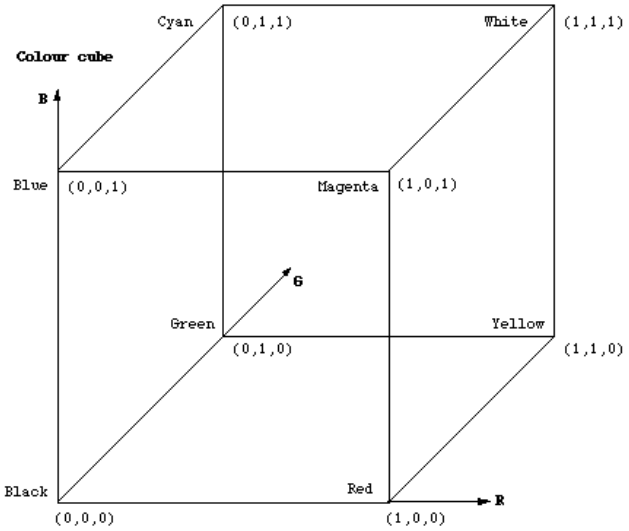


Figure 2.3: RGB color space.

- This RGB color space lies within our perceptual space, that is, the RGB cube is smaller and represents fewer colors than we can see.

CMY color space

CMY color space is used for color printing and it uses as primary colors cyan, magenta and yellow (CMY). Cyan, Magenta, Yellow are the complements of Red, Green, and Blue obtained by subtracting light from white. Therefore, white is at (0,0,0) and black is at (1,1,1) in the CMY color space as shown in Figure 2.4.

Conversion from RGB to CMY can be done by converting RGB into XYZ and hence converting XYZ into CMY. RGB colors are purer than the corresponding colors in CMY (i.e., the color solid is smaller in CMY) and this must be taken into account in order to print an image of a computer monitor on a color printer.

YIQ color space

The NTSC format is used in televisions in the United States. One of the main advantages of this format is that gray-scale information is separated from color data, so the same signal can be used for both color and black and white sets. In the NTSC format, image data consists of three components: luminance (Y), hue (I), and saturation (Q). The first component, luminance,

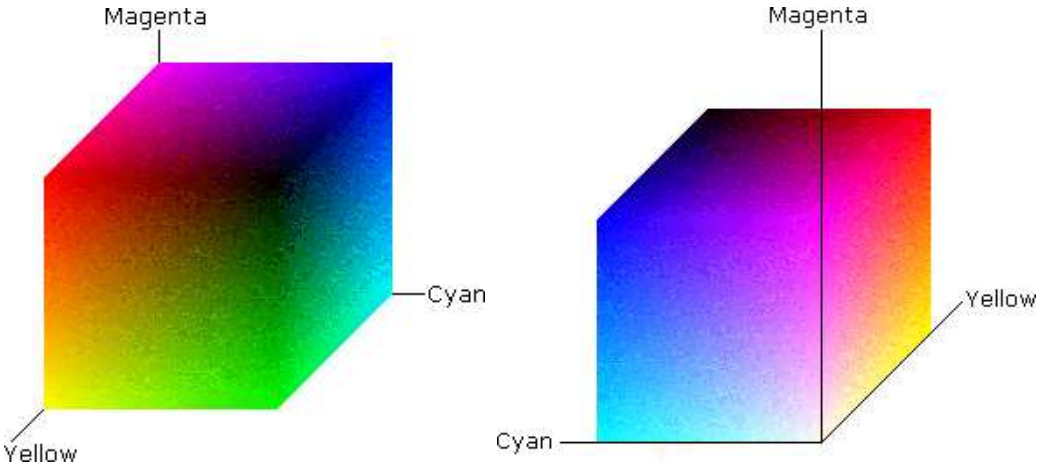


Figure 2.4: Color representations of the CMY color space.

represents gray-scale information, while the last two components make up chrominance (color information).

Similar to the YIQ are the YUV, and YCrCb color spaces. Each of these produces a linear transform of RGB which generates one luminance channel and two chrominance channels. The transformations were designed specifically to the parameters of the expected display devices: YIQ - NTSC color television, YUV - PAL and SECAM color television, and YCrCb - color computer display. The YCrCb color space is used in the JPEG digital image standard. None of these color spaces is *uniform*.[3] As such, the color distances in these transform color spaces do not correspond to color dissimilarities.

2.1.2 User-oriented models

HIS (Hue, Intensity, and Saturation), HCV (Hue, Chroma, and Value), HSV (Hue, Saturation, and Value), and HSB (Hue, Saturation, and Brightness) color models are closely related to each other while they all are approximations of the Munsell system (Figure 2.5) and support an intuitive notion of color since they separate luminance from the other components. Color components can be derived from either RGB or XYZ through linear or non-linear transformations. These spaces have been used extensively in computer vision

[3]Uniform color spaces are spaces such that a color difference perceived by a human observer is approximated as the Euclidean distance between two points in the color space.

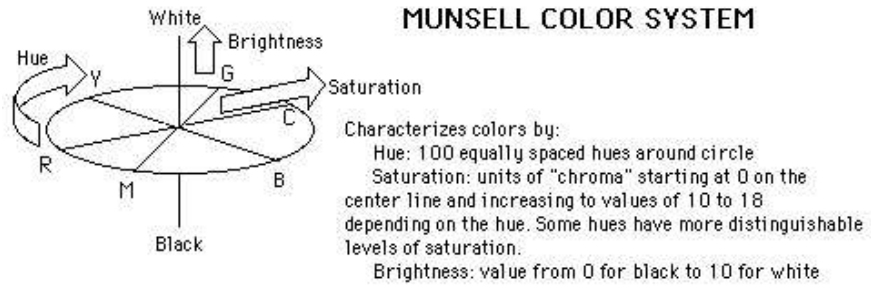


Figure 2.5: Munsell system.

and computer graphics. They all are *device dependent*[4] and non-linear.

MTM color space

Mathematical Transformation to Munsell system (**MTM**) space is a color space which is obtained from RGB through a simple transformation. It is perceptually uniform and very closely represents the human way of perceiving colors. Transformation from RGB to MTM is reported in [17].

HIS color space

The **HIS** model is represented as a double cone as in (Figure 2.6). The axis of the cone is the gray scale with White in the top cone vertex and Black in the bottom cone vertex. Hue is represented by the angle around the vertical axis. Primary colors are located on the maximum circle, equally spaced at 60 degrees (counterclockwise: Red, Yellow, Green, Cyan, Blue, Magenta).

HSB color space

The **HSB** color space derives from Hurvich and Jamenson's opponent color theory[10]. This theory is based on observation that opponent hues (Yellow and Blue, Green and Red) cancel each other when superimposed. HSB is a polar coordinate model. This model can represent with a certain fidelity a great range of psychophysical phenomena. From RGB opponent colors can

[4] Each color device has a particular set of colors that it can produce. This color set is known as its *gamut*

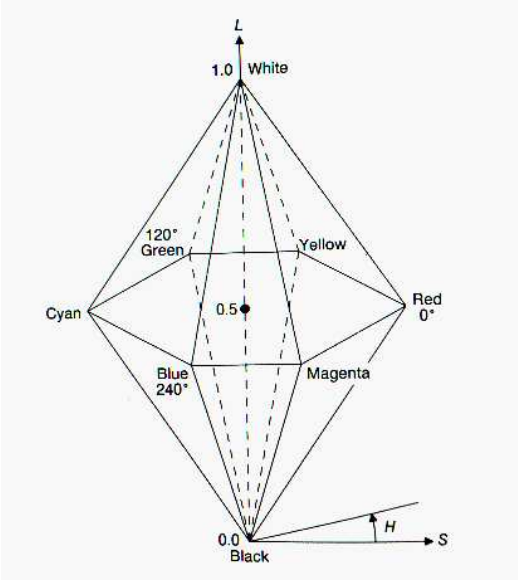


Figure 2.6: HIS color model.

be defined through the linear transformations:

$$rg = R - G$$
$$by = 2^B - R - G$$
$$wb = R + G + B$$

The intensity axis wb can be more coarsely sampled than the other two.

HSV color space

The **HSV (Hue, Saturation, Value)** color model (Figure 2.7) is represented by a cone where the cone axis represents the line of grey. HSV can be derived from the RGB space according to the following transformation:

$$V = \frac{1}{3}(R + G + B)$$
$$S = 1 - \frac{3}{R + G + B} min(R, G, B)$$
$$H = 180 \frac{0.5(R - G) + (R - B)}{((R - G) + (R - B)(G - B))^{1/2}}$$
$$H = \textit{undefined} \text{ if } S = 0$$
$$H = 360 - H \text{ if} B/V > G/V$$

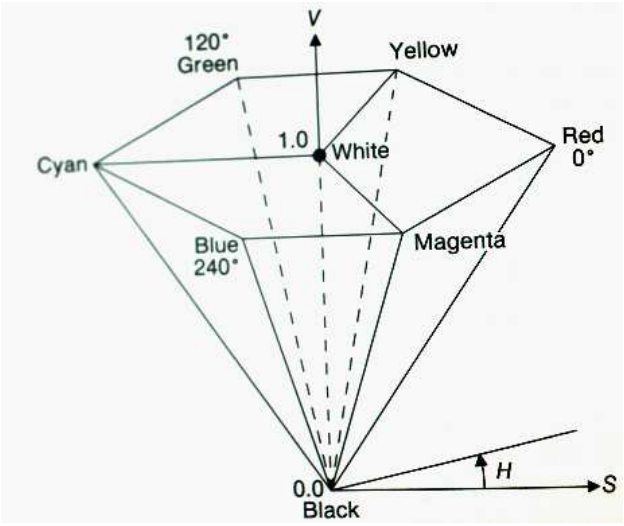


Figure 2.7: HSV model.

In our program we use the functions rgb2hsv and hsv2rgb from Matlab Image Processing Toolbox that convert images between the RGB and HSV color spaces.

As hue varies from 0 to 1.0, the corresponding colors vary from red, through yellow, green, cyan, blue, and magenta, back to red, so that there are actually red values both at 0 and 1.0. As saturation varies from 0 to 1.0, the corresponding colors (hues) vary from unsaturated (shades of gray) to fully saturated (no white component). As value, or brightness, varies from 0 to 1.0, the corresponding colors become increasingly brighter (see Figure 2.8).

2.2 Saturation, Brightness, and Color Histogram

The HSV space provides two interesting descriptors. Figure 2.9 shows that the average saturation of cartoons is much higher when compared against photographic images. We compute our first descriptor as the average color saturation using the S channel of the image in the HSV color space.

We call our second descriptor 'threshold brightness', that is, the percentage of pixels having brightness (the V channel of the image in the HSV color space) above a certain threshold. The average brightness for cartoons (Figure 2.10) is much higher than photos and we obtained best results using a

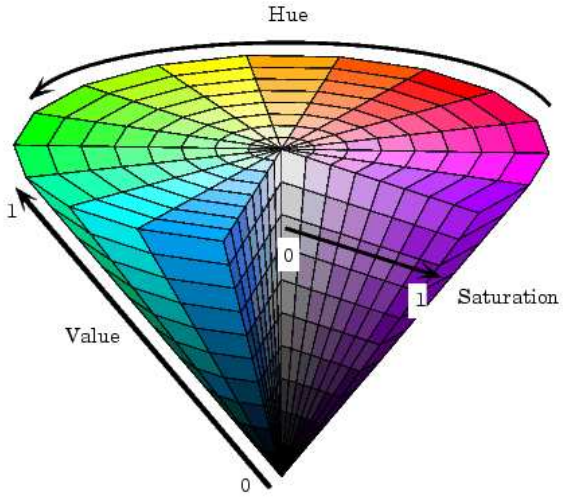


Figure 2.8: HSV color model.

threshold of 0.2, as explained in Chapter 5.

Similar descriptors were used in [26] and we also observed good correlation with class membership.

The color histogram is the most traditional way of describing low-level color properties of images. It can be represented as three independent color distributions in each primary, as two independent distributions (for color spaces which separate chromatic information from luminance) or, more frequently, as one distribution over the three primaries, obtained by discretizing image colors and counting how many pixels belong to each color. Following the most frequent use of the color histogram to obtain our third descriptor, we compute a $3 \times 3 \times 5$ histogram of the image in the HSV color space and use the 45 numbers (normalized by the number of pixels) as 45 image descriptors. Figures 2.11–2.14 confirm the intuition that cartoons have few colors and therefore few highly-filled bins in their color histogram.

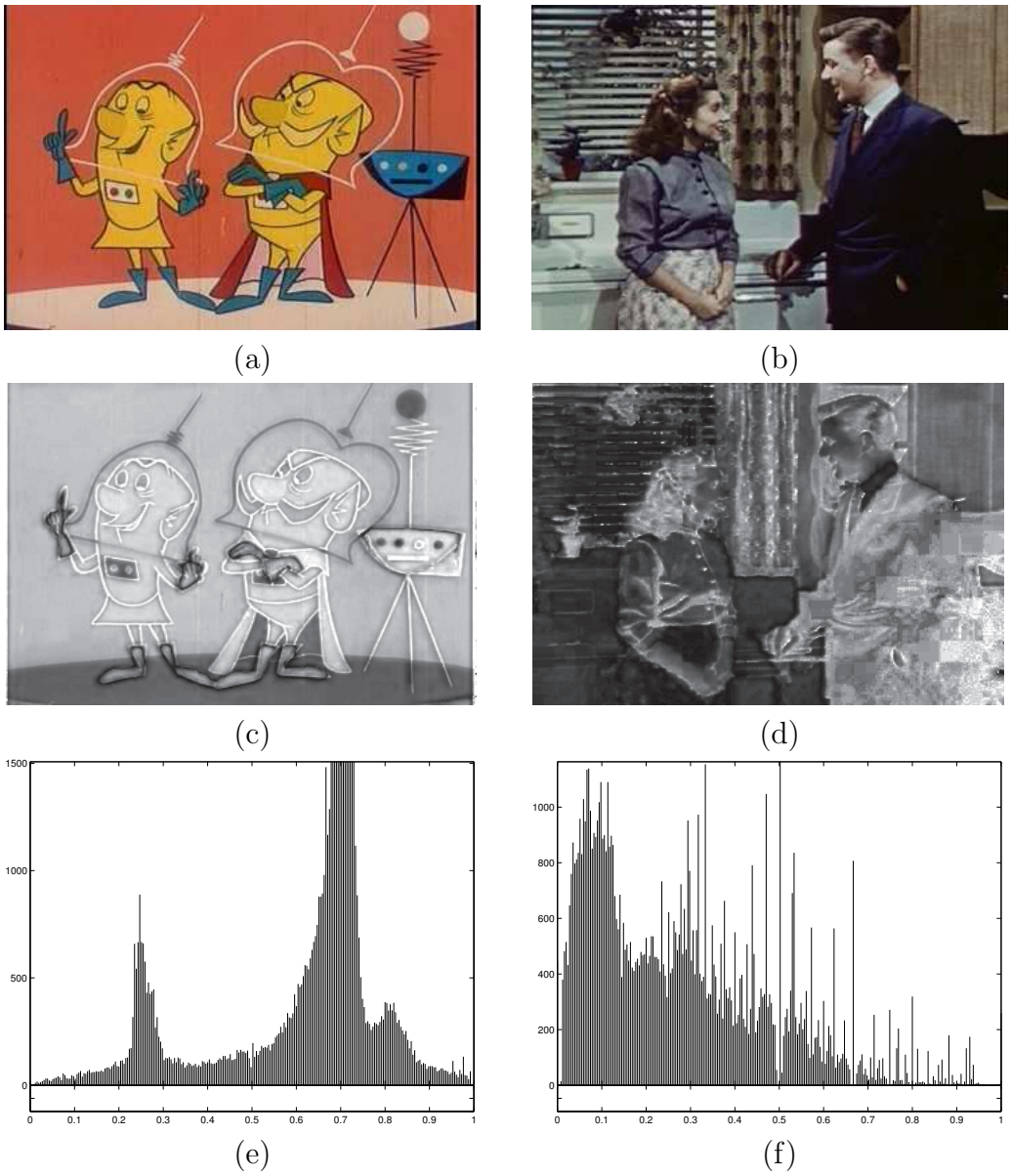


Figure 2.9: Saturation in cartoon and photo images. An original (rgb) (a) cartoon and (b) photographic image. Saturation of (c) cartoon and (d) photographic image (S channel of HSV color space). Histogram: (e) Computed average saturation **0.6231** in cartoon. Note that the scale here goes to 1500. (f) Computed average saturation **0.2880** in photographic image.

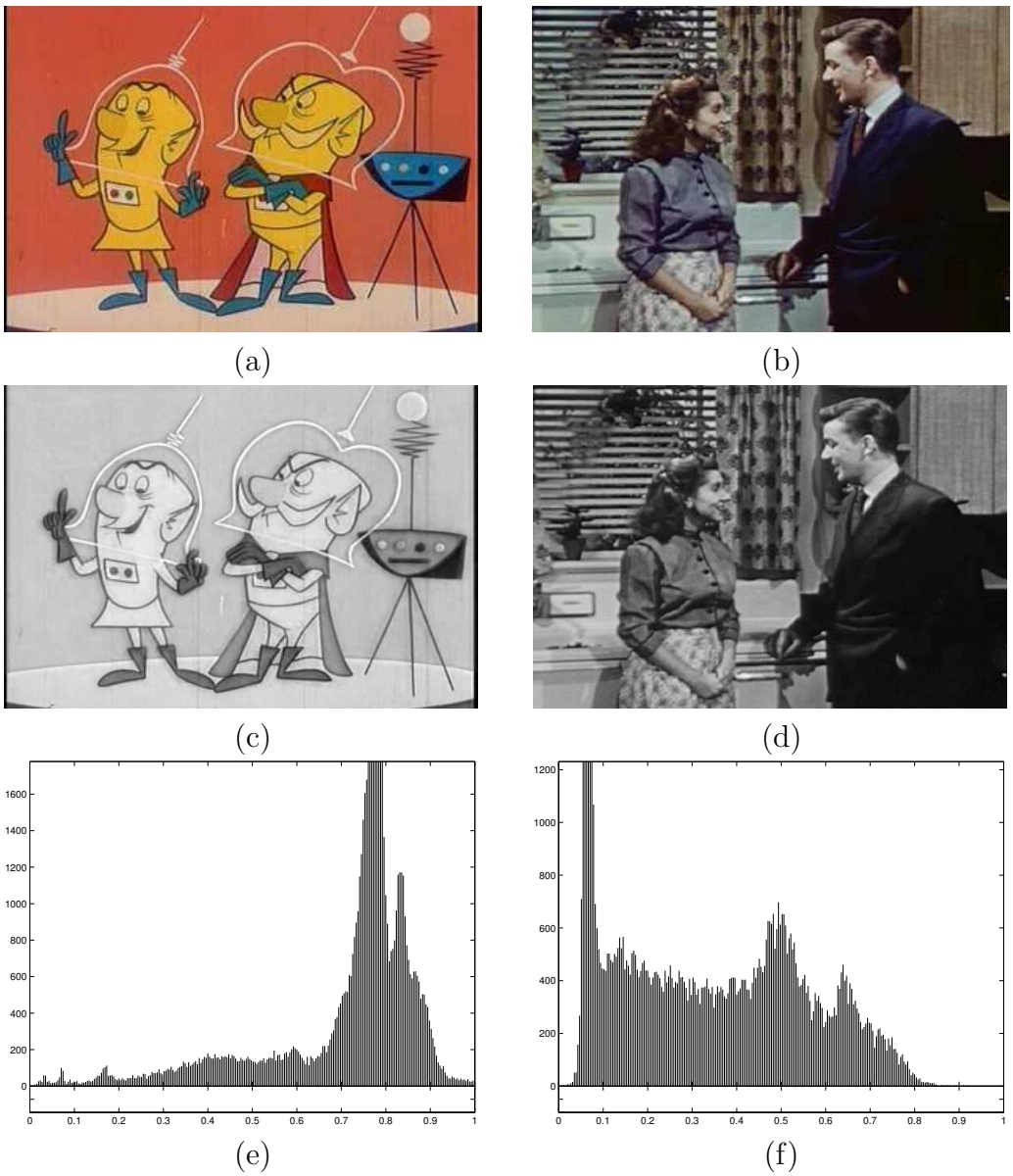


Figure 2.10: Brightness in cartoon and photo images. An original (rgb) (a) cartoon and (b) photographic image. Grayscale (c) cartoon and (d) photographic image (HSV color space). Histogram. Computed threshold (0.4) brightness **0.9775** in (e) cartoon and **0.6459** in (f).

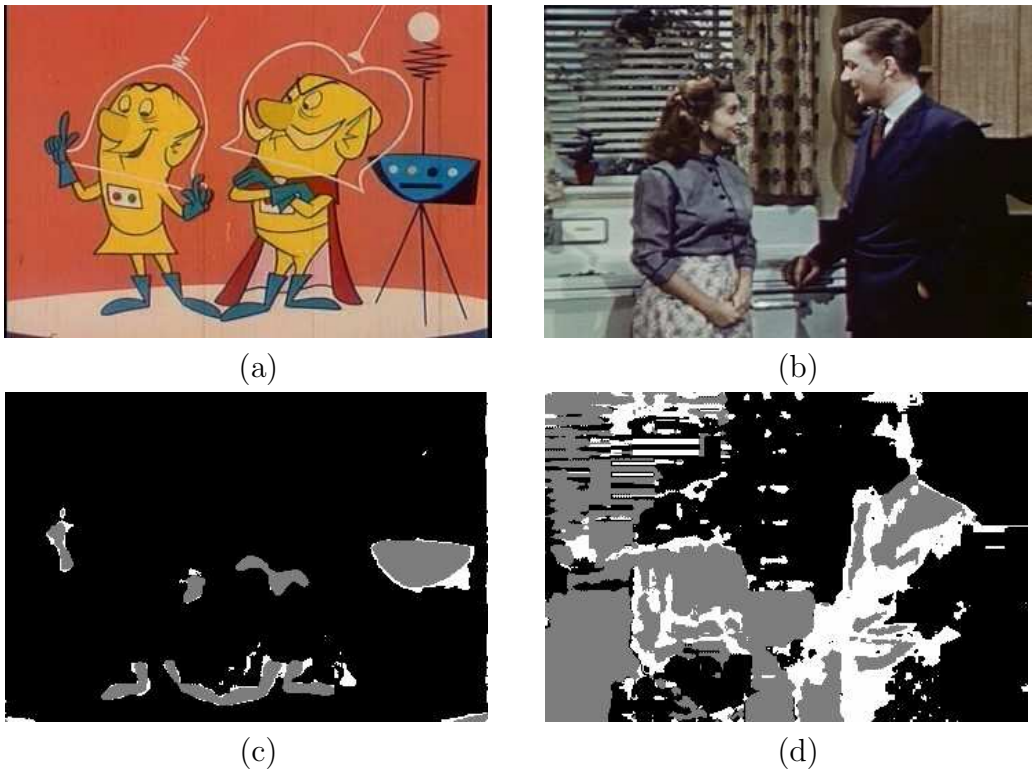
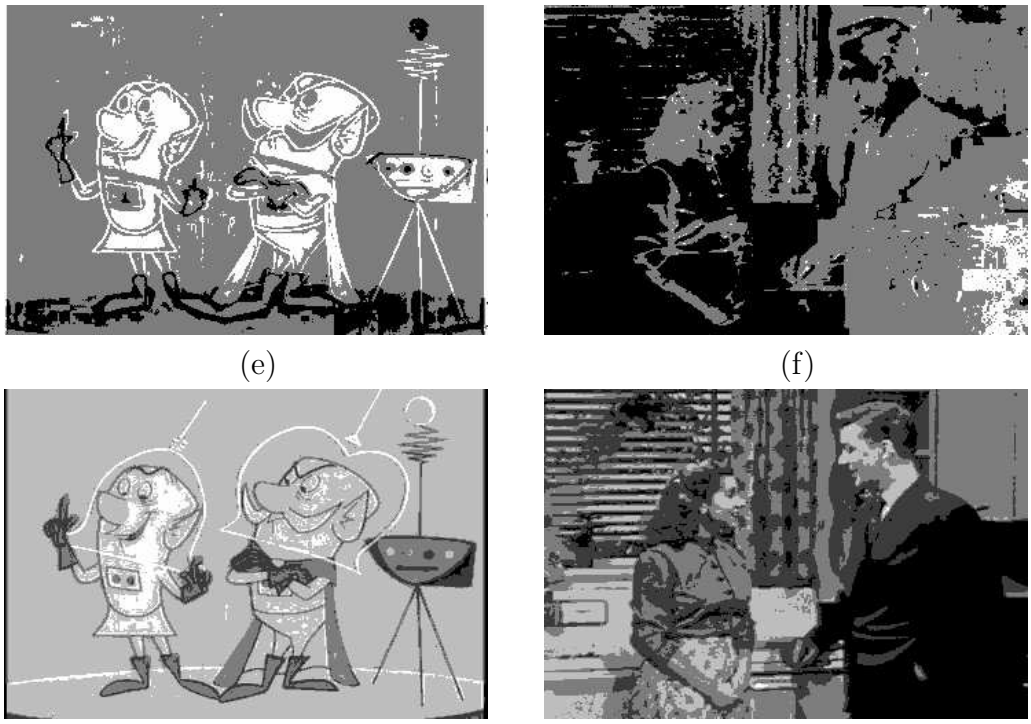


Figure 2.11: Color histogram descriptors in cartoon and photo images based on 45-bin HSV color space. An original (RGB) (a) cartoon and (b) photographic image. H channel of the (c) cartoon and (d) photographic image quantized to 3 values.



(e) (f)

(g) (h)

Figure 2.12: S channel of the (e) cartoon and (f) photographic image quantized to 3 values. V channel of the (g) cartoon and (h) photographic image quantized to 5 values.

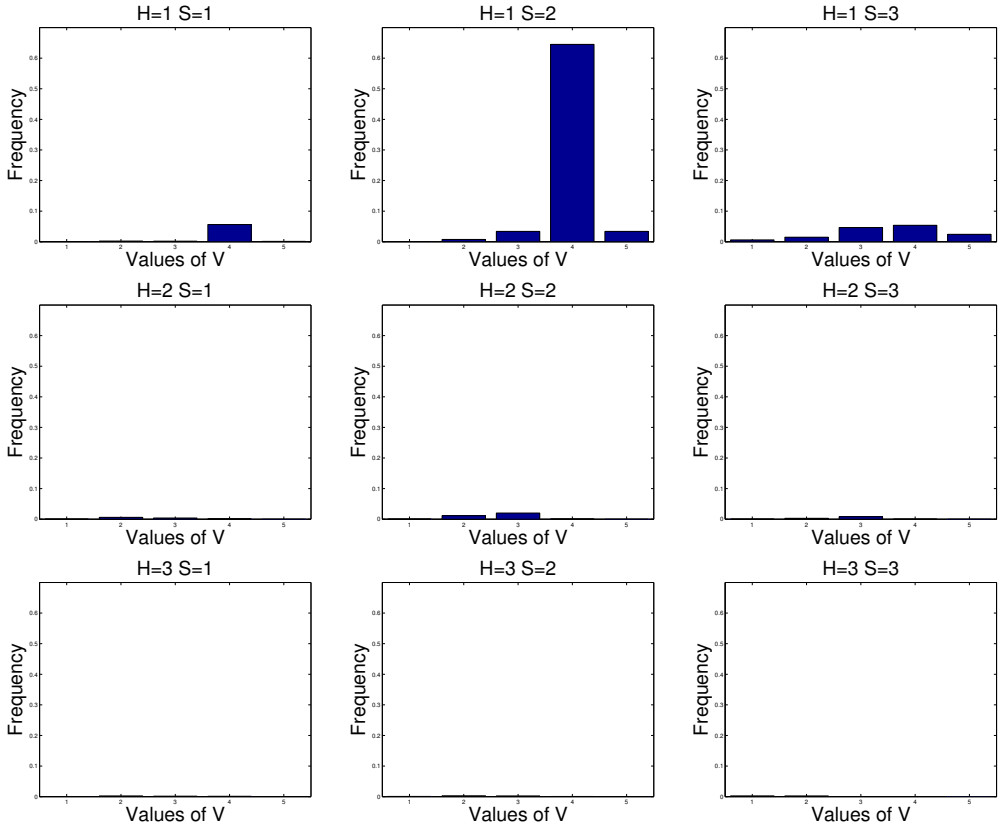


Figure 2.13: (i) – color histogram for cartoon. The $3 \times 3 \times 5$ histogram in the HSV color space is given by one-dimensional histograms for the V channel for each of the possible quantized values of the H and S channel.

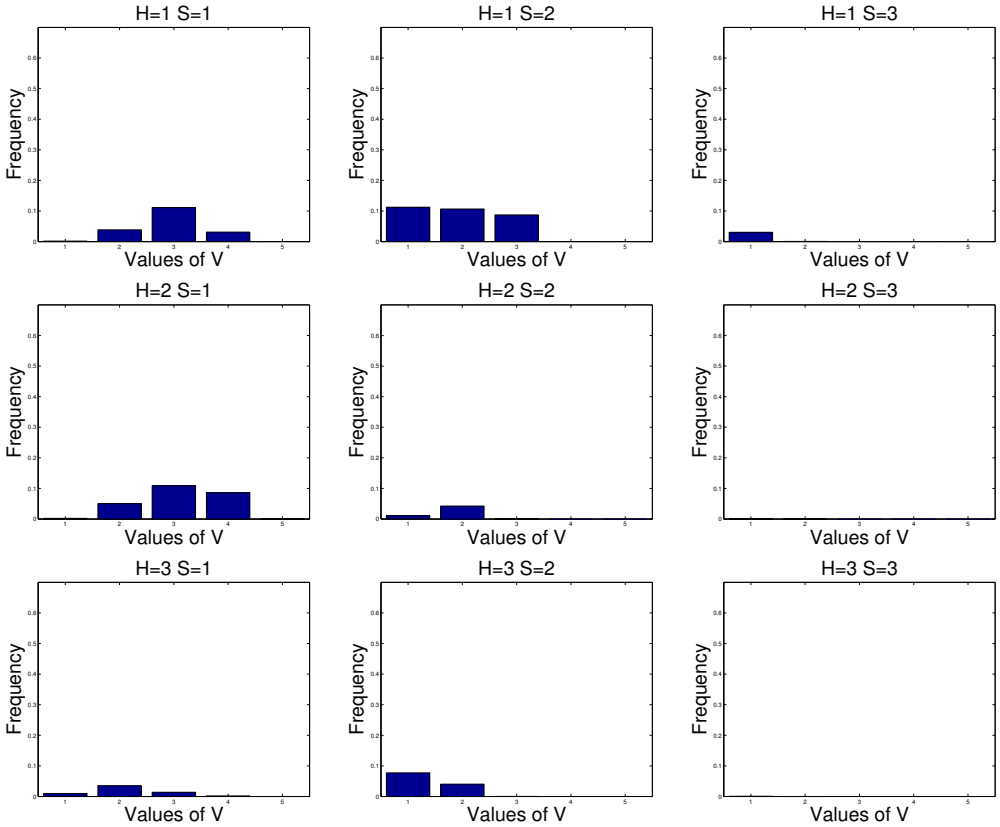


Figure 2.14: (j) – color histogram for photographic image. The $3 \times 3 \times 5$ histogram in the HSV color space is given by one-dimensional histograms for the V channel for each of the possible quantized values of the H and S channel.

2.3 Edge Detection

Cartoons are expected to have strong black edges. See Figure 2.15 for an example, where areas of uniform color are surrounded by black lines. To derive an image descriptor for measuring this property, we reduce the image to gray-scale (example in Figure 2.16) and interpret this gray-scale image as a real function $I(x, y)$ of two variables; this allows us to reason about the local rate of change and the direction of the steepest ascent or descent. Coordinates (x, y) where $I(x, y)$ changes a lot are likely to be part of an edge and therefore we classify local change.

Our edge-detection image descriptor will be based on the gradient of I. The gradient of I in point (x, y) is

$$\nabla I(x, y) = \left(\frac{\partial}{\partial x} I(x, y), \frac{\partial}{\partial y} I(x, y) \right)$$

where the horizontal partial derivative $\partial/\partial x I(x, y)$ in point (x, y) is the rate of change (derivative) of I when x is varied and y kept constant. (The same holds vice versa for the vertical partial derivative.)

To approximate the horizontal and vertical derivatives, respectively, we filter the image using the horizontal and vertical Sobel filters, respectively:

$$S_{\text{horizontal}} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \text{ and } S_{\text{vertical}} = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

For example, when filtering the image with $S_{\text{horizontal}}$,

$$\begin{aligned} (S_{\text{horizontal}} * I)(x, y) = &- I(x-1, y-1) + I(x+1, y-1) \\ &- 2I(x-1, y) + 2I(x+1, y) \\ &- I(x-1, y+1) + I(x+1, y+1) \end{aligned}$$

approximates the horizontal change by subtracting the values of neighboring pixels at the left from the values of neighboring pixels at the right, giving greater weight to closer neighbors. Figures 2.17 and 2.18 show the effect of applying the horizontal and vertical Sobel filter to our example image.

The gradient $\nabla I(x, y)$ is a vector pointing in the direction of the greatest ascent. Therefore, the angle $\theta(x, y)$ of the gradient at (x, y) is defined via

$$\tan \theta(x, y) = \frac{\frac{\partial}{\partial y} I(x, y)}{\frac{\partial}{\partial x} I(x, y)}.$$

Figure 2.15: A typical cartoon from TREC-2002 with clearly defined black edges and a photographic image from TREC-2002 for comparison.

Figure 2.16: The preceding images reduced to gray-scale.

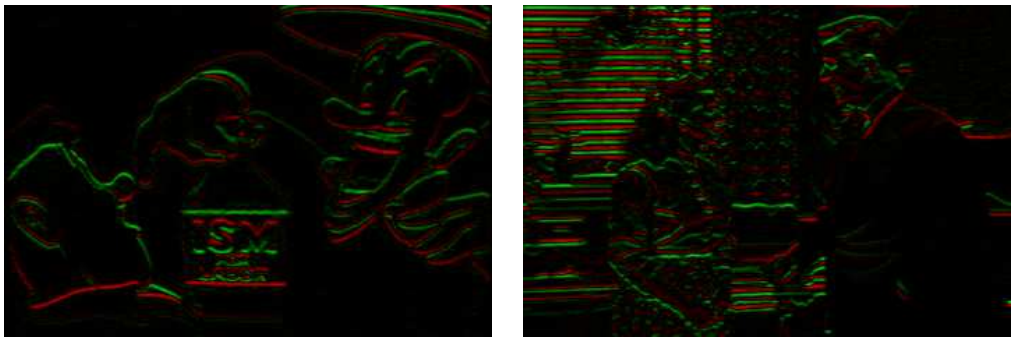


Figure 2.17: The horizontal Sobel filter applied to the gray-scale images. Sharp increases in brightness when going from left to right are represented by strong green pixels (these are the positive values in the filtered image); sharp decreases by strong shades of red (negative values in the filtered image).

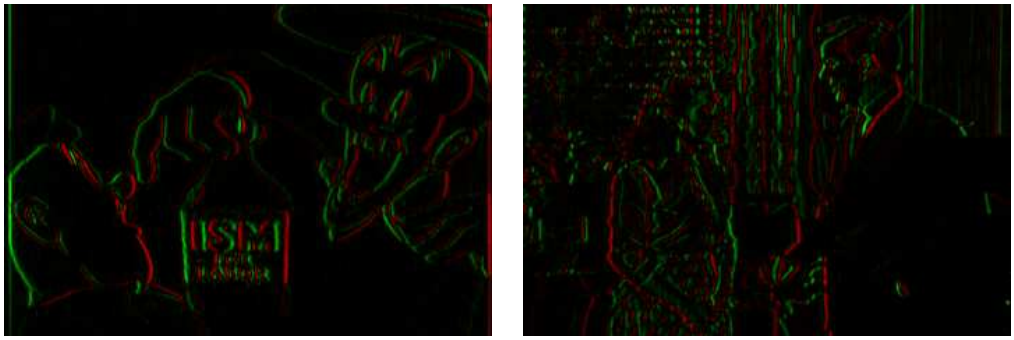


Figure 2.18: The vertical Sobel filter applied to the gray-scale images. Sharp increases in brightness when going downwards are represented by strong green pixels (these are the positive values in the filtered image); sharp decreases by strong shades of red (negative values in the filtered image).

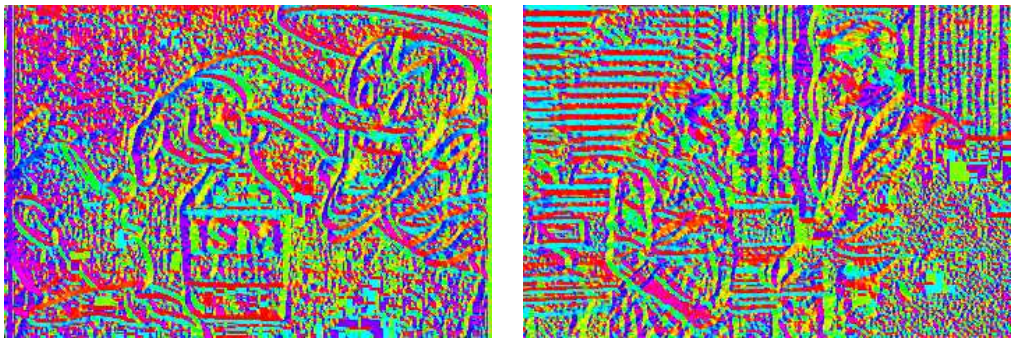


Figure 2.19: The edge angles presented by false saturated colors. For instance, blue marks upward edges, green downward edges, and red horizontal edges.

Figure 2.20: The edge magnitude.

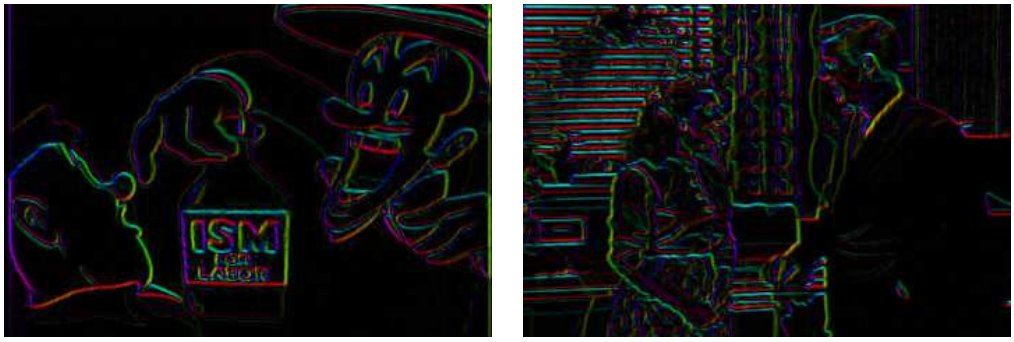


Figure 2.21: The edge angles with the edge magnitude as brightness. Our image descriptor is essentially a histogram of this image.

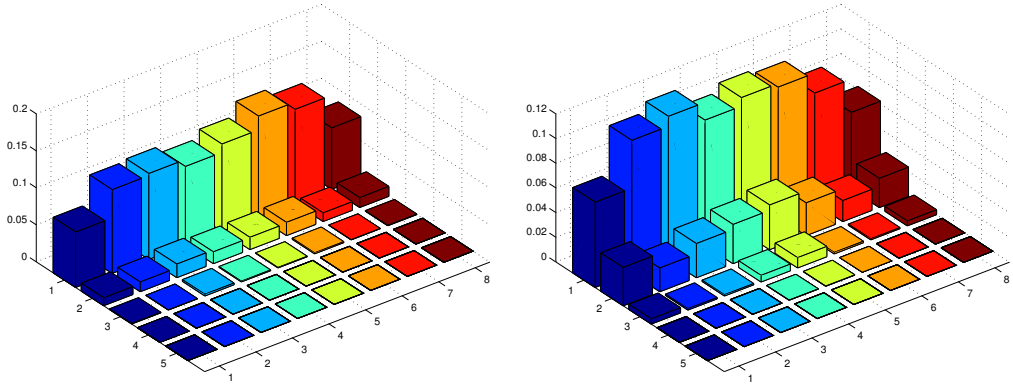


Figure 2.22: The edge histogram of an image (the cartoon on the left, the photographic image on the right) gives for each of five intervals of magnitudes and eight intervals of angles the fraction of pixels that fall into a given magnitude interval and a given angle interval.

The length of the gradient vector, its norm $m(x, y) = |\nabla I(x, y)|$, is

$$m(x, y) = \sqrt{\left(\frac{\partial}{\partial x} I(x, y)\right)^2 + \left(\frac{\partial}{\partial y} I(x, y)\right)^2}.$$

Our approximations of the partial horizontal and vertical derivatives permit us to compute approximations of $\theta(x, y)$ and $m(x, y)$. This is illustrated in Figures 2.19 and 2.20.

The approximations of $\theta(x, y)$ and $m(x, y)$ are gray-scale images themselves and we compute a histogram of the frequencies for tuples of particular values (θ, m). Concretely, we divided the range of angles $0 \ldots 2\pi$ into eight uniform intervals and the range of magnitudes $0 \ldots \sqrt{20}$ into five uniform intervals. This yields $8 \cdot 5 = 40$ frequencies normalized to values between 0 and 1, which are stored as a vector of 40 floating-point numbers. This approach is inspired by [7] where a similar technique is outlined.

Chapter 3

NOVEL DESCRIPTORS

In this chapter we develop two new classes of image descriptors to supplement the commonly used descriptors presented in the preceding chapter (color histogram and edge based descriptors, average saturation and threshold brightness). The first class of descriptors attempts to recognize the simple composition of images by measuring the compression ratio, whereas the second approach is to use pattern spectra to differentiate between cartoons and photographic images.

3.1 Compression

3.1.1 Descriptive Complexity

Cartoons are expected to have a more simple composition than photographic images: they typically have few colors, simple geometric shapes, etc. Hence, an image descriptor measuring the 'complexity' of an image should be useful in distinguishing cartoons from photographic images.

There is an extensive theory about the descriptive complexity of data objects [14]. Here (finite) data objects x and (finite) descriptions p are abstracted to (finite) bit strings $x, p \in \bigcup_{n\geq 0}\{0,1\}^n$; description methods are represented by partial functions T mapping descriptions p to outputs x (or no output). We are only interested in description methods that can in principle be implemented on a computer. Therefore we require that T is computable. With these preliminaries, we can define the complexity of x with respect to description method T as

$$C_T(x) := \min\{|p| : \text{given input } p, \text{ method } T \text{ produces } x\}$$

where $|p|$ is the length of the bit string p. For example, T could be the LZ77

decompression algorithm [13] used by the GNU `gunzip` utility [15, 5], p the compressed file and x the uncompressed file.

Surprisingly, there exists an effective description method U so that for any other description method T, $C_U(x) \leq C_T(x) + c$ where $c \in \mathbb{N}$ is a constant depending only on the choice of U and T but not on x. Therefore U is called an *universal* description method; the theorem proving its existence is called the 'Invariance Theorem' and given in [14]. $C(x) := C_U(x)$ is called the Kolmogorov complexity of x. If we interpret $C_T(x)$ as the length of the file x compressed so that it can be uncompressed with method T, then the Invariance Theorem says that the compressed version of x with respect to U is only a longer by a constant number of bits than the compressed version of x with respect to T for any (de)compression method T.

Since the Kolmogorov complexity $C(x)$ measures truly detects *any* algorithmic regularity of x, it is the ideal formalization for the intuitive inherent complexity of given image: there is no need to specify what kind of simplicity we are looking for, because $C_U(x)$ will measures *all* regularity exploitable by a computer. Hence, $C(x)$ would be a very useful image descriptor. However, the nice properties of $C(x)$ come at a large price: finding the *shortest* compressed version of x with respect to the universal method U is *uncomputable* and, worse, essentially any approximation of $C(x)$ with meaningful overall guarantees other than $C(x) \leq |x| + c'$ (for some constant c') is also uncomputable. Consequently, we are forced to resort to approximation methods that come without any guarantee of how much of the existing algorithmic regularity they can find. Our approach was to take the size of the images when compressed with commonly used image-compression techniques as an approximation to $C(x)$. Note, however, that for lossy compression, this approximation can even be smaller than the true $C(x)$.

3.1.2 Image Compression

Image-compression algorithms fall into two categories: lossy and non-lossy image compression. As the name suggests, non-lossy compression permits the perfect reconstruction of the input image from the compressed data. On the other hand, lossy compression aims at discarding irrelevant parts of the image to reduce the size of its representation.

Lossless image compression is usually implemented by applying a reversible filter to the image and then compressing the result using a generic compression algorithm. The compression algorithm is typically a method such as run-length encoding, entropy coding (Huffman or arithmetic coding), or techniques originally developed for text such as LZ77 [13]. Because these algorithms operate on a bit string, which conceptually is one-dimensional

data, the preceding filtering stage can improve compression. For example, in the PNG format [8], the image is compressed horizontal line by horizontal line with the option of first applying a two-dimensional filter that attempts to predict the value of a pixel given its surrounding pixels. A good prediction leads to small output values, thus biasing the distribution toward small values, which can then be compressed well.

Compared to lossless compression, lossy compression can achieve dramatically greater compression ratios. This is because the compressor can discard information that the casual observer will not notice. The point of depart for lossy compression algorithms are assumptions on the physiological capabilities of the (human) observer. For instance, one of the design considerations in the baseline JPEG format [32] was that the human eye is more sensitive to rapid variations in brightness than to color and therefore the image is transformed to the YCrCb color space and the different channels quantized to different precision. Another assumption of baseline JPEG is that high-frequency information is less important than low-frequency information. Therefore the channels of the quantized image are transformed in blocks of 8×8 pixels into a frequency representation using the discrete cosine transform (DCT); the frequencies are scaled by hand-tuned coefficients and further scaled linearly to accommodate varying quality settings. Finally, the data resulting from the previous steps is compressed using Huffman compression, just as in lossless image compression. See Figure 3.1 for examples of JPEG compression.

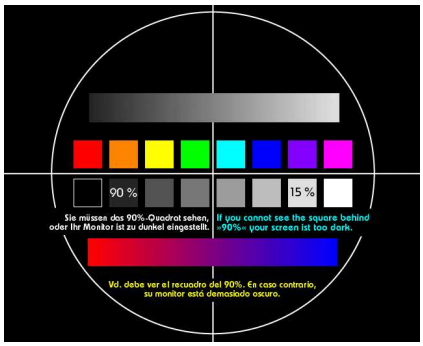
3.1.3 Compression Image Descriptors

The preceding discussion suggests that the compressed size of an image will measure its local complexity of composition. Indeed, an image descriptor computed by quantizing the input image to 256 colors, applying lossless compression to the PNG format [8] and using the

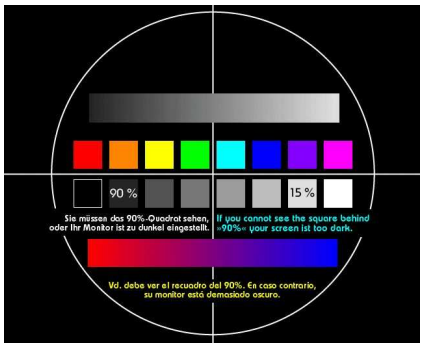
$$\text{compression ratio} = \frac{\text{compressed size}}{\text{original size}}$$

as a single real-valued image descriptor led to high predictive power for distinguishing cartoons from photographic images. Results of the experiments are given in Chapter 5. We included the quantization step in order to avoid problems with compressing the inherent noise in the source material from TREC-2002.

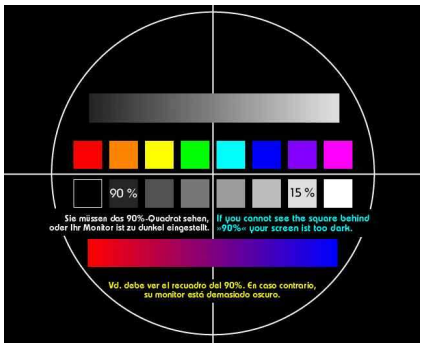
Experiments with lossy compression with JPEG at various quality settings gave little correlation. This is probably due to our source material, which was captured from TV signals (blurring sharp edges and adding noise



(a)



(b)



(c)

Figure 3.1: JPEG image-compression examples: (a) original of a monitor test image. (b) the same image compressed at a high quality setting—differences are hardly visible. (c) the same image compressed with low quality and high compression ratio—the deterioration in fidelity apparent: artifacts and smearing are clearly visible.

to uniform colors) and already stored in the JPEG format—hence, the distinctive features of cartoons were smoothed out and lost.

3.2 Pattern Spectrum

3.2.1 Introduction to morphological image analysis

Mathematical morphology (MM) can be defined not only as a theory for the analysis of spatial structures, but also as a powerful image analysis technique. It is called morphology because it aims at analyzing the shape and form of objects. It is mathematical in the sense that the analysis is based on set theory, integral geometry and lattice algebra. MM has an increasing success because of its simple mathematical description and the many powerful image analysis tools it provides. Each object can be characterized using morphological measurements. Image measurements aim at characterizing the objects of an image by some numerical values. The measurement is discriminant for a given criterion if the values obtained for objects satisfying this criterion are very different from those obtained for all other objects. These measurements define a vector of features that can be used as input to a statistical or neuronal classification method and this morphological approach is what we use. Our multi-scale pattern spectrum descriptors are the output of morphological measurements of the image after applying *granulometries*. Granulometries are explained later in Subsection 3.2.5 after giving a glossary of terms necessary for their better understanding.

3.2.2 Background notions

Discrete images

Most image analysis technologies use digital image data. The discrete version Z^2 of the 2-dimensional $(2-D)$ Euclidean space R^2 is achieved by sampling R^2. A network of evenly distributed points (pixels) is usually considered. The shape of the sampled object depends on the positioning of the sampling grid and the size of the sampling window.

Images are defined over a rectangular frame called the *definition domain* of the image.

A *binary image* f is a mapping of a subset D_f of Z^n called the definition domain of f into the set $\{0,1\}$:

$$f : D_f \subset Z^n \longrightarrow \{0,1\}.$$

The value of a pixel of a binary image is either 1 or 0 depending on whether the pixel belongs to an object or to its background. In morphology, image objects as considered as *sets*.

A *grayscale image* f is a mapping of a subset D_f of Z^n called the definition domain of f into a bounded set of finite chain of nonnegative integers N_0 :

$$f : D_f \subset Z^n \longrightarrow \{0, 1, \ldots, t_{max}\},$$

where t_{max} is the maximum value of the data type used for storing the image (i.e., $2^n - 1$ for pixels coded in n bits). More formally, grayscale images are considered as sets through their graphs and subgraphs. The *graph* G of an image f is the set of points (x, t) such that x belongs to the image plane of f and $t = f(x)$:

$$G(f) = \{(x, t) \in Z^n \times N_0 | t = f(x)\}.$$

The *subgraph*, SG of an image, f is the set of points of $Z^n \times N_0$ lying below the graph of the image and over the image plane:

$$SG(f) = \{(x, t) \in Z^n \times N_0 | 0 \leq t \leq f(x)\}.$$

Image to image transformations

Morphological image transformations are *image* to *image transformations*, i.e., the transformed image has the some definition domain as the input image and it is still a mapping (ψ) of this definition domain into the set of nonnegative integers. The *Identity transform* (I), is a trivial example of image to image transformation:

$$\forall f, I(f) = f.$$

The *cross-section* ($CS_t(f)$) of a grayscale image f at level t is the set of pixels of the image whose values are greater than or equal to t. Morphological transformations are *neighborhood image transformations*, i.e., the output value at a given pixel is a function of the values of the pixels falling within a neighboring region centered on the considered pixel.

Basic set operators applied to images

The basic set operators used for defining morphological transformations are the *union* $\cup$ and the *intersection* $\cap$. For grayscale images, the union becomes the *point-wise maximum* operator and the intersection is replaced by the *point-wise minimum* operator. The point-wise maximum $\vee$ and point-wise

minimum $\wedge$ between two images f and g with identical definition domains are defined for each point x as follows:

$$(f \vee g)(x) = \max[f(x), g(x)],$$
$$(f \wedge g)(x) = \min[f(x), g(x)].$$

Another basic set operator is *complementation*, denoted by $\mathbf{C}$: $\mathbf{C}(f) = f^c$. The complementation f^c of an image f is defined for each image x as the maximum value of the data type used for storing the image minus the value of the image f at position x:

$$f^c(x) = t_{max} - f(x).$$

Ordering relations

Ordering is a key notion in mathematical morphology. An image f is less than or equal to an image g with the same definition domain if the subgraph of f is included in that of g:

$$f \leq g \Leftrightarrow SG(f) \subseteq SG(g).$$

By analogy ordering on image transformations is defined as: a transformation ψ_1 is less than or equal to a transformation ψ_2 if and only if, for all images f, $\psi_1(f)$ is less than or equal to $\psi_2(f)$:

$$\psi_1(f) \leq \psi_2(f) \Leftrightarrow \forall f, \psi_1(f) \leq \psi_2(f).$$

Convexity

A Euclidean set is *convex* if and only if it contains all line segments connecting any pair of its points. The convexity property is an important *shape* descriptor. In the discrete case there may be more than one connected digital line segment linking two points, hence, the definition becomes:

A set of grid nodes S is said to be convex if it is equivalent to the grid nodes falling within the intersection of all Euclidean half-planes containing S. The smallest convex set containing a given set is called the *convex hull* of this set.

Image transformation properties

Some of the most fundamental problems in image analysis concern the choice of which operators to use, when to apply them, and how large they should be.

Knowing the properties of a transformation allows us to predict its behavior and hence will help us to choose the appropriate transformations when solving an image analysis problem. Key image transformation properties in mathematical morphology are:

Idempotence: A transformation Ψ is idempotent if applying it twice to any image f is equivalent to applying it only once:

$$\psi \text{ is idempotent } \Leftrightarrow \psi\psi = \psi.$$

Extensivity: A transformation Ψ is extensive, if for all images f, the transformed image is greater than or equal to the original image, i.e., if ψ is greater than or equal to the identity transform I:

$$\psi \text{ is extensive } \Leftrightarrow I \leq \psi.$$

Anti-extensivity: A transformation Ψ is anti-extensive, if for all images f, the transformed image is less than or equal to the identity transform I:

$$\psi \text{ is anti-extensive } \Leftrightarrow I \geq \psi.$$

Increasingness: A transformation Ψ is increasing, if it preserves the ordering relation between images:

$$\psi \text{ is increasing } \Leftrightarrow \forall f, g \Rightarrow \psi(f) \leq \psi(g).$$

Duality: Two transformations Ψ and Φ are dual with respect to complementation if applying Ψ to an image is equivalent to applying ϕ to the complement of the image and taking the complement of the result:

$$\psi \text{ and } \phi \text{ are dual with respect to complementation } \mathbf{C} \Leftrightarrow \psi = \mathbf{C}\phi\mathbf{C}.$$

De Morgan formula: $X \cap Y = (X^c \cup Y^c)^c$ - the set intersection with respect to the complementation is the set union, is an example of a dual operator.

3.2.3 Erosion and Dilation

Morphological operators aim at extracting relevant structures of the image through its subgraph representation. This is achieved by probing the image with another set of known shape called *structuring element* (SE)- a movable image mask within which the image values are evaluated. The shape of the SE is usually chosen according to some *a priori* knowledge about the geometry of the relevant and irrelevant (noise or objects we would like to suppress) image structures.

Structuring element

A SE is a small set used to probe the image under study. One of the pixels in the SE is the *reference point*, commonly, though not necessarily at the center. The operations proceed by placing the reference point at every pixel in the image. At each location the image value at the reference point is changed (or not) depending on the image values within the SE. To investigate the morphology of n-dimensional image objects, n-dimensional SEs (i.e., subsets of the image definition domain) can be used, referred to as *flat* SEs, or $n+1$-dimensional SEs called *nonflat* SEs. The shape of flat structuring elements does not depend on the scaling of the image gray levels. The grayscale values of nonflat SEs should have the same units and scalings as those of the input image. The two dual morphological operators *erosion* and *dilation* are the foundation of morphological analysis with structuring elements.

Erosion

The erosion of an image I by a flat structuring element B is denoted by ε_B and the eroded value at a given pixel x is the minimum value of the image in the window defined by the structuring element when its origin is at x:

$$[\varepsilon_B(f)] = \min_{b \in B} f(x+b)$$

In binary morphology erosion shrinks the input image. The following figure (Figure 3.2) illustrates the erosion of a binary image. The structuring element in this case is a square (3×3 array of pixels) and the reference point is at the center. The structuring element is moved across the source (left) image. If at any location a '0' appears within the structuring element, the value at the reference point in the destination (right) image is zero. If the structuring element is full of '1's then the reference point in the destination image is set to 1. Another way to look at it is to say that the reference point is set 1 if the structuring element fits entirely within the object. Small-scale protrusions and connections are removed by this process. A larger structuring element would have stripped off a thicker skin (and in this particular case caused our object to disappear).

Grayscale erosion with a flat disk shaped SE element will generally darken the image. Bright regions surrounded by dark regions shrink in size, and dark regions surrounded by bright regions grow in size. Small bright spots in images will disappear as they are eroded away down to the surrounding intensity value, and small dark spots will become larger spots. Figure 3.3 shows a vertical cross-section through a grayscale image and the effect of erosion using a disk shaped SE. Figure 3.4 – Figure 3.6 illustrate grayscale

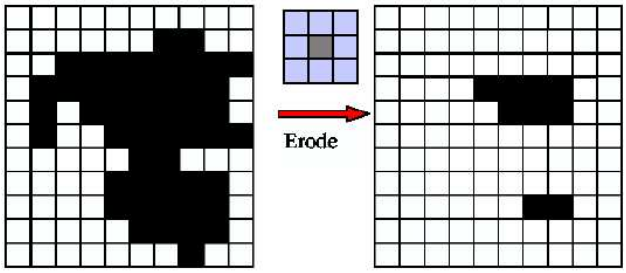


Figure 3.2: Binary Image Erosion.

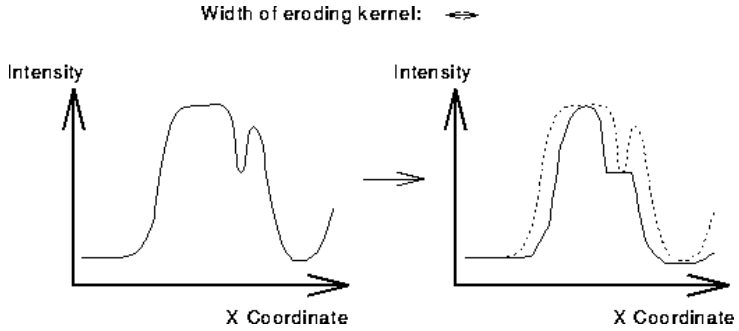


Figure 3.3: Grayscale erosion using a disk shaped structuring element. The flat disk shaped structuring element causes small peaks in the image to disappear and valleys to become wider.

erosion using a 3×3 flat square structuring element.

Nonflat SEs have grayscale values for their domain of definition. The erosion by a nonflat SE B is defined as follows:

$$[\varepsilon_B(f)] = \min_{b \in B}\{f(x+b) - B(b)\}$$

Dilation

Dilation is the opposite of erosion. We now use

$$[\delta_B(f)](x) = \max_{b \in B} f(x+b).$$

In other words, the dilated value at a given pixel x is the maximum value of the image in the window defined by the structuring element when its origin is at x.

In the morphological dilation, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image.

Rule for Binary Dilation
The value of the output pixel is set to 1 if any of the pixels in the input pixel's neighborhood is set to 1.

Figure 3.7 illustrates the dilation of a binary image. It shows how the neighborhood of the pixel of interest (the circled one) is defined by the structuring element. The dilation function applies the appropriate rule to the pixels in the neighborhood and assigns a value to the corresponding pixel in the output image.

In binary morphology dilation grows the input image (Figure 3.8). In this case, as the structuring element (3×3) moves, if any pixel in the source image has the value '1', then the reference point in the destination image is set to 1. The effect is to add a layer of pixels around the object. This results in filling in 'small' (with size of the structuring element or less) holes and gaps in the structure, and joining objects together that are separated by small distances.

Rule for Grayscale Dilation
The value of the output pixel is the *maximum* value of all the pixels in the input pixel's neighborhood.

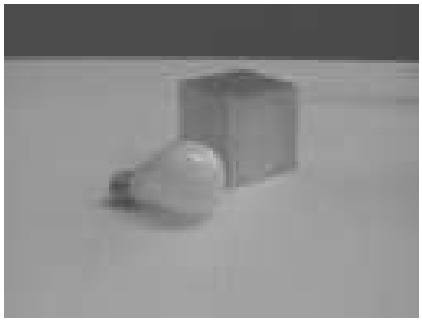


Figure 3.4: Original image.

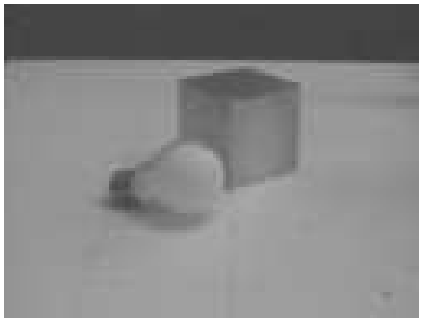


Figure 3.5: Produced image by two erosion passes using a 3×3 flat square structuring element. Note that the highlights have disappeared, and that many of the surfaces seem more uniform in appearance due to the elimination of bright spots. The body of the cube has grown in size since it is darker than its surroundings.

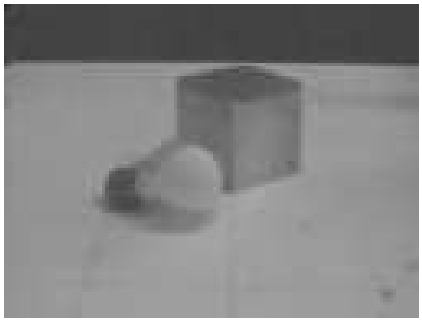


Figure 3.6: The effect of five passes of the same erosion operator on the original image.

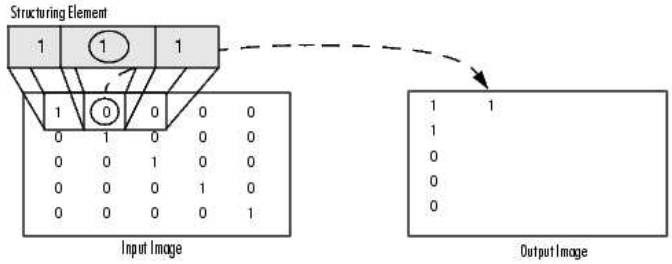


Figure 3.7: Morphological Processing of a Binary Image. The morphological dilation function sets the value of the output pixel to 1 because of one of the elements in the neighborhood defined by the structuring element is on.

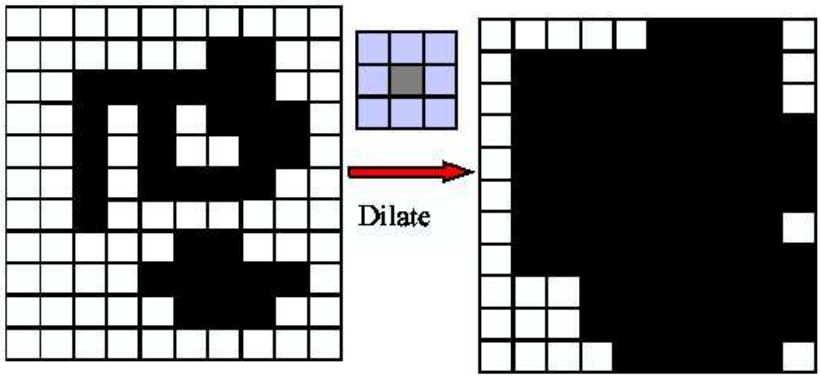


Figure 3.8: Binary Image Dilation. The dilation with 3×3 SE fills most holes and gaps, but the gap between the objects at the right hand side of the image, where it is larger, is retained, but narrowed, in the destination image.

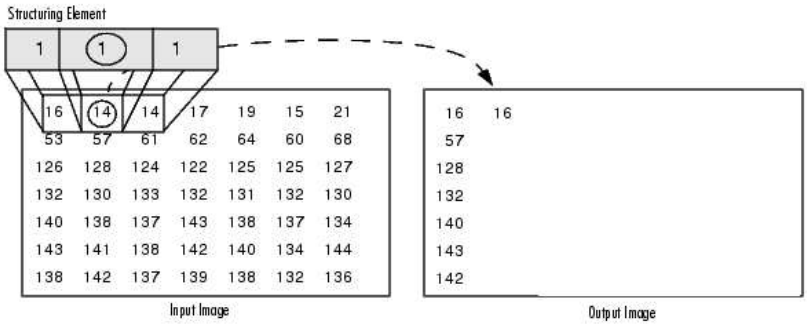


Figure 3.9: Morphological Processing of grayscale Images.

Figure 3.9 illustrates morphological processing for a grayscale image. It shows the processing of a particular pixel in the input image. The function applies the rule to the input pixel's neighborhood and uses the highest value of all the pixels in the neighborhood as the value of the corresponding pixel in the output image.

Grayscale dilation with a flat disk shaped structuring element will generally brighten the image. Bright regions surrounded by dark regions grow in size, and dark regions surrounded by bright regions shrink in size. Small dark spots in images will disappear as they are 'filled in' to the surrounding intensity value. Small bright spots will become larger spots. Figure 3.10 shows a vertical cross-section through a grayscale image and the effect of dilation using a disk shaped structuring element. Figure 3.11 – Figure 3.13

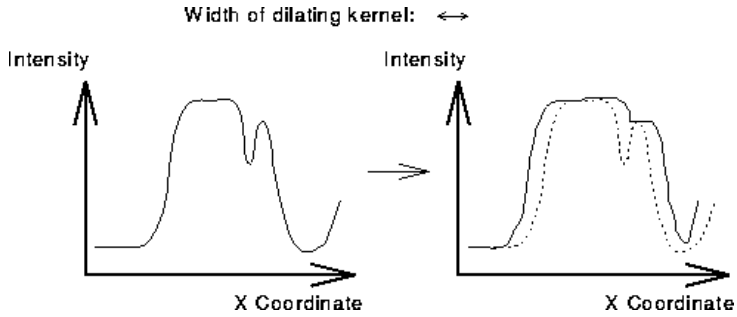


Figure 3.10: Grayscale dilation using a disk shaped structuring element.

illustrate grayscale dilation using a 3×3 flat square structuring element.

Nonflat SEs have grayscale values for their domain of definition. The dilation by a nonflat SE B is defined as follows:

$$[\delta_B(f)] = \max_{b \in B}\{f(x+b) + B(b)\}$$

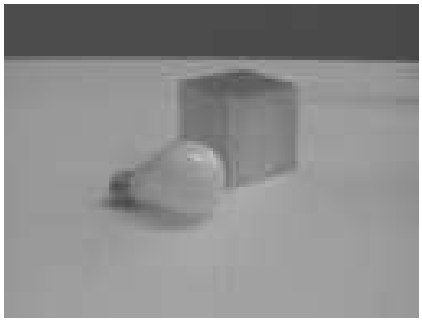


Figure 3.11: Original image.

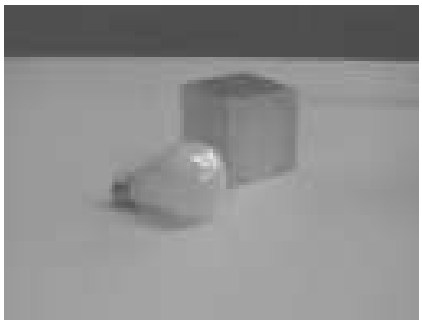


Figure 3.12: Produced image by two dilation passes using a 3×3 flat square structuring element. The highlights on the bulb surface have increased in size and have also become squared off as an artifact of the structuring element shape. The dark body of the cube has shrunk in size since it is darker than its surroundings, while within the outlines of the cube itself, the darkest top surface has shrunk the most. Many of the surfaces have a more uniform intensity since dark spots have been filled in by the dilation.

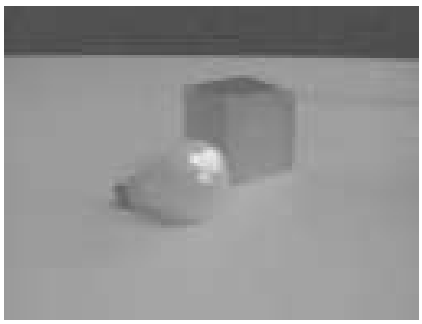


Figure 3.13: The effect of five passes of the same dilation operator on the original image.

Some properties

- **Duality:** The dilation and erosion are dual transformations with respect to complementation, i.e., any erosion of an image is equivalent to a complementation of the dilation of the complemented image with the same structuring element (and vice-versa):

  $$\varepsilon_B = C\delta_B C$$

  The erosion *shrinks* the objects but *expands* their background and vice-versa for the dilation.

- **Ordering relations**

  $$\varepsilon_B \leq \delta_B \Leftrightarrow B \text{ contains its origin.}$$

- **Erosions and dilations are invariant to translations.**

- **Erosions and dilations are increasing transformations:**

  $$f \leq g \Rightarrow \varepsilon(f) \leq \varepsilon(g),\ \delta(f) \leq \delta(g).$$

3.2.4 Opening

Morphological *opening* γ of an image f by a structuring element B is an erosion of f by B followed by the dilation with the transposed SE $\hat{B}$:

$$\gamma_B(f) = \delta_{\hat{B}}[\varepsilon_B(f)],$$

i.e., $\gamma_B = \delta_{\hat{B}}\varepsilon_B$.

The opening of an image is independent from the origin of the SE. Erosion and dilation cause the removal of small-scale structures and gaps respectively, but result in the remaining objects shrinking or growing. Opening results in the same 'cleaning' effect on the images while retaining objects at their original size and shape (approximately). Figure 3.14 shows a 'smoothing' effect of the opening process.

A grayscale opening consists simply of a grayscale erosion followed by a grayscale dilation. The grayscale opening can similarly be used to select and preserve particular intensity patterns while attenuating others. As a simple example see Figures 3.15 – 3.16.

- Morphological opening is an increasing transformation:

  $$f \leq g \Rightarrow \gamma(f) \leq \gamma(g).$$

- Morphological opening is an idempotent transformation:

  $$\gamma\gamma = \gamma.$$

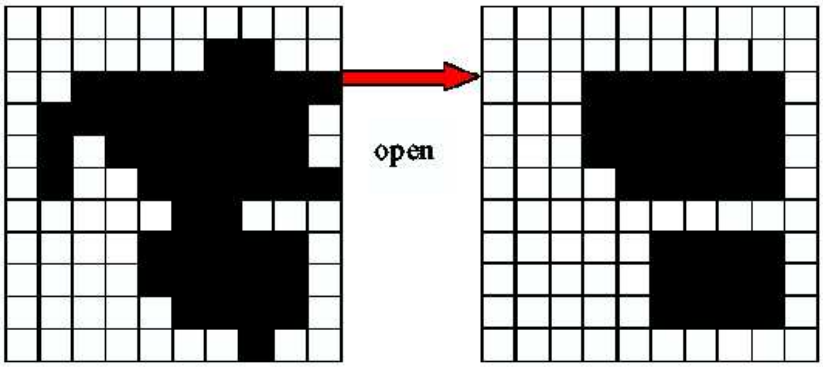


Figure 3.14: Binary Image Opening. The opening with 3×3 SE ends up with object of the same size as in the original case, and approximately the same shape, with small scale structure removed.

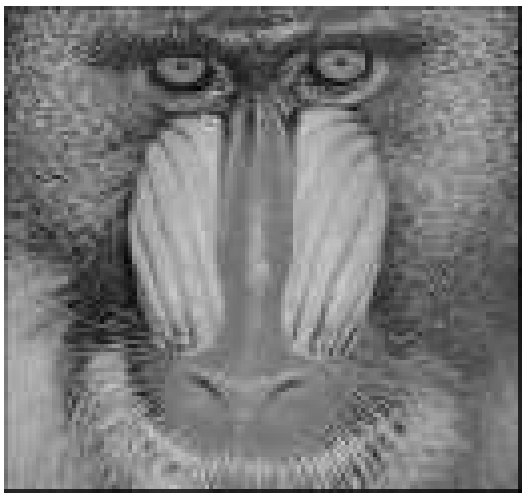


Figure 3.15: Original image.

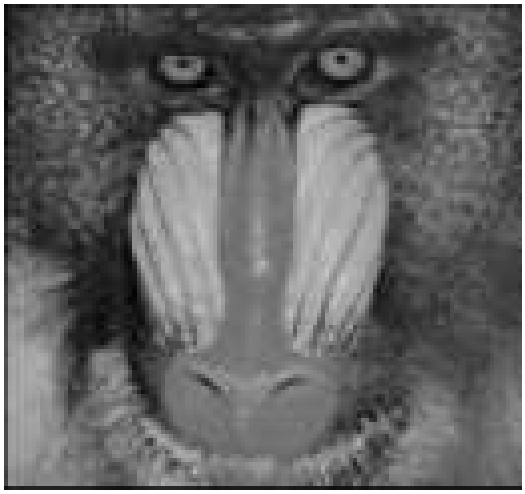


Figure 3.16: Produced image by grayscale opening with a flat 5×5 square structuring element. The effect of the opening is that bright features smaller than the structuring element have been greatly reduced in intensity, while larger features have remained more or less unchanged in intensity. Thus the fine grained hair and whiskers in the image have been much reduced in intensity, while the nose region is still at much the same intensity as before. The image does have a more mat appearance than before since the opening has eliminated small peculiarities and texture fluctuations.

3.2.5 Granulometries

Granulometry is a widely used tool in mathematical morphology for determining the size distribution of objects in an image without explicitly segmenting each object first. Intuitively, granulometry treats image objects as particles whose sizes can be established by sifting them through sieves of increasing mesh width, and collecting what remains in the sieve after each pass. One pass of sifting and retaining the residue is analogous to the morphological opening of an image using a structuring element of a certain size. The areas of the images occupied at different scales can be determined by successively opening the binary image with larger and larger structuring elements. At each scale objects corresponding to the size of the structuring element will be removed from the image and the detected area will decrease. By measuring the loss of area at each scale we can obtain a graph which shows the relative area as a function of scale (structuring-element size). Figure 3.17 shows a collection of objects of different sizes and the size of the structuring element that causes different components to disappear due to an opening operation. The graph at the bottom illustrates the form of output resulting from such an operation.

In mathematical terms, a granulometry is defined by a transformation Φ_r with size parameter r that satisfies the *Anti-extensivity*, *Increasingness* (as defined in 3.2.2) and *Absorption* ($\Phi_r\Phi_\upsilon = \Phi_\upsilon\Phi_r = \Phi_{max(r,\upsilon)}$) axioms [25].

Of particular interest are granulometries generated by openings by scaled versions of a single convex structuring element B, i.e., $\Phi_r(f) = f \circ rB$, where rB denotes the structuring element B at scale r. For values $r_1 < \cdots < r_k$, the normalized size distribution induced by the granulometry Φ_r is defined as

$$s(i) = 1 - \frac{1}{\sum_{x,y} f(x,y)} \sum_{x,y} [\Phi_{r_i}(f)](x,y)$$

and the corresponding *pattern spectrum* (loss of surface area between Φ_r and Φ_{r+1}) is

$$p(i) = s(i+1) - s(i).$$

3.2.6 Structuring Elements Parabola and Disk

There exist many applications that use flat structuring elements; e.g., rectangular size distributions are used in an effective way to characterize visual similarity of document images [2]. However, in our case of different photographic and cartoon images the objects do not have a particular geometric shape and therefore we need a generic and nonflat structuring element. Jackway [12] and van den Boomgaard et al. [30], and van den Boomgaard and

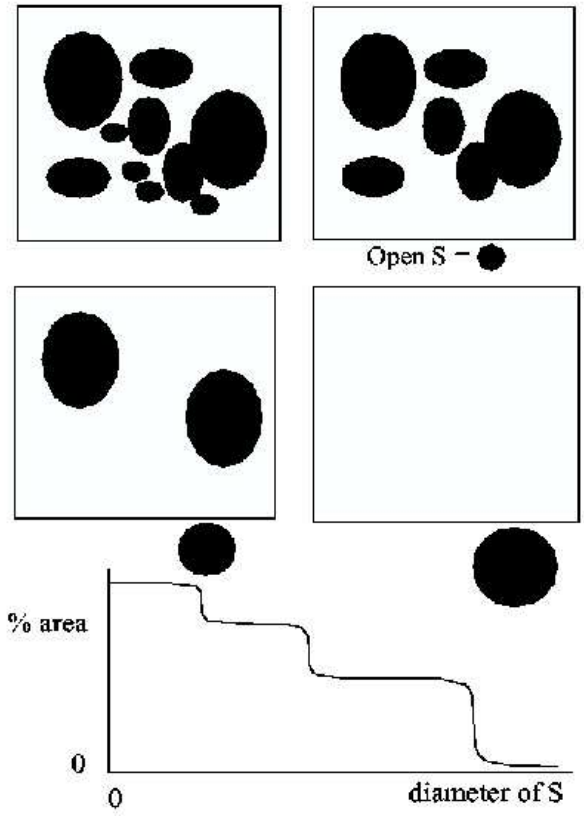


Figure 3.17: Binary Granulometry.

Smeulders [30] have shown that the Gaussian function used in linear convolutions has as morphological analogue the parabola. We decided to use a parabola (to be more precise, the volume delimited by the *paraboloid* which is the surface of revolution of the parabola) as one of the structuring elements because it is the unique structuring function that is both

- rotational symmetric, and
- dimensional decomposable [27].

The first property ensures that we have a generic structuring element that does not favor any particular shape in the image. The second property ascertains that we can compute precise openings quickly. Note that the flat disk structuring element does not have exact and efficient decompositions into smaller structuring elements. Even approximations to the disk do not yield dimensional decompositions—the structuring elements of the radial decomposition of the discrete disk are one–dimensional but not aligned with one of the axes (see Figure 3.22).

Formally, the two-dimensional parabola (*paraboloid*) of scale λ is

$$P_\lambda(x, y) = -\frac{x^2 + y^2}{\lambda^2}$$

Recall that dilation with a non-flat structuring element B is defined as

$$[\delta_B(f)] = \max_{b \in B}\{f(x + b) + B(b)\} \tag{3.1}$$

Since the domain of a parabola is infinite, the maximum ranges at first glance over an infinite set. However, since our grayscale images have pixel values between 0 and 1, we obtain the exact dilation with the (infinite) parabola of scale λ considering only the finite domain $\{(x, y) : P_\lambda(x, y) > -1\}$ for computing the maximum in eq. (3.1). Of course, the same holds analogously for erosion.

In the definition of P_λ, we choose the scale factor $1/(\lambda^2)$ so that the volume of the scale-λ paraboloid delimited by the plane $z = -1$,

$$V_P = \int_0^{-1} \pi \left\| P_\lambda^{-1}(z) \right\|^2 \mathrm{d}\, z = \pi\lambda^2 \int_0^{-1} z \,\mathrm{d}\, z = \frac{\pi\lambda^2}{2} \ ,$$

is the same as the the volume of the cylinder of radius-r and height 1 (which is the same as the area of the disk of radius-r),

$$V_B = \int_0^{-1} \pi \left\| B_r^{-1}(z) \right\|^2 \mathrm{d}\, z = 2\pi r^2 \int_0^{-1} z \,\mathrm{d}\, z = \pi r^2 \ ,$$

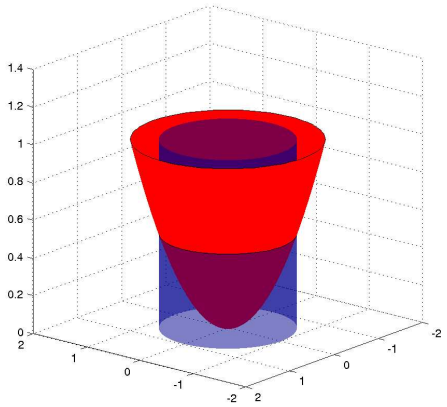


Figure 3.18: Parabola and Disk SE have the same volume

therefore

$$V_P = \pi\frac{\lambda^2}{2} = \pi r^2 = V_B \quad \text{and} \quad \lambda = r\sqrt{2}\ .$$

In this way, the parabola P_λ and the disk B_r of radius r have comparable size as structuring elements (Figure 3.18): when used in dilation of a single white point on a black background, the increase in brightness is the same for both structuring elements.

Dilation and erosion with a n-dimensional parabola can be decomposed into the dilation (and erosion, respectively) with n one-dimensional parabolae, reducing the complexity by a factor n [6]. This decomposition property allows us to compute dilation with P_λ efficiently using one-dimensional structuring elements (Figure 3.19–Figure 3.21).

We define the one-dimensional horizontal and vertical scale-λ parabolae, respectively:

$$H_\lambda(x,y) = \begin{cases} -x^2/(\lambda^2) & \text{for } y = 0 \\ -\infty & \text{for } y \neq 0 \end{cases}$$

and

$$V_\lambda(x,y) = \begin{cases} -y^2/(\lambda^2) & \text{for } x = 0 \\ -\infty & \text{for } x \neq 0 \end{cases}$$

Then dilation with P_λ is the same as a dilation with H_λ followed by a dilation with V_λ:

$$[\delta_{P_\lambda}(f)] = [\delta_{V_\lambda}(\delta_{H_\lambda}(f))]$$

Again, it suffices to consider the finite domains in which $H_\lambda > -1$ and $V_\lambda > -1$, and of course these considerations extend to erosion as well.

As disk approximations satisfying the absorption property lead to interesting granulometries as well, we use radial decompositions of discrete disks of increasing size from cascades of dilations with periodic lines along several directions (Figure 3.22 - Figure 3.23) to generate a granulometric function with size parameter given by the radius of the disk.

3.2.7 Pattern-Spectrum Image Descriptors

Based on the distinguishing features of cartoons like large patches of uniform color we expect differences between cartoons and photographs in the pattern spectrum: a peak in the pattern spectrum at a given size indicates that there are many objects of that size in the image. Hence, we store as image descriptors a 'small-scale parabola' pattern spectrum with $\lambda_i = i\sqrt{2}$, $i = 1, \ldots, 20$ (Figure 3.24 - Figure 3.27), a 'small-scale disk' pattern spectrum with $r_i = i$ (Figure 3.31 - Figure 3.33), $i = 1, \ldots, 20$, a 'large scale-parabola' pattern spectrum (Figure 3.28 - Figure 3.30) with $\lambda_i = i5\sqrt{2}$, $i = 1, \ldots, 10$ and a 'large scale-disk' (Figure 3.34 - Figure 3.36) pattern spectrum with $r_i = 5i$, $i = 1, \ldots 10$.

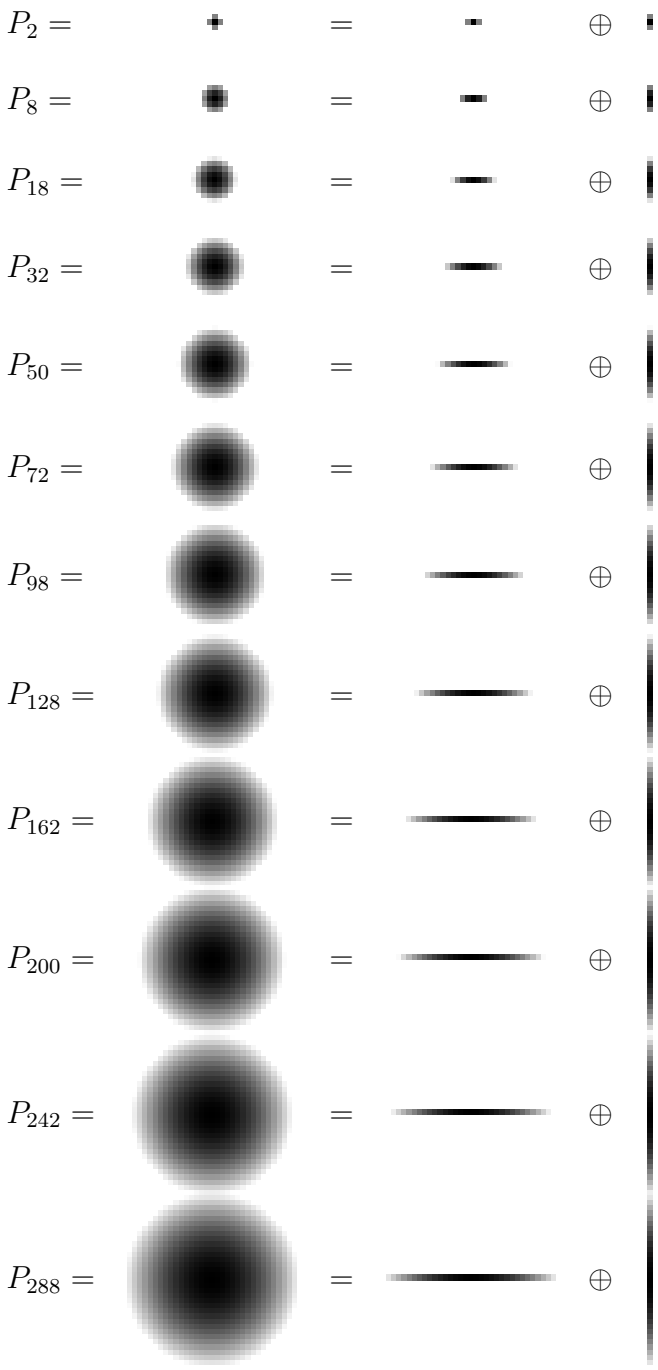


Figure 3.19: Examples of decomposition of the parabola structuring element

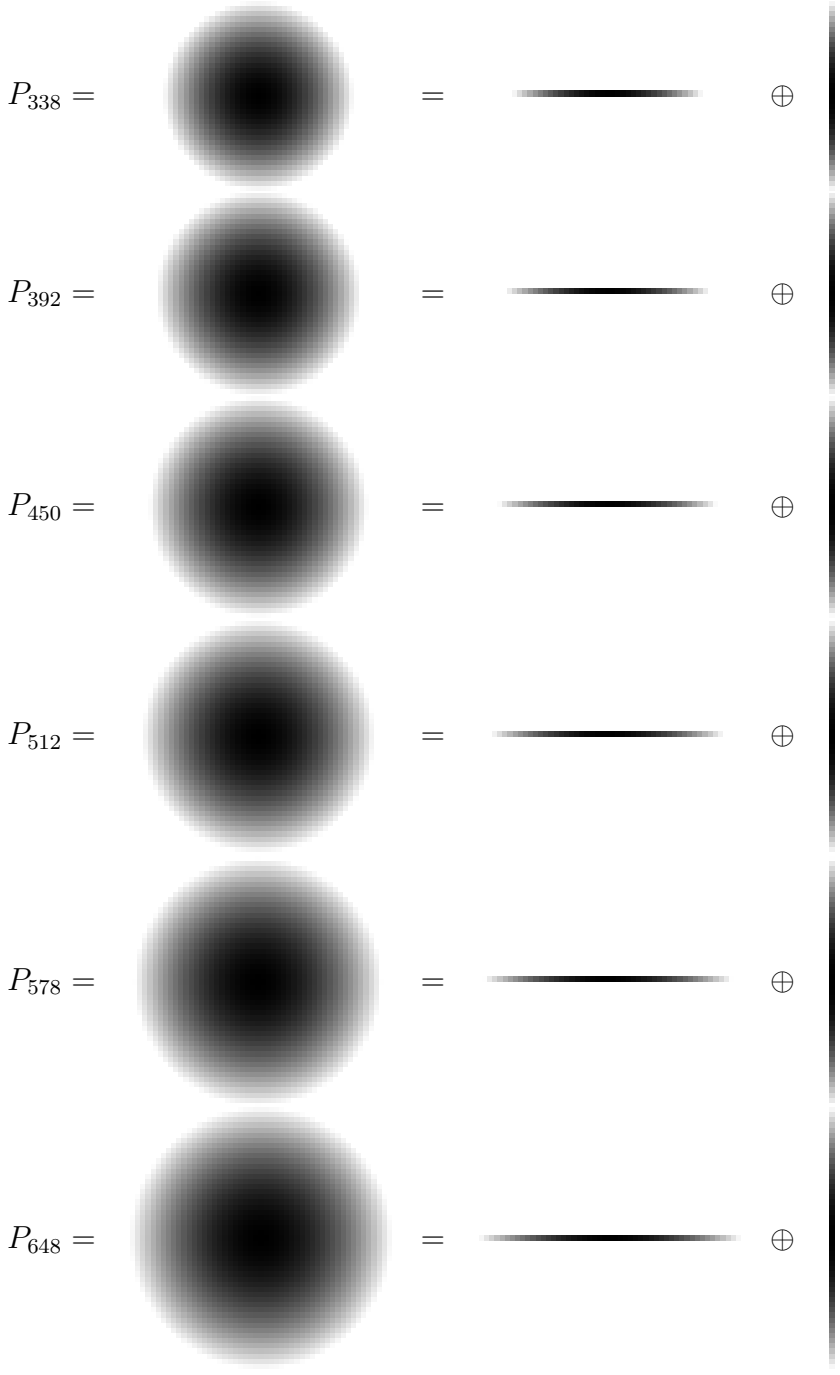


Figure 3.20: Examples of decomposition of the parabola structuring element

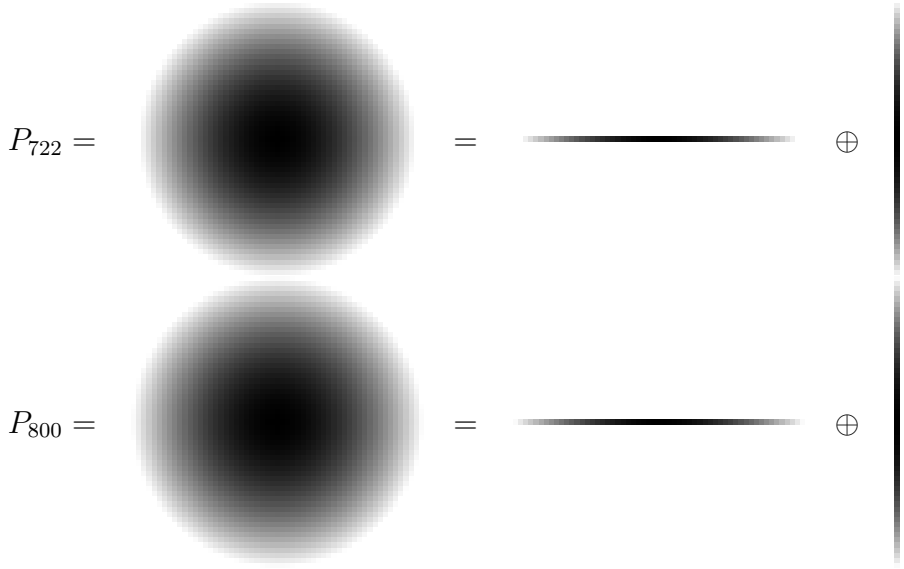


Figure 3.21: Examples of decomposition of the parabola structuring element

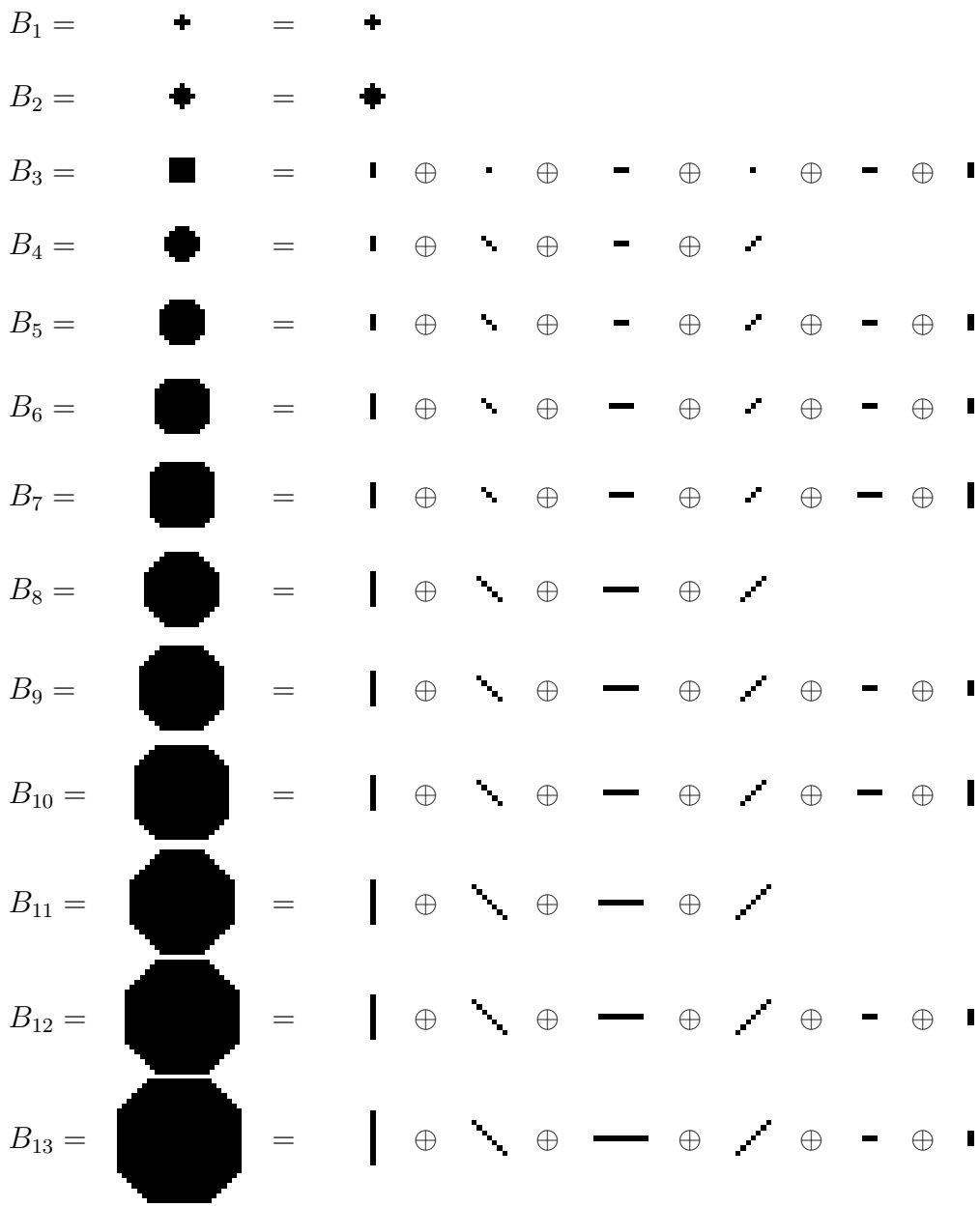


Figure 3.22: Examples of decomposition of the disk structuring element

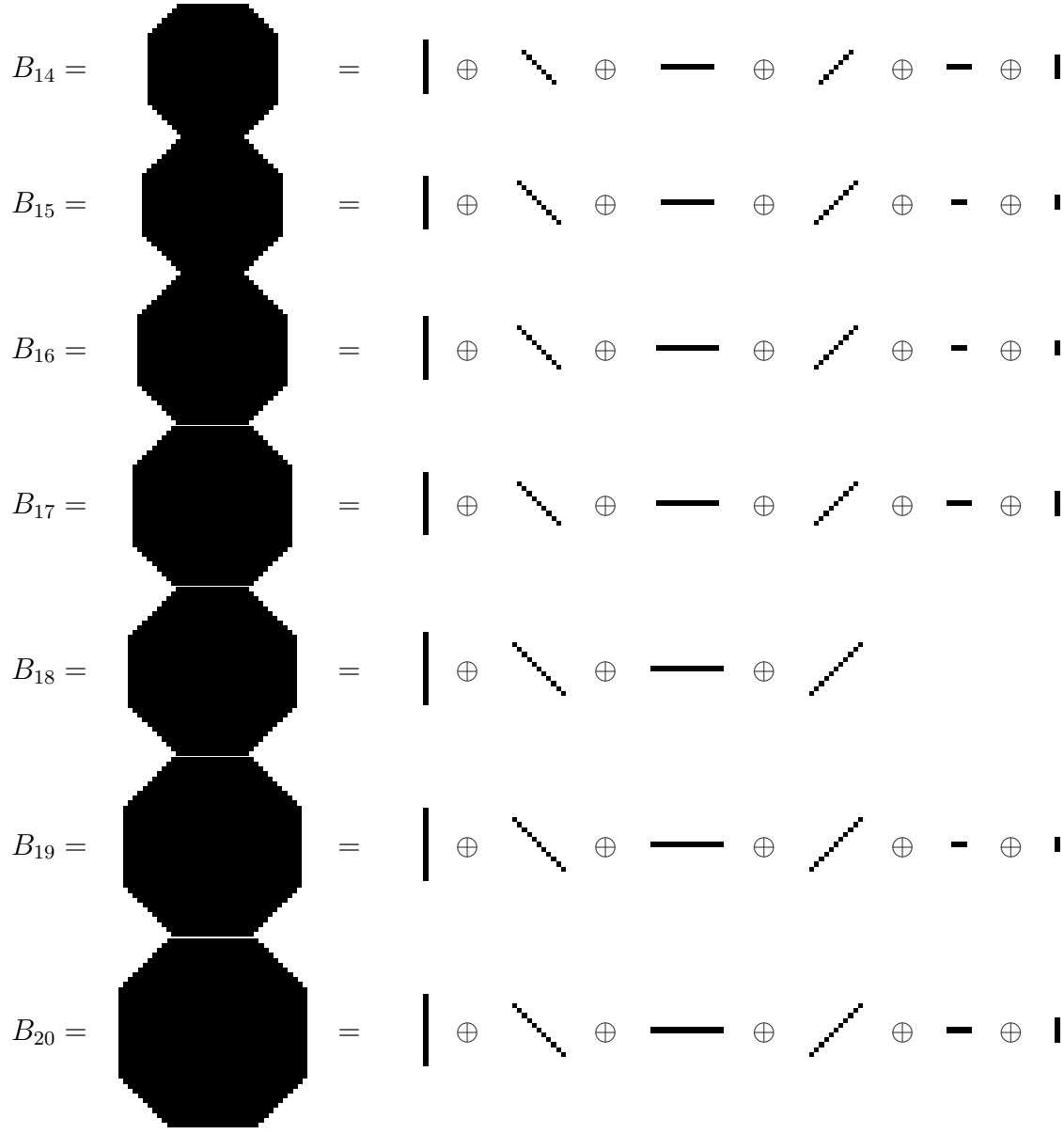
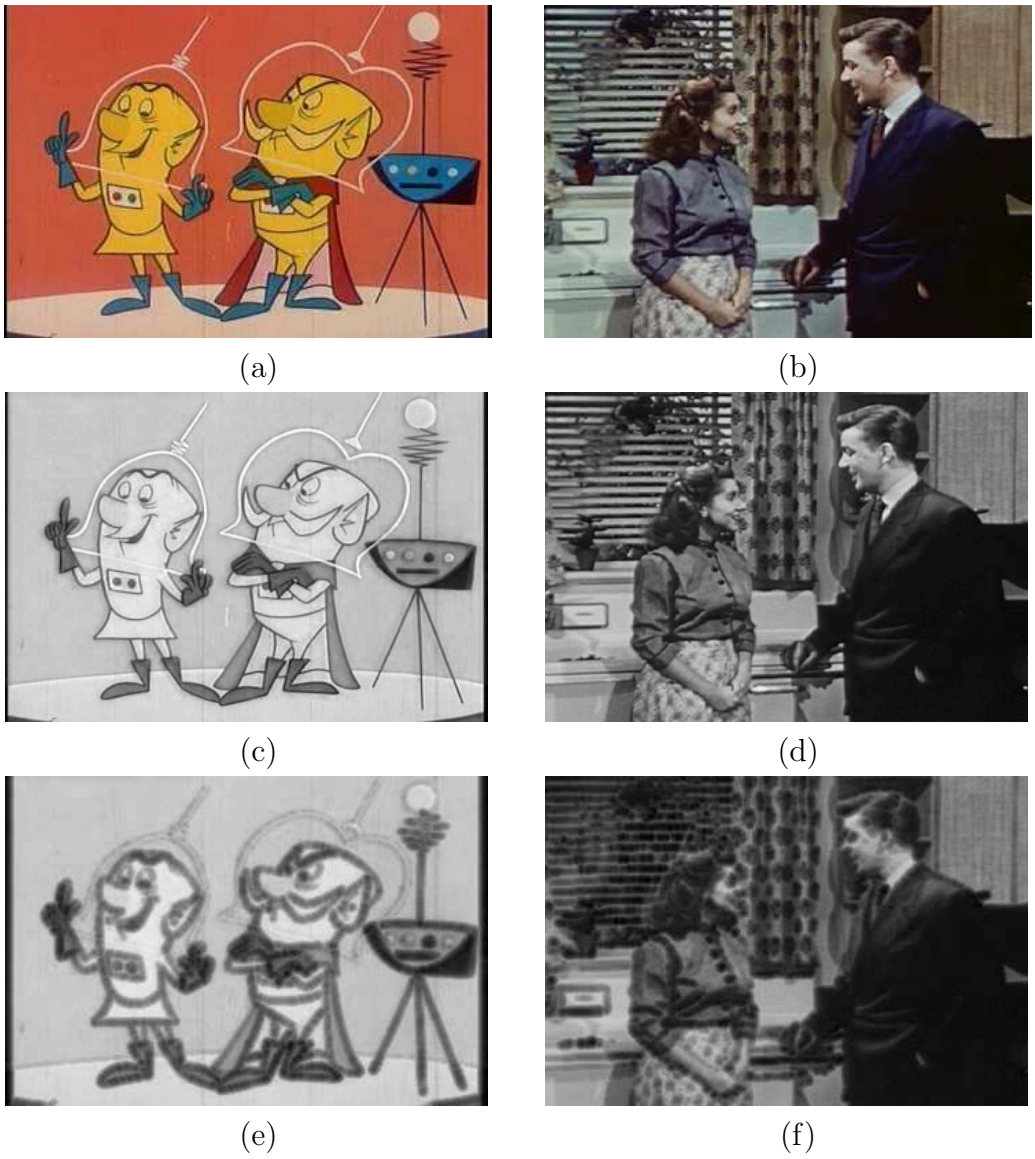


Figure 3.23: Examples of decomposition of the disk structuring element



(a) (b) (c) (d) (e) (f)

Figure 3.24: Process of obtaining the 20 small-scale parabola pattern spectrum descriptors. An original (RGB) (a) cartoon and (b) photographic image. Grayscale (c) cartoon and (d) photographic image. Eroded (e) cartoon and (f) photographic image with SE parabola, $\lambda_5 = 5\sqrt{2}$.

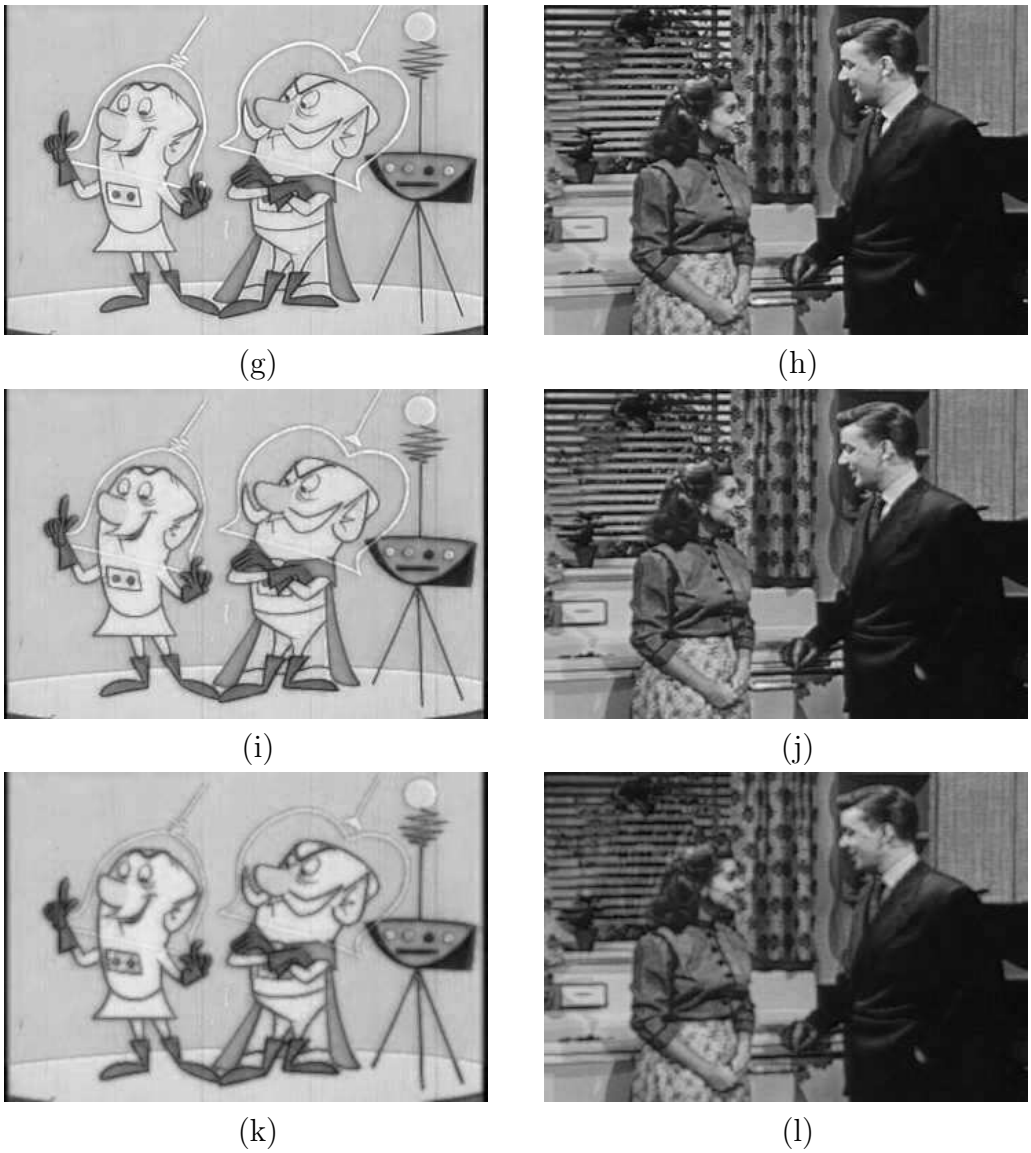
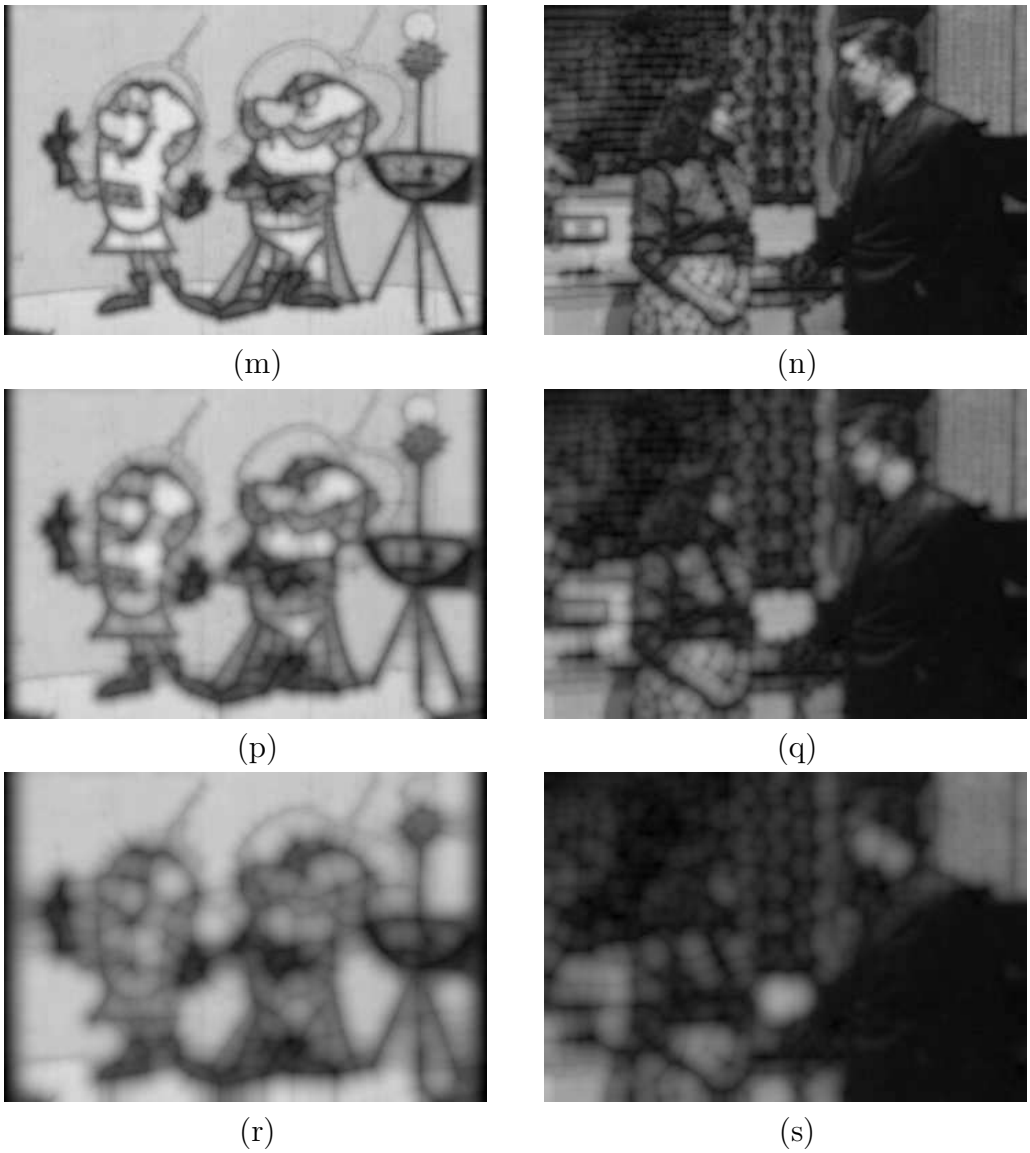


Figure 3.25: Continuation of small-scale parabola pattern spectrum descriptors example. Opened (g) cartoon and (h) photographic image with SE parabola, $\lambda_1 = \sqrt{2}$. Opened (i) cartoon and (j) photographic image with SE parabola, $\lambda_2 = 2\sqrt{2}$. Opened (k) cartoon and (l) photographic image with SE parabola, $\lambda_4 = 4\sqrt{2}$.



(m) (n)

(p) (q)

(r) (s)

Figure 3.26: Continuation of small-scale parabola pattern spectrum descriptors example. Opened (m) cartoon and (n) photographic image with SE parabola, $\lambda_8 = 8\sqrt{2}$. Opened (p) cartoon and (q) photographic image with SE parabola, $\lambda_{16} = 16\sqrt{2}$. Opened (r) cartoon and (s) photographic image with SE parabola, $\lambda_{20} = 20\sqrt{2}$.

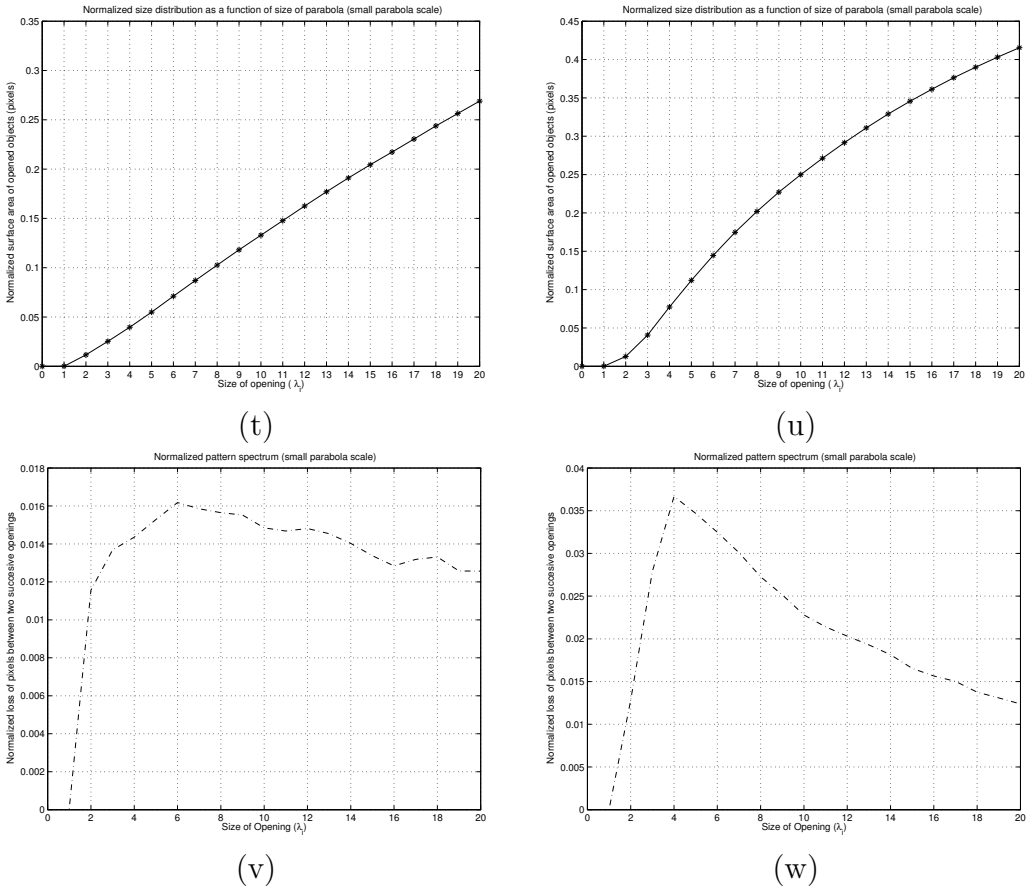


Figure 3.27: Continuation of small-scale parabola pattern spectrum descriptors example. Size distribution of the (t) cartoon and (u) photographic image with SE parabola, $\lambda_i = i\sqrt{2}$, $i = 1, \ldots, 20$. Derivative (v) of (t). Derivative (w) of (u).

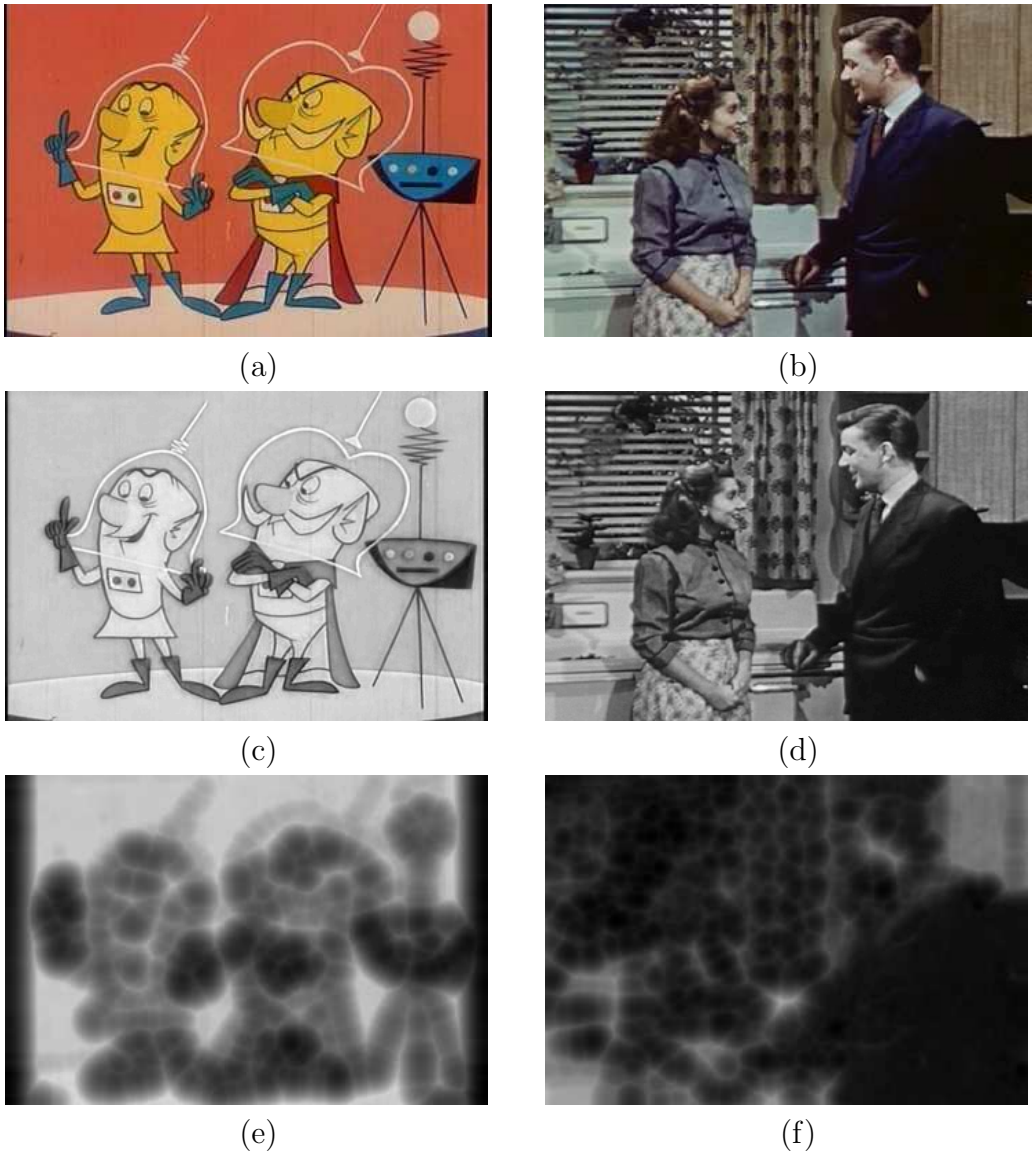


Figure 3.28: Process of obtaining the 10 large-scale parabola pattern spectrum descriptors. An original (RGB) (a) cartoon and (b) photographic image. Grayscale cartoon (c) and (d) photographic image (HSV color space). Eroded (e) cartoon and (f) photographic image with SE parabola, $\lambda_4 = 20\sqrt{2}$.

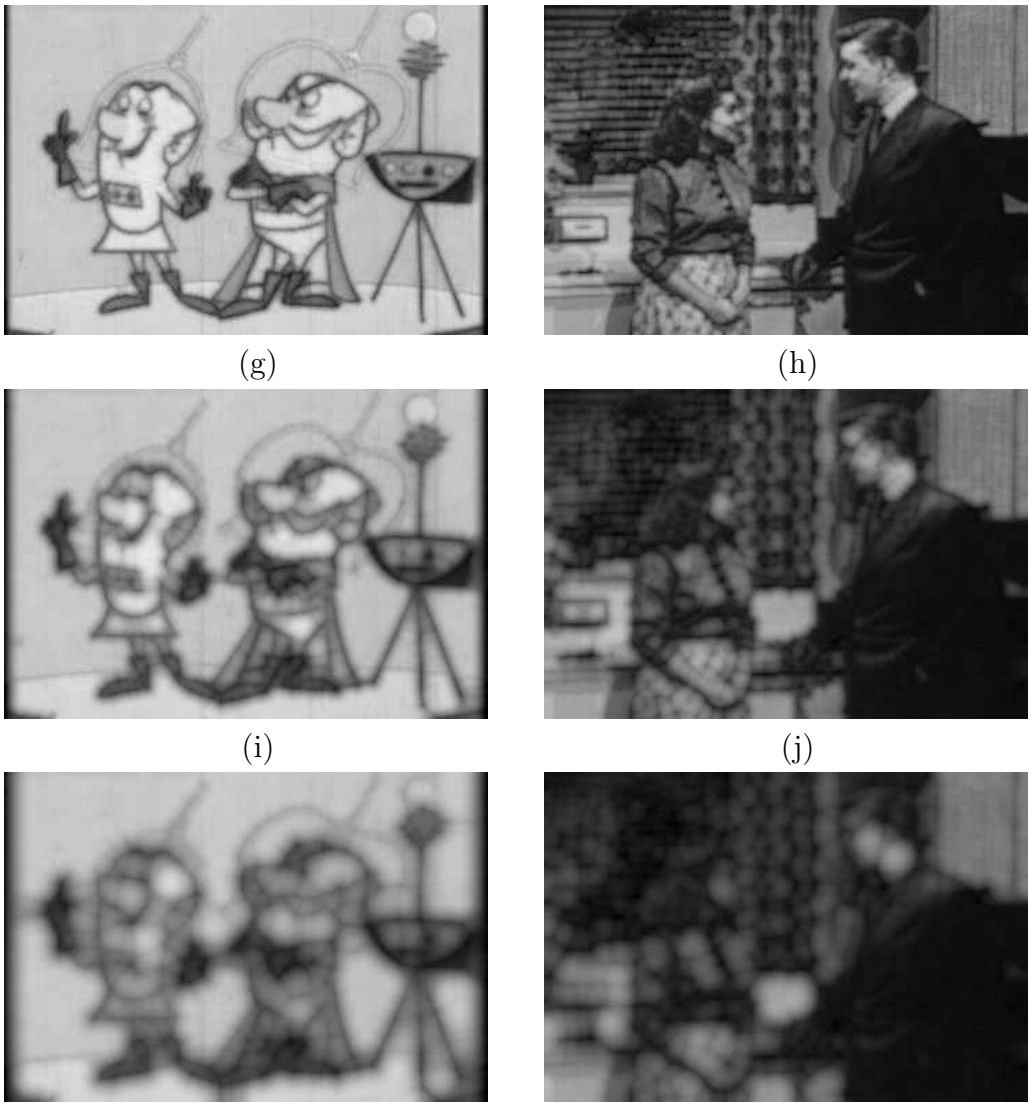


Figure 3.29: Continuation of large-scale parabola pattern spectrum descriptors example. Opened (g) cartoon and (h) photographic image with SE parabola, $\lambda_1 = 5\sqrt{2}$. Opened (i) cartoon and (j) photographic image with SE parabola, $\lambda_2 = 10\sqrt{2}$. Opened (k) cartoon and (l) photographic image with SE parabola, $\lambda_3 = 15\sqrt{2}$.

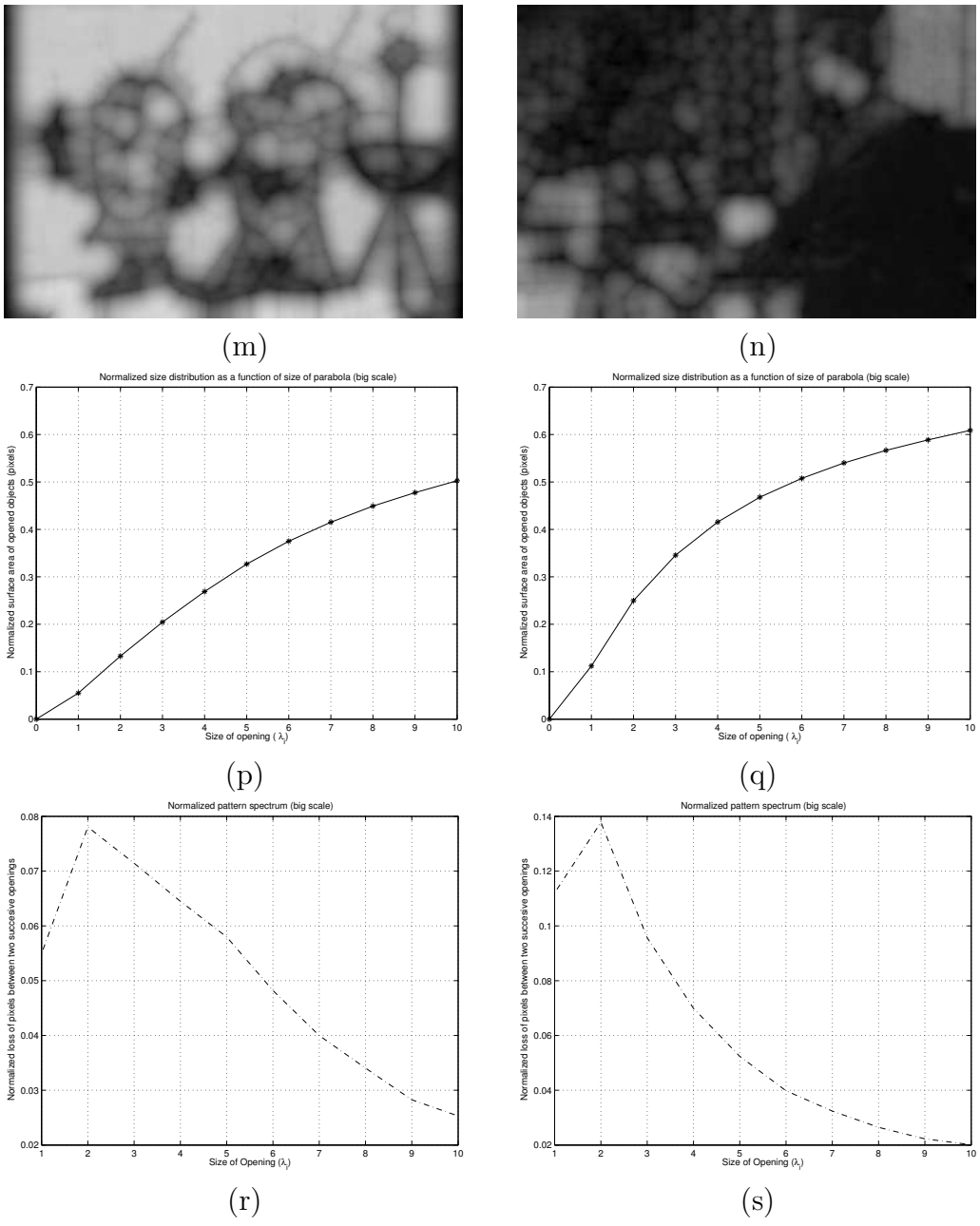


Figure 3.30: Continuation of large-scale parabola pattern spectrum descriptors example. Opened (m) cartoon and (n) photographic image with SE parabola, $\lambda_4 = 20\sqrt{2}$. Size distribution of the (p) cartoon and (q) photographic image with SE parabola, $\lambda_i = i5\sqrt{2}$, $i = 1, \ldots, 10$. Derivative (r) of (p). Derivative (s) of (q).

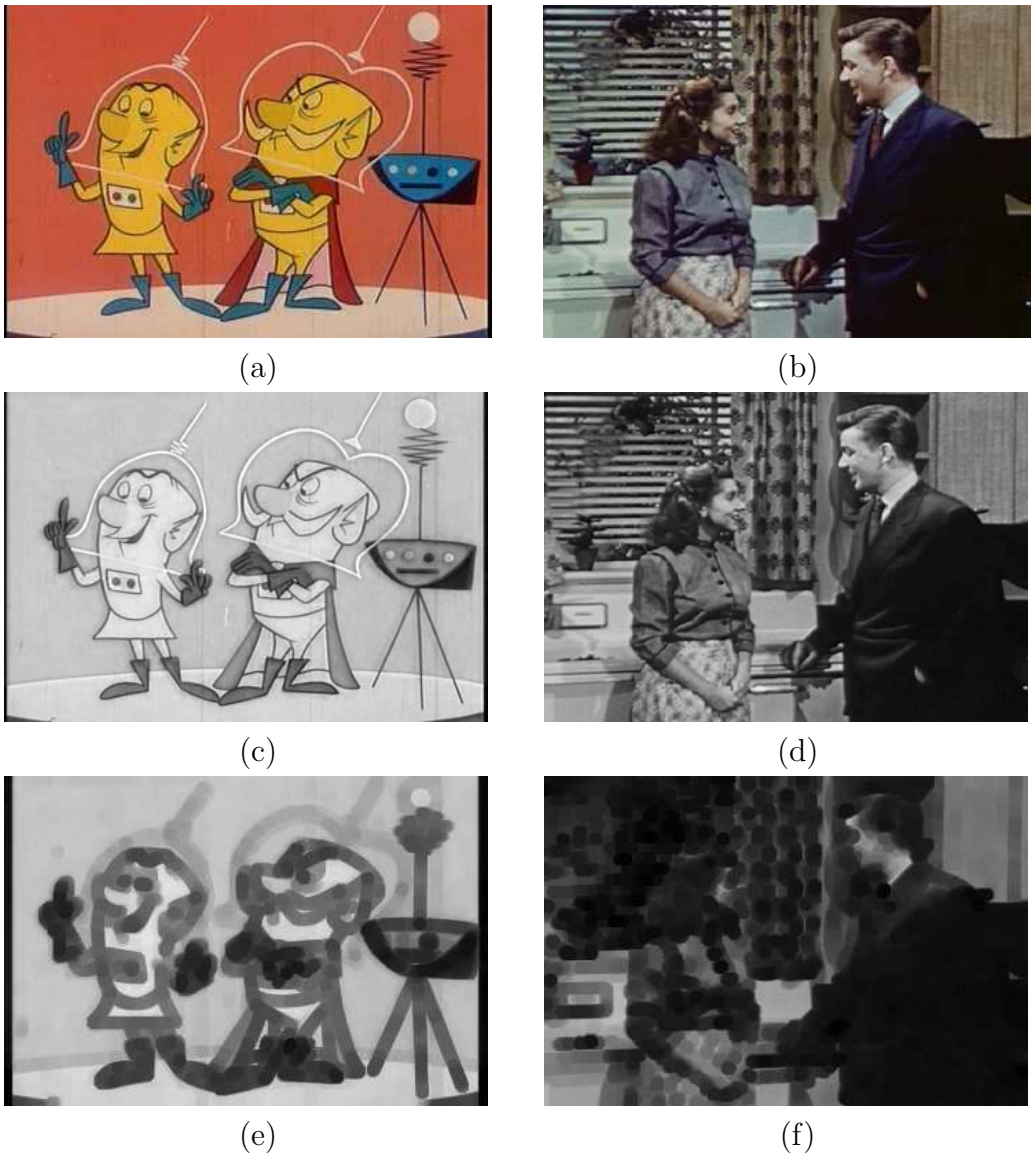


Figure 3.31: Process of obtaining the 20 small-scale disk pattern spectrum descriptors. An original (RGB) (a) cartoon and (b) photographic image. Grayscale (c) cartoon and (d) photographic image (HSV color space). Eroded (e) cartoon and (f) photographic image with SE disk, $r_5 = 5$.

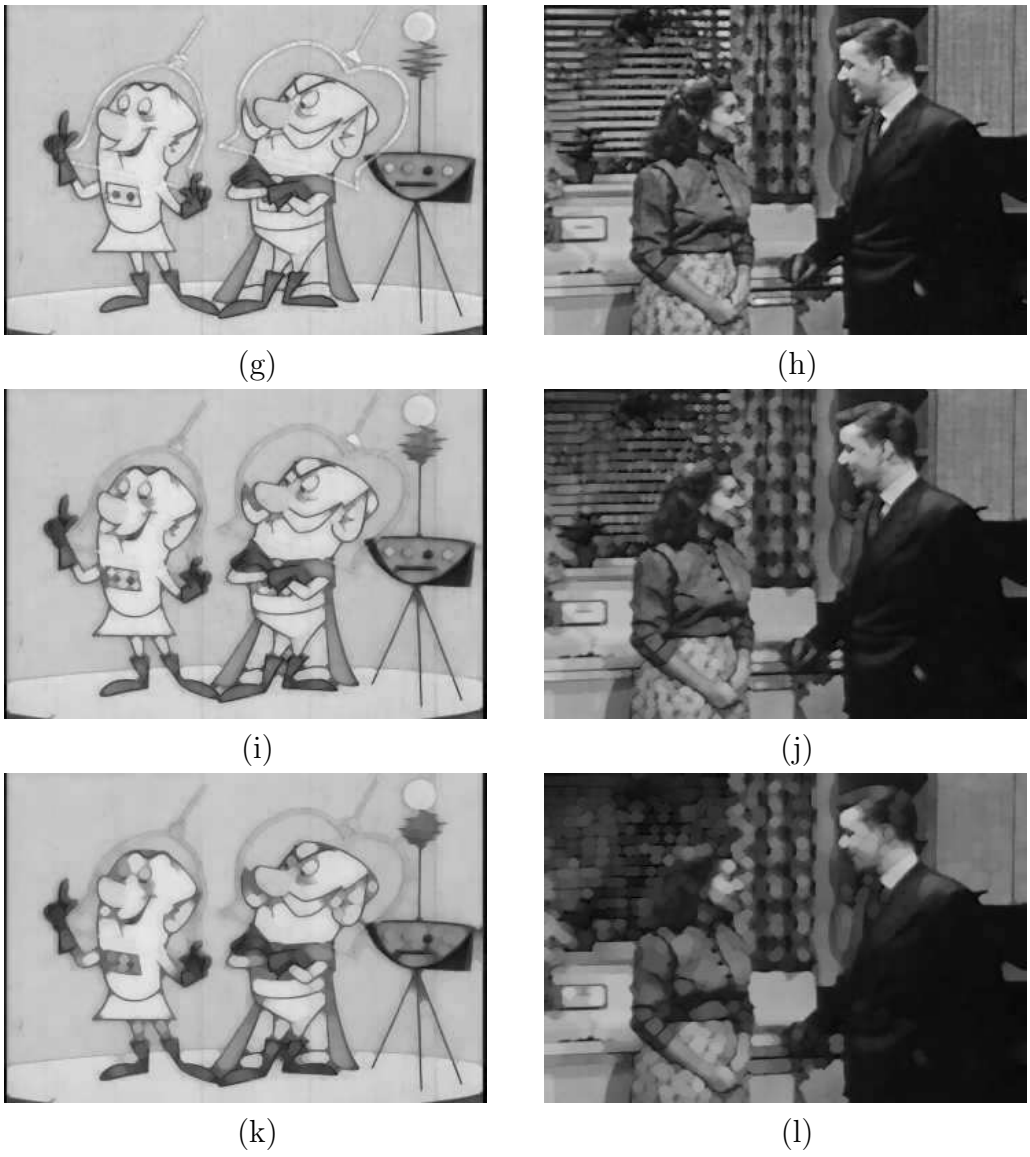


Figure 3.32: Continuation of small-scale disk pattern spectrum descriptors example. Opened (g) cartoon and (h) photographic image with SE disk, $r_1 = 1$. Opened (i) cartoon and (j) photographic image with SE disk, $r_2 = 2$. Opened (k) cartoon and (l) photographic image with SE disk, $r_4 = 4$.

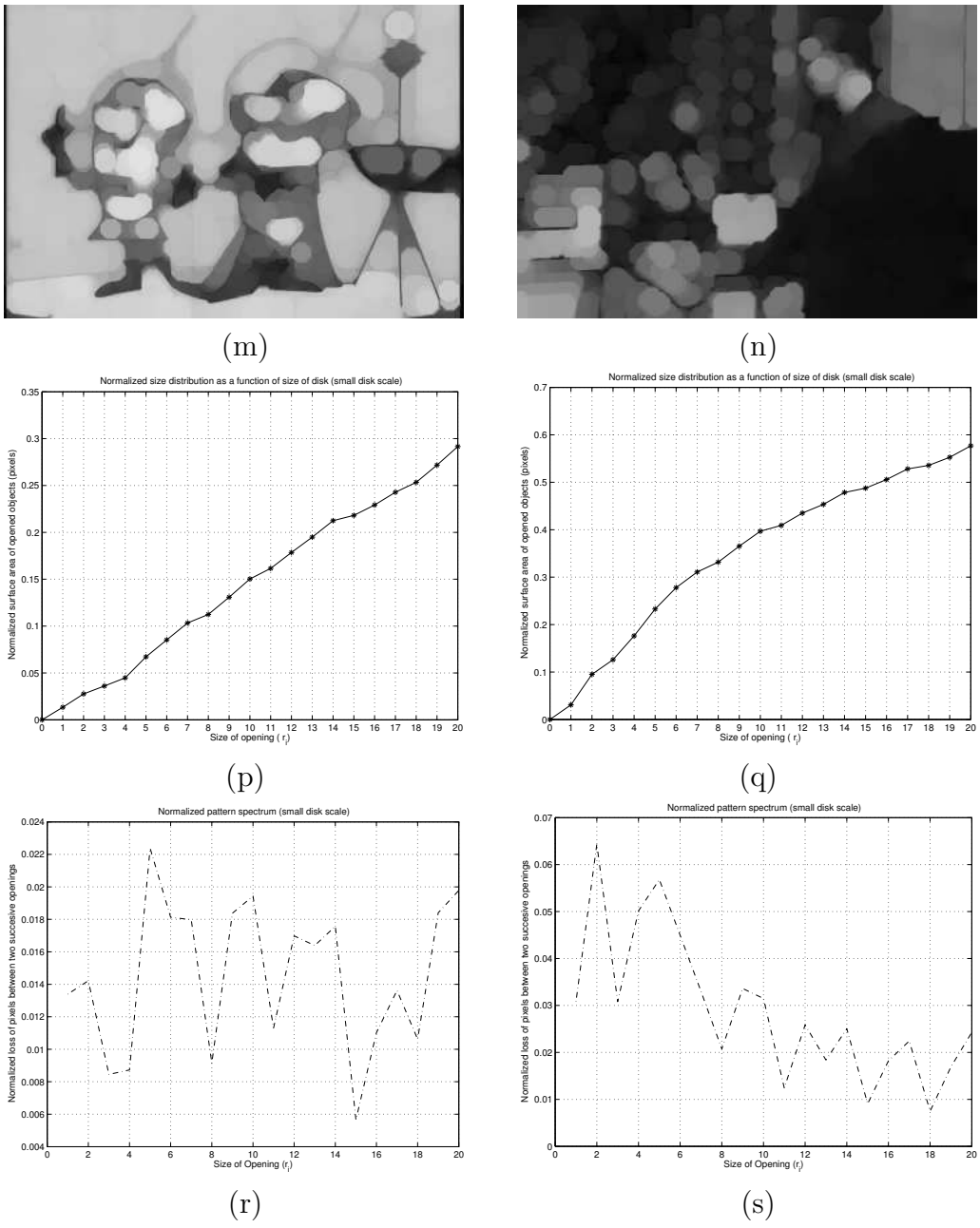
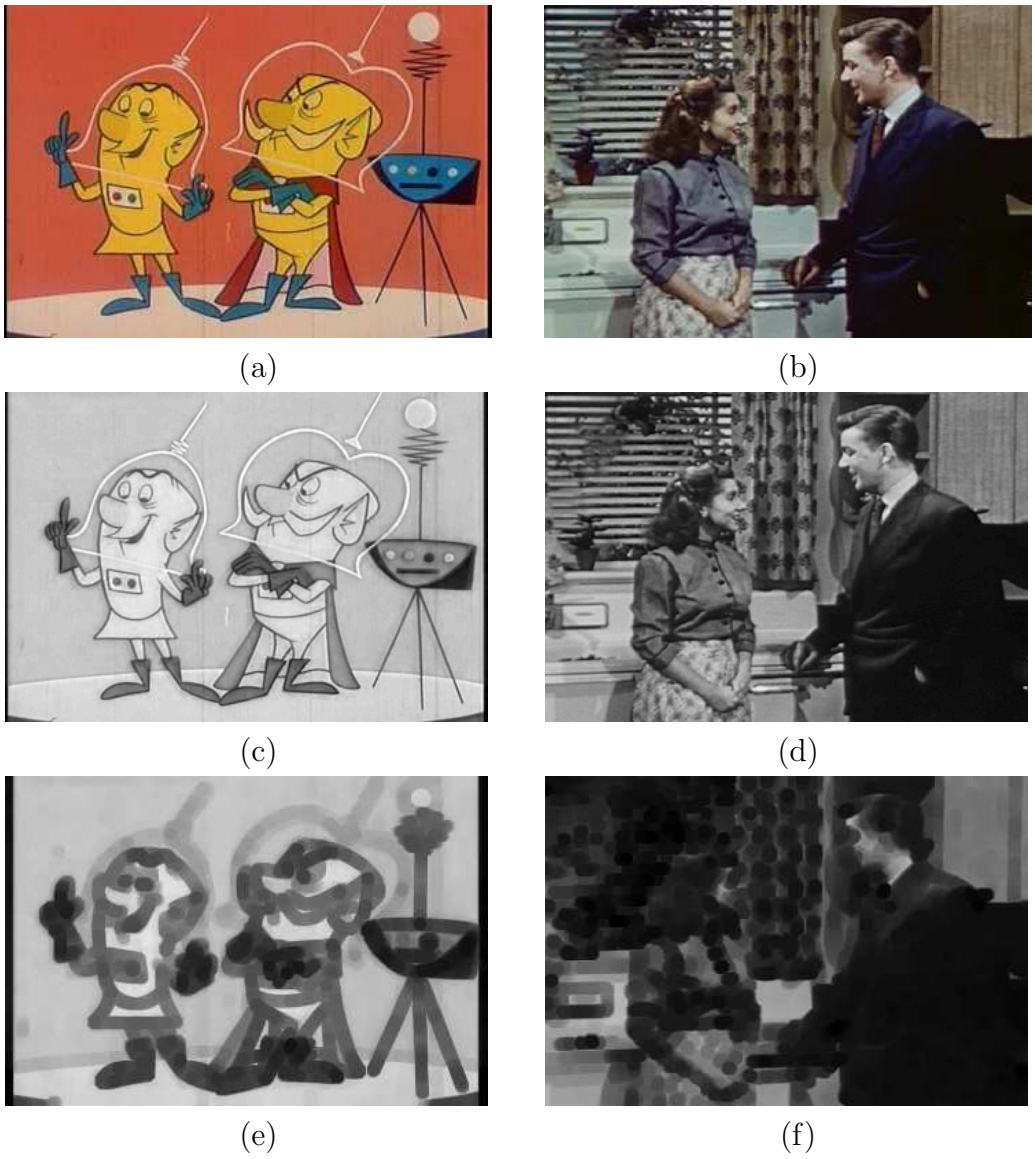


Figure 3.33: Continuation of small-scale disk pattern spectrum descriptors example. Opened (m) cartoon and (n) photographic image with SE disk, $r_8 = 8$. Size distribution of the (p) cartoon and (q) photographic image with SE disk, $r_i = i$, $i = 1, \ldots, 20$. Derivative (r) of (p). Derivative (s) of (q).



(a) (b) (c) (d) (e) (f)

Figure 3.34: Process of obtaining the 10 large-scale disk pattern spectrum descriptors. An original (RGB) (a) cartoon and (b) photographic image. Grayscale cartoon (c) and (d) photographic image (HSV color space). Eroded (e) cartoon and (f) photographic image with SE disk, $r_1 = 5$.

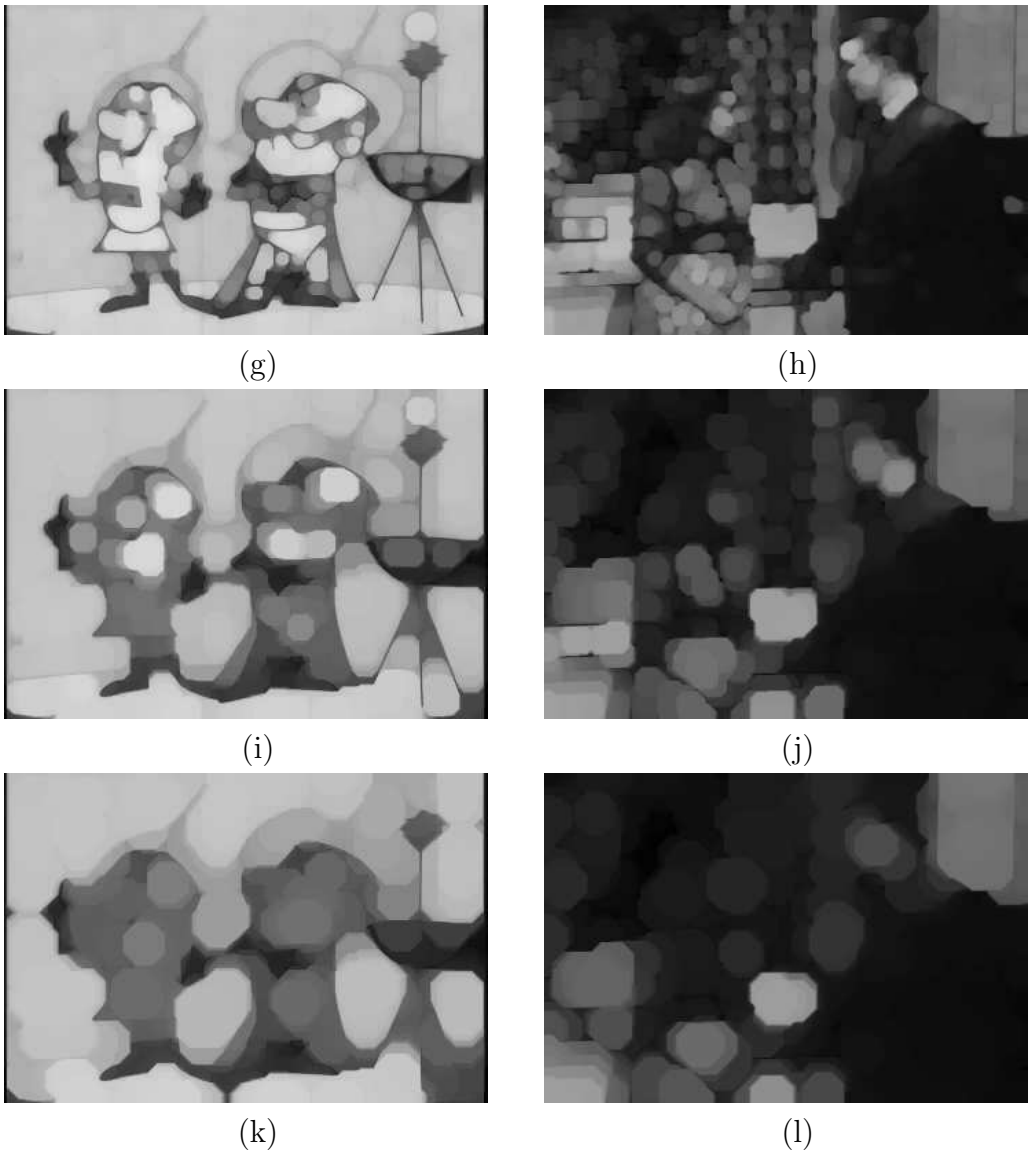


Figure 3.35: Continuation of large-scale disk pattern spectrum descriptors example. Opened (g) cartoon and (h) photographic image with SE disk, $r_1 = 5$. Opened (i) cartoon and (j) photographic image with SE disk, $r_2 = 10$. Opened (k) cartoon and (l) photographic image with SE disk, $r_3 = 15$.

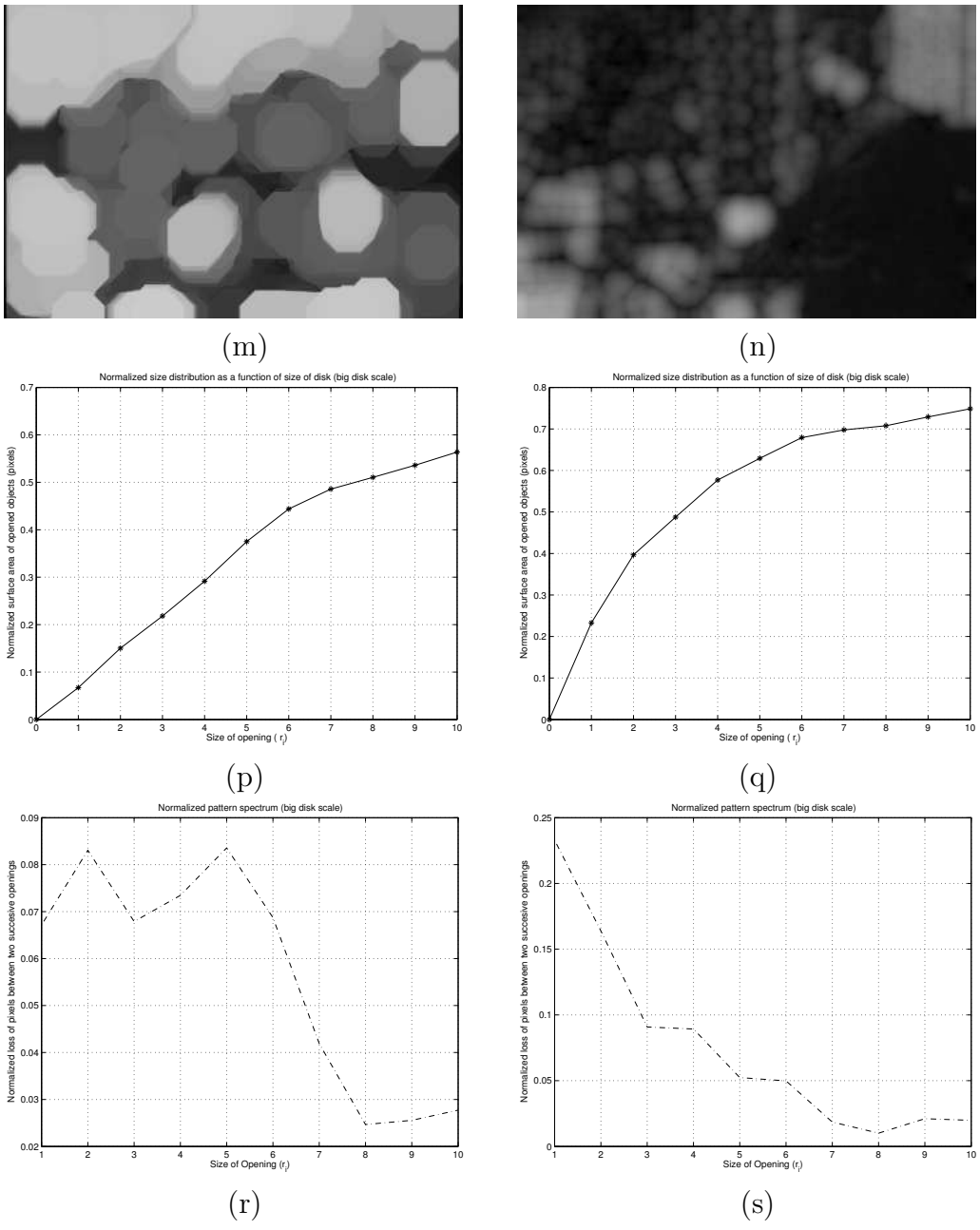


Figure 3.36: Continuation of large-scale disk pattern spectrum descriptors example. Opened (m) cartoon and (n) photographic image with SE disk, $r_4 = 20$. Size distribution of the (p) cartoon and (q) photographic image with SE disk, $r_i = i5$, $i = 1 \ldots 10$. Derivative (r) of (p). Derivative (s) of (q).

Chapter 4

SUPPORT VECTOR MACHINES

For some of our one-dimensional image descriptors, we have a clear intuition how they distinguish cartoons from photographs and 'learning' to use such a descriptor reduces to determining a good threshold value. For others, especially the various histograms, it is more convenient to classify automatically the patterns that are typical for cartoons or photographs. For such generic classification tasks, a popular and often successful technique is Support Vector Machine (SVM) learning [21], which has built-in guards against *overfitting* and can be tailored to known or conjectured structure in the data by the choice of the *kernel* function.

4.1 Learning from data

There are two ways to learn from data: *supervised* and *unsupervised* learning. For supervised learning a supervisor is available in the form of observed response data. *Regression estimation* and *classification* are typical problems solved through supervised learning. In unsupervised learning, no supervisor is available, thus no response data. Density estimation is a typical example of unsupervised learning. Our compression ratio image descriptor is another example of unsupervised learning.

The supervised learning problem is the problem of finding a desired dependency using a limited number of observations. The general model of learning from examples consists of three components:

1. A generator (G) generates vectors $x \in R^n$ from a fixed, unknown probability distribution function $P(x)$.

2. A supervisor (S) returns an output value $y \in R$ to every input vector x according to a conditional distribution function $P(y|x)$, $y \in Y$, also

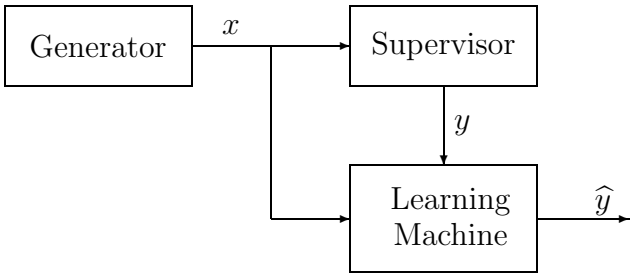


Figure 4.1: The General Learning Machine.

unknown.

3. A learning machine (LM) selects the best approximating function from a set of functions $f(x) \in F$, on the basis of the given observations (x_i, y_i) $i = 1, \ldots, l$.

Figure 4.1 shows the relationships between the three components (G), (S) and (LM). The learning machine has to choose from the set $f(x) \in F$, the function which best approximates the response y of the supervisor. All learning machines basically map an input vector x into a higher dimensional feature space $\mathcal{F}$ and then construct an approximation function in this feature space. The best approximating function f is the function that has the lowest risk of making errors. That is, when the difference between the response y of the supervisor of a given input x and the response $\widehat{y} = f(x)$ of the learning machine is the lowest. This difference is determined by a so-called *Loss function*, $L(y, f(x))$. The goal is to minimize the expected value of the loss, given by the following (*Risk functional*) equation

$$R[f] = \int_{R^n \times Y} L(y, f(x)) \mathrm{d}P(x, y). \tag{4.1}$$

Note that the joint probability function $P(x, y) = P(x)P(y|x)$ is unknown. The only information available is the training set of l independent and identically distributed (i.i.d.) observations,

$$(x_1, y_1), \ldots, (x_l, y_l), \tag{4.2}$$

drawn according to $P(x, y)$.

The difference between different types of learning machines is based on two features: First, on which *inductive principle* is used; second, on which

type of loss function is used. For example, the Empirical Risk Minimization (ERM) inductive principle is used by the least squares and maximum likelihood methods, whereas the L_2-loss function defines a regression learning machine. Next, we state the two main learning problems for supervised learning: classification and regression estimation. Each learning problem requires the use of a different loss function.

Classification

Classification is a main learning problem for supervised learning. The task of classification is to find a rule that, based on external observations, assigns an object to one of several classes. In our case there are only two different classes, cartoon images and photographic images. One possible formalization of this task is to estimate a function $f : R^n \to \{-1, +1\}$, using input – output training data pairs

$$(x_1, y_1), \ldots, (x_l, y_l) \in R^n \times Y, \qquad Y = \{-1, +1\}$$

generated i.i.d. according to an unknown probability distribution $P(x, y)$ such that f will correctly classify unseen examples (x, y) (a training pattern and the label). An example is assigned to the class $+1$ if $f(x) \geq 0$ and to the class -1 otherwise. The test examples are assumed to be generated from the same probability distribution $P(x)$ as the training data. For classification problems the response y of the supervisor takes only discrete values. Every value identifies a certain class. In our case y takes only two values and we consider the loss-function:

$$L(y, f(x)) = \begin{cases} 0 & \text{if } y = f(x) \\ 1 & \text{if } y \neq f(x) \end{cases} \tag{4.3}$$

The risk functional (4.1), using the loss-function in (4.3), determines the probability that the indicator function $f(x)$ gives a different answer than the supervisor does, that is to make a *classification error*. The learning problem for classification is therefore defined as finding the indicator function which minimizes the probability of making classification errors when the probability measure $P(x, y)$ is unknown, but the learning data in (4.2) are given.

Regression estimation

In the problem of regression estimation, the response y of the supervisor can be any real value. The functions $f(x)$ are also real valued and represent

the potential regression function. A typical loss function that is used for regression problems is given by

$$L(y, f(x)) = (y - f(x))^2. \tag{4.4}$$

The risk functional that uses (4.4) as loss function determines the prediction error of the regression function $f(x)$ with respect to the supervisor's response. Therefore, the learning problem for regression is that of finding the regression function which minimizes the probability of making prediction errors when the probability measure $P(x, y)$ is unknown, but the learning data in (4.2) are given.

4.1.1 The empirical risk minimization principle

Unfortunately, the expected error (risk) $R[f]$ cannot be minimized directly, since the underlying probability distribution $P(x, y)$ is unknown. Therefore, we have to try to estimate a function that is *close* to the optimal one based on the available information. To this end, we need what is called an *induction principle*. The ERM inductive principle is very important in learning theory. It consists in approximating the minimum of the risk (4.1) by the minimum of the *empirical risk functional*

$$R_{emp}[f] = \frac{1}{l} \sum_{i=1}^{l} L(y_i, f(x_i)), \tag{4.5}$$

which is constructed on the basis of the given learning data set $(x_i, y_i), i = 1, \ldots, l$.

Consistency of the ERM principle

It is possible to give conditions on the learning machine which ensures that asymptotically (as $l \to \infty$), the empirical risk will converge towards the expected risk (Figure 4.2). From the figure is quite clear why learning machines based on the ERM principle have difficulties with small samples. The ERM will only converge to the actual risk ($\inf R[f]$ – the infimum of the expected risk), when the number of observations in the learning data tends to infinity. In other words, for small sample size large deviations are possible and *overfitting* may occur (see Figure 4.3). Hence, a small generalization error cannot be obtained by simply minimizing the training error (4.5). One way to avoid the overfitting is to *restrict* the complexity of the function class F that one chooses the function f from [28]. The intuition is that a "simple" (e.g., linear) function that explains most of the data is preferable to a complex one.

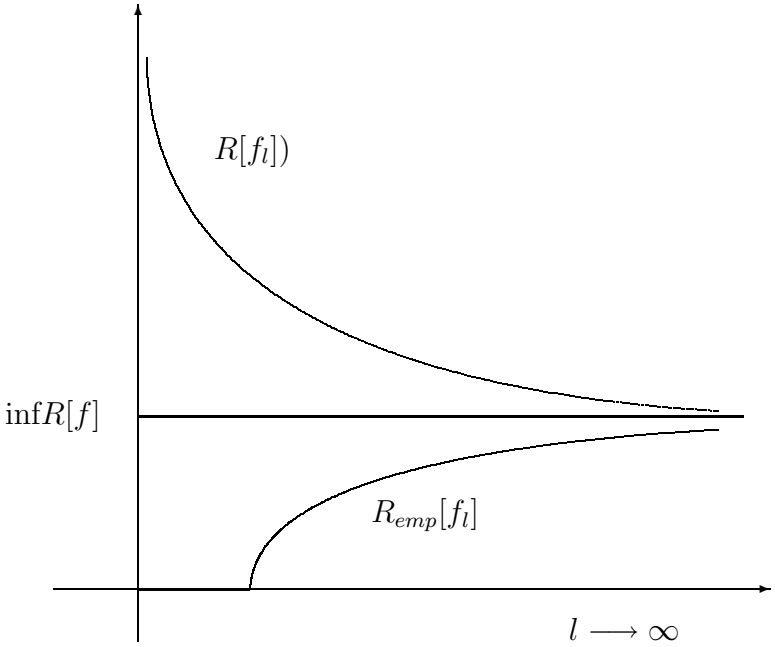


Figure 4.2: The consistency of the learning process

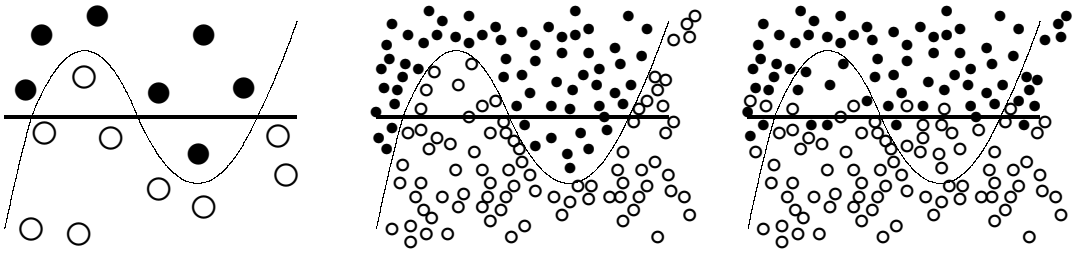


Figure 4.3: Illustration of the overfitting dilemma: Given only a small sample (a) either, the solid or the thin hypothesis might be true, the thin one being more complex, but also having a smaller training error. Only with a large sample we are able to see which decision reflects the true distribution more closely. If the thin hypothesis is correct the solid would underfit (b); if the solid were correct the thin hypothesis would overfit (c).

A specific way of controlling the complexity of a function class is given by VC theory and the structural risk minimization (SRM) principle [28, 29].

4.1.2 Structural risk minimization principle

Recall that the ERM principle is only intended for large sample sizes; for small samples the set of functions used in the learning process should have the right level of complexity in order for the learning machine to be able to generalize well. The concept of complexity is captured by Vapnik-Chervonenkis (VC) dimension h of the function class F from which the estimate f is chosen. The VC dimension measures how many (training) points can be separated for all possible labelings using functions of the class. Constructing a nested family of function classes $F_1 \subset \cdots \subset F_k$ with non-decreasing VC dimension the SRM principle proceeds as follows: Let $f_1, \ldots, f_k$ be the solution of the empirical risk minimization (4.5) in the functional classes F_i. SRM chooses the function class F_i (and the function f_i) such that an upper bound on the generalization error is minimized which can be computed making use of the following theorem (see also Figure 4.4):

Theorem 1 *([28, 29]) Let h denote the VC dimension of the function class F and let R_{emp} be defined by (4.5) using the* 0/1*-loss.*[1] *For all $\delta > 0$ and $f \in F$ the inequality bounding the risk*

$$R[f] \leq R_{emp}[f] + \sqrt{\frac{h(\ln \frac{2l}{h} + 1) - (\ln \frac{\delta}{4})}{l}} \tag{4.6}$$

holds with probability of at least $1 - \delta$ for $l > h$.

The goal is to minimize the generalization error $R[f]$ by obtaining a small training error $R_{emp}[f]$ while keeping the function class as small as possible. Avoiding the two extremes enforced by (4.6):

1. a very small function class (like F_1) keeps a large training error, as reflected by $R_{emp}[f]$, while

2. a huge function (like F_k) gives a large square root term.

We would like to obtain a function that explains the data quite well *and* to have a small risk in obtaining that function. The best class is usually in "the middle" (Figure 4.4). In Figure 4.4 it is clear that the SRM principle is concerned with finding the right balance between the learning ability and

[1] $L(y, f(x)) = \Theta(-yf(x))$, where $\Theta(z) = 0$ for $z < 0$ and $\Theta(z) = 1$ otherwise.

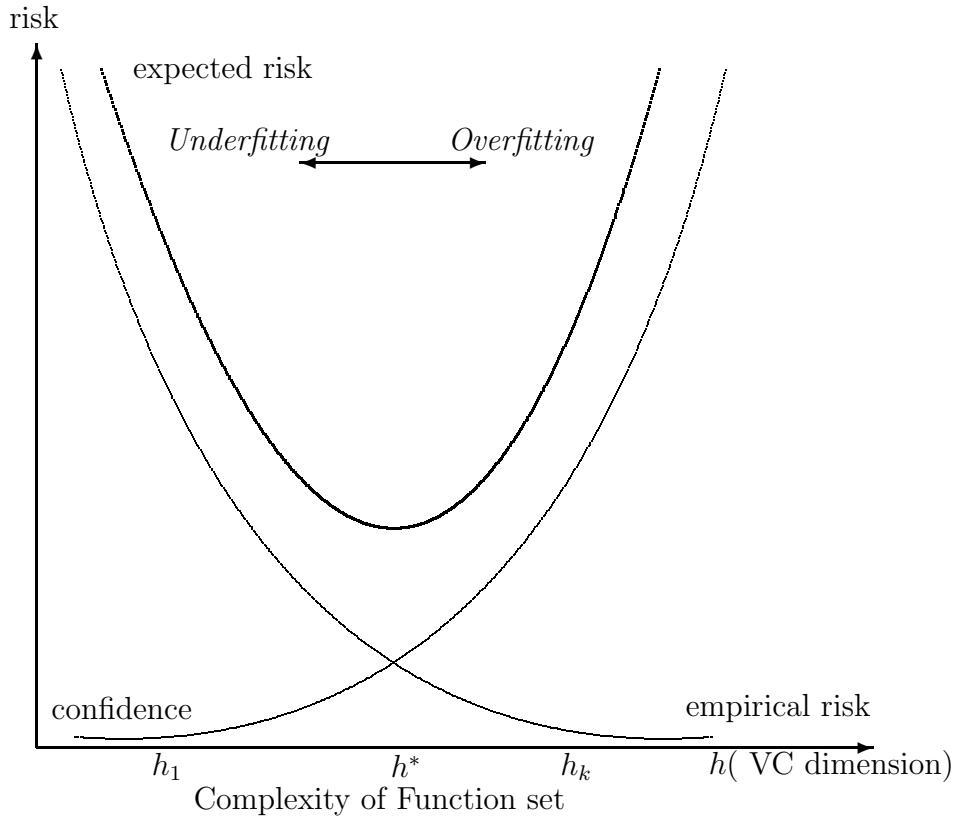


Figure 4.4: The consistency of the learning process. In practice the goal is to find the best trade-off between empirical error and complexity.

the generalization ability of the learning machine. If a too high complexity is used by the learning machine, the learning ability may be good but the generalization ability not. The learning machine will overfit the data. In Figure 4.4, we see that the right region of the complexity corresponds with overfitting of the learning data. On the other hand, when the learning machine uses too little complexity, it may have a good generalization ability, but not a good learning ability. This underfitting of the learning machine corresponds with the left region of the complexity. The optimal complexity of the learning machine is the set of approximating functions with lowest VC dimension and lowest training error.

Constructing of learning machines using the SRM

The implementation of the SRM principle requires *a priori* specification of the structure on the set of approximating (or loss) functions. For such a given set, the optimal model estimation amounts to the following two steps.

1. Select an element of the structure which has optimal complexity.
2. Estimate the best model of this element.

Therefore, unlike classical methods, learning machines that implement the SRM principle, provide analytical estimates for model selection based on the bounds for generalization error. If $\delta = \min(\frac{4}{\sqrt{l}}, 1)$ [29], the bound in (4.6) can be represented as

$$R[f] \leq R_{emp}[f] + \Phi\left(\frac{l}{h}, \frac{-\ln \delta}{4}\right), \tag{4.7}$$

where $\Phi(\cdot)$ is a scalar and called the VC *confidence interval*, because it estimates the difference between the empirical error and the actual error. There are two approaches of implementing the SRM inductive principle in learning machines:

1. Keep the VC confidence interval $\Phi(\cdot)$ fixed and minimize the empirical risk $R_{emp}[f]$.
2. Keep the empirical risk $R_{emp}[f]$ fixed and minimize the VC confidence interval $\Phi(\cdot)$.

Neural networks (NN) algorithms implement the first approach, since the number of hidden nodes is defined *a priori* and therefore the complexity of the structure is kept fixed. The second approach is implemented by the Support Vector Machines (SVM) method where the empirical risk is either chosen to be zero or set to an *a priori* level and the complexity of the structure is optimized.

4.2 Support Vector Machines

4.2.1 Separating hyperplanes

Suppose we are given a set of pattern vectors $x_1, \ldots, x_l \in R^n$ which we would like to classify depending on which side of a "separating" hyperplane they are. Any hyperplane in R^n can be written as

$$\{x \in R^n | (w \cdot x) + b = 0\}, \quad w \in R^n, \quad b \in R, \tag{4.8}$$

where $(w \cdot x)$ is the scalar product between w and x. In this formulation, w is a vector orthogonal to the hyperplane: If w has unit length, then $w \cdot x$ is the length of x along the direction of w.

By requiring the scaling of w and b to be such that the point(s) closest to the hyperplane satisfy $|(w \cdot x_i) + b| = 1$, we obtain the *canonical* form (w, b) of the hyperplane

Definition 1 (Canonical hyperplane) *The pair $(w, b) \in R^n \times R$ is called a canonical form of the hyperplane (4.2.1) with respect to $x_1, \ldots x_l \in R^n$, if it is scaled such that*

$$\min_{i=1,\ldots,l} |(w \cdot x_i) + b| = 1, \tag{4.9}$$

which amounts to saying that the point closest to the hyperplane has a distance of $1/\|w\|$.

4.2.2 Margins and VC dimension

Let us assume that the training sample is separable by a hyperplane, i.e., we choose functions of the form

$$(w \cdot x) + b = 0$$

Using $\|w\|$ we define a structure on the set of approximating functions

$$f(x) = (w \cdot x) + b,$$

for which the VC dimension is minimized, such that the elements of the set F_A are analyzed using $\|w\| \leq A$. Then, if $A_1 \leq A_2 \leq A_3 \leq \ldots \leq A_n$, the set F_A can be nested such that $F_{A_1} \subset F_{A_2} \subset F_{A_3} \subset \ldots \subset F_{A_n}$. Vapnik proved in [29] that the VC dimension h of a set of canonical hyperplanes in R^n (4.9) such that $\|w\| \leq A$ is

$$h = \min(R^2 A^2, n) + 1,$$

where all the training data points (vectors) are enclosed by a sphere of the smallest radius R.

Therefore, minimization of $\|w\|$ is an implementation of the SRM principle because a small value of $\|w\|$ will result in a small value of h and the VC dimension h is consequently minimized.

In classification problems we want to find a decision rule $D(x) = (w \cdot x) + b$ that classifies without error learning data

$$(x_1, y_1), \ldots, (x_l, y_l), \quad x_i \in R^n, \quad y_i \in \{-1, +1\}.$$

into two classes $\{x|D(x) > 0\}$ and $\{x|D(x) < 0\}$. It is said that such a decision rule (*hyperplane*), separates the data set without error. If we assume that the data is separable into two classes (see Figure 4.5), i.e., we choose functions of the form

$$f(x) = (w \cdot x) + b, \tag{4.10}$$

there exist many hyperplanes that separate the data, but we are looking for the one that is *optimal*.

The optimal separating hyperplane not only separates the data without error, it also has the largest distance between the closest vectors to either side of the hyperplane as seen in Figure 4.5. This distance is called the *margin* and can be measured by the length of the weight vector w in (4.10): as we assumed that the training sample is separable we can rescale w and b such that the points closest to the hyperplane satisfy $|(w \cdot x_i) + b| = 1$ (i.e., obtain the canonical hyperplane). If we consider two samples x_1 and x_2 from different classes with $(w \cdot x_i) + b = 1$ and $(w \cdot x_i) + b = -1$ respectively; then the margin is given by the distance of these two points, measured perpendicular to the hyperplane, i.e., $\left(\frac{w}{\|w\|} \cdot (x_1 - x_2)\right) = \frac{2}{\|w\|}$.

In other words, the maximum margin is achieved by choosing the hyperplane with the smallest norm of coefficients. This has the implication that the smaller the value of $\|w\|$ the larger is the value of the margin. Recall that it was argued that minimizing $\|w\|$ results in minimizing the VC dimension. Therefore, the optimization problem which implements the SRM principle can be stated as

$$\min_{w,b} \quad \|w\|, \tag{4.11}$$

$$\text{subject to} \quad y_i((w \cdot x_i) + b) \geq 1, \quad i = 1, \ldots, l. \tag{4.12}$$

4.2.3 Separable case

In the "support vector machine" method, the empirical risk is kept fixed while the VC confidence interval is minimized. In general, the SVM maps

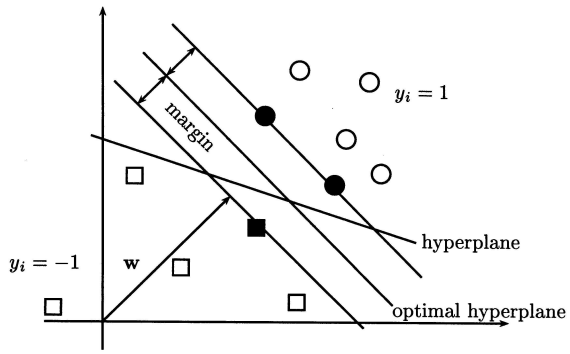


Figure 4.5: The optimal separating hyperplane. The margin of a linear classifier is the minimal distance of any training point to the hyperplane.

the input data $x_1, \ldots, x_l \in R^n$ into a higher dimensional feature space $\mathcal{F}$. The mapping can be done nonlinearly and the transformation function $\phi(x)$ is chosen *a priori*.

$$\begin{array}{rcl} \phi : R^n & \rightarrow & \mathcal{F} \\ x & \mapsto & \phi(x) \end{array}$$

Now one works with the sample

$$(\phi(x_1), y_1), \ldots, (\phi(x_l), y_l) \in \mathcal{F} \times Y.$$

in $\mathcal{F}$ instead of R^n. Statistical learning theory [28] tells us that not the dimensionality but the complexity of the function class matters. This idea can be understood from the example in Figure 4.6: in the feature space of second order monomials (4.13) all one needs for separation is a *linear* hyperplane, whereas in two dimensions

$$\begin{array}{rcl} \phi : R^2 & \rightarrow & R^3 \\ (x_1, x_2) & \mapsto & (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2) \end{array} \tag{4.13}$$

a rather complicated *nonlinear* decision surface is necessary to separate the classes.

In the feature space the SVM finally constructs an optimal approximation function which is linear in its parameters

$$f_{w,b}(x) = (w \cdot \phi(x)) + b,$$

where w and b have to be determined. The notation $(w \cdot \phi(x))$ defines the scalar product between w and $\phi(x)$. In the classification case, $f_{w,b}(x) = 0$ is

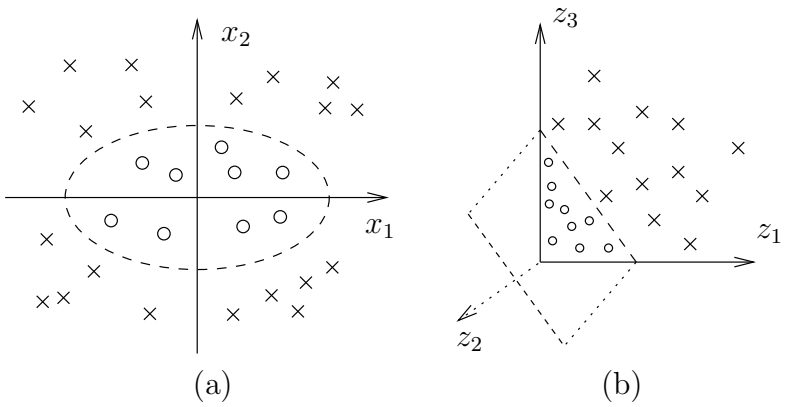


Figure 4.6: Two dimensional classification example. Using the second order monomials $x_1^2, \sqrt{2}x_1x_2$ and x_1^2 as features a separation in feature space can be found using a *linear* hyperplane (b). In input space this construction corresponds to a *non-linear* ellipsoidal decision boundary (a)

called a decision function or hyperplane and the optimal function is called the optimal separating hyperplane.

Since the square root function is a monotonic function, one can minimize the squared norm in (4.12) and reach the same result.

The optimization problem becomes a quadratic programming (QP) problem with convex constraints:

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2, \tag{4.14}$$

$$\text{subject to} \quad y_i((w \cdot \phi(x_i)) + b) \geq 1, \qquad i = 1, \ldots, l. \tag{4.15}$$

The QP problem is solved by forming the dual optimization problem. Introducing Lagrange multipliers $\alpha_i \geq 0, i = 1, \ldots, l$, one for each constraint in (4.15), we get the following Lagrangian:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i (y_i((w \cdot \phi(x_i)) + b) - 1). \tag{4.16}$$

The task is to minimize (4.16) with respect to w, b and to maximize it with respect to α_i. At the optimal point (w^*, b^*, α^*), we have the following saddle point equations:

$$\frac{\partial L(w^*, b^*, \alpha^*)}{\partial b} = 0 \quad \text{and} \quad \frac{\partial L(w^*, b^*, \alpha^*)}{\partial w} = 0,$$

which translate into

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \quad \text{and} \quad w = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i). \tag{4.17}$$

By substituting (4.17) into (4.16) and by replacing $((\phi(x_i) \cdot \phi(x_j))$ with *kernel functions* $k(x_i, x_j)$ (discussed in the next section), we get the dual quadratic optimization problem:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{subject to} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, l, \\ & \sum_{i=1}^{l} \alpha_i y_i = 0, \end{aligned} \tag{4.18}$$

where $k(\cdot, \cdot)$ is a scalar product in feature space $\mathcal{F}$.

By solving the dual optimization problem, we obtain the coefficients $\alpha_i, i = 1, \ldots, l$, which we need to express the w which solves (4.14). Let $\alpha^* = (\alpha_1^*, \ldots, \alpha_l^*)$ be the solution of the QP problem above. The vectors for which the corresponding Lagrange multipliers are positive are called *support vectors*. The values of the Lagrange multipliers assign weights to the corresponding vectors. These vectors and their weights are then used to define the decision rule or model. Therefore the learning machine in (4.18) is called the *support vector* machine. The decision rule for the classification problem is then expressed in terms of the set of support vectors SV as

$$f(\widehat{x}) = \operatorname{sgn}\left(\sum_{i \in SV} y_i \alpha_i^* (\phi(\widehat{x}) \cdot (\phi(x_i)) + b^* \right) = \operatorname{sgn}\left(\sum_{i \in SV} y_i \alpha_i^* k(\widehat{x}, x_i) + b^* \right).$$

where b^* is the constant threshold or bias determined by

$$b^* = \frac{1}{2} \left[(w^* \cdot x_1^*) + (w^* \cdot x_{-1}^*) \right],$$

with x_1^* any support vector belonging to the class with output data 1, and x_{-1}^* any support vector belonging to the class with output data -1. In Figure 4.5 the support vectors are the vectors on the margin, indicated with black markers. The support vectors are the vectors that lie on the margin and they typically represent the input data that are the most difficult to classify.

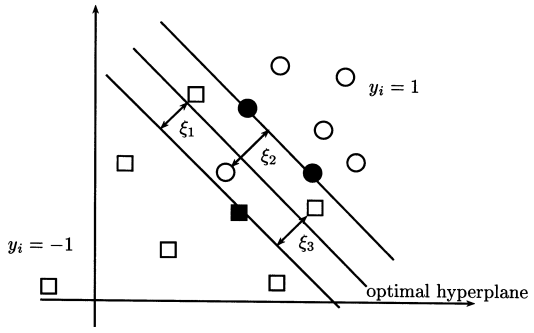


Figure 4.7: The optimal Hyperplane for the non-separable case.

4.2.4 Non-separable case.

For noisy data, the data set is not separable. There are no hyperplanes that can separate the data set without error and no maximal margin can be constructed (Figure 4.7). The slack variables ξ are introduced to penalize for those vectors lying within the margin:

$$y_i((w \cdot \phi(x_i)) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, l.$$

The value of a stack variable is the distance of the corresponding vector to the margin The SVM solution can then be found by keeping the upper bound on the VC dimension small and by minimizing an upper bound $\sum_{i=1}^{n} \xi_i$ on the empirical risk i.e., the number of training errors. Thus, we minimize

$$\min_{w,b,\xi} \quad \tfrac{1}{2}\|w\|^2 + C \sum_{i=1}^{l} \xi_i,$$

where the regularization constant $C > 0$ determines the trade-off between the empirical error and the complexity term. From introducing the slack variables ξ_i, we get the box constraints that limit the size of the Lagrange multipliers:

$$\alpha_i \leq C, \quad i = 1, \ldots, l,$$

in the corresponding dual problem:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, l, \\ & \sum_{i=1}^{l} \alpha_i y_i = 0. \end{aligned} \tag{4.19}$$

4.3 Kernels functions

In many learning machines the learning data is mapped (nonlinearly) into a higher dimensional feature space. The mathematics of the SVM expresses these transformations as (linear) scalar products of the input data. However, for large real world problems even if we could control the statistical complexity of the functional class, it becomes not easy to control the algorithmic complexity of the learning machine. For instance, if we have images of 16×16 pixels as patterns and 5th order monomials as mapping ϕ, then one would map to a $\binom{5+256-1}{5} \approx 10^{10}$-dimensional space.

Fortunately, for certain feature spaces $\mathcal{F}$ and corresponding mappings ϕ there is a highly effective trick for computing scalar products in feature spaces using *kernel functions*. In the example from Eq. (4.13), the computation of a scalar product between two feature space vectors, can be readily reformulated in terms of a kernel function k

$$\begin{aligned}(\phi(x) \cdot \phi(y)) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2)(y_1^2, \sqrt{2}y_1x_2, y_2^2)^T \\ &= ((x_1, x_2)(y_1, y_2)^T)^2 \\ &= (x \cdot y)^2 \\ &=: k(x, y).\end{aligned}$$

The interesting point about kernel functions is that the scalar product can be *implicitly* computed in $\mathcal{F}$, *without* explicitly using or even knowing the mapping ϕ. So, kernels allow to compute scalar products in spaces where we could otherwise hardly perform any computations.

4.4 Types of kernels

Many different kernel functions can be constructed as long as they satisfy Mercer's conditions [28]:

$$\int\int K(x, \hat{x})g(x)g(\hat{x})\mathrm{d}x\mathrm{d}\hat{x} > 0, \quad \text{for all } g \neq 0, \quad \int g^2(x)\mathrm{d}x < \infty. \quad (4.20)$$

For finite dimensional input spaces, the conditions (4.20) states that $K(x, \hat{x})$ is an admissible kernel function if it produces a kernel matrix that is symmetric and positive semi-definite. However, there are two main types of kernels, namely *local* and *global* kernels. In local kernels only the data that are close or in the proximity of each other have a noticeable influence on the kernel values. In contrast, a global kernel allows data points that are far away from each other to have an influence on the kernel values as well. An example of

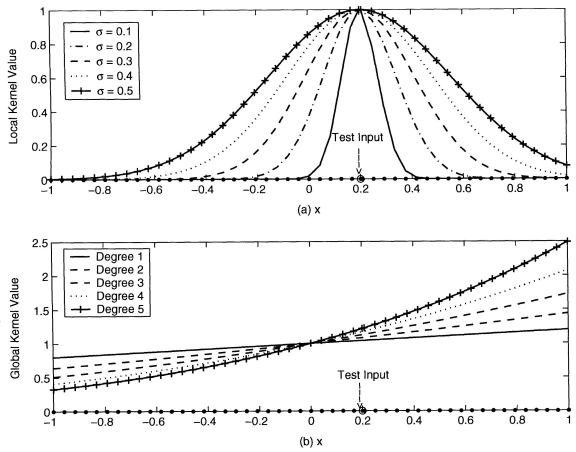


Figure 4.8: Examples of (a) a local kernel (RBF) and (b) a global kernel (polynomial).

a typical local kernel is the radial basis function (RBF) kernel,

$$K(x, \hat{x}) = \exp\left\{-\frac{\|x - \hat{x}\|^2}{2\sigma^2}\right\}, \tag{4.21}$$

where the kernel parameter, σ, is the width of the radial basis function. The polynomial kernel, a typical example of a global kernel, is defined as

$$K(x, \hat{x}) = [(x \cdot \hat{x}) + 1]^q, \tag{4.22}$$

where the kernel parameter q is the degree of the polynomial to be used. The behavior of the two types of kernels is shown in Figure 4.8 where for linearly distributed data over $[-1, 1]$ the kernel values with respect to a specific test point at 0.2 are determined. In Figure 4.8(a) the local effect of the RBF kernel is shown for the chosen test input, for different values of the width σ. One can clearly see that at some point, the kernel values essentially level off to zero. A local kernel therefore only has an effect on the data points in the neighborhood of the test point. In Figure 4.8(b), the global effect of the polynomial kernel of various degrees can be seen. Consider the same test data point as used in the case of the local kernel. For each degree of the polynomial, all data points in the input domain have nonzero kernel values. The test data point has a global effect on the other data points. This can be explained by that in (4.22) every data point from the set x has an influence on the kernel value of the test point $\hat{x}$, irrespective of its the actual distance from $\hat{x}$.

The exact formulations that we use in our C-Support Vector Classification case (cartoon and photographic images) in matrix notation and the algorithmic pseudocode for the SVM classification are given in Subsection 5.2.2 of Chapter 5. Examples of how our two SVM classifiers perform (one with polynomial and the other with radial kernel) can be found in Section 5.4 and Section 5.5 of Chapter 5.

Chapter 5

EXPERIMENTS

5.1 Overview

For each input image (keyframe) we extract 148 descriptors as outlined in Chapter 2 and Chapter 3. This process is depicted in Figure 5.1; our descriptors are summarized in Table 5.1. Their individual 'naive' usefulness is

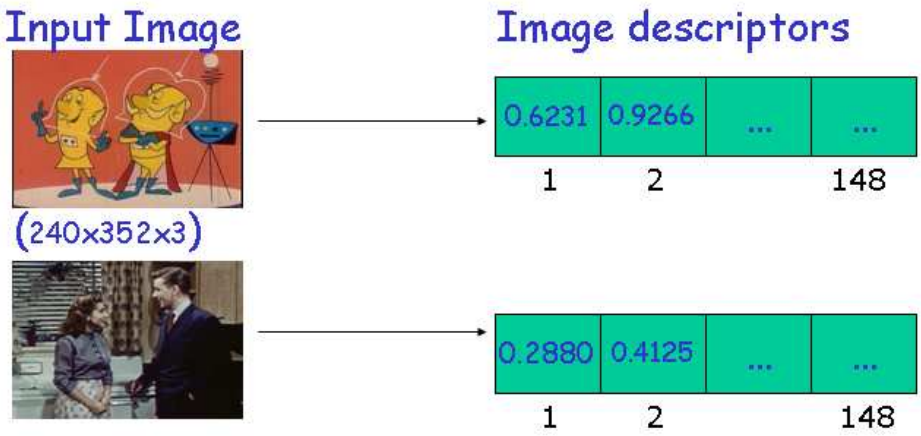


Figure 5.1: Image-descriptor extraction

given in Section 5.4 by the error on photos $\mathcal{E}(p)$, error on cartoons $\mathcal{E}(c)$, and total error $\mathcal{E}(t)$, when performing classification with SVM machine learning using the given image descriptor alone. Combined performance is discussed in Section 5.5.

Image Descriptors	Dimension
average saturation	1
threshold brightness	1
color histogram	45
edge-direction histogram	40
compression ratio	1
multi-scale pat.spectrum	60

Table 5.1: Overview of our all image descriptors

5.2 The experimental setup

5.2.1 Matlab and the OSU SVM classifier toolbox

Matlab is a comprehensive technical computing software package by MathWorks Inc. Matlab is oriented towards efficient numeric operations on floating-point numbers; data is typically organized in matrices (hence the name Matlab = 'Matrix Laboratory') or in arrays of higher dimension. The software consists of a graphical user interface for writing and running scripts in the Matlab programming language ('`.m` files'), which access the underlying computing kernel.

In addition to built-in functions, a wide variety of special-purpose libraries, called Toolboxes, exist from MathWorks and third-party software vendors; we made use of the Matlab Image Processing Toolbox, which is bundled with MathWorks Release 12 and 13, and the OSU SVM Classifier Toolbox for Matlab [16]. The Matlab Image Processing Toolbox supplies functions for

- reading, displaying, and writing bitmapped graphics files;
- color-space and image-depth conversion; and
- applying filters by convolution and other techniques.

The toolbox has functions for morphological operations. However, they are limited to flat structuring elements and, hence, not sufficient for our purposes. For the same reason, we opted not to use the SDC Morphology Toolbox for Matlab [22].

The OSU SVM Classifier Toolbox for Matlab is a high-performance implementation of state-of-the art algorithms for Support Vector Machines [16]. The Matlab interface uses as computational back end the LIBSVM library [3]. Both packages are open-source software and available for download from the Internet. The central functions that we use are `PolySVC` and `RbfSVC` for

constructing SVM classifiers with polynomial and radial kernels, respectively, `SVMTest` for cross-validation, and `SVMClass` for classifying new data.

5.2.2 Formulations

We have the following C-Support Vector Classification (C-SVC) binary case. Given training vectors $x_i \in R^{148}, \quad i = 1, \ldots, l$, in two classes (cartoon images and photographic images) labeled by $y_i \in \{1, -1\}$, C-SVC solves the following primal problem:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i, \\ & y_i(w^T \phi(x_i)) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, \quad i = 1, \ldots, l. \end{aligned}$$

Its dual is:

$$\min_{\alpha} \quad \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \tag{5.1}$$

$$0 \leq \alpha_i \leq C, \quad i = 1, \ldots, l, \tag{5.2}$$

$$y^T \alpha = 0, \tag{5.3}$$

where e is the vector of all ones, $C \geq 0$ is the upper bound, Q is an $l \times l$ positive semidefinite matrix with $Q_{ij} =: y_i y_j K(x_i, x_j)$ and $K(x_i, x_j) := \phi(x_i^T)\phi(x_j)$ is the kernel. Here training vectors x_i are mapped into a higher dimensional space by the function ϕ. The decision function expressed in terms of the support vectors (SV) is:

$$f(\widehat{x}) = \operatorname{sgn}\left(\sum_{i \in SV} y_i \alpha_i^* K(x_i, x_j) + b^* \right)$$

where b^* is the constant threshold or bias determined by

$$b^* = \frac{1}{2} \left[(w^* \cdot x_1^*) + (w^* \cdot x_{-1}^*) \right], \tag{5.4}$$

with x_1^* any support vector belonging to the class with output data 1, and x_{-1}^* any support vector belonging to the class with output data -1.

The algorithmic pseudo code for the SVM for classification is:

Algorithm: SVM for Classification

1. Select the learning data $(x_i, y_i), i = 1, \ldots, l$, kernel function K and C (default 1).
2. Construct the matrices Q for the quadratic-programming problem.
3. Solve (5.1) subject to (5.2) and (5.3) for α.
4. Calculate the bias b using (5.4).
5. Calculate $\|w\|^2 = \alpha^T Q \alpha$.
6. Calculate size of margin $\gamma = \frac{2}{\|w\|}$.
7. Identify support vectors as $\{x_i | \alpha_i > 0, i = 1, \ldots, l\}$.
8. Construct the index set SV of the support vectors.
9. Construct the SVM classification model using the determined parameters

$$f_{\alpha,b}(\widehat{x}) = \text{sgn}\left(\sum_{i \in SV} y_i \alpha_i K(x_i, \widehat{x}) + b\right).$$

5.3 The data

5.3.1 The principal data set: TREC-2002 keyframes

As mentioned in the introduction, we used keyframes extracted by Westerveld et al. [33] from the TREC-2002 video track [23] collection. These keyframes were selected by using the middle frame from each shot as representative of the shot [33] and our full collection of images contains all these keyframes, totalling 24,264 JPEG images of dimension 340×252 and average file size 46 kBytes.

15,000 of these images were classified manually into the categories 'photograph' (13,026), 'cartoon' (1,620), and 'borderline' (354). From this data we randomly selected subsets of 30% of the data for training and cross-validation i.e., 3,908 photographs and 486 cartoons.

5.3.2 For comparison: images from the web

Data set of 14,039 photographic and 9,512 graphical images harvested from the WWW was used for comparison. Subsets of 30% of the data were used for training, i.e., 4,239 photographics and 2,826 graphics, and the rest for cross-validation.

5.3.3 Layout of the data

Image files are stored in individual files located subdirectories:

```
image-corpus -+- xaa-images
              |
              +- xab-images
              |
              +- xac-images
              :
              +- xba-images
              :
              +- xzz-images
```

The directories `x*-images` contain the actual image files. The rationale behind this two-level structure is

1. to better cope with large number of files: having substantially more than 1,000 files in a directory make directory operations such as `ls` and thumbnail generation very slow, especially when the files are accessed over the network.

2. to cluster the data for parallel computation: one image directory represents in this case one chunk of data to be processed in a batch job.

The top-level directory name (here `image-corpus`) is arbitrary; we used `trec2002` for the TREC-2002 keyframes and `windhouw-class` for the web data (provided by M. Windhouwer). The assumptions on file names are as follows:

- recognized extensions are `jpg` for files in the JPEG format [32] and `gif` for files in the Graphics Interchange Format [9];

- images that are to be used for training or checking automatic classifiers start with 0 (for photos) or 1 (for cartoons).

- files with the extension `mat` are binary Matlab files storing variables such as our precomputed image descriptors.

5.3.4 Our programs

Our most important Matlab scripts are listed in Appendix C. The whole code is included on the CD-ROM accompanying this work. Overview of all files together with some important notes can be found in the README.txt file. Each file `functionName.m` contains the source code to the Matlab function `functionName`. A typical cycle on new data is as follows:

1. Edit `computeCarAll.m` (see Section C.2.5) to decide which image descriptors should be computed.

2. Run `computeCarAll.m` in all `x*-images` directories. This invokes the selected `computeCar`*`DescriptorName`*`.m` functions (see Section C.1), which write
   `car_`*`descriptorName`*`.mat` files containing the image descriptors for all images in the directory. For efficiency, some of these functions compute more than one descriptor and, hence, produce more than one `car_`*`descriptorName`*`.mat` file.

3. Edit `loadAllCar.m` (see Section C.2.3) to select descriptors for training and testing.

4. Run `mergeAllCarDir.m` (see Section C.2.2) to collect the selected descriptors from all `x*-images` directories.

5. Run `learnAndTest.m` (see Section C.2.1).

6. Run `ClassifyDirectory.m` (see Section C.3) to classify unseen (unlabeled) data, using the classifier build in 5.

The rationale for separating the training and testing of the classifier from computing descriptors is that the latter task can be very expensive computationally and we desired the freedom to experiment with different parameters of the classifiers.

Table 5.2 gives an overview over our descriptor-computing functions. Applying the functions individually on each subdirectory allows us to run several instances of Matlab in parallel on different machines or processors on multiprocessor systems. For example, with ca. 15,000 images in 15 directories, we could use 15 workstations each operating on 1,000 images. The time to compute any of our descriptors is largely independent of the concrete input data, hence this approach made good use of the computational resources with minimal overhead; more fine-grained parallelization would have made our code substantially more complicated. Indeed, the only 'scheduler' we used were shell scripts (`computeCarAll.sh` and `computeCar.sh`) that log in to every system from a fixed list of workstations and start an instance of Matlab in a given `xa*-images` subdirectory. The first script computes all the descriptors while the second one computes a given single descriptor. This approach was also sufficiently flexible to accommodate the addition of new image descriptors without recomputing all previous descriptors.

A Matlab `.mat` file stores one or several Matlab variables, i.e., typically matrices. Our convention was that k-dimensional image descriptors for n

Image Descriptor	Function `computeCar...`	Data File
average saturation	`AvgSatThrBrightness`	`car_avg_bright`
threshold brightness	`AvgSatThrBrightness`	`car_avg_sat_thr`
color histogram	`ColHist`	`car_colorhist`
edge-direction histogram	`EdgeDir`	`car_edge_dir`
compression ratio	`Compression`	`car_compression`
small-scale parabola pat.spect.	`CarGranulometry1`	`car_granulom1_parsmall`
large-scale parabola pat.spect.	`CarGranulometry1`	`car_granulom1_parbig`
small-scale disk pat.spectrum	`CarGranulometry1`	`car_granulom1_disksmall`
large-scale disk pat.spectrum	`CarGranulometry1`	`car_granulom1_diskbig`
scaled aspect ratio	`Dim`	`car_dim_aspect`
scaled min dimension	`Dim`	`car_dim_min`
file type	`FileType`	`car_file_type`

Table 5.2: Computing Image Descriptors

images are stored in a $n \times k$ matrix `cars` with double-precision floating-point entries. The file name of the image corresponding to the k descriptors in row i is stored in row i of the $n \times \ell$ matrix `fileNames` (Matlab treats character strings as one-dimensional arrays with integer entries; accordingly, ℓ is the maximum length of any file name in the directory, with shorter file names padded by space characters). Each `car_`*`descriptorName`*`.mat` file contains the two variables `fileNames` and `car` for the descriptor *`descriptorName`*.

After the the `computeCar*` functions have been run in all subdirectories, the data from the individual `car_`*`descriptorName`*`.mat` files must be merged for the currently selected subset of image descriptors and concatenated over all the subdirectories. The `loadAllCar` function determines the subset of descriptors to be used by calling the `mergeCar` function (see Section C.2.4), which performs the per-directory merging of descriptors (in database terminology, this is a join on the `fileNames` key). `mergeCar` is used as a subroutine of `mergeAllCarDir` that iterates over all subdirectories and concatenates the lists. The output of `mergeAllCarDir` is a file `car_all.mat` in the top-level directory (`image-corpus` in the example above). As for an individual descriptor, the `car_all.mat` contains two variables, `fileNames` and `car`, where `car` is a $n \times k$ matrix for a total of n files and k real-valued descriptors or descriptor components.

5.4 Performance of the image descriptors

We use the following formula to measure the individual performance of each descriptor:

$$E_t = E_p \frac{|p|}{|p| + |c|} + E_c \frac{|c|}{|p| + |c|},$$

where $|p|$ denotes the number of photos and $|c|$ denotes the number of cartoons. E_p is the error on photos, i.e., the fraction of all photos that is classified incorrectly. Analogously E_c is the ratio of incorrectly classified cartoons and E_t is the total error.

The experimental setup is: we use 30% of the manually classified data for training and the whole set of manually classified TREC2002 keyframes for testing. In other words, we train on (random) $3,908$ photos and 486 cartoons and test on $13,026$ photos and $1,620$ cartoons. We repeat these experiments several times to smooth out the variance in the results and report here average error values.

Note that we have many more photos than cartoons in our collection and this is to be expected in new material as well as there are more "photographic" movies on TV than cartoons. Hence, a trivial classifier, which

always classifies the input as photo will have rather small total error, namely

$$E_t = 0 \cdot \frac{13,026}{13,026 + 1,620} + 1 \cdot \frac{1,620}{13,026 + 1,620} \approx 0.1106 \ .$$

Since $E_t = 0.1106$ can be achieved without looking at the test data at all, a good classifier using our image descriptors should have much smaller total error.

Recall that the threshold-brightness descriptor is a value between 0 and 1 that gives the percentage of pixels in an image that have brightness greater than a predetermined threshold. To determine the best threshold, we ran the following experiments. In Figure 5.2 we give for increasing thresholds the error using a simple linear classifier[1]. As we see from Figure 5.2, the smallest error is achieved using a threshold of 0.2 and this is the value we used subsequently.

In Figures 5.3 to 5.20 we show the results of our experiments for probing the power of the individual descriptors by themselves. For each descriptor, we tried an RBF kernel and a polynomial kernel with a range of kernel parameters. The different errors are plotted with respect to the kernel parameter σ^2 or D, respectively. Average saturation improves the total error to 0.108, color histogram to 0.0914, pattern spectrum 0.911. The other descriptors are useful by themselves as their error is larger than 0.1106; however, as we show below, even threshold brightness, edge-direction histogram, and the compression descriptors *are* useful when used in conjunction with other descriptors.

5.5 Combining image descriptors

5.5.1 Choice of the SVM kernel and parameters

To use SVM learning, we need to fix the kernel (i.e., the inner product in the higher-dimensional space) and the parameters applying to the particular choice of kernel. The RBF kernel has the variance σ^2 of the Gaussian as parameter; the concrete polynomial kernel is determined by its degree D. The smaller σ^2 or the larger D, the more complex will the resulting model be, hence the more training data is needed and the greater is the danger of overfitting. On the other hand, decreasing σ^2 or increasing D shows at first great improvement and then levels off as the complexity of the model becomes sufficient to model the data. This can be seen for example in Figure 5.7 where

[1] We did not use SVM learning in this case for simplicity because we have clear intuition that cartoons should have greater brightness.

the total error flattens at $\sigma^2 = 1/14$. Similar "convergence" can be seen in experiments to validate the size of the training set: Figure 5.23 shows why we are confident that our SVM classifiers are well-trained: with larger and larger training sets, the error converges, showing that the accessible information in the data has been found.

In all cases, the RBF kernel gave better results. This is why we concentrated on it in our paper [11]. For the combined descriptors, we determined the optimal value of the variance as $\sigma^2 = 1/12$.

5.5.2 Relative descriptor performance

In Section 5.4 we gave a first assessment of the utility of our different descriptors by checking how useful they are individually for separating photos and cartoons. To determine how crucial the descriptors are for the combined result, we also conducted analogous experiments with all descriptors minus each individual descriptor (Figures 5.21 and 5.22). Here a large increase in error when removing a descriptor indicates great usefulness. This leads to the following ranking (in order of decreasing utility): color histogram, pattern spectrum, average saturation, edge-direction histogram, compression, threshold brightness. Regarding pattern spectra, the individual performance may suggest that small disk is most useful. However, the combined experiments confirmed our intuition that parabola ranks slightly higher than disk and large scale is more useful than small scale.

5.6 Performance on images from the web

For comparison with other classifying efforts, we also trained a classifier on the previously mentioned Web data. Even without using image descriptor depending on file type, dimension ratio, or smallest dimension (very good indicators of logos, banners, etc.), we obtain a 93% correct classification (slightly better than the decision tree classifier mentioned before [1] that uses file type and image size). When adding these properties to the feature vectors, our classification rate improves marginally (to 95.5%); unlike [1], we have not noticed a significant difference in accuracy depending on the image file type (JPEG or GIF).

Conversely, when training and testing the decision tree on our data, the classification accuracy suffers severely from the fact that all keyframes are of the same size; the error rate goes up to over 45% (Figure 5.23). We believe that this comparison demonstrates that our approach is generic and can be applied equally well to distinguish graphics from photos on the web

— without using derived properties like image size: *only the visual content is modeled in our characteristics!*

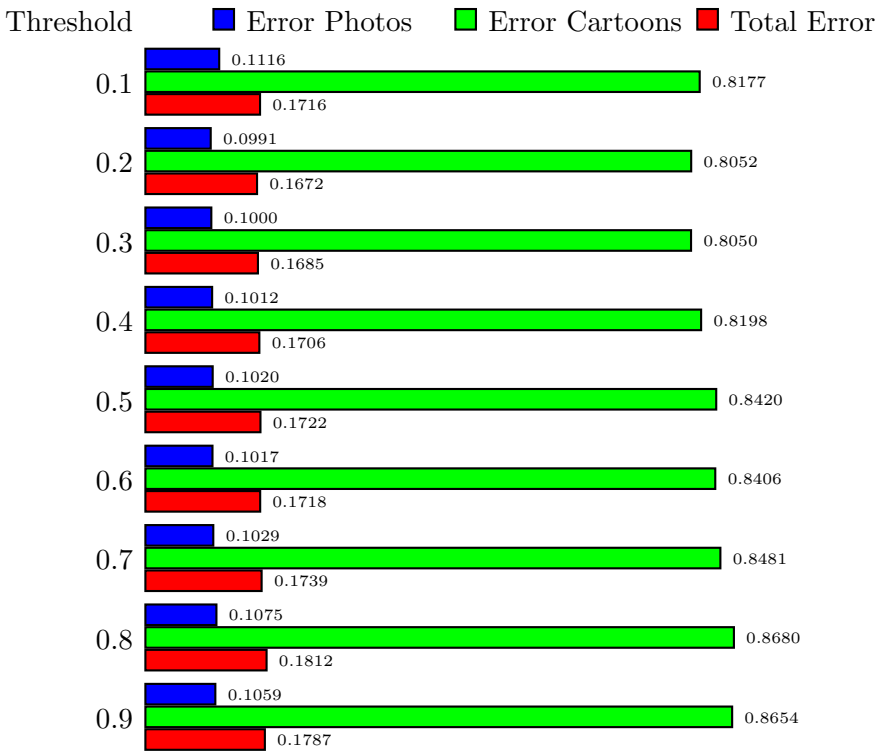


Figure 5.2: Performance of different thresholds for brightness (using a linear classifier)

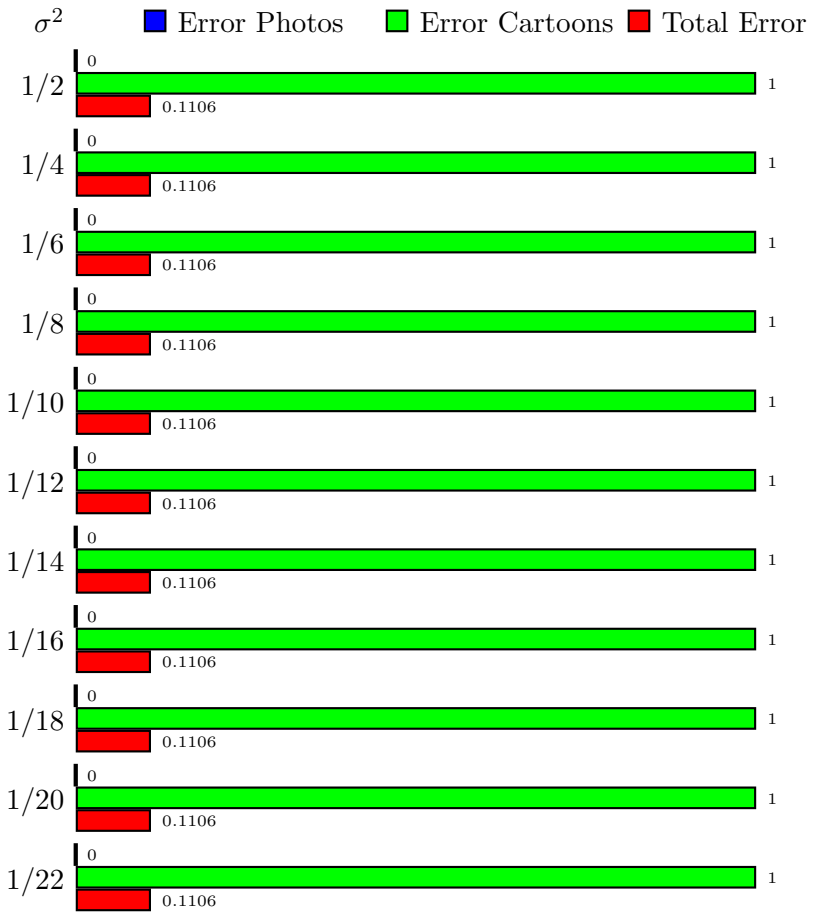


Figure 5.3: Individual performance of the threshold-brightness descriptor (RBF kernel)

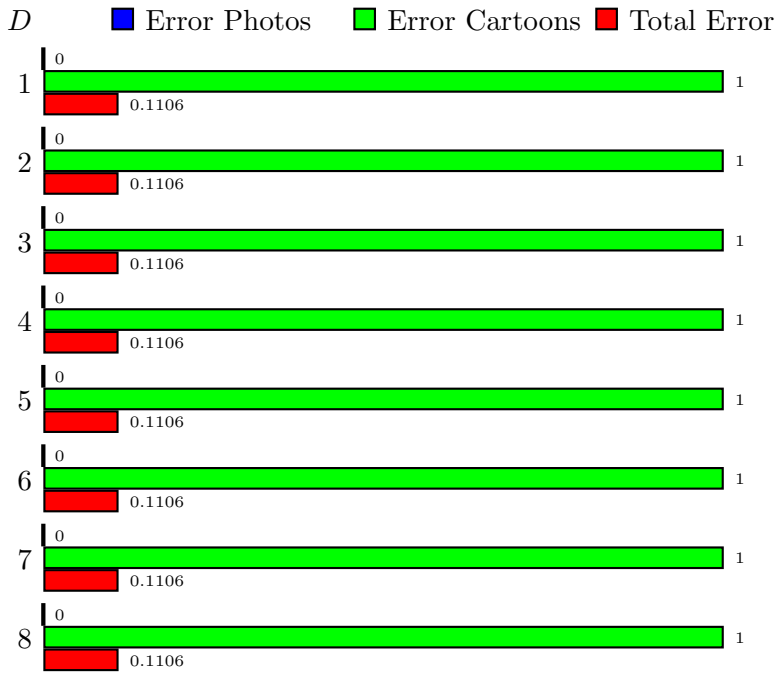


Figure 5.4: Individual performance of the threshold-brightness descriptor (polynomial kernel)

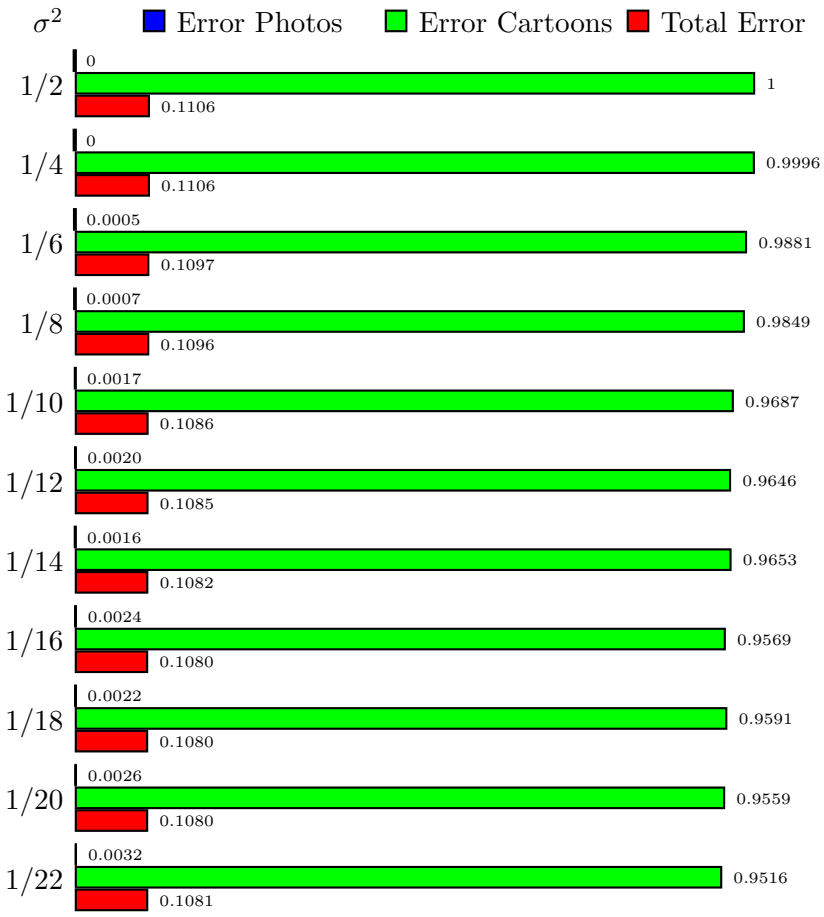


Figure 5.5: Individual performance of the average-saturation descriptor (RBF kernel)

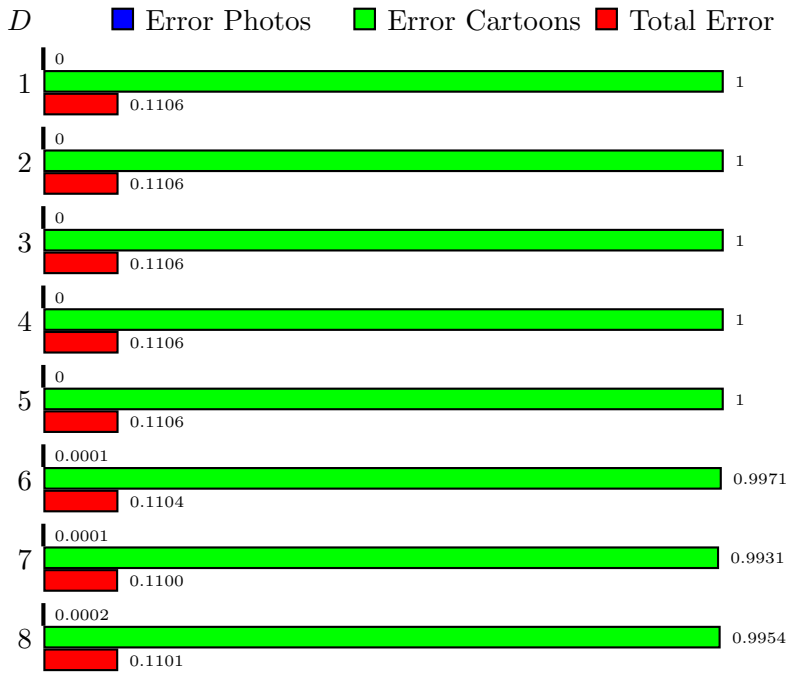


Figure 5.6: Individual performance of the average-saturation descriptor (polynomial kernel)

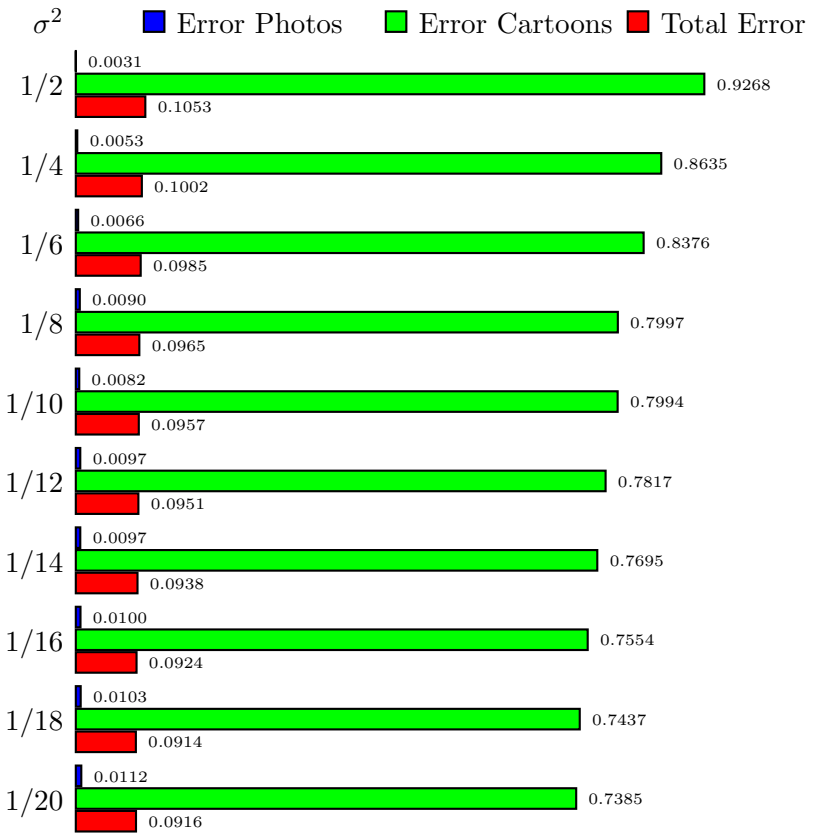


Figure 5.7: Individual performance of the color-histogram descriptors (RBF kernel)

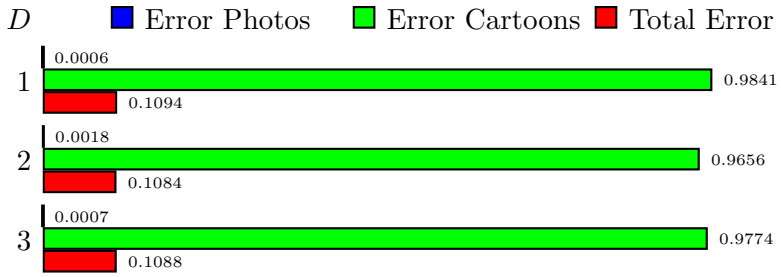


Figure 5.8: Individual performance of the color-histogram descriptors (polynomial kernel)

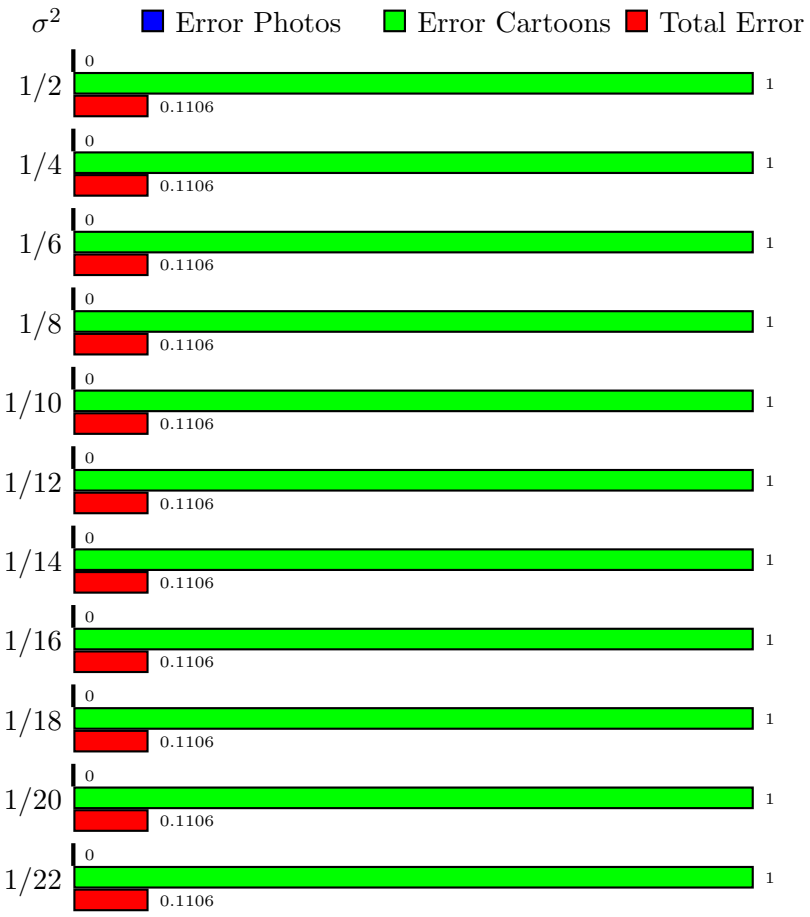


Figure 5.9: Individual performance of the edge-histogram descriptors (RBF kernel)

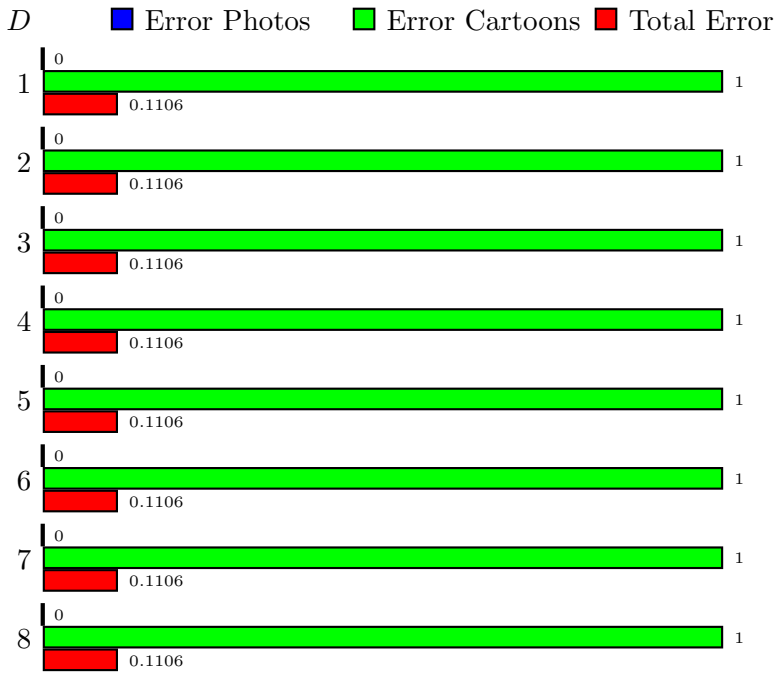


Figure 5.10: Individual performance of the edge-histogram descriptors (polynomial kernel)

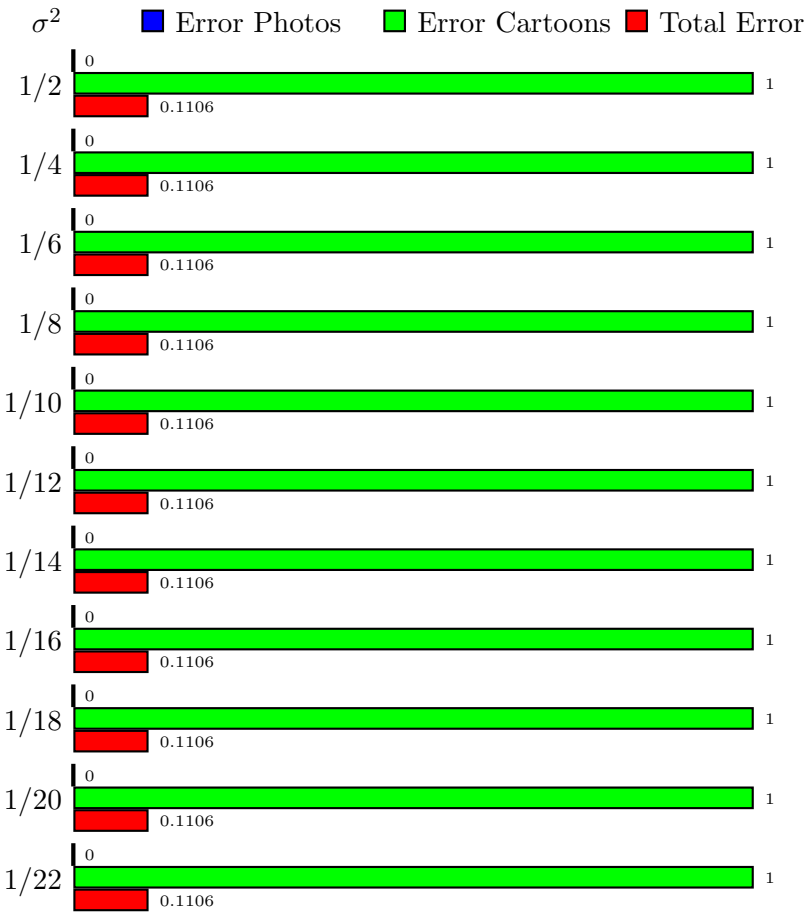


Figure 5.11: Individual performance of the compression-ratio descriptor (RBF kernel)

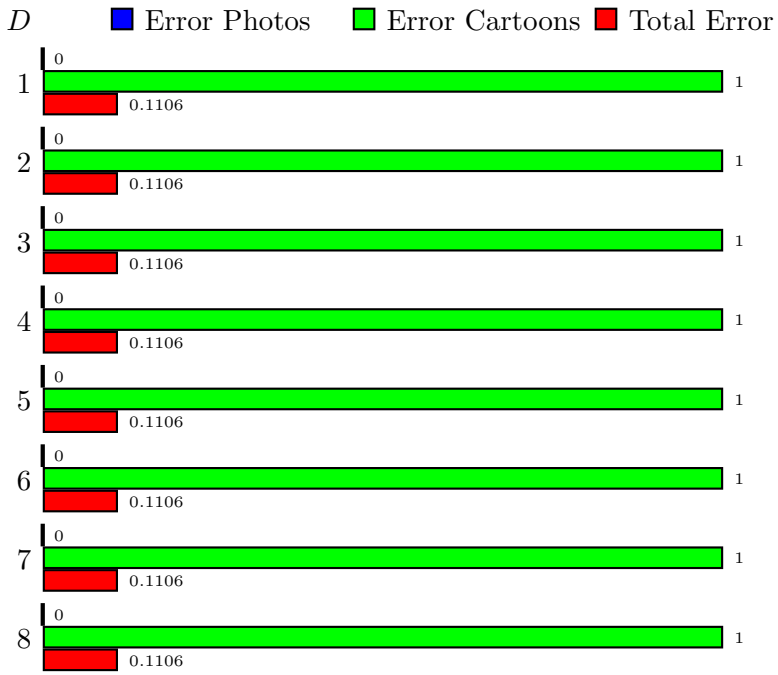


Figure 5.12: Individual performance of the compression-ratio descriptor (polynomial kernel)

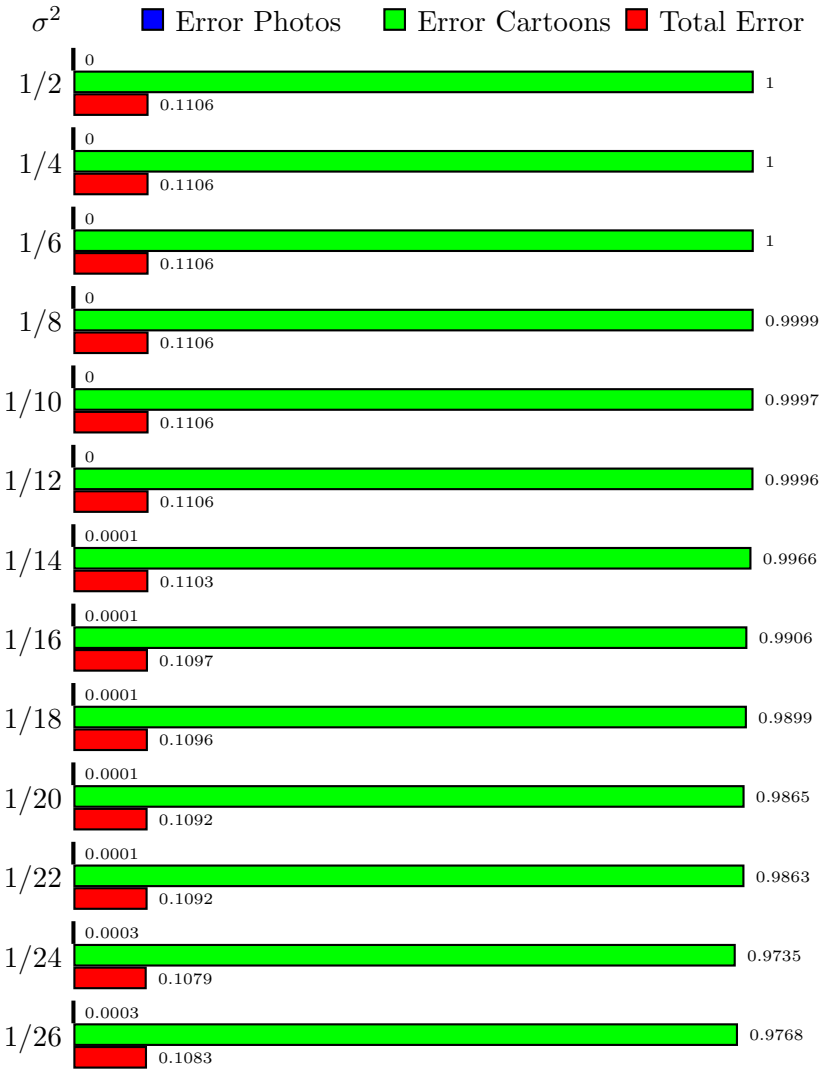


Figure 5.13: Individual performance of the large-scale disk pattern spectrum descriptor (RBF kernel)

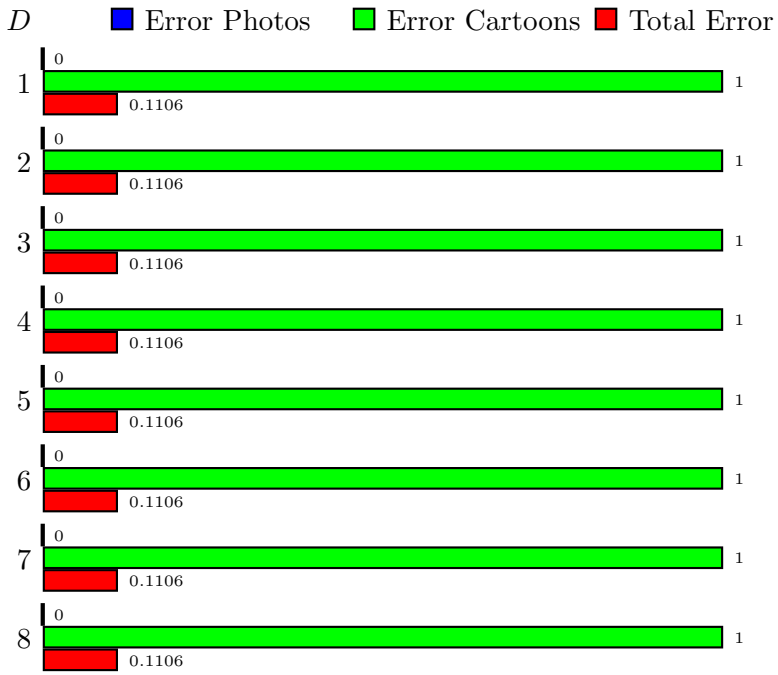


Figure 5.14: Individual performance of the large-scale disk pattern spectrum descriptor (polynomial kernel)

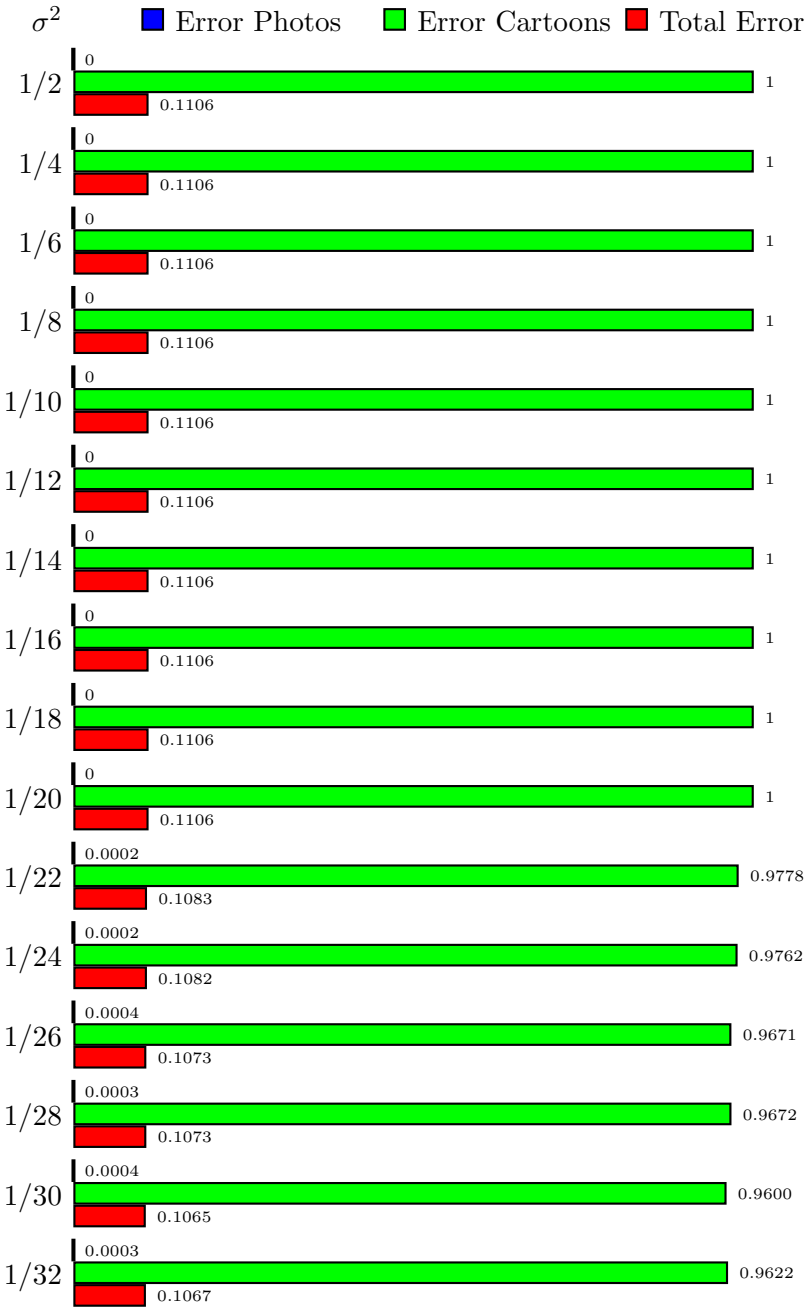


Figure 5.15: Individual performance of the large-scale parabola pattern spectrum descriptor (RBF kernel)

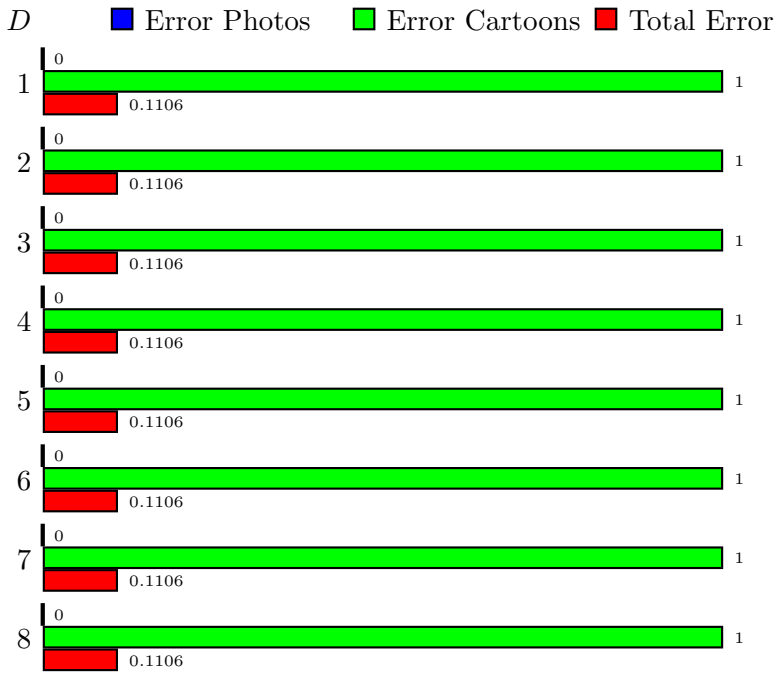


Figure 5.16: Individual performance of the large-scale parabola pattern spectrum descriptor (polynomial kernel)

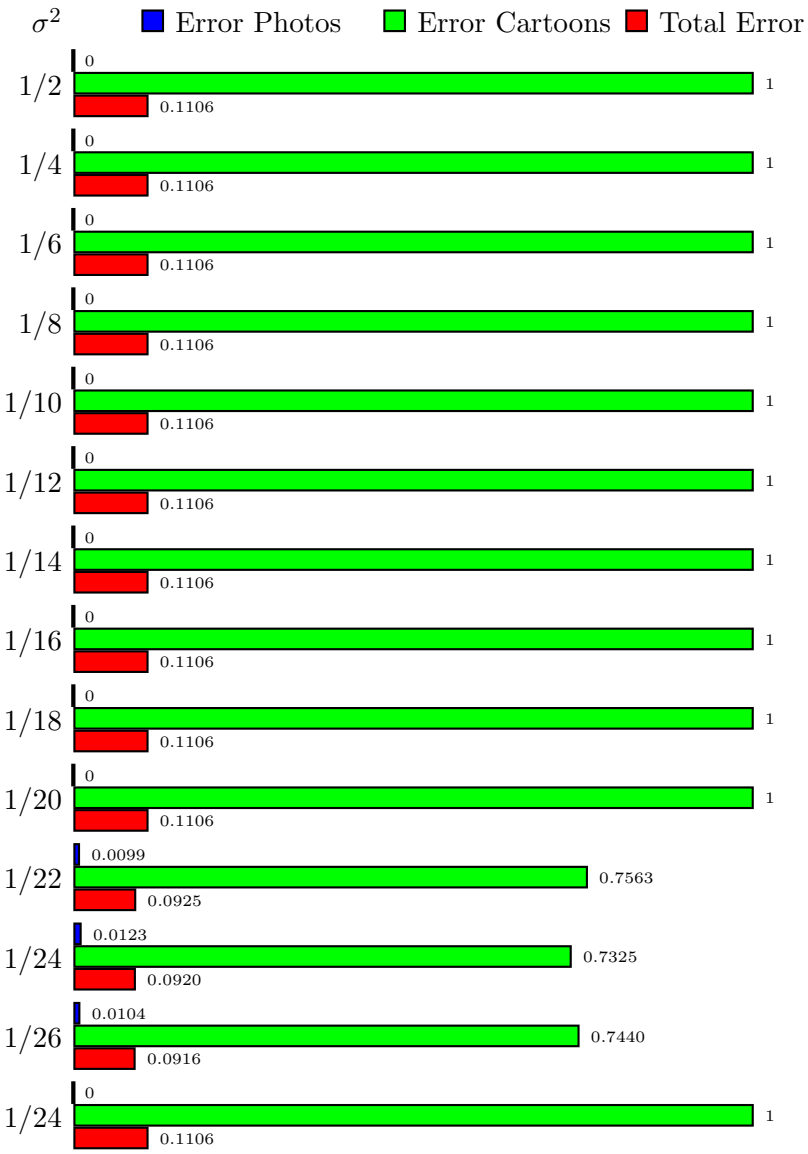


Figure 5.17: Individual performance of the small-scale disk pattern spectrum descriptor (RBF kernel)

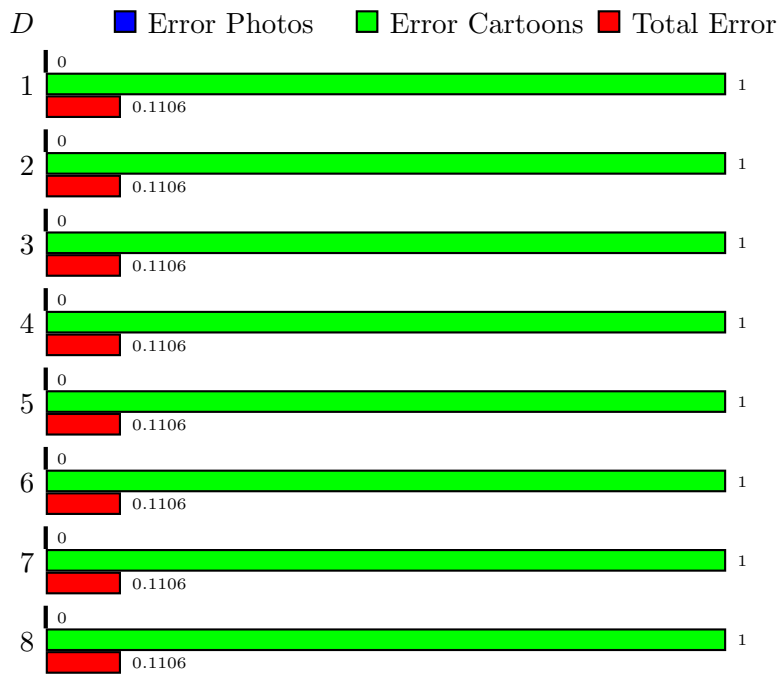


Figure 5.18: Individual performance of the small-scale disk pattern spectrum descriptor (polynomial kernel)

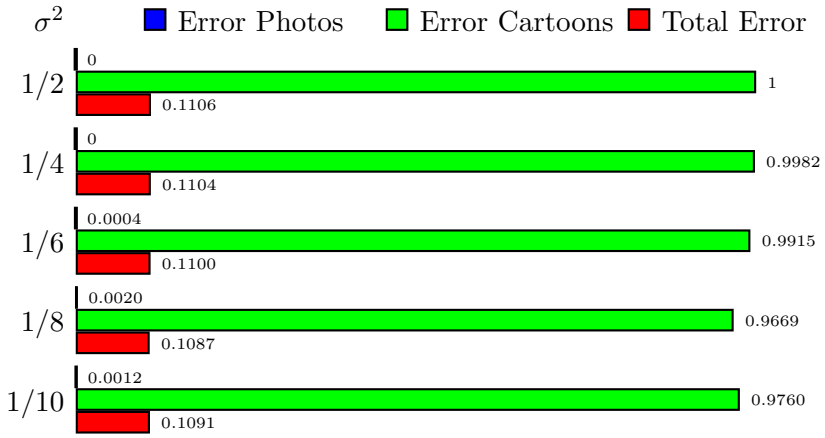


Figure 5.19: Individual performance of the small-scale parabola pattern spectrum descriptor (RBF kernel)

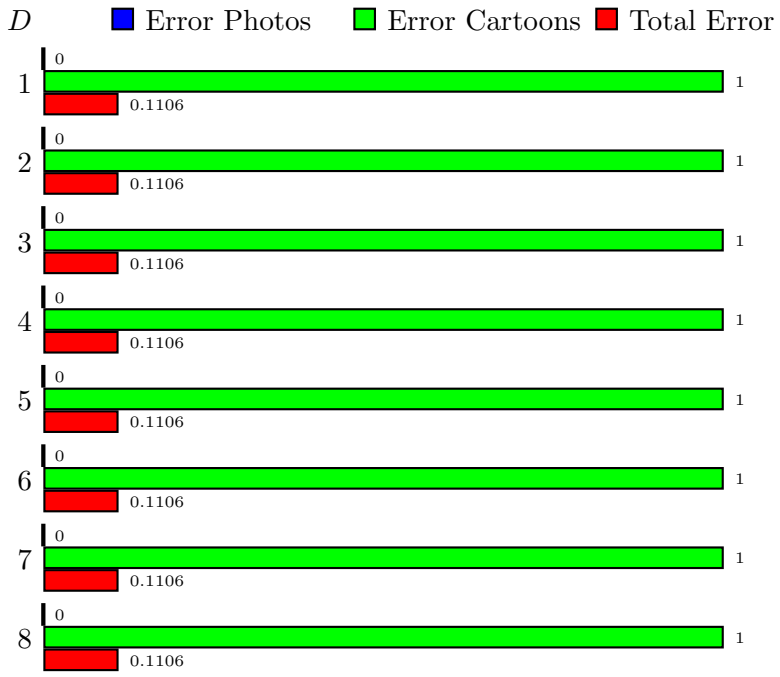


Figure 5.20: Individual performance of the small-scale parabola pattern spectrum descriptor (polynomial kernel)

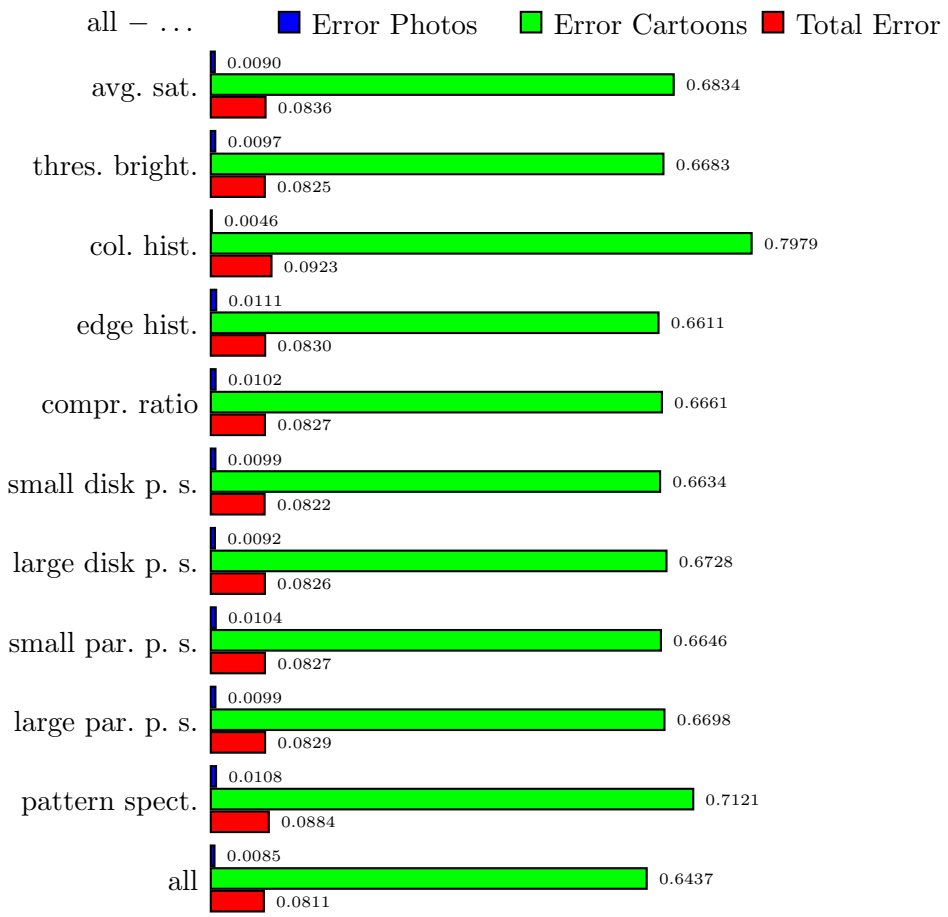


Figure 5.21: Combined performance (RBF kernel, $\sigma^2 = 1/12$)

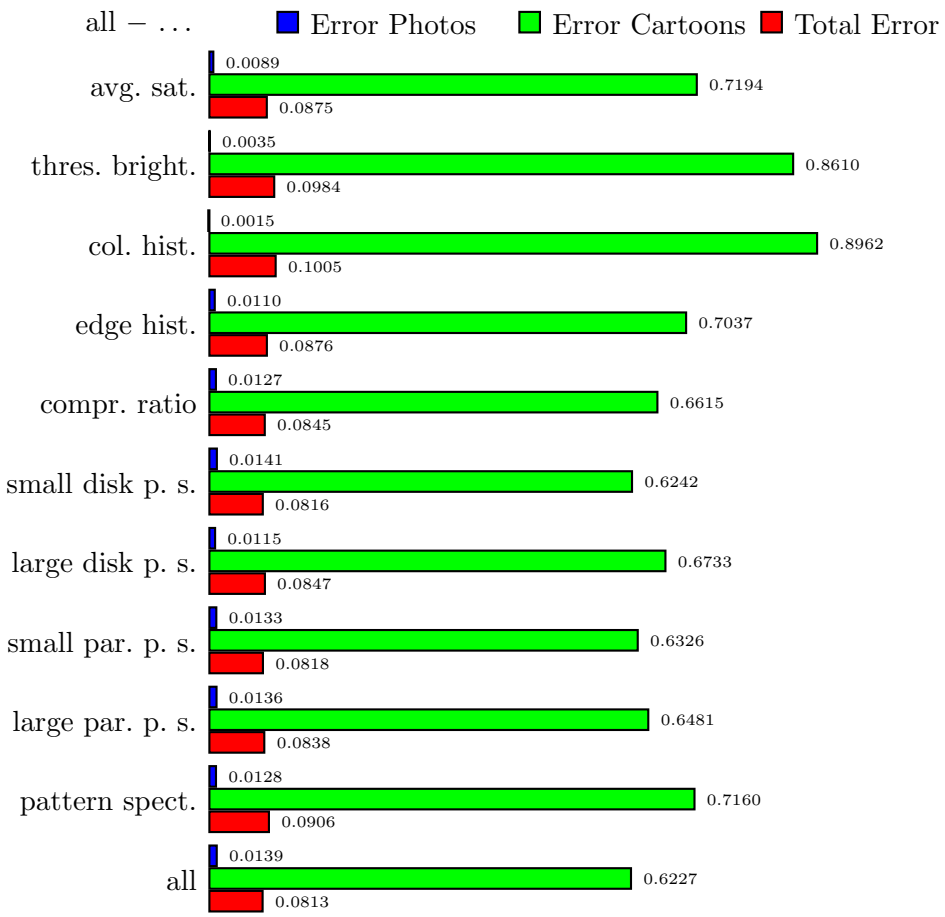


Figure 5.22: Combined performance (polynomial kernel, $D = 7$)

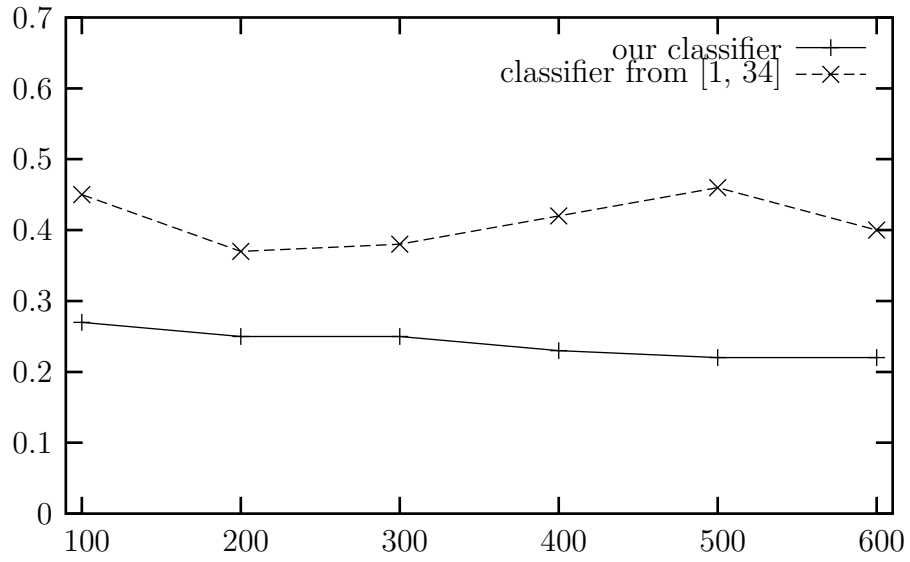


Figure 5.23: Size of learning set versus error. Test set constant size 1,000 photos and 1,000 cartoons. (From [11].)

Chapter 6

CONCLUSIONS

The objective of this work was to filter out cartoons from the returned results in the TREC-2002 in order to improve the retrieval performance of the Probabilistic Gaussian Mixture Model. For this reason we built a classifier used as cartoon detector.

We considered a wide range of image descriptors to capture texture, color, edges, size distribution, and the complexity of the images.

We use SVM learning for training and cross validation. In the model selection we spent considerable effort at tuning the parameters. It is a well-known open research problem to do this automatically, so we had to determine experimentally which type of kernel to use and fix the parameters for the particular kernel, while avoiding overfitting by studying the data too much. The RBF kernel with a variance of $\sigma^2 = 1/12$ turned out to be the most useful when we used all of our descriptors together.

6.1 Application for TREC-2002

How can our classifier be used in the video retrieval? The video-retrieval part of the TREC competition proceeds roughly as follows (see Section 1.2 for definitions etc.): the organizer creates a list topics; each participating group uses their software for retrieving a list of shots for each topic and submits them to the referee. The referee manually classifies each shot from each participant as relevant or irrelevant to the topic. These assessments form the basis to compute recall and precision for all participants.

Our aim in this process was to improve the list of shots returned for a given topic by using our classifier to remove cartoons from the list. We ran our filter on 24 out of 25 topics from the TREC-2002 (see Appendix B; we excluded Topic 088 because it asks for cartoons!) In Table 6.1 we only list

run	topic	# retrieved	avg. prec.	filtered avg. prec.
run1	76	1000	0.5443	0.5444
	84		0.0054	0.0058
	91		0.0146	0.0152
	92		0.0006	0.0007
	total		0.0331	0.0332
run2	76	1000	0.2171	0.2181
	82		0.0222	0.0225
	84		0.0057	0.0061
	91		0.1488	0.1517
	total		0.0225	0.0227
CMU	75	70	0.0793	0.0798
	76		0.1997	0.2125
	78		0.0064	0.0067
	82		0.0021	0.0025
	83		0.0586	0.0492
	84		0.0769	0.0916
	85		0.0032	0.0033
	85		0.0032	0.0033
	86		0.0000	0.0001
	97		0.0088	0.0050
	total		0.0240	0.0248

Table 6.1: Cartoon-filtering effect on TREC-2002 video track search results

the runs and topics where the filtering made a difference. The "run" column in the table indicates the retrieval method used:

run1 text + NIST images from [33]

run2 TEXT + selected components from NIST images from [33]

CMU CMU_MANUAL2 results[1] from [23]

6.2 Discussion

Recall our original question: can we automatically filter out cartoons from the returned results in the TREC-2002 video-track search task? Yes, we can. Can we do it sufficiently well to actually improve the search results? Yes: as we show in Table 6.1, our filter *does* increase the average precision of the search results. In fact, the small magnitude of the difference in the unfiltered and filtered average precision is largely due to the scarcity of cartoons in the full collection—in fact, the retrieval lists contain even fewer cartoons.

We have shown that a generic image classifier based on well-chosen visual features can distinguish cartoons from photos on a difficult video corpus, and identify the graphics in a collection of Web images. The results can most likely be improved further only using higher-level, semantic descriptions. Even for an anticipated easy problem like this one, our experience shows that people use a significant amount of world knowledge (like shining objects, shadows, human body parts, and so on). Low-level characteristics that we have not used (like the spatio-temporal structure of the shot) may help a little bit further, but it is unlikely that this can bridge the semantic gap.

As our project was successful, we employed similar methods in the TREC-2003 video-track search task: we implemented "anchor person," "weather map," "people," and "studio setting" filters and used them in some of the submitted runs.

[1] `http://www-nlpir.nist.gov/projects/t2002v/results/search/search.submissions.without.holes.treceval/`

Appendix A

BIBLIOGRAPHY

[1] V. Athitsos, M. J. Swain, and C. Frankel. Distinguishing photographs and graphics on the world wide web. In *Workshop on Content-Based Access of Image and Video Libraries*, Puerto Rico, June 1997.

[2] A. D. Bagdanov and M. Worring. Multi-scale document description using rectangular granulometries. In D. Lopresti, J. Hu, and R. Kashi, editors, *Document Analysis Systems V*, volume 2423 of *LNCS*, pages 445–456, Princeton, NJ, August 2002. Springer.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: Introduction and benchmarks. Technical report, Department of Computer Science, National Taiwan University, Taipei, Taiwan, 2000. See also `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] A. del Bimbo. *Visual Information Retrieval*. Morgan Kauffman Publisher, 1999.

[5] L. P. Deutsch. RFC 1952: GZIP file format specification version 4.3, 1996. `ftp://ftp.internic.net/rfc/rfc1952.txt`.

[6] E. A. Engbers, R. v. d. Boomgaard, and A. W. M. Smeulders. Decomposition of separable concave structuring functions. *Journal of Mathematical Imaging and Vision*, 15:181–195, 2001.

[7] B. Adams et al. IBM research TREC-2002 video retrieval system. In Voorhees and Buckland [31], pages 289–298.

[8] M. Adler et al. PNG portable network graphics specification. Technical report, W3C, 1996. `http://www.w3.org/Graphics/PNG/`.

[9] Graphics interchange format version 89a. `ftp://ftp.ncsa.uiuc.edu/misc/file.formats/graphics.formats/gif89a.doc`.

[10] L.M. Hurvich and D. Jameson. An oponent process theory of color vision. *Psychological Review*, 45(6):384–404, 1957.

[11] T. I. Ianeva, A. P. de Vries, and H. Röhrig. Detecting cartoons: a case study in automatic video-genre classification. In *2003 IEEE International Conference on Multimeda & Expo*, 2003.

[12] P. Jackway and M. Deriche. Scale-space properties of the multiscale morphological dilation-erosion. *IEEE transactions on pattern analysis and machine intelligence*, 18:38–51, 1996.

[13] A. Lempel and J. Ziv. A universal algorithm for sequential data compression. *IEEE Transactions of Information Theory*, 23(3):337–343, 1977.

[14] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, Berlin, second edition, 1997.

[15] Jean loup Gailly and Mark Adler. `http://www.gzip.org/`.

[16] J. Ma, Y. Zhao, and S. Ahalt. OSU SVM classifier Matlab toolbox. `http://eewww.eng.ohio-state.edu/~maj/osu_svm/`.

[17] M. Miyahara and Y. Yoshida. Matematical transform of rgb color data to muncell hvc color data. *SPIE Visual Communication and Image Processing*, 88(1001):650–657, 1988.

[18] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[19] M. Roach, J. S. Mason, and M. Pawlewski. Motion-based classification of cartoons. In *Int. Symposium on Intelligent Multimedia*, 2001.

[20] N. C. Rowe and B. Frew. Automatic caption localization for photographs on word wide web pages. Technical report, Department of Computer Science, Naval Postgraduate School, 1997.

[21] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[22] Morphology toolbox for matlab. SDC Information Systems. `http://www.mmorph.com/`.

[23] A.F. Smeaton and P. Over. The TREC-2002 video track report. In Voorhees and Buckland [31], pages 69–85.

[24] J. R. Smith and S.-F. Chang. Searching for images and videos on the world wide web. Technical Report 459-96-25, Center for Communications Research, Columbia University, 1996.

[25] P. Soille. *Morphological Image Analysis Principles and Applications*. Springer-Verlag, Berlin,Heidelberg, 1999.

[26] B. T. Truong, C. Dorai, and S. Venkatesh. Automatic genre identification for content-based video categorization. In *Proc. 15th International Conference on Pattern Recognition*, volume II, pages 230–233, Barcelona, Spain, September 2000.

[27] R. v. d. Boomgaard. The morphological equivalent of the Gauss convolution. *Nieuw Archief voor Wiskunde*, 38(3):219–236, 1992.

[28] V.N. Vapnik. *The nature of statistical learning theory*. Springer Verlag, New York, 1995.

[29] V.N. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1995.

[30] R. v.d. Boomgaard and A. W. M. Smeulders. The morphological structure of images, the differential equations of morphological scale-space. *IEEE transactions on pattern analysis and machine intelligence*, 16(4):1101–1113, 1994.

[31] E. M. Voorhees and L. P. Buckland, editors. *The Eleventh Text Retrieval Conference TREC 2002*, number 500–251 in NIST Special Publication, Washington, 2003.

[32] G. K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, April 1991.

[33] T. Westerveld, A.P. de Vries, A. van Ballegooij, F.M. G. de Jong, and D. Hiemstra. A probabilistic multimedia retrieval model and its evaluation. *EURASIP Journal on Applied Signal Processing*, 2003.

[34] M. A. Windhouwer, A. R. Schmidt, and M. L. Kersten. Acoi: A System for Indexing Multimedia Objects. In *International Workshop on Information Integration and Web-based Applications & Services*, Yogyakarta, Indonesia, November 1999.

Appendix B

TREC-2002 TOPICS

Video Topic 075

Text Description: Find shots with Eddie Rickenbacker in them

Image Example: Rickenbacker behind desk

`http://media.auburn.edu/media/952639326043.jpg`

Image Example: Rickenbacker head and torso

`http://media.auburn.edu/media/952639326731.jpg`

Video Example: Rickenbacker on the left

file: 01870b.mpg start: 22m12.077s stop: 22m18.617s

Video Example: Rickenbacker at desk

file: 01870b.mpg start: 22m18.717s stop: 22m43.576s

Video Topic 076

Text Description: Find additional shots with James H. Chandler

Video Example: Chandler

file: 07087.mpg start: 08m02.820s stop: 08m08.359s

Video Example: Chandler on the right

file: 07087.mpg start: 08m14.599s stop: 08m29.614s

Video Example: Chandler behind a desk

file: 07087.mpg start: 09m06.318s stop: 09m12.424s

Video Topic 077

Text Description: Find pictures of George Washington

Image Example: face

`http://www.cia.gov/csi/monograph/firstln/955pres2.gif`

Video Example: face

file: 01681.mpg start: 09m25.938s stop: 09m29.308s

Video Topic 078

Text Description: Find shots with a depiction of Abraham Lincoln

Image Example: shoulders and head

`http://www.americaslibrary.gov/assets/jb/jb_0304_lincoln_1_m.jpg`

Video Example: shoulders and head

file: 01681.mpg start: 09m29.408s stop: 09m31.076s

Video Topic 079

Text Description: Find shots of people spending leisure time at the beach, for example: walking, swimming, sunning, playing in the sand. Some part of the beach or buildings on it should be visible.

Video Example: people on beach, 2 in foreground

file: 01681.mpg start: 08m08.226s stop: 08m10.895s

Video Example: crowds on beach

file: 08698.mpg start: 14m18.616s stop: 14m22.771s

Video Example: kids play in the sand

file: 07977a.mpg start: 08m00.652s stop: 08m08.359s

Video Example: people on the beach with palm trees

file: 06085b.mpg start: 01m44.839s stop: 01m48.209s

Video Topic 080

Text Description: Find shots of one or more musicians: a man or woman playing a music instrument with instrumental music audible. Musician(s) and instrument(s) must be at least partly visible sometime during the shot.

Video Example: Man on stage playing guitar and singing

file: 01880b.mpg start: 03m19.835s stop: 03m28.277s

Video Example: Man playing saxophone

file: 01880b.mpg start: 02m54.109s stop: 03m11.827s

Video Topic 081

Text Description: Find shots of football players

Video Example: two players and a coach

file: 00453.mpg start: 00m27.226s stop: 00m31.732s

Video Example: teams running down the football field

file: 00453.mpg start: 06m02.566s stop: 06m11.565s

Video Example: game close-up

file: 01681.mpg start: 08m00.852s stop: 08m04.849s

Video Example: view from above, then zooming in

file: 08698.mpg start: 16m46.215s stop: 16m52.855s

Video Topic 082

Text Description: Find shots of one or more women standing in long dresses. Dress should be one piece and extend below knees. The entire dress from top to end of dress below knees should be visible at some point.

Video Example: woman on the right in long white dress

file: 00488.mpg start: 10m47.387s stop: 10m48.554s

Video Example: 2 women in long dresses (one blue, one white) walking from left to right

file: 00488.mpg start: 16m17.987s stop: 16m29.332s

Video Example: woman on the left in long yellow dress

file: 19029.mpg start: 07m32.457s stop: 07m38.463s

Video Topic 083

Text Description: Find shots of the Golden Gate Bridge

Image Example: clear side view

`http://www.nonprofitcenters.org/documents/images/golden-gate-view.jpg`

Image Example: red tower and part of span from side

http://www.onroute.com/guides/sanfrancisco/images/golden-gate-sun.gif

Image Example: tower and span from underneath

http://caswww.colorado.edu/courses.d/NFEM.d/NFEM.Images.d/Golden.Gate.gif

Image Example: on bridge

http://www.windscreen.tv/images/San%20Francisco%20Golden%20Gate%20Bridge%2006.12.00.jpg

Image Example: tower and span in fog

http://www.wetterschau.de/wecke/Nebel/golden%20gate.jpg

Video Topic 084

Text Description: Find shots of Price Tower, designed by Frank Lloyd Wright and built in Bartlesville, Oklahoma

Image Example:

http://www.bartlesville.com/images/market/full/price-tower.jpg

Video Topic 085

Text Description: Find shots containing Washington Square Park's arch in New York City. The entire arch should be visible at some point

Video Example: the arch in middle distance

file: 19567b.mpg start: 04m51.394s stop: 04m53.029s

Video Topic 086

Text Description: Find overhead views of cities - downtown and suburbs. The viewpoint should be higher than the highest building visible

Video Example: panoramic view of a city

file: 02103.mpg start: 01m12.006s stop: 01m21.048s

Video Example: aerial view of Detroit by night

file: 08698.mpg start: 01m11.539s stop: 01m30.958s

Video Example: view of city

file: 36539.mpg start: 03m57.373s stop: 03m59.842s

Video Example: aerial view of suburbs

file: 08276d.mpg start: 02m31.353s stop: 02m33.722s

Video Topic 087

Text Description: Find shots of oil fields, rigs, derricks, oil drilling/pumping equipment. Shots just of refineries are not desired

Video Example: pumping equipment, derricks

file: 15965.mpg start: 01m14.976s stop: 01m20.581s

Video Topic 088

Text Description: Find shots with a map (sketch or graphic) of the continental US.

Video Example: flat outline map with graphics superimposed

file: 02103.mpg start: 06m57.621s stop: 07m18.442s

Video Example: flat outline map with graphics superimposed

file: 08829.mpg start: 07m43.334s stop: 08m48.467s

Video Example: US as part of North America - on a curved surface

file: 19304.mpg start: 16m38.007s stop: 16m37.974s

Video Example: flat outline map with graphics superimposed

file: 19400.mpg start: 02m44.466s stop: 03m15.798s

Video Topic 089

Text Description: Find shots of a living butterfly

Image Example:

`http://pt-lobos.parks.state.ca.us/nathis/Monarch.jpg`

Image Example:

`http://insects.ummz.lsa.umich.edu/Images/Lepidoptera/monarch.JPG`

Video Topic 090

Text Description: Find more shots with one or more snow-covered moutain peaks or ridges. Some sky must be visible them behind

Video Example:

file: 01681.mpg start: 02m28.650s stop: 02m31.653s

Video Example:

file: 01681.mpg start: 02m31.686s stop: 02m35.857s

Video Example:

file: 11842.mpg start: 01m07.735s stop: 01m13.941s

Video Topic 091

Text Description: Find shots with one or more parrots

Image Example: close-up of upper half

`http://www.papageienstammtisch-stuttgart.de/LOLA_3.jpg`

Video Example: whole body

file: 15965.mpg start: 01m26.187s stop: 01m33.895s

Video Topic 092

Text Description: Find shots with one or more sailboats, sailing ships, clipper ships, or tall ships - with some sail(s) unfurled

Image Example: clipper ship

`http://tinyurl.com/oy4q`

Image Example: tallship

`http://www.ospreysguide.com/Gallery/Greenport/tall-ship-4.jpg`

Image Example: tallship

`http://www.aweisbecker.com/images/eagle01a.jpg`

Image Example: sailboat

`http://topex-www.jpl.nasa.gov/science/images/sailboat-racing.jpg`

Video Example: multiple sailboats

file: 08261.mpg start: 03m29.511s stop: 03m34.149s

Video Example: multiple sailboats

file: 36539.mpg start: 07m58.883s stop: 08m01.419s

Video Topic 093

Text Description: Find shots about live beef or dairy cattle, individual cows or bulls, herds of cattle.

Video Example: isolated groups in field, audio:'cattle'

file: 00535.mpg start: 01m59.454s stop: 02m05.627s

Video Example: large group

file: 00535.mpg start: 02m05.760s stop: 02m11.933s

Video Example: a few grazing in the foreground

file: 00535.mpg start: 09m59.605s stop: 10m05.878s

Video Example: a few grazing in middle of scene

file: 07980.mpg start: 12m12.239s stop: 12m19.279s

Video Example: horsemen driving a couple head of cattle

file: 06085a.mpg start: 00m43.310s stop: 00m56.624s

Video Topic 094

Text Description: Find more shots of one or more groups of people, a crowd, walking in an urban environment (for example with streets, traffic, and/or buildings).

Video Example: crowd crossing street.

file: 08698.mpg start: 11m09.843s stop: 11m15.915s

Video Example: aerial view of crowd crossing street, light pole, cars passing.

file: 11240a.mpg start: 00m49.884s stop: 01m06.300s

Video Example: urban environment: car, building, sky. Moving people.

file: 19632.mpg start: 01m32.060s stop: 01m36.764s

Video Topic 095

Text Description: Find shots of a nuclear explosion with a mushroom cloud

Video Example:

file: 08829.mpg start: 05m17.954s stop: 05m37.974s

Video Example:

file: 08829.mpg start: 19m52.036s stop: 19m58.709s

Video Example: mushroom cloud against dark background

file: 14104.mpg start: 09m16.995s stop: 09m17.696s

Video Topic 096

Text Description: Find additional shots with one or more US flags flapping

Video Example: closeup

file: 01681.mpg start: 09m52.498s stop: 10m04.176s

Video Example: flag in front of building

file: 38848a.mpg start: 10m21.694s stop: 10m29.435s

Video Topic 097

Text Description: Find more shots with microscopic views of living cells

Video Example:

file: 01356a.mpg start: 10m10.649s stop: 10m16.389s

Video Example:

file: trchns.mpg start: 13m19.373s stop: 13m28.950s

Video Topic 098

Text Description: Find shots with a locomotive (and attached railroad cars if any) approaching the viewer

Video Example: from right

file: 36539.mpg start: 01m31.659s stop: 01m39.934s

Video Example: steam locomtive and train from left

file: 36539.mpg start: 03m22.004s stop: 03m26.141s

Video Example: from left

file: 36539.mpg start: 05m04.374s stop: 05m09.345s

Video Example: from right

file: 36539.mpg start: 06m54.718s stop: 06m59.457s

Video Example: steam locomotive from right

file: 10150.mpg start: 00m01.335s stop: 00m33.300s

Video Topic 099

Text Description: Find shots of a rocket or missile taking off. Simulations are acceptible

Video Example: leaving launching pad

file: 01681.mpg start: 07m38.496s stop: 07m41.966s

Video Example: continues to climb

file: 01681.mpg start: 07m42.800s stop: 07m47.672s

Appendix C

MATLAB SCRIPTS

Here we provide the source code to the central Matlab functions that we wrote in the course of our research.

C.1 Computing Image Descriptors

C.1.1 computeCarAvgSatThrBrightness

```
function computeCarAvgSatThrBrightness(directoryName,threshold)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: computeCarAvgSatThrBrightness(directoryName,threshold)      %
%                                                                  %
% This function computes average saturation and threshold          %
%   brightness of each image in directory directoryName. Results   %
%   are saved in the corresponding 'car_avg_bright.mat' and        %
%   'car_avg_sat_thr.mat' files.                                   %
%                                                                  %
% Parameters:                                                      %
% directoryName -->a full path of image location                   %
% threshold     -->a number between (0.1:0.1:1)                    %
%                                                                  %
% Tzveta Ianeva, 2003                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('computeCarAvgSatThrBrightness');
cd(directoryName);
D=[ dir('*.jpg'); dir('*.gif'); ];
numImages=length(D);
for i=1:numImages
    disp(D(i).name);
    I=imageLoad(D(i).name);
```

```
    Y=rgb2hsv(I);
    V=Y(:,:,3);
    [h,w]=size(V);
    [Vcount,Vx]=imhist(V);
    VcountTr=Vcount(Vx(:,1)>=threshold);
    percentBr=sum(VcountTr)/(h*w);
    % figure,imshow(V),title(['original grayscale image' D(i).name]);
    % figure,imhist(V),title(['grayscale histogram ' D(i).name]);
    S=Y(:,:,2);
    % figure,imhist(S),title(['saturation ' D(i).name]);
    [Scount,Sx]=imhist(S);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % avgS is the average saturation                                    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    avgS = Scount'*Sx/(h*w);
    xName(i,1:length(D(i).name))=D(i).name;
    imageCarsBr(i,:) =[ percentBr];
    imageCarsSat(i,:)=[ avgS ];
end;
saveCar('car_avg_bright', xName, imageCarsBr);
saveCar('car_avg_sat_thr', xName, imageCarsSat);
```

C.1.2 computeCarCompression

```
function computeCarCompression(directoryName)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: computeCarCompression(directoryName)                             %
%                                                                       %
% This function computes the compression descriptor of each image in    %
%   directory directoryName. Results are saved in the corresponding     %
%   'car_compression.mat' file.                                         %
%                                                                       %
% Parameters:                                                           %
% directoryName --> a full path of image location                       %
%                                                                       %
% Tzveta Ianeva, 2003                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('computeCarCompression');
cd(directoryName);
D=[ dir('*.jpg'); dir('*.gif'); ];
%D=dir('*.gif');
numImages=length(D);
for i=1:numImages
    imageName=D(i).name;
    disp(imageName);
```

```
    I=imread(imageName);
    [height,width,channels]=size(I);
    imageSize=height*width*channels;
    if 0 ~= regexpi(imageName, '\.jpg$', 'once')
        decompress = 'djpeg ''%s''';
    elseif 0 ~= regexpi(imageName, '\.gif$', 'once')
        decompress = 'giftopnm ''%s''';
    else
        error('don''t know how to convert to pnm');
    end
    shellCmd=sprintf([decompress,
        '|ppmquant 256 2>/dev/null|pnmtopng|wc -c'],...
        imageName);
    [s,r]=system(shellCmd);
    [ value, count] = sscanf(r,'%d');
    if count ~= 1
        quantCompression(i) = 0;
    else
        quantCompression(i)=value/imageSize;
    end
    xName(i,1:length(D(i).name))=D(i).name;
    imageCars(i,:) = [ quantCompression(i) ];
end;
saveCar('car_compression', xName, imageCars);
```

C.1.3 computeCarColHist

```
function computeCarColHist(directoryName, hBins, sBins, vBins)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: computeCarColHist(directoryName,3,3,5), 3x3x5 comes from 45 %
%        bins color histogram.                                     %
%                                                                  %
% This function computes the color histogram descriptor of each    %
%  image in directory directoryName. Results are saved in the      %
%  corresponding 'car_colorhist.mat' file.                         %
%                                                                  %
% Parameters:                                                      %
% directoryName --> a full path of image location                  %
% hBins         --> a number of bins in the h (hue) chanel of HSV  %
% sBins         --> a number of bins in the s (saturation) chanel  %
% vBins         --> a number of bins in the v (grayscale) chanel   %
%                                                                  %
% Tzveta Ianeva, 2003                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('computeCarColHist');
```

```
cd(directoryName);
D=[ dir('*.jpg'); dir('*.gif'); ];
%D=dir('*.gif');
%D=dir('*.jpg');
numImages=length(D);
for i=1:numImages
    disp(D(i).name);
    I=imageLoad(D(i).name);
    % figure,imshow(I),title('original image');
    HSV=rgb2hsv(I);
    %V=HSV(:,:,3);
    [height,width]=size(HSV(:,:,3));
    % figure,imshow(V),title(['original grayscale image' D(i).name]);
    % Y=rgb2ycbcr(I);
    % S=HSV(:,:,2);
    % H=HSV(:,:,1);
    ch = round(HSV(:,:,1)*(hBins-1))+1;
    sh = round(HSV(:,:,2)*(sBins-1))+1;
    vh = round(HSV(:,:,3)*(vBins-1))+1;
    flat = ch + hBins*(sh-1) + hBins*sBins*(vh-1);
    coldistrib = histc(flat(:)', 1:hBins*sBins*vBins)/(height*width);
    % count=zeros(hBins,sBins,vBins);
    % [h,s,v]=size(count);
    %for x=1:height
    %    for y=1:width
    %        %ch=round(H(x,y)*(hBins-1))+1;
    %        %sh=round(S(x,y)*(sBins-1))+1;
    %        %vh=round(V(x,y)*(vBins-1))+1;
    %        %count(ch(x,y),sh,vh)=count(ch,sh,vh)+1;
    %        %           count(ch,sh,vh)
    %    end;
    % end;
    %coldistrib=[count(:)/(height*width)];
    imageCars(i, :) = coldistrib;
    xName(i,1:length(D(i).name))=D(i).name;
    %cars(1,i)={coldistrib}
    %% FOR VISUALIZATION UNCOMMENT THE FOLLOWING
    %figure;
    %phandles = contourslice(count, [], [], 1:vBins, 5);
    %view(3); axis tight
    %set(phandles,'LineWidth',2)
    xName(i,1:length(D(i).name))=D(i).name;
end;
saveCar('car_colorhist', xName, imageCars);
```

C.1.4 computeCarEdgeDir

```
function computeCarEdgeDir(directoryName)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: computeCarEdgeDir(directoryName)                                       %
%                                                                             %
% This function computes the edge descriptor of each image in                %
%   directory directoryName. Results are saved in the corresponding           %
%   'car_edge_dir.mat' file.                                                  %
%                                                                             %
% Parameters:                                                                 %
% directoryName -->a full path of image location                              %
%                                                                             %
% Tzveta Ianeva, 2003                                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('computeCarEdgeDir');
cd(directoryName);
D=[ dir('*.jpg'); dir('*.gif'); ];
% D=dir('*.gif');
% D=dir('*.jpg');
numImages=length(D);
horiz_sobel=fspecial('sobel');
vert_sobel=horiz_sobel';
nAngle=8;
nMagnitude=5;
maxMagnitude=sqrt(20);
for i=1:numImages
    disp(D(i).name);
    xName(i,1:length(D(i).name))=D(i).name;
    I=imageLoad(D(i).name);
    % figure,imshow(I);
    V=im2double(rgb2gray(I));
    % figure,imshow(V);
    X=conv2(V,horiz_sobel,'valid');
    % figure,imshow(abs(X),256);
    Y=conv2(V,vert_sobel,'valid');
    if min(size(X)) > 0 && min(size(Y)) > 0
        % figure,imshow(abs(Y),256);
        % figure,imshow(abs(X)+abs(Y),256);
        Angle=atan2(Y,X);
        Magnitude=sqrt(X.*X+Y.*Y);
        AngleQuant = 1+round((Angle+pi)/(2*pi)*(nAngle-1));
        MagnitudeQuant = 1+round(Magnitude / ...
            maxMagnitude*(nMagnitude-1));
        Quant = (AngleQuant-1)*nMagnitude + MagnitudeQuant;
```

```
        [ height, width ] = size(Angle);
        histogram = histc(Quant(:)', 1:(nMagnitude*nAngle)) / ...
            (height*width);
        imageCars(i,:) = histogram;
    else
        imageCars(i,:) = zeros(1,nAngle*nMagnitude);
    end
end;
saveCar('car_edge_dir', xName, imageCars);
```

C.1.5 computeCarGranulometry1

```
function computeCarGranulometry1(directoryName)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: computeCarGranulometry1(directoryName)                              %
%                                                                          %
% This function computes the multi-scale pat. spectrum descriptor of       %
%   each image in directory directoryName.                                 %
%                                                                          %
% Results are saved in the corresponding 'car_granulom1_parbig.mat', %
%   'car_granulom1_disksmall.mat', 'car_granulom1_diskbig.mat', and        %
%   'car_granulom1_disksmall.mat' files.                                   %
%                                                                          %
% Parameters: directoryName --> a full path of image location,             %
%                                                                          %
% Tzveta Ianeva, 2003                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('computeCarGranulometry1');
cd(directoryName);
D=[ dir('*.jpg'); dir('*.gif') ];
numImages=length(D);
%element='Parabola';
for i=1:numImages
    disp(D(i).name);
    I=imageLoad(D(i).name);
    % figure,imshow(I),title('original image');
    Y=rgb2hsv(I);
    V=Y(:,:,3);
    % figure,imshow(V),title(['original grayscale image' D(i).name]);
    % figure,imhist(V),title(['grayscale histogram ' D(i).name]);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % sDisrib1 is a function that returns the remaining object       %
    % surface area after each opening with SE parabola               %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % surfareaS=sDistrib1(V,20,0.01); % old version with smallest
```

```
    %                                    scale parabola
    % surfareaB=sDistrib1(V,10,5); % old version with SE parabola
    surfareaParSmall=sDistrib1(V,20,1);
    surfareaDiskSmall=DiskDistrib(V,20,1);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % DiskDistrib is a function that returns the remaining object   %
    % surface area after each opening with SE disk                  %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    surfareaParBig=sDistrib1(V,10,5);
    surfareaDiskBig=DiskDistrib(V,10,5);
    surfParSmall(1,i)={surfareaParSmall};
    surfDiskSmall(1,i)={surfareaDiskSmall};
    surfParBig(1,i)={surfareaParBig};
    surfDiskBig(1,i)={surfareaDiskBig};
    %surfS(1,i)={surfareaS}; % old version
    %surfB(1,i)={surfareaB}; % old version
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % diff --> First derivative of the surface area array, which     %
    % contains the size distributions of the objects in the image    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    derivsurfareaParSmall=diff(surfareaParSmall);
    derivsurfareaDiskSmall=diff(surfareaDiskSmall);
    derivsurfareaParBig=diff(surfareaParBig);
    derivsurfareaDiskBig=diff(surfareaDiskBig);
    % derivsurfareaS  =diff(surfareaS);
    % derivsurfareaM  =diff(surfareaM);
    derivsurfParSmall(1,i)={derivsurfareaParSmall};
    derivsurfDiskSmall(1,i)={derivsurfareaDiskSmall};
    derivsurfParBig(1,i)={derivsurfareaParBig};
    derivsurfDiskBig(1,i)={derivsurfareaDiskBig};
    % derivsurfBig(1,i)={derivsurfareaBig};
    xName(i,1:length(D(i).name))=D(i).name;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % imageCars contains measurements of the image                   %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    imageCarsParSmall(i,:)   = [derivsurfareaParSmall(:)]';
    imageCarsDiskSmall(i,:) = [derivsurfareaDiskSmall(:)]';
    imageCarsParBig(i,:)     = [derivsurfareaParBig(:)]';
    imageCarsDiskBig(i,:)    = [derivsurfareaDiskBig(:)]';
    % imageCarsS(i,:) = [derivsurfareaS(:)]';
    % imageCarsB(i,:) = [derivsurfareaB(:)]';
end;
% saveCar('car_granulom1_small',    xName, imageCarsS); % old version
% saveCar('car_granulom1_big',      xName, imageCarsB); % old version
```

```
saveCar('car_granulom1_parsmall',  xName, imageCarsParSmall);
saveCar('car_granulom1_parbig',    xName, imageCarsParBig);
saveCar('car_granulom1_disksmall', xName, imageCarsDiskSmall);
saveCar('car_granulom1_diskbig',   xName, imageCarsDiskBig);
% x=0:numOpenings;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOT OF THE SIZE DISTRIBUTION                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% UNCOMMENT THE FOLLOWING LINES FOR VISUALIZATION
% plotSurf1(20,numImages,surfParSmall(1,:),  xName, 'small parabola');
% plotSurf1(20,numImages,surfDiskSmall(1,:), xName, 'small disk');
% plotSurf1(10,numImages,surfParBig(1,:),    xName, 'big parabola');
% plotSurf1(10,numImages,surfDiskBig(1,:),   xName, 'big disk');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOT OF THE PATTERN SPECTRUM                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% UNCOMMENT THE FOLLOWING LINES FOR VISUALIZATION
% plotDerivSurf1(numImages,derivsurfParSmall(1,:),  xName, 'small parabola');
% plotDerivSurf1(numImages,derivsurfDiskSmall(1,:), xName, 'small disk');
% plotDerivSurf1(numImages,derivsurfParBig(1,:),    xName, 'big parabola');
% plotDerivSurf1(numImages,derivsurfDiskBig(1,:),   xName, 'big disk');
```

C.1.6 computeCarDim

```
function computeCarDim(directoryName)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: computeCarDim(directoryName)                                  %
%                                                                    %
% This function computes the dimension ratio descriptors of each     %
%   image in directory directoryName. Usefull just for WWW images.   %
%   Results are saved in the corresponding 'car_dim_aspect.mat' and  %
%   'car_dim_min.mat' files.                                         %
%                                                                    %
% Parameters:                                                        %
% directoryName --> a full path of image location                    %
%                                                                    %
% Tzveta Ianeva, 2003                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('computeCarDim');
cd(directoryName);
D=[ dir('*.jpg'); dir('*.gif'); ];
%D=dir('*.gif');
numImages=length(D);
for i=1:numImages
    disp(D(i).name);
```

```
    I=imageLoad(D(i).name);
    height = size(I, 1);
    width = size(I, 2);
    aspect = height / width;
    if aspect < 1
        aspectAdjusted = [ aspect / 2 ];
    else
        aspectAdjusted = [ 1 - 1/(2*aspect) ];
    end
    minDimAdjusted = min(height, width) / 2000;
    if minDimAdjusted > 1
        error('minDimAdjusted greater 1')
    end
    % aspectAdjusted
    % minDimAdjusted
    imageCarAspect(i, :) = [ aspectAdjusted ];
    imageCarMinDim(i, :) = [ minDimAdjusted ];
    xName(i,1:length(D(i).name))=D(i).name;
end;
saveCar('car_dim_aspect', xName, imageCarAspect);
saveCar('car_dim_min',    xName, imageCarMinDim);
```

C.1.7 computeCarFileType

```
function computeCarFileType(directoryName)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: computeCarFileType(directoryName)                                  %
%                                                                         %
% This function computes the file type descriptor of each                 %
%   image in directory directoryName. Usefull just for WWW images.        %
%   Results are saved in the corresponding 'car_fyle_type.mat' file.      %
%                                                                         %
% Parameters:                                                             %
% directoryName --> a full path of image location                         %
%                                                                         %
% Tzveta Ianeva, 2003                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('computeCarCompression');
cd(directoryName);
D=[ dir('*.jpg'); dir('*.gif'); ];
% D=dir('*.gif');
numImages=length(D);
for i=1:numImages
    imageName=D(i).name;
    disp(imageName);
```

```
    if 0 ~= regexpi(imageName, '\.jpg$', 'once')
        isGif = 0;
    elseif 0 ~= regexpi(imageName, '\.gif$', 'once')
        isGif = 1;
    else
        error('don''t_know_file_type');
    end
    xName(i,1:length(D(i).name))=D(i).name;
    imageCars(i,:) = [ isGif ];
end;
saveCar('car_file_type', xName, imageCars);
```

C.2 Learning and Classifying

C.2.1 learnAndTest

```
function learnAndTest( topDirectory, learnDirectory, numLearn0, ...
    numLearn1, numTest0, numTest1, kernel, degree )
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: learnAndTest(topDirectory, learnDirectory, numLearn0,                   %
%        numLearn1, numTest0, numTest1, kernel, degree )                       %
%                                                                              %
% This function loads all characteristics from                                 %
%   topDirectory/xa?-images and performs training and crossvalidation.         %
%                                                                              %
% Parameters:                                                                  %
% topDirectory --> top directory of xa?-images directories.                    %
% kernel       --> use 'r' for RBF and 'p' for Polinomial kernel               %
% degree       --> polinomial kernel, (<X(:,i),X(:,j)+1)^ degree               %
%                  RBF kernel, exp(-degree*|X(:,i)-X(:,j)|^ 2                  %
%                                                                              %
% For learning:                                                                %
% If learnDirectory == '' then randomly select from xa?-images                 %
%   numLearn0 photos for learning and numLearn1 cartoons for learning.         %
%   Otherwise use all photos and cartoons from learnDirectory.                 %
%                                                                              %
% For testing:                                                                 %
% Load numTest0 photos and numTest1 cartoons. If numTest0 == 0 all             %
%   remaining photos will be used for testing and if numTest1 == 0             %
%   all remaining cartoons will be used for testing.                           %
%                                                                              %
% Tzveta Ianeva, 2003                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[ fileNames, car ] = loadAllCarDir(topDirectory);
% to get more random selections:
```

```
rand('state',sum(100*clock));
% to output the state of the random number generator
% (for repeatability:) randomState=rand('state')
[ numFiles, strLength ] = size(fileNames);
perm = randperm(numFiles);
fileNames = fileNames(perm,:);
car = car(perm,:);
indicator0 = fileNames(:, 1) == '0';
indicator1 = fileNames(:, 1) == '1';
num0 = sum(indicator0)
num1 = sum(indicator1)
if numTest0 == 0
    numTest0 = num0 - numLearn0;
end
if numTest1 == 0
    numTest1 = num1 - numLearn1;
end
if numLearn0 + numTest0 > num0
    error('not enough photos');
end
if numLearn1 + numTest1 > num1
    error('not enough cartoons');
end
car0 = car(indicator0, :);
car1 = car(indicator1, :);
TestLabels   = [ zeros(1, numTest0),  ones(1, numTest1)  ];
TestSamples  = [ car0((numLearn0+1):(numLearn0+numTest0), :) ;
    car1((numLearn1+1):(numLearn1+numTest1), :) ]';
if strcmp(learnDirectory, '')
    LearnSamples = [ car0(1:numLearn0, :) ;
        car1(1:numLearn1, :) ]';
else
    if numLearn0 ~= 0 || numLearn1 ~=0
        error('numLearn0 and numLearn1 must be zero');
    end;
    [ learnFileNames, learnCar ] = loadAllCar(learnDirectory);
    learnIndicator0 = learnFileNames(:, 1) == '0';
    learnIndicator1 = learnFileNames(:, 1) == '1';
    numLearn0 = sum(learnIndicator0)
    numLearn1 = sum(learnIndicator1)
    LearnSamples = [ learnCar(learnIndicator0, :) ;
        learnCar(learnIndicator1, :) ]';
end
LearnLabels  = [ zeros(1, numLearn0), ones(1, numLearn1) ];
```

```
% numLearn0
% numLearn1
% numTest0
% numTest1

% used for classifying input patterns using the nonlinear
% SVM Classifier that will be constructed
TrueLabelsTrain=LearnLabels;
SamplesTrain=LearnSamples;
save DemoData SamplesTrain TrueLabelsTrain

% save debug LearnLabels LearnSamples learnFileNames TestLabels
% TestSamples;

if strcmp(kernel,'r')
    % RBF kernel
    degree
    [AlphaY, SVs, Bias, Parameters, nSV, nLabel] = ...
        RbfSVC(LearnSamples, LearnLabels, degree);
    nSV
    save SVMClassifierRbf AlphaY SVs Bias Parameters nSV nLabel;
elseif strcmp(kernel,'p')
    % Polynomial kernel
    degree
    [AlphaY, SVs, Bias, Parameters, nSV, nLabel] = ...
        PolySVC(LearnSamples, LearnLabels, degree);
    nSV
    save SVMClassifierPol AlphaY SVs Bias Parameters nSV nLabel;
else
    error('Please use ''r'' for RBF or ''p'' for polinomial kernel !');
end

[ClassRate, DecisionValue, Ns, ConfMatrix, PreLabels] = ...
    SVMTest(TestSamples, TestLabels, AlphaY, SVs, Bias,Parameters, ...
    nSV, nLabel);
ClassRate;
ErrorRate = 1 - ClassRate
ConfMatrix;
ErrorMatrix = 1 - ConfMatrix

cartoonDecisionValue = DecisionValue(PreLabels == 1)';
cartoonLabels = TestLabels(PreLabels == 1)';
[sortedCartoonDecisionValue, permutation] = ...
```

```
    sortrows(cartoonDecisionValue);
sortedCartoonLabels = cartoonLabels(permutation);
```

C.2.2 mergeAllCarDir

```
function [ fileNames, car ] = mergeAllCarDir(topDirectory)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: mergeAllCarDir(topDirectory)                                          %
%                                                                            %
% This function loads and merges all characteristics from xa?-images        %
%   below topDirectory.                                                      %
%                                                                            %
% Parameters:                                                                %
% topDirectory --> a full path of image location top directory               %
%                                                                            %
% Output:                                                                    %
% FileNames    --> a (n-by-m) matrix with n file names as length-m           %
%                  strings                                                   %
%                                                                            %
% car          --> a (n-by-k) matrix for a total of n files and k            %
%                  real-valued descriptors or descriptors components         %
%                                                                            %
% Tzveta Ianeva, 2003                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cd(topDirectory);
delete('car_all.mat');
dirs=dir('x*-images');
fileNames = [ ];
car = [ ];
for d=1:length(dirs)
    disp(dirs(d).name);
    fullDirectoryName = [ topDirectory, '/', dirs(d).name ];
    [ newFileNames, newCar ] = loadAllCar(fullDirectoryName);
    l = max(size(fileNames, 2), size(newFileNames, 2));
    fileNames = [ padStrings(fileNames, l); padStrings(newFileNames, l) ];
    car = [ car; newCar ];
end
cd(topDirectory);
save('car_all', 'fileNames', 'car');
```

C.2.3 loadAllCar

```
function [ fileNames, car ] = loadAllCar(directoryName)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: loadAllCar(directoryName)                                             %
%                                                                            %
% This function selects descriptors for training and crossvalidation.        %
```

```
%                                                                       %
% Parameters:                                                           %
% directoryName -->a full path of image location                        %
%                                                                       %
% Output:                                                               %
% FileNames     -->a (n-by-m) matrix with n file names as length-m      %
%                   strings                                             %
%                                                                       %
% car           -->a (n-by-k) matrix for a total of n files and k       %
%                   real-valued descriptors or descriptors components   %
%                                                                       %
% Tzveta Ianeva, 2003                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cd(directoryName);
fileNames = ['invalid'];
car = [];
[fileNames, car] = mergeCar('car_granulom1_disksmall', fileNames, car);
[fileNames, car] = mergeCar('car_granulom1_diskbig',   fileNames, car);
[fileNames, car] = mergeCar('car_granulom1_parsmall',  fileNames, car);
[fileNames, car] = mergeCar('car_granulom1_parbig',    fileNames, car);
[fileNames, car] = mergeCar('car_avg_bright',          fileNames, car);
[fileNames, car] = mergeCar('car_avg_sat_thr',         fileNames, car);
[fileNames, car] = mergeCar('car_compression',         fileNames, car);
[fileNames, car] = mergeCar('car_colorhist',           fileNames, car);
[fileNames, car] = mergeCar('car_edge_dir',            fileNames, car);

% Additional descriptors on WWW images
[fileNames, car] = mergeCar('car_dim_aspect',        fileNames, car);
[fileNames, car] = mergeCar('car_dim_min',           fileNames, car);
[fileNames, car] = mergeCar('car_file_type',         fileNames, car);

% OTHER IMAGE DESCRIPTORS THAT WE DO NOT USE IN OUR FINAL VERSION:
%[fileNames, car] = mergeCar('car_avg_bright_recip',   fileNames, car);
%[fileNames, car] = mergeCar('car_avg_sat_thr_recip',  fileNames, car);
%[fileNames, car] = mergeCar('car_granulom_10',        fileNames, car);
%[fileNames, car] = mergeCar('car_granulom_10_recip',  fileNames, car);
%[fileNames, car] = mergeCar('car_granulom_var',       fileNames, car);
%[fileNames, car] = mergeCar('car_granulom_var_recip', fileNames, car);
%[fileNames, car] = mergeCar('car_compression_recip',  fileNames, car);
%[fileNames, car] = mergeCar('car_granulom1_big',      fileNames, car);
%[fileNames, car] = mergeCar('car_granulom1_small',    fileNames, car);
```

C.2.4 mergeCar

```
function [newFileNames, newSamples] = mergeCar(carName, ...
```

```
    oldFileNames, oldSamples)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: mergeCar(carName, oldFileNames, oldSamples                          %
%                                                                          %
% This function merges (i.e., join) characteristics by fileNames          %
%                                                                          %
% Parameters:                                                              %
% oldFileNames --> a (n-by-m) matrix with n file names                     %
%                    as length-m strings                                   %
% oldSamples   --> a (n-by-k) matrix with k characteristics for each       %
%                    file                                                  %
% carName      --> the name for the characteristics to be added            %
%                                                                          %
% Output:                                                                  %
% newFileNames --> a (p-by-q) matrix with p<=n and q>=m with length-q      %
%                    names of the files for which there exist old and      %
%                    new characteristics                                   %
%                                                                          %
% newSamples   --> a (p-by-r) matrix with r=k+s containing for each        %
%                  of the p files the k old and the s new                  %
%                  characteristics                                         %
%                                                                          %
% Tzveta Ianeva, 2003                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% join with empty set gives empty set
if length(oldFileNames) == 0
    newFileNames = [ ];
    newSamples = [ ];
    return;
end

% if the characteristic doesn't exist return empty file names and
% charceteristics
carFileName = [ carName, '.mat' ];
if exist(carFileName, 'file') ~= 2
    warning(sprintf('%s/%s does not exist', pwd, carFileName));
    newFileNames = [ ];
    newSamples = [ ];
    return;
end

% if this is the first characteristic load instead of join
if length(oldSamples) == 0
    load(carName, 'fileNames', 'car');
```

```
    % newFileNames we use when there was mistake in the manual
    % classification and we change the file name (to avoid
    % recomputing of the caracteristics)
    % newFileNames = fixFileNames(fileNames);
    newFileNames = fileNames;
    newSamples = car;
    return;
end;

% now we have handled all boudary cases and can do the real work

[ n, m ] = size(oldFileNames);
[ nn, k ] = size(oldSamples);
if n ~= nn
    error('need the same number of old files and old characteristics');
end

% load characteristics
load(carName, 'fileNames', 'car');
% fileNames = fixFileNames(fileNames);
[ t, l ] = size(fileNames);
[ tt, s ] = size(car);
if t ~= tt
    error('need the same number of new files and new characteristics');
end

% compute output sizes
r = k+s;
q = max(m, l);

% bring fileNames and oldFileNames to the same length
if m<q
    spaces=' ';
    for i=2:q-m
        spaces=[ spaces, ' ' ];
    end
    for i=1:n
        oldFileNames(i, :)=[ oldFileNames(i, :), spaces ];
    end
end
if l<q
    spaces=' ';
    for i=2:q-l
        spaces=[ spaces, ' ' ];
```

```
    end
    for i=1:t
        fileNames(i,:)=[fileNames(i,:), spaces];
    end
end

% sort everything alphabetically by file name
[sortedOldFileNames, oldPerm] = sortrows(oldFileNames);
oldFileNames=sortedOldFileNames;
oldSamples=oldSamples(oldPerm,:);
[sortedFileNames, perm] = sortrows(fileNames);
fileNames=sortedFileNames;
car=car(perm,:);

% merge the arrays and keep only files that are in the old and the
% new chars
done = false;
oldIndex = 1;
index = 1;
newIndex = 1;
% oldFileNames
% fileNames

% the next line pre-allocates the newSamples array for performance
% reasons
newSamples = zeros([min(size(oldSamples,1), size(car,1)), ...
        size(oldSamples,2) + size(car,2)]);
while ~done
    if isLessOrEqualLexi(fileNames(index), oldFileNames(oldIndex))
        while index <= t && ...
                ~isLessOrEqualLexi(oldFileNames(oldIndex), ...
                fileNames(index))
            index = index + 1;
        end
    else
        while oldIndex <= n && ~isLessOrEqualLexi(fileNames(index), ...
                OldFileNames(oldIndex))
            oldIndex = oldIndex + 1;
        end
    end
    if index > t
        done = true;
    elseif oldIndex > n
        done = true;
```

```
    end
    if ~done
        if ~strcmp(fileNames(index), oldFileNames(oldIndex))
            error('something_went_wrong');
        end
        newFileNames(newIndex, :) = fileNames(index, :);
        newSamples(newIndex, :) = ...
            [ oldSamples(oldIndex, :), car(index, :) ];
        newIndex = newIndex + 1;
        oldIndex = oldIndex + 1;
        index = index + 1;
        if index > t
            done = true;
        elseif oldIndex > n
            done = true;
        end
    end
end
% throw away what we pre-allocated too much
newSamples = newSamples(1:newIndex-1, :);

% (lexicographically) compare stringA and stringB
function result = isLessOrEqualLexi(stringA, stringB)
i = 1;
l = min(length(stringA), length(stringB));
while i <= l && stringA(i) == stringB(i)
    i = i+1;
end
if i <= l
    result = (stringA(i) <= stringB(i));
else
    result = (length(stringA) == l);
end
```

C.2.5 computeCarAll

```
function computeCarAll(directoryName)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: computeCarAll(directoryName)                                          %
%                                                                            %
% This function first deletes all 'car_*.mat' files and                      %
%   calls all other functions that compute image descriptors                 %
%                                                                            %
% Parameters:                                                                %
% directoryName --> a full path of image location                            %
```

```
%                                                                       %
% Tzveta Ianeva, 2003                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cd(directoryName);
delete('car_*.mat');
computeCarAvgSatThrBrightness(directory);
computeCarCompression(directory);
computeCarColHist(directory, 3, 3, 5);
computeCarEdgeDir(directory);
computeCarGranulometry1(directory);
computeCarDim(directory);       % to use just on WWW images
computeCarFileType(directory); % to use just on WWW images
```

C.3 Classifying new data

C.3.1 classifyDirectory

```
function classifyDirectory(classifier, directory)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use: classifyDirectory(classifier, directory)                         %
%                                                                       %
% This function classifies new (unlabeled) data in 'directory' as       %
%   cartoon (1) or  photo (0).                                          %
%                                                                       %
% Parameters:                                                           %
% classifier -->a full path of the classifier location                  %
% directory  -->a full path of new (unlabeled) data to be classified    %
%                                                                       %
% Tzveta Ianeva, 2003                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load(classifier);
[fileNames, car] = loadAllCar(directory);
[ num, len ] = size(fileNames);
Samples = car';
[Labels, DecisionValue] = ...
    SVMClass(Samples, AlphaY, SVs, Bias, Parameters, nSV, nLabel);
fileNamesPhoto(:,:) = fileNames(Labels' == 0,:);
[ numPhoto, lenPhoto ] = size(fileNamesPhoto);
numPhoto
fileNamesCartoon(:,:) = fileNames(Labels' == 1,:);
[ numCartoon, lenCartoon ] = size(fileNamesCartoon);
numCartoon
cd(directory);
fd=fopen('classification.txt', 'W');
fprintf(fd,'%s\n\n', directory);
```

```
for i=1:num
    if (Labels(1,i) == 1)
        disp(sprintf('%d %f %s', Labels(1,i), DecisionValue(1,i), ...
            fileNames(i,:)));
        fprintf(fd, '%d %s\n', Labels(1,i), fileNames(i,:));
    end
end
fclose(fd);
```