Chapter 13
Optimizing Feature Calculation in Adaptive Machine Vision Systems

Christian Eitzinger and Stefan Thumfart

Abstract A classifier's accuracy substantially depends on the features that are utilized to characterize an input sample. The selection of a representative and—ideally—small set of features that yields high discriminative power is an important step in setting up a classification system. The features are a set of functions that transform the raw input data (an image in the case of machine vision systems) into a vector of real numbers. This transformation may be a quite complex algorithm, with lots of parameters to tune and consequently with much room for optimization. In order to efficiently use this additional room for optimizing the features, we propose an integrated optimization step that adapts the feature parameters in such a way that the separation of the classes in feature space is improved, thus reducing the number of misclassifications. Furthermore, these optimization techniques may be used to "shape" the decision boundary in such a way that it can be easily modeled by a classifier. After covering the relevant elements of the theory behind this automatic feature optimization process, we will demonstrate and assess the performance on two typical machine vision applications. The first one is a quality control task, where different types of defects need to be distinguished, and the second example is a texture classification problem as it appears in image segmentation tasks. We will show how the optimization process can be successfully applied in morphological and textural features that both offer a number of parameters to tune and select.

13.1 Introduction

In many classification tasks, the investigations start with a set of features that are the input to various machine learning structures, such as classifiers. In nonstationary learning environments,the classifier is adjusted to adapt to changes in the concepts

C. Eitzinger (✉) • S. Thumfart
Profactor GmbH Im Stadtgut A2, 4407 Steyr-Gleink, Austria
e-mail: christian.eitzinger@profactor.at; stefan.thumfart@profactor.at

M. Sayed-Mouchaweh and E. Lughofer (eds.), *Learning in Non-Stationary Environments: Methods and Applications*, DOI 10.1007/978-1-4419-8020-5_13,

or rules that apply. However, quite often we find situations where the rules should remain the same, but the underlying processes that generate the input data (the features) have changed. This may quickly turn an initially well-posed classification task into an overly complex problem with weird decision boundaries. This may happen, for example, in a machine vision system, when the surface properties of the objects change. The statistical properties of the features will change, which may increase the complexity of the following classification tasks and may make it hard for the classifier to find reasonable decision boundaries. On-line adaptation and evolution of the classifiers (as applied in Chap. 7 for on-line surface inspection problems) may help to overcome these problems by permanent adjustments of the decision boundaries; however, the real class labels are not always available or may require high costs due to significant operators' efforts.

Another promising approach to counter such a change is to adapt the procedures that generate the features in such a way that they compensate for the change in surface properties. This chapter thus focuses on what can be achieved by adapting feature calculation. Features are always application specific and are often assumed to be carefully chosen by an expert, who makes sure that these features are relevant for the task. However, it has to be understood that the resulting feature vector is just a very low-dimensional representation of the object and that much—possibly relevant—information is already lost by converting the raw data of the object into a set of features. In fact, the feature calculation is the step that performs the largest reduction in the dimension of the problem. A typical image used in machine vision applications is several thousand pixels wide and thus may be considered a data vector coming from a 10^6- to 10^7-dimensional space. Clearly, the gray values of neighboring pixels are highly correlated, and images used in typical machine vision applications are usually very far from filling up this huge space. Instead, they are restricted to a comparably small subspace, which allows a compression of the information by means of features. Feature calculation thus reduces this high-dimensional space to a representation with dimensions in a typical range of 20–200. Depending on the number of samples that are available as training data and also on the properties of the features, the dimension needs to be reduced further before applying machine learning methods and classifiers. Therefore, a feature selection step is used that typically reduces the number of features down to 5–15 features.

Essentially, feature selection tries to select a subset of features that is optimal for the task at hand. Optimal here means that the subset contains low redundancy and that correlated features are removed while preserving most of the relevant information. This approach depends on various hypotheses about the distribution of the features and is called "filter approach". Alternatively, one may directly select a subset in such a way that the classification accuracy is optimized. This "wrapper approach" does not require any additional hypotheses, but depends on the classifier that is used. Some classifiers have a built-in feature selection process, for example, decision trees that select a single feature for the decision that is to be made at each node.

Table 13.1 Reduction of dimension when processing an image

Processing step	Data representation	Dimension
Raw (image) data	Image	10^6–10^7
Initial feature space	Feature vector	20–200
Feature space after selection	Feature vector	5–15
Classification result	Discrete value	1

The classifier then performs the final data reduction step and reduces the feature space to a one-dimensional space of a small, discrete set of classes. Table 13.1 illustrates this data reduction.

Within this chapter, we want to focus on the first processing step that converts the raw input data to an initial set of features. This processing step builds the basis for the downstream processing, and it also performs the most significant reduction and compression of data. The algorithms used for this reduction may be highly complex and are domain specific. For example, in the analysis of time signals, one may apply spectral methods to characterize the signals, whereas in image processing, texture analysis might be appropriate to calculate a set of relevant features. In any case, these algorithms include a large number of parameters that have to be chosen and that can be tuned to a particular application. This tuning process is often left to the expert in the field and is sometimes done on an intuitive basis coming from past experience and from the particular requirements of the task. We claim that these parameters can be used with great effect to improve the accuracy of downstream classification by directly optimizing feature calculation during the off-line and on-line adaptation of classification systems.

At this point, we would also like to make a clear distinction between feature selection and feature optimization. Feature selection converts a high-dimensional set of features into a smaller set [8] while maintaining most of the relevant information. In the case of a filter approach, various hypotheses are used that lead to optimization criteria based on distance, information, consistency, or dependency [7, 15, 27]. In the case of wrapper approaches [18], the goal is to directly improve classification accuracy. At the heart of the problem is a subset selection task that has a runtime of $O(2^N)$, but good approximations can be obtained using heuristic methods with a runtime of $O(N^2)$. A wide range of algorithms have been developed for this task, such as RELIEF [19, 28], the decision tree method [4], or branch and bound [25]. A recent survey lists 42 different algorithms [14, 24]. Feature selection may thus be considered a projection of the features to a low-dimensional space. This transformation is continuous, and its main property is that objects that were close together in the initial feature space will also be close together in the reduced feature space. If these two objects belong to different classes, then the margin between the two classes will be narrow no matter how the features are selected.

Feature optimization, on the other hand, has access to the original raw data for adapting the features. Even if two objects are identical in the initial feature space, they will not necessarily be so in the space of raw data. If these objects belong to different classes, then feature optimization may be used to create or tune features to

put particular emphasis on this difference in the raw data and thus optimize feature calculation. It thus has the potential of increasing the margin between the two classes beyond what is possible with projections or other continuous transformations of feature space.

13.2 Parameterized Image Features

In the following, we will focus on two different types of features, both of which are widely used in image processing. The first set of features includes shape descriptors that are used to characterize the properties of image regions (so-called "blobs"). "Blobs" usually refer to small-image regions that are darker or brighter than the background and that can be more or less easily detected using (locally adaptive) thresholding methods or edge detectors. In the area of surface inspection, these blobs often correspond to the defects that have to be found and analyzed. For this purpose, the defects are characterized by a set of descriptors such as the total area of the blob, the position of the blob in the image, the ratio of area and circumference of the blob, or the inner structure of the blob.

The second set contains texture features that are used for image segmentation. Texture is a small-scale, visible surface structure that is characterized by local similarity. Small patches of a single texture, so-called texels, thus share a set of properties or features that are similar for all texels coming from this texture. These features may thus be used to characterize the texture and to enable texture segmentation. By using a classifier, one may determine which texels belong to the texture and which do not to establish a boundary between regions of different texture. There is a huge variety of texture features among which Gabor features are a popular choice for segmentation, classification, and image retrieval.

13.2.1 Blob Features

The notion "blob" is a commonly used abbreviation for "binary linked objects". "Linked objects" mean that the objects are usually connected in the sense that there is a path from one pixel of the object to any other pixel of the object that is fully inside the object. Such linked objects can be easily extracted from the image using various (sometimes recursive) algorithms, for example, by following the edge of the object. "Binary" refers to the fact that the algorithms are often applied to bi-valued images, where pixel values of 0 corresponds to background and 1 corresponds to pixels inside the blob. Such binarization is obtained using thresholding algorithms that search for dark or bright areas in the image. This threshold value is also a very important parameter for optimizing feature calculations. This is demonstrated in Fig. 13.1.

If we consider a simple "area" feature that describes the number of pixels covered by the blob, then this feature will substantially depend on the threshold value that

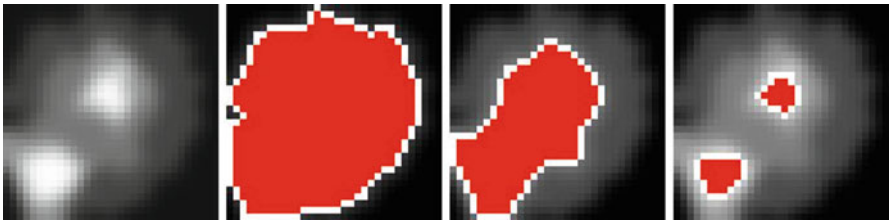

Fig. 13.1 "Area" feature for different threshold values in an 8-bit gray-level image. From *left* to *right*: original image; segmented area using a threshold of 55, 115, and 200. With increasing threshold value, the area is becoming smaller, and may even split into two disconnected regions

Table 13.2 Typical shape features used in blob analysis

Feature	Description	Parameter(s)
Area	Number of pixels inside the blob	Threshold value
Bounding rectangle	Width and height of the bounding rectangle	Threshold value, parameters dealing with outliers, for example, percentage of outlying pixels not considered
Roundness	Ratio of the principal axis to the secondary axis of the circumscribing ellipse	Threshold value, parameters dealing with outliers
Second-order moments	Second-order moments calculated along an axis (column-wise, row-wise, or arbitrary angle)	Threshold value, angle of the axis relative to the principal axis
Circumference	Number of pixels along the edge of the blob	Threshold value
Perimeter	Perimeter of a circle covering the same area as the blob	Threshold value, parameters dealing with outliers
Compactness	Ratio of the total area of the blob and the area of the circular disc that fully covers the blob	Threshold value, parameters dealing with outliers
p-Percentile 1	Gray value for which p percent of the total number of pixels inside the blob are above that value	p
p-Percentile 2	Percentage of pixels inside the blob below/above a certain threshold value	Gray value threshold

is chosen for binarization of the image during feature calculation. The region may even split into two separate blobs for higher threshold values. The correct choice of the threshold value is not immediately clear, and obviously, there is room for optimization.

There is a huge variety of features used in blob analysis, most of which describe the shape of the blob, but there are also features that provide information about the contrast or the inner structure (texture) of the blob. Table 13.2 below lists 9 examples

of such features, including a description of the feature and parameters that can possibly be used for optimization. We have not included formulas or algorithms for the calculation of each feature and refer the reader to the vast amount of literature on basic image processing algorithms, for example, Chap. 10 of [9].

13.2.2 Gabor Features

A wide variety of features has been developed for the characterization of textures. The goal of these features is to capture the local properties of textures and to enable higher-level processing of textures for applications such as image segmentation, content-based image retrieval, or texture defect detection. The main approach in these tasks is to use the features as input to a machine learning structure such as classifiers or regression models. In the following, we describe Gabor features as an example of parameterized texture features and later show how these parameters can be tuned to a specific task.

Gabor features are often used in the above mentioned applications, and their importance also comes from the fact that in some way they resemble processing steps going on in the human visual cortex. However, Gabor features are just one example of a huge set of texture features, and we use them purely for demonstrating the general concept of feature adaptation. For the purpose of texture segmentation, which will be the example that we investigate here, the Gabor features are calculated for every image pixel to obtain a feature map. Thus, each image location $\omega_i \in \Omega$ is characterized by a d-dimensional vector of Gabor features. The actual value of d depends on the number of Gabor filters that form the applied Gabor filter bank and the type of Gabor features.

The calculation of a Gabor feature map for the image requires an intermediate step, in which a set of Gabor filters (a Gabor filter bank) is applied to the image. The feature map is calculated from the individual filter responses. A two-dimensional Gabor filter $g(x,y), (x,y) \in \Omega$ is a sinusoidal plane wave, modulated by a Gaussian envelope given by

$$g_{\lambda,\theta,\gamma,\varphi}(x,y) = \mathrm{e}^{-\frac{x'^2+\gamma^2 y'^2}{\sigma^2}} \cos\left(2\pi j \frac{x'}{\lambda} + \varphi\right),$$

where

$$x' = x\cos\theta + y\sin\theta,\ y' = -x\sin\theta + y\cos\theta \quad \text{and}$$
$$\sigma = c_\sigma \lambda.$$

This filter acts as an oriented local-band-pass filter that is optimal in terms of joint localization and resolution in image and frequency domain [16]. The parameters that can be tuned to obtain a selective Gabor filter are:

λ : Wavelength of the sinusoidal plane wave
θ : Orientation of the Gabor filter

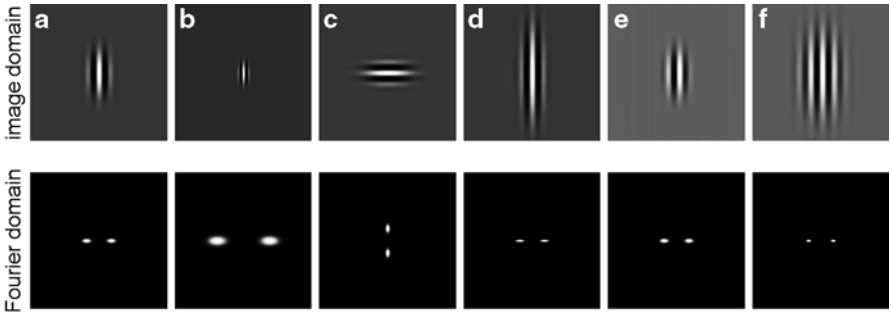

Fig. 13.2 Gabor filters in image and Fourier domain for various filter parameters. Compared to the mother wavelet filter ($\lambda = 1/23, \theta = 0, \gamma = 0.5, \varphi = 0, c_\sigma = 0.56$) in (**a**), the filters (**b**) to (**f**) are different in one parameter setting: (**b**) $\lambda = 1/49$, (**c**) $\theta = \pi/2$, (**d**) $\gamma = 0.25$, (**e**) $\varphi = -\pi/2$, (**f**) $c_\sigma = 1.0$. All filters in image domain are cropped and zoomed by factor 2 to increase visibility

γ : Elongation of the Gaussian envelope
φ : Shift of the sinusoidal plane wave that determines whether the filter is symmetric ($\varphi = 0$) or antisymmetric $\left(\varphi = \frac{\pi}{2}\right)$
c_σ : Factor that determines the size of the Gaussian envelope

A plot of typical Gabor filters for different parameters is shown in Fig. 13.2. In Sect. 13.5.3.2, we will demonstrate the importance of tuning these parameters to obtain highly accurate segmentation results.

By calculating the convolution of a Gabor filter g with an image I as in

$$r(x,y) = \iint_{\Omega} I(x,y)g(x-\xi, y-\eta)\mathrm{d}\xi\,\mathrm{d}\eta,$$

a new image (the filter response r) is obtained as shown in Fig. 13.3.

Usually, one does not only apply a single Gabor filter, but a family of filters—often called a filter bank—generated by varying the wavelength and orientation of a mother wavelet filter. A typical value is to use six orientations and four different scales, resulting in 24 different Gabor filters and thus in 24 filtered images. Spatial frequencies (scale) are chosen to cover the relevant frequencies in the image.

In many applications, these images are then converted into so-called Gabor energy maps. The energy map $e(x,y)$ of a Gabor filter [13] is calculated by considering filter responses to symmetric and antisymmetric Gabor filters:

$$e_{\lambda,\theta,\gamma}(x,y) = \sqrt{r^2_{\lambda,\theta,\gamma,0}(x,y) + r^2_{\lambda,\theta,\gamma,-\pi/2}(x,y)}.$$

The Gabor energy e is related to the local power spectrum p as follows:

$$p_{\lambda,\theta,\gamma}(x,y) = \mathrm{e}^2_{\lambda,\theta,\gamma}(x,y),$$

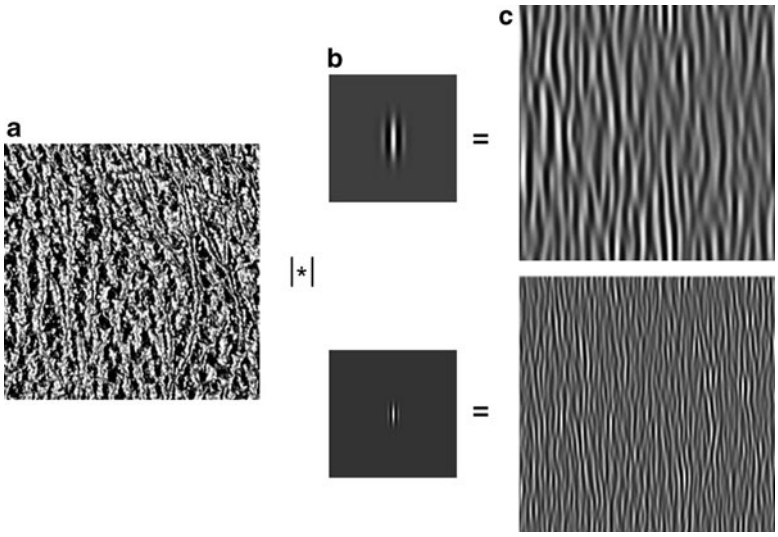

Fig. 13.3 The Gabor filter responses r for an image I from the Brodatz texture album [3]. The operator $|*|$ represents a convolution, which is typically done in Fourier domain to reduce the computational load for larger images

as the Gabor filters are essentially a Fourier transformation with a Gaussian windowing function. The resulting image contains an energy value per pixel and describes the local energy contained in the different spatial frequencies and directions. In order to finally obtain a single feature value, some kind of aggregated information is calculated for the image. This can be done in several different ways, some of which are described in the following:

The *sum of Gabor orientation energy difference* (SGOED) [17] is obtained by summing over all pixels of the energy map:

$$E_{\lambda,\theta,\gamma} = \iint_{\Omega} p_{\lambda,\theta,\gamma}(x,y)\mathrm{d}x\mathrm{d}y.$$

Assuming that we operate with a filter bank with Θ different orientations and Λ different scales, we can further sum up the entries of all energy maps $E_{*,*,\gamma}$ along scales or orientations such that we obtain two vectors

$$E_{\Theta} = \sum_{i=1}^{\Lambda} E_{i,\theta,\gamma}$$

and

$$E_{\Lambda} = \sum_{i=1}^{\Theta} E_{\lambda,i,\gamma},$$

with Θ and Λ elements, respectively. The SGOED [17] is then found by

$$f_{\mathrm{SGOED}} = \sum_{i=1}^{\Theta} \left| E_{\Lambda}(\theta_i) - E_{\Lambda}(\theta_{i+1}) \right|,$$

where the index θ_i is to be understood as being calculated $\theta_i = \mod(i, \Theta)$. SGOED generates large values for abrupt orientation changes between energy values. The features thus measure in some sense whether the texture is isotropic (low value for SGOED) or whether it has a strong directional structure (high values for SGOED) in a certain spatial frequency range. In a similar fashion, one can derive the sum of Gabor scale energy differences (SGSED) by

$$f_{\text{SGSED}} = \sum_{i=1}^{\Lambda} |E_{\Theta}(\lambda_i) - E_{\Theta}(\lambda_{i+1})|,$$

where the index λ_i is to be understood as being calculated $\lambda_i = \mod(i, \Lambda)$. The SGSED is high if the image under investigation shows a dominant texture scale, while it is low if multiple texture scales are present.

While the aggregation of Gabor energy map entries as described so far has been applied successfully, for example, to distinguish natural from man-made objects in [17] or to texture classification and content-based retrieval [31], their basic idea can hardly be transferred to texture segmentation as these features aim to capture properties of larger image areas.

For pixel-wise segmentation, the energy map entries $e_{\lambda,\theta,\gamma}(x,y)$ for a fixed image location ω_i but different filter parameters (i.e., the energy maps obtained from a Gabor filter bank) are directly used to build a Gabor energy feature vector f_{ω_i}. Grigorescu et al. [13] evaluated the segmentation accuracy for more complex Gabor filter-based features, inspired by the early processing in human vision system and could show that Gabor energy features outperform complex moment features [13]. Superior performance was reported for grating cell operator features that are computed in a two-stage process. The first stage, based on a Gabor filter bank as described above, detects the presence of three parallel bars at any image location. The second stage integrates the output of the first within a certain surrounding and therefore detects the presence of multiple combinations of parallel bars.

13.3 Feature Adaptation Concepts (Off-line)

In the following, we first outline existing approaches to feature adaptation for blob and Gabor features. Next, we describe the general concept of feature optimization for off-line processing which will also be the basis for the on-line adaptation. To motivate the necessity for feature adaptation, let us consider a simplified inspection task, where we have to detect elongated scratches of predefined orientation and size. These scratches are present on a milled surface that shows an oriented surface texture itself. Given an image of a part with a scratch orientation of 10° and a surface texture orientation of 35°, the energy response for two Gabor filters with $\theta = 10°$ and $\theta = 35°$ with an appropriate scale λ would be highest. For an image without scratch, only the Gabor filter with $\theta = 35°$ would yield a high energy response. Obviously, the energy for the Gabor filter with $\theta = 10°$ can be used to distinguish defective and defect-free surface patches. However, if we would rely on a fixed

Gabor filter set with $\theta = \{0°, 45°, 90°, 135°\}$, the distinction between defective and defect-free areas cannot be based on a single filters' energy response any more. In real-world inspection tasks, we can never be sure that the properties of a (defect free) surface or the scratches remain constant over time (e.g., wear of the machine tools, material differences, temperature difference, etc.). Thus, it is crucial to adapt the feature parameters to discriminate between defective and defect-free surface areas.

13.3.1 Blob Feature Adaptation

For blob features, the main approach to feature adaptation is a manual selection of the relevant parameters by an expert. Automatic adaptation is rarely implemented, because of the perceived simplicity of the features. In some applications, however, such as tracking of objects, the features will change, for example, because the object that needs to be tracked is seen from a different viewpoint. Consequently, the relevance of features may change depending on the angle of view, because certain parts of the object will not be visible any more. Some recent results that cover this topic can be found in [5] and [6]. An adaptive feature transformation is described in [1], where the feature vector is postprocessed by an adaptive transformation matrix that is used to make the features invariant to environmental changes. The joint optimization of classifiers and features is investigated in [20], but also in this case, the optimization of features is done by a selection process rather than by adapting parameters inside the feature calculation.

13.3.2 Gabor Feature Adaptation

The filter bank which is required to derive Gabor energy map features obviously offers a set of parameters that can be used for optimizing the subsequent processing steps. These parameters include the scale λ, the orientation θ, as well as the number of scales Λ and directions Θ used. Gabor filter banks are usually designed to optimally represent the texture signal of the image. Optimality is measured by the mean squared error between the reconstructed and the original signal [21]. The goal of texture segmentation, however, is not to optimally represent the single textures but to divide the image into regions that contain the different textures. The focus of the optimization should thus be on identifying those parameters for the Gabor filters that allow an optimal discrimination between the different textures.

The second aspect that needs to be considered is computing time. Calculating the convolution of a Gabor filter with a whole image is computationally costly, and one is thus interested in minimizing the number of filters used and on choosing those with the highest discriminative power.

For the optimization, we may either focus on a single filter or the whole filter bank. Both approaches are slightly different.

In the filter bank design, one generally uses a set of directions that cover 180° (360° is not required, because the filter is symmetric), and the only question that needs to be answered is how many directions we want to consider. This decision is made purely on computational cost, and typical values are in the range of four (0°, 45°, 90°, 135°) to eight directions. The scale parameter of the Gaussian function is generally selected intuitively and assumed to be a constant. In [11], guidelines for selecting values for the scale parameter are proposed, but quite often, human intervention is still required to assist in selecting the appropriate filter parameters for texture segmentation. The selection of filters is then done by visually inspecting the filtered images, to identify those filters that best characterize different textures in the image.

For filter design, the focus is on selecting one or very few filters with parameters that are optimal for the task. This avoids some of the problems of filter banks, especially the high computational effort that is required for calculating the convolution with a larger number of filters. In many cases, the choice of parameters for the single filters is made based on a priori knowledge about the texture. Such knowledge may include typical dominating directions in the textures or the visual coarseness of the textures. This may be put on a more objective basis by performing a Fourier analysis on the whole image to identify the most significant (or discriminative) spectral components to deduce the scale and orientation for the Gabor filter. An explicit method for the selection of scale and direction parameters is proposed in [32]. The main idea is to solve a max-min optimization problem that maximizes the minimal ratio of Gabor energies contained in the different textures. The idea has a strong similarity to Fisher discriminant analysis [10] (Sect. 3.8.2) and to the optimization criteria used in support vector machines. It basically tries to maximize the margin between the two Gabor energy representations of the textures. A more recent approach presented in [29] aims at maximizing the fraction of separable harmonic signal pairs in a given frequency range. Separable here means that the filter responses are disjoint in at least one component of the response vector. This method proved to be more efficient than previous methods while achieving the same accuracy on texture segmentation tasks. For solving the resulting optimization problem, a range of methods has been applied such as genetic algorithms [22], simulated annealing [10] and any other kind of optimization method that can deal with the discrete nature of the optimization problem.

13.3.3 General Feature Adaptation Concept

Optimization of features as proposed in this context is an optimization step that is done once the classification system is already in place. The assumption is that a set of features has been selected and that a classifier has been trained off-line to achieve reasonable accuracy. At this stage, it makes sense to reconsider and optimize the feature calculation in order to make the classification system more robust and to further increase its accuracy. Optimizing features “from scratch” is not likely

Table 13.3 Different target functions J for commonly used classifiers

Classifier	Within-class scatter s_W	Between-class scatter s_B	J
Nearest neighbor	$\frac{1}{n_0}\sum_{j=1\ldots n_0}\left\|\mathbf{F}(x_{0,j},\mathbf{p})-\mu_0\right\|^2$ $+\frac{1}{n_1}\sum_{j=1\ldots n_1}\left\|\mathbf{F}(x_{1,j},\mathbf{p})-\mu_1\right\|^2$	$\left\|\mu_0-\mu_1\right\|^2$	$\frac{s_W(\mathbf{p})}{s_B(\mathbf{p})}$
Linear classifier	$\frac{1}{n_0}\sum_{j=1\ldots n_0}\left(\mathbf{c}^T(\mathbf{F}(x_{0,j},\mathbf{p})-\mu_0)\right)^2$ $+\frac{1}{n_1}\sum_{j=1\ldots n_1}\left(\mathbf{c}^T(\mathbf{F}(x_{1,j},\mathbf{p})-\mu_1)\right)^2$	$\left(\mathbf{c}^T(\mu_0-\mu_1)\right)^2$	$\frac{s_W(\mathbf{c},\mathbf{p})}{s_B(\mathbf{c},\mathbf{p})}$
Decision tree	$\frac{1}{n_0}\sum_{j=1\ldots n_0}\left(\mathbf{e}_i^T(\mathbf{F}(x_{0,j},\mathbf{p})-\mu_0)\right)^2$ $+\frac{1}{n_1}\sum_{j=1\ldots n_1}\left(\mathbf{e}_i^T(\mathbf{F}(x_{1,j},\mathbf{p})-\mu_1)\right)^2$	$\left(\mathbf{e}_i^T(\mu_0-\mu_1)\right)^2$	$\sum_{i=1\ldots m}\frac{s_W(\mathbf{e}_i,\mathbf{p})}{s_B(\mathbf{e}_i,\mathbf{p})}$

to succeed, because the search space is huge and features are in the danger of degenerating into measurements for properties that are totally different from the originally intended meaning of the feature. This would impair the interpretability of the classification system and should thus be avoided.

For the adaptation of a set of features $F_i, i = 1,\ldots,m$, which we collect in a feature vector $\mathbf{F} \in \mathbb{R}^m$ and which depend on a parameter vector $\mathbf{p}$, we try to minimize the within-class scatter s_W in relation to the between-class scatter s_B. Feature adaptation should lead to a decision boundary that can be easily reproduced by the classifier at hand and that maximizes the distance between the classes perpendicular to the decision boundary. The way how we quantify the scatter thus depends on the classifier that is used. We will discuss this for three commonly used types of classifiers: nearest neighbor classifiers, linear classifiers, and decision trees.

Nearest neighbor classifiers use a distance measure $\|.\|$ to determine the distance of a feature vector to the class center. Classification performance will improve if the feature parameters are adapted to maximize the distance between the classes with respect to this distance measure. Linear classifiers create a separating hyperplane in feature space. This hyperplane is usually characterized by its normal vector $\mathbf{c}^T$. Feature parameters should thus be chosen to optimize the distance perpendicular to this hyperplane. Adaptation requires a joint optimization of feature parameters and classifier. Finally, for decision trees, the decision boundaries are in general parallel to the feature space axes $\mathbf{e}_i^T$. Thus, the scatter perpendicular to the feature space axes needs to be considered. The different methods of calculating the within- and between-class scatter and the target function J are shown in Table 13.3, where μ_0 and μ_1 define the average feature vectors over the class 0 (with n_0 representatives) and class 1 (with n_1 representatives), respectively. The target function aims at maximizing the distance between the classes, while minimizing the scatter within each class. The distance measure is selected in such a way that it favors the separation perpendicular to boundaries that can be easily modeled by the different classifiers.

Clearly, it will be beneficial for the simplicity of the optimization if the features can be optimized independent of the current parametrization of the classifier, but this will not always be the case. We can thus think of three different strategies:

- For decision-tree classifiers and also for the nearest neighbor classifiers using the L2-norm ($\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T\mathbf{x}}$) as a distance measure, one can optimize the features independent of the current classifier. The optimization process thus consists of calculating the target function and applying a numerical optimization method.
- For linear classifiers, the target function depends on the particular parametrization of the classifiers, as the direction of the decision boundary may change. A similar situation is found if the Mahalanobis distance is used instead of the L2 norm for the nearest neighbor classifiers. The Mahalanobis distance accounts for the variance differences of the individual features and scales the features accordingly:

$$\|\mathbf{x}\|_{\text{Mahalanobis}} = \sqrt{(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)},$$

  where Σ is the covariance matrix of $\mathbf{x}$.

  In this case, the distance measure (and thus the target function) will change depending on the distribution of the features. Clearly, this distribution will change whenever the feature calculation is adapted. These changes have to be included in the target function during the optimization. Alternatively, one may think of optimizing the features for a fixed classifier (i.e., fixed direction of the decision boundary or fixed distance measure) coming from the initial off-line training step. This clearly simplifies the optimization process, but probably gives away some potential for improvement.
- For some types of classifiers, such as neural networks, the decision boundary is so complex that the basic concept of optimizing the scatter perpendicular to the decision boundary cannot be directly applied. In this case, we propose to implement a strategy similar to the "wrapper" approach that directly optimizes classification accuracy. This joint feature and classifier optimization will lead to a quite complex optimization problem, which is limiting its applicability.

Independent of the particular choice of strategy the resulting optimization problems are usually nonsmooth and—considering the discrete nature of the threshold—even discontinuous. If there are only few parameters to tune, an exhaustive search is possible. For more complicated data sets, we found that gradient descent methods with a simple numerical estimation of the gradient work well. As usual, only a local minimum can be guaranteed in this case. More details of the algorithms are presented in [12].

13.4 Feature Adaptation Concept (On-line)

The basic assumption for the on-line adaptation of features is that a reasonable initial set of parameters has been found and that also an initial set of training data is available from which we can estimate certain feature statistics. The general approach will then be a gradient-based optimization that uses a numeric estimate of

the gradient. This is due to the discrete nature of many parameters, especially of the threshold values that are usually chosen from a discrete set of $0, 1, \ldots, 255$ for 8-bit gray-level images.

Referring to Table 13.3, we find for the linear classifier that the gradient of the target function J with respect to the classifier's parameter $\mathbf{c}$ and the feature parameters $\mathbf{p}$ can be calculated analytically. For the derivation, we reformulate s_W and s_B of $J(\mathbf{c},\mathbf{p}) = \frac{s_\mathrm{W}(\mathbf{c},\mathbf{p})}{s_\mathrm{B}(\mathbf{c},\mathbf{p})}$, compute the partial derivatives $\frac{\partial s_\mathrm{W}(\mathbf{c},\mathbf{p})}{\partial \mathbf{c}}$, $\frac{\partial s_\mathrm{W}(\mathbf{c},\mathbf{p})}{\partial \mathbf{p}}$, $\frac{\partial s_\mathrm{B}(\mathbf{c},\mathbf{p})}{\partial \mathbf{c}}$, $\frac{\partial s_\mathrm{B}(\mathbf{c},\mathbf{p})}{\partial \mathbf{p}}$, and combine them using the quotient rule

$$\frac{\partial J(\mathbf{c},\mathbf{p})}{\partial \cdot} = \frac{\frac{\partial s_\mathrm{W}(\mathbf{c},\mathbf{p})}{\partial \cdot} s_\mathrm{B}(\mathbf{c},\mathbf{p}) - s_\mathrm{W}(\mathbf{c},\mathbf{p}) \frac{\partial s_\mathrm{B}(\mathbf{c},\mathbf{p})}{\partial \cdot}}{s_\mathrm{B}(\mathbf{c},\mathbf{p})^2}. \tag{13.1}$$

By introducing the per class feature covariance matrices

$$\Sigma_k(\mathbf{p}) = \frac{1}{n_k} \Big(\mathbf{F}(x_k,\mathbf{p}) - \mu_k \Big) \Big(\mathbf{F}(x_k,\mathbf{p}) - \mu_k \Big)^T, \tag{13.2}$$

we can rewrite the scatter terms into the quadratic forms

$$s_\mathrm{W}(\mathbf{c},\mathbf{p}) = \mathbf{c}^T \Big(\Sigma_0(\mathbf{p}) + \Sigma_1(\mathbf{p}) \Big) \mathbf{c} \tag{13.3}$$

and

$$s_\mathrm{B}(\mathbf{c},\mathbf{p}) = \mathbf{c}^T (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T \mathbf{c}. \tag{13.4}$$

The vector-valued derivatives are then

$$\frac{\partial s_\mathrm{W}(\mathbf{c},\mathbf{p})}{\partial \mathbf{c}} = 2 \Big(\Sigma_0(\mathbf{p}) + \Sigma_1(\mathbf{p}) \Big) \mathbf{c} \tag{13.5}$$

$$\frac{\partial s_\mathrm{B}(\mathbf{c},\mathbf{p})}{\partial \mathbf{c}} = 2 (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T \mathbf{c} \tag{13.6}$$

$$\frac{\partial s_\mathrm{W}(\mathbf{c},\mathbf{p})}{\partial \mathbf{p}} = \mathbf{c} \left(\frac{\partial \Sigma_0(\mathbf{p})}{\partial \mathbf{p}} - \frac{\partial \Sigma_0(\mathbf{p})}{\partial \mathbf{p}} \right) \mathbf{c}^T \tag{13.7}$$

$$\frac{\partial s_\mathrm{B}(\mathbf{c},\mathbf{p})}{\partial \mathbf{p}} = 2 \mathbf{c} (\mu_0 - \mu_1) \left(\frac{\partial \mu_0}{\partial \mathbf{p}} - \frac{\partial \mu_1}{\partial \mathbf{p}} \right) \mathbf{c}^T. \tag{13.8}$$

Regarding the on-line adaptation, the key issue is to avoid the repeated processing of a large number of test images. We thus have to make some compromises with respect to the quality of the estimates of the second-order statistics and the gradients.

For the second-order statistics, we create two buffers in which we store samples. These buffers are of fixed length with a typical size of 25–100 samples. Each buffer corresponds to one of the two classes (accept/reject). Depending on the label

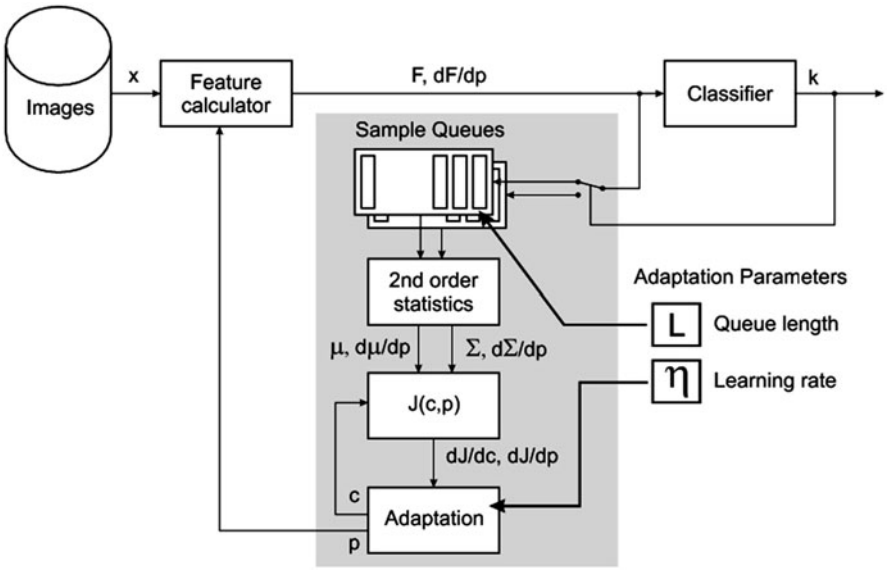

Fig. 13.4 Overview of the concept for on-line feature optimization

obtained from the classifier or from an expert, the feature vectors **F** from the images as well as their gradients with respect to the feature parameters **p** are routed to one of the two buffers. The buffers can also be thought of as a sliding window over the sequence of sample images. Based on the buffered samples, second-order statistics (the average μ and the variances Σ) of the features and the feature gradients are computed along with their derivatives with respect to their parameters. From the statistics and the classifier's parameters **c**, the target function J and its gradients with respect to **c** and **p** can be computed. Incremental optimization of the quality criterion is then performed by updating the **p** and **c** in the direction of the negative gradient of J. The step length is controlled by a fixed learning rate η:

$$\mathbf{p}_{n+1} = \mathbf{p}_n - \eta \nabla_p J_n$$
$$\mathbf{c}_{n+1} = \mathbf{c}_n - \eta \nabla_c J_n.$$

An overview of the whole concept is shown in Fig. 13.4.

A basic evaluation of this method on a simple test case showed that learning rates of 0.001–0.01 lead to good results and that a buffer size of 25–100 is required. After about 500 to 1,000 iterations (samples), the parameters were reasonably close to their optimal values. It should be noted, however, that depending on the length of the queue, there will also be fluctuations in the parameters.

13.5 Experimental Evaluation

In order to assess the influence of feature optimization on the final classification accuracy, we perform experiments with artificial and real-world test data. Starting from a standard off-line classification approach, we investigate how much can be gained by including a separate optimization step for the features.

13.5.1 Test Data

13.5.1.1 Test Data for Blob Analysis

For optimizing the blob-analysis features, an artificial data set of images was created. The images are preprocessed so that they only show the potential defects (objects). These images are called "contrast images," whose (gray scale) pixel values depend on the degree of deviation from the "normal" appearance of the part (white denoting complete similarity, black denoting complete dissimilarity). This may be considered an abstraction of a surface inspection task in machine vision. By using contrast images, we remove the application-specific low-level image processing from our consideration and focus purely on the classification problem. Figure 13.5 shows an example that represents a deviation image from a printing process: blobs that correspond to potential defects are highlighted in different gray levels.

We used five sets of artificial test data, each with 20,000 images, which were labeled automatically either as good (accept) or as bad (reject) with about 10,000 images in each class. In order to generate the labels, a set of rules was used for

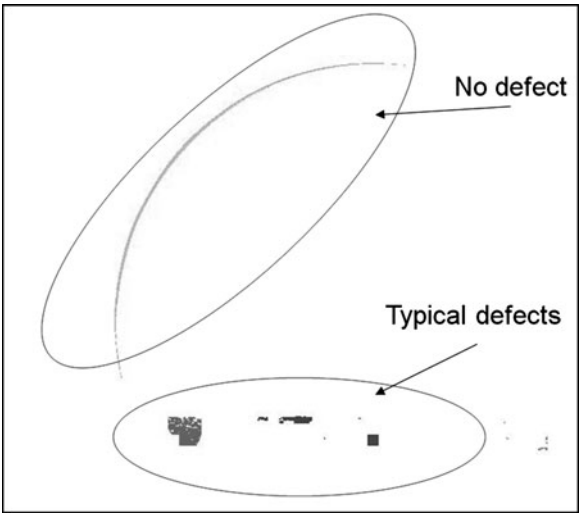

Fig. 13.5 For testing, we use contrast images. Only deviations from the normal appearance are shown in the image, the background is removed

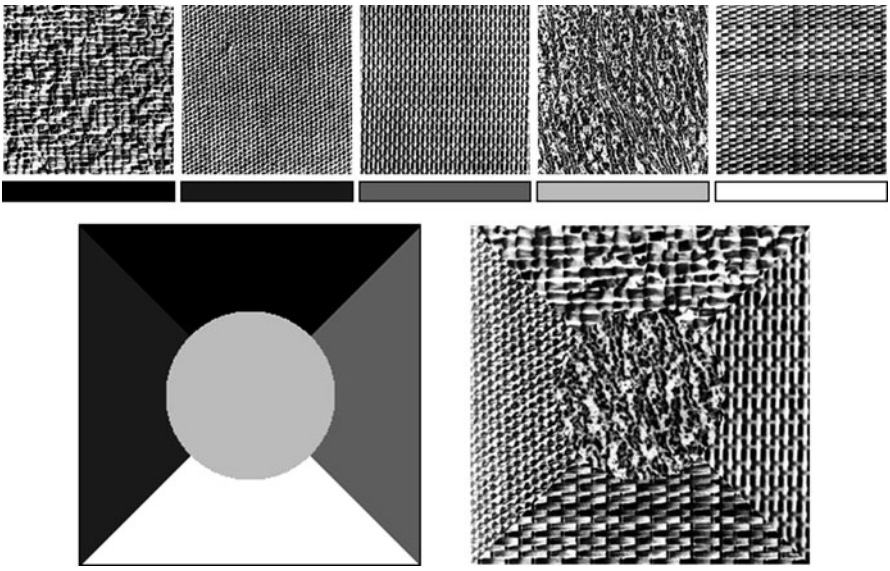

Fig. 13.6 Texture mosaic m_1 (*right*) built from five source textures (*top row*) according to the ground truth map (*left*)

each set of test images. The rules were based on descriptions that are regularly found in quality control instructions, such as "part is bad, if there is a fault with size $>$1.5 mm." The rules also included more complicated combinations, such as "part is bad, if there is a cluster of 4 faults, each with a size $>$0.75 mm." Three to five such rules were logically combined for each set of images. The images and the rules were chosen to have some resemblance to inspection of machined parts.

13.5.1.2 Test Data for Texture Segmentation

In order to assess the discriminative power of texture features, we perform a texture segmentation experiment on so-called texture mosaics. A texture mosaic $m(x,y)$ is built from n source textures $s_i(x,y), i = 1,\ldots,n$. For each pixel of $m(x,y)$, the ground truth map $t(x,y)$ defines its source texture index s_i. Figure 13.6 shows the source textures, ground truth map, and the resulting mosaic for $n = 5$.

In order to investigate the adaptation properties of Gabor-feature-based segmentation, we utilize mosaics combined from Brodatz [3], VisTex [23], and MeasTex [30] source textures to ensure diverse texture properties. Diversity of the source textures is crucial for our experiments as it calls for Gabor filter (banks) that adapt to the specific properties of each source texture. Table 13.4 contains detailed information about the selected mosaics. Figure 13.7 shows all texture mosaics and their ground truth information, respectively.

Table 13.4 All evaluated texture mosaics. The mosaics m_1 to m_5 are composed from five source textures each and are available on-line [26]. Mosaics m_6 and m_7 are composed from two source textures in order to obtain reasonable test data for single Gabor filter adaptation. All texture mosaics are shown in Fig. 13.7

ID	Size	n	Source of textures
m_1	256×256	5	Brodatz
m_2	256×256	5	VisTex
m_3	256×256	5	VisTex
m_4	256×256	5	VisTex
m_5	256×256	5	MeasTex
m_6	256×256	2	Brodatz, Vistex
m_7	256×256	2	Brodatz, Vistex

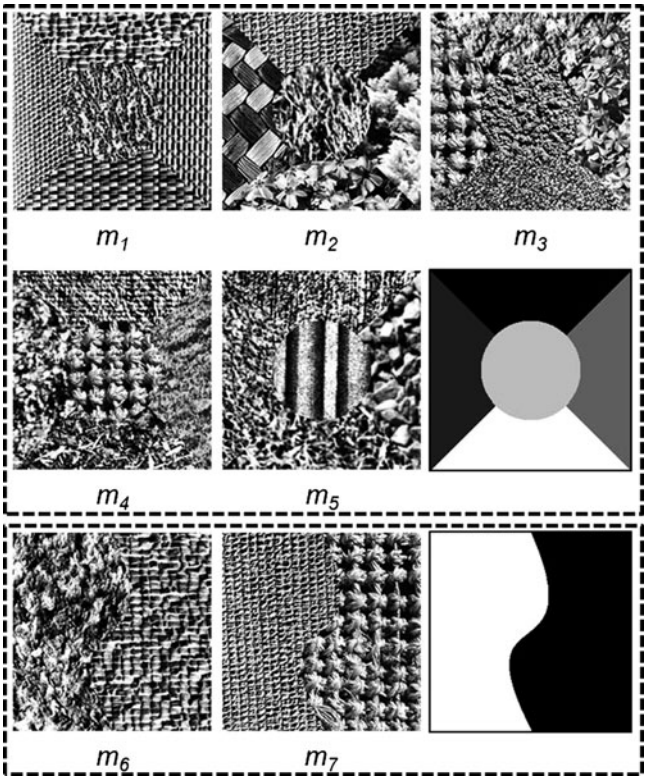

Fig. 13.7 Texture mosaics m_1 to m_7 and the corresponding ground truth maps

13.5.2 Classification with Adaptive Features

This section describes how the classification experiment based on adaptive features was conducted. The quantitative results of these experiments are discussed in Sect. 13.5.3.

13.5.2.1 Blob Analysis Features

For those features that allow an optimization, one parameter was chosen, which in most cases was a gray-level threshold value. If the pixel was above the threshold, it was considered as belonging to the fault and otherwise as belonging to the background. A reasonable range for the threshold was between 0 and 255 (as we deal with 8-bit gray-level images, these are the minimum and maximum values of the pixels). For the sake of optimization, this range was scaled to an interval of $[0,1]$. The goal was to adapt these thresholds in such a way that the classification accuracy is improved and that the decision boundary can be more easily reproduced by the classifier.

Regarding the computational complexity, it should be noted that the computational effort of optimizing the feature parameters can be significant. This is caused by the fact that for each iteration, the features of all objects in all images need to be calculated with the current parameter settings. Depending on the size and number of the images, this optimization may take several hours. On the other hand, once the optimal parameters are found, the feature calculation and classification takes the same amount of time as for any other method. With respect to the application of surface inspection, this is important because the "on-line" processing of the images, which usually has tight constraints on computing time, is not affected. It is just the "off-line" training process that becomes quite time-consuming.

13.5.2.2 Gabor Features

We choose a fixed feature extraction method, classification, and postprocessing setup in order to investigate the influence of feature adaptation on the segmentation accuracy. These subsequent processing steps are briefly outlined below as it is not the aim of our evaluation to optimize classification or postprocessing. A detailed discussion of Gabor filter (bank) optimization can be found in Sect. 13.5.3.2.

For *feature extraction*, we choose Gabor energy features to compute a feature map as described in Sect. 13.2.2 as their computational costs are reasonably low to perform multiple segmentation runs. Thus, we obtain a feature vector $\mathbf{F}$ of dimensionality d for each texture mosaic location $\omega_i = (x,y)$, where d is the number of Gabor filters in the filter bank. The training feature matrix is created by randomly selecting 200 feature vectors per source texture.

Table 13.5 Change of classification accuracy achieved by optimizing the feature calculation for different classifiers

Classifier	Initial (%)	Optimized (%)
C4.5	74.2	88.08
CART	73.8	87.77
Cluster-based	74.0	77.9
kNN	68.7	87.8

For the *pixel classification*, we choose two methods. We compute the mean Fisher criterion as described above to obtain a score that is independent from the classifier and describes the separation between the different texture classes for the current feature parameters. In order to obtain mosaic class-label results, we train a Random forest (RF) [2] with 300 trees on the source textures in order to classify each pixel of the texture mosaic. We rely on the R-package of the Random forrest available at http://cran.r-project.org/web/packages/randomForest/.

It is well known [13] that *filtering* of the class label improves the segmentation accuracy which is measured as ratio of misclassified pixels divided by the total number of mosaic pixels.

To remove isolated pixels and to smooth the boundaries between the segmented regions, we apply a filtering approach [13] of the classification result. Therefore, we assign that class label to a pixel that occurs most frequently in its 24×24 neighborhood.

For all experiments, we postprocess our class labels by applying a filter that replaces each label with the mode of its surrounding (size 24×24).

13.5.3 Results

The results are discussed separately for blob and Gabor features as these tasks differ in terms of classification and postprocessing of the results. Despite these differences, we show that both machine vision tasks benefit from feature adaptation.

13.5.3.1 Blob Analysis Features

We have chosen four different classifiers that performed well on surface inspection tasks and used target functions of Table 13.3 for optimization. Even though this required the repeated processing of 10,000 images an exhaustive search was performed in order to be sure that the global optimum is found. The improvement in classification accuracy that could be achieved is shown in Table 13.5.

For some classifiers, the change is quite substantial, and we find improvements of more than 10%. It should be noted however that the gain that can be achieved highly depends on the initial parameter settings. If these parameters are preset by a machine vision expert, then the improvement may be minimal, if any. Feature

Table 13.6 Parameter ranges that constitute the parameter grid for single Gabor filter optimization. The reciprocal of the filter wavelength $(1/\lambda)$ denotes the number of cycles of the plane wave for the whole image

Parameter	Range
Wavelength: λ	$[1/11, 1/47]$
Orientation: θ	$[0, \pi]$
Elongation: γ	$[0.3, 0.7]$
Size of Gaussian: c_σ	$[0.3, 0.7]$

Table 13.7 This table shows the filter parameters that resulted in the highest classification accuracy per mosaic. It lists the average Fisher criterion, the accuracy for classification with density estimation, and the corresponding filter parameters for single Gabor filter optimization

Mosaic ID	Fisher criterion	Accuracy (%)	Parameters
m_1	0.946	61.5	$\lambda = 0.026, \theta = 0.000, \gamma = 0.5, c_\sigma = 0.7$
m_2	0.858	53.7	$\lambda = 0.029, \theta = 1.396, \gamma = 0.3, c_\sigma = 0.7$
m_3	0.959	49.9	$\lambda = 0.091, \theta = 0.000, \gamma = 0.7, c_\sigma = 0.5$
m_4	0.788	46.2	$\lambda = 0.091, \theta = 0.000, \gamma = 0.7, c_\sigma = 0.5$
m_5	0.691	40.2	$\lambda = 0.091, \theta = 1.047, \gamma = 0.7, c_\sigma = 0.5$
m_6	0.950	92.5	$\lambda = 0.032, \theta = 1.396, \gamma = 0.7, c_\sigma = 0.7$
m_7	1.968	98.2	$\lambda = 0.091, \theta = 0.000, \gamma = 0.7, c_\sigma = 0.7$

adaptation, however, may help for those features that have multiple parameters. The results given as "initial" in Table 13.5 were obtained for parameter values chosen as a first guess by an expert without further manual optimization.

13.5.3.2 Gabor Features

Gabor filters offer many tuning knobs that are likely to influence the discriminative power of any Gabor feature. These parameters (λ, θ, γ, c_σ) were already discussed in Sect. 13.2.2. The parameter φ, which determines the shift of the sinusoidal plane wave will not be tuned as we require both the symmetric and the antisymmetric filter response to compute Gabor energy features. Table 13.6 contains the parameters' ranges that form our search grid for single Gabor filter optimization.

As we are dealing with a one-dimensional feature space in single filter optimization, we do not apply a complex RF but a classifier that fits a Gaussian density to each texture class and assigns a mosaic pixel to the class with highest probability for the pixels' feature value. The results for single filter optimization are listed in Table 13.7 and visualized in Fig. 13.8. The classification accuracy is substantially above the baseline accuracy (i.e., 20% for m_1 to m_5 and 50% for m_6, m_7) for all mosaics. We can also observe that the Fisher criterion scores do not necessarily correlate with the classification accuracy because of the postprocessing

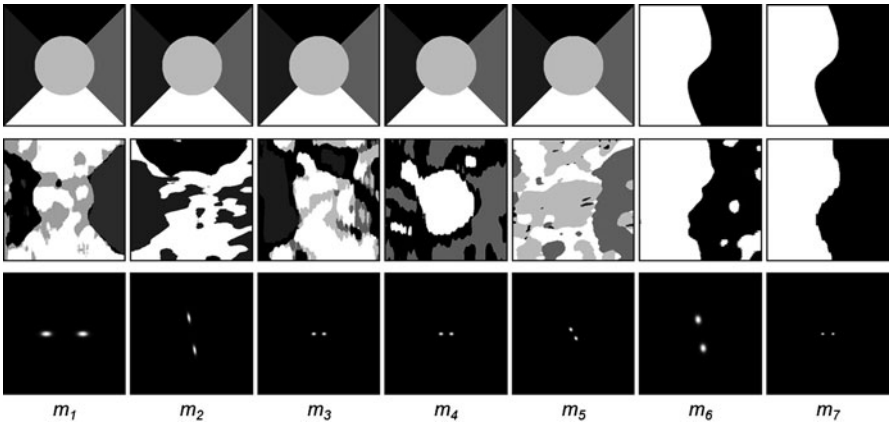

Fig. 13.8 Results for single Gabor filter optimization. The *top row* shows the ground truth map, and the *middle row* contains the best segmentation result obtained for the Gabor filter show in the *bottom row* (Fourier domain)

Table 13.8 Parameter ranges that constitute the parameter grid for filter bank optimization

Parameter	Range	Comments
Wavelength: λ_l	$[1/11, 1/47]$	Wavelength of the filter with lowest frequency
Orientation offset: o_θ	$[0, \pi/\Theta]$	o_θ is added to the equidistant orientation values
Elongation: γ	0.5	Not tuned to reduce the computational costs
Size of Gaussian: c_σ	$[0.3, 0.7]$	
No. scales: Λ	$[1, 4]$	For example $\Lambda = 3$, $\lambda_l = 11$ then $\lambda \in \{1/11, 1/19, 1/35\}$
No. orientations: Θ	$[2, 8]$	Orientation of filter j: $\theta_j = (j-1) * (\pi/\Theta)$

of the segmentation result. In general, we can observe that a single filter is not sufficient to distinguish between five texture classes while a two-class segmentation problem can be solved with reasonable accuracy.

However, a single Gabor filter is likely to be insufficient for real-world segmentation tasks as the classification decision is obtained on the basis of a one-dimensional feature space. Thus, Gabor filter banks that contain a family of self-similar Gabor filters are typically applied for classification, segmentation, and retrieval systems. The optimization of a whole Gabor filter bank introduces several additional parameters such as the number of filter orientations Θ and scales Λ. In order to reduce the computation cost for optimization, it is worthwhile to consider interdependencies between different parameters. For instance, the filter orientation which can be chosen in the range $[0, \pi]$ for a single feature can be treated as orientation offset o_θ in case of a filter bank that alters the filters orientation that is typically chosen equidistant to cover the range of $[0, \pi]$. The tuned parameters are listed in Table 13.8.

The results for the Gabor filter bank optimization are listed in Table 13.9 and visualized in Fig. 13.9. We compare our results to the baseline, obtained for the Gabor filter bank proposed in [13] that is based on experimental findings on early

Table 13.9 This table shows the filter bank parameters that resulted in the highest RF classification accuracy per mosaic. It lists the average Fisher criterion, the baseline accuracy obtained for the filter bank proposed in [13], the RF accuracy, and the corresponding Gabor filter bank parameters

Mosaic ID	Fisher criterion	Baseline accuracy (%)	RF accuracy (%)	Parameters
m_1	2.520	95.3	97.0	$\lambda_l = 0.029, o_\theta = 0.196, c_\sigma = 0.5, \Lambda = 3, \Theta = 8$
m_2	0.524	70.8	77.0	$\lambda_l = 0.037, o_\theta = 0.000, c_\sigma = 0.3, \Lambda = 3, \Theta = 2$
m_3	1.178	69.4	75.6	$\lambda_l = 0.053, o_\theta = 0.000, c_\sigma = 0.5, \Lambda = 3, \Theta = 6$
m_4	1.129	65.8	72.8	$\lambda_l = 0.053, o_\theta = 0.000, c_\sigma = 0.3, \Lambda = 3, \Theta = 6$
m_5	1.979	77.1	82.0	$\lambda_l = 0.029, o_\theta = 0.000, c_\sigma = 0.7, \Lambda = 3, \Theta = 8$
m_6	1.390	98.9	99.1	$\lambda_l = 0.029, o_\theta = 0.000, c_\sigma = 0.5, \Lambda = 3, \Theta = 4$
m_7	2.251	98.9	99.5	$\lambda_l = 0.037, o_\theta = 0.000, c_\sigma = 0.5, \Lambda = 3, \Theta = 8$

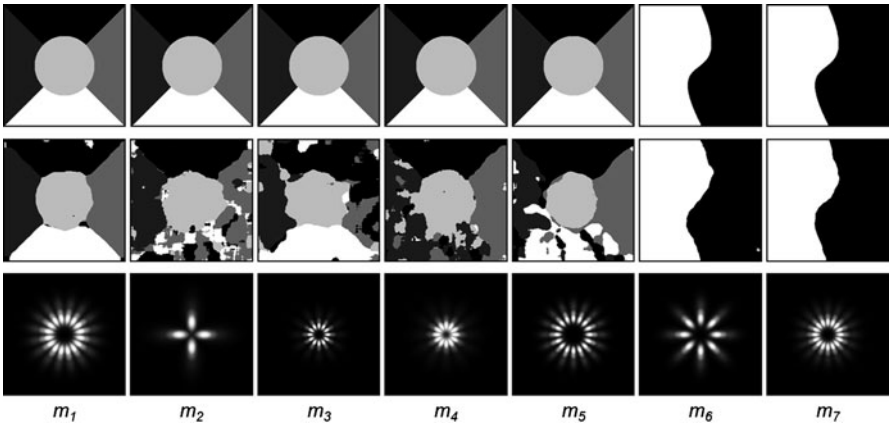

Fig. 13.9 Segmentation results for an optimized Gabor filter bank. The *top row* shows the ground truth map, and the *middle row* contains the best segmentation result obtained for the Gabor filter bank shown in the *bottom row* (Fourier domain)

human vision processes. Therefore, this baseline is no arbitrary choice of filter bank parameters but a setting that is hard to improve without excessive knowledge about Gabor filter design and the application domain.

Compared to the baseline, a substantial gain in segmentation accuracy could be achieved with Gabor filter bank optimization. Interestingly, the best filter banks differ with respect to many parameters. For instance, the best segmentation for m_2 was obtained for a filter bank with $\Theta = 2$, whereas for m_5, eight different orientations gave the best result. In general, a substantial reduction of the segmentation error rate (between 54.6% and 18.2%, average 27.5%) could be achieved. This clearly demonstrates that Gabor filter optimization should be conducted on a common basis, as feature selection and classifier hyperparameter selection, whenever a segmentation system is designed.

13.6 Conclusion

Features are a way of compressing raw data into a set of real numbers that represent the relevant information contained in the original data.

The properties of the raw data that are relevant cannot be determined a priori, but are application dependent. To tune features to the particular application, one may use the parameters available in feature calculation to optimize the features in such a way that the margin between the classes in a classification problem is increased. Furthermore, it may also shape the decision boundary so that it can be more easily reproduced by the classifier. This optimization may take place in an off-line mode or in an on-line mode.

In most cases, the assumption is that an initial set of features and possibly also an initial classifier are already in place. The purpose of feature optimization is then to provide an additional increase in robustness and classification performance. Tests on two quite different applications (decision making based on blob features and texture segmentation) have shown that parameter tuning has a positive influence on the accuracy of a classifier. This increase may be substantial in some situations and feature adaptation may be particularly helpful if the underlying processes that generate the raw data are instationary. The improvement that can be achieved, however, depends on the ability of the machine vision expert to preset the parameters. The main area in which feature optimization can be used with great effect are thus features with a larger number of tunable parameters and complex machine learning problems, where the effects of parameter changes cannot be easily assessed.

In this chapter, we demonstrated the effect of feature adaptation for two kinds of applications using blob features that are common in real-world inspection systems, as well as Gabor features that are a common choice for tasks related to segmentation, classification and retrieval of (textured) images. For both scenarios, we demonstrated a substantial reduction of the classification error rate that underpins the need for feature adaptation whenever a feature-based machine vision system is designed.

References

1. Azimi-Sadjadi, M.R.D.Y., Dobeck, G.J.: Adaptive feature mapping for underwater target classification. In: IJCNN '99. International Joint Conference on Neural Networks, vol. 5, pp. 3221–3224 (1999)
2. Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001)
3. Brodatz, P.: A Photographic Album for Artists and Designers. Dover Publications, New York (1966)
4. Cardie, C.: Using decision trees to improve case-based learning. In: Proceedings of 10th International Conference on Machine Learning, pp. 25–32 (1993)
5. Chen, H.T., Liu, T.L., Fuh, C.S.: Probabilistic tracking with adaptive feature selection. In: 17th International Conference on Pattern Recognition (ICPR'04), volume 2, pp. 736–739 (2004)
6. Collins, R., Liu, Y.: On-line selection of discriminative tracking features. In: Proc. of the 2003 International Conference of Computer Vision (ICCV 03), pp. 346–352 (2003)

7. Costanza, C.M., Afifi, A.A.: Comparison of stopping rules in forward stepwise discriminant analysis. Journal Amer. Statist. Assoc. **74**, 777–785 (1979)
8. Dash, M., Liu, H.: Feature selection for classification. International Journal of Intelligent Data Analysis **1**, 131–156 (1997)
9. Demant, C., Streicher-Abel, B., Waszkewitz, P.: Industrielle Bildverarbeitung. Springer-Verlag, Berlin Heidelberg New York (1998)
10. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edition. John Wiley & Sons, New York (2001)
11. Dunn, D., Higgins, W., Wakeley, J.: Texture segmentation using 2-d Gabor elementary functions. Pattern Analysis and Machine Intelligence, IEEE Transactions on **16**(2), 130 –149 (1994). DOI 10.1109/34.273736
12. Eitzinger, C., Gmainer, M., Heidl, W., Lughofer, E.: Increasing classification performance with adaptive features. In: A. Gasteratos, M. Vincze, J. Tsotsos (eds.) Proceedings of ICVS 2008, *LNCS*, vol. 5008, pp. 445–453. Springer, Santorini Island, Greece (2008)
13. Grigorescu, S.E., Petkov, N., Kruizinga, P.: Comparison of texture features based on gabor filters. In: IEEE Trans. on Image Process., vol. 11, pp. 1160–1167 (2002)
14. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. Journal of Machine Learning Research **3**, 1157–1182 (2003)
15. Hand, D.J.: Discrimination and classification. Wiley Series in Probability and Mathematical Statistics, Wiley, Chichester, UK (1981)
16. Jain, A.K., Farrokhnia, F.: Unsupervised texture segmentation using gabor filters. Pattern Recogn. **24**(12), 1167–1186 (1991). DOI http://dx.doi.org/10.1016/0031-3203(91)90143-S
17. Kim, M., Park, C., Koo, K.: Natural / man-made object classification based on gabor characteristics. In: W.K. Leow, M. Lew, T.S. Chua, W.Y. Ma, L. Chaisorn, E. Bakker (eds.) Image and Video Retrieval, *Lecture Notes in Computer Science*, vol. 3568, pp. 550–559. Springer Berlin / Heidelberg (2005)
18. Kohavi, R., John, G.: Wrappers for feature subset selection. Artificial Intelligence **97**(1–2), 273–324 (1997)
19. Kononenko, I.: Estimating attributes: Analysis and extensions of relief. In: Proceedings of ECML-94, pp. 171–182. Springer Verlag, Catania, Sicily (1994)
20. Krishnapuram, B., Hartemink, A.J., Carin, L., Figueiredo, M.A.T.: A Bayesian approach to joint feature selection and classifier design. IEEE Transactions on Pattern Analysis and Machine Intelligence **26**(9), 1105–1111 (2004)
21. Lee, T.S.: Image representation using 2d gabor wavelets. Pattern Analysis and Machine Intelligence, IEEE Transactions on **18**(10), 959–971 (1996). DOI 10.1109/34.541406
22. Li, M., Staunton, R.: Optimum gabor filter design and local binary patterns for texture segmentation. Pattern Recognition Letters **29**(5), 664–672 (2008). DOI 10.1016/j.patrec.2007.12.001
23. MIT Media Laboratory Cambridge: Vistex - Vision Texture Database. http://vismod.media.mit.edu/pub/VisTex/ (1995)
24. Molina, L.C., Belanche, L., Nebot, A.: Feature selection algorithms: A survey and experimental evaluation. In: ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining, pp. 306–311. Maebashi City, Japan (2002)
25. Narendra, P., Fukunaga, K.: A branch and bound algorithm for feature subset selection. IEEE Transactions on Computer **26**(9), 917–922 (1977)
26. University of Oulu, C.S., Laboratory, E.: Supervised texture segmentation mosaics. http://www.cse.oulu.fi/MVG/SupervisedTextureSegmentation/ (2011)
27. Rao, C.R.: Linear statistical inference and its applications. John Wiley & Sons, Inc., NY, U.S.A. (1965)
28. Reisert, M., Burkhardt, H.: Feature selection for retrieval purposes. In: Proceedings of the ICIAR'06, Vol. 1, pp. 661–672. Pavoa do Varzim, Portugal (2006)
29. Sandler, R., Lindenbaum, M.: Optimizing gabor filter design for texture edge detection andclassification. International Journal of Computer Vision **84**, 308–324 (2009). DOI 10.1007/s11263-009-0237-x

30. Smith, G., Burns, I.: Measuring texture classification algorithms. Pattern Recognition Letters **18**(14), 1495–1501 (1997). Http://www.cssip.uq.edu.au/staff/meastex/meastex.html
31. Thumfart, S., Heidl, W., Scharinger, J., Eitzinger, C.: A quantitative evaluation of texture feature robustness and interpolation behaviour. In: X. Jiang, N. Petkov (eds.) Computer Analysis of Images and Patterns, *Lecture Notes in Computer Science*, vol. 5702, pp. 1154–1161. Springer Berlin / Heidelberg (2009)
32. Tsai, D.M., Wu, S.K., Chen, M.C.: Optimal gabor filter design for texture segmentation using stochastic optimization. Image and Vision Computing **19**(5), 299–316 (2001). DOI 10.1016/S0262-8856(00)00078-0