Chapter 2

Introduction to Object Recognition

This chapter is a brief introduction to the principles of automatic object recognition. We introduce the basic terms, concepts, and approaches to feature-based classification. For understanding the rest of the book, the most important part of this chapter is the introduction of the term *invariant* and an overview of the invariants which have been proposed for visual object description and recognition. In addition to that, we concisely review the existing classifiers. The chapter starts with a short introduction to feature metric spaces[1].

2.1 Feature space

As we mentioned in Chapter 1, the features are measurable quantitative characteristics of the objects and images. From a mathematical point of view, the features are elements of a *feature space.*

Let us now look closer at the feature space, at its required and desirable properties, and at some connections with metric spaces, equivalence relations, set partition, and group theory. At this moment, we are not going to define any particular feature set. We stay on a general level such that the following considerations are valid regardless of the specific choice of the features.

All object recognition algorithms that act in a feature space use a measure of object similarity or dissimilarity as the central tool. Hence, most of the feature spaces are constructed such that they are *metric spaces.* For certain special features where the construction of a metric is not appropriate or possible, we weaken this requirement and create *pseudometric*, *quasimetric*, *semimetric*, or *premetric* spaces. In some other cases, the similarity/dissimilarity measure is not explicitly defined, but it is still present implicitly.

[1]Readers who are familiar with these topics at least on an intermediate level may skip this chapter and proceed directly to Chapter 3, where the theory of moment invariants begins.

2.1.1 Metric spaces and norms

A metric space is an ordered pair (P, d), where P is a non-empty set and d is a *metric* on P.

Definition 2.1: A real-valued function d defined on $P \times P$ is a *metric* if it satisfies the four following axioms for any $\mathbf{a}, \mathbf{b}, \mathbf{c} \in P$.

1. $d(\mathbf{a}, \mathbf{b}) \geq 0$ (positivity axiom),
2. $d(\mathbf{a}, \mathbf{b}) = 0 \quad \Leftrightarrow \quad \mathbf{a} = \mathbf{b}$ (identity axiom),
3. $d(\mathbf{a}, \mathbf{b}) = d(\mathbf{b}, \mathbf{a})$ (symmetry axiom),
4. $d(\mathbf{a}, \mathbf{b}) \leq d(\mathbf{a}, \mathbf{c}) + d(\mathbf{b}, \mathbf{c})$ (triangle inequality).

The first axiom follows from the other three ones and could be omitted. The function d is also called *distance function* or simply *distance* between two points. The conditions 1–4 express our intuitive notion about the distance between two points. If axiom 2 is weakened such that only $d(\mathbf{a}, \mathbf{a}) = 0$ is required but possibly also $d(\mathbf{a}, \mathbf{b}) = 0$ for some distinct values $\mathbf{a} \neq \mathbf{b}$, then d is called *pseudometric*. If d satisfies 1, 2, and 4 but it is not necessarily symmetric, we call it *quasimetric*; *semimetric* is such d that satisfies the first three axioms, but not necessarily the triangle inequality. The weakest version is a *premetric*, which only requires $d(\mathbf{a}, \mathbf{b}) \geq 0$ and $d(\mathbf{a}, \mathbf{a}) = 0$. Some of these modifications were inspired by real-life situations. Imagine the "distance" which is measured by the time you need to get from place $\mathbf{a}$ to place $\mathbf{b}$ by bike. Obviously, if the road is uphill, then $d(\mathbf{a}, \mathbf{b}) > d(\mathbf{b}, \mathbf{a})$ and d is a quasimetric. If the distance between two countries is defined as the shortest distance between the points on their borders, then the distance between any two neighboring countries is zero and we get a (symmetric) premetric. The reader can find many other examples easily.

For the given set P, there may exist many (even infinitely many) various metrics which all fulfill the above definition but induce different distance relations between the points of P. Hence, when speaking about a metric space, it is always necessary not only to mention the set P itself but also to specify the metric d.

The definition of a metric space does not require any algebraic structure on P. However, most feature spaces are *normed vector spaces* with the operations addition and multiplication by a constant and with a norm $\|\cdot\|$. Let us recall that any norm must satisfy certain conditions.

Definition 2.2: A real-valued function $\|\cdot\|$ defined on a vector space P is a *norm* if it satisfies the four following axioms for any $\mathbf{a}, \mathbf{b} \in P$ and a real or complex number α.

1. $\|\mathbf{a}\| \geq 0$,
2. $\|\mathbf{a}\| = 0 \quad \Rightarrow \quad \mathbf{a} = \mathbf{0}$,
3. $\|\alpha \cdot \mathbf{a}\| = |\alpha| \cdot \|\mathbf{a}\|$ (homogeneity axiom),

4. $\|\mathbf{a}+\mathbf{b}\| \leq \|\mathbf{a}\| + \|\mathbf{b}\|$ (triangle inequality).

Similarly to the case of metric, the first axiom follows from the homogeneity and triangle inequality and could be omitted.

If we now define d as

$$d(\mathbf{a},\mathbf{b}) = \|\mathbf{a}-\mathbf{b}\|,$$

then, thanks to the properties of the norm, such d actually fulfills all axioms of a metric. We speak about the metric that is *induced* by the norm[2]. Any norm induces a metric, but only some metrics are induced by a norm. The norm is a generalization of our intuitive notion of the vector "length" and can be also understood as a distance of $\mathbf{a}$ from the origin. If P is a space with a scalar product, then the norm is induced by this scalar product as $\|\mathbf{a}\| = \sqrt{(\mathbf{a},\mathbf{a})}$.

Feature spaces of finite dimensions[3] are mostly (but not necessarily) equivalent to n-dimensional vector space of real numbers $\mathbb{R}^n$, and we use the notation $\mathbf{a} = (a_1, a_2, \ldots, a_n)^T$. The most common norms in such spaces are ℓ_p norms

$$\|\mathbf{a}\|_p = (\sum_{i=1}^{n} |a_i|^p)^{1/p}$$

and weighted ℓ_p norms

$$\|\mathbf{a}\|_p^w = (\sum_{i=1}^{n} w_i|a_i|^p)^{1/p},$$

where $p \geq 1$ and w_i are positive constants. The ℓ_p norms are well known from linear algebra and for certain p's they are of particular interest. For $p = 1$, it turns to the *city-block norm* popular in digital geometry. Euclidean norm, probably the most intuitive and the most frequently used norm ever, is a special case if $p = 2$ (we mostly drop the index p for simplicity when using Euclidean norm and write just $\|\mathbf{a}\|$). If $p \to \infty$, the ℓ_p norm converges to the *maximum norm* ℓ_∞

$$\|\mathbf{a}\|_\infty = \max_i(|a_i|)$$

which is also known as the *chessboard norm.* If $0 < p < 1$, then the triangle inequality is violated, and the corresponding ℓ_p-norm is actually only a seminorm, but it still finds applications: for instance, in robust fitting the feature points with a curve/surface.

Introducing the weights w_i into the ℓ_p norm is of great importance when constructing the feature space and often influences significantly the classification results. The goal of the weights is to bring all the individual features (i.e. components a_i of the feature vector) to approximately the same range of values. It is equivalent to

[2]Although this is the most common way to induce a metric by the norm, it is not the only one. However, the other options are rather obscure and of low importance when working in feature spaces. An example is the rail metric defined as $d(\mathbf{a},\mathbf{b}) = \|\mathbf{a}\| + \|\mathbf{b}\|$ for distinct $\mathbf{a},\mathbf{b}$ and $d(\mathbf{a},\mathbf{a}) = 0$. Its name somehow ironically reflects the fact that a railway trip usually passes through the main hub regardless, where the final destination is.

[3]Feature spaces of infinite dimensions are of very little practical interest, and we do not discuss them in this book.

a non-uniform scaling of the feature space. Why is it so important? The features may be quantities of very different character and range. Suppose that, for example, $a_1 \in (-1, 1)$ while $a_2 \in (-10^{10}, 10^{10})$. When using the pure Euclidean norm, we actually lose the information that is contained in the feature a_1 because the a_2 values are so big that $\|\mathbf{a}\| \doteq |a_2|$. However, if we use the weights $w_1 = 1$ and $w_2 = 10^{-20}$, then small relative changes of a_1 have approximately the same impact on the norm as small relative changes of a_2.

There are of course many vector norms which are not ℓ_p-norms. An example is a Hamming norm (sometimes referred to as ℓ_0-norm), which is simply the number of the non-zero components of $\mathbf{a}$.

Although the (weighted) metrics induced by ℓ_p norms are almost a standard choice, there exist feature spaces where these metrics are not appropriate. Consider a feature space that consists of individual features, which are very different from one another in their nature. The first feature is real-valued, the second is a periodic quantity (for instance the orientation angle or the phase of a complex number), the third feature is a logical variable, and the last one expresses a subjective rating on the scale "excellent-good-fair-average-poor". Here the ℓ_p norm would not be a good choice, and we have to construct the metric carefully such that it reflects the information contained in all features and weights it according to its actual importance for the given task.

2.1.2 Equivalence and partition

The most transparent way to explain the principles of object classification is through two elementary terms of the set theory – *partition* and *equivalence*. Given a non-empty set M, a set of its subsets $\{M_1, \ldots, M_C\}$ such that[4]

1. $M_i \neq \emptyset \quad \forall i$,
2. $M_i \cap M_j = \emptyset \quad \forall i \neq j$,
3. $\bigcup\limits_{i=1}^{C} M_i = M$

is called partition of M.

Partition is closely connected with the equivalence relation on $M \times M$. Let us recall that a binary relation $\approx$ is called equivalence if and only if it is reflexive, symmetric, and transitive, that is,

1. $a \approx a$ for any $a \in M$,
2. If $a \approx b$, then $b \approx a$ for any $a, b \in M$,
3. If $a \approx b$ and $b \approx c$, then $a \approx c$ for any $a, b, c \in M$.

Any equivalence on $M \times M$ unambiguously determines a partition of M and vice versa. The components M_i are then called *equivalence classes*. Clearly, if equivalence $\approx$ is given on $M \times M$, we can choose an arbitrary $a \in M$ and find a set $M_a =$

[4]We use a notation for finite number of components C because object classification is always to a finite number of classes. A set partition in general may have an infinite (countable or uncountable) number of components.

$\{x \in M | a \approx x\}$. This is our first partition component. Then we find $b \in M$ such that $b \notin M_a$ and construct the partition component M_b. We repeat this process until the whole M is covered. The sets $M_a, M_b, \ldots$ fulfill the definition of the partition. The backward implication is even more apparent. If partition $\{M_1, \ldots, M_C\}$ of M is given, then the relation

$$a \approx b \quad \Leftrightarrow \quad a, b \text{ lie in the same component } M_i$$

is an equivalence on $M \times M$.

If we want to abstract away from the properties that may change inside the equivalence classes, we may work with a set of the classes that is called the *quotient set* and is denoted as $(M/\approx)$. Working with quotient sets is very common in mathematics because it helps to concentrate on the substantial properties of the objects and to ignore the insignificant ones. If there has been some algebraic and/or metric structure defined on M, it usually smoothly propagates into $(M/\approx)$. In object classification, the classes (determined by the user) are in fact equivalence classes in the object space. In face recognition, for instance, the person should be identified regardless of the head pose, facial expression, and hair style; in character recognition, the type and the color of the pen, the font, size and orientation of the character are absolutely insignificant parameters, which should in no way influence the recognition (provided that we are interested in "reading" the characters, not in font/size recognition). Hence, the equivalence classes are defined as "all pictures of the person A" and "all possible appearance of the letter B", etc. The variability within each class, which is insignificant for the given task, is called *intraclass variability.* The intraclass variability can be described by an operator $\mathcal{D}$, which maps any object to another object *of the same class.* The relation between two objects

$$f \approx g \quad \Leftrightarrow \quad \text{There exists } \mathcal{D} \text{ such that } g = \mathcal{D}(f) \text{ or } f = \mathcal{D}(g)$$

should be an equivalence that induces the same partition into classes as the one specified by the user. Note that the operator $\mathcal{D}$ is *not* defined "automatically" by selecting the object space. In one object space, we may have several meaningful task formulations, each of them leading to distinct space partitions. As an example, consider again the space of portrait photographs. One task could be person recognition, the other one gender recognition, and yet another one classification into age categories.

Very often, $\mathcal{D}$ describes all possible object degradations that may arise in the given task: if f is an "ideal" image of an object from the class M_f, then $g = \mathcal{D}(f)$ can be understood as a degraded version of f but still belonging to M_f. The ideal classification algorithm should assign g into the class M_f regardless of particular form and parameters of $\mathcal{D}$ because they are usually unknown.

From the above considerations, we can clearly see one of the fundamental requirements of object recognition. The operator of the intraclass variability $\mathcal{D}$ must be *closed under a composition* in each equivalence class. If $\mathcal{D}$ is applied twice (possibly with different parameters), then the composition $\mathcal{D}_c = \mathcal{D}_1\mathcal{D}_2$ must be of the same type. Without this *closure property* the task would be ill-posed because the "classes" would not be equivalence classes and would not define a partition, so an object could be an element of multiple classes. Since the closure property is often coupled with *associativity* and *invertibility* of $\mathcal{D}$ (which is, however, much less important here), the intraclass transformations form a *group* (or a *monoid* if $\mathcal{D}$ is not invertible) and the action of $\mathcal{D}$ is a *group action.* Typical examples of intraclass variations that form a group are object rotation, translation, and scaling (similarity group), affine transformation (affine group), perspective projection (projective group). Blurring the image by a Gaussian kernel does not form a group because it is not invertible but forms a monoid. On the other hand, quadratic transformation of the image coordinates is not closed and does not form equivalence classes.

2.1.3 Invariants

Now we approach the explanation of the central term of the feature-based object recognition – the notion of invariants. The *invariant feature* or simply the *invariant* is a functional that maps the image space into the feature space such that $I(f)$ depends on the class f belongs to but does not depend on particular appearance of f. In other words, $I(f) = I(\mathcal{D}(f))$ for any f and any instance of $\mathcal{D}$. In yet other words, I is actually defined on a quotient object space factorized by $\mathcal{D}$. Simple examples of the invariants under a similarity transformation are angles and ratios, the object size is invariant to rotation, and the property of two lines "to be parallel" is invariant under affine transformation.

The functional I that satisfies the above definition is sometimes called the *absolute* invariant, to emphasize the difference from *relative* invariants. The relative invariant satisfies a weaker and more general condition $I(f) = \Lambda(\mathcal{D})I(\mathcal{D}(f))$, where Λ is a scalar function of the parameters of $\mathcal{D}$. If $\mathcal{D}$ is for instance an affine transformation, $\Lambda(\mathcal{D})$ is often a function of its Jacobian. Relative invariants cannot be used directly for classification since they vary within the class. It is usually not difficult to normalize a relative invariant by another relative invariant to cancel the factor $\Lambda(\mathcal{D})$ and to get an absolute invariant.

The invariance property is the necessary but not sufficient condition for I to be really able to classify objects (for example, the trivial feature $I(f) = 0$ is perfectly invariant but completely useless). Any functional I defines another equivalence (and hence another partition and also another quotient space) in the object space

$$f \sim g \quad \Leftrightarrow \quad I(f) = I(g).$$

The partition induced by equivalence $\sim$ cannot be arbitrary. Since I is supposed to be an invariant, the quotient space $(M/\sim)$ can only be the same or coarser than $(M/\approx)$. In an ideal case both partitions are equal, i.e. $(M/\sim) = (M/\approx)$, and I is said to be *discriminative* because it has distinct values on objects that belong to distinct user classes. If it is not the case, then there exist at least two objects f and g belonging to different classes but fulfilling $I(f) = I(g)$, which means they are not discriminable by means of I.

Invariance and discriminability are somehow opposed properties. One should keep in mind that the invariance property is not something "always advantageous".

Making the invariance broader than necessary decreases the recognition power (for instance, using affine invariants instead of similarity invariants leads in the feature space to mixing all parallelograms together). The user must tune these two properties of the features with respect to the given data and to the purpose. Choosing a proper trade-off between the invariance and the discrimination power is a very important task in feature-based object recognition (see Fig. 2.1 for an example of a desirable situation).

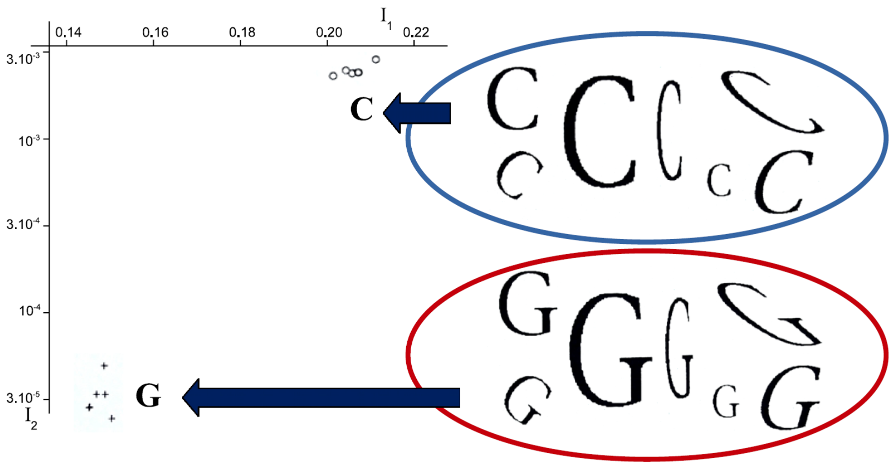


Figure 2.1: Two-dimensional feature space with two classes, almost an ideal example. Each class forms a compact cluster (the features are invariant to translation, rotation, scaling, and skewing of the characters) and the clusters are well separated from one another (the features are discriminative although the characters are visually similar to each other).

Usually, a single real-valued invariant does not provide enough discrimination power, and several invariants $I_1, \ldots, I_n$ must be used simultaneously. Then we speak about an *invariant vector* and the feature space has n dimensions. The invariant vector should be *independent*, which means that none of its components is a function of the others[5]. This is a natural and important requirement. Dependent invariants do not contribute to the discrimination power of the vector, they only increase the dimensionality of the feature space. This leads not only to increasing the time complexity of the classification algorithms and of the feature measurement (and in practice often to a growth of the cost), but also may cause a drop of performance.

Apart from the object recognition, invariants often play an important role in the reconstruction of the original object from its feature vector. We face this situation when the invariants are considered an alternative representation of the objects and are stored in a database instead of the original images. The loss-less reconstruction is possible only if the vector is *complete*, which means that other independent features do not exist. Such a reconstruction is of course possible only modulo $\mathcal{D}$ since there is no way to recover the particular appearance from the invariants. In case of digital images, complete invariant vectors usually have almost the same number of components as the number of the pixels in the input image. Hence, the requirement of completeness is for classification purposes often highly excessive and useless.

[5]The feature dependence/independence in this sense is not necessarily connected with the correlation, which is evaluated pair-wise on the training set and reflects a linear component of the dependency only.

2.1.4 Covariants

Unlike the invariants, there exist features $C(f)$ that change within the classes, i.e. $C(f) \neq C(\mathcal{D}(f))$. They are called *covariants*[6]. A covariant usually does not have the ability to discriminate the classes (if it had such ability, it could be turned into an invariant by thresholding) but carry the information about the particular intra-class parameters of f, which may be very useful when transforming the object into certain canonical (standard) positions. Such transformation is called *object normalization*. If, for instance, the intra-class variation is a rotation, then the directions of the principal axes of the objet are covariants. When defining the standard position such that the principal axes are horizontal and vertical, we need to know their actual orientation in order to properly rotate the object. If we want to reconstruct the object including its particular position in the class, we also need, in addition to a complete invariant vector, a complete covariant vector.

2.1.5 Invariant-less approaches

For the sake of completeness, let us mention that along with the invariant approach, there exist two other approaches to visual object recognition – *brute force* and *image normalization*. In the brute force approach, we search the parametric space of all possible image degradations. That means the training set of each class should contain not only all class representatives but also all their rotated, scaled, blurred, and deformed versions. Clearly, this approach would lead to extreme time complexity and is practically inapplicable. In the normalization approach, the objects are transformed into a certain standard position before they enter the classifier. This is very efficient in the classification stage, but the object normalization itself usually requires solving difficult inverse problems that are often ill-conditioned or even ill-posed. For instance, in case of image blurring, "normalization" means in fact blind deconvolution [1], and in case of spatial image deformation, "normalization" requires performing registration of the image to some reference frame [2].

2.2 Categories of the invariants

The existing invariant features for object description and recognition can be categorized from various points of view (see the survey papers [3] and [4]). The main categorization criteria, which have appeared in literature, are the following.

1. The dimension of the objects the invariants describe,
2. The type of the invariance (i.e., the type of the intra-class variations),
3. The range of intensity values of the object,
4. The part of the object which is necessary for the invariant computation,
5. The mathematical tools used for the construction of the invariants.

[6]The terminology has not been unified here. Some authors use the term "variant feature". The term "covariant" has also other meanings in tensor algebra, in programming languages, and in quantum mechanics.

The first three categorizations are very natural and straightforward; they, however, sort the invariants into only a few categories. When sorting according to the dimensionality[7], we speak about 2D invariants, 3D invariants, and d-D invariants for $d > 3$. According to the type of the invariance, we have geometric invariants or intensity-based ones. In this first category we recognize translation, rotation, scaling, affine, and projective geometric invariants. Invariants to intensity variations exist with respect to linear contrast stretching, non-linear intensity transformations, and convolution. Sorting according to the range of the intensity values distinguishes invariants of binary, gray-level, and color objects and of vector-valued images.

The fourth viewpoint reflects what part of the object is needed to calculate the invariant. *Global* invariants are calculated from the whole image (including background if no segmentation has been performed). Most of them include projections of the image onto certain basis functions and are calculated by integration. Comparing to local invariants, global invariants are much more robust with respect to noise, inaccurate boundary detection, and similar factors. On the other hand, their serious drawback is the fact that any local change of the image influences the values of all invariants and is not “localized” in a few components only. This is why global invariants cannot be used when the object in question is partially occluded by another object and/or when a part of it is out of the visual field. *Local* invariants are, on the contrary, calculated from a certain neighborhood of dominant points only. Their primary purpose is to recognize objects which are visible only partially. Differential invariants are typical representatives of this category. *Semi-local* invariants attempt to keep the strengths of the two above groups and to overcome their weaknesses. They divide the object into stable parts and describe each part by some kind of global invariants.

Categorization according to the mathematical tools used is probably the most detailed type. It provides a grouping, where the common factor is the mathematical background, which at the same time often implicitly determines the feature categorization according to the previous four criteria. One can find numerous categories of this kind in the literature, depending on the “depth” of sorting. Here we present eight basic ones – simple shape features, complete visual features, transformation coefficient features, wavelet-based features, textural features, differential invariants, point set invariants, and moment invariants.

2.2.1 Simple shape features

Simple shape features, sometimes referred to as *visual features*, are low-dimensional features of very limited discriminability. On the other hand, they are easy to compute (or even visually estimate) and interpret, which implies their popularity. They have been designed for binary objects (shapes) and are defined for 2D objects only, although some of them can be readily extended to 3D as well. They are mostly based on the measurement of object similarity to a certain reference shape – a circle, an ellipse, or a rectangle.

One of the oldest and widely used is *compactness* defined as

$$c = \frac{4\pi S(A)}{O(A)^2},$$

[7]The term “dimension” here refers to the dimension of the object/images, not to the dimension of the invariant vector.

where $S(A)$ is the area and $O(A)$ is the perimeter of object A. It holds $0 < c \leq 1$, where $c = 1$ comes only if A is a circle. The name of this feature originates from our intuitive notion of the circle as the "most compact" object. That is why this feature is sometimes called *circularity*, although other definitions of circularity which come from the fit of the object boundary by a circle can be found in the literature (see [5], for instance).

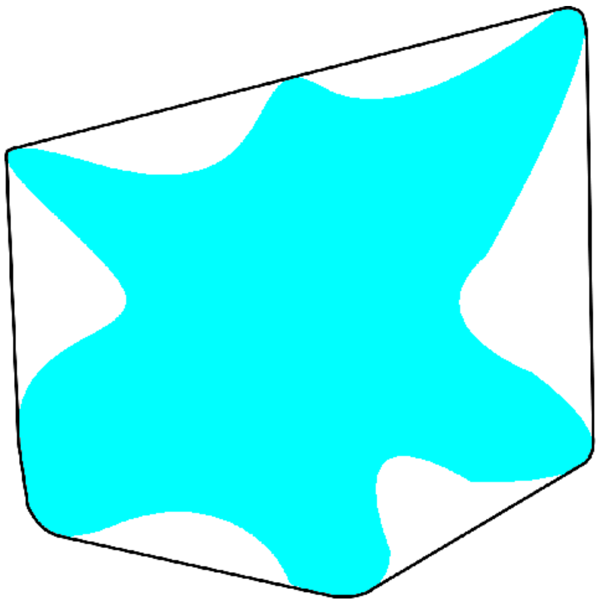


Figure 2.2: The object and its convex hull.

The feature, which reflects the similarity between the object and its convex hull is called *convexity*, defined as

$$v = \frac{S(A)}{S(C_A)},$$

where C_A is the convex hull of A (see Fig. 2.2). Clearly, $0 < v \leq 1$, and $v = 1$ for convex shapes only.

Another feature called *rectangularity* measures the similarity to the minimum bounding rectangle R_A (see Fig. 2.3)

$$r = \frac{S(A)}{S(R_A)}.$$

Again, $0 < r \leq 1$ and the equality arises for rectangles only. There is another popular feature of this kind called *elongation*. It is again derived from the minimum bounding rectangle as the ratio of its side lengths.

Two other simple features refer to the properties of ellipses. The *ellipticity* is given by an error of the fitting the object by an ellipse. The *eccentricity* is simply the eccentricity of the fitted ellipse (other definitions can be found in literature).

Another scalar feature is the *bending energy* [6] of the object boundary. The bending energy is in the discrete case defined as a sum of the square of the boundary

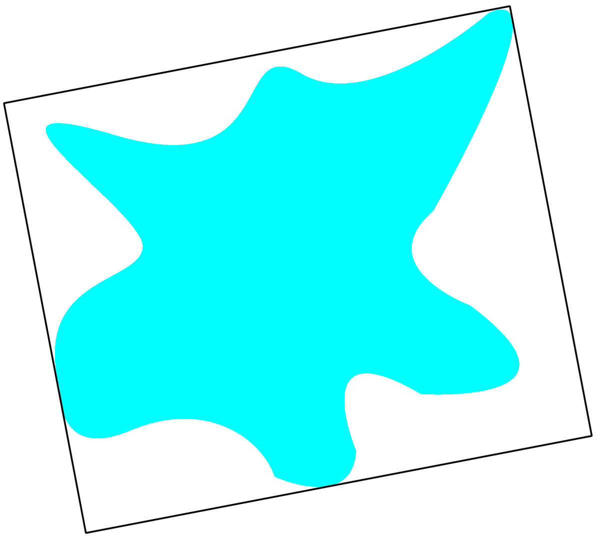


Figure 2.3: The object and its minimum bounding rectangle.

curvature over all boundary points, normalized by the total boundary length. We can imagine this feature as a potential energy of an ideally flexible metal wire which is deformed into the shape of the boundary. The bending energy is minimum for a circle, which corresponds to the physical interpretation.

All the simple shape features mentioned above are inherently invariant to translation, rotation, and uniform scaling. Individual discrimination power of each of them is poor, but when used together, they may provide enough discriminability to resolve at least easy tasks or to serve as pre-selectors in a more difficult recognition. To learn more about simple shape features we refer to the monograph [5] and to the papers by Rosin et al. [7, 8, 9, 10, 11, 12].

2.2.2 Complete visual features

A natural step forward is to stay with binary objects but to require a completeness of the features. This cannot be fulfilled by the features from the previous section. Probably the simplest way to reach the completeness while keeping the translation, rotation, and scaling invariance is to use polar encoding of the shape.

We explain this possibility for *star-shaped* objects first (the object is called star-shaped if any half line originating in the object centroid intersects the object boundary just once). Let us put the coordinate origin into the object centroid and construct boundary radial function $r(\theta)$, $\theta \in \langle 0, 2\pi)$. This function fully describes the object (see Fig. 2.4). Instead of computing features of the object, it is sufficient to compute features of the radial function. The radial function is invariant to object translation and undergoes a cyclic shift when the object rotates. The change of the starting point at the boundary has the same impact. If we now sample the radial function in a constant angular step, we obtain the *radial shape vector* [13] (see Fig. 2.5). The radial shape vector can be used directly as the feature vector. We can control the

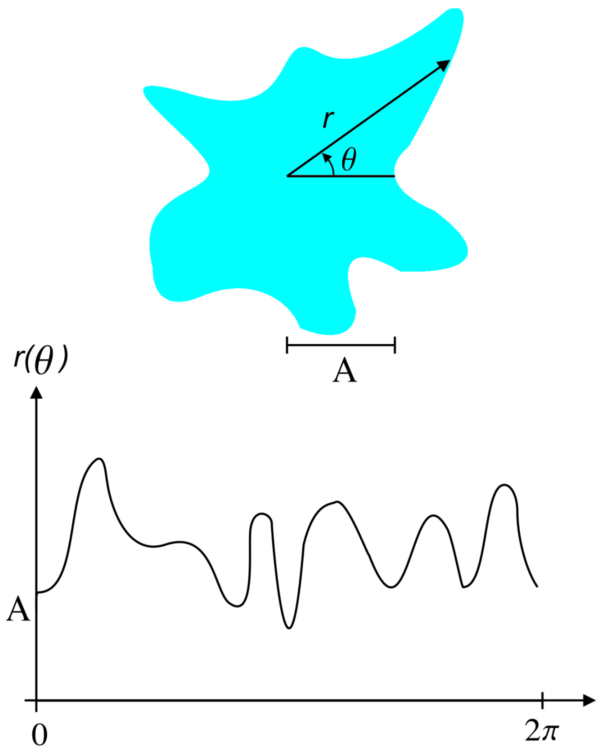


Figure 2.4: Radial function of the object.

discrimination power by increasing/decreasing the shape vector length (i.e., by changing the angular sampling step). If the similarity (distance) between two radial shape vectors $\mathbf{a}, \mathbf{b}$ is defined as the minimum of $d(\mathbf{a}, \mathbf{b})$, where the minimum is searched over all cyclic shifts of $\mathbf{b}$, then we get a reasonable metric which ensures invariants to rotation. Scaling invariance can be achieved by normalizing the shape vector by its largest element.

The concept of the polar encoding can be generalized into the *shape matrix*, which is a binary matrix of user-defined dimensions [14] (see Fig. 2.6). A polar grid of concentric circles and radial lines is placed into the object centroid such that the largest circle circumscribes the object. Each segment of the grid corresponds to one element of the shape matrix. If the segment is covered by the object, the corresponding value in the matrix is 1 and is zero otherwise. Since the segments have different sizes, appropriate weighting of the matrix elements was proposed in [15]. The shape matrix can be used for any object of a finite extent regardless of its particular shape. It works also for the shapes with holes and with multiple isolated components.

The *chain code*, which was introduced by Freeman in [16] and further developed in [17, 18] has become very popular in loss-less binary image compression and can also be considered a complete visual feature. The discrete object boundary is represented by a sequence of elementary directional unit vectors, each direction being encoded by an integer (usually from 0 to 3 or from 0 to 7, depending on the topology). If

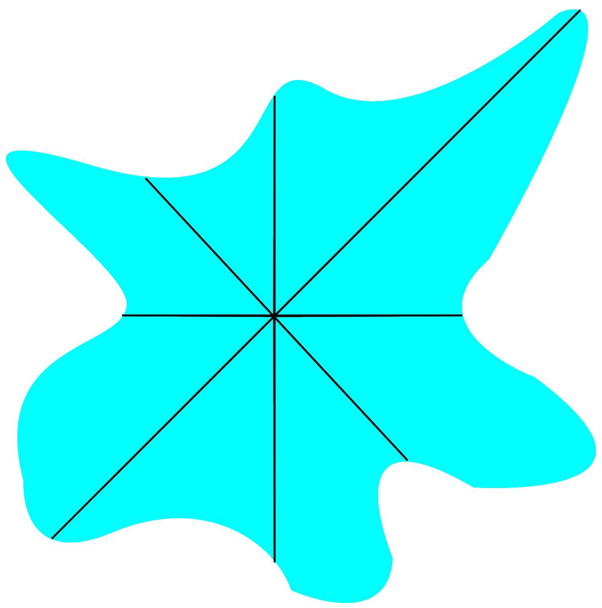
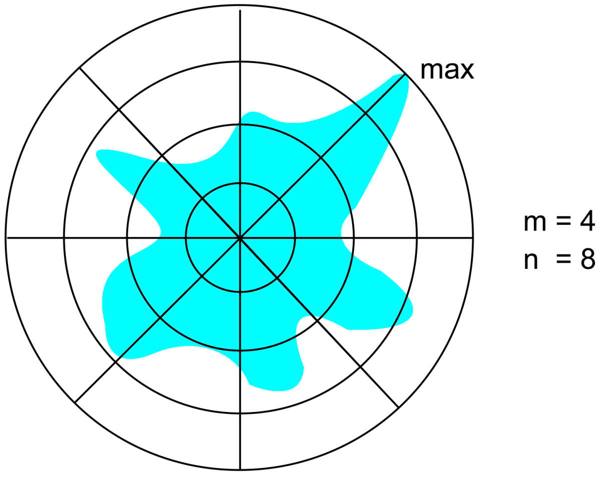


Figure 2.5: Star-shaped object and its radial shape vector.



$$B = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Figure 2.6: The object and its shape matrix.

implemented as a relative (differential) code, it provides rotational invariance while the invariance to the starting point is achieved by cyclic shifting of the code. Chain codes are, however, not frequently used as features for object recognition because there is no simple metric which would be consistent with the actual difference between the objects. Hence, a comparison of two chain codes should be done via maximum substring matching, which is relatively slow.

2.2.3 Transformation coefficient features

These features have been primarily designed for binary objects, both 2D and 3D. They are calculated from the coefficients of certain transformation of the object boundary. Fourier transformation has been used most often for this purpose [19, 20, 21] but using Walsh-Hadamard and other similar transformations is also possible. The key idea behind these features is to use polar or log-polar transformations in order to transfer rotation and scale into a (cyclic) shift, which is much easier to handle because it preserves the Fourier transformation magnitude as is implied by the Fourier Shift Theorem.

If an object has a well-defined boundary radial function $r(\theta)$, we compute its 1D Fourier transformation. Since the object rotation results in a cyclic shift of $r(\theta)$ and the scaling with a factor s results in the radial function $s \cdot r(\theta)$, the Fourier transformation magnitude is under these two transformations only multiplied by s. Hence, the magnitude normalized by the first coefficient is invariant w.r.t. TRS (translation, rotation, and uniform change of scale) transformation. In practice, we take only the first n coefficients of discrete Fourier transformation; by increasing n we increase the discriminability. These features are known as the *Fourier descriptors.*

In the general case, the function $r(\theta)$ does not exist, but we still can construct the Fourier descriptors in another way. Consider the boundary pixels (x_k, y_k) as points in the complex plane $z_k = x_k + iy_k$. Now we can calculate discrete Fourier transformation of $(z_1, z_2, \ldots)$. The position of the object in the plane is encoded solely in the Fourier coefficient Z_0. Discarding them, we get a translation invariance. Rotation invariance along with the invariance to the origin of the sampling is achieved by taking the magnitudes $|Z_k|$'s, similarly to the previous case. Finally, the normalization by $|Z_1|$ yields the scaling invariance. Alternatively, we can use the phase of Z_1 for normalization of the other phases to rotation and sampling origin. The phases are not changed under scaling.

Fourier descriptors are very popular and powerful features in the presence of TRS transformation. In case of binary images, they compete with moment invariants. In a different form, they can be applied to graylevel images as well. If only an image translation was considered, we could use the Fourier transformation magnitude directly as an invariant. If only a rotation and scaling (but no translation) were present, then we could make a log-polar transformation of the image and again to use its Fourier transformation magnitude. However, if a full TRS transformation is present, we have to insert an intermediate step. First, we make a Fourier transformation of the original image and take its magnitude (which is invariant to shift). The log-polar transformation is now applied in the frequency domain to the magnitude. Then another Fourier transformation of this magnitude yields the TRS invariants.

Radon transformation offers another, yet similar possibility, of to handle object rotations. The rotation (and the choice of the origin of angular projections) generates a one-dimensional cyclic shift in the Radon spectrum, which can be again eliminated by 1D Fourier transformation or by calculating the distance over all possible shifts.

2.2.4 Textural features

A particular group of features has been designed for description of images with prominent *textures*. The texture is a repeating pattern in graylevels or colors which may not be strictly periodic – it may contain irregularities and random perturbations. For objects made from materials such as wood, sand, skin, and many others, their texture is a dominant property; more class-specific than the shape and the color (see Fig. 2.7 for some examples).

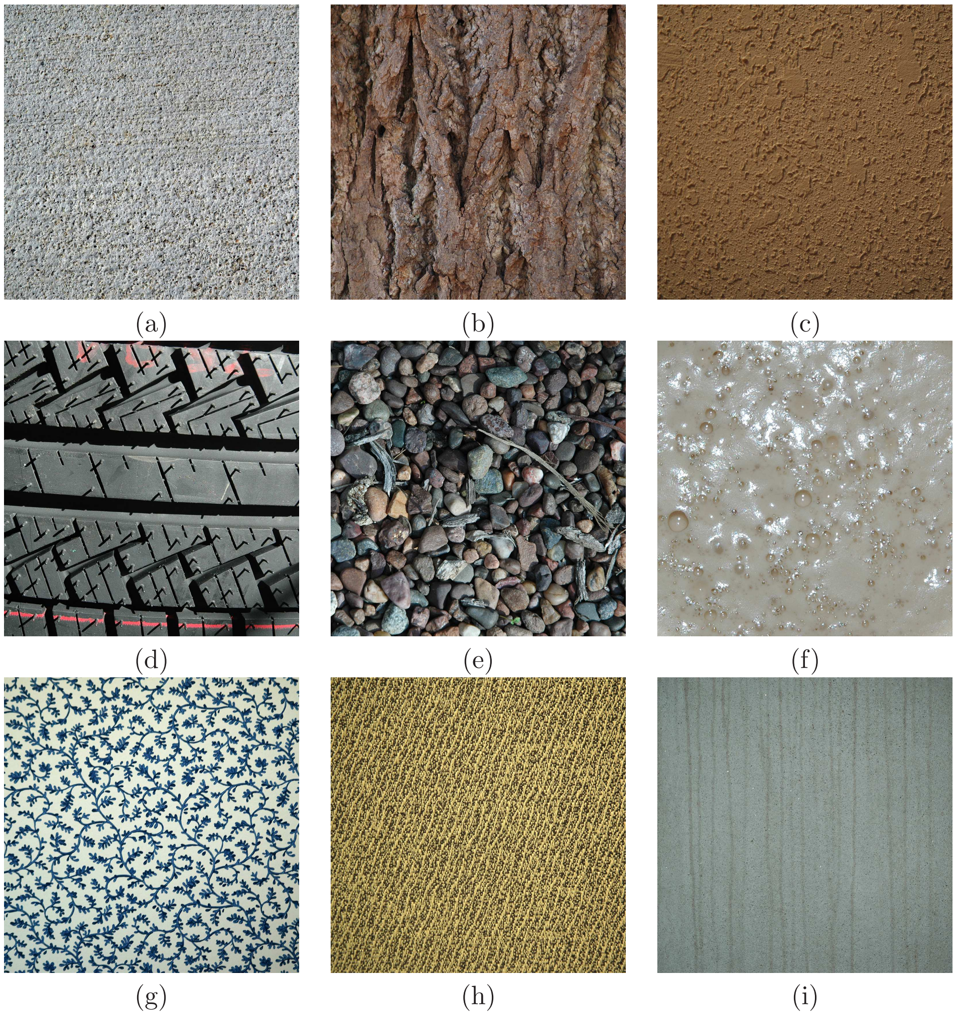


Figure 2.7: Examples of textures. The texture is often a more discriminative property than the shape and the color.

All textural features try to somehow capture the spatial appearance and its possible repetition frequency in different directions. In principle, any local directional features can be used for texture description. Autocorrelation function and its spectrum are straightforward general solutions. The review of textural features can be found in [22] and in the monograph [23], where the reader will also find many other relevant topics such as texture image preprocessing and texture modeling.

One of the first specialized textural features were proposed by Haralic et al. [24] in the form of spatial-dependence matrix. This idea has been re-used many times and has led to co-occurrence matrices. Another approach models the texture as a random process (AR, ARMA, and Markov models have been used for this purpose), the parameters of which serve as the features. This approach requires estimating the process parameters from the image, which may not be trivial.

Very powerful textural features are the *local binary patterns* (LBPs) introduced by Ojala et al. [25]. The LBPs are high-dimensional features the core idea of which is to encode, for each pixel, whether or not its value is higher than that of the neighboring pixels. The LBPs exist in many versions and extensions, the most important are multiresolution LBPs [26] and 3D LBPs [27].

Most recently, the attention of researchers has been drawn to the *Bidirectional texture function* (BTF), which describes a kind of texture the appearance of which depends not only on the position in the scene but also on the view and illumination spherical angles. BTF is highly redundant description, so for recognition it is usually used after a lossy compression. The notion of BTF was introduced by Dana et al. [28]. Later on, the methods for modelling, recognition, and synthesis of the BTF were developed by Haindl et al. [29, 30].

2.2.5 Wavelet-based features

In fact, this category could be included in the previous two sections, but due to its importance it is handled separately. Wavelet-based features make use of the *wavelet transformation* (WT) [31], which provides a spatial-frequency representation of the analyzed image (see Fig. 2.8 for an example), and thus it is able to capture both the shape and textural information of the object. The two most frequent ways to incorporate wavelet transformation in the object description are the following. For binary objects, the features are constructed from 1D WT of the boundary [32, 33] which is analogous to Fourier descriptors. Using a proper boundary parametrization, these features can be made even affine invariant [34, 35]. A comparison of the efficiency of the Fourier and wavelet boundary coefficients can be found in [36] (the authors claimed wavelets slightly outperform Fourier descriptors).

Graylevel images are described by features which are calculated from individual bands of 2D WT and capture the object texture. The texture-oriented descriptors are based on the representation of an energy distribution over wavelet subbands, such as overall energy, entropy, or covariance between decompositions of individual color channels (in the case of color images) [37]. The decomposition can be done by means of wavelet transformation often using Gabor or Daubechies wavelets, but other approaches are used too, such as wavelet packet transformation [38] and wavelet frames. One of the first applications was described in [39]. Great attention has been paid to proper selection of the bands used for the descriptor construction [40].

Recently, with increasing computational power, the third approach to wavelet-based object description has begun to appear. The overcomplete representation of objects is created by means of all coefficients at chosen coarser levels of the object wavelet decomposition [41]. Such feature vectors are then used as object descriptors.

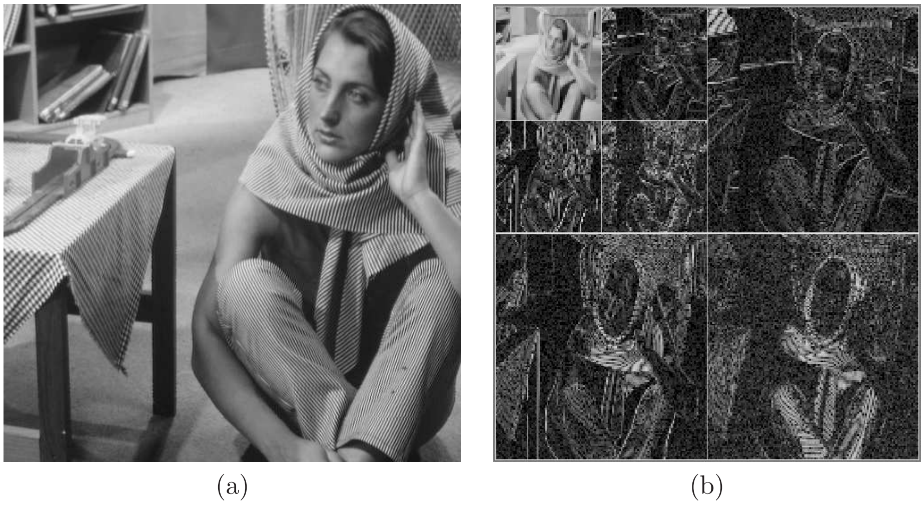


Figure 2.8: The original Barabara image (left) and its wavelet decomposition into depth two (right).

2.2.6 Differential invariants

Differential invariants are local features based on the derivatives of the image intensity function in case of graylevel objects or, when dealing with binary objects, on the derivatives of the object boundary.

In the case of binary objects with a smooth boundary, the invariants are calculated for each boundary point as functions of the boundary derivatives from order two to eight. In that way they map the boundary onto a so-called *signature curve* which is an invariant to affine or even projective transformation. The signature curve depends at any given point only on the shape of the boundary in its immediate vicinity. Any local change or deformation of the object has only a local impact on the signature curve. This property of locality, along with the invariance, makes them a seemingly perfect tool for recognition of partially occluded objects. Recognition under occlusion can be performed by substring matching in the space of the signatures. Differential invariants of this kind were discovered by Wilczynski [42], who proposed to use derivatives up to the eighth order. Weiss introduced differential invariants to the computer vision community. He published a series of papers [43, 44, 45] on various invariants of orders from four to six. Although differential invariants may seem to be promising from a theoretical point of view, they have been experimentally proven to be extremely sensitive to inaccurate segmentation, sampling errors, and noise.

Such high vulnerability of "pure" differential invariants led to development of *semi-differential* invariants. This method divides the object into affine-invariant parts. Each part is described by some kind of global invariants, and the whole object is then characterized by a string of vectors of invariants. Recognition under occlusion is again performed by maximum substring matching. Since inflection points of the boundary are invariant to affine (and even projective) deformation of the shape, they have become a popular tool for the definition of the affine-invariant parts. This approach was used by Ibrahim and Cohen [46], who described the object by the area ratios of two neighboring parts. Horacek et al. [47] used the same division of the object by means of inflection points but described the cuts by affine-invariant modification of the shape vector. Other similar variants can be found in [48] and [49].

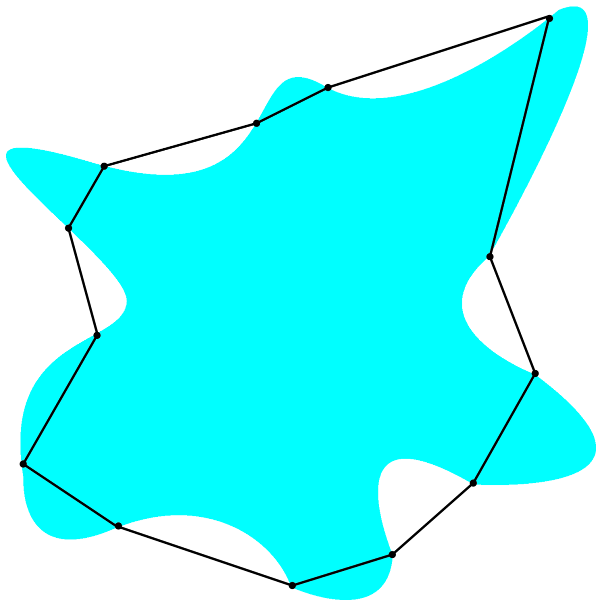


Figure 2.9: Semi-differential invariants. The object is divided by inflection points. Both convex and concave cuts can be used for a description by global invariants.

As a modification which does not use inflection points, concave residua of a convex hull can be used. For polygon-like shapes, however, inflection points cannot be used since they do not exist. Instead, one can construct the cuts defined by three or four neighboring vertices. Yang and Cohen [50] and Flusser [51] used the area ratios of the cuts to construct affine invariants. A similar method was successfully tested for perspective projection by Rothwell et al. [52].

A slightly different approach, but still belonging to semi-differential invariants, is the *curvature scale space* (CSS) proposed in [53]. It detects inflection points of the shape and investigates the movement trajectory and the "lifetime" of each inflection point under progressive smoothing of the boundary by a Gaussian filter; see Figure 2.9. Under smoothing, neighboring inflection points converge to each other and then vanish. The number of inflection points on each smoothing level, the speed of their convergence and the number of smoothings that each inflection point survives are encoded into a curve which characterizes the object (although not completely). This curve can be made rotationally (and even affine) invariant and for objects with a number of inflection points provides enough discriminability.

Local differential invariants exist also for graylevel and color images. Probably the most famous representative is the *scale-invariant feature transformation* (SIFT) [54]. This method detects dominant points in the image at different scales, looking for the extrema of the scale space created by means of the *difference of Gaussians* (DoG) and describes their neighborhood by means of orientation histograms. The descriptors of individual dominant points are then concatenated into a long feature vector, which is invariant to image translation, scaling, and rotation, and robust to illumination changes. SIFT method has inspired numerous authors to improvements, modifications, and developing of other methods of the same nature, among which the *histogram of oriented gradients* (HOG) [55] and *speeded-up robust features* (SURF) [56] belong to the most popular ones.

2.2.7 Point set invariants

Invariants of point sets form a relatively narrow, specialized category. They were developed for binary objects with polygonal boundary (i.e., for objects where the differential invariants are not well defined). The vertices of the polygon fully describe the complete shape, so it is meaningful to use them for feature generation. Point set invariants were usually constructed to be invariant w.r.t. projective transformation and also to the vertex labeling order. They are defined either as products or ratios of the distances between two points or as products or ratios of various triangle areas which are formed by point triplets [57, 58, 59, 60]. The simplest projective invariant of this kind – the *cross ratio* – was known already to ancient mathematicians Euclid and Pappus.

2.2.8 Moment invariants

Moment invariants are special functions of *image moments.* Moments are scalar quantities, which have been used for more than hundred years to characterize a function and to capture its significant features. They have been widely used in statistics for description of the shape of a probability density function and in classic rigid-body mechanics to measure the mass distribution of a body. From the mathematical point of view, moments are "projections" of a function onto a polynomial basis (similarly, Fourier transformation is a projection onto a basis of harmonic functions). Numerous polynomial bases have been used to construct moments. The standard power basis $\{x^p y^q\}$ leads to geometric moments; various orthogonal bases lead to orthogonal moments. Although all polynomial bases are theoretically equivalent, it makes sense to use different bases (and different moments) in different situations, either for the sake of simplicity or because some bases provide better numerical properties than the others.

The history of moment invariants began in the 19th century, many years before the appearance of the first computers, under the framework of the group theory and of the theory of algebraic invariants. The theory of algebraic invariants was thoroughly studied by famous German mathematicians P. A. Gordan and D. Hilbert [61] and was further developed in the 20th century in [62, 63] and [64], among others.

Moment invariants were first introduced to the pattern recognition and image processing community in 1962, when Hu employed the results of the theory of algebraic invariants and derived his seven famous invariants to rotation of 2-D objects [65]. Since that time, thousands of papers[8] have been devoted to various improvements,

[8]According to [66], about 18,000 research papers relevant to moments and/or moment invariants in image analysis have appeared in SCOPUS.

extensions and generalizations of moment invariants and also to their use in many areas of applications. Moment invariants were extensively reviewed in four monographs. The first one by Mukundan [67], published as early as in 1998, covered only selected subtopics. The second one by Pawlak [68] is focused mainly on computational aspects of moments and detailed error analysis in the discrete domain rather than on their invariant properties. Various moment invariants in 2D have been the main topics of the monograph by Flusser et al. [69]. The most recent book edited by Papakostas [66] reflects the latest development on the field.

Moment invariants have become important and frequently used shape descriptors. Nowadays, they exist for 2D as well as for 3D objects with a possible extension into arbitrary dimensions in some cases. We have moment invariants for binary, gray-level, color, and even vector-valued images. Concerning the type of invariance, there exist moment invariants to similarity and affine object transformations as well as invariants w.r.t. image blurring with certain types of the blurring filters.

Even though moment invariants suffer from certain intrinsic limitations (the worst of which is their globalness, which prevents a direct utilization for occluded object recognition), they frequently serve as the "first-choice descriptors" and as the reference method for evaluating the performance of other shape features.

2.3 Classifiers

A classifier is an algorithm which "recognizes" objects by means of their representations in the feature space and assigns each object to one of the pre-defined classes $\omega_1, \ldots, \omega_C$. The input of a classifier is the feature vector $(I_1(f), \ldots, I_n(f))^T$ of an unknown object f, and the output is the label of the class the object has been assigned to. The classifiers are totally independent of what the original objects actually look like and also of the particular kind of features I_k.

In a *supervised classification*, the classes are specified beforehand by means of a *training set*. The training set is a set of objects the class membership of which is supposed to be known. It should contain enough typical representatives of each class including intra-class variations. Selection of the training set is the role of a human expert in the application domain. This is not a mathematical/computer science problem and cannot be resolved by any automatic algorithm because it requires a deep knowledge of the application area. Since the training set is selected manually, it often contains misclassified objects. This is caused not only by human mistakes but also because some objects may be so deformed or of such non-typical appearance that the expert may be in doubt which class they should be assigned to.

In an *unsupervised classification*, the training set is not available; all objects are of unknown classification at the beginning. Sometimes even the number of classes is unknown. The classes are formed iteratively or hierarchically during the classification process such that they create compact clusters in the feature space. Since the unsupervised classification is rarely used in visual object recognition, we mention this approach for the sake of completeness but do not go into details.

Constructing a classifier is equivalent to a partition of the feature space (see Fig. 2.10). Each partition component is assigned to certain class (multiple partition components can be assigned to the same class but not vice versa), and the unknown

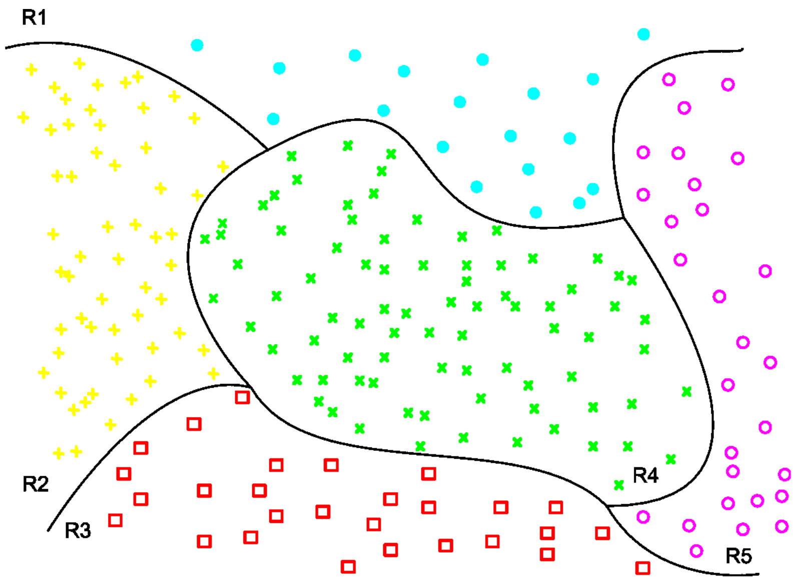


Figure 2.10: Partition of the feature space defines a classifier.

object f is classified according to the position of $(I_1(f), \ldots, I_n(f))^T$. Hence, constructing a classifier means defining the boundaries of the partition. The partition boundaries are set up by means of the training set. This stage is called *training* or *learning* of the classifier, and it is the most difficult and challenging part of the classifier design and usage. The main challenge follows from the fact that the training set is always of a limited size (sometimes of a very small one), but we try to set up the classification rules, which should work well on a high number of unknown objects. So, it is a kind of generalization or extrapolation from a small sample set, where the result is always uncertain. As soon as the classifier has been trained, the rest (i.e., the recognition of unknown objects) is a routine task.

Theoretically, if all the assumptions were perfectly fulfilled, the training should be very simple. Assuming that $I_1, \ldots, I_n$ are invariant and discriminatory, then *all* members of each class are mapped into a single point in the feature space irrespective of the intra-class variations and the points representing different classes are distinct. Then a one-element training set per class would be sufficient, and the classifier would just check which training sample the incoming object coincides in the feature space. This model is, however, very far from reality. The features I_k, although theoretically invariant, may not provide a perfect invariance within the classes simply because the intra-class variations are broader than they have been expected to be. Moreover, in all stages of object recognition we encounter errors of various kinds, which all contribute to the violation of the theoretical assumptions. In such a case, the training samples of the same class may fall into distinct points in the feature space.

In the literature, the classifiers are sometimes formalized by means of *decision functions*. Each class ω_i is assigned to its real-valued decision function g_i which is defined in the feature space. Then an object f represented by a feature point $\mathbf{a} = (I_1(f), \ldots, I_n(f))^T$ is classified into class ω_k if and only if $g_k(\mathbf{a}) > g_i(\mathbf{a})$ for all

$i \neq k$. This concept is equivalent to the previous one. If the decision functions have been given, then the partition boundaries can be found as solutions of the equations $g_j(\mathbf{a}) = g_i(\mathbf{a})$. On the other hand, given the partition, we can easily construct the decision functions as characteristic functions of the partition components.

There have been many approaches to the classifier training and to how the partition is set up. In some classifiers, the partition boundaries are defined explicitly, and their parameters are found via training; some others do not use any parametric curves/surfaces, and the partition is defined implicitly or by means of another quantity. For the same training set, we can construct infinitely many classifiers (by means of different training algorithms) which will most probably differ one another by its performance on independent data. If we use an over-simplified model of the partition, then we only roughly approximate the "ground-truth" partition, and the number of misclassified objects is high even inside the training set. If, on the other hand, we allow relatively complex boundaries and minimize the classification error on the training set, we end up with an *over-trained* classifier, which is optimal on the training set, but there is no guarantee of its optimality on independent data (more precisely, it is highly probable that such classifier will perform poorly). One feels that the "optimal" classifier should be somewhere in between (see Fig. 2.11).

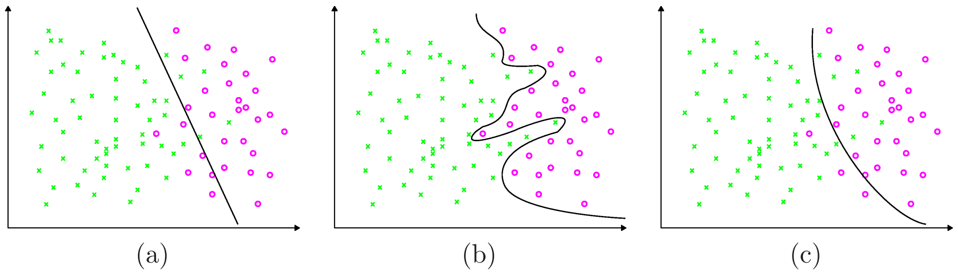


Figure 2.11: Three different classifiers as the results of three training algorithms on the same training set. (a) Over-simplified classifier, (b) over-trained classifier, and (c) close-to-optimal classifier.

In the rest of the section we briefly review the basic classifier types. For details and for other classifiers, we refer to the specialized monographs [70, 71].

2.3.1 Nearest-neighbor classifiers

The *nearest-neighbor* (NN) classifier, sometimes also called the *minimum distance* classifier, is the most intuitive classifier. Vector $\mathbf{a}$ is assigned to the class which minimizes the distance $\varrho(\mathbf{a}, \omega_i)$ between $\mathbf{a}$ and respective classes. The feature metric space with the metric d offers a variety of possibilities of how the distance ϱ between a point and a set (class) can be defined. The one which is common in metric space theory

$$\varrho_1(\mathbf{a}, \omega) = \min_{\mathbf{b} \in \omega} d(\mathbf{a}, \mathbf{b})$$

can of course be used, but it is very sensitive to outliers in the training set. To avoid this sensitivity, the "mean distance"

$$\varrho_2(\mathbf{a}, \omega) = d(\mathbf{a}, \mathbf{m}_\omega),$$

where $\mathbf{m}_\omega$ is the centroid of the class ω, is sometimes used instead. Using the mean distance is also justified by a known paradigm from statistics, which says that no algorithm working with real data should use extreme points or values, because they tend to be unstable and most often influenced by measurement errors and noise. On the other hand, ϱ_1 reflects the size and the shape of the classes in a better way than ϱ_2 (see Fig. 2.12).

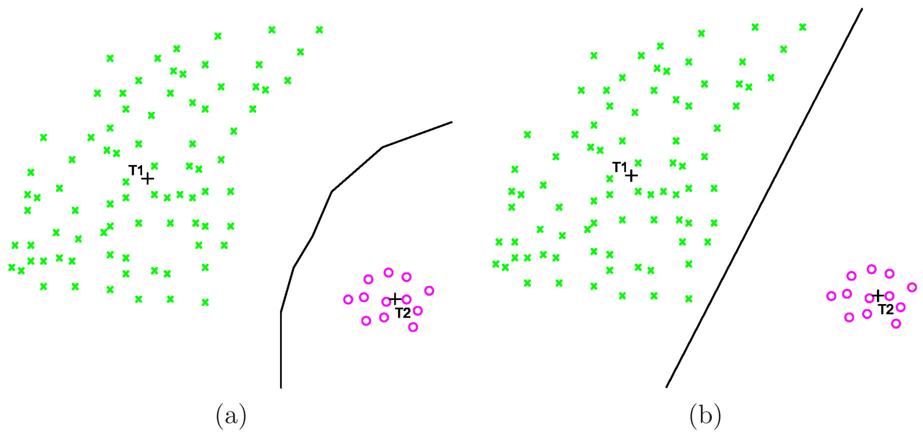


Figure 2.12: Decision boundary of the NN classifier depends on the used distance: (a) the nearest distance ϱ_1 and (b) the mean distance ϱ_2.

If each class is represented by a single training sample, then of course $\varrho_1 = \varrho_2$. In the case of two classes, this is the simplest situation ever, which leads to a linear decision boundary. In the case of more classes, the feature space partition is given by Voronoi tessellation, where each training sample is a seed point of a Voronoi cell that corresponds to the respective class.

In case of multiple training samples in each class, there is of course a difference between the NN classifiers which use ϱ_1 and ϱ_2 distances. While ϱ_2 again leads to a Voronoi tessellation the seeds of which are the class centers, the use of ϱ_1 generates complex curved decision boundaries.

The sensitivity of the NN classifier with ϱ_1 to outliers can be suppressed by introducing a more general k-NN classifier, with k being a (usually small) integer user-defined parameter (see Fig. 2.13). We can find two versions of this algorithm in the literature.

Version 1

1. Unknown feature vector $\mathbf{a}$ is given.

2. Find k training samples which are the closest (in the sense of metric d) to sample $\mathbf{a}$.

3. Check for each class how many times it is represented among the k samples found in Step 1.

4. Assign $\mathbf{a}$ to the class with the maximum frequency.

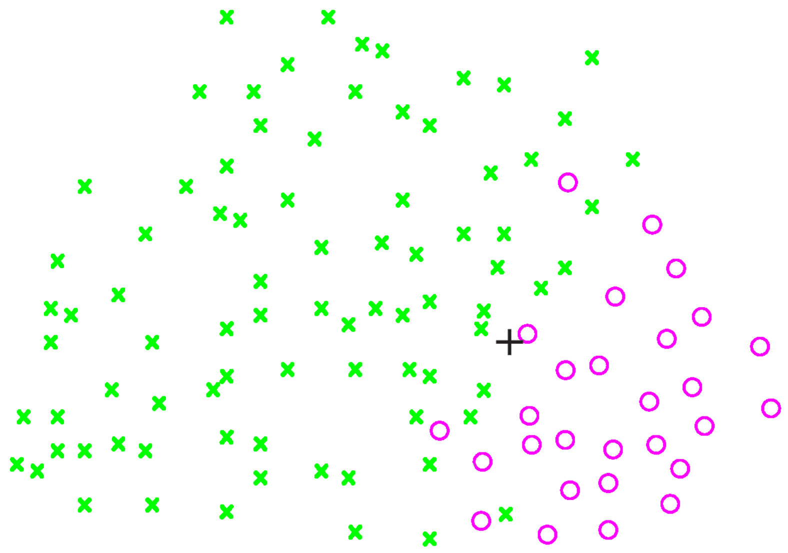


Figure 2.13: Robustness of the k-NN classifier. The unknown sample "+" is classified as a circle by the NN classifier and as a cross by the 2-NN classifier. The later choice better corresponds to our intuition.

Version 2

1. Unknown feature vector $\mathbf{a}$ is given.
2. Denote the number of the closest training samples from ω_i as k_i. Set $k_i = 0$ for $i = 1, 2, \ldots, C$.
3. Until $k_j = k$ for some j do

   Find training sample $\mathbf{c}$ which is the closest (in the sense of metric d) to sample $\mathbf{a}$.

   If $\mathbf{c} \in \omega_i$, then set $k_i = k_i + 1$.

   Exclude $\mathbf{c}$ from the training set.
4. Assign $\mathbf{a}$ to the class ω_j.

Version 1 is simpler because we need to find just k closest neighbors, but it may not provide an unambiguous decision. Version 2 requires finding between k and $C(k-1)+1$ closest neighbors (their exact number cannot be predicted) but mostly yields a decision[9]. Although the meaning of k is slightly different in both versions, its choice is always a heuristic often done on a trial and error basis. Small k provides less robustness to outliers, but the classification is faster. The corresponding decision boundary may be curved and complex, and the classifier tends to get overtrained. For $k = 1$, we obtain a plain NN classifier. When k is higher, the decision boundary is

[9]The decision still may not be unique because there may exist multiple minima, but this ambiguity is rare comparing to the ambiguity of Version 1.

smooth and less distinct. More training samples may be classified incorrectly (which says nothing about the overall quality of the classifier). Anyway, the upper limit on k is that it should be by an order smaller than the number of the training samples in each class. A typical choice of k in practice is from 2 to 10, depending on the size and reliability of the training set.

The k-NN classifier is relatively time expensive, especially for high k. Several efficient implementations can be found in the literature, and some are even available on the internet.

2.3.2 Support vector machines

Classifiers called the *Support vector machines* (SVMs) are generalizations of a classical notion of linear classifiers. The SVMs were invented by Vapnik [72]. In the training stage, the SVM classifier looks for two parallel hyperplanes which separate the training samples such that the distance (which is called *margin*) between these hyperplanes is maximized. The decision boundary is then another hyperplane parallel with these two and lying in the middle between them (see Fig. 2.14). The margin hyperplanes always pass through some training samples; these samples are called *support vectors*. The support vectors can lie on the convex hull of the training set only. The SVM classifier satisfying this constraint is called the *hard margin* SVM.

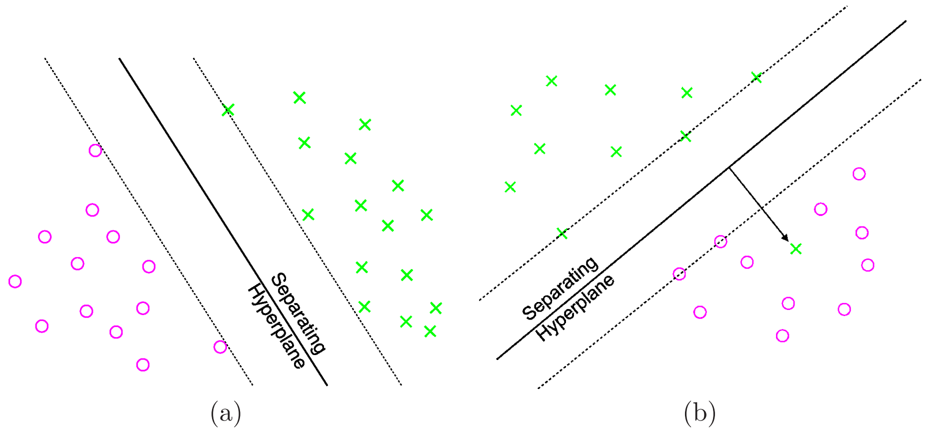


Figure 2.14: SVM classifiers. The hard margin (left) and the soft margin (right) constraints.

Training of the SVM is a constrained optimization problem, where we minimize the norm of the hyperplane normal vector subject to the condition that the margin hyperplanes separate the data. This can be efficiently solved by quadratic programming. The hard margin constraint is applicable only to linearly separable classes with almost error-free samples. If there are some outliers in the training set, we can apply the *soft margin* constraint proposed by Cortes [73]. Soft margin constraint allows some training samples to lie inside the margin or even on its other side. It again leads to a constrained optimization problem, but the objective function contains an additional penalty term. For the difference between the hard- and soft-margin classifiers, see Figure 2.14.

Boser et al. [74] generalized the idea of the maximum margin hyperplanes to linearly non-separable classes. They proposed a mapping of the current feature space into a new one, in which the classes are linearly separable. This mapping is defined by means of various radial basis functions and is known as the *kernel trick*.

The SVMs have become very popular namely because their relatively fast training. They provide a good trade-off between speed and accuracy, namely for mid-sized training sets (although they can be applied to large training sets as well). However, they also exhibit several potential drawbacks. The hard margin version ignores the distribution of the samples inside the training set; it considers only the samples on the convex hull. Unlike probabilistic classifiers, the SVM decision is deterministic and does not take into account prior probability of individual classes. SVMs were originally proposed for a two-class (dichotomic) problem. Generalization to more classes is commonly done by decomposition into several dichotomies [75], which may be time-expensive if C is large.

2.3.3 Neural network classifiers

Artificial neural networks (ANNs) are "biologically inspired" classifiers. The original idea was to employ a massive parallelism offered by the most powerful computers and to construct a "network" of many "neurons", which can perform simple operations in parallel. These neurons are organized into layers. The first (input) layer contains as many neurons as is the dimensionality of the feature vector. It reads the object features, makes a simple operation(s) with them, and sends these results to the second layer. This process is repeated until the last (output) layer has been reached; the output layer provides a decision about the class membership. The layers between the input and output layers are called *hidden* layers. The neurons of the same layer do not communicate among themselves, so they can actually work in parallel.

The ANNs were firstly used for classification purposes probably by Rosenblatt [76], who proposed the first single-layer *perceptron*. It was, in fact, linear dichotomic classifier. Each neuron was assigned a weight, and the perceptron calculated the linear combination of the features (possibly with an offset) weighted by these weights. The set where this linear combination was zero, defined a decision hyperplane. So, the perceptron only checked the sign of the output and classified the object according to this sign. The training of the perceptron consists of setting up the neuron weights such that the training set (provided it is linearly separable) is classified correctly. The training algorithm is iterative and upgrades the weights after the processing of each training sample.

Later, the multilayer ANNs and especially the popular *radial basis function* (RBF), networks broke the limitation to linearly separable tasks. The RBF networks have at least one hidden layer, where the features are used as arguments of certain RBFs. So, the evaluation of the RBF is used instead of doing linear combinations. The RBFs are popular, smooth, and "infinitely flexible" functions known from approximation theory (multidimensional thin-plate splines and Gaussian functions are examples of the RBFs). When used in the ANN classifier, they can create an arbitrary smooth decision boundary [77].

The popularity of the ANN's in the 1970s and 1980s originated from two common beliefs. The first one was that artificial intelligence algorithms should copy the "algorithms" in a human brain (even though the computer and brain architectures are different) and the second one that the future of computers is in very powerful mainframes equipped by thousands of CPUs. As we know today, the recent development of computer technologies followed a different way to the distributed computing and personal computers.

The ANN classifiers have been compared in many studies to the SVM with ambiguous results. There is a belief these two types of classifiers are more or less comparable in performance (although they use different terminology and mathematical background). We can find many parallels between them. The single-layer perceptron corresponds to the hard-margin linear SVM; the RBF networks use the same "linearization" principle as the kernel trick in SVM (both even use the same functions to create the decision boundary). The differences in performance reported in individual experiments were caused probably by the properties of the particular data rather than by general properties of these classifiers. In the 1980s, the ANN classifiers were believed to overcome a construction of small sophisticated feature sets, which is sometimes difficult. In many tasks, such as character recognition, the ANN was trained directly on images, taking each pixel value as a feature. This was, however, a dead-end road because these "features" did not provide any invariance to elementary image transformations. All the above facts implied that neural network classifiers were gradually overtaken in popularity and in the number of reported applications by support vector machines in 1990–2005. After 2005, the new concept of *deep learning* and *convolution networks* resurrected interest in the ANN's.

Deep convolution neural networks (CNN) are modern classifiers which go beyond the conventional framework of separated feature design and classifier training. Although they were firstly proposed as early as 1980 [78], they have attracted noticeable attention very recently when they repeatedly achieved an excellent performance in the *Large Scale Visual Recognition Challenge* [79].

Instead of working with features "manufactured" beforehand, the CNN generates the features by a cascade of convolutions and downsampling. The parameters of each convolution kernel are learned by a backpropagation algorithm. There are many convolution kernels in each layer, and each kernel is replicated over the entire image with the same parameters. The function of the convolution operators is to extract different features of the input. The capacity of a neural net varies, depending on the number of layers. The first convolution layers obtain the low-level features, such as edges, lines, and corners. The more layers the network has, the higher-level features it produces. The CNN's virtually skip the feature extraction step and require only basic preprocessing, which makes them, if enough computing power is available, very powerful [80].

The absence of sophisticated features defined in advance is not only an advantage but also a drawback of the CNN's. For CNN's it is very difficult (if not impossible) to generate features (by their inner layers), invariant to rotation or affine transformations (at least no such methods have been reported in the literature). In general, it is possible to reach translation and scale invariance, but probably the only way of dealing with other intraclass variabilities is a brute force approach or prior image normalization.

2.3.4 Bayesian classifier

The Bayesian classifier is a statistical classifier which considers the features to be random variables. It is suitable when the training set is large. It makes a decision by maximizing the posterior probability $p(\omega_i|\mathbf{a})$ of assigning an unknown feature vector $\mathbf{a}$ to class ω_i. This is a reasonable idea because $p(\omega_i|\mathbf{a})$ is the probability that an object whose feature vector is $\mathbf{a}$ belongs to the class ω_i. Maximizing the posterior probability assures that the probability of misclassification is minimized [70]. This is why the Bayesian classifier is sometimes called the "optimal" classifier.

Since the posterior probabilities cannot be estimated directly from the training set, we take advantage of the Bayes formula, which express the posterior probability in terms of prior probability and class-conditional probability

$$p(\omega_i|\mathbf{a}) = \frac{p(\mathbf{a}|\omega_i) \cdot p(\omega_i)}{\sum_{j=1}^{C} p(\mathbf{a}|\omega_j) \cdot p(\omega_j)}.$$

Since the denominator is constant over i, maximizing posterior probability is equivalent to maximizing the numerator.

The prior probability $p(\omega_i)$ express the relative frequency with which the class ω_i appears in reality. In other words, $p(\omega_i)$ is a probability that the next object, which we have not seen/measured yet, will be from the class ω_i. The priors are unknown and must be estimated during the classifier training. There are basically three approaches how to estimate the priors. The best possibility is to use prior knowledge, which might be available from previous studies. In medical diagnostics, for instance, we know the incidence of certain diseases in the population. In character recognition, the probabilities of the frequency of occurrence of each character is well known from extensive linguistic studies (in English, for instance, the most probable letter is E with $p(E) = 0.13$ while the least probable is Z with $p(Z) = 0.0007$ [81]).

If such information is not available, we may estimate $p(\omega_i)$ by the relative frequency of occurrence of ω_i in the training set. This is, however, possible only if the training set is large enough and has been selected in accordance with reality. If it is not the case and no prior knowledge is available, we set $p(\omega_i) = 1/C$ for any i.

The class-conditional probability $p(\mathbf{a}|\omega_i)$, sometimes referred to as the *likelihood* of the class ω_i, is a probability of occurrence of point $\mathbf{a}$ in ω_i. It is given by the respective probability density function (pdf), which is to be estimated from the training data independently for each class. This estimation is usually a parametric one, where a particular form of the pdf is assumed and we estimate only their parameters. Although any parametric pdf can be employed, the Gaussian pdf is used most frequently. Hence, we assume the n-dimensional pdf in the form

$$p(\mathbf{a}|\omega_i) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_i|}} \exp\left(-\frac{1}{2}(\mathbf{a}-\mathbf{m}_i)^T \Sigma_i^{-1} (\mathbf{a}-\mathbf{m}_i)\right), \tag{2.1}$$

where mean vector $\mathbf{m}_i$ and covariance matrix Σ_i are estimated by a sample mean and a sample covariance matrix from the training data[10]. In this way, the classifier no longer works with the individual training samples, it works only with the pdf

[10]In some cases when the features are independent, we can assume all Σ_i being diagonal. Such classifier is called the *naive* Bayesian classifier.

parameters. In the case of equal priors, the classifier maximizes the likelihood. The decision boundary in the feature space between two classes is given by the equation

$$p(\mathbf{a}|\omega_i) = p(\mathbf{a}|\omega_j),$$

which always leads to a hyperquadric (a conic if $n = 2$). If the covariance matrices are equal, $\Sigma_i = \Sigma_j$, then the decision boundary between ω_i and ω_j is a hyperplane (a line if $n = 2$), and vice versa (in this case, the joint covariance matrix Σ can be robustly estimated from the training samples from both ω_i and ω_j simultaneously). In such a case, the classifier can be understood as a minimum distance classifier which minimizes the *Mahalanobis distance*

$$\varrho_M(\mathbf{a}, \omega_i) = (\mathbf{a} - \mathbf{m}_i)^T \Sigma^{-1} (\mathbf{a} - \mathbf{m}_i).$$

If Σ is diagonal with equal variances, we end up with a standard Euclidean minimum distance classifier.

If the classes are not normally distributed, their pdf's can be estimated using other parametric models or, alternatively, by non-parametric techniques (the estimation by means of Parzen window uses to be applied, see [70] for details).

Since the Bayesian classifier requires a large number of training samples in each class, it is not convenient for face and fingerprint recognition, where typically each class is represented by a single (or very few) sample(s). On the other hand, it is widely used and performs very well in the pixel-wise classification of multispectral and hyperspectral data (see Fig. 2.15).

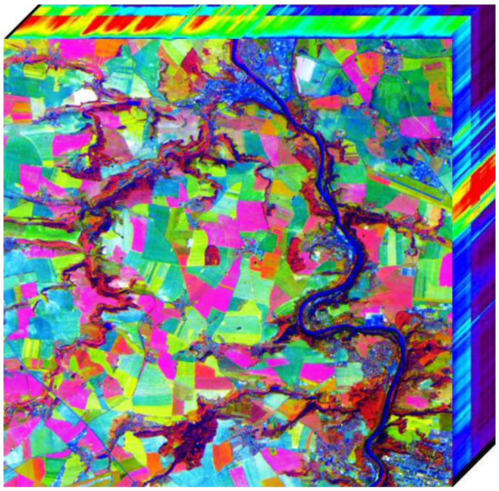


Figure 2.15: Multispectral satellite image. The objects are single pixels, the features are their intensities in the individual spectral bands. This kind of data is ideal for the Bayesian classifier.

2.3.5 Decision trees

Decision trees are simple classifiers designed particularly for logical “yes/no” features and for categorial features[11], where no “natural” metric exists. The classifier is arranged into a (usually but not necessarily binary) tree, where the unknown object enters the root and passes the tree until it reaches a leaf. In each node, a simple question is asked, and the next move is chosen according to the answer. Each leaf is associated with just one class, and the object is classified accordingly (several leaves may be associated with the same class). Decision trees can be applied to real-valued features as well. The queries have the form of simple inequalities (is the feature a_i less or greater than threshold t?) or compound inequalities, where a function of several features is compared to the threshold.

In the training phase, we must design the tree shape, choose the queries in all nodes and select the thresholds (and the compound functions) the real-valued features (if any) are to be compared with. Obviously, our goal is to “grow” a small tree with few nodes and simple decision queries. This is not a trivial task, and there have been several approaches to tree training. The most popular one relies on the concept of *purity* or *impurity* of the node. The node is called *pure* if it contains only the objects belonging to one class (clearly, the only pure nodes are the leaves); the node with a maximum impurity contains different classes with the same frequency.

The principle underlaying the tree creation is that we want to maximize the decrease of impurity between the node and the immediate descendent nodes. Hence, a proper query at the node should split the data into the parts which are as pure as possible. There have been several definitions of the impurity. Most of them are inspired by information theory, such as *entropy impurity, information impurity, and Gini impurity*, some others follow from our intuitive understanding of the problem (*variance impurity, misclassification impurity*) [70]. The chosen impurity measure is optimized in a “greedy” manner, which means a maximum drop is required in each node. It is well known that this locally optimal approach may not lead to the simplest tree but global tree optimization is not feasible due to its complexity.

2.3.6 Unsupervised classification

As we already mentioned, unsupervised classification (also called *clustering*) has in object recognition less importance than the supervised one, because in most tasks we do have a training set. Still, clustering may be applied in the preliminary stage before the training set has been selected, for instance in multispectral pixel-wise classification, to see how many potential classes there are.

Clustering is a traditional discipline of statistics and data analysis. Unlike the supervised classification, the desired number of clusters may be unknown. Intuitively, the cluster is a subset of the data, which is “compact” and “far from other clusters” in the feature space. The meaning of these terms, of course, depends on the chosen metric and may be defined in various ways.

Clustering methods can be divided into two major groups – *iterative* and *hierarchical* methods. Iterative methods typically require the number of the clusters as an

[11]Categorial features can take only few distinct values, for instance “good”, “average”, and “poor”. They are not very common in image analysis but are of great importance in medical diagnostics and social science statistics.

input parameter. In each iteration, they try to re-assign the data points such that a given quality criterion is maximized. A typical representative of iterative methods is the famous C-means clustering.

Hierarchical methods built the cluster structure "from scratch". *Agglomerative clustering* starts from the initial stage, where each data point is considered to be a cluster. On each level of the hierarchy, the two "most similar" clusters are merged together. Various criteria can be used to select the clusters to be merged. The stopping level and the final configuration are usually chosen by means of the *dendrogram*, a symbolic tree graph showing the clustering process. A complementary approach is provided by *divisive* algorithms, which on the initial level consider all data to be contained in a single cluster. In each level, the algorithm selects the cluster to be split and then divides this cluster into two parts, trying to optimize the same or similar criteria as are used in agglomerative methods. Although agglomerative and divisive methods may seem to converge to the same clustering, they are not completely "inverse" and may lead to different results. Agglomerative methods are usually faster and more frequently used, particularly if the desired/expected number of clusters is high. To learn more about clustering techniques, we refer to [71].

2.4 Performance of the classifiers

Before we proceed to the explanation of how we can increase the classifier performance, let us mention how the classifier performance should be evaluated. After that, we briefly review two popular techniques used for improving the classification – *classifier fusion* and *dimensionality reduction*.

2.4.1 Measuring the classifier performance

Before we proceed to the explanation of how to increase the classifier performance, let us mention how the classifier performance should be evaluated. Intuitively, one may expect that the classifier performance could be measured by a single scalar indicator called the *success rate*, which is a ratio of the number of successfully recognized objects to all trials (it is often presented as a percentage). The success rate, however, shows the quality of the classifier in a limited way, and sometimes it is even completely misleading. Consider a medical prophylactical screening for a disease whose statistical frequency in the population is 0.001. If any tested person was automatically "classified" as a healthy person, then the classifier would reach an excellent success rate 0.999 but such classifier is totally useless since it does not detect any disease occurrence. Hence, the success rate can be used only if the relative frequency of all classes is the same, but that is not the only limitation.

Let us again imagine a medical test for whether or not the test person has a cancer. Since only suspected persons pass this test, the relative frequency of the cancer occurrence can be considered 0.5. Still, the success rate itself does not say anything about the quality of the test because the two possible misclassifications have completely different weights. While the "false positive" results will be later disproved by other tests (which may be expensive and unpleasant for the patient but does not do any harm), the "false negative" result leads to a delay in the treatment and may be fatal for the patient. The success rate does not distinguish these two cases.

The two above examples illustrate why the classifier performance should be reported by a full *confusion table*, which is a $C \times C$ matrix with the ij-th element being the number of the ω_i members which have been assigned to ω_j.

The confusion table provides a very good insight into the quality of the classifier if it has been evaluated on a representative set of objects. It is seriously misleading to use the training set for this purpose. The error estimates would be highly optimistic, and if the confusion table was diagonal, we could not conclude that the classifier is perfect, but it rather shows it is heavily overtrained. The correct way is to evaluate the classifier on the *test set*, which should be selected in the same way as the training set but has not been used for the training. If the user has not provided us with the separate test set, we can just leave out 10% of the training set and use it for the evaluation. By repeating this evaluation several times with randomly selected 10% of the training samples and calculating the means and standard deviations, we obtain a very reliable and informative confusion table.

2.4.2 Fusing classifiers

Fusing or combining the results of different classifiers together in order to reach better performance is a modern trend in classifier design. This idea was proposed by Kittler et al. [82], who presented several fusion schemes – algorithms, how to combine the decisions of several classifiers together. By the term "different classifiers" we understand either two classifiers of the same nature but being trained on different training sets and/or in different feature spaces, or two classifiers of different nature trained on the same or different data. For deterministic classifiers such as the k-NN and SVM, there is not much freedom in constructing the fusion rule. The absolute majority and the simple majority vote are probably the only reasonable choices. In case of probabilistic classifiers, such as Bayesian classifier, the individual probabilities of the assignments are fused instead of fusing final decisions. Then we maximize the product, the sum, the median or the maximum of individual posterior probabilities.

The classifier fusion has been successfully used in handwritten character recognition [83, 84], in medical diagnostics [85, 86], and in many other problems. It is, however, true that classifier fusion does not always guarantee better accuracy than the best-performing individual classifier. The best chance of improvement is if the input classifiers have a comparable (and not very high) accuracy. The fusion rules may be not only fixed but may be also adaptive w.r.t. the dataset, which improves the performance particularly if more classifiers and hierarchical fusion schemes are used [87]. For an exhaustive overview of classifier fusion techniques we refer to the recent monograph [88].

2.4.3 Reduction of the feature space dimensionality

We should keep the dimensionality of the feature space as low as possible. The reason for that is twofold – working in fewer dimensions is faster, and the classifier might be even more accurate than in higher dimensions. While the former proposition is evident, the latter one might be a bit surprising because one could intuitively expect that the more features, the better classification results. To explain this paradox, let us imagine we are using a Bayesian classifier with normally distributed classes. In n dimensions, we have to estimate about $n^2/2$ parameters in each class. If we double the dimensionality, then the number of the unknown parameters increases four times but the number of the training samples is still the same. Hence, increasing the dimensionality yields less accurate parameter estimation. If we consider, in addition to that, that the features are often correlated, the need for dimensionality reduction is straightforward.

There are basically two goals (criteria) of the dimensionality reduction, which determine the used approaches. The first goal is to remove the correlation (or other dependencies) between the features globally, regardless of whether or not any classes have been specified. The most famous algorithm of this kind is the *principal component transformation* (PCT)[12]. The joint covariance matrix, estimated from all available data, is diagonalized. This is always possible because the covariance matrix is symmetric; their eigenvectors are orthogonal and create a new coordinate system in which the features are uncorrelated. The transformation from the old to the new coordinates is a rotation. Then, the features are sorted according to their variance, and p features called *principal components* with the highest variance are kept while the others are removed from the system. Hence, the PCT creates new synthetic features, which are linear combinations of the original ones and are not correlated (see Fig. 2.16). It should be, however, noted that the PCT suppresses only the linear component of the dependency between the features and cannot identify and handle higher-order dependencies.

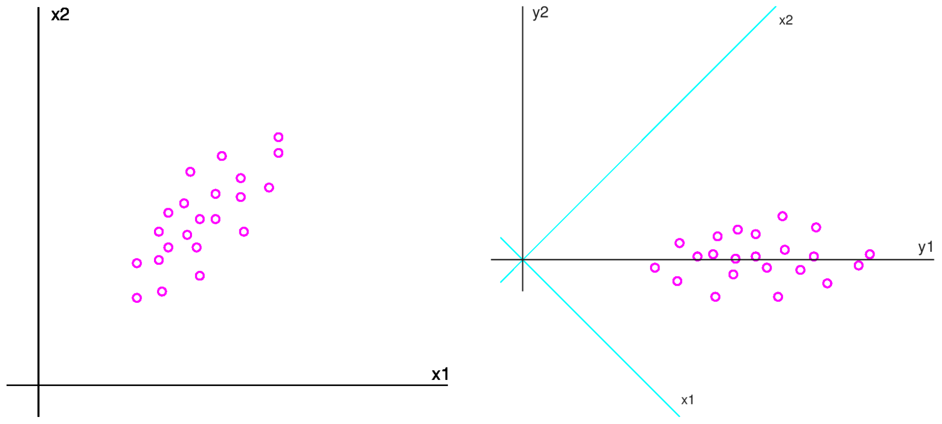


Figure 2.16: Principal component transformation: (a) unstructured original data in (X_1, X_2) feature space, correlated (b) transformation into new feature space (Y_1, Y_2), decorrelated. The first principal component is Y_1.

The PCT is an efficient tool suitable for many purposes, such as for multichannel image compression, but its application to classification tasks is limited. As we have

[12]The term *principal component analysis* (PCA) has been equivalently used in the literature for this technique.

seen, the PCT evaluates the "quality" of the features solely according to their variance, which may be different from their actual discrimination power (see Fig. 2.17).

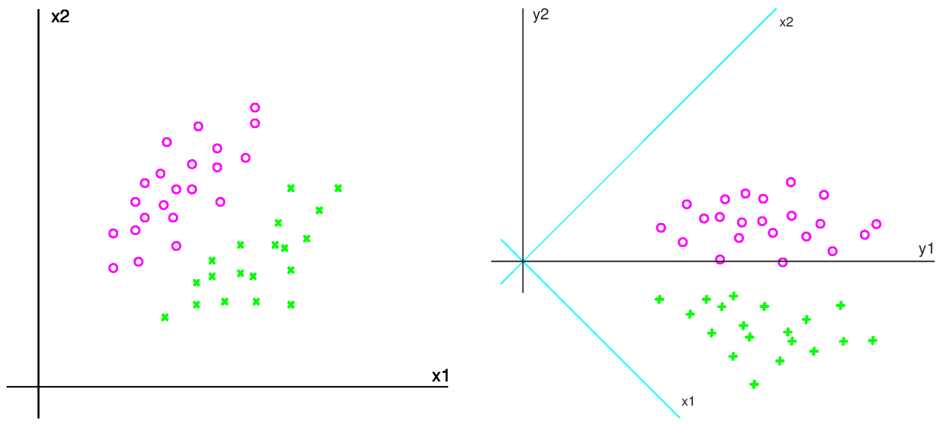


Figure 2.17: PCT of data consisting of two classes: (a) the original (X_1, X_2) feature space, (b) new feature space (Y_1, Y_2) after the PCT. The first principal component is Y_1 thanks to higher variance but the between-class separability is provided solely by Y_2.

Another criterion is to select such features, which maximize the between-class separability on the training set. There are two conceptual differences from the PCT – the new features are selected among the old ones (no features are artificially created) and the labeled training set is required. The separability can be measured in various ways. For normally distributed classes, the simplest separability measure is the *Mahalanobis distance* between two classes

$$s_M(\omega_i, \omega_j) = (\mathbf{m}_i - \mathbf{m}_j)^T (\Sigma_i + \Sigma_j)^{-1} (\mathbf{m}_i - \mathbf{m}_j).$$

More sophisticated (but not necessarily better) separability measures can be derived from between-class ant within-class scatter matrices [89], trying to select the feature subspace such that the classes are "compact" and "distant" from one another. In this way, the feature selection problem has been reformulated as a maximization of the chosen separability measure. To resolve it, several optimization algorithms have been proposed, both optimal [90] as well as suboptimal [91, 92, 93] ones.

The advantage of the latter approach is that it considers the between-class separability, preserves the original meaning of the features, and makes the choice on the training set before the unknown objects are actually measured, so the features which have not been selected can be excluded from the measurements.

Most recently, the *wrappers* feature selection methods have appeared [94]. They do not maximize any separability measure but directly optimize the performance of the classifier on a test set. Since this approach requires training and evaluating the classifier repeatedly many times, it is significantly slower than the previous one, but it may be better for the given classifier.

2.5 Conclusion

This chapter presented a glance at the whole object recognition process. Our aim was not to go into particular details but rather show briefly the context in which moment invariants appear, their role in object description, and what we expect from them. In the next chapters we study moments and moment invariants in detail.

References

[1] D. Kundur and D. Hatzinakos, "Blind image deconvolution," *IEEE Signal Processing Magazine*, vol. 13, no. 3, pp. 43–64, 1996.

[2] B. Zitová and J. Flusser, "Image registration methods: A survey," *Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[3] S. Lončarić, "A survey of shape analysis techniques," *Pattern Recognition*, vol. 31, no. 8, pp. 983–1001, 1998.

[4] D. Zhang and G. Lu, "Review of shape representation and description techniques," *Pattern Recognition*, vol. 37, no. 1, pp. 1–19, 2004.

[5] F. B. Neal and J. C. Russ, *Measuring Shape.* CRC Pres, 2012.

[6] I. T. Young, J. E. Walker, and J. E. Bowie, "An analysis technique for biological shape. i," *Information and Control*, vol. 25, no. 4, pp. 357–370, 1974.

[7] J. Žunić, K. Hirota, and P. L. Rosin, "A Hu moment invariant as a shape circularity measure," *Pattern Recognition*, vol. 43, no. 1, pp. 47–57, 2010.

[8] P. L. Rosin and J. Žunić, "2D shape measures for computer vision," in *Handbook of applied algorithms : solving scientific, engineering and practical problems* (A. Nayak and I. Stojmenovic, eds.), pp. 347–372, Wiley, 2008.

[9] P. L. Rosin and C. L. Mumford, "A symmetric convexity measure," *Artificial Intelligence*, vol. 103, no. 2, pp. 101–111, 2006.

[10] P. L. Rosin and J. Žunić, "Measuring rectilinearity," *Computer Vision and Image Understanding*, vol. 99, no. 2, pp. 175–188, 2005.

[11] P. L. Rosin, "Measuring sigmoidality," *Pattern Recognition*, vol. 37, no. 8, pp. 1735–1744, 2004.

[12] J. Žunić and P. L. Rosin, "A new convexity measure for polygons," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 7, pp. 923–934, 2004.

[13] T. Peli, "An algorithm for recognition and localization of rotated and scaled objects," *Proceedings of the IEEE*, vol. 69, no. 4, pp. 483–485, 1981.

[14] A. Goshtasby, "Description and discrimination of planar shapes using shape matrices," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, no. 6, pp. 738–743, 1985.

[15] A. Taza and C. Y. Suen, "Description of planar shapes using shape matrices," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1281–1289, 1989.

[16] H. Freeman, "On the encoding of arbitrary geometric configurations," *IRE Transactions on Electronic Computers*, vol. 10, no. 2, pp. 260–268, 1961.

[17] H. Freeman, "Computer processing of line-drawing images," *ACM Computing Surveys*, vol. 6, no. 1, pp. 57–97, 1974.

[18] H. Freeman, "Lines, curves, and the characterization of shape," in *International Federation for Information Processing (IFIP) Congress* (S. H. Lavington, ed.), pp. 629–639, Elsevier, 1980.

[19] C. T. Zahn and R. Z. Roskies, "Fourier descriptors for plane closed curves," *IEEE Transactions on Computers*, vol. C-21, no. 3, pp. 269–281, 1972.

[20] C. C. Lin and R. Chellapa, "Classification of partial 2-D shapes using Fourier descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 686–690, 1987.

[21] K. Arbter, W. E. Snyder, H. Burkhardt, and G. Hirzinger, "Application of affine-invariant Fourier descriptors to recognition of 3-D objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 640–647, 1990.

[22] J. Zhang and T. Tan, "Brief review of invariant texture analysis methods," *Pattern Recognition*, vol. 35, no. 3, pp. 735–747, 2002.

[23] M. Petrou and P. G. Sevilla, *Image Processing: Dealing with Texture.* Wiley, 2006.

[24] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 3, no. 6, pp. 610–621, 1973.

[25] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on feature distributions," *Pattern Recognition*, vol. 19, no. 3, pp. 51–59, 1996.

[26] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[27] J. Fehr and H. Burkhardt, "3D rotation invariant local binary patterns," in *19th International Conference on Pattern Recognition ICPR'08*, pp. 1–4, IEEE, 2008.

[28] K. J. Dana, B. van Ginneken, S. K. Nayar, and J. J. Koenderink, "Reflectance and texture of real-world surfaces," *ACM Transactions on Graphics*, vol. 18, no. 1, pp. 1–34, 1999.

[29] M. Haindl and J. Filip, *Visual Texture.* Advances in Computer Vision and Pattern Recognition, London, UK: Springer-Verlag, 2013.

[30] J. Filip and M. Haindl, "Bidirectional texture function modeling: A state of the art survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 11, pp. 1921–1940, 2009.

[31] S. Mallat, *A Wavelet Tour of Signal Processing.* Academic Press, 3rd ed., 2008.

[32] G. C.-H. Chuang and C.-C. J. Kuo, "Wavelet descriptor of planar curves: theory and applications," *IEEE Transactions on Image Processing*, vol. 5, no. 1, pp. 56–70, 1996.

[33] P. Wunsch and A. F. Laine, "Wavelet descriptors for multiresolution recognition of handprinted characters," *Pattern Recognition*, vol. 28, no. 8, pp. 1237–1249, 1995.

[34] Q. M. Tieng and W. W. Boles, "An application of wavelet-based affine-invariant representation," *Pattern Recognition Letters*, vol. 16, no. 12, pp. 1287–1296, 1995.

[35] M. Khalil and M. Bayeoumi, "A dyadic wavelet affine invariant function for 2D shape recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 10, pp. 1152–1163, 2001.

[36] S. Osowski and D. D. Nghia, "Fourier and wavelet descriptors for shape recognition using neural networks - a comparative study," *Pattern Recognition*, vol. 35, no. 9, pp. 1949–1957, 2002.

[37] G. V. de Wouwer, P. Scheunders, S. Livens, and D. V. Dyck, "Wavelet correlation signatures for color texture characterization," *Pattern Recognition*, vol. 32, no. 3, pp. 443–451, 1999.

[38] A. Laine and J. Fan, "Texture classification by wavelet packet signatures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, pp. 1186–1191, 1993.

[39] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 1, pp. 674–693, 1989.

[40] K. Huang and S. Aviyente, "Wavelet feature selection for image classification," *IEEE Transactions on Image Processing*, vol. 17, no. 9, pp. 1709–1720, 2008.

[41] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International Journal of Computer Vision*, vol. 38, no. 1, pp. 15–33, 2000.

[42] E. Wilczynski, *Projective Differential Geometry of Curves and Ruled Surfaces.* Leipzig, Germany: B. G. Teubner, 1906.

[43] I. Weiss, "Noise resistant invariants of curves," in *Geometric Invariance in Computer Vision* (J. L. Mundy and A. Zisserman, eds.), (Cambridge, Massachusetts, USA), pp. 135–156, MIT Press, 1992.

[44] I. Weiss, "Projective invariants of shapes," in *Proceedings of the Computer Vision and Pattern Recognition CVPR '88*, pp. 1125–1134, IEEE, 1988.

[45] A. M. Bruckstein, E. Rivlin, and I. Weiss, "Scale space semi-local invariants," *Image and Vision Computing*, vol. 15, no. 5, pp. 335–344, 1997.

[46] W. S. Ibrahim Ali and F. S. Cohen, "Registering coronal histological 2-D sections of a rat brain with coronal sections of a 3-D brain atlas using geometric curve invariants and B-spline representanion," *IEEE Transactions on Medical Imaging*, vol. 17, no. 6, pp. 957–966, 1998.

[47] O. Horáček, J. Kamenický, and J. Flusser, "Recognition of partially occluded and deformed binary objects," *Pattern Recognition Letters*, vol. 29, no. 3, pp. 360–369, 2008.

[48] Y. Lamdan, J. Schwartz, and H. Wolfson, "Object recognition by affine invariant matching," in *Proceedings of the Computer Vision and Pattern Recognition CVPR'88*, pp. 335–344, IEEE, 1988.

[49] F. Krolupper and J. Flusser, "Polygonal shape description for recognition of partially occluded objects," *Pattern Recognition Letters*, vol. 28, no. 9, pp. 1002–1011, 2007.

[50] Z. Yang and F. Cohen, "Image registration and object recognition using affine invariants and convex hulls," *IEEE Transactions on Image Processing*, vol. 8, no. 7, pp. 934–946, 1999.

[51] J. Flusser, "Affine invariants of convex polygons," *IEEE Transactions on Image Processing*, vol. 11, no. 9, pp. 1117–1118, 2002.

[52] C. A. Rothwell, A. Zisserman, D. A. Forsyth, and J. L. Mundy, "Fast recognition using algebraic invariants," in *Geometric Invariance in Computer Vision* (J. L. Mundy and A. Zisserman, eds.), pp. 398–407, MIT Press, 1992.

[53] F. Mokhtarian and S. Abbasi, "Shape similarity retrieval under affine transforms," *Pattern Recognition*, vol. 35, no. 1, pp. 31–41, 2002.

[54] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision ICCV'99*, vol. 2, pp. 1150–1157, IEEE, 1999.

[55] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the Conference on Computer Vision and Pattern Recognition CVPR05*, vol. 1, pp. 886–893, IEEE, 2005.

[56] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[57] J. L. Mundy and A. Zisserman, *Geometric Invariance in Computer Vision.* Cambridge, Massachusetts, USA: MIT Press, 1992.

[58] T. Suk and J. Flusser, "Vertex-based features for recognition of projectively deformed polygons," *Pattern Recognition*, vol. 29, no. 3, pp. 361–367, 1996.

[59] R. Lenz and P. Meer, "Point configuration invariants under simultaneous projective and permutation transformations," *Pattern Recognition*, vol. 27, no. 11, pp. 1523–1532, 1994.

[60] N. S. V. Rao, W. Wu, and C. W. Glover, "Algorithms for recognizing planar polygonal configurations using perspective images," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 4, pp. 480–486, 1992.

[61] D. Hilbert, *Theory of Algebraic Invariants.* Cambridge, U.K.: Cambridge University Press, 1993.

[62] J. H. Grace and A. Young, *The Algebra of Invariants.* Cambridge, U.K.: Cambridge University Press, 1903.

[63] G. B. Gurevich, *Foundations of the Theory of Algebraic Invariants.* Groningen, The Netherlands: Nordhoff, 1964.

[64] I. Schur, *Vorlesungen über Invariantentheorie.* Berlin, Germany: Springer, 1968. (in German).

[65] M.-K. Hu, "Visual pattern recognition by moment invariants," *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 179–187, 1962.

[66] G. A. Papakostas, ed., *Moments and Moment Invariants – Theory and Applications.* Xanthi, Greece: Science Gate Publishing, 2014.

[67] R. Mukundan and K. R. Ramakrishnan, *Moment Functions in Image Analysis.* Singapore: World Scientific, 1998.

[68] M. Pawlak, *Image Analysis by Moments: Reconstruction and Computational Aspects.* Wrocław, Poland: Oficyna Wydawnicza Politechniki Wrocławskiej, 2006.

[69] J. Flusser, T. Suk, and B. Zitová, *Moments and Moment Invariants in Pattern Recognition.* Chichester, U.K.: Wiley, 2009.

[70] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification.* Wiley Interscience, 2nd ed., 2001.

[71] S. Theodoridis and K. Koutroumbas, *Pattern Recognition.* Academic Press, 4th ed., 2009.

[72] V. N. Vapnik, "Pattern recognition using generalized portrait method," *Automation and Remote Control*, vol. 24, pp. 774–780, 1963.

[73] V. N. Vapnik and C. Cortes, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[74] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory COLT'92*, pp. 144–152, ACM Press, 1992.

[75] K.-B. Duan and S. S. Keerthi, "Which is the best multiclass SVM method? An empirical study," in *Multiple Classifier Systems MCS'05* (N. C. Oza, R. Polikar, J. Kittler, and F. Roli, eds.), vol. 3541 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg, Germany), pp. 278–285, Springer, 2005.

[76] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[77] B. D. Ripley, *Pattern Recognition and Neural Networks.* Cambridge University Press, 3rd ed., 1996.

[78] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[79] Stanford Vision Lab, "Imagenet large scale visual recognition challenge (ILSVRC)," 2015. `http://www.image-net.org/challenges/LSVRC/`.

[80] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence IJCAI'11*, vol. 2, pp. 1237–1242, AAAI Press, 2011.

[81] Algoritmy.net, "Letter frequency (English)," 2008–2015. `http://en.algoritmy.net/article/40379/Letter-frequency-English`.

[82] J. Kittler, M. Hatef, R. P. Duin, and J. Matas, "On combining classifiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226–239, 1998.

[83] L. Prevost and M. Milgram, "Static and dynamic classifier fusion for character recognition," in *Proceedings of the Fourth International Conference on Document Analysis and Recognition ICDAR'97*, vol. 2, pp. 499–506, IEEE, 1997.

[84] Q. Fu, X. Q. Ding, T. Z. Li, and C. S. Liu, "An effective and practical classifier fusion strategy for improving handwritten character recognition," in *Ninth International Conference on Document Analysis and Recognition ICDAR'07*, vol. 2, pp. 1038–1042, IEEE, 2007.

[85] I. N. Dimou, G. C. Manikis, and M. E. Zervakis, "Classifier fusion approaches for diagnostic cancer models," in *Engineering in Medicine and Biology Society EMBS'06.*, pp. 5334–5337, IEEE, 2006.

[86] D. Wang, J. M. Keller, C. A. Carson, K. K. McAdoo-Edwards, and C. W. Bailey, "Use of fuzzy-logic-inspired features to improve bacterial recognition through classifier fusion," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 28, no. 4, pp. 583–591, 1998.

[87] C.-X. Zhang and R. P. W. Duin, "An experimental study of one- and two-level classifier fusion for different sample sizes," *Pattern Recognition Letters*, vol. 32, no. 14, pp. 1756–1767, 2011.

[88] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, 2004.

[89] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. Computer Science and Scientific Computing, Morgan Kaufmann, Academic Press, 2nd ed., 1990.

[90] P. Somol, P. Pudil, and J. Kittler, "Fast branch & bound algorithms for optimal feature selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 7, pp. 900–912, 2004.

[91] P. Pudil, J. Novovičová, and J. Kittler, "Floating search methods in feature selection," *Pattern Recognition Letters*, vol. 15, no. 11, pp. 1119–1125, 1994.

[92] P. Somol, P. Pudil, J. Novovičová, and P. Paclík, "Adaptive floating search methods in feature selection," *Pattern Recognition Letters*, vol. 20, no. 11–13, pp. 1157–1163, 1999.

[93] J. Novovičová, P. Pudil, and J. Kittler, "Divergence based feature selection for multimodal class densities," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 1, pp. 218–223, 1996.

[94] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1–2, pp. 273–324, 1997.

Index