CHAPTER 1

INTRODUCTION TO PATTERN RECOGNITION

Godfried Toussaint

ABSTRACT

This chapter introduces the basic building blocks of a typical pattern recognition system. The emphasis is on application-independent concepts rather than real industrial problems. These concepts will be treated in greater depth later in the course.

1. The Structure of a Pattern Recognition System

1.1 Automatic Recognition of Patterns

If I write one of the letters of the alphabet in a reasonably neat manner on a piece of cardboard, show it to you and ask you which letter it is you no doubt have no trouble at all in solving this seemingly simple problem, i.e., in identifying the letter. The problem of pattern recognition by machines is the same except that now I would like to place the cardboard in front of a camera connected to my computer and I would like the computer to identify the letter, hopefully, with as much success as you enjoy. It turns out that for a computer this task is much more difficult than appears at first glance. We will try to shed some light on this problem and the techniques that have been brought to bear on its attempted solution. It is instructive to divide into several intermediate stages, the process of starting with a pattern in the physical world and ending with a decision of class-membership of that pattern. It is instructive consider each of these stages separately in the spirit often used when one encounters a difficult problem: divide & conquer. It is also useful to attempt to solve each sub-problem optimally. However, we must remember that ultimately all the solutions of these sub-problems must be woven back together again and the optimality of the entire system is only as optimal as the weakest link in the chain.

A frequently found and rather detailed decomposition of the pattern recognition problem (or also *computer vision* problem) into a series of subproblems is illustrated in Fig. 0. The purpose of a pattern recognition program is to analyze a scene in the real world with the aid of an input device which is usually some form of transducer such as a digital camera and to arrive at a description of the scene which is useful for the accomplishment of some task. For example, the scene may consist of an envelope in the post office, the description may consist of a series of numbers supposedly accurately identifying the zip code on the envelope, and the task may be the sorting of the envelopes by geographical region for subsequent distribution. Typically the camera yields a two-dimensional array of numbers each representing the quantized amount of light or brightness of the real world scene at a particular location in the field of view. The first computational stage in the process consists of segmenting the image into meaningful objects. The next stage usually involves processing the objects to remove noise of one form or another. The third stage consists of feature extraction or measuring the "shape" of the objects. The final stage is concerned with classifying the object into one or more categories on which some subsequent task depends. Let us consider some of these

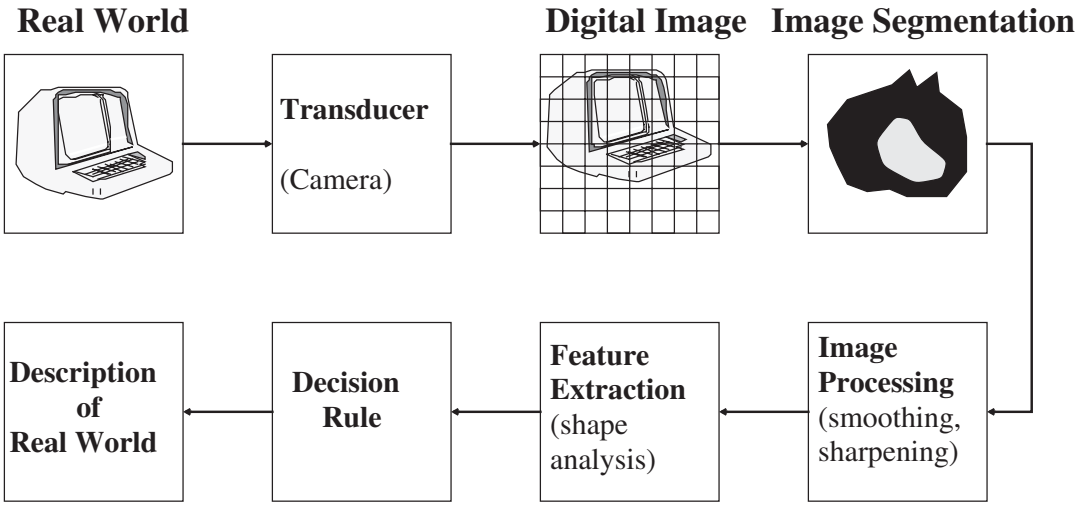


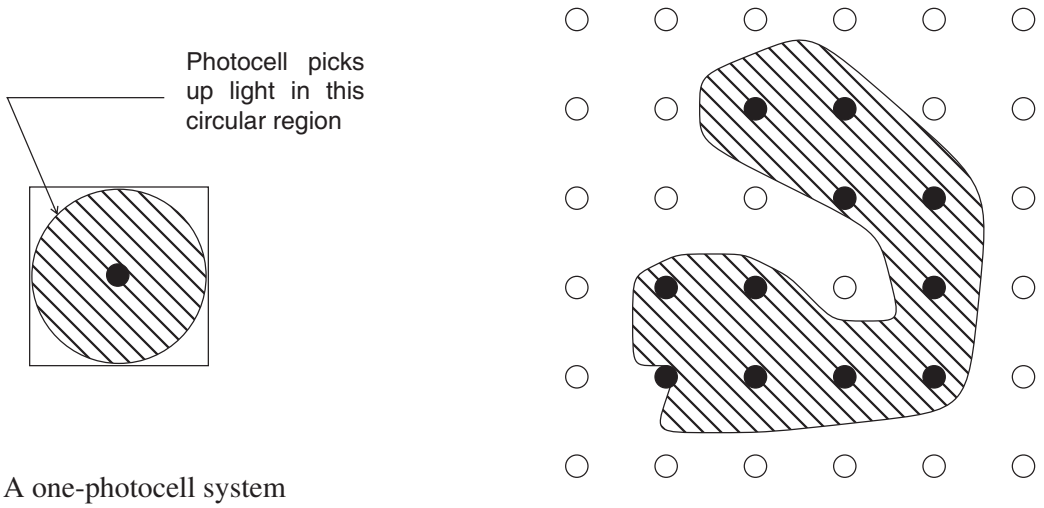
Fig. 0 Decomposing the *pattern recognition* problem into sub-problems.

sub-problems in more detail.

1.2 Transducers

A *transducer* converts an energy pattern from the "real" physical world, such as a *light-intensity* pattern, into a "Platonic" digital pattern in the computer. This digital pattern we call a digital image. It consists usually of a square array or matrix of numbers. Each element or cell in the array of a digital picture is called a *pixel* (a word derived from picture element) and the number associated with a pixel represents the light intensity at that location in the picture. The simplest pattern recognition problem, if it even merits the name, is to decide if a small region of interest in the field of vision is either black or white. Consider for example a machine that examines questionnaires. Questionnaires typically have little square boxes that have to be filled in with a black pencil if appropriate. For example a portion of the questionnaire may have the typed words FEMALE and MALE each followed by a small square. In this case one of the two boxes should be filled in black pencil and the other should be left white (blank). An appropriate transducer for this simple problem would be a single *photo-electric cell* located at the center of the square. A *photo-electric cell*, or photocell for short, converts light to electricity: the more light that is sensed by the photocell's input, the higher the voltage that will be produced at the photocell's output. The photocell will be sensitive to light in some region around its center. Let us assume that this region is a disc contained in the square of interest as illustrated in Fig. 1 (a) and that the photocell output voltage has been calibrated so that if the entire circle is perfectly black (there is no light) the output is zero volts whereas if the entire circle is completely white the output voltage is one volt. Then we can attach a *quantizer* at the output so that for any value of voltage V in between zero and one, the output will be ONE if $V \geq 0.5$ and ZERO if $V < 0.5$. Thus if approximately at least half of the area of the circular region is completely black then the output of the photocell will read ZERO.

Given a much larger and complicated pattern of light in the real world there are many pos-



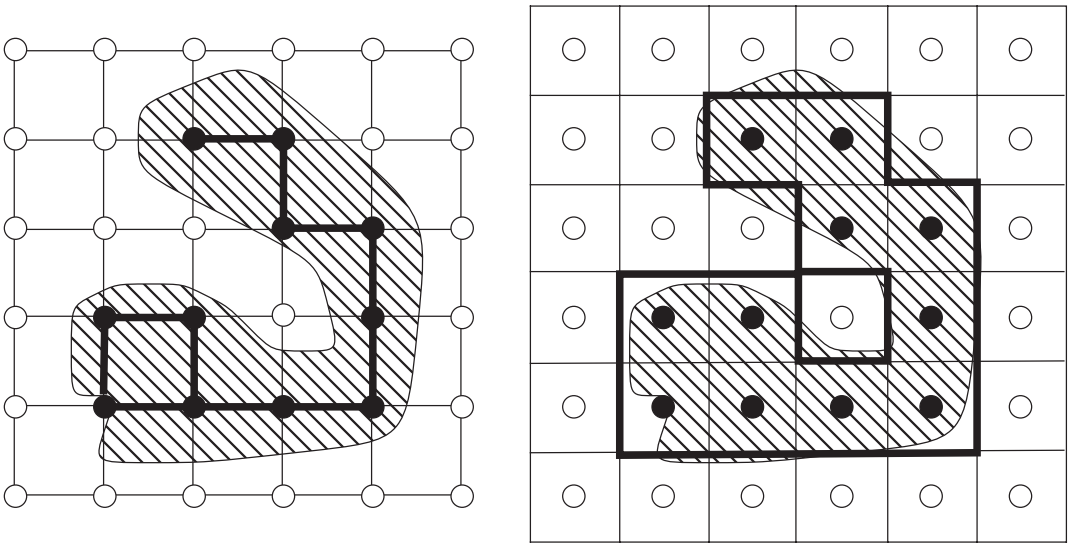
(a) A one-photocell system

(b) A square array of photocells.

Fig. 1 A transducer made up of photocells.

sible ways of designing transducers using the basic single photocell. For example, the cell could be moved around quickly to different locations on the pattern and the output voltage sampled at each location. To keep things simple let us assume that we attach a group of these photocells together in the form of a two-dimensional array, matrix, or grid as in Fig. 1 (b). With all the cells calibrated just as before a complicated pattern now becomes an array of ONE's and ZERO's, a *digital image*.

There are at least two fundamental ways of representing a digital image in a computer and these are illustrated in Fig. 2. The first method superimposes a square *grid graph* on the centers of the photocells. In other words the locations of the photocells correspond to the *vertices* of the graph and two vertices of the graph are connected with an *edge* if their corresponding photocells are vertically or horizontally adjacent to each other. This method is illustrated in Fig. 2 (a) where the vertices of the ZERO output cells are labelled black and the others labelled white. In this representation a subgraph of the grid graph, shown in bold lines in the figure corresponds to the original pattern. A second method is to superimpose a square tessellation of the area of interest such that the location of the center of each photocell corresponds to the center of its corresponding square in the tessellation as illustrated in Fig. 2 (b). In this case a square of the tessellation is labelled black if its corresponding photocell output is a ZERO. In this representation a subregion of the tessellation corresponds to the original pattern. In the figure it is illustrated by outlining its boundary in bold lines. Since most existing systems have not incorporated the grid graph approach, we will be concerned with the second representation. Image processing algorithms that use this representation are known as *pixel-based* or *raster-graphics* methods. In this representation the boundary of a connected component of the pattern is often conveniently represented as a simple polygon. A long row or column of pixels is then represented by a single line segment affording the possibility of saving in subsequent storage and computation time. Therefore algorithms that take as input a simple polygon are also a central concern of pattern recognition. Algorithms that use this representation are known



(a) A square *grid graph*. (b) A square *tessellation*

Fig. 2 Two ways of representing a digital image.

as *geometric* or *vector-graphics* methods.

1.3 Preprocessing

Preprocessing is the name given to a family of procedures for smoothing, enhancing, filtering, cleaning-up and otherwise massaging a digital image so that subsequent algorithms along the road to final classification can be made simple and more accurate.

For example, it may be the case that the transducer is sensitive to specks of dust in the input field of vision that cause random changes of pixels from ONE to ZERO and vice-versa yielding so-called *salt-and-pepper* noise. A typical noise removal procedure is to change the value of a pixel from ONE to ZERO if all its neighboring pixels have a value of ZERO. Similarly, a pixel's value is changed from ZERO to ONE if all its neighboring pixels have a value of ONE.

As a second example of preprocessing consider smoothing a simple polygon that represents the boundary of a digital image. It may be the case that the boundary has been distorted by some process, perhaps the poor writing of a person in the case of hand-printed characters. A possible way in which to smooth the polygon to reduce this kind of noise is to create a new polygon by joining the midpoints of all the original polygon's edges in the same order defined by the polygon. This procedure can be repeated several times as illustrated in Fig. 3 where it has been applied twice to obtain the polygon with thick dashed lines. This is a very efficient smoothing procedure. For a polygon with n sides the algorithm clearly runs in $O(kn)$ time where k is the number of times it is applied. Of course it is assumed here that only a few iterations are necessary. If k is very large the algorithm quickly slows down. Several questions immediately arise concerning this algorithm. If the input polygon is simple (non self-crossing) is the output polygon always guaranteed to be simple? Given a simple or non-simple polygon as input will the output polygon always be convex in

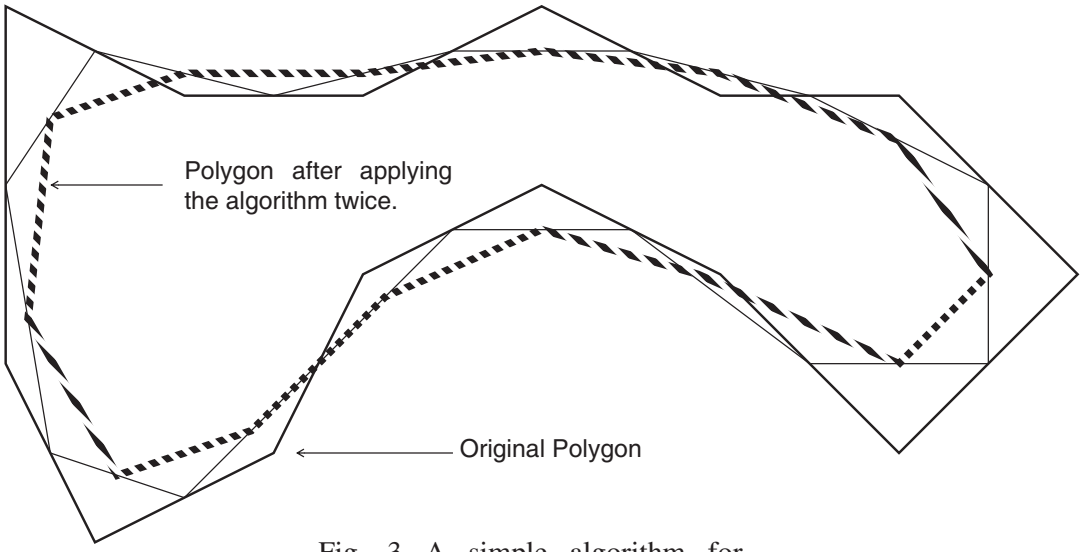


Fig. 3 A simple algorithm for *smoothing* simple polygons.

the limit, i.e., as the number of iterations approaches infinity?

1.4 Feature Extraction

Feature extraction is the name given to a family of procedures for measuring the relevant shape information contained in a pattern so that the task of classifying the pattern is made easy by a formal procedure. For example, in character recognition a typical feature might be the height-to-width ratio of the letter. Such a feature would be useful in differentiating between a W and an I in some machine fonts such as *Times*, where the W is much wider than the I. On the other hand this feature would be quite useless in distinguishing between an E and an F. The task of designing a feature extractor is one of finding as few features as possible that adequately differentiate the patterns in a particular application into their corresponding pattern classes. There is no theory either in computer science or psychology to solve this problem optimally. Therefore this is an intuitive and ad hoc "science." We close this section with a hypothetical problem where two geometric features suffice to solve the problem in a satisfactory manner.

Consider the following problem encountered by ROBY the robot. Let us assume ROBY is standing at the end of a long conveyor belt on which are arriving, at a steady and humanly exhausting speed, juicy mangoes from Brazil and ripe oranges from Seville. ROBY's television-camera-like eye is pointing down towards the conveyor belt and is busy taking pictures of the advancing fruit. ROBY is equipped with a mechanical arm and its task is to place the oranges in a box on its left and the mangoes in a box on its right. ROBY must therefore first classify the objects pictured in its "retinas" into mangoes or oranges. ROBY's eyes contain a micro-chip with a standard image-processing program (such as we will encounter later) that, given a picture of an object, quickly extracts the object's boundary, in the form of a polygon and sends the polygon to that part of ROBY's "brain" that contains a specialized program for making decisions.

Given a polygon it is very easy for ROBY to determine if it is *convex* or not. Recall that an object is convex if for every pair of points x and y contained in the object, the line segment $[x,y]$

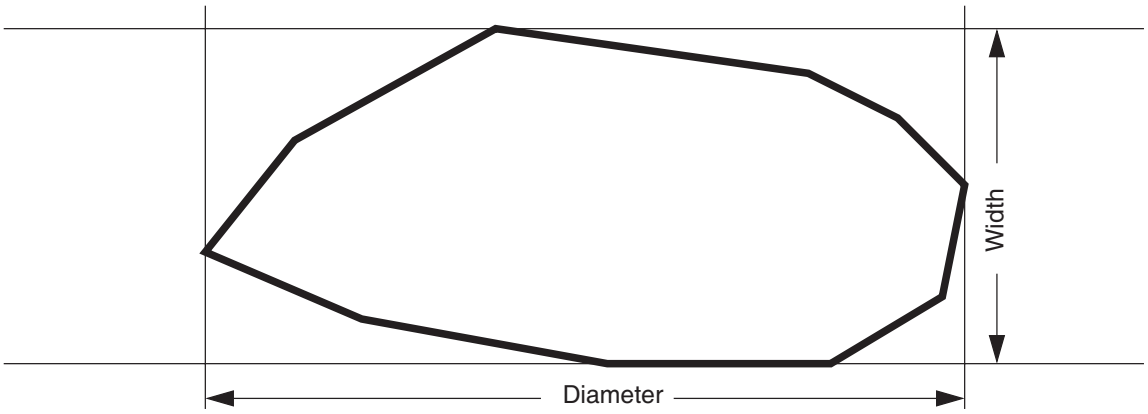


Fig. 4 The *width* and *diameter* of a convex polygon.

joining *x* and *y* is also contained in the object. Convexity would be a perfect measure of shape for distinguishing between oranges from Seville and Chiquita bananas because oranges are always convex and bananas never (at least not in Canada although in Fiji one can find short two-inch bananas which are convex). On the other hand convexity is a useless morphological discrimination measure for oranges and mangoes because both types of fruit are convex (almost always). However, we do know that, although oranges and mangoes are convex, the boundary polygons taken from the pictures of oranges tend to be *circular* whereas those from mangoes tend to be *oblong*. Now, imagine placing the convex polygon, obtained from the image of either an orange or a mango, between two parallel lines and slide the parallel lines toward each other until they both just touch the polygon. In this position measure the separation between these two lines which we call tangent lines of support. This distance measures the *thickness* of the convex polygon for that particular placement of the parallel lines. If we now rotate the polygon slightly in say the clockwise manner and repeat the above procedure, and continue in this way computing the *thickness* for all possible rotated positions of the polygon, two very interesting measures of the shape of the polygon are immediately evident. The *maximum* thickness obtained over all rotated positions of the polygon is called the *length* or *diameter* of the object whereas the *minimum* thickness obtained is known as the *width* of the object. The width and diameter of a convex polygon are illustrated in Fig. 4. We know from our experience with mangoes that their length is about twice as large as their width. For oranges on the other hand length and width are about the same since oranges tend to resemble spheres quite closely. Therefore we can program a very simple rule in ROBY's brain as follows. First compute the length and width of the boundary polygon extracted from the image of the fruit under consideration. Next divide the length by the width. If this ratio is greater than 1.5 then call the fruit a mango and put it in the appropriate box. Otherwise conclude the fruit is an orange. With a simple and rather "stupid" rule like this we can make ROBY give the illusion that it is quite intelligent, especially to people who do not know the actual decision rule used by ROBY. In fact one could say that Artificial Intelligence in general is the art of inventing stupid and simple mechanisms that give the illusion of being intelligent and exhibiting complex behavior. With the aid of *computational geometry* (the science of the design of efficient algorithms for solving geometric problems) we can compute many geometric features or properties of objects, such as the width and

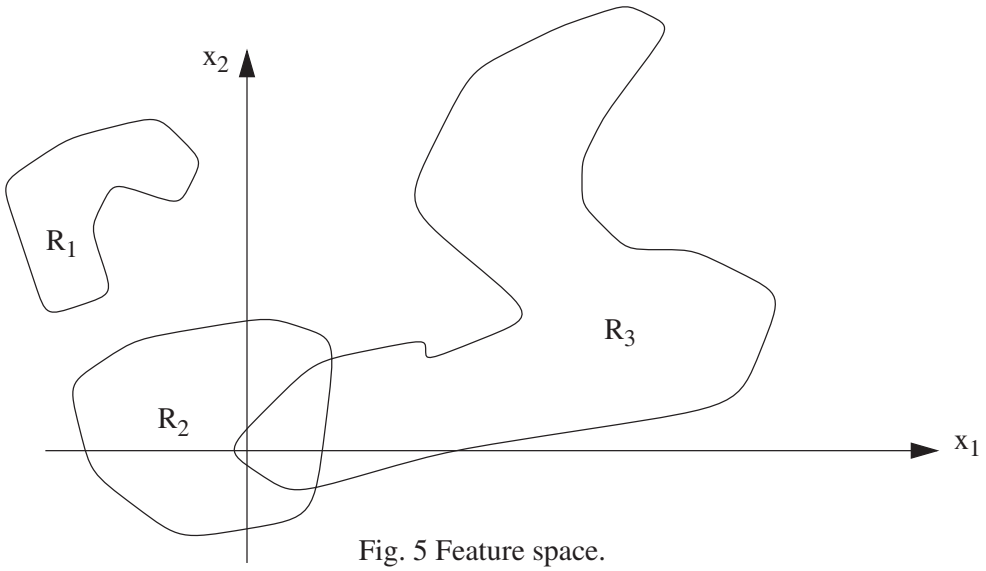


Fig. 5 Feature space.

diameter in the above example, in an efficient manner.

1.5 Classification

Classification is concerned with making decisions concerning the class membership of a pattern in question. The task in any given situation is to design a decision rule that is easy to compute and will minimize the probability of misclassification relative to the power of the feature extraction scheme employed. This part of pattern recognition is quite formal since we can make *optimal* decisions using *Bayesian decision theory* and incorporate several tools from the areas of *probability* and *statistics* such as *estimation theory*.

If we assume that *d* measurements (features) are observed on a pattern or object then we can represent the pattern by a *d*-dimensional vector $X = (x_1, x_2,..., x_n)$ and usually refer to X as a *feature vector* and the space in which X lies as the *feature space*. Patterns are thus transformed by the feature extraction process into points in *d*-dimensional feature space. A pattern class can then be represented by a region or sub-space of the feature space. Classification then becomes a matter of determining in what region of the feature space an unknown pattern falls into. Fig. 5 illustrates a 2-dimensional feature space with three pattern classes represented as regions R_1, R_2 and R_3.

1.6 Sequential Classification

Sometimes measurements (features) are very expensive and rather than making all the measurements in a batch mode we make measurements one at a time and at each step we decide, depending on our confidence obtained so far, whether to make another measurement or whether to make a final decision. This is referred to as sequential classification.

1.7 Contextual Information

Sometimes it is impossible to decide the class membership of a pattern without looking at the context in which the original pattern is embedded. It such situations it is necessary to use contextual information at the classification stage. For example, consider the letter in the middle in Fig. 5. Viewed in isolation this letter is ambiguous and it is impossible to decide if it is a corrupted “A” or a corrupted “H.” On the other hand, if we know the letters appearing before and after it in a text recognition task then as the figure shows we can easily determine whether the entire word is “THE”

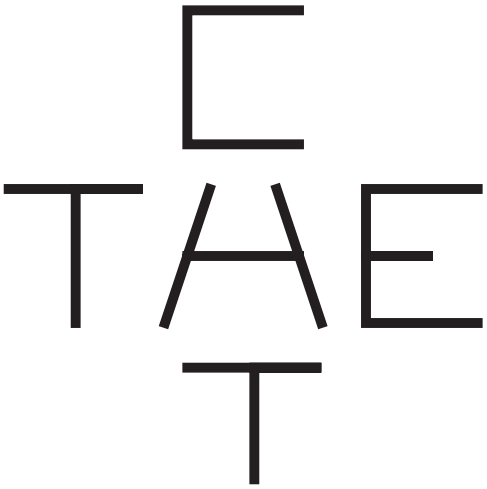


Fig. 5 Using contextual information to disambiguate a pattern.

or "CAT." More involved procedures use Markovian decision theory and dictionary look up methods as well as post-processing spelling correction algorithms to accomplish this goal.

2. Template Matching

In some applications the input patterns can be more or less controlled so that they are almost always identical to some idealized prototype. In such situations a simple template matching method may give satisfactory results. A template for each prototype is kept in memory. A new unknown pattern is then matched to each stored template in turn and when a close match is found it is classified into the class of this "winning" template.

3. An Industrial Example - The Bankers Machine

See class notes.

Another industrial example of a pattern recognition machine that must meet many constraints is the *pay-phone coin-validation* system [Ba90]. Here the machines must have small memory and simple decision rules because, with millions of pay phones sitting idle, the electrical power consumption becomes a limiting factor.

4. References

[Ba90] Barlach, F., "Pay phone validation using pattern recognition," *Pattern Recognition,* vol. 23, No. 3-4, 1990, pp. 379-384.

[DH73] Duda, R. O. and Hart, P. E., *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[Ho86] Horn, B. K. P., *Robot Vision,* The MIT Press, 1986.

[Ne82] Nevatia, R., *Machine Perception*, Prentice-Hall, Inc., New Jersey, 1982.

[Ni90] Nilsson, N. J., *The Mathematical Foundations of Learning Machines,* Morgan Kaufmann, Inc., San Mateo, California, 1990.

[To88] Toussaint, G. T., (Ed.), *Computational Morphology*, North-Holland, Amsterdam, 1988.