FEATURES / SHAPE

Distributed Learning of Texture Classification

John R. Sullins
Center for Automation Research
University of Maryland
College Park, MD 20742

Abstract

A large number of statistical measures have been postulated for the description and discrimination of textures. While most are useful in some situations, none are totally effective in all of them. An alternative approach is to *learn* which measures are best for particular circumstances. In this paper the distributed learning system of *constraint motion* is used to learn relevant texture descriptors from a set of well-known first and second order grey-level statistics. Using this system, a network of distributed units partitions itself into sets of units that detect one and only one of the given classes of textures. Each of these sets is further partitioned into individual units that detect natural subtypes of these texture classes, ones which do not necessarily produce the same types of statistics at the local level. Together, these units form a network capable of determining the texture classification of an image.

1. Introduction

"Texture" is an important cue for segmentation and for the description of objects that do not always have fixed shapes (such as "trees" or "grass"). However, defining our notions of texture is a difficult problem. Consider the textures in Figure 1. To our eyes they are obviously different from one another, but it is not easy to say exactly why. We can give vague descriptions like "rough", "uniform", "chaotic", etc., but unless we can define these terms mathematically they are of no use for machine vision.

Many statistical measures that attempt to simulate such descriptions have been proposed for texture classification; these include uniform density of image features (Aloimonos 1988), texture energy templates (Laws 1980), and second order grey-level dependencies (Kruger et al. 1974; Hall et al. 1971; Haralick et al. 1973). The general idea is to create a measurement or small set of measurements that describe textures. Theoretically, textures that "look alike" to our eyes should receive similar scores when such a measure is applied to them. Unfortunately, there is no such single measure that will accomplish this for all of the many different textures that exist in the world, so most classification systems use a combination of several of these features. Each of the n measures in the feature vector is applied to the texture sample in question, giving a point in the n-dimensional *feature space*. In the best case, points from similar textures will form distinguishable *clusters* in the feature

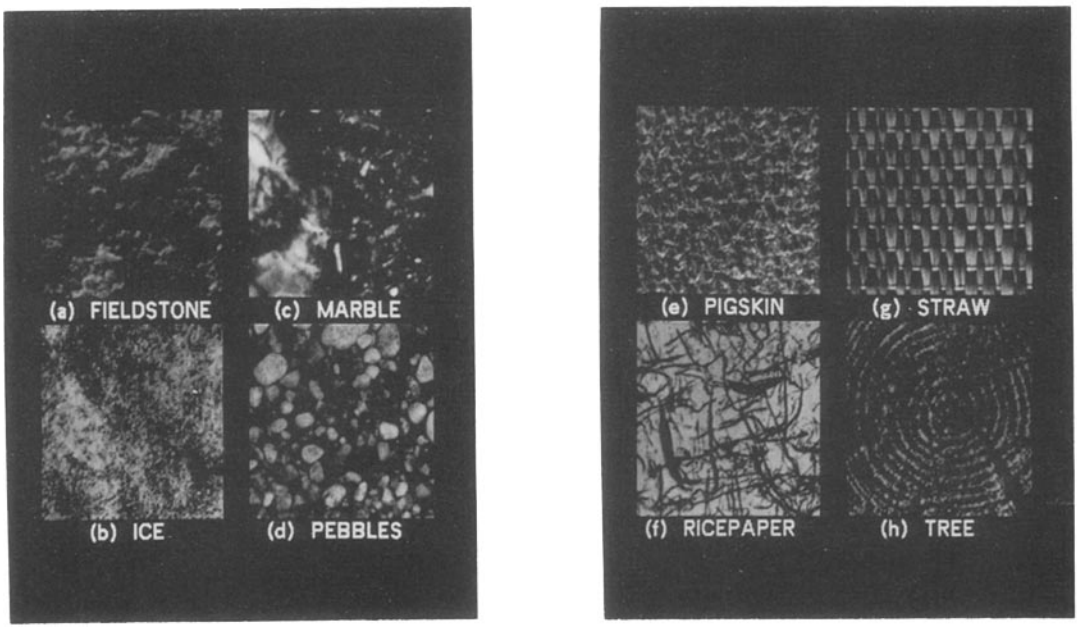

Figure 1

space, groups of points easily separable from other groups. If so, methods (Fukunaga 1972; Tou and Gonzolas 1974; Fu 1974) exist to *learn* a suitable partitioning of the feature space from a set of example texture images.

Such simple clustering may not be possible, however, especially when the image size is small and we can see only a small portion of the texture. Consider the textures shown in Figure 2. The smaller squares give several dissimilar samplings of the textures "marble", "pebbles", and "tree". It may well be that no single range of features is sufficient to discriminate them from other textures. We may need different detectors for each subtype of a given texture. Learning systems that use simple clustering may fail to partition the problem in this manner.

In this paper we take a different approach, applying the general learning system we call *motion in constraint space* (Sullins 1988) to the problem of texture classification. Textures are described in terms of a large number of simple first and second order statistics rather than a few complicated ones. The learning process is more complex, however. Each processor in the network focuses on a *portion* of a particular target texture class. That processor generalizes from a set of example textures by determining which of the statistics are relevant to the task of separating that subtexture from the other non-target texture classes, and the ranges of those statistics within which the subtexture lies. When combined into a network, those

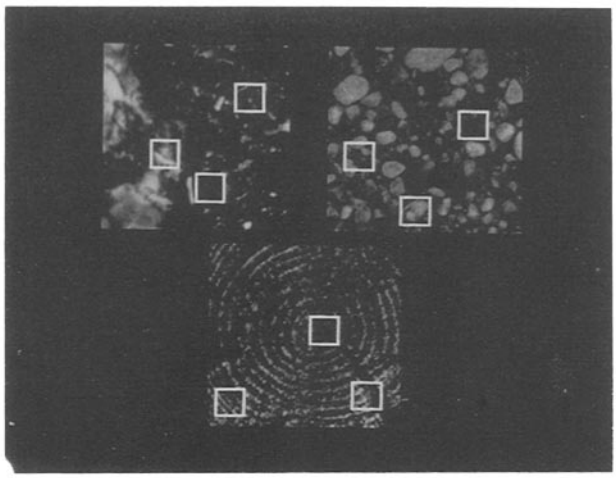

Figure 2

subtexture processors detect those and only those textures that are members of the target class.

2. Texture classification statistics

Since this learning algorithm is designed to run on a distributed system, first and second order grey level statistics were deemed especially appropriate for the problem. The co-occurrence matrices needed to compute the second order statistics may be accumulated in parallel.

The first order statistical properties of the individual grey levels used were the *mean* grey level $\mu \equiv \frac{1}{N}\sum i\ P_1(i)$, and the *variance* $\sigma^2 \equiv \frac{1}{N}\sum(i-\mu)^2\ P_1(i)$, where $P_1(i)$ are the number of pixels in the image with grey level i and N is the total number of pixels in the image. Second order grey level statistics are relationships between grey levels at nearby pixels. These are generally computed in term of *co-occurrence matrices* $P_2(i,j|d,\theta)$, the number of points (x,y) and $(x+d\cos\theta,\ y+d\sin\theta)$ that have grey levels i and j respectively. The second order statistics (Haralick et al. 1973) used were

Energy: $\quad E(d,\theta) \equiv \sum_i \sum_j\ [P_2(i,j|d,\theta)]^2$

Inertia: $\quad I(d,\theta) \equiv \sum_i \sum_j\ (i-j)^2\ P_2(i,j|d,\theta)$

Correlation: $\quad C(d,\theta) \equiv \dfrac{\sum_i \sum_j\ i\ j\ P_2(i,j|d,\theta) - \mu_x \mu_y}{\sigma_x \sigma_y}$ where

$$\mu_x \equiv \sum_j i \sum_i P_2(i,j|d,\theta) \qquad \sigma_x^2 \equiv \sum_j (i-\mu_x)^2 \sum_i P_2(i,j|d,\theta)$$

$$\mu_y \equiv \sum_i j \sum_j P_2(i,j|d,\theta) \qquad \sigma_y^2 \equiv \sum_i (j-\mu_y)^2 \sum_j P_2(i,j|d,\theta)$$

The second order statistics were computed with values of 1 and 2 for d and values of 0, $\pi/4$, $\pi/2$, and $3\pi/4$ for θ (statistics for $\theta + \pi$ were classified with those of θ). Counting the two first order statistics, this gave a total of 26 statistics from an example image. The second order statistics were scaled logarithmically in order to give an approximately linear distribution, and all values were normalized to lie between 0 and 8.

3. Defining learning

We define *learning* as the duplication of a given input-output behavior by some system. Given a set of binary-valued inputs and outputs, this means that for all possible combinations of values of the input units (that is, all possible *input vectors*) the system activates the correct set of output units (that is, the correct *output vector*). In this context the behavior of an output may be likened to a Boolean formula in disjunctive normal form, and learning may be defined as the determination of that formula over all possible input vectors. For the problem of texture discrimination, the inputs are statistical measures of the sample texture, and each output unit is active if a particular texture type is present.

If the DNF expression has n conjunctive subexpressions (joined by "or"s) then it may be simulated in parallel by a machine with n processors, with the speedup associated with such distribution. Since the expression is in disjunctive normal form, if any of the subexpressions are true then the entire expression is true. Each subexpression is the conjunction of a subset of the inputs (or their negations), and is true only if all of those inputs are in the correct state.

In this sense, each input which is a part of a processor's conjunctive subterm is a *constraint* for that processor. Consider a particular input that corresponds to the binary variable "A". If the processor's conjunctive term contains A, the input must have the value 1 for the processor to be active. If it contains $\overline{A}$, the input must have the value 0. If it contains neither (which we represent as "-"), then its value does not matter to the processor.

As with most learning algorithms, we will assume that there exists some form of *supervision* that gives the network the correct classification (as *target texture* or *non-target texture*) for any given input texture. This supervision may not always be accurate, of course. Sometimes it may report that the target texture is present when it is not, and vice-versa. A learning system with applications in the real world must be able to cope with such errors.

3.1. Representing the statistics

Since the learning algorithm is designed for binary input, the continuous-valued (from 0 to 8) texture statistics were converted by assigning each value 5 binary inputs (giving a total of 130 binary inputs) whose activation depended on whether its value

S fell within a certain range:

input 0 active if $S \leq 4$ input 1 active if $1 \leq S \leq 5$
input 2 active if $2 \leq S \leq 6$ input 3 active if $3 \leq S \leq 7$
input 4 active if $4 \leq S$

This representation has the advantage of being able to represent many *ranges* of acceptable values for a statistic, depending on the number of active constraints and how their ranges overlap. For example, a statistic with value 2.4 would be represented as 11100. From this, we could derive a very restrictive set of constraints (if all 5 were constraints, then the value of the statistic would have to lie between 2 and 3 in order for the processor to be activated), less restrictive sets of constraints (for instance, 11--- would confine the statistic to lie between 1 and 4), or no constraints (----- would mean that the statistic is unimportant to the texture).

4. The learning algorithm

In this section we outline a system called *motion in constraint space* that is designed to learn texture classifications from a set of supervised examples. While it will be described in terms of texture discrimination, a more general and complete description of its capabilities may be found in (Sullins 1988).

Initially, each processor chooses a *seed texture* from the input texture examples receiving positive indication from an output. These input vectors correspond to images containing the texture type of that output (the *target texture* of that output). The states of the input units, which represent the values of the statistics of the seed texture, are the *potential constraints* for that processor. When all these constraints are enforced, the processor is restricted to detecting only those textures that have identical statistics to the seed texture. Focusing on a particular seed texture at each processor and eliminating input statistics that have little or no effect on it helps the system to properly distribute the task of learning the behavior.

The processor will then *generalize* to detect the subclass of the target texture class which contains the seed texture by removing these constraints. Eliminating constraints widens the range that a statistic may lie in in order for it to be accepted. In many cases we will eliminate *all* constraints for a particular statistic, making it completely irrelevant to the processor. As long as each major subclass is represented by a seed texture at at least one processor, this algorithm will form a network that correctly detects the target texture in most cases.

The core of this system is this addition and subtraction of constraints on the statistics at the processors in order to minimize the difference between the expected and the actual texture classifications -- that is, the *constraint motion.* Simply put, we will want to place restrictions on the values of statistics (by adding constraints) when doing so would prevent non-target textures from being accepted and we will want to remove restrictions (by removing constraints) that would prevent target textures from being accepted. The performance of the network formed by this algorithm will not be perfect, of course, as no natural texture can be represented by a simple DNF expression. However, the processors will tend to choose those constraints that maximize the correctness of the texture classifications.

4.1. Adding constraints

Generally speaking, constraints should be added to processors that accept too many non-target input textures. Because of the DNF structure of the network, a texture incorrectly accepted by any processor is also incorrectly accepted by the entire network. A non-target texture must differ from a processor's seed texture in the value of one or more inputs (otherwise, that seed texture would not have received positive indication). Each of those inputs is a potential constraint that would prevent the texture from being accepted in the future, as they would restrict the values of certain statistics to a point where the input texture would no longer lie within their ranges. Each of these potential constraints receives a *positive vote* for change at that processor.

On the other hand, we do not wish to add a constraint to a processor if the constraint would prevent too many target texture images from being accepted, specifically those target textures *not accepted by any other processor.* If that processor were no longer able to accept such "uniquely accepted" textures, then the entire network would incorrectly reject them as well. For all target textures uniquely accepted by the processor, each input that has a different value from that of the seed texture (and thus would cause the texture to be rejected if it were to become a constraint) receives a *negative vote* against change.

Positive and negative votes are collected for each potential constraint over a large sampling of the input vectors. This insures that supervisor error will have little effect on the system, as the incorrect data will usually be outvoted by the correct data. At the end of that time, potential constraints with significantly more positive than negative votes are added to the processor, narrowing the acceptance range of the processor for those statistics.

4.2. Removing constraints

The removal of constraints is somewhat similar to the addition of them, removing constraints that prevent target texture images from being accepted and not removing those that prevent non-target texture images from being accepted. The main difference has to do with the conjunctive structure of the processors. A texture may have values that lie outside of the accepted ranges of many of its statistics, so it might not meet *many* existing constraints of a processor. We want to remove constraints when doing so would cause more target textures to be accepted, but for most textures removing a single constraint will not make any difference.

We allow all textures that are not accepted by the network to influence the removal of constraints, but using an exponential function to give "near miss" textures more influence than others. If a target texture is not accepted by the network, then each constraint at processor i that prevented it from being accepted at that processor is given a positive vote for removal proportional to σ^{-n_i}, where n is the number of constraints that the vector failed to meet at processor i. That is, each processor is changed in proportion to how close it already is to accepting the texture; this helps the system to properly distribute the responsibility for accepting textures by assuring that only a few processors are forced to learn each one. Initially σ is 1

(each texture has an equal effect on the voting) and over time it is increased. This forces a processor to focus on a particular subclass of the target texture and helps to stabilize the behavior of the network in the long run.

Negative votes against removing constraints are collected in a similar manner. If a constraint helps to prevent a non-target texture from being accepted, it receives a negative vote against removal proportional to σ^{-n_i}. As in the case of adding constraints, the input is sampled over a certain period of time. If the number of positive votes is greater than the number of negative votes, the constraint is removed, widening the acceptance range of the processor for that statistic.

5. Experimental results

Eight 96 × 96 texture images (shown in Figure 1) were chosen for the tests. In order to have co-occurrence arrays P_2 of manageable sizes, the number of grey levels was reduced from 256 to 9 using the Isodata algorithm. A particular target texture was chosen for each run represented by a single output which was active when the target texture was present, inactive otherwise. This output was assigned 10 intermediate processors. An example texture image was chosen from the target texture with probability 50%, from one of the other textures otherwise. The image was a randomly chosen 16 × 16 subpicture of the texture. This gave a total of 6400 possible examples for each texture.

5.1. Learning ability of the system

Several runs of the learning system were made (with no supervisor error) for each of the eight target textures shown in Figure 1. Figure 3 gives the average learning curves for each texture. The X axis represents the total number of examples presented to the network at that point, and the Y axis represents the percentage of the time that the network correctly classified the input as target or non-target texture, over samplings of 1000 input vectors.

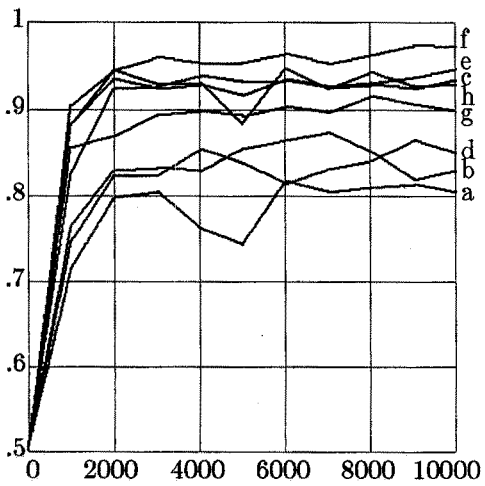

Figure 3

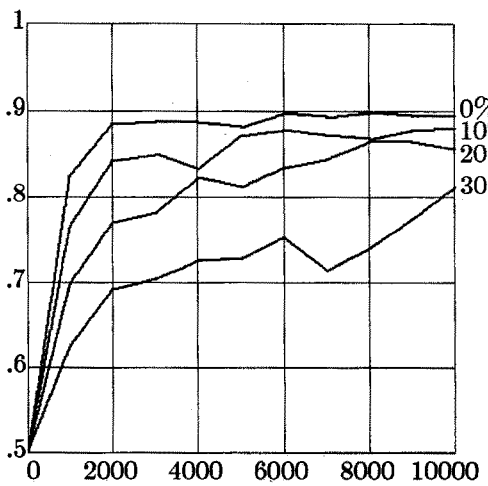

Figure 4

The network reached a certain level of correctness very quickly (2000 - 3000 sample input vectors), and made any further improvements very slowly. The system was able to make generalizations about the textures after seeing a small percentage of the 51,200 possible texture samples. The textures with the least correctness were those most easily confused with the others. The average correctness over all of the textures quickly stabilizes at close to 90%. This is a very good result, especially for sample images of size 16×16. The typical correctness of other feature classification systems is considered to be around 90% for 64×64 images.

5.2. Dealing with supervisor error

Several runs were also made for each texture at various levels of *supervisor error*. Figure 4 gives the learning curves for the average correctness over all textures for error percentages of 0%, 10%, 20%, and 30% in the supervision. In this case, the Y-axis is a measure of the *true* correctness of the responses -- how well they matched the actual desired output value, before any corruption by supervisor error. The learning is very resilient to levels of error less than 30% percent, and even for 30% the correctness seems to approach that of other levels. In fact, the correctness of the system for these high levels was greater than the correctness of the supervision itself, indicating that the network was able to find good features despite the error.

5.3. Importance of distribution

As mentioned above, this system creates networks capable of recognizing textures from a very small sample, 16 × 16 versus the usual 64 × 64. This is due in large part to the system's ability to properly *distribute* the detection of the different types of samples about the network. The smallness of the sample size would often cause some of the expected properties of the target texture to be absent for a particular sample. Distribution of the target description allows the creation of different detectors for each of these situations.

We measure the importance of distribution by keeping track of the percentage of time that each intermediate processor is active when the output is correctly activated, and looking at the processor with the highest percentage of activation. If it is close to 100%, then the set of features detected by it is a sufficient description of the target texture and no distribution is needed. The lower it is, the more distribution was necessary. Table 1 gives percentages taken from the test runs for each target texture at 0% supervisor error.

Comparing this with Figure 3 shows that the amount of distribution for a texture was directly related to the detection error -- that is, the difficulty in discriminating the target texture from other textures. In the cases where textures were similar, many different detectors with tighter constraints were needed to detect the target texture and only the target texture.

Table 2 gives this distribution measure, averaged over all target textures, for the different levels of supervisor error. The amount of distribution increases with the error. As the process of texture classification becomes more confused, tighter constraints are needed to discriminate the textures. Since this decreases the number

of textures that a processor can detect, more processors are needed to cover them all. This distribution is one of the main reasons for the system's good performance at high levels of supervisor error.

fieldstone	82.19
ice	79.60
marble	84.28
pebbles	78.58
pigskin	88.43
ricepaper	94.89
straw	87.24
tree	82.08

Table 1

0%	84.66
10%	75.57
20%	64.76
30%	60.95

Table 2

5.4. Understanding the networks

Table 3 shows what the constraints of a typical processor might look like after a while. This particular processor was one of those set up to learn the texture "marble". The table gives the constraints for the features described in section 3 (with angles and distances for the second order constraints), formed after 10,000 input vectors. In effect, it characterizes "marble" as having low mean, high energy and low inertia at distance 1 in the direction $\pi/2$, and low correlation at distance 1 in all directions.

statistic	constraints					seed	statistic	constraints					seed
mean	-	1	-	-	-	11000	variance	-	-	-	-	-	10000
$E(1,0)$	-	-	-	-	-	00111	$E(2,0)$	-	-	-	-	-	00111
$E(1,\pi/4)$	-	-	-	-	-	00111	$E(2,\pi/4)$	-	-	-	-	-	00111
$E(1,\pi/2)$	-	0	0	-	-	00011	$E(2,\pi/2)$	-	-	-	-	-	00011
$E(1,3\pi/4)$	-	-	-	-	-	00111	$E(2,3\pi/4)$	-	-	-	-	-	00111
$I(1,0)$	-	-	-	-	-	00011	$I(2,0)$	-	-	-	-	-	00001
$I(1,\pi/4)$	-	-	-	-	-	00111	$I(2,\pi/4)$	-	-	-	-	-	00001
$I(1,\pi/2)$	1	-	-	-	0	11100	$I(2,\pi/2)$	-	-	-	-	-	11100
$I(1,3\pi/4)$	-	-	-	-	-	00111	$I(2,3\pi/4)$	-	-	-	-	-	00001
$C(1,0)$	-	-	-	0	-	11100	$C(2,0)$	-	-	-	-	-	10000
$C(1,\pi/4)$	-	-	-	0	-	11100	$C(2,\pi/4)$	-	-	-	-	-	10000
$C(1,\pi/2)$	-	-	-	0	-	11100	$C(2,\pi/2)$	-	-	-	-	-	11000
$C(1,3\pi/4)$	-	-	-	0	-	11000	$C(2,3\pi/4)$	-	-	-	-	-	10000

Table 3

This table demonstrates another advantage of the constraint representation. Besides producing networks capable of correct texture discrimination, it can also show us things about the textures themselves. The statistical measures used here (and elsewhere) to classify textures usually do not correspond to our intuitive measures

(such as "roughness"), so it is less than obvious how to simply describe them in those terms. The final states of the constraints in processors such as Table 3 can give us those descriptions.

6. Conclusions

We have presented and tested a distributed system that learns texture discrimination in terms of simple first and second order statistics. The networks formed by this learning system were capable of performing this task quite well, at a level comparable to that of more complex measures of texture. This high level of correctness was maintained despite small sample size and high supervisor error.

In addition, the creation of distributed networks of different, specialized detectors (versus a single, general detector) was shown to be important for the texture discrimination problem. This was especially true for textures that were very similar, or under conditions of small sample size or high supervisor error, situations that often occur in the real world.

6.1. Acknowledgements

This work was supported by the Defense Advanced Research Projects Agency and the U. S. Center for Night Vision and Electro-Optics under Contract DAABo7-86-KF073. The author wishes to thank John Aloimonos and Azriel Rosenfeld for their advice and constructive criticism.

Bibliography

Aloimonos, J. (1988). Shape from texture. *Biological Cybernetics, 58*, 345-360.

Fu, K. S. (1974) *Syntactic Methods in Pattern Recognition* New York: Academic Press.

Fukunaga, K. (1972) *Introduction to Statistical Pattern Recognition.* New York: Academic Press.

Hall, E. L., Kruger, R. P., Dwyer, S. J., Hall, D. L., McLaren, R. W., and Lodwick, G. S. (1971). A survey of preprocessing and feature extraction techniques for radiographic images. *IEEE Transactions on Computers, 20.*

Haralik, R. M., Shanmugam, R., and Dinstein, I. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man and Cybernetics, 3*, 610-621

Kruger, R. P., Thompson, W. B., and Twiner, A. F. (1974). Computer diagnosis of pneumoconiosis. *IEEE Transactions on Systems, Man, and Cybernetics, 45*, 40-49.

Laws, K. I. (1980). Textured image segmentation. Ph.D. dissertation, Department of Engineering, University of Sothern California.

Sullins, J. R. (1988). Distributed learning: motion in constraint space. University of Maryland Technical Report CAR-412.

Tou, J. T. and Gonzalez, R. C. (1974) *Pattern Recognition Principles.* Reading, MA: Addison-Wesley.