Brock University

Department of Computer Science

Machine Perception from the Evolutionary Viewpoint: The Hyperball Algorithms

Vlad Wojcik and Behzad Salami
Technical Report # CS-08-08
September 2008

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

Machine Perception and Learning from the Evolutionary Viewpoint: The Hyperball Algorithms

Vlad Wojcik and Behzad Salami
Department of Computer Science of Brock University and IBM Canada
September 2008

ABSTRACT:

We present a set of highly parallel (fine grain), general pattern classification algorithms aimed at mimicking pattern identification, classification and learning skills of animals. We cover first an idealized case of supervised learning from an infallible expert, followed by the realistic cases of learning from fallible experts as well as the case of autonomous (i.e. self-supervised) learning. To ensure wide range of applicability in our algorithms we use only the basic concepts of mathematics: set theory and theory of metric spaces. The proposed methodology allows for creation of expert systems capable of continuous learning and forgetting of less useful facts, and so to maintain their adaptation to an environment evolving sufficiently slowly.

As proof of concept we offer a sketch of automated deep sky observation in search for unknown objects together with a detailed solution to the problem of automated mineral identification in petrographic thin sections. These illustration problems seem intractable using traditional means.

KEY WORDS:

Sensory perception; machine perception; continuous machine learning; adaptive systems; evolution; massive parallelism; pattern recognition; pattern clustering; pattern classification; visual cortex; cortex simulation; fault tolerance; mineral identification.

1. Adaptation to the environment through learning

Given the existence of a plethora of pattern recognition and classification algorithms [1], [2], [3], [4], [5], an obvious question emerges: Do we really need some more?

As it turns out, most algorithms group their computations into three fundamental steps: (a) preprocessing, (b) feature extraction, and (c) classification. For example, anglers keen on distinguishing between various species of salmonids classify their catch by size, number and position of fins, number of rays in a particular fin, etc. A similar procedure would be followed by a sorting machine in a fish packing plant, performing automated sort by species of fish arriving on a conveyor belt.

A troubling issue emerges here: On what grounds have the designers of the automated fish sorting algorithm chosen these features, and not the fish colour, the colour of their gill covers and / or of

blood, or presence / absence of heads and / or tails, etc.? A superficial answer might be: Because the former set of features allows classifying the salmonids by species, while the latter set does not.

We usually overlook here that the designers performed the preliminary classification of fish features into two groups: these useful in classifying fish by species and those that are not. That higher-level feature classification knowledge is not built into the fish-sorting machinery; that machinery cannot learn.

It is no wonder that such incomplete algorithms perform rather well in their intended (and very narrow) context, and are utterly useless otherwise. In fact, we cannot identify *a priori* any one of them to perform better than the others [1, p.454].

Unlike man-made machines, animals (including humans) have keen classification and learning skills – needed for survival. It is highly doubtful that fish or mice use a plethora of algorithms described in literature with an uncanny *a priori* knowledge of which one to invoke in a given situation. It is much more plausible that they use one, highly parallel (fine-grain) general algorithm that allows them to classify objects in their environment in real time and so ensure their survival as a species. No algorithm is perfect – so the classification errors can and will occur. If an animal survives such an error, it then may upgrade its algorithm's knowledge base (introducing a new reference pattern into a known pattern category, or introducing the first reference pattern into a new category) in order to avoid unpleasant surprises in the future. The animal can and will learn.

The human visual cortex consists of billions of neurons connected together in highly regular fashion and carrying out vast number of repetitive operations. The number of processing elements (neurons) is at least an order of magnitude higher than the number of sensors (rods and cones) on our retinas, which exceeds one hundred million. In this paper we are about to reveal part of the mystery how from this massively-parallel but highly repetitive activity an experience as rich as the perception of our environment can emerge.

In contradiction with the above, our current pattern classification machinery utilizes many more sensors than processing elements. We (the authors) cannot help but feel that the current machine pattern classification algorithms are vastly oversimplified, and that further technological advances both in parallel hardware and in algorithms are needed.

If is worthy of note that the neural pathway between our retinas and the human visual cortex is of dual nature. One part of the pathway, containing the superior colliculus (SC), is responsible for spatial perception, i.e. for our understanding *WHERE* objects are. The other part, containing the lateral geniculate nucleus (LGN) is responsible for object classification, i.e. for determining *WHAT* are the objects we see. People with damaged SC lose spatial perception, but are capable of identifying *WHAT* objects they see. People with damaged LGN cannot classify objects (i.e. lose the *WHAT* function), but retain the *WHERE* function. Further details about human perception can be found at [6, p.112-114].

We follow these hints and recommend data preprocessing in two different ways – one needed to determine where objects are, and the other needed for object classification.

Today, in machine vision (robotics, etc.) the *WHERE* function is no longer a big technical problem anymore. The generality of the *WHAT* function remains a mystery. Our algorithms partially lift the veil shrouding that mystery.

To ensure generality, pattern classification algorithms should use only tools lying at the foundation of mathematics. We use only set theory and the theory of metric spaces. The obtained algorithms are universally applicable. We present a sketch of their applicability by suggesting applications to astronomy (deep sky observation and search for unknown objects) and by a detailed illustration of application to geology (automatic mineral identification in petrographic thin sections). This is a non-exhaustive list, of course, but these illustrations offer solutions to problems that seem currently intractable. In order to fully appreciate the potential of our algorithms the reader is encouraged to consider its impact on automated manufacturing, space exploration, computer assisted medical diagnosis, etc.

2. A quick note on metric spaces

Intuitively, a metric space is a set of objects, typically called "points", between which a way of measuring the distance has been defined. In everyday life we use Euclidean distance; however, this is only one of the many possible ways of measuring distances.

Let $\mathbf{S}$ be our space of interest.

Def. 1. Measure of distance: Any real-valued function $d : \mathbf{S} \times \mathbf{S} \rightarrow \mathbf{R}$ can be used as a measure of distance, provided that for all $x, y, z \in \mathbf{S}$ it has the following properties:

(2.1) $d(x, x) = 0$
(2.2) if $x \neq y$ then $d(x, y) > 0$, i.e. the distance between two distinct points is positive;
(2.3) $d(x, y) = d(y, x)$, i.e. the distance does not depend on the direction of measurement;
(2.4) $d(x, z) \leq d(x, y) + d(y, z)$, i.e. the distance cannot be diminished by measuring it via some intermediate point.

The last property is frequently called the *triangle law*, because the length of each side of a triangle cannot be greater than the sum of lengths of two other sides.

Example: Consider a two-dimensional space $\mathbf{R^2}$ and two points $a = \langle x_1, y_1 \rangle$ and $b = \langle x_2, y_2 \rangle$.
We may define distance $d(a, b)$ as:

(2.5) Euclidean distance: $d_1(a, b) = \text{sqrt}((x_1 - x_2)^2 + (y_1 - y_2)^2)$ or
(2.6) Manhattan distance: $d_2(a, b) = |x_1 - x_2| + |y_1 - y_2|$ or
(2.7) perhaps simply as $d_3(a, b) = \max\{|x_1 - x_2|, |y_1 - y_2|\}$

All three functions d_1, d_2, and d_3 meet the necessary requirements (2.1), (2.2), (2.3) and (2.4).

Def. 2. A **metric space** is the configuration $\langle \mathbf{S}, d \rangle$, where $\mathbf{S}$ is a set of points, and d is some measure of distance between them.

In our study we will be frequently interested in measuring distance between subsets of **S**, rather than between points of **S**. One of the simplest subsets of **S** is a *ball*.
Def. 3. A **ball** of radius $r \geq 0$ around the point $c \in \mathbf{S}$ is the set

(2.8) $$\{ x \in \mathbf{S} \mid d(c, x) \leq r \}$$

which we will denote as $B(c, r)$ without mentioning either **S** or d when confusion can be avoided. The point c is called the center of the ball B.

Note: In older textbooks the term "sphere" is frequently used instead of a "ball". This usage is currently being phased out, as we prefer now the "sphere" to mean the surface of a "ball".

Observe that the "shape" of a ball depends on the way we measure distance, as per Fig.1.

It is worth noticing that the "volumes" of such balls may depend on the metrics used. In particular, the Manhattan ball (b) is most specific about its center, i.e. it has the smallest "volume" (i.e. area in $\mathbf{R}^2$), and also its metric function computes faster than metrics (a) and (b). This is important, given that our pattern recognition algorithms will be very intensive computationally.

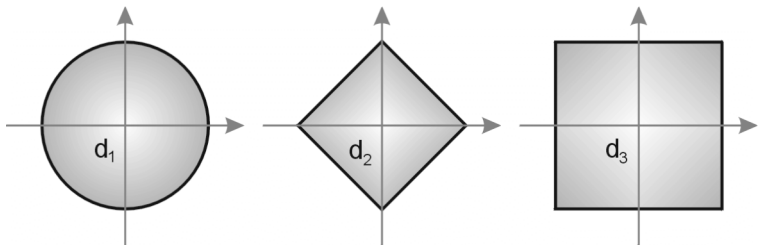

Fig.1. The balls in $\mathbf{R}^2$ of same radius, with center in the origin of the system of coordinates, defined using metrics d_1, d_2, and d_3 in the previous example. Similar balls in $\mathbf{R}^3$ would be a sphere, an octahedron, and a cube, correspondingly. Generalizations of balls in higher-dimensional spaces are conceptually straightforward, although not easily drawn.

A more detailed analysis of metric spaces can be found at [7].

3. A new method for computing distances between sets

Intuitively: Given any two non-empty sets $A, B \subset \mathbf{S}$ we need to construct a function $D(A, B)$ to measure distance between A and B. That function should retain the properties (2.1), (1.2), (2.3) and (2.4). In particular, observe that the properties (2.1) and (2.2) imply that $D(A, B) > 0$ even if the sets A and B touch (i.e. have at least one common element), or perhaps one of them contains another (say, $A \subset B$). In fact, we must have $D(A, B) = 0$ if and only if $A = B$.

Let d be our chosen function for measuring distance between points of space **S**. We will use that function to construct our function D.

Def.4. Distance between a point and a set: Let $\langle \mathbf{S}, d \rangle$ be a metric space and let $A \subset \mathbf{S}$ be a non-empty set. A distance between a point $x \in \mathbf{S}$ and A, denoted $\delta(x, A)$ is given by

(3.1) $$\delta(x, A) = \inf \{ d(x, a) \mid a \in A \}$$

Intuitively, it is the distance between x and a point $a \in A$ closest to x. Using δ, we can introduce the concept of a hyperball:

Def.5. A hyperball of radius $r \geq 0$ around a set $A \subset \mathbf{S}$ is the set

$$\{ x \in \mathbf{S} \mid \delta(x, A) \leq r \} \tag{3.2}$$

Observe some properties of the hyperball:

- The hyperball reduces to the familiar ball around point $a \in \mathbf{S}$, if the set $A \subset \mathbf{S}$ consists of a single point, i.e.: $A = \{a \in \mathbf{S}\}$;
- The hyperball reduces to the center set $A \subset \mathbf{S}$ when its radius reduces to zero, i.e.: $\{ x \in \mathbf{S} \mid \delta(x, A) = 0 \} = A$.

In other words, the concept of the hyperball is a generalization of the familiar concept of a ball. For the sake of brevity in the rest of this paper we will therefore use the single term "ball" in the broader, hyperball sense.

Let us also define

Def.6. Pseudo-distance between two sets: Let $\langle \mathbf{S}, d \rangle$ be a metric space and let $A, B \subset \mathbf{S}$ be two non-empty sets. A pseudo-distance from A and B, denoted $\Delta(A, B)$ is given by

$$\Delta(A, B) = \sup \{ \delta(a, B) \mid a \in A \} \tag{3.3}$$

That is, pseudo-distance from A and B is the distance from the most distant point of A (from B) to B. It is not a distance, but merely pseudo-distance, because it is directional, i.e. the property (2.3) does not hold in general, given that $\Delta(A, B) \neq \Delta(B, A)$. However, we can quickly remedy this by applying a quick technical fix:

Def.7. Distance between two sets: Let $\langle \mathbf{S}, d \rangle$ be a metric space and let $A, B \subset \mathbf{S}$ be two non-empty sets. A distance between A and B, denoted $D(A, B)$ is given by

$$D(A, B) = \max \{\Delta(A, B), \Delta(B, A)\} \tag{3.4}$$

Observe that with the $\mathcal{WHERE}$ information regarding the spatial positioning of sets deliberately destroyed through preprocessing (i.e. using linear transformations), the function D can be seen as measure of dissimilarity between sets. $D(A, B) = 0$ implies $A = B$. The function D can be also used as a pattern classifier. With properly selected small value of ε, the condition $D(A, B) \leq \varepsilon$ implies that A and B are sufficiently similar to be included in the same category.

4. Coding notation and fundamental pseudocodes

It is impossible to represent continuous sets or spaces on any digital computer, therefore we must limit ourselves to sets consisting of a finite number of points. This limitation allows us, however, to inspect such sets in finite number of steps (and so in finite time).

Similarly, a digital computer can represent exactly only integers, as long as they lay in a finite range. Representation of other values (real, complex, etc.) is only approximate. Therefore, in the domain of digital computers our pseudocodes might deal exclusively with integers without loss of computability. We will adhere to the existing convention and use the "real" types as well, keeping these limitations in mind. The terms we will use, like "the smallest positive real value" have a proper meaning only within that context.

4.1. Pseudocode notation

We use standard notation, enhanced only in order to express parallelism and mutual exclusion to protect write access to shared variables. The notation

```
parbegin
   statement1;
   statement2;
   statement3;
parend;
statement4;
```

implies that statements 1, 2 and 3 are to be executed in parallel, and the execution of statement 4 may begin only after all the three preceding statements in the **parbegin** block have terminated. Each of these three statements may be simple or a composite (i.e. a block). A composite statement is made of several statements (optionally preceded by declarations) and enclosed within **begin** and **end**.

For example, let us write the pseudocode of the parallel function computing Euclidean distance between points in a 3D space. In this space each point $P = \langle x, y, z \rangle$ is an ordered triple, where x, y, z are the coordinates of P. In the pseudocode the notation `P.x` means the x –coordinate of point P.

```
function d1(P1, P2 : Point) return real
is
   disx2, disy2, disz2 : real;
begin
   parbegin
      disx2 := square(P1.x – P2.x);
      disy2 := square(P1.y – P2.y);
      disz2 := square(P1.z – P2.z);
   parend;
   return sqrt(disx2 + disy2 + disz2);
end d1;
```

Several statements executing in parallel may attempt to modify some shared variables. To ensure integrity of such data we will enforce mutual exclusion of write access by using semaphores. A semaphore can store only non-negative values and can be initialized to any such value. Only two operations on a semaphore are possible:

```
signal(s);      -- increments the value of semaphore s by 1
wait(s);        -- decrements the value of semaphore s by 1 as soon as
                -- it is possible to do so without yielding s negative
```

More info on handling concurrency issues, such as access to shared variables, use of mutual exclusion and semaphores can be found in [8], [9].

Using a mutual exclusion semaphore we may provide pseudocode for the metric (2.7), but between points in a 3D space:

```
function d3(P1, P2 : Point) return real
is
   distance :   real := 0;
   mutex : semaphore := 1; -- semaphore initialized to 1
begin
   parbegin
      begin disx : real := abs(P1.x – P2.x);
            wait(mutex);
            if distance < disx then distance := disx; end if;
            signal(mutex);
      end;
      begin disy : real := abs(P1.y – P2.y);
            wait(mutex);
            if distance < disy then distance := disy; end if;
            signal(mutex);
      end;
      begin disz : real:= abs(P1.z – P2.z);
            wait(mutex);
            if distance < disz then distance := disz; end if;
            signal(mutex);
      end;
   parend;
   return distance;
end d3;
```

In this solution some sections of the code execute in parallel, but updating of the shared variable `distance` is protected by mutual exclusion. The three updates can be performed in arbitrary order. The scope of variables `disx, disy, disz` and of the semaphore `mutex` is the block within which they are declared, and its inner blocks.

In our pseudocodes we will also use the **`parfor`** statement. The notation

```
parfor i in 1..10 do
   worker(i);
parend;
statement;
```

represents simultaneous launching of ten instances of the procedure `worker`, each with its own argument. It resembles the familiar **`for`** loop, but here the "iterations" are executed in parallel. As before, the execution of `statement` at end of this pseudocode may begin only after all ten worker processes in the **`parfor`** block have terminated.

Finally, the familiar **`return`** statement is used to immediately terminate the execution of the procedure or function containing it, while returning the execution control to the calling environment. Additionally it terminates all parallel execution threads that were created as a result of calling the procedure or function. Examples of use follow – see pseudocodes (6.8), (7.1). If used within a function, it also returns its computed value.

Observe that our pseudocodes are perfectly suitable for execution on a uniprocessor machine. The **parfor** block reduces then to the usual **for** loop, while the statements within a **parbegin** block can be executed in any order.

5. Pseudocodes for computing distances between sets

The digital implementation of the set A being the argument of the function $\delta(x, A)$ defined in (3.1) is a finite set of n points

(5.1) $$A = \{ a_1, a_2, \ldots a_n \}$$

Using an arbitrary function d for computing distances between points, we can write the following pseudocode for computing $\delta(x, A) = \inf \{ d(x, a) \mid a \in A \}$ defined in (3.1):

(5.2)
```
function δ(x : Point; A : Set) return real
is
   distance : real      := ∞; -- initialized to infinity
   mutex    : semaphore := 1;
begin
   parfor i in 1 .. card(A) do -- card(A) is the number of elements in A
      dis : real := d(x, a(i));
      wait(mutex);
      if dis < distance then distance := dis; end if;
      signal(mutex);
   parend;
   return distance;
end δ;
```

We are now ready to define the pseudocode for $\Delta(A, B) = \sup \{ \delta(a, B) \mid a \in A \}$ given in (3.2) as a pseudo-distance between two sets $A = \{ a_1, a_2, \ldots a_n \}$ and $B = \{ b_1, b_2, \ldots b_m \}$:

(5.3)
```
function Δ(A, B : Set) return real
is
   distance : real      := 0;
   mutex    : semaphore := 1;
begin
   parfor i in 1 .. card(A) do -- card(A) is the number of elements in A
      dis : real := δ(a(i), B);
      wait(mutex);
      if dis > distance then distance := dis; end if;
      signal(mutex);
   parend;
   return distance;
end Δ;
```

Finally the distance D(A, B) between two sets defined in (3.4) as $D(A, B) = \Delta(A, B) + \Delta(B, A)$ can be implemented simply as:

(5.4)
```
function D(A, B : Set) return real
is
   disAB, disBA : real;
```

```
begin
   parbegin
      disAB := Δ(A, B);
      disBA := Δ(B, A);
   parend;
   return disAB + disBA;
end D;
```

Following this logic, an arbitrary function d used to measure distance between points of a metric space has the properties (2.1), (2.2), (2.3) and (2.4) and induces another function D that can be used to measure distances between non-empty subsets of that space. That new function D also meets the requirements (2.1), (2.2), (2.3) and (2.4).

Furthermore, if the function d can be implemented on a digital computer, then the function D can also be implemented on that machine, as per pseudocodes (5.2), (5.3) and (5.4).

In our further analysis we therefore will only need to show how to implement the function d on a digital computer to assume the existence of the function D.

Armed with these concepts, we can now present our theory.

6. The general algorithm for pattern recognition and object classification

Machine learning algorithms fall into two groups: supervised and unsupervised (a.k.a. autonomous, i.e. self-supervised) learning algorithms. The main distinction here is that in the first category the intelligent (but not yet knowledgeable) systems can learn from a training expert, while in the second category the intelligent systems can learn only from their own mistakes, and only if they prove non-fatal.

The theory of evolution of biological systems tells us that luck plays a considerable role in the survival of a system. However, the probability of survival for a given period of time increases as the knowledge about the environment of the system increases. More complex systems in their initial stages of life benefit from having an expert trainer (a.k.a. parent) and assume autonomous life only after having acquired enough knowledge to significantly reduce the probability of committing a fatal error. More about this follows in the section (6.4) about autonomous learning.

Our learning algorithms can be used in supervised and autonomous learning modes. The implementation details of an intelligent system (a robot) and its environment will dictate whether the system will consult a trainer when facing an error or a "don't know" situation, or whether it will resort to experimentation using all its senses in order to learn.

A single sense – like vision– as one of the senses – is not enough for learning. The biological systems have a number of senses, and the data they gather are useful in learning using several senses in a complementary fashion. Humans learned to use specialized equipment to enhance their senses.

For example: Most of us have learned that mohair sweaters are soft and cuddly. When seeing mohair we tend to anticipate the softness, while forgetting that softness is an experience gathered

through the sense of touch. Should we encounter a particular a mohair-like fabric that would be harsh to the touch, we would be surprised and treat this event as an opportunity for learning more about mohair-like fabrics. In this way systems equipped with several senses can use them in a voting fashion: the experiential inconsistencies that could otherwise be fatal now may become the opportunities to learn.

Another example: When young we learned about fruits. We encountered such things as cherries and plums. Question arose: Do these fruit specimen belong to the same category? They all grow on trees, attached via stems, have stones inside, are more-or-less round, are green when not ripe, acquire reddish or yellowish hues when ripe. Large cherries are roughly of the same relative size as small plums, etc. It is impossible to decide purely visually whether they should be grouped together or not. It is the senses of touch, taste, different consequences emerging from eating too many of each, that made us decide that it is worthwhile to keep in our knowledge base two distinct categories of "cherries" and "plums".

A bit later, perhaps, we encountered another fruit (which we call now "olives"). A question arose: were they variants of "cherries", or "plums", or still something else? It is through non-visual means that we decided on creating a new category of "olives" in our knowledge base.

In our algorithms we use a knowledge bank K being a set of categories $K = \{ C_1, C_2, \ldots C_k \}$. All categories $C_1, C_2, \ldots C_k$ represent classes of objects (viz. "plums", "cherries", "olives", etc.) They are non-empty – each category contains balls centered on patterns representative of objects belonging to it. The categories are numbered with an index spanning the values $1, 2, \ldots k$.

Shortly we will offer several pseudocodes for supervised and autonomous machine learning. Pattern classification is their common component.

6.1. Pattern classification

The classifier function has the following specification:

```
function Classifier (P : Pattern; K : KnowBank) return CatIndex;
```

It inspects in a massively parallel way all categories in the knowledge bank for balls which might contain a given pattern `P`. Should such a ball be found, the index of the category to which the ball belongs is returned; otherwise 0 (for "unknown") is returned. The pseudocode is:

(6.1)
```
function Classifier (P : Pattern; K : KnowBank) return CatIndex
is
begin
   parfor k in 1 .. card(K) do          -- inspecting all categories:
      parfor i in 1 .. card(K.C(k)) do -- inspecting all balls in C(k):
         if D(P, K.C(k).B(i).P) ≤ K.C(k).B(i).r then  -- category found:
            return k;
         end if;
      parend;
   parend;
   return 0; -- pattern unknown
end Classifier;
```

To fully appreciate the inherent parallelism of this classifier function recall that the function

$D(P_1, P_2)$ used to distinguish between patterns P_1 and P_2 is itself massively parallel.

6.2. Supervised learning from an infallible expert

In this idealized situation we assume that the learning system has access to a training expert who never makes mistakes (a "god" of some sort). Whenever the system is unable to classify a given pattern, it consults its perfect expert who instructs the system to insert in its knowledge bank the ball $B(P, r)$ of given radius and centered on a specified pattern.

The procedure used for machine learning has therefore the following specification:

```
procedure Learn (P   : in Pattern;  r : Radius;
                 idx : in CatIndex; K : in out KnowBank);
```

The procedure accepts a given pattern `P` as belonging to a category number given in `idx`, and updates the proper category of the knowledge bank `K` with a ball $B(P, r)$. The pseudocode is:

(6.2)
```
procedure Learn (P   : in Pattern;  r : Radius;
                 idx : in CatIndex; K : in out KnowBank)
is
   B : PatBall'(P, r);     -- our new ball has radius r around pattern P
   k : Natural := card(K); -- k is the current number of categories in K
begin
   if k < idx then – we need to create new category
      k := k+1;
      Create new empty category C(k):= Ø;
      Insert C(k) into K;
   else
      k := idx;
   end if;
   Insert B into K.C(k);
end Learn;
```

We demand here of the perfect expert to have the perfect knowledge and so to select the smallest possible number of optimal balls $B_i(P, r)$ characteristic of a given category C_{idx}. The radii of these balls are strategically chosen for the representation of that category to be as concise as possible.

The values of the category index `idx` always fall within the range `1 .. card(K)+1`, so that the number of categories in the knowledge base increases sequentially, one-by-one.

6.3. Supervised learning from a fallible expert

Perfect experts are hard to come by. Available experts occasionally commit mistakes, which require later corrections. Typical expert mistakes:

- Training balls may not be centered on patterns of a given category;
- Training patterns may not be the most representative of a given category;
- Training ball radii may not be as large as possible for given training patterns, or
- Training ball radii may overshoot the maximum values for given training patterns.

There is no remedy, should a learning system choose to follow a false expert, i.e. an expert committing too many mistakes too frequently. However, with mistakes sufficiently rare, there is hope. Our learning procedure, to be invoked as usual when the classifier function faces an unknown pattern, has to be modified in order to allow it to eliminate the knowledge imperfections as learning progresses:

(6.3)
```
procedure Learn (P   : in Pattern;  r : Radius;
                 idx : in CatIndex; K : in out KnowBank)
is
   B : PatBall'(P, r);     -- our new ball has radius r around pattern P
   k : Natural := card(K); -- k is the current number of categories in K
   ε : constant Radius := initialized to smallest positive value;
begin
   if k < idx then – we need to create new category
      k := k+1;
      Create new empty category C(k):= Ø;
      Insert C(k) into K;
   else
      k := idx;
   end if;
   Insert B into K.C(k);                    -- housekeeping chores follow:
   Sort balls in K.C(k) in descending order of radius;
   Remove redundant balls from K.C(k) if any;

   parfor n in 1 .. card(K) and n ≠ k do -- remove other categ. conflicts:
      parfor i in 1 .. card(K.C(n)) do
         if D(P, K.C(n).B(i).P) ≤ r then
            Remove ball K.C(n).B(i) from K.C(n); -- it conflicts with B
         else if D(P, K.C(n).B(i).P) ≤ r + K.C(n).B(i).r then
            K.C(n).B(i).r := D(P, K.C(n).B(i).P) – r – ε;
            -- the conflicting ball is reduced to proper size
            end if;
         end if;
      parend;
      Sort balls in K.C(n) in descending order of radius;
      Remove redundant balls from K.C(n) if any;
   parend;
   Remove empty categories from K if any;
end Learn;
```

Given that with a fallible expert we cannot be assured that the representation of each category is optimally concise, it is wise to sort the balls belonging to each updated category (in the descending order of radius) and try to remove redundant balls. Having done this we are certain that the balls $B_i(P, r)$, $i = 1, 2, \ldots n$ characteristic of a given category C_{idx} either stand apart or only partially overlap.

In the same vein, a check is to be made whether balls in other categories conflict with the ball just inserted into C_{idx}. If so, the radii of these balls are reduced in order to eliminate the conflict. If this is not possible, then the conflicting balls are removed from the knowledge base, thus eliminating prior learning errors.

Question is: Do animals perform these "housekeeping chores" while asleep? If this were the case, then their sleep should have several phases, or "rhythms" …

Another interesting aside: Observe that an intelligent system learning from a cautious expert, who routinely underestimates the ball radii to "play it safe" may attempt a generalization, in order to come up on its own at a more concise representation of a given category. The problem is as follows:

> Given a cluster of balls $B_i(P, r)$, $i = 1, 2, \ldots m$ that partially overlap, inscribe within them the largest possible ball $B(P^*, r^*)$, centered on a strategically chosen pattern P^* and of a maximum possible radius r^* so as to make redundant as many cluster balls as possible. These balls should be removed from the cluster and replaced with $B(P^*, r^*)$.

We do not offer the solution to this problem here, but merely point out that repeated application of this simplification procedure leads to making the representations of each category more concise, in spirit of Occam's razor.

In any case, our training expert will make occasional mistakes, so we need a procedure to insert corrections into our knowledge base when errors are discovered. The specification is:

```
procedure Fix (P   : in Pattern;  r : Radius;
               idx : in CatIndex; K : in out KnowBank);
```

Procedure `Fix` allows the expert to declare that notwithstanding his wisdom some of his previous pronouncements were wrong and the ball $B(P, r)$ should really belong to category C_{idx} in the knowledge bank K, or cover patterns still unknown, if `idx = 0`. Of course the legal values for the category index `idx` are `0 .. card(K)` here. The pseudocode is:

(6.4)
```
procedure Fix (P   : in Pattern;  r : Radius;
               idx : in CatIndex; K : in out KnowBank)
is
   B : PatBall'(P, r);     -- our ball has radius r around pattern P
   bad : CatIndex := Classifier (P, K); -- identifies bad category
   ε : constant Radius := initialized to smallest positive value;
begin
   parfor i in 1 .. card(K.C(bad)) do -- correct info in the bad cat.:
      if D(P, K.C(bad).B(i).P) ≤ r then
         Remove ball K.C(bad).B(i) from K.C(bad); -- conflicts with B
      else if D(P, K.C(bad).B(i).P) ≤ r + K.C(bad).B(i).r then
         K.C(bad).B(i).r := D(P, K.C(bad).B(i).P) - r - ε;
         -- we have just shrunk the conflicting ball to proper size
         end if;
      end if;
   parend;
   if idx > 0 then -- update info in the correct category:
      Insert B into K.C(idx);
      Sort balls in K.C(idx) in descending order of radius;
      Remove redundant balls from K.C(idx) if any;
      -- getting rid of possible new conflicts with other categories:
      parfor n in 1 .. card(K) and n ≠ idx do
         parfor i in 1 .. card(K.C(n)) do
            if D(P, K.C(n).B(i).P) ≤ r then
               Remove ball K.C(n).B(i) from K.C(n); -- it conflicts with B
            else if D(P, K.C(n).B(i).P) ≤ r + K.C(n).B(i).r then
               K.C(n).B(i).r := D(P, K.C(n).B(i).P) - r - ε;
               -- we have just shrunk the conflicting ball to proper size
               end if;
```

```
            end if;
          parend;
          Sort balls in K.C(n) in descending order of radius;
          Remove redundant balls from K.C(n) if any;
        parend;
      end if;
      Remove empty categories from K if any;
end Fix;
```

If this appears a little complicated, we have a consolation: This scheme allows intelligent systems to learn from each other, not only from their fallible experts. A system being particularly reliable in identifying patterns of a given category may share its definition of that category with other systems. Such reciprocal cooperation among learning systems may lead them to eventually outperform their fallible experts.

Observe that with infallible experts, the systems had no need nor incentive to learn from each other, because the information they obtained from their perfect experts could not be improved upon.

6.4. Autonomous learning

In the beginning, there were no experts. How then the first intelligent systems came to be?
An unsupervised, intelligent system has no choice but to resort to observation and experimentation in order to learn. This is dangerous: too much exposure to the unknown can kill. That is why most animals today flee the unknown. It takes big brains to handle the unknown.

When experimenting, a system can observe patterns but can only take guesses at the ball radii. A system can be a slow or a fast learner. A slow learner prefers to err on the side of caution by underestimating the radii values, i.e. “rote learning”. This is relatively safe, but requires large memory capacity, since the specifications of categories will contain very many small balls. That situation can later be remedied by subsequent generalizations, i.e. attempts to inscribe large balls within clusters of many small, partially overlapping balls. It looks like evolution has chosen this path: The first learners were slow, but luckily there were very many of them – they were simple organisms. There were so many of them that some of them were lucky to survive and refine their adaptation to the environment. After all there is nothing very special about intelligence and knowledge: these are merely some of many forms of adaptation to the environment.

We (the humans) want to build robots that would be autonomous and fast learners. Such robots should prefer to overestimate the ball radii. A little knowledge is a dangerous thing, and our robots may have a tendency of “jumping to conclusions”. To avoid that we will have to build in a fair amount of knowledge into them, before turning our robots loose. The knowledge of a sufficient number of categories puts natural limits on the maximum values of the new ball radii: New balls must stand apart from balls belonging to other categories.

In fact we can put an a priori limit r_{max} on the maximum radius of a ball guessed by a robot. A low value of r_{max} would make a robot cautious and shy when facing the unknown, a large value of r_{max} may make it courageous, or even foolish. Probably an optimal value r_{max} exists that would make a robot cautious enough but still willing to take risks of experimentation with the unknown. A knob on the back of a robot adjusting its r_{max} value can be seen as controlling its “personality”.

There is no need to put the r_{max} limit on the inferred balls when trying to generalize by inscribing larger ball into a cluster of overlapping smaller balls, as such generalizations lead to simplifications and clarifications of the knowledge base.

The procedure used for autonomous machine learning has therefore the following specification:

```
procedure Learn (P   : in Pattern;
                 idx : in CatIndex; K : in out KnowBank);
```

The procedure accepts a given pattern `P` as belonging to a category number given in `idx`, and updates the proper category of the knowledge bank `K` with a ball $B(P, r_{max})$, or a smaller ball if the value r_{max} would make this ball conflict with a ball from another category. The pseudocode is:

(6.5)
```
procedure Learn (P   : in Pattern;
                 idx : in CatIndex; K : in out KnowBank)
is
   ε     : constant Radius := initialized to smallest positive value;
   rmax  : constant Radius := initialized to a suitable value;
   mutex : semaphore := 1;
   B : PatBall'(P, rmax);  -- our new candidate ball around pattern P
   k : Natural := card(K); -- k is the current number of categories in K
begin
   if k < idx then – we need to create new category
      k := k+1;
      Create new empty category C(k):= Ø;
      Insert C(k) into K;
   else
      k := idx;
   end if;
   parfor n in 1 .. card(K) and n ≠ k do -- trimming the new ball radius:
      parfor i in 1 .. card(K.C(n)) do
         r : Radius := D(P, K.C(n).B(i).P) - K.C(n).B(i).r – ε;
         wait(mutex);
         if B.r > r then B.r := r; end if;
         signal(mutex);
      parend;
   parend;
   Insert B into K.C(k);                  -- housekeeping chores follow:
   Sort balls in K.C(k) in descending order of radius;
   Remove redundant balls from K.C(k) if any;
end Learn;
```

As a matter of routine now we have to define the procedure `Fix` that would allow the learner to correct its own previous mistakes. The specification is:

```
procedure Fix (P   : in Pattern;
               idx : in CatIndex; K : in out KnowBank);
```

This procedure is to be called in similar circumstances to its sister procedure (7.4). We have to deal with two situations here: (a) the pattern `P` has been misclassified and should remain unknown (i.e. `idx = 0`), or (b) the misclassified pattern `P` belongs really to another category (`idx > 0`).

In the first case, we just shrink balls of the misclassified category to exclude P. In the second case, we need to insert a strategically sized ball centered at P into the correct category and to reduce selected balls in other categories to avoid conflict with the new ball. The pseudocode is:

(6.6)
```
procedure Fix (P   : in Pattern;  r : Radius;
               idx : in CatIndex; K : in out KnowBank)
is
   ε     : constant Radius := initialized to smallest positive value;
   rmax  : constant Radius := initialized to a suitable value;
   mutex : semaphore := 1;
   B : PatBall'(P, rmax);  -- our ball has radius rmax around pattern P
   bad : CatIndex := Classifier (P, K); -- identifies bad category
begin
   if idx = 0 then -- pattern P should remain "unknown" :
      parfor i in 1 .. card(K.C(bad) do
         r : Radius := D(P, K.C(bad).B(i).P) - ε;
         if K.C(bad).B(i).r > r then K.C(bad).B(i).r := r; end if;
         if K.C(bad).B(i).r < 0 then
            wait(mutex); Remove B(i) from K.C(bad); signal(mutex);
         end if;
      parend;

      Sort balls in K.C(bad) in descending order of radius;
      Remove redundant balls from K.C(bad) if any;
   else            -- pattern P should belong to K.C(idx) :
      parfor n in 1 .. card(K) and n ≠ idx do -- sizing new ball properly:
         parfor i in 1 .. card(K.C(n)) do
            r := Radius := D(P, K.C(n).B(i).P)/2;
            wait(mutex);
            if B.r > r then B.r := r; end if;
            signal(mutex);
         parend;
      parend;
      Insert B into K.C(idx);
      Sort balls in K.C(idx) in descending order of radius;
      Remove redundant balls from K.C(idx) if any;
      parfor n in 1 .. card(K) and n ≠ idx do -- eliminating conflicts:
         parfor i in 1 .. card(K.C(n)) do
            r := Radius := D(P, K.C(n).B(i).P) - B.r - ε;
            if K.C(n).B(i).r > r then K.C(n).B(i).r := r; end if;
            wait(mutex);
            if K.C(n).B(i).r < 0 then Remove B(i) from K.C(n) end if;
            signal(mutex);
         parend;
         Sort balls in K.C(n) in descending order of radius;
         Remove redundant balls from K.C(n) if any;
      parend;
   end if;
   Remove empty categories from K if any;
end Fix;
```

As in the case of supervised learning from a fallible expert, the intelligent systems can also learn from each other by comparing notes about category definitions.

This concludes our theory of reasoning about and classifying visual patterns. Before proceeding to illustrative examples of applicability, one final clarification:

Observe that in our theory each pattern can belong to one category and one category only. What about hierarchical categories? We contend that those are not observed visually, but inferred.

For example: The category "person" can contain patterns belonging to categories "man", "woman" and "child". However, we do not visually perceive persons. We perceive only men, women and children. This does not prevent us from constructing an abstract category "person" which would contain all balls from "man", "woman" and "child". We can mentally simplify the category "person" even further, by sorting the balls, removing redundant balls, and then by inscribing into its cluster of remaining balls some large, strategic balls allowing us to eliminate smaller, original balls from "man", "woman" and "child".

The category "person" remains abstract, nevertheless.

7. Case study #1: Recognition of minerals in petrographic thin sections

The following study is an example of supervised learning from a fallible expert – how else?

7.1. Thin sections and petrographic microscopes

A rock is a collection of mineral grains. Geologists frequently study structure and composition of rocks under polarizing microscopes. Such a microscope consists of a light source, of a stage on which rock samples are placed and of an optics system to perform observations.

The samples observed under such microscopes are thin (approx. 30 μm) slivers of rock. Fig.2. illustrates the process of making a petrographic thin section. At that thickness, most mineral grains appear translucent.

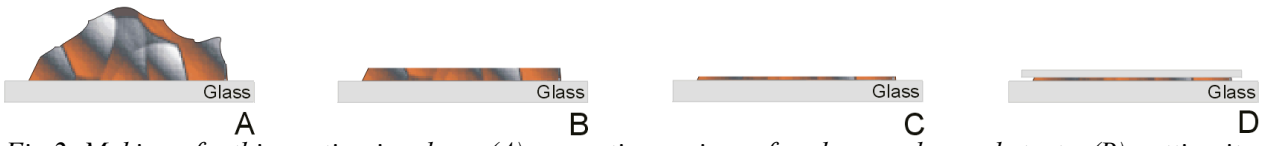

Fig.2. Making of a thin section involves: (A) cementing a piece of rock on a glass substrate, (B) cutting it and then (C) lapping it to the desired thickness, and finally (D) covering it with a protective thin glass.

Given that most minerals are birefringent, it is easier to identify them by observing their grain crystals in polarized light. A polarizing material is inserted between the light source and the stage.

We may insert another polarizer between the sample and the microscope optics, and orient it perpendicularly to the first polarizer. All isotropic materials will appear black now, but anisotropic minerals will take on an appearance as per Fig.3. Most microscopes allow rotating the stage, and so to vary the polarization angle of the light beam traversing the thin section. When this angle is varied, the mineral grains tend to shimmer in various colours. A trained geologist is capable of identifying the mineral grains based on their appearance.

It is of interest to the mining and drilling industries, to space exploration, etc., to be able to automate this process. To that end Dr. Frank Fueten of Brock University has designed a novel

microscope with a stationary stage and rotating polarizers [10], [11]. A thin section viewed through this microscope remains stationary, but the grains shimmer as the polarizers are rotated. In short, Dr. Fueten has accomplished the first step in automating this process: his equipment performs the pre-processing by separating the *WHAT* from the *WHERE*.

Fig.3. A thin section between two crossed polarizers.

Our algorithm allows us to complete the process of automated mineral recognition, and so to gather information about the chemical properties of rocks.

7.2. Representation of petrographic samples

For the purposes of our application, a "petrographic sample" is a series of thin section images, captured using a modified petrographic microscope with rotating polarizers and equipped with a video camera. There will be N images taken in planarly polarized light, and N images taken using crossed polarizers. Every image in a series is taken after rotating the polarizers by one step. The microscope made available to us is capable of rotating the polarizers 180^0 in $N = 200$ steps.

Every image is rectangular and is composed of pixels of varying colours. Each pixel corresponds to a given location in a thin section. That location may belong to a certain mineral grain, or may lie on the grain boundary. If it belongs to a grain, we identify its mineral type. Our goal is to identify the mineral grains; we group the unidentified locations (and their pixels) into grain boundaries.

Consider a series of pixels of a chosen location belonging to consecutive images $1, 2, \ldots, N$. As the polarizers rotate, the pixels change colours. However, having rotated the polarizers 180^0, each final pixel is identical to the pixel we started with. In other words, the colours of a chosen sample location form a closed loop in the RGB space, when we rotate the polarizers 180^0. We postulate that every mineral generates a loop of characteristic shape, which can be used in mineral identification.

Let us formalize our approach.

7.3. Sample location (and pixel) representation

We represent pixels in terms of primary colours. The colour intensity $I = \{0, 1, 2, \ldots, 255\}$ is a byte. The pixel is a structure $p = \langle r, g, b \rangle \in I^3$ where r, g, b are intensities of red, green, and blue,

while I^3 is the discrete and finite colour space. Let us define the difference in appearance of two pixels as a "distance" $mh(p_1, p_2)$ between two pixels $p_1, p_2 \in I^3$ using the Manhattan metric

(7.1) $$mh(p_1, p_2) = | r_1 - r_2| + | g_1 - g_2| + | b_1 - b_2|$$

We stress here that this function is a measure of difference in the appearance of two pixels, and not the measure of the geometrical distance between two pixels in an image, which could be measured in many ways, including the usual (Euclidean) way. Given that mineral grains in a sample appear as sets of adjacent pixels, we use the function mh to build other functions to measure differences of appearance between sets of pixels.

7.4. Representation of locations and regions within a sample

Our approach is straightforward: Given an unknown grain X and sets of pixels defining the appearances of minerals $M_1, M_2, \ldots, M_k$, we identify X as one of $M_1, M_2, \ldots, M_k$. Specifically, which one? The one from which the appearance of X differs the least, and stays within a predefined tolerance.

What if the appearance of X does not resemble any of $M_1, M_2, \ldots, M_k$, within the predefined tolerances? In that case we may face two situations: (a) we have encountered a new mineral and some representation of X is to be inserted into the "knowledge bank" $M_1, M_2, \ldots, M_k$ of known minerals, or (b) we have encountered a new form of a mineral M_i, and its representation in the knowledge bank must be updated.

As said before, when the polarizers rotate 180^0, the collection of pixels in a sample corresponding to a given location will form a loop $\lambda = \langle p_1, p_2, \ldots p_N \rangle$. In fact, every location L in a sample corresponds to two loops: $L = \langle \lambda_p, \lambda_c \rangle$ obtained using plane light and crossed polarizers.

Henceforth, when speaking of a sample location, we will mean the pair of such loops. In that vein, a mineral grain is a set of adjacent sample locations of similar appearance.

We therefore need to define a function for comparing appearances (i.e. shapes in I^3) of loops, and use that function to compare appearances of locations, and of regions, i.e. sets of adjacent locations.

7.5. Manhattan differences between appearances of loops

Given that a loop is just a set of pixels, we proceed as in (3.1) to define a function $mhpl(p, \lambda)$ being the difference in appearance between a given pixel p and a loop λ as

(7.2) $$mhpl(p, \lambda) = \inf \{ mh(p, q) \mid q \in \lambda \}$$

where q is a pixel of λ most similar to p.

Then, the pseudo-difference in appearance pmhll of two loops λ_1 and λ_2 is

(7.3) $$pmhll(\lambda_1, \lambda_2) = \sup \{ mhpl (p, \lambda_2) \mid p \in \lambda_1 \}$$

where p is a pixel of λ_1 most differing from all pixels of λ_2.

Finally, the Manhattan difference in appearance $mhll(\lambda_1, \lambda_2)$ between two loops λ_1 and λ_2 simply becomes

(7.4) $$mhll(\lambda_1, \lambda_2) = pmhll(\lambda_1, \lambda_2) + pmhll(\lambda_2, \lambda_1)$$

7.6. Manhattan differences between appearances of sample locations

We have previously defined a sample location L in as a pair of loops: $L = \langle \lambda_p, \lambda_c \rangle$ obtained using plane light and crossed polarizers. Using previous findings we may define the Manhattan function $MH(L_1, L_2)$, being the measure of the difference of appearance between two sample locations L_1 and L_2 as

(7.5) $$MH(L_1, L_2) = mhll(\lambda_{p1}, \lambda_{p2}) + mhll(\lambda_{c1}, \lambda_{c2})$$

We proceed in this fashion and define the measure of differences of appearance of sample regions. After all, regions are merely sets of adjacent locations. We leave this as an exercise to the reader; in our Petrographic Image Recognition System (PIRS) we will use extensively the function MH.

At this stage it is clear that repeated calculations of the function MH is computationally very expensive. To speed up calculations on a uniprocessor, the function calls within the body of MH should be in-lined, but the entire PIRS software should preferably run on a parallel computer.

7.7. Training PIRS

Consider a space of all possible locations M. The locations can be grouped into sets $M_i \subset M$, $i = 1, 2, \ldots, k$, where M_i is the set of all locations (i.e. appearances) characteristic of a particular mineral.

Initially the knowledge base of PIRS is empty. Eventually, every mineral will be represented in PIRS knowledge bank by a series of balls of carefully chosen radii around characteristic locations. In this context a ball of a radius r around a location $L \in M_i$ is the set $B(L, r)$ of all locations of appearance sufficiently similar to L so that the human trainer of PIRS (a

Fig.4. A grain of plagioclase (in red), identified by one ball.

mineralogist) will classify them as belonging to M_i. In that sense, the set M_i will be considered a union of a number of balls, carefully chosen by the training expert. Fig.4. shows a grain of plagioclase, identified by one such ball. Only adjacent pixels showing plagioclase are highlighted.

Fig.5. shows all plagioclase identified in the entire sample, using only three balls. Note the sensitivity of our method: there is a blob of glue used to make a thin section, to the left of the lowest plagioclase grain. That grain has some cracks in it; the glue has flown into the cracks. Similarly, the blob of glue has some crumbs of plagioclase in it, which fell into glue during the lapping process. This is significant, as the glue is intended to be as inconspicuous as possible.

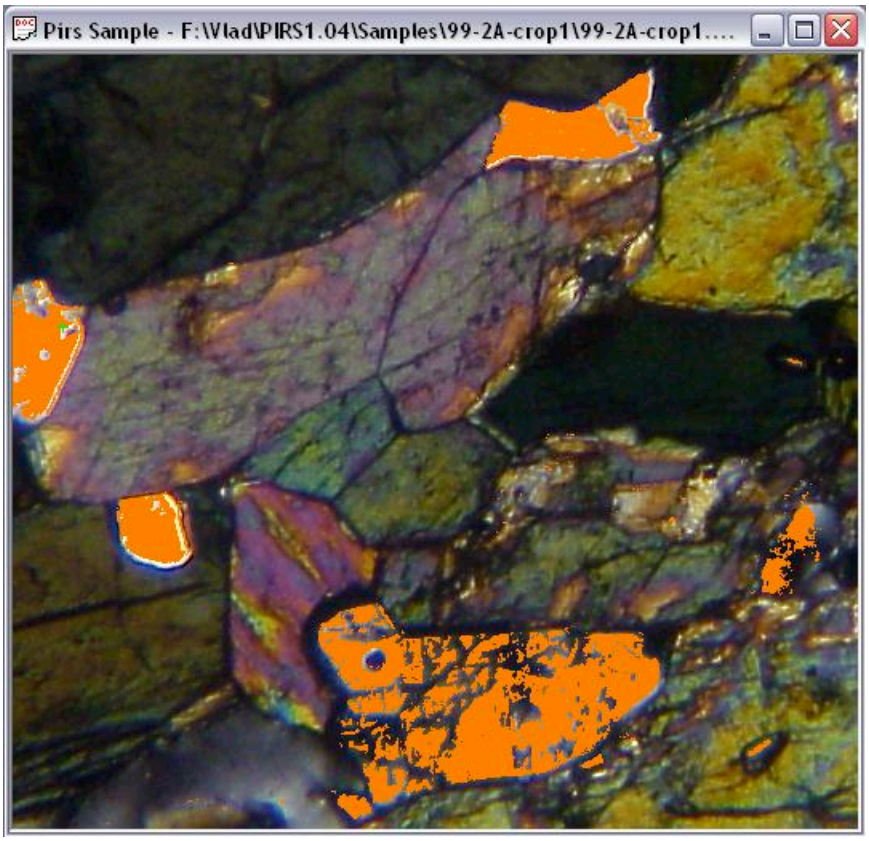

Fig.5. Plagioclase grains (orange), found using only three balls. All plagioclase in the sample has been identified.

The problem of detecting grain boundaries is worth mentioning here, given that many attempts are made to identify the boundaries first, and the grains within them later. We argue that a human expert is unable to see a "raw" sample image; the image perceived is unconsciously pre-processed by the brain. Do boundaries objectively exist? The mineral grains physically exist for sure, but the boundaries we see are the results of our unconscious image interpretation. If in doubt, ask yourself: What happens if we smash a grain? It disintegrates, of course. But: What happens to the grain boundary? It "grows". If it were real, this would be the only object we know that "grows" when we attempt to destroy it.

Consequently, we make PIRS identify grains. At the end of this process the unidentified locations represent grain boundaries or holes in the sample.

Let us illustrate the training process by an example. The new PIRS system has an empty knowledge base; it needs training. We capture a sample of, say, amphibolite, which contains grains of hornblende, plagioclase, garnet and quartz.

The invited expert is asked to identify an obvious grain of, say, plagioclase, and within it a location L_1 that is unmistakably characteristic of plagioclase. A mouse click will make PIRS note this location L_1. In order to identify the radius r_1 of the first ball $B_1(L_1, r_1)$ describing plagioclase in PIRS knowledge bank the expert operates a GUI slider. In response, PIRS highlights all locations on the screen that fall within the ball. The goal of the human trainer is to select the value r_1 as large as

possible so that as many plagioclase locations are included in the ball B_1, but only such locations. (Selection of too large value of r_1 would result in inclusion in B_1 some locations that are not plagioclase!). Selection of proper value r_1 is done by visual check of the expert trainer.

We could perform a safety check by analyzing other sample(s) of other rocks containing plagioclase grains and the value r_1 should be reduced accordingly to make sure that the ball B_1 is characteristic of plagioclase only. Then the ball B_1 should be filed in the PIRS knowledge bank under "Plagioclase". Fig.4. illustrates this situation.

Did we make PIRS capture all knowledge regarding the appearance of plagioclase? That is unlikely. Coming back to the original sample, the PIRS trainer may wish to select another location L_2 of plagioclase, which PIRS failed to recognize as such. By repeating the previously described procedure, the ball B_2 could be identified, then B_3, B_4, etc. Given that the search space is finite, a finite number of balls is being needed to capture all the appearances of plagioclase in to the knowledge base. Each time a new ball B_i is identified the expert selects a plagioclase location not recognized by previous balls $B_1, B_2, \ldots, B_{i-1}$. PIRS is smart enough to accelerate the identification process of sample locations: it keeps its mineral knowledge base sorted by ball radius (in descending order).

7.8. Fine-tuning the knowledge base of PIRS

What has been said about plagioclase could be repeated for all other minerals. It is obvious that the process of mineral identification is very compute-intensive but easily parallelizable. Fig.6. shows the resulting final interpretation of our thin section.

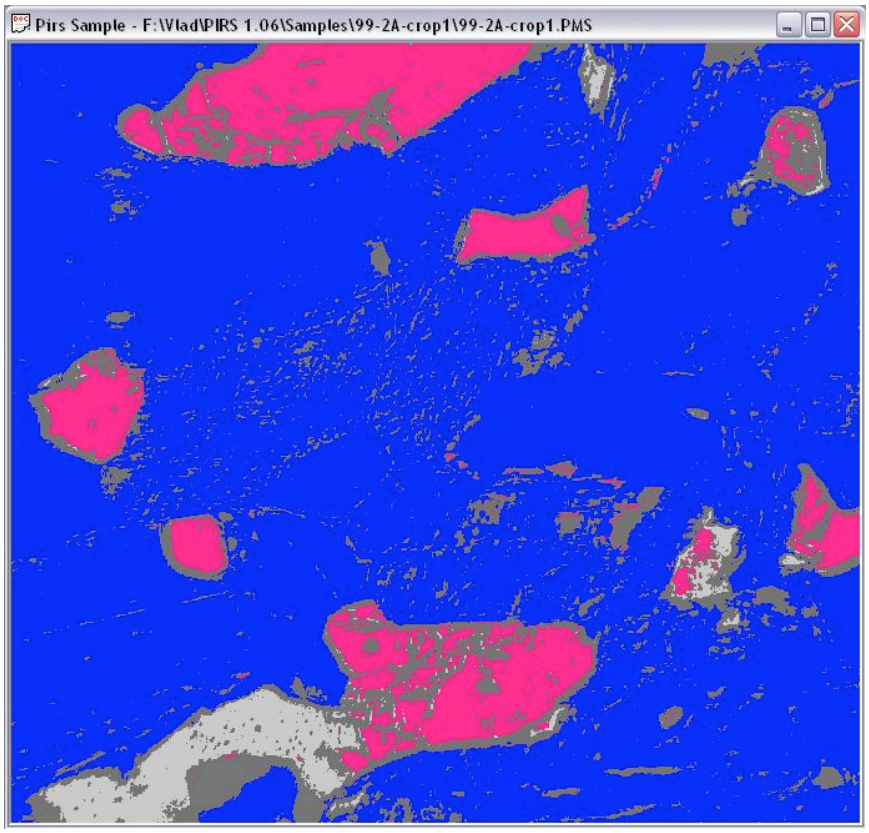

Fig.6. Thin section interpreted: plagioclase is shown in red, hornblende in blue, glue is light gray and dark gray represents the mineral boundaries and unknown impurities, etc.

We could accelerate this mineral recognition process even on a single-CPU machine by keeping the knowledge base about each mineral sorted in the descending order of the ball radii and by removing redundant balls. Also, for each ball a count of times it was found useful in mineral identification could be kept, and balls of low usefulness could be deleted from the knowledge bank, provided that they were given "a fair chance" to participate in the identification process.

The knowledge bank of PIRS is perhaps as valuable as PIRS itself. The same software could be used with specialized knowledge banks useful for drilling for oil, searching for various ores, or automated inspections for metallurgical defects, etc.

8. Case study #2: Deep sky search for unknown objects

Peering into deep space in many ways resembles petrographic mineral identification. Deep sky remains invariant: the diameter of the orbit of Earth around the Sun is negligibly small to account for any noticeable parallax. The same goes for the diameter of the orbit of the Sun around the center of our Galaxy. Telescopes peering into deep space have very small angle of view, and once oriented in a particular direction, the images they see are all the same all the time.

Because of their long focal length (i.e. small angle of view) such telescopes are natural image preprocessing devices: they present only the *WHAT*, while losing the *WHERE* information. In fact, aiming such telescopes visually presents a problem significant enough that larger telescopes are twinned with smaller scopes offering a wider angle of view. The smaller scopes are used to aim their bigger twins.

When performing observations, the astronomers can use various ranges of electromagnetic spectrum. They talk of observations in visible light, IR, UV, various radio frequencies, X-ray, gamma rays etc. Such set of observations of the same section of the sky is analogous to the series of images of a petrographic thin section. If we time such observations so that the intervening close objects (clouds, planets, etc.) are eliminated, the only remaining preprocessing needed is to perform the "colour correction" to account for the red shift.

We can train our system to recognize empty space (i.e. the background microwave radiation, the noise of the observational equipment, etc.) in much the same way we taught PIRS to recognize glue in petrographic samples. Everything not being an empty space is a celestial object of some kind. Some objects are so distant that they appear as points of electromagnetic radiation (i.e. they take up one pixel only in our images), while other objects may be large enough to cover some regions in the sky – they are akin to mineral grains. In fact, some larger objects may partially overlap, as can mineral grains: the grain boundaries in a thin section are not always perpendicular to the surface of a section.

With all the preprocessing accomplished, our PIRS software can be used to classify celestial objects. After training, the system can be used to call our attention to the "unknown" objects – i.e. objects it could not classify.

9. Concluding remarks and emergent questions

We have presented here the core of a bio-inspired methodology for pattern perception and machine learning which seems to us to be universal, because:

- It relies only on the foundation of mathematics (using only set theory and the basic concepts of metric spaces) and is therefore widely applicable;

- The mathematical toolset used makes it useful to any of the senses – the vision examples were merely conceptual illustrations;
- It is massively parallel, thus ensuring evolutionarily competitive execution times at any stage of development of computing technology;
- It is scaleable up and down, to match the parallel computing hardware available;
- It is fault tolerant, because all computations that can be executed in parallel can be executed in any partially sequential order, should the available processing elements suddenly become scarce;
- It offers suggestions of future directions of advancement of parallel computing, thus allowing us to leave the current *impasse* [17].

The systems emerging from the application of this methodology will:

- Be capable of using multiple senses, as long as their sensory subsystems consist of a finite number of distinct sensors; the sensors yielding continuous or discrete signals;
- Exhibit human-like need of extensive training before being put into reliable use;
- Be capable of continuous (on-the-job) learning, therefore ever improving their adaptation to their work environment;
- Be capable of keeping their adaptation to an environment that changes sufficiently slowly, by selective forgetting of less useful facts (i.e. balls) from their knowledge banks.

Their continuous adaptation through learning and selective forgetting can be achieved by keeping the working set of balls in their associative memories. The theory of working sets is already well known. Details at [8], [9].

* * *

Colleagues: let us take a final look at the expression (3.4) for computing distances between sets. A computer performing these calculations could as well consist of two structurally identical parts, executing identical algorithms but on slightly different data: one calculating $\Delta(A, B)$, while the other working on $\Delta(B, A)$. The final result could be obtained by comparing their results via the communication through a "corpus callosum". Could that be the model of human visual cortex spanning both brain hemispheres?

Given that our system is fault tolerant, a damage or total destruction of a half of that computer could merely lead to its performance degradation. Interestingly enough, the surgical procedure of hemispherectomy (removal or disabling of one of the hemispheres of the brain) in severely epileptic children does not lead to their total loss of vision, although it has other severe consequences: paralysis of one half of the body, among others. Is this just a coincidence? We, the authors, do not think so.

ACKNOWLEDGMENT

The authors would like to express thanks to Dr. F. Fueten of the Department of Earth Sciences, Brock University, for granting access to his equipment and petrographic samples that were used to demonstrate the applicability of our ideas to geology.

REFERENCES

[1] R.O. Duda, P.E. Hart, D.G. Stork: *Pattern Classification*, 2nd ed., John Wiley & Sons 2001.

[2] K. Fukunaga: *Statistical pattern recognition,* 2nd ed., Academic Press, 1990.

[3] J.C. Russ: *The image processing handbook,* CRC Press, 1992.

[4] B.V. Dasarathy: *Nearest neighbor (NN) norms: NN pattern classification techniques*, IEEE Computer Society Press, 1991.

[5] R. Klette, P. Zamperoni: *Handbook of image processing operators*, John Wiley & Sons, 1996.

[6] H.R. Schiffman: *Sensation and Perception - An Integrated Approach*, 5th ed., John Wiley & Sons, Inc. ISBN: 0-471-24930-0

[7] A.M. Gleason, *Elements of Abstract Analysis*, Jones and Bartlett Publishers, Boston 1991.

[8] A.M. Lister, R.D. Eager: *Fundamentals of operating systems*, 4th ed., Macmillan 1988.

[9] W. Stallings: *Operating systems: Internals and design principles*, 5th ed, Pearson / Prentice Hall, 2005.

[10] F. Fueten: *A computer controlled rotating polarizer stage for the petrographic microscope,* Computers and Geosciences (1997) 23, 203-208.

[11] F. Fueten: http://spartan.ac.brocku.ca/~ffueten/stage/WelcomeF.html, December 2004.

[12] I. Aleksander (ed.): *Neural computing architectures: The design of brain-like machines*, MIT Press, 1989

[13] E.R. Davies: *Machine vision: Theory, Algorithms, Practicalities*, Academic Press, 1990.

[14] H. Freeman (ed.): *Machine vision for three-dimensional scenes*, Academic Press, 1990.

[15] V.K.P. Kumar (ed.): *Parallel architectures and algorithms for image understanding,* Academic Press, 1991.

[16] V. Kumar, P.S. Gopalakrishnan, L.N. Kanal: *Parallel algorithms for machine intelligence and vision*, Springer Verlag, 1990.

[17] http://www.sysmannews.com/content/article.aspx?ArticleID=32043 , September 2008.