A Parallel Computation Algorithm for Image Feature Extraction

A. Belousov and J. Ratsaby*

Department of Electrical and Electronics Engineering, Ariel University, Israel

Abstract: We present a new method for image feature-extraction for learning image classification. An image is represented by a feature vector of distances that measure the dissimilarity between regions of the image and a set of fixed image prototypes. The method uses a text-based representation of images where the texture of an image corresponds to patterns of symbols in the text string. The distance between two images is based on the LZ-complexity of their corresponding strings. Given a set of input images, the algorithm produces cases that can be used by any supervised or unsupervised learning algorithm to learn image classification or clustering. A main advantage in this approach is the lack of need for any image processing or image analysis. A non-expert user can define the image-features by selecting a few small images that serve as prototypes for each class category. The algorithm is designed to run on a parallel processing platform. Results on the classification accuracy and processing speed are reported for several image classification problems including aerial imaging.

Keywords: Image classification, Parallel distributed processing, String distance.

1. INTRODUCTION

Image classification research is an area of research which is part of the field of computer vision and image processing. It aims at finding representations of images that can be automatically used to categorize images into a finite set of categories (classes). Applications of image classification include the major areas of remote sensing image classification [14], medical image analysis, retrieval and computer aided diagnosis [4], industrial inspection and anomalies detection [16], and pattern recognition, as in for instance, fingerprint classification (used for efficient fingerprint identification) [9]. Typically, algorithms that classify images require some form of pre-processing of an image prior to classifying it. This process usually includes an extraction of relevant features or descriptors that describe certain characteristics of the image. For instance, low level local descriptors give a description of the image in terms of its color, shape, regions, and texture. It is well known that texture is a very important feature for describing regions of an image and plays a significant role in image classification [12, 17]. Texture dictates how smooth or coarse a part of an image is. There are different ways in which texture can be represented. Among the more popular ones is the spectral approach where the texture of an image is extracted from properties of the Fourier transform of the image [10], and the local binary pattern (LBP) representation [15] which uses a binary code to describe local texture patterns.

Another approach, introduced in [5] is to convert an image into text and represent its texture by patterns of symbols. It is then possible to use string-distances, such as the one introduced in [19], to measure the dissimilarity between the texture of two images. A pair of images that possess similar regions of texture will have similar textual patterns in their corresponding strings. Provided that the distance ignores 'irrelevant patterns' that are introduced by noise or that are non-representative of the overall texture of the image, then it is possible to accurately measure the level of dissimilarity between the two images based on their different textures.

Using this idea, [5] introduced a new distance function, called Universal Image Distance (UID), for measuring the distance between two images. The UID first transforms each of the two images into a string of characters from a finite alphabet and lets the distance value between the images be the value given by a string distance (defined in [19]) between their corresponding strings. According to [19] the distance between two strings x and y is a normalized difference between the complexity of the concatenation xy of the strings and the minimal complexity of each of x and y. The complexity of a string x is the Lempel-Ziv complexity [20].

In [6], the UID was applied in a serial algorithm to convert images into feature vectors that can be used as training cases for learning image classification. The ith dimension is a feature that measures the UID between the image and prototype images of the ith feature category. For instance, if the problem is to learn to classify images into the transportation categories of *cars*, *trucks*, *boats* and *airplanes*, then feature categories of *cars* may be for instance *wheel*, *door*, *headlights* and feature categories of airplanes may be *wing* and *tail*.

*Address correspondence to this author at the Department of Electrical and Electronics Engineering, Ariel University, Israel; Tel:+972-3-907-6587;
E-mail: ratsaby@ariel.ac.il

E-ISSN: 2409-5761/19 © 2019 Avanti Publishers

An advantage of the UID is that it can compare the distance between two images of different sizes and thus the prototypes which are representative of the different feature categories may be relatively small and hence easy to pick by the user that runs the algorithm. For instance, an aerial image of the category industry of size 128×128 can be compared against a prototype aerial image of a roof of a warehouse of size 40×30. This facilitates the process of selecting prototypes since it is easy to choose small images as prototypes that have a homogenous texture that is representative of a feature of the category.

Several basic problems of learning image classification and image clustering based on this algorithm were investigated in [5, 6] and resulted in high accuracy. The disadvantage with the serial algorithm is the amount of time it takes to produce the training cases. For this reason, [3], initiated experiments with a parallel distributed algorithm that achieves what the serial algorithm does but much faster. On a standard graphics processing unit (GPU) it improves the execution speeds relative to [6] by more than three orders of magnitude. The algorithm converts an input image into a labeled case and by doing this for the set of images, each labeled by its class, it yields a data set that can be used to train any 'off-the-shelf' supervised or unsupervised learning algorithm.

This process of converting an image into a finite dimensional feature vector does not involve any prior domain knowledge or image analysis expertise. Compared to other image classification approaches that extract features based on sophisticated mathematical analysis [10, 17, 18], for instance, analyzing the texture by various filters, or checking for special spectral properties of an image, our approach represents two-dimensional images by a naive one-dimensional horizontal trace of the pixel values that yields a string. As we present in section 5, surprisingly, this naive representation of an image apparently captures enough texture information about the image that leads to accurate image-classification.

The current paper presents an improved version of the parallel algorithm of [3] and reports on new experimental results of several image classification learning problems, including texture images and aerial images classification which serve as test-bench problems in the field of research. The paper is organized as follows: in section 2 the concept of string complexity and string distance are defined, in section 3 we define the image distance, in section 4 the algorithm is described and in section 5 we present the result of several problems of learning image classification.

2. LZ-COMPLEXITY AND STRING DISTANCES

The UID function [5] is based on the LZ- complexity of a string. The definition of this complexity follows [19, 20]: let S, Q and R be strings of characters that are defined over the alphabet A. Denote by $l(S)$ the length of S, and $S(i)$ denotes the ith element of S. We denote by $S(i,j)$ the substring of S which consists of characters of S between position i and j (inclusive). An extension $R = SQ$ of S is reproducible from S (denoted as $S \to R$) if there exists an integer $p \le l(S)$ such that $Q(k) = R(p+k-1)$ for $k = 1,\ldots,l(Q)$. For example, $aacgt \to aacgtcgtcg$ with $p = 3$ and $aacgt \to aacgtac$ with $p = 2$. R is obtained from S (the seed) by first copying all of S and then copying in a sequential manner $l(Q)$ elements starting at the pth location of S in order to obtain the Q part of R.

A string S is *producible* from its prefix $S(1,j)$ (denoted $S(1,j) \Rightarrow S$), if $S(1,j) \to S(1,l(S)-1)$. For example, $aacgt \Rightarrow aacgtac$ and $aacgt \Rightarrow aacgtacc$ both with pointers $p = 2$. The production adds an extra 'different' character at the end of the copying process which is not permitted in a reproduction.

Any string S can be built using this *production process* where at its ith step we have the production $S(1,h_{i-1}) \Rightarrow S(1,h_i)$ where h_i is the location of a character at the ith step. (Note that $S(1,0) \Rightarrow S(1,1)$).

An m-step production process of S results in parsing of S in which $H(S) = S(1,h_1) \cdot S(h_1+1,h_2) \cdots S(h_{m-1}+1,h_m)$ is called the *history* of S and $H_i(S) = S(h_{i-1}+1,h_i)$ is called the ith component of $H(S)$. For example, for $S = aacgtacc$, we have $H(S) = a \cdot ac \cdot g \cdot t \cdot acc$ as the history of S.

If $S(1,h_i)$ is not reproducible from $S(1,h_{i-1})$ then the component $H_i(S)$ is called *exhaustive*, meaning that the copying process cannot be continued and the component should be halted with a single character *innovation*. Every string S has a unique exhaustive history [20].

Let us denote by $c_H(S)$ the number of components in a history of S. The LZ complexity of S is defined as $c(S) = \min\{c_H(S)\}$, where the minimum is over all histories of S. It can be shown that $c(S) = c_E(S)$ where

$c_E(S)$ is the number of components in the exhaustive history of S.

A distance for strings based on the LZ-complexity was introduced in [19] and is defined as follows: given two strings, X and Y, denote by XY their concatenation then define

$$d(X,Y) := \max\left\{c(XY) - c(X), c(YX) - c(Y)\right\}.$$

In [5], it was found that the following normalized distance

$$d(X,Y) := \frac{c(XY) - \min\left\{c(X), c(Y)\right\}}{\max\left\{c(X), c(Y)\right\}}. \quad (2.1)$$

performs well in classification and clustering of images. d is normalized because $c(XY) - \min\left\{c(X), c(Y)\right\} \leq c(X) + c(Y) - \min\left\{c(X), c(Y)\right\} = \max\left\{c(X), c(Y)\right\}$.

We note in passing that (2.1) resembles the normalized compression distance of [7], which rely on data compression for approximating a string's complexity. In contrast, the LZ-complexity is computed exactly with no need for a compressor. Note that d is not a metric since a distance of 0 implies that the two strings are close but not necessarily identical. We refer to d as a universal distance because it applies to any two individual strings with no assumption or prior knowledge about the problem or source that generated them. d only depends on the LZ-complexity of each of the two strings (and their concatenation), which does not assume anything about the origin or the source of the strings.

3. IMAGE DISTANCE

Based on (2.1), we now define a distance between images. The idea is to convert each of two images I and J into strings $X^{(I)}$ and $X^{(J)}$ of characters from a finite alphabet of symbols. Once in string format, we use $d(X^{(I)}, X^{(J)})$ as the distance between I and J. The details of this process are described in Procedure UID.

Procedure UID (Universal Image Distance)

1. **Input**: two color images I, J in jpeg format (RGB representation)
2. Transform the RGB matrices into gray-scale. Each pixel is now a single numeric value in the range of 0 to 255. We refer to this set of values as the alphabet and denote it by A.
3. Scan each of the grayscale images from top left to bottom right and form a string of symbols from A. Denote the two strings by $X^{(I)}$ and $X^{(J)}$.
4. Compute the LZ-complexities: $c\left(X^{(I)}\right)$, $c\left(X^{(J)}\right)$ and the complexity of their concatenation $c\left(X^{(I)}X^{(J)}\right)$
5. **Output**: $UID(I,J) := d\left(X^{(I)}, X^{(J)}\right)$.

The transformation into gray-scale is a matter of representational convenience and in this paper, we only deal with classifying images based on their gray-scale versions. Clearly, classifying color images is an easier problem since color information adds significant information, for instance, detecting fire in an image of a forest is much easier with a color image than with grayscale. As a note in passing, in order to deal with color images without this transformation, one can create a three-dimensional alphabet' whereby each 'letter' in this alphabet corresponds to an RGB triple with each component in the range of 0 to 255. Strings would comprise of sequences of such three-dimensional symbols whose LZ-complexity can be computed, albeit taking considerably more time. This way color information about the image can be included in the string representation.

Figure **1** displays hierarchical clustering of images of objects from categories $cars$, $trucks$ using the UID; it is based on a matrix of distances between all pairs of objects (Table **1**). The labels b through g in the dendrogram correspond to the labels of the objects (b) to (g). The vertical axis of the dendrogram indicates the distance value (which by definition is between 0 and 1). There are two clearly distinguishable clusters and they correspond to the two category groups of objects. The categories of the images are not given to the clustering algorithm but only the distance values hence the UID successfully measures objects of the same category as closer than objects from different categories.

Another example with three categories is displayed in Figure **2**, which is based on a distance matrix of Table **2**. Here too the UID succeeds and measures objects of different categories as more distant than objects from the same categories.

4. ALGORITHM

In this section we describe the parallel distributed processing algorithm for extracting features and learning image classification. Given a set of images, the algorithm produces the corresponding cases (feature vectors) that can be used by any supervised learning algorithm to learn the image classification task.

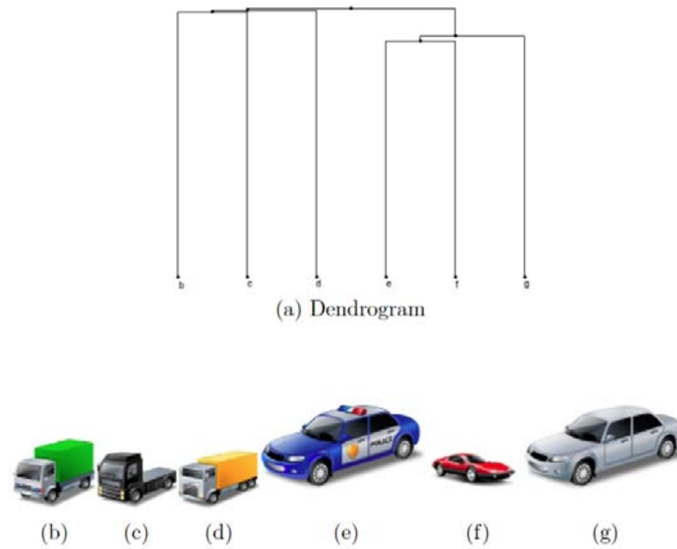
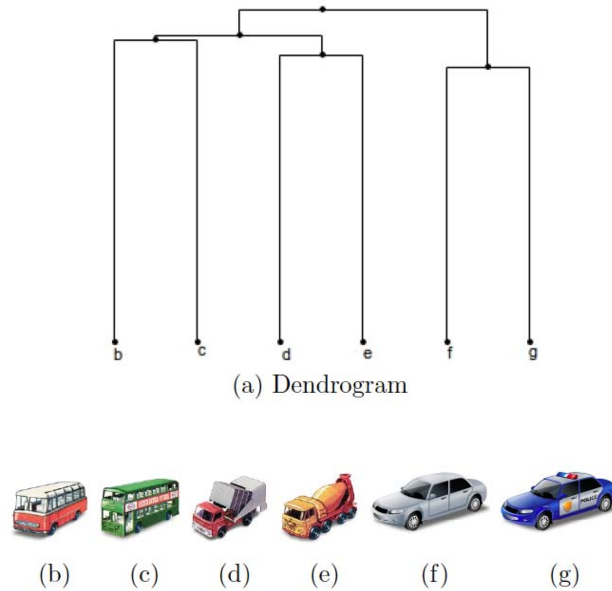


Figure 1: Hierarchical clusters of images of objects from two categories *trucks* and *cars*; distances based on Procedure UID. As can be seen, the UID is able to distinguish between trucks and cars and place them in distinct clusters.

Table 1: Distance Matrix for Objects in Figure 1

Image	f	e	g	c	d	b
f	0.001	0.842	0.85	0.901	0.89	0.911
e	0.85	0.001	0.862	0.891	0.9	0.904
g	0.845	0.846	0.001	0.886	0.898	0.915
c	0.889	0.875	0.883	0.001	0.882	0.881
d	0.893	0.892	0.902	0.893	0.001	0.902
b	0.897	0.888	0.906	0.877	0.883	0.001



(b) (c) (d) (e) (f) (g)

Figure 2: Hierarchical clusters of images of objects from three categories *buses*, *trucks* and *cars*; distances based on Procedure UID. The UID is able to distinguish between buses, trucks and cars and place them in distinct clusters. This is based on the patterns that it picks out in the string-representation of these images.

Table 2: Distance Matrix for Objects in Figure 2

Image	c	b	e	d	g	f
c	0	0.888	0.893	0.902	0.921	0.926
b	0.892	0	0.892	0.9	0.92	0.928
e	0.889	0.89	0	0.879	0.916	0.917
d	0.896	0.896	0.882	0	0.91	0.915
g	0.915	0.912	0.909	0.902	0	0.87
f	0.921	0.916	0.92	0.91	0.869	0

The algorithm is designed for scalable distributed computations and takes advantage of relatively inexpensive yet massively-parallel GPU processors that are ubiquitous in today's technology.

The algorithm is divided into several stages. The first, which we denote as Algorithm 2P, is presented in section 4.2. It selects prototype images for each of the feature categories. A prototype image of a feature category is defined as a small image that has some repetitive pattern of texture which may appear, perhaps at different scales, in local regions of typical images that have this feature. The second stage is Algorithm 3P which is described in section 4.3. It computes the cases (feature vectors) for images in the input set. The final stage is Algorithm 4 (section 4.4) which uses the cases for training any supervised learning algorithm.

At the very basis of the algorithm, there are three procedures, each designed to work in a parallel processing manner. We describe them in the next section.

4.1. Procedures

Procedure LZMP (Lempel-Ziv Massively Parallel) is described on page 10. It takes a string and computes its LZ-complexity, building the exhaustive history (as described in section 2) by searching for the components in parallel. Compared to the standard serial procedure of computing the LZ-complexity of a string, LZMP has a speedup factor which is linear in the number of computing cores [2]. The implementation of Procedure LZMP in CUDA is described in [1] section 3.3.

Let us define the function,

$$dp(X,Y,a,b) := \frac{LZMP(XY) - \min\{a,b\}}{\max\{a,b\}} \tag{4.1}$$

where a, b are the LZ-complexity values of the string X, Y, respectively. This function computes the distance (2.1) where the complexity $c(XY)$ of their concatenation is computed by LZMP while the complexity values, a, b, of the individual strings, X, Y is given (precomputed at an earlier stage). dp is used in procedure DMat, step 3, when computing the distance between all pairs of strings that are elements of two vectors.

A key to our parallel approach is to compute distances between multiple subregions of an image and prototypes from all categories, in parallel. This is achieved by Procedure VLZMP page 11, which computes the LZ-complexity of all elements of a vector of strings and executes the LZMP procedure multiple times in parallel. The implementation of Procedure VLZMP in CUDA is described in [1] section 4.2.

Procedure DMat page 11 computes in parallel the distance dp between every pair of elements of two vectors of strings. It uses VLZMP to precompute the LZ-complexity of each individual string of the two vectors and then computes dp in parallel for all pairs. The variable $i_{p,q}$ denotes an index variable of the computing block $B_{p,q}$ (each block has its own local memory and set of variables). The implementation of Procedure DMat in CUDA is described in [1] section 5.3.

4.2. Algorithm 2P

Our approach to image classification is to extract features automatically by computing distances from a set of prototypes images that are selected manually, before learning. Algorithm 2P, displayed on page 11, queries the user iteratively for these prototypes, for each feature category. It displays a dendrogram of the hierarchical clusters of these prototypes. If clusters are formed such that they correspond to the feature categories then Algorithm 2P halts which indicates that

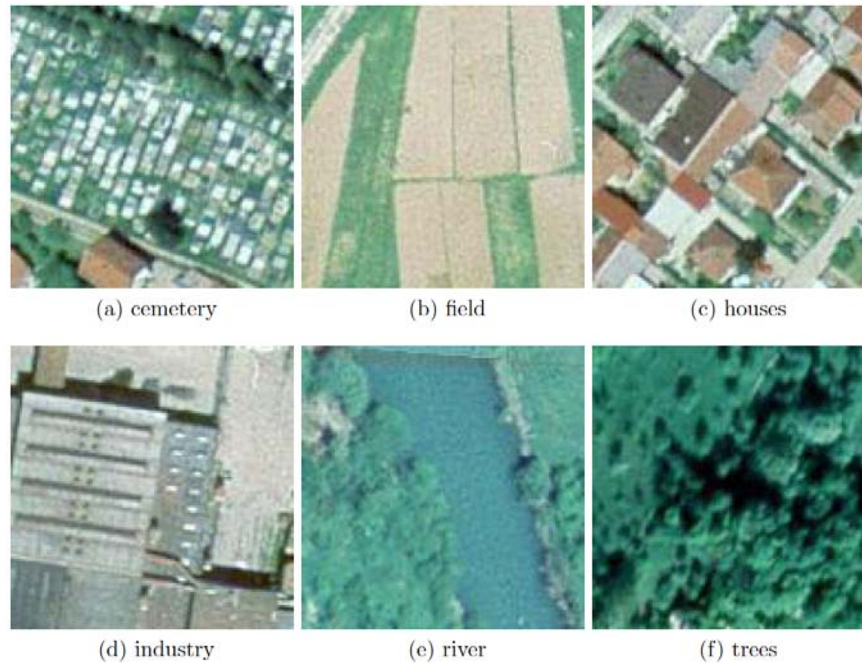


Figure 3: Examples of images from which prototypes are selected. Prototypes are depicted by the small rectangular parts. They are easily chosen by a non-expert.

the prototypes are good. Subsequently, they are used by Algorithm 3P.

Since Algorithm 2P involves the user in providing candidates as image prototypes and interpreting the dendrogram, there are some basic heuristic guidelines for the user to follow in order to select good prototypes. As an example, consider the Banja Luka dataset [18] (discussed in more details in section 5) which consists of six categories of aerial images: *houses*, *cemetery*, *industry*, *field*, *river* and *trees*.

Figure **3** shows typical images from this dataset. In this particular problem, we let the feature categories be the classification categories themselves. In general, the feature categories may be different and more specific than the classification categories and there can be more than one feature category for every classification category.

Figure **4** shows an image of category *houses* and the selected prototypes of smaller size taken from this image. Each prototype is predominantly made of a single texture. The left and bottom prototypes capture the texture of roofs while the right prototype captures the texture of a street. Both, roofs and streets make up features of the category houses.

To explain the heuristic of prototype selection, we start by showing what happens when one chooses bad prototypes. Figure **5** displays these prototypes which are numbered as follows: 1 - 4 (cemetery), 5 - 8 (field), 9 - 12 (houses), 13 - 16 (industry), 17 - 20 (river), 21 - 24 (trees). They are considered bad because each contains more than a single feature of its category; for instance, prototypes of category *houses* contain not only images of houses but also of streets and vegetation, which appear in other categories. This can lead to a similarity between prototypes of different categories and result in a higher misclassification error. One can see in these prototypes an object or a region with texture that is not characteristic of the corresponding category. Algorithm 2P detects this fact and displays a dendrogram with bad clusters, for instance, prototypes 5 to 8 do not belong to the same cluster, as seen in Figure **6**. This leads to poor accuracy results, as reported in section 5.

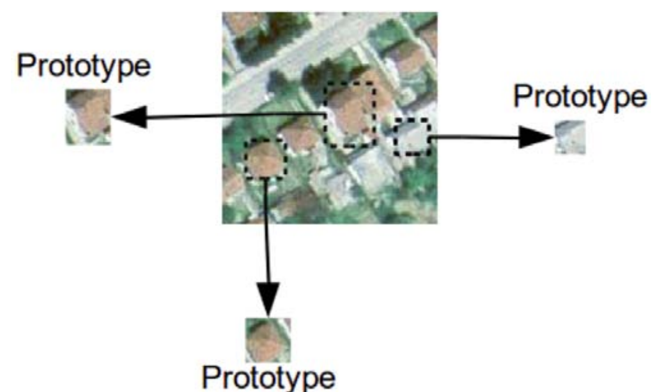


Figure 4: Selecting prototypes.

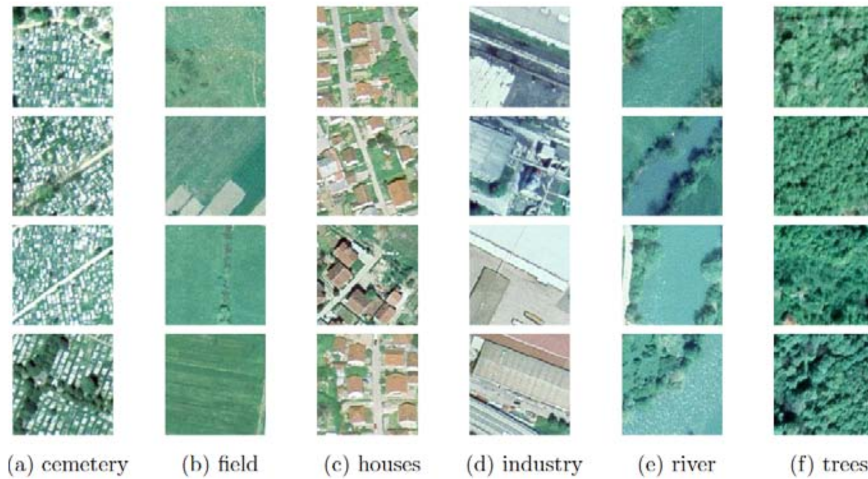


Figure 5: Examples of bad prototypes. They contain features that appear in more than one category, for instance, the prototypes for houses contain not only houses but also streets and vegetation which appear in other categories.

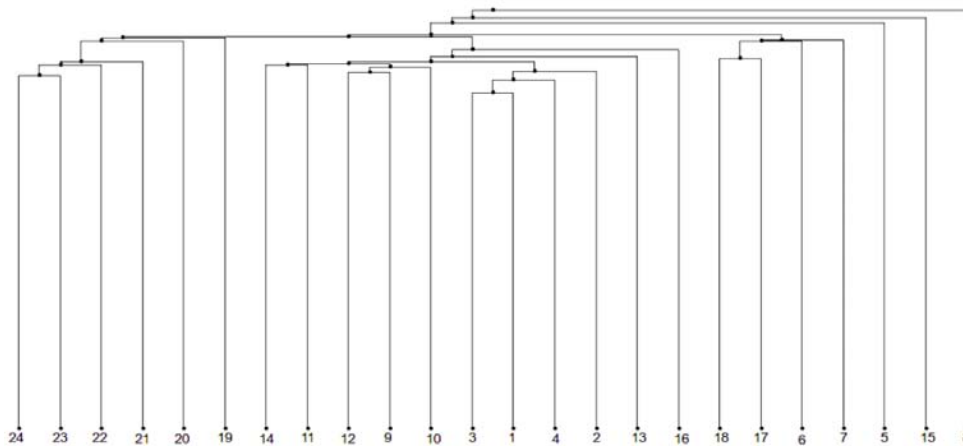


Figure 6: Dendrogram corresponding to bad prototypes. For instance, prototypes 5 and 8 which are from the *field* category are not in the same local cluster.

For this same problem, Figure **7** displays a set of good prototypes numbered as follows: 1 - 4 (cemetery), 5 - 8 (field), 9 - 12 (houses), 13 - 16 (industry), 17 - 20 (river), 21 - 24 (trees). In each category, we selected prototypes that each possesses predominantly a single texture that is characteristic of the category. The aim is not to include more than a single feature in a single prototype. In this particular problem, since we chose feature categories as the classification categories, then prototypes in the same category had a similar texture while those in different categories had different texture. Clear hierarchical clusters are formed that correspond to the classification categories, as displayed in the dendrogram of Figure **8**. One advantage of the UID is the fact that we can measure the distance between a pair of images of different sizes. This enables one to choose prototypes that are small while leaving the size of the input image different. Being small means that a prototype can be made to have a single, or few, features of its category and this is desirable as mentioned above.

4.3. Algorithm 3P

Algorithm 3P, displayed page 12 and page 13, uses the prototypes selected by Algorithm 2P and computes for every image in the input set a feature vector (training case). Algorithm 3P reads every input image and divides it into a number of non-overlapping square (window) regions, denoted as sub-images. It then outputs a case which consists of the proportion number of times that a prototype was closest to any of these sub-images. Each component of the case represents this proportion for a different prototype. The algorithm utilizes a number of computing blocks which begin to run in parallel in step 12. Steps 6 to 11 which run in serial are responsible for converting each of the N input images I_i into a vector v_i of strings $X_{i,j}$ that correspond to sub-images $I_j^{(i)}$ of $I_i, 1 \le i \le N$, and

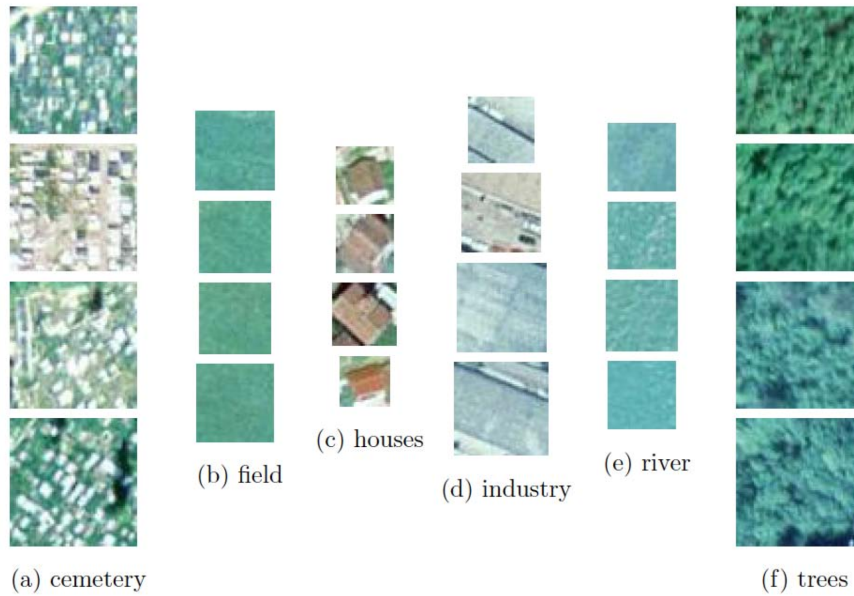


Figure 7: Good prototypes for the Banja Luka dataset.

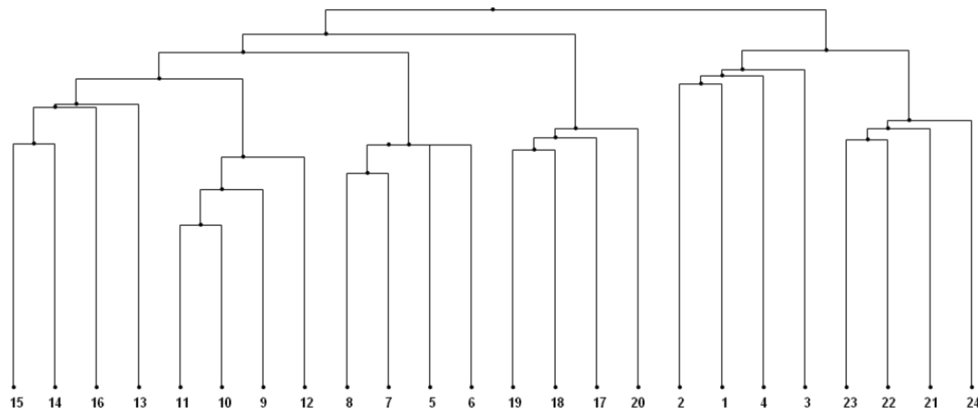


Figure 8: Dendrogram of good prototypes in Figure **7**. Based on the UID, hierarchical clustering discovers the six clusters that correspond to the six categories of prototypes in Figure **7**.

converting each of the L prototype images into strings which together form the vector u. Figure **9** depicts an example of an input image I_i which is split into sixteen sub-images $I_j^{(i)}$, $1 \leq j \leq 16$, shown in Figure **10**.

Figure 9: Example of an input image.

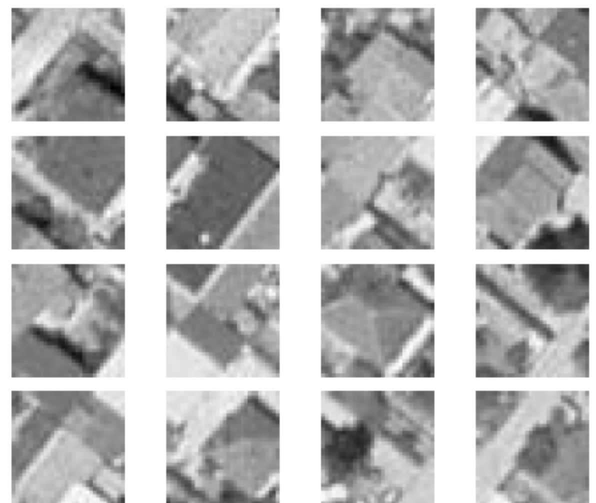


Figure 10: Sub-images of the image in Figure **9** obtained by Algorithm 3P.

Next, we highlight some of the main steps of Algorithm 3P. Each computing block B_q (there are a total of Q blocks) processes one image at a time thus a total of Q images are processed in parallel at any given time. A block B_q, which on a typical GPU may use thousands of threads in parallel, executes Procedure DMat on a pair of vectors v_{i_q} and u, where v_{i_q} consists of the strings that correspond to all sub-images of image I_{i_q}. On Block B_q Procedure DMat returns a matrix D_q whose component $D_q[j,k]$ is the UID value between the kth prototype and j^{th} sub-image of image I_{i_q}. Block B_q then determines for each subimage of image I_{i_q} the closest prototype to it (the winner), and keeps the count of the number of times that a prototype won and eventually produces the case for that image. The above is repeated until all images I_i are processed, $1 \le i \le N$, where, except possibly for the last iteration, there are always Q images being processed concurrently. Table **3** displays cases produced by Algorithm 3P for four input images shown in Figure **11** with one prototype per category hence the dimension of a case equals the number of categories.

4.4. Algorithm 4

Given the input set I of N images, as described in the previous section, Algorithm 3P produces a set R of L-dimensional normalized feature vectors, where $L = \sum_{l=1}^{M} L_l$ is the total number of prototypes. Denote by T the category target variable according to which the images are to be classified and denote by τ the set of values that T takes. In the transportation problem (section 1), the set τ consists of the six category values. Label each case with the category of the image

Figure 11: Input images.

Table 3: Cases Produced by Algorithm 3P for Images in Figure 11

	trees	cemetery	industry	river	houses	field
Figure **11a**	0	0	0	0	0.125	0.875
Figure **11b**	0	0	0	0	0.0625	0.9375
Figure **11c**	0	0	0.125	0.0625	0.8125	0
Figure **11d**	0	0	0.125	0	0.875	0

that it represents and let the resulting set of labeled cases be denoted by D_T. The process of learning classification from D_T is described in Algorithm 4 on page 14.

Procedure LZMP: computes LZ complexity of a string (parallel processing over all symbols of string)

1. **Input**: string $S = \{S[i]\}_{i=1}^{n}$
2. **Initialize:** H history buffer, $m := 0$ is length of history buffer, $d := 0$ is number of components in exhaustive history, SM is shared memory variable common to all threads, Q is number of computing threads. $\{T_q\}_{q=1}^{Q}$, T_q is a single computing thread
3. **Launch threads** T_q , $1 \le q \le Q$, in parallel, each executes the code below
   - I. **while**$(m < n)$
     - A. $SM := 0$
     - B. **for**$(l = 0$ **to** $\lfloor m/Q \rfloor)$
       - i. initialize variable $j^{(q)} = q + l \cdot Q$
       - ii. **if**$(j^{(q)} < m)$
         - a. initialize variable $i_{j^{(q)}} := 0$
         - b. initialize variable $k_{j^{(q)}} := j^{(q)}$
         - c. initialize variable $h_{j^{(q)}} := m - j^{(q)}$
         - d. **while**$(H[k_{j^{(q)}}] = S[m + i_{j^{(q)}}])$
           - 1. $i_{j^{(q)}} := i_{j^{(q)}} + 1$
           - 2. $k_{j^{(q)}} := k_{j^{(q)}} + 1$
           - 3. $h_{j^{(q)}} := h_{j^{(q)}} - 1$
           - 4. **if**$(h_{j^{(q)}} = 0$ **or** $m + i_{j^{(q)}} = n)$
             - I. **break;**
           - 5. **end if;**
         - e. **end while;**
         - f. **if**$(h_{j^{(q)}} = 0$ **and** $m + i_{j^{(q)}} < n)$
           - 1. initialize $z_{j^{(q)}} := m$
           - 2. **while**$(S[z_{j^{(q)}}] = S[m + i_{j^{(q)}}])$
             - I. $z_{j^{(q)}} := z_{j^{(q)}} + 1$
             - II. $i_{j^{(q)}} := i_{j^{(q)}} + 1$
             - III. **if**$(m + i_{j^{(q)}} = n)$
               - A. **break;**
             - IV. **end if;**
           - 3. **end while;**
         - g. **end if;**
         - h. **if** $(i_{j^{(q)}} > SM)$
           - 1. $SM := i_{j^{(q)}}$, // winner thread overrides
         - i. **end if;**
       - iii. **end if;**
     - C. **end for;**
     - D. synchronize all threads T_q, $1 \le q \le Q$
     - E. **if**$(q = 1)$
       - i. $H := H + \text{substring}(S[m], S[m + SM + 1])$
       - ii. $d := d + 1$
       - iii. $m := m + SM + 1$
     - F. **end if;**
     - G. synchronize all threads T_q, $1 \le q \le Q$
   - II. **end while;**
4. **Output**: $LZMP(S) = d$, the LZ-complexity of string S

Procedure VLZMP: computes a vector of LZ complexities for multiple input strings in parallel

1. **Input**: vector $v := \{v[i]\}_{i=1}^{k} = [S_1, S_2, S_3, ..., S_k]$ where S_i is a string
2. **Initialize:**
   I. $u = \{u[i]\}_{i=1}^{k}$
   II. n number of parallel computing blocks
   III. $\{B_q\}_{q=1}^{n}$, B_q is block of multiple computing cores (threads)
3. **Launch blocks** B_q , $1 \leq q \leq n$, in parallel, each executes the code below
   I. **for**($l = 0$ **to** $\lfloor k/n \rfloor$)
      A. initialize index vector $i = [i_1, \ldots, i_n]$ where $i_q = q + l \cdot n$
      B. **if** ($i_q \leq k$)
         i. $u[i_q] = LZMP(v[i_q])$
      C. **end if;**
   II. **end for;**
4. **Output:** $VLZMP(v) = u$

Procedure DMat: computes dp distance for all pairs of input strings in parallel

1. **Input**:
   I. $v := \{v[i]\}_{i=1}^{m} = [S_1, S_2, ..., S_m]$, where S_i is a string
   II. $u := \{u[j]\}_{i=1}^{n} = [S'_1, S'_2, ..., S'_n]$, where S'_j is a string
2. **Initialize:**
   I. D matrix of $m \times n$ elements, $D := \{D[i,j]\}_{i=1,j=1}^{m,n} =$
   $$\begin{pmatrix} d_{11} & d_{12} & d_{13} & ,.., & d_{1n} \\ d_{21} & d_{22} & d_{23} & ,.., & d_{2n} \\ d_{31} & d_{32} & d_{33} & ,.., & d_{3n} \\ ,..., & ,..., & ,..., & ,..., & ,..., \\ d_{m1} & d_{m2} & d_{m3} & ,.., & d_{mn} \end{pmatrix}$$
   II. $M \cdot N$ number of parallel computing blocks
   III. $\{B_{p,q}\}_{p=1,q=1}^{M,N}$, $B_{p,q}$ is a block of multiple computing cores (threads)
   IV. $a := \{a[i]\}_{i=1}^{m} = VLZMP(v)$, $b := \{b[i]\}_{i=1}^{n} = VLZMP(u)$, LZ-complexity vectors
3. **Launch** blocks $B_{p,q}$, $1 \leq p < M$, $1 \leq q < N$, in parallel, each executes the code below
   I. **for** ($x = 0$ **to** $\lfloor n/N \rfloor$)
      A. initialize index $i_{p,q} = q + x \cdot N$
      B. **for** ($y = 0$ **to** $\lfloor m/M \rfloor$)
         i. initialize index $j_{p,q} = p + y \cdot M$
            a. **if** ($i_{p,q} \leq m$ **and** $j_{p,q} \leq n$)
               1. $D[i_{p,q}, j_{p,q}] = \mathtt{dp}(v[i_{p,q}], u[j_{p,q}], a[i_{p,q}], b[j_{p,q}])$
            b. **end if;**
      C. **end for;**
   II. **end for;**
4. **Output:** $DMat(v, u) = D$

Algorithm 2P: Prototypes selection

1. **Input**: M image feature categories, and a set $\mathcal{C}_N$ of N unlabeled colored (RGB) images $\{I_j\}_{j=1}^N$.
2. **for** $(i := 1$ to $M)$ **do**
   I. Based on *any* of the images I_j in $\mathcal{C}_N$, let the user **select** L_i prototype images $\left\{P_k^{(i)}\right\}_{k=1}^{L_i}$ and set them as feature category i. Each prototype is contained by some image, $P_k^{(i)} \subset I_j$, and the size of $P_k^{(i)}$ can vary, in particular it can be much smaller than the size of the images I_j, $1 \leq j \leq N$.
   II. Transform each of the images of feature category i into grayscale. Each pixel is now a single numeric value in the range of 0 to 255 . We refer to this set of values as the alphabet and denote it by $\mathcal{A}$.
   III. Scan each of the grayscale images from top left to bottom right and form a string of symbols from $\mathcal{A}$. Denote the the string of grayscale image I as $X^{(I)}$.
3. end for**;**
4. **Enumerate** all the prototypes into a single *unlabeled* set $\{P_k\}_{k=1}^L$, where $L = \sum_{i=1}^M L_i$.
5. Vector of strings that corresponds to the set of all prototypes be, $v = \left[X^{(P_k)}\right]_{k=1}^L$.
6. Calculate the distance matrix $H = DMat(v, v)$
7. **Run** hierarchical clustering on H and obtain the associated dendrogram (note: H does not contain any 'labeled' information about feature-categories, as it is based on the unlabeled set).
8. **If** there are M clusters with the ith cluster consisting of the prototypes $\left\{P_k^{(i)}\right\}_{k=1}^{L_i}$ **then** terminate and **go to** step 10.
9. **Else go to** step 2**.**
10. **Output**: the set of labeled prototypes $\mathcal{P}_L := \left\{\left\{P_k^{(i)}\right\}_{k=1}^{L_i}\right\}_{i=1}^M$ where L is the number of prototypes.

Algorithm 3P: produces a set of cases from input images (in parallel)

1. Input set $\mathcal{P} := \left\{\left\{P_k^{(i)}\right\}_{k=1}^{L_i}\right\}_{i=1}^M$ of labeled prototype images, where $P_k^{(i)}$ is k^{th} prototype of feature category i (obtained from Algorithm 2P, on the preceding page).
2. Let $L := |\mathcal{P}|$ be the total number of prototypes
3. Input the set of all images $\mathcal{I} := \{I_l\}_{l=1}^N$ to be represented as cases of feature vectors
4. Q is number of parallel computing blocks
5. $\{B_q\}_{q=1}^Q$, B_q is a block of multiple computing cores (threads)
6. Let W be a rectangle of size equal to the maximum prototype size
7. **for** $(i := 1$ to $N)$
   I. Scan a window W across I_i from top-left to bottom-right in a non-overlapping way, and let the sequence of obtained subimages of I_i be denoted by $\left\{I_j^{(i)}\right\}_{j=1}^{m_i}$ (m_i is the number of windows W inside I_i).
   II. **for** $(j := 1$ to $m_i)$
      A. Transform $I_j^{(i)}$ into grayscale. Each pixel is represented by a single numeric value in the range of 0 to 255. Denote by $\mathcal{A}$ the alphabet of these values (same as $\mathcal{A}$ of Algorithm 2P).
      B. Scan grayscale of $I_j^{(i)}$ from top left to bottom right to form a string of symbols from $\mathcal{A}$.

C. Denote the string by $X_{i,j}$
III. **end for;**
IV. $v_i = [X_{i,1}, \ldots, X_{i,m_i}]$
8. **end for;**
9. **for** $(l := 1 \text{ to } M)$
I. **for** $(k := 1 \text{ to } L_l)$
A. Transform $P_k^{(l)}$ into grayscale. Each pixel is represented by a single numeric value in the range of 0 to 255. Denote by $\mathcal{A}$ the alphabet of these values (same as $\mathcal{A}$ of Algorithm 2P).
B. Scan grayscale image of $P_k^{(l)}$ from top left to bottom right to form a string of symbols from $\mathcal{A}$
C. Denote the string by $Y_{l,k}$
II. **end for;**
10. **end for;**
11. $u := [Y_{1,1}, Y_{1,2}, \ldots, Y_{1,L_1} \ldots, Y_{M,1}, \ldots Y_{M,L_M}]$

Algorithm 3P: continued...

12. **Launch** blocks B_q, $1 \leq q < Q$, in parallel, each executes the code below
1. **for** $(x = 0$ **to** $\lfloor N/Q \rfloor)$
I. initialize index vector $i = [i_1, \ldots, i_Q]$ where $i_q = q + x \cdot Q$
II. **if** $(i_q \leq N)$
A. $D_q = \texttt{DMat}\left(v_{i_q}, u\right)$
B. **Initialize** counts $c_r = 0$, $1 \leq r \leq L$
i. **for** $(j := 1 \text{ to } m_{i_q})$ **do**
a. $minDist := 1$
b. $closestProt := 1$
c. $protIndex = 1$
d. **for** $(l := 1 \text{ to } M)$ **do**
1. **for** $(k := 1 \text{ to } L_l)$ **do**
I. **if** $(D_q[j,k] < minDist)$
A. $minDist := D_q[j,k]$
B. $closestProt := protIndex$
II. $protIndex := protIndex + 1$
III. **end if;**
2. **end for;**
e. **end for;**
f. **Increment** the count, $c^{(q)}_{closestProt} := c^{(q)}_{closestProt} + 1$
ii. **end for;**
C. **Normalize** the counts, $V_r^{(q)} := \frac{c_r^{(q)}}{\sum_{r'=1}^{L} c_{r'}^{(q)}}$, $1 \leq r \leq L$
D. $V^{(q)} = \left[V_1^{(q)}, \ldots V_L^{(q)}\right]$ as the L-dimensional feature-vector (case) representation for image I_{i_q}
E. $R[i_q] = V^{(q)}$
2. **end for;**
13. Output: the set R of cases corresponding to the set $\mathcal{I}$ of input images

Algorithm 4: Image classification learning

1. **Input**: (1) a target class variable T taking values in a finite set $\mathcal{T}$ of class categories, (2) a set $\mathcal{D}_T$ of L-dimensional cases labeled with values in $\mathcal{T}$ (3) any supervised learning algorithm $\mathcal{A}$
2. Partition $\mathcal{D}_T$ using n-fold cross validation into Training and Testing sets of cases
3. Train and test algorithm $\mathcal{A}$ and produce a classifier C which maps the feature space $[0,1]^L$ into $\mathcal{T}$
4. Define Image classifier as follows: given any image I the classification of I is $F(I) := C(v(I))$, where $v(I)$ is the L-dimensional feature vector of I
5. **Output**: classifier F

5. EXPERIMENTS AND RESULTS

In this section, we describe experiments and report on the classification accuracy and the computational processing performance. The setup consists of a PC with a 2.8Ghz AMD Phenom II X6 1055T CPU (with six cores). A GPU hardware on this PC is a Tesla K20C board with a single GK110 GPU from Nvidia Corp.. This GPU is based on the Keppler architecture (with compute capabilities of 3.5). The algorithm is implemented on the CUDA programming platform (release 6.0) and the operating system is Ubuntu Linux 2.6.38-11-generic.

We started by testing the algorithm on several two-category image classification problems that were obtained from the CALTECH-101 testbench image set [8]. We present one such problem of classification into the categories *airplane* and *ketch* (yacht). Ten prototypes of each category were chosen by collecting small images of airplanes and boats. The prototypes of *airplane* are of size 150×70 pixels and the prototypes of *ketch* are of size 150×130. Figure **12** shows a few examples of such prototypes. The set of input images consists of 74 images of airplanes of size 420×200 and 100 images of yachts of size 300×300. It takes 345 seconds for Algorithm 3P to produce the 174 cases from the input image set. Figure **13** displays two examples of input images, one from category *airplane* and one from *ketch* along with their corresponding partition into sub-images of size 150×150 (obtained in Algorithm 3P, step 7, om page 17). As mentioned in section 4.2 the algorithm permits the size of prototypes to differ and the size (or number) of sub-images to differ from one feature category to another. We ran four learning algorithms, multi-layer perceptrons, decision trees J48, naive-Bayes and lazy IB1, on a ten-fold cross validation using the 174 input images. Table **4** presents the accuracy results versus the baseline algorithm (rules.ZeroR), which classifies based on the prior class probability. The configuration parameter values of the learning algorithms used in WEKA [11] are displayed under the accuracy result. As can be seen, the $J48$ decision tree learner achieves the highest accuracy of 96.54% (relative to the baseline accuracy of 57.52%).

Next, we considered a problem of recognizing different image textures. The input set consists of the 1000 images of the Texture Database [13] test-bench which has 25 categories of various types of real textures, for instance, the texture of glass, water, wood, etc.. Each category has 40 images of size 640×480. We chose as feature categories the classification categories themselves and selected five small prototypes of size 150×150 from each one without using Algorithm 2P (just picking parts of images in a random way to be prototypes). It takes about 25 hours for Algorithm 3P to produce 1000 cases corresponding to the images. We ran the following classification learning algorithms: lazy IB1, decision trees J48, multi-layer perceptrons, naive Bayes, random forest. Ten fold cross validation accuracy results are displayed in Table **5** (parameter settings are displayed under the accuracy results). As shown, the best obtained accuracy result is 70.73% which is achieved by the

Figure 12: Three prototypes from category *airplane*.

Table 4: Classification Result for *Airplane* v.s. *ketch* Problem

Dataset	(1)	(2)	(3)	(4)	(5)
airplane-ketch	57.52	83.65 ∘	93.82 ∘	96.54 ∘	86.75 ∘

∘ , • statistically significant improvement or degradation
(1) rules.ZeroR '' 48055541465867954
(2) functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a' -5990607817048210779
(3) lazy.IB1 '' -6152184127304895851
(4) trees.J48 '-C 0.25 -M 2' -217733168393644444
(5) bayes.NaiveBayesMultinomialUpdateable '' -7204398796974263186

Figure 13: Input images from category *airplane* and *ketch* and their respective sub-images.

Table 5: Classification Result for the Texture Problem

Dataset	(1)	(2)	(3)	(4)	(5)	(6)
cat-40img	4.00	63.16 ∘	58.59 ∘	66.50 ∘	61.92 ∘	70.73 ∘

∘ , • statistically significant improvement or degradation.
(1) rules.ZeroR '' 48055541465867954
(2) lazy.IB1 '' -6152184127304895851
(3) trees.J48 '-C 0.25 -M 2' -217733168393644444
(4) functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a' -5990607817048210779
(5) bayes.NaiveBayesMultinomialUpdateable '' -7204398796974263186
(6) trees.RandomForest '-I 100 -K 0 -S 1' -2260823972777004705

random forest algorithm; this is 17.6 times better than the baseline ZeroR classification rule.

As the final problem, we tested the algorithm on the classification of aerial images. We used the Banja Luka dataset[1] from [18] which consists of the six categories (mentioned in section 4.2) and 606 images of size 128×128 pixels. As the feature categories, we use the categories themselves. Four prototypes from each feature category were selected in a way that captures the different textures that we see in typical images of that category. As mentioned above, a main advantage of our approach is that the selection of prototypes can be done by a non-expert and does not involve any sophisticated data analysis or interpretation. Figue **14**

[1]Available: http://dsp.etfbl.net/aerial/

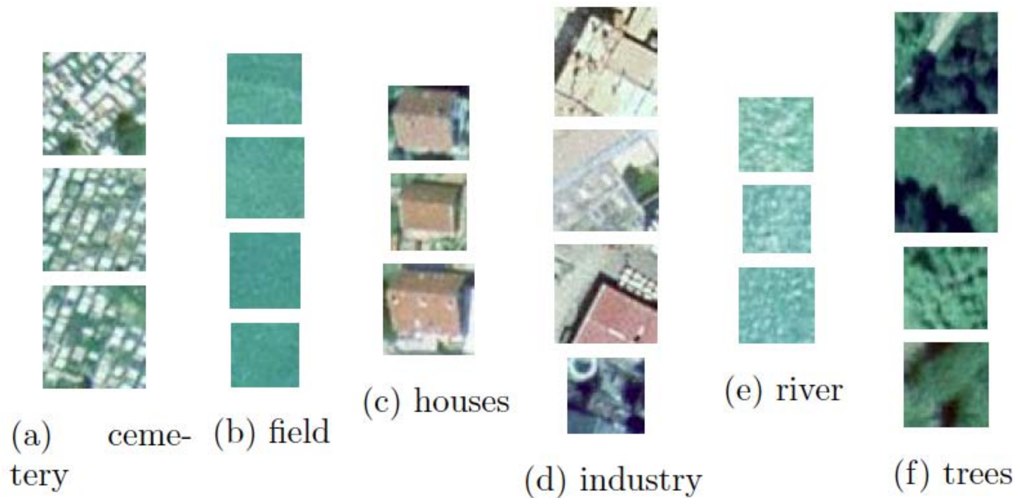


Figure 14: Prototypes for the Banja Luka dataset.

displays the set of prototypes that were used. It takes 0.9 second for Algorithm 2P to run, namely, to read the 21 prototype images, convert them into strings, compute the $21^2 = 441$ pairs of inter-prototype distances, and finally write the distance matrix into an output file, which is then displayed as a dendrogram by an external Java-based code.

Figure **3** shows examples of input images from the six categories and Figure **10** displays the 16 sub-images that are created by Algorithm 3P for input image in Figure **9**. It takes 223 seconds for Algorithm 3P to produce the 606 cases. Note that this involves computing $606 \times 16 \times 21 = 203,616$ UID calculations. Each case corresponds to one input image, and is a vector of dimensionality 21 which consists of the normalized counts of the number of times that a prototype was closest to a submerge of the image.

With the cases ready, we ran the following learning algorithms, SVM (libSVM), SMO, lazy.IBk, J48, RandomForest and RandomTree each on a ten-fold cross validation using the 606 cases that were produced by Algorithm 3P. Table **6** presents the accuracy results versus the baseline algorithm (rules.ZeroR), which classifies based on the prior class probability. The configuration parameter values of the learning algorithms, used in WEKA [11] are displayed under the accuracy result. As can be seen, the RandomForest learner achieves the highest accuracy of 80.42% (relative to the baseline accuracy of 29.37%). The reason that the baseline accuracy is not $(1/6)100\%$ is because the distribution of number of cases per category is not uniform. Our accuracy result is within the range of accuracies of 79.5% to 87.3% reported in [18] who use more sophisticated and advancd image-processing based on ST-descriptors and a complex filter bank. Table **7** displays the confusion matrix for the Banja Luka problem as learnt by the RandomForest algorithm and is similar to the one reported in [18].

All the experiments that we have conducted, including the above, led us to use the heuristic of choosing prototypes (described in section 4.2). In selecting prototypes, every prototype should capture a characteristic of its feature category which is represented via the texture, rather than letting a prototype contain a mixture of many different textures. The confusion between images from the *industry* category, which are being classified as *houses*, is relatively high as can be seen from the fourth row of the confusion matrix (Table **7**). Trying a different choice of prototypes for *industry* did not improve this misclassification but we believe that with some more trial and error, one can find prototypes that improve it.

Table 6: Classification Result for the BanjaLuka Problem, 24 Prototypes

Dataset	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Zone-21-16X16	30.36	54.07 ∘	73.89 ∘	70.53 ∘	72.58 ∘	80.42 ∘	66.42 ∘

∘ , • statistically significant improvement or degradation.
(1) ZeroR ''
(2) LibSVM '-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model Weka-3-7 -seed 1'
(3) SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K PolyKernel -E 1.0 -C 250007'
(4) IBk '-K 1 -W 0 -A LinearNNSearch -A EuclideanDistance -R first-last'
(5) J48 '-C 0.25 -M 2'
(6) RandomForest '-I 100 -K 0 -S 1 -num-slots 1'
(7) RandomTree '-K 0 -M 1.0 -V 0.001 -S 1'

Table 7: Confusion Matrix for the Banja-Luca Problem, as Learnt by the Random Forest Algorithm

cemetery	field	houses	industry	river	trees	⇐ classified as
6	1	16	0	0	0	cemetery
0	159	1	3	3	1	field
2	5	120	3	0	3	houses
0	10	15	38	1	1	industry
0	22	2	1	36	6	river
0	3	3	0	3	86	trees

Considering how little effort and no-expertise is needed in our approach to learning image classification, and the naive representation of an image as a string of symbols, it is surprising that we obtained accurate classification rates on a variety of problems; in the Banja Luka test bench, our best result is in the accuracy range of results reported in [18] who use sophisticated image descriptors.

Our algorithm can serve well in settings with little or no domain knowledge and relying solely on a non-expert to select the prototypes. The algorithm can also serve as a starting point from which more sophisticated analysis and specialized feature extraction can be made. Lastly, the cases produced by Algorithm 3P may also be used for unsupervised learning and to learn clustering of images.

REFERENCES

[1] Belousov A. Massively parallel computations for image classification. Master's thesis, Ariel University, http://www.ariel.ac.il/sites/ratsaby/Theses/alex.pdf, 2015.

[2] Belousov A, Ratsaby J. Massively parallel computations of the LZ-complexity of strings,. In Proc. of the 28th IEEE Convention of Electrical and Electronics Engineers in Israel (IEEEI'14), pages pp. 1-5, Eilat, Dec. 3-5 2014. https://doi.org/10.1109/EEEI.2014.7005885

[3] Belousov A, Ratsaby J. A parallel distributed processing algorithm for image feature extraction. In Advances in Intelligent Data Analysis XIV - 14th International Symposium, IDA 2015, Saint-Etienne, France, October 22-24, 2015. Proceedings, volume 9385 of Lecture Notes in Computer Science. Springer, 2015.

[4] Cheng H. D. Shan J, Ju W, Guo Y, Zhang L. Automated breast cancer detection and classification using ultrasound images: A survey. Pattern Recognition 2010; 43(1): 299-317. https://doi.org/10.1016/j.patcog.2009.05.012

[5] Chester U, Ratsaby J. Universal distance measure for images. In Proceedings of the 27th IEEE Convention of Electrical Electronics Engineers in Israel (IEEEI'12), pages 1-4, Eilat, Israel, November 14-17, 2012. https://doi.org/10.1109/EEEI.2012.6377115

[6] Chester U, Ratsaby J. Machine learning for image classification and clustering using a universal distance measure. In Brisaboa N, Pedreira O, Zezula P, Eds., Proceedings of the 6th International Conference on Similarity Search and Applications (SISAP'13), volume 8199 of Springer Lecture Notes in Computer Science 2013; pp. 59-72. https://doi.org/10.1007/978-3-642-41062-8_7

[7] Cilibrasi R, Vitanyi P. Clustering by compression. IEEE Transactions on Information Theory 2005; 51(4): 1523-1545. https://doi.org/10.1109/TIT.2005.844059

[8] Fei-Fei L, Fergus R, Perona P. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. Computer Vision and Image Understanding 2007; 106(1): 59-70. Special issue on Generative Model Based Vision. https://doi.org/10.1016/j.cviu.2005.09.012

[9] Galar M, Derrac J, Peralta D, Triguero I, Paternain D, Lopez-Molina C, García S, Benítez J. M. Pagola M, Barrenechea E, Bustince H, Herrera F. A survey of fingerprint classification part i: Taxonomies on feature extraction methods and learning models. Knowledge-Based Systems 2015; 81: 76-97. https://doi.org/10.1016/j.knosys.2015.02.008

[10] Gonzalez RC, Woods R. E. Digital Image Processing (3rd Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[11] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten I. H. The WEKA data mining software: An update. SIGKDD Explorations 2009; 11(1): 10-18. https://doi.org/10.1145/1656274.1656278

[12] Haralick R. M. Shanmugam K, Dinstein I. Textural features for image classification. Systems, Man and Cybernetics, IEEE Transactions on, SMC 1973; 3(6): 610-621. https://doi.org/10.1109/TSMC.1973.4309314

[13] Lazebnik S, Schmid C, Ponce J. A sparse texture representation using local affine regions. IEEE Trans Pattern Anal Mach Intell 2005; 27(8): 1265-1278. https://doi.org/10.1109/TPAMI.2005.151

[14] Lu D, Weng Q. A survey of image classification methods and techniques for improving classification performance. Int J Remote Sens 2007; 28(5): 823-870. https://doi.org/10.1080/01431160600746456

[15] Ojala T, Pietikainen M, Harwood D. A comparative study of texture measures with classification based on featured distributions. Pattern Recognition 1996; 29(1): 51-59. https://doi.org/10.1016/0031-3203(95)00067-4

[16] Pham D.T. Alcock RJ. Chapter 5 -classification. In D.T. PhamR.J. Alcock, editor, Smart Inspection Systems, Academic Press, London 2003; pp. 129-155. https://doi.org/10.1016/B978-012554157-2/50005-5

[17] Raju J, Durai C. A. D. A survey on texture classification techniques. In Information Communication and Embedded Systems (ICICES), 2013 International Conference on, 2013; pp. 180-184. https://doi.org/10.1109/ICICES.2013.6508183

[18] Risojevic V, Babic Z. Aerial image classification using structural texture similarity. In Proceedings of the IEEE International Symposium on Signal Processing and Information Technology (ISSPIT) 2011; pp. 190-195. https://doi.org/10.1109/ISSPIT.2011.6151558

[19] Sayood K, Otu H. H. A new sequence distance measure for phylogenetic tree construction. Bioinformatics 2003; 19(16): 2122-2130.
https://doi.org/10.1093/bioinformatics/btg295

[20] Ziv J, Lempel A. On the complexity of finite sequences. IEEE Transactions on Information Theory 1976; 22(3): 75-81.
https://doi.org/10.1109/TIT.1976.1055501

Received on 15-07-2019 Accepted on 03-10-2019 Published on 14-10-2019

DOI: http://dx.doi.org/10.15377/2409-5761.2019.06.1

© 2019 Belousov and Ratsaby; Avanti Publishers.
This is an open access article licensed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted, non-commercial use, distribution and reproduction in any medium, provided the work is properly cited.