A Parallel Distributed Processing Algorithm for Image Feature Extraction

Alexander Belousov and Joel Ratsaby(✉)

Electrical and Electronics Engineering Department, Ariel University, Ariel, Israel
ratsaby@ariel.ac.il
http://www.ariel.ac.il/sites/ratsaby/

Abstract. We present a new parallel algorithm for image feature extraction. which uses a distance function based on the LZ-complexity of the string representation of the two images. An input image is represented by a feature vector whose components are the distance values between its parts (sub-images) and a set of prototypes. The algorithm is highly scalable and computes these values in parallel. It is implemented on a massively parallel graphics processing unit (GPU) with several thousands of cores which yields a three order of magnitude reduction in time for processing the images. Given a corpus of input images the algorithm produces labeled cases that can be used by any supervised or unsupervised learning algorithm to learn image classification or image clustering. A main advantage is the lack of need for any image processing or image analysis; the user only once defines image-features through a simple basic process of choosing a few small images that serve as prototypes. Results for several image classification problems are presented.

1 Introduction

Image classification research aims at finding representations of images that can be automatically used to categorize images into a finite set of classes. Typically, algorithms that classify images require some form of pre-processing of an image prior to classification. This process may involve extracting relevant features and segmenting images into sub-components based on some prior knowledge about their context [1,2]. In [3] we 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 then uses the string distance of [4] to give the distance value between the images. According to [4] 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. By complexity of a string x we mean the Lempel-Ziv complexity [5]. In [6] we presented a serial algorithm to convert images into feature vectors where the i^{th} dimension is a feature that measures the UID distance between the image and the i^{th} feature category. One of the advantages of the UID is that it can compare the distance between two

© Springer International Publishing Switzerland 2015
E. Fromont et al. (Eds.): IDA 2015, LNCS 9385, pp. 61–71, 2015.
DOI: 10.1007/978-3-319-24465-5_6

images of different sizes and thus the prototypes which are representative of the different feature categories may be relatively small. For instance, a prototype of *airplane* category can be a small image of an airplane over a simple background such as blue sky.

In this paper we introduce a parallel distributed algorithm which is based on the serial algorithm of [6]. Compared to [6] the current version of the algorithm offers a very large acceleration in processing speed which allows us to test the algorithm on more challenging image classification tasks. 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 doing this for the corpus of images, each labeled by its class, yields a data set that can be used to train any 'off-the-shelf' supervised or unsupervised learning algorithm. After describing our method in details we report on the speed and accuracy that are achieved by this method.

It is noteworthy that our process for converting an image into a finite dimensional feature vector is very straightforward and does not involve any domain knowledge or image analysis expertise. Compared to other image classification algorithms that extract features based on sophisticated mathematical analysis, for instance, analyzing the texture, or checking for special properties of an image, our approach is very basic and universal. It is based on the complexity of the 'raw' string-representation of an image. Our approach is to extract features automatically just by computing distances from a set of prototypes images that are selected once at the first stage.

The algorithm that we present here is designed with the main aim of *scalable* distributed computations. Building on recent ideas [7], we designed it to take advantage of relatively cheap and massively-parallel processors that are ubiquitous in today's technology. Our method extracts image features that are unbiased in the sense that they do not employ any heuristics in contrast to other common image-processing techniques [1,2]. The features that we extract are based on information implicit in the image and obtained via a complexity-based UID distance which is an information-theoretic measure.

2 Distance

The UID distance function [3] is based on the LZ- complexity of a string. The definition of this complexity follows [4,5]: let S, Q and R be strings of characters that are defined over the alphabet $\mathcal{A}$. Denote by $l(S)$ the length of S, and $S(i)$ denotes the i^{th} 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 $S \rightarrow S$) if there exists an integer $p \leq l(S)$ such that $Q(k) = R(p+k-1)$ for $k = 1, \ldots, l(Q)$. For example, $aacgt \rightarrow aacgtcgtcg$ with $p = 3$ and $aacgt \rightarrow aacgtac$ 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 p^{th} 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 $S(1,j) \rightarrow S(1,l(S)-1)$. 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 a *production process* where at its i^{th} 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 i^{th} 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 i^{th} 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*. A history is called exhaustive if each of its components is exhaustive. Every string S has a unique exhaustive history [5]. Let us denote by $c_H(S)$ the number of components in a history of S. The LZ complexity of S is $c(S) = \min\{c_H(S)\}$ 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 [4] and is defined as follows: given two strings X and Y of any finite alphabet, denote by XY their concatenation then define

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

(see several normalized versions of d in [4]). In [3,6] we have found that the following normalized distance

$$\mathtt{d}(X,Y) := \frac{c(XY) - \min\{c(X), c(Y)\}}{\max\{c(X), c(Y)\}} \tag{1}$$

is useful for image classification.

In [7] we introduced a parallel distributed processing algorithm (LZMP) for computing the complexity $c(X)$ of a string X. Let us denote by $\mathtt{dp}(X,Y)$ the distance between X and Y where the complexity c is computed by the LZMP algorithm. Thus (1) is now represented by its parallel counterpart

$$\mathtt{dp}(X,Y,a,b) := \frac{LZMP(XY) - \min\{a,b\}}{\max\{a,b\}} \tag{2}$$

where a, b are the LZ-complexity values of the string X, Y, respectively, and for efficiency they are pre-computed as seen for instance in Procedure DMat, step 2(IV).

3 The Algorithm

We describe the parallel algorithm for image feature extraction, starting with a listing of several procedures followed by the main part which is split into several

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:**
   - I. H history buffer
   - II. $m \coloneqq 0$, length of history buffer
   - III. $d \coloneqq 0$ number of components in exhaustive history
   - IV. SM shared memory variable common to all threads
   - V. Q number of computing threads
   - VI. $\{T_q\}_{q=1}^{Q}$, 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. $SM \coloneqq 0$
     - 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 $i_{j^{(q)}} \coloneqq 0$
         - b. initialize variable $k_{j^{(q)}} \coloneqq j^{(q)}$
         - c. initialize variable $h_{j^{(q)}} \coloneqq m - j^{(q)}$
         - d. **while**$(H[k_{j^{(q)}}] = S[m + i_{j^{(q)}}])$
           1. $i_{j^{(q)}} \coloneqq i_{j^{(q)}} + 1$
           2. $k_{j^{(q)}} \coloneqq k_{j^{(q)}} + 1$
           3. $h_{j^{(q)}} \coloneqq h_{j^{(q)}} - 1$
           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 $z_{j^{(q)}} \coloneqq m$
           2. **while**$(S[z_{j^{(q)}}] = S[m + i_{j^{(q)}}])$
              - I. $z_{j^{(q)}} \coloneqq z_{j^{(q)}} + 1$
              - II. $i_{j^{(q)}} \coloneqq i_{j^{(q)}} + 1$
              - 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. $SM \coloneqq i_{j^{(q)}}$, // 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. $H \coloneqq H + \text{substring}(S[m], S[m + SM + 1])$
       - ii. $d \coloneqq d + 1$
       - iii. $m \coloneqq m + SM + 1$
     - 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

sub-algorithms. Procedure LZMP computes the LZ-complexity of a given string. It runs in parallel over the symbols that comprise the string. The procedure appears in [7] and we enclose it here for completeness. Procedure VLZMP computes the LZ-complexity of a set of input strings in parallel. Procedure DMat computes the UID distance of every pair of strings from two given input lists, in parallel. 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 main algorithm is split into sub-algorithms (as done in [6]) which are numbered from 2 to 4 and the letter P denotes that it is a parallel computing algorithm. Algorithm 2P selects the prototype images (its serial version is Algorithms 1 and 2 of [6]).

Algorithm 2P. Prototypes selection

1. **Input**: M image feature categories, and a corpus $\mathcal{C}_N$ 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 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 i^{th} 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 computes the set of cases (feature vectors) for images in the input corpus. 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 the input images into strings of symbols.

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

1. **Input**: vector $v := \{v[i]\}_{i=1}^{k} = [S_1, S_2, S_3, \ldots, S_k]$ 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 , $1 \le q \le n$, 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** $(i_q \le k)$
         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 $B_{p,q}$, $1 \le p < M$, $1 \le q < N$, 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** $(i_{p,q} \le m$ **and** $j_{p,q} \le n)$
               1. $D\left[i_{p,q}, j_{p,q}\right] = \mathtt{dp}\left(v\left[i_{p,q}\right], u\left[j_{p,q}\right], a\left[i_{p,q}\right], b\left[j_{p,q}\right]\right)$
            b. **end if;**
      C. **end for;**
   II. **end for;**
4. **Output:** $DMat(v,u) = D$

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).
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 sub-images of 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** $(l := 1$ to $M)$
   I. **for** $(k := 1$ 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 4 is identical to that of [6] and we present it for completeness. It uses the training cases that are produced in Algorithm 3P and uses any off-the-shelf supervised learning algorithm to produce a classifier.

4 Results

The following hardware was used: a 2.8 Ghz AMD Phenom©II X6 1055T Processor with number of cores $n = 6$ and a Tesla K20C board with a single GK110 GPU from nVIDIA. This GPU is based on the Keppler architecture

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. $D_q = \texttt{DMat}\left(v_{i_q}, u\right)$
                i. **for** $(j := 1$ to $m_{i_q})$ **do**
                    a. **for** $(l := 1$ to $M)$ **do**
                        1. $temp := 0$
                        2. **for** $(k := 1$ to $L_l)$ **do**
                            I. $temp := temp + (D_q[j,k])^2$
                        3. **end for;**
                    b. $temp = (1/L_l) \cdot temp$
                    c. $r_l^{(q)} := \sqrt{temp}$
                    d. **end for;**
                    e. Let $l_q^*(j) := \text{argmin}_{1 \leq l \leq M} r_l^{(q)}$, this is the decided feature category for sub-image $I_j^{(i_q)}$
                    f. **Increment** the count, $c_{l_q^*(j)}^{(q)} := c_{l_q^*(j)}^{(q)} + 1$
                ii. **end for;**
            B. **Normalize** the counts, $V_l^{(q)} := \frac{c_l^{(q)}}{\sum_{z=1}^{M} c_z^{(q)}}$, $1 \leq l \leq M$
            C. $V^{(q)} = \left[V_1^{(q)}, \ldots V_M^{(q)}\right]$ as the feature-vector (case) representation for image I_{i_q}
            D. $W[i_q] = V^{(q)}$
    2. **end for;**
13. Output: the array W of cases corresponding to the set $\mathcal{I}$ of input images

(with compute capabilities of 3.5). The CUDA is release 6.0 and the operating system is Ubuntu Linux 2.6.38-11-generic.

We tested the algorithm on several two-category image classification problem obtained from the CALTECH-101 corpus [1]. Due to the lack of space, we present one such problem which has as categories, *airplane* and *ketch* (yacht). We chose 10 prototypes of each category simply 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 1 shows a few examples of such prototypes.

The corpus of input images consist 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 starting from the image corpus. Figure 2 displays two examples of input images, one from category *airplane* and one from *ketch* and their corresponding divisions into sub-images of size 150×150 (obtained in Algorithm 3P, step 7). Note that 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

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

Fig. 1. Three prototypes from category *airplane*

Fig. 2. Input images from category *airplane* and *ketch* and their respective sub-images

trees J48, naive-Bayes and lazy IB1, on a ten-fold cross validation using the 174 input images. Table 1 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 [8] 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 %).

Table 1. 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

Next, we considered a more challenging problem of recognizing different image textures. We obtained the 1000 images of the Texture Database [2] which has 25 categories of various types of real textures, for instance, glass, water, wood, with 40 images each of size 640 × 480 per category. We chose as feature categories the 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 h for Algorithm 3P to produce the 1000 cases starting from the image corpus. 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 2 (parameter settings are displayed under the accuracy results). As shown, the best obtained accuracy result is 70.73 % which is achieved by the random forest algorithm; this is 17.6 times better than the baseline ZeroR classification rule.

Table 2. Classification result for the texture problem

Dataset	(1)	(2)	(3)	(4)	(5)	(6)
25cat-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

Considering how little effort and no-expertise is needed in our approach to image feature extraction, we believe that the results are impressive and can serve well in settings where very little domain knowledge is available, or as a starting

point from which additional analysis and specialized feature extraction can be made.

5 Conclusions

In this paper we introduce a new parallel processing algorithm for image feature extraction. Given an input corpus of raw RGB images the algorithm computes feature vectors (cases) that represent the images with their associated classification target labels. Using these cases, any standard supervised or unsupervised learning algorithm can learn to classify or cluster the images in the database. A main advantage in our approach is the lack of need for any kind of image or data analysis. Aside of picking once at the start a few small image prototypes, the procedure is automatic and applies to any set of images. It can therefore be very useful in settings with little domain knowledge or as a starting point for a more specialized image data analysis. Our experiments indicate that the algorithm yields relatively high accuracies on image texture classification problems.

Acknowledgement. We acknowledge the support of the nVIDIA corporation for their donation of GPU hardware.

References

1. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories (2004)
2. Lazebnik, S., Schmid, C., Ponce, J.: A sparse texture representation using local affine regions. IEEE Trans. Pattern Anal. Mach. Intell. **27**(8), 1265–1278 (2005)
3. Chester, U., Ratsaby, J.: Universal distance measure for images. In: Proceedings of the 27th IEEE Convention of Electrical Electronics Engineers in Israel (IEEEI 2012), pp. 1–4. Eilat, Israel, 14–17 November 2012
4. Sayood, K., Otu, H.H.: A new sequence distance measure for phylogenetic tree construction. Bioinformatics **19**(16), 2122–2130 (2003)
5. Ziv, J., Lempel, A.: On the complexity of finite sequences. IEEE Trans. Inf. Theory **22**(3), 75–81 (1976)
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.) SISAP 2013. LNCS, vol. 8199, pp. 59–72. Springer, Heidelberg (2013)
7. Belousov, A., Ratsaby, J.: Massively parallel computations of the LZ-complexity of strings. In: Proceedings of the 28th IEEE Convention of Electrical and Electronics Engineers in Israel (IEEEI 2014), pp. 1–5. Eilat, 3–5 December 2014
8. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explor. **11**(1), 10–18 (2009)