Procedia Computer Science

Volume 53, 2015, Pages 400–409

2015 INNS Conference on Big Data

Efficient Skin Segmentation via Neural Networks: HP-ELM and BD-SOM

C. Swaney[1], A. Akusok[2], K.-M. Björk[3], Y. Miche[4], and A. Lendasse[2]

[1] Tippie College of Business, The University of Iowa, Iowa City, USA
[2] Department of Mechanical and Industrial Engineering and the Iowa Informatics Initiative, The University of Iowa, Iowa City, USA
[3] Arcada University of Applied Sciences, Helsinki, Finland
[4] Nokia Solutions and Networks Group, Espoo, Finland

Abstract

This paper presents two novel methods for skin detection: HP-ELM and BD-SOM. Both SOM and ELM are fast for large data sets, but not yet suitable for Big Data. We show how they can be improved in order to fulfill the strict requirements for Big Data. Both new methods are described and their implementations are explained. A comparison on a large example is presented in the experiment section. We find that BD-SOM is more accurate but not as computationally efficient as HP-ELM. As a result, we show that both methods work well on a Big Data task. The given task deals with the classification of more than one billion samples (pixels) between Skin and Non Skin categories.

Keywords: SOM, ELM, Big Data, Image Processing, Skin Detection

1 Introduction

Skin segmentation is a common problem in the field of image processing, where it typically appears as a key preprocessing step in larger problems. For example, a face detection system might benefit from first identifying the parts of an image that represent skin. Another commonly cited application where skin detection is useful is the filtering of adult images on the Internet. There are many and varied skin segmentation systems proposed in the literature, but the most common methods are the pixel-based ones. Amongst this group of classifiers popular methods include Gaussian and mixture of Gaussian models (EM-Algorithm) [1], [2], multilayer perceptron (MLP) [3], and histogram-based or Bayesian classifier methods [4]. There are also numerous studies that provide comparisons of the accuracy of proposed skin segmentation methods, a of which a good example is [5]. Few of these articles take in to consideration computational aspects of skin segmentation, and the applicability of these methods in the presence of big data therefore remains unclear. In this study we improve two general methods

Selection and peer-review under responsibility of the Scientific Programme Committee of INNS-BigData2015

doi:10.1016/j.procs.2015.07.317

from soft-computing–Extreme Learning Machines (ELM) and Self-Organizing Maps (SOM)– and show how they can be adapted to provide big data skin segmentation solutions. In this paper, we only consider the problem of skin segmentation, but we hope to illustrate the ability of these two methods to provide efficient answers to a variety of classification problems involving large amounts of data. These methods may be useful in medical research labs, for example, where the ability of researchers to produce data (possibly in the form of highly detailed images) may outpace the lab's ability to analyze data. Properly packaged the methods we consider here provide researchers with efficient tools to increase productivity. The remainder of the paper is organized as follows. Section 2 explains the novel HP-ELM and BD-SOM methods that we implement in our experiments. In Section 3, we describe our experiment and provide results. Section 4 concludes.

2 Methods

2.1 ELM

Extreme Learning Machine [6, 7] (ELM) algorithm is a fast way of training a Single Hidden Layer Feed-forward Neural Network (SLFN). The main concept behind the ELM is the random initialization of the SLFN internal weights and biases, which reduces a computationally expensive iterative optimization (back-propagation [8], Levenberg-Marquardt [9], etc.) to a linear system problem in the output layer, with an exact solution and efficient programming libraries for computing it (LAPACK, MAGMA)[1]. An ELM is a universal function approximator [10] given that the hidden nodes weights are generated randomly and the activation function is a bounded non-constant piecewise continuous one. It is a widely applicable regression and classification algorithm [11]. It is expressed as the following: Consider a set of N data samples in d-dimensional space $\mathbf{X} \in \mathbb{R}^{N\times d}$, and the associated c-dimensional output targets $\mathbf{T} \in \mathbb{R}^{N\times c}$. For classification, there are c targets for c classes given as $\mathbf{T} \in \{0,1\}^{N\times c}$, where the correct class for a sample is set to 1, and all irrelevant classes are set to 0. A binary classification problem is considered a multi-class problem with 2 classes. A SLFN with nn neurons in the hidden layer can be expressed as:

$$\sum_{i=1}^{nn} f\left(\sum_{j=1}^{d} \mathbf{X}_{nj}\mathbf{W}_{ji} + b_i\right)\boldsymbol{\beta}_{ik} = \mathbf{T}_{nk} + \epsilon,\ n = \{1, N\},\ k = \{1, c\} \tag{1}$$

where $\mathbf{W}$ are hidden layer weights, b are hidden layer biases, f is an activation function (sigmoid or hyperbolic tangent), $\boldsymbol{\beta}$ output weights and ϵ is noise because in general an ELM has more data samples than neurons, thus the linear system is over-determined and an exact solution is impossible. Weights $\mathbf{W}$ are generated randomly from a normal distribution $\mathbb{N}(0, \sqrt{3/d})$ to compensate for a large input dimensionality, and biases b from a normal distribution $\mathbb{N}(0, 1)$. Compactly that system is written as $\mathbf{H}\boldsymbol{\beta} = \mathbf{T} + \epsilon$, where the matrix $\mathbf{H}_{N\times l}$ is hidden layer output with elements

$$\mathbf{H} = \begin{bmatrix} \phi(\sum_{j=1}^{d} \mathbf{X}_{1j}\mathbf{W}_{j1} + b_1) & \cdots & \phi(\sum_{j=1}^{d} \mathbf{X}_{1j}\mathbf{W}_{jnn} + b_{nn}) \\ \vdots & \ddots & \vdots \\ \phi(\sum_{j=1}^{d} \mathbf{X}_{Nj}\mathbf{W}_{j1} + b_1) & \cdots & \phi(\sum_{j=1}^{d} \mathbf{X}_{Nj}\mathbf{W}_{jnn} + b_{nn}) \end{bmatrix} \tag{2}$$

[1] LAPACK: `http://www.netlib.org/lapack/`, MAGMA: `http://icl.cs.utk.edu/magma/`

The system in equation 1 is over-specified with $N > nn$, and weights $\boldsymbol{\beta}$ are calculated by the Ordinary Least Squares method using a Moore-Penrose pseudo-inverse [12] $\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}$, and the ELM predictions $\mathbf{Y}$ for data $\mathbf{X}$ projected as $\mathbf{H}$ are calculated as $\mathbf{Y} = \mathbf{H}\boldsymbol{\beta}$.

2.2 High Performance ELM (HP-ELM)

An ELM is a universal approximator if there are enough neurons. In practice, if the amount of data samples is very large, the optimal amount of neurons is also large. This makes the size of matrix $\mathbf{H}$ prohibitively big to store it is any computer's memory, which is required for the solution. A large memory-constrained ELM task is solved differently. The Best Linear Unbiased Estimator (BLUE) gives the optimal least squares solution to the matrix equation $\mathbf{X}\boldsymbol{\beta} = \mathbf{T}$ for stochastic vectors $\mathbf{x}$ and $\mathbf{t}$ combined into the corresponding matrices. It uses two theoretical correlation matrices

$$\mathbb{E}[\mathbf{x}^T\mathbf{x}] = \mathbf{C}_{xx},\ \mathbb{E}[\mathbf{x}^T\mathbf{t}] = \mathbf{C}_{xt} \tag{3}$$

which are assumed to be known. The BLUE of $\mathbf{T}$, denoted by $\mathbf{Y}$, is then

$$\mathbf{Y} = \mathbf{C}_{xx}^{-1}\mathbf{C}_{xt}\mathbf{X} = \boldsymbol{\beta}\mathbf{X}. \tag{4}$$

The inverse of $\mathbf{C}_{xx}$ exists because x is a stochastic variable for which $\mathbf{C}_{xx} = \mathbb{E}[\mathbf{x}^T\mathbf{x}]$ has a full rank.

The ELM problem has a finite amount of projected data samples $\mathbf{H}$ and corresponding targets $\mathbf{T}$, so the correlation matrices are replaced by their estimations

$$\mathbf{C}_{xx} \approx \mathbf{H}^T\mathbf{H} = \boldsymbol{\Omega}_h,\ \mathbf{C}_{xt} \approx \mathbf{H}^T\mathbf{T} = \boldsymbol{\Omega}_t, \tag{5}$$

and the ELM output weights are computed from those estimates

$$\boldsymbol{\beta} = \boldsymbol{\Omega}_h^{-1}\boldsymbol{\Omega}_t = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{T}. \tag{6}$$

The inverse of $\boldsymbol{\Omega}_h = \mathbf{H}^T\mathbf{H}$ matrix exists if it has full rank. In ELM model, the random projection produces almost orthogonal features (columns of $\mathbf{H}$) which are linearly independent. The number of hidden neurons (columns of $\mathbf{H}$) is smaller than the number of training samples (rows of $\mathbf{H}$), otherwise the liner model will learn training samples perfectly and overfit. Under such constraints, rank of matrix $\mathbf{H}$ equals its number of columns, thus matrix $\mathbf{H}^T\mathbf{H} = \boldsymbol{\Omega}_h$ is full rank and its inverse exists. If numerical instabilities are faced in the inverse, a regularization term is applied to the correlation matrix $\boldsymbol{\Omega}_h = \mathbf{H}^T\mathbf{H} + \alpha\mathbf{I}$, where α is a small positive constant.

Model Structure Selection of HP-ELM

An optimal number of hidden neurons nn (which does not lead to over-fitting) is selected by a minimum validation error. However, an ELM model must be built for each particular nn. The validation process can be sped up by pre-computing the $\boldsymbol{\Omega}_h$ and $\boldsymbol{\Omega}_t$ matrices for an ELM with the largest nn. Obtaining the $\boldsymbol{\Omega}_h = \mathbf{H}^T\mathbf{H}$ matrix is a computationally expensive operation because it has $\mathcal{O}(nn^2)$ complexity. But the result matrix is symmetric, and a solution of a linear system $\boldsymbol{\Omega}_h\boldsymbol{\beta} = \boldsymbol{\Omega}_t$ with a symmetric matrix $\boldsymbol{\Omega}_h$ is easier to compute. The difference in computational time approximately equals the cost of obtaining $\boldsymbol{\Omega}_h = \mathbf{H}^T\mathbf{H}$. Once the matrices $\boldsymbol{\Omega}_h$ and $\boldsymbol{\Omega}_t$ are computed, ELM models with different numbers of neurons $k \leq nn$ can be computed for the validation process by taking sub-matrices $\boldsymbol{\Omega}_h[1..k, 1..k]$ and $\boldsymbol{\Omega}_t[1..k, 1..c]$ and solving the output weights $\boldsymbol{\beta}$ from them.

Accelerated HP-ELM

In ELM implementation with BLUE, only the $\mathbf{\Omega}_h = \mathbf{H}^T\mathbf{H}$ and $\mathbf{\Omega}_t = \mathbf{H}^T\mathbf{T}$ matrices need to be kept in memory — with sizes being independent of the number of data samples N. Furthermore, these matrices may be computed in k separate batches, which reduces the memory requirement for storing the $\mathbf{H}$ matrix k times.

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}^1 \\ \vdots \\ \mathbf{H}^k \end{bmatrix}, \quad \mathbf{\Omega}_h = \mathbf{H}^T\mathbf{H} = \mathbf{H}^{1T}\mathbf{H}^1 + \ldots + \mathbf{H}^{kT}\mathbf{H}^k. \tag{7}$$

The batch computation of $\mathbf{H}^T\mathbf{H}$ is a simple matrix multiplication with relatively low memory requirements and a high computational cost (constituting more than 95% of runtime for an ELM with $nn > 10,000$), so this, along with $\mathbf{H}^T\mathbf{T}$, are ideal parts for GPU acceleration, which significantly reduces the total ELM computational time. The output matrices are accumulated in the GPU memory, and the solution of $\boldsymbol{\beta}$ from $(\mathbf{H}^T\mathbf{H})\boldsymbol{\beta} = (\mathbf{H}^T\mathbf{T})$ is also accelerated by GPU, although this operation is fast anyway because of the symmetric matrices involved.

2.3 BD-SOM

Prior research has identified SOM as a possible alternative pixel-level skin segmentation method [13]. In that paper the authors suggest that the SOM may possess computational advantages that make it more appropriate for real-time applications. We suggest several refinements that further improve both the suitability of the SOM for big data (in terms of computational cost), and the accuracy of the SOM in the face of more heterogeneous data sets requiring larger maps. We refer to this combination of strategies as Big Data SOM, or BD-SOM.

2.3.1 One-Step Training

Standard SOM classification involves two steps: data is first clustered by training the SOM without using class data, and the resulting map is then calibrated by comparing each data point to the map, with neurons that are more frequently "activated" by skin pixels being identified as skin. In the context of big data, the calibration step is costly–in effect it requires us to classify each pixel in our training set in addition to our validation or testing set. We instead employ a one-step version of SOM suggested by [14] for the problem of missing value imputation. Here we regard the unknown class of a pixel as a "missing value". The basic idea of the algorithm is to integrate the calibration step into the training step, which is achieved by using non-class features (colors) to register neuron activation, and using all features (colors and skin) to update map neurons. Formally, we denote an observation by $\mathbf{x}_i = (\tilde{\mathbf{x}}_i, s_i) \in \mathbb{R}^{K+1}$, where $\tilde{\mathbf{x}}_i \in \mathbb{R}^K$ is a vector of non-class features, and $s_i \in \{0, 1\}$ is a binary class variable identifying a pixel as skin ($s_i = 1$) or not skin ($s_i = 0$). Similarly, we denote a map neuron by $\mathbf{m}_j = (\tilde{\mathbf{m}}_j, \sigma_j) \in \mathbb{R}^{K+1}$, where $\tilde{\mathbf{m}}_j \in \mathbb{R}^K$ and σ_j lines in the interval $[0, 1]$. The collection of neurons (i.e., the map), denoted by $\mathcal{M}$, is organized in to a two-dimensional grid so that each neuron occupies a fixed location $\mathbf{q}_j \in \{1, \ldots, D_1\} \times \{1, \ldots, D_2\}$. With this notation an iteration of the one-step SOM involves: (1) randomly selecting a data point $\mathbf{x}_i$, (2) calculating its best-matching unit (BMU) using *non-class* features:

$$\mathbf{m}_{\mathrm{BMU}}(\mathbf{x}_i) = \underset{\mathbf{m}_j \in \mathcal{M}}{\arg\min} \parallel \tilde{\mathbf{x}}_i - \tilde{\mathbf{m}}_j \parallel, \tag{8}$$

and (3) updating the map using all features of the observation and neuron vectors:

$$\mathbf{m}_j^{(t)} = \mathbf{m}_j^{(t-1)} + \theta_t(\mathbf{m}_j^{(t)}; \mathbf{m}_{BMU})(\mathbf{m}_j^{(t)}) L_t(\mathbf{x}_i - \mathbf{m}_j^{(t-1)}), \quad \forall\ \mathbf{m}_j \text{ s.t. } d(\mathbf{m}_{BMU}, \mathbf{m}_j) < R_t. \quad (9)$$

In Eq. (9) d is a specified distance function, θ_t is a decreasing function of $d(\mathbf{m}_{BMU}, \mathbf{m}_j)$, and L_t and R_t are functional parameters chosen to control the rate of learning and the size of the neighborhood to be updated at each iteration, respectively. A map is trained by repeating these steps until a either a convergence criteria or a pre-specified number of iterations is reached. If the non-class features are in fact predictors of the class variable, then we expect neurons in the map that are close to each other in the map space (i.e., $d(\mathbf{q}_j, \mathbf{q}_k)$ is small) to arrive at values that are close in the combined feature space (i.e., $\parallel \mathbf{m}_i - \mathbf{m}_j \parallel$ is small). In particular, we expect $|\sigma_i - \sigma_j|$ to be small.

2.3.2 GPU Classification

Following the training task observations are classified in two steps. First, the BMU of each observation is found via Eq. (8), and the class variables associated with the BMUs (σ_{BMU}) are assigned to the observations. Second, the predicted class of each observation is determined according to a chosen threshold value $\tau \in [0, 1]$: we predict an observation $\mathbf{x}_i$ represents skin if $\sigma_{BMU}(\mathbf{x}_i) > \tau$. For large data sets pixel-by-pixel (serial) classification can become extremely time consuming. Fortunately, unlike the the training task, classification involves a series of independent tasks, which can be performed in parallel. We therefore perform classification in "chunks" of several hundred thousand to a few million pixels (i.e., an image) using a GPU accelerator. Details of our parallel scheme are provided in Section 3.2.

3 Experiments

3.1 Data

Our tests are performed on the Face and Skin Detection (FSD) Database [15], which was constructed by researchers at the University of Wollongong as a test set to aid research on skin segmentation.[2] The data set consists of 4,000 images that have been carefully segmented, and contain a variety of skin colors, backgrounds and lighting conditions. For our experiment we select a subset of 2000 images for training (which is further subdivided for validation), and 2000 images for testing. Summary statistics of the data sets are shown in Table 1. Additional details concerning the distribution of images can be found in [5]. We preprocess the images in this data set as follows. For each pixel we construct a 7×7 pixel "window" centered around the pixel and represent each pixel by the average RGB values over its window. We then attach a dummy variable that indicates whether the pixel corresponds to skin or not. For pixels on the boundary of an image for which a 7×7 window cannot be constructed we use the average RGB values from the closest pixel for which a window can be constructed. This procedure results in a $1,110,708,015 \times 4$ observation matrix.

3.2 Implementation

3.2.1 ELM

The HP-ELM code is written in the Python language, using fast accelerated libraries for numerical computations (Numpy linked with MKL BLAS, and Numexpr for non-linear function

[2] Available at: http://www.uow.edu.au/ phung/download.html.

Table 1: Summary statistics for the data sets used in validation and training.

	Images	Pixels	Pixels/Image	Min/Max	Skin (%)	Min/Max
Training	1300	349,925,302	269,173	11,583/1,228,800	15.70	0.86/65.13
Validation	700	187,544,581	267,920	60,000/1,040,400	18.78	1.13/71.08
Testing	2000	573,238,132	286,619	30,000/3,000,000	20.78	0.75/75.43

application). The GPU-accelerated part uses a MAGMA library based on CUDA, with CUDA matrix-matrix multiplication and a mixed-precision MAGMA linear system solver (which computes the solution in the single precision and performs gradient updates in double precision until convergence). The GPU accelerates a BLUE solution of HP-ELM for 3000 neurons, while HP-ELM prediction and solution with 500 neurons is done on a CPU. The workstation to run experiments has a 4-core 4.5 GHz CPU and GTX Titan Black GPU. The inputs and outputs for all pixels are pre-computed and stored in an HDF5 [3] file format, which provides a matrix-like interface (PyTables [16]) to huge matrix objects kept on a hard drive. The data is normalized by subtracting feature mean and dividing by the standard deviation (both computed from a test set). With 3000 neurons, the solution of HP-ELM has numerical instabilities, thus a regularization parameter α is increased from a default value 1e-9 to 1e-2. For 500 neurons, the solution is numerically stable.

3.2.2 SOM

We implement the BD-SOM described in Section 2.3 using a combination of MATLAB and Nvidia's Compute Unified Device Architecture (CUDA). Our training code is written entirely in MATLAB, while the core of our classification method relies on MATLAB's Parallel Toolbox to interface CUDA code. Our validation and ensemble procedures require us to train maps of various sizes. Using the University of Iowa High Performance Computing Neon Cluster we are able to train these maps simultaneously so that the total time required to train the the maps–in both the validation and testing stages–is the time required to train the largest of the maps. For classification, we also make use of parallel computation, but in this case we pursue a massively parallel track: we use a GPU to classify pixels image-by-image. In our samples, this corresponds to batch classification of between 3×10^4 and 3×10^6 pixels. Our parallel scheme consists of a one-to-one mapping between the pixels in an image and GPU "blocks", and a one-to-one mapping between the neurons in a map and GPU "threads". Conceptually each block receives a copy of the entire map, so that each thread is associated with both a neuron and a pixel. Each thread computes the distance between the neuron it is identified with and the pixel identified with its block. In this way, blocks compute the BMU of their corresponding pixel, and return the associated prediction, simultaneously (see Figure 1). This strategy works best for small maps: for maps with less than 300 neurons classification requires less than 1 second per image.

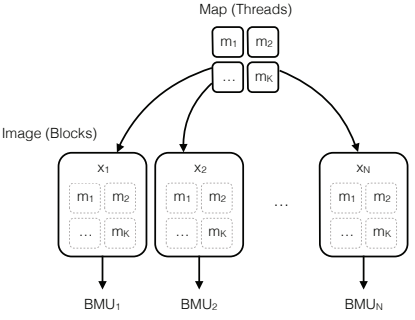

Figure 1: Conceptual depiction of the parallel scheme used to classify pixels with BD-SOM with K threads.

[3] https://hdfgroup.org/HDF5/

For large maps, however, the number of threads per block approaches the physical limit of the GPU device and classification times slow down considerably.

In addition to language and hardware decisions, training SOMs requires the choice of many parameters. As the primary goal of SOM is to cluster data by mapping it in to a lower dimensional space, faithfulness of the resulting map requires that the specified dimensions are large enough to capture the inherent complexity of the data. Small maps may not contain enough neurons to capture finer details of the data, while large maps become computationally demanding, and possibly result in over-fitting. We determine the optimal size of the SOM through a simple validation procedure described below. The dimensions of the map should also reflect the relative importance of the dimensions of the data captured. A dimension associated with more variability requires additional space in the map. We determine the dimensions of our map using a heuristic approach based on principal component analysis (PCA): the ratio of the map dimensions is set (approximately) equal to the ratio of the eigenvalues associated with the first and second principal components of the data.[4] For our data set this ratio is approximately $\lambda_1/\lambda_2 = 3$. SOM implementations generally specify learning rates $L_t \in [0, 1]$ and neighborhood radii $R_t \in \mathbb{R}^+$ that decay over time. In the early stages of training large values of L_t and R_t identify coarse features of data; later stages capture finer details as each iteration produces marginal changes to a single neuron. In our experiment we use an exponential decay function for both parameters:

$$L_t = L_0 \times \exp(-t/\lambda), \qquad R_t = R_0 \times \exp(-t/\lambda). \tag{10}$$

Values of L_0, R_0 and λ were chosen based on our own prior experiments and the results reported in [18], and all maps were trained for exactly 500 times the number of neurons in the map.

3.3 Results

3.3.1 Validation

We use a normal validation procedure to determine the optimal number of neurons to use for each method. We subdivide our training set of 2000 images into a set of 1300 images (the training set) and a set of 700 images (the validation set).

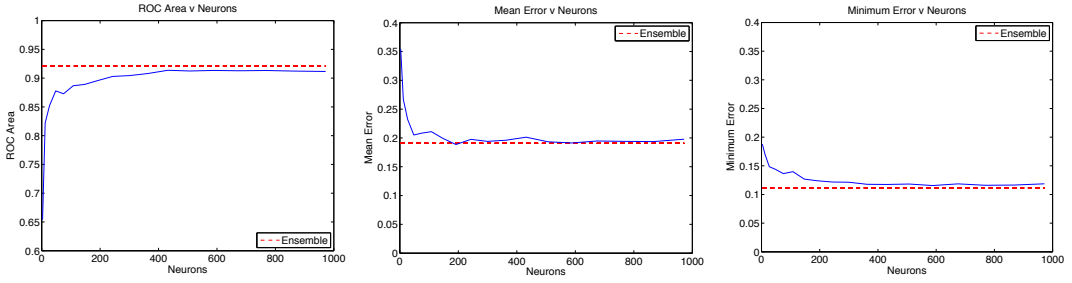

Figure 2: BD-SOM classification performance on validation set.

BD-SOM For each choice of number of neurons we train a network using the training set and evaluate the performance on the validation set. The BD-SOM classification requires a choice of threshold. We consider threshold values $\tau = 0, 0.05, 0.10, \ldots, 1$ and calculate measures of performance: ROC area (Δ), mean error (e_{mean}), and minimum error (e_{min}). ROC area is

[4] This is the same heuristic used in [17].

the area under the ROC curve spanned by our set of threshold values. Mean and minimum error are the mean and minimum errors across all threshold levels for a given map (the minimum error is attained at a level between 0.40 and 0.50 for all but the smallest map). The results of this procedure for the BD-SOM method are shown in Fig. 2. The maximum Δ was achieved by the 12×36 map. The minimum e_{mean} was achieved by the 8×24 map, and the minimum e_{min} is achieved by the 14×42 map. While there is no single map size that optimizes more than one of our criteria, an ensemble of the best maps does. In particular, we combine the predictions of the maps containing between 240 (8×24) and 588 (14×42) neurons by taking the average prediction of each of these maps *before* apply the threshold. Fig. 2 shows that this ensemble improves on both the maximum Δ and the minimum e_{min}, and is only narrowly worse than the optimal e_{mean}.

Table 2: Confusion matrices for HP-ELM predictions with nn=500.

ELM Prediction for Test Set		
	Skin	Non-Skin
Skin	0.9040	0.0960
Non-Skin	0.1570	0.8430

Table 3: Computation times of BD-SOM and HP-ELM, with GPU acceleration on Nvidia GK110-based cards.

	BD-SOM	HP-ELM
Training	5h 7m (972 nn)	3h 30m (3000 nn)
Validation	1h 19m	1h 44m
Retraining	1h 54m (588 nn)	43m (500 nn)
Testing	1h 40m	21m

HP-ELM The correlation matrices $\mathbf{\Omega}_h$ and $\mathbf{\Omega}_t$ are computed on a training set for the largest number of hidden neurons $nn = 3000$. Then the validation error (Fig. 3, *left*) is computed on a validation set by training an HP-ELM with sub-matrices $\mathbf{\Omega}_h^k[1..k, 1..k]$, $\mathbf{\Omega}_t^k[1..k, 1..c]$, $k \in \{1, nn\}$ with 30 distinct values of k equally spaced on a logarithmic scale. The minimum validation error corresponds to $nn = 500$, which is chosen as a target number of neurons for testing.

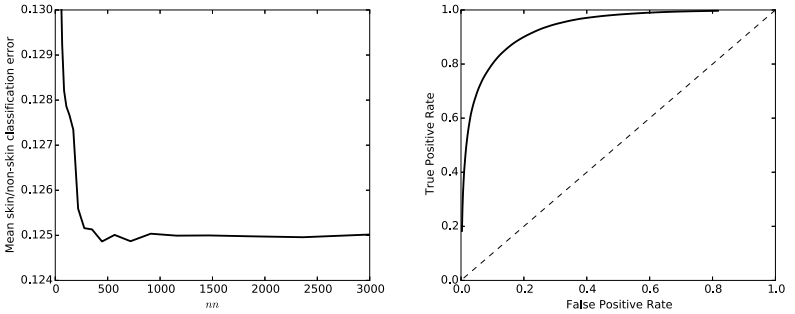

Figure 3: HP-ELM classification performance on validation set (*left*) and ROC curve for the test set with nn=500 (*right*).

3.3.2 Testing and Visualization

The results obtained with both methods are quite different. HP-ELM is approximately 3 times faster than BD-SOM. HP-ELM is also better in terms of skin detection accuracy (see Tables 2,4), but BD-SOM is better for non-skin detection. Therefore, globally HP-ELM is not as accurate as BD-SOM : 85% instead of 88% global accuracy. It is also important to notice that HP-ELM is much faster for the final classification (21 minutes instead of 1h 40 minutes, see Table 3), or roughly 5 times faster.

An interesting feature of SOMs is that they can be easily visualized. Our experiment provides an excellent example of this feature because the map neurons contain a component

Table 4: Confusion matrices for optimal maps using a threshold value of $\tau = 0.50$.

	SOM Prediction							
	Max. ROC Area		Min. Mean Error		Min. Min. Error		Ensemble	
	Skin	Non-Skin	Skin	Non-Skin	Skin	Non-Skin	Skin	Non-Skin
Skin	0.5473	0.4527	0.5541	0.4459	0.6660	0.3340	0.6271	0.3729
Non-Skin	0.0505	0.9495	0.0547	0.9453	0.0699	0.9301	0.0535	0.9465

representing averaged RGB values, as well as a skin variable that can be represented in gray-scale. This is demonstrated in Fig. 4, which shows the global accuracy maximizing map from our full testing. A comparison of the RGB and gray-scale representations shows that there is a cluster of roughly skin-colored neurons that coincides with neurons having the highest skin variable values.

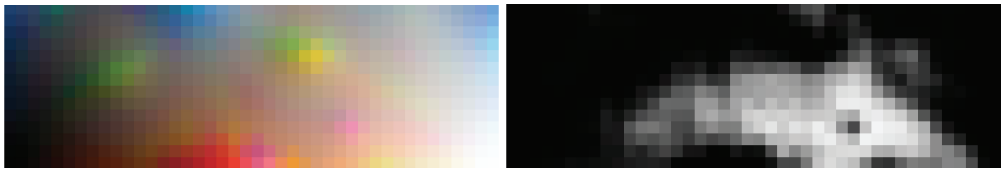

Figure 4: *RGB* and gray-scale representations of the 14×42 map shown after 294,000 iterations. Lighter shades of gray correspond to neurons that are more strongly associated with skin.

4 Conclusion

This paper presents two new techniques for skin detection. Both techniques are suitable for Big Data. Globally, BD-SOM is more accurate for skin detection, but is not as efficient in terms of computational time. Batch training of the SOM may reduce the computation time required for larger maps without requiring a reduction in the number of iterations. It may also be the case that the number of iterations in SOM training can be reduced without affecting global accuracy. In the future, we will investigate how to ensemble or merge BD-SOM and HP-ELM in order to obtain a more accurate classifier in an even smaller amount of time. Additionally, we plan to demonstrate the robustness of our results by applying these methods to other data sets.

References

[1] M.-H. Yang, et al., Gaussian mixture model for human skin color and its applications in image and video databases, in: Proceedings of SPIE 1999, 1999, pp. 458–466.

[2] H. Greenspan, et al., Mixture model for face-color modeling and segmentation, Pattern Rec. Lett. 22 (14) (2001) 1525 – 1536.

[3] S. L. Phung, et al., A universal and robust human skin color model using neural networks, in: Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on, Vol. 4, 2001, pp. 2844–2849.

[4] M. J. Jones, et al., Statistical color models with application to skin detection, Int. J. Comput. Vision 46 (1) (2002) 81–96.

[5] S. L. Phung, et al., Skin segmentation using color pixel classification: Analysis and comparison, PAMI 27 (1) (2005) 148–154.

[6] G.-B. Huang, et al., Extreme learning machine: Theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.

[7] Y. Miche, et al., OP-ELM: optimally pruned extreme learning machine, Neural Networks, IEEE Transactions 21 (1) (2010) 158–162.

[8] S. Haykin, Neural Networks: A Comprehensive Foundation (2nd Edition), 2nd Edition, Prentice Hall, 1998.

[9] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, USA, 1996.

[10] G.-B. Huang, et al., Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE transactions on neural networks 17 (4) (2006) 879–892.

[11] E. Cambria, et al., Extreme learning machines., IEEE Intelligent Systems 28 (6) (2013) 30–59.

[12] C. R. Rao, et al., Generalized Inverse of Matrices and Its Applications, John Wiley & Sons Inc, 1971.

[13] D. Brown, et al., A som based approach to skin detection with application in real time systems, in: Proceedings of the British Machine Vision Conference, 2001, pp. 491–500.

[14] M. Cottrell, et al., Missing values: Processing with the kohonen algorithm, in: Applied Stochastic Models and Data Analysis, 2005, pp. 489–496.

[15] S. Phung, et al., Skin segmentation using color pixel classification: analysis and comparison, Pattern Analysis and Machine Intelligence, IEEE Transactions on 27 (1) (2005) 148–154.

[16] F. Alted, et al., PyTables: Hierarchical datasets in Python, http://www.pytables.org/ (2002–).

[17] J. Vesanto, et al., Self-organizing map, in: Proceedings of the Matlab DSP Conference, Vol. 99, 1999, pp. 16–17.

[18] H. S. Tan, et al., Investigating learning parameters in a standard 2-d som model to select good maps and avoid poor ones, in: AI 2004: Advances in Artificial Intelligence, Vol. 3339 of LNCS, Springer Berlin Heidelberg, 2005, pp. 425–437.