TWO FUZZY LATTICE REASONING (FLR) CLASSIFIERS AND THEIR APPLICATION FOR HUMAN FACIAL EXPRESSION RECOGNITION

S. E. PAPADAKIS[1,2*], V. G. KABURLASOS[2†], G. A. PAPAKOSTAS[2‡]

[1] *Department of Business Planning and Information Systems, TEI of Crete*
P.O. Box 128, GR-72100, Agios Nikolaos, Greece
[2] *Human-Machines Interaction (HMI) Laboratory*
Department of Industrial Informatics, TEI of Kavala
GR-65404 Agios Loukas, Kavala, Greece

Received 30 November 2012; In final form 1 April 2013

We deal with the problem of human facial expression recognition from digital images. A digital image is preprocessed for feature extraction using moment descriptors; then, it is represented in the product lattice $(\mathbb{F}^{100}, \leq)$ of *Intervals' Numbers* (*IN*s). Learning as well as generalization are carried out in space $(\mathbb{F}^{100}, \leq)$ by two different *Fuzzy Lattice Reasoning* (*FLR*) classifiers based on an inclusion measure function $\sigma : \mathbb{F}^{100} \times \mathbb{F}^{100} \to [0, 1]$. We pursue both a stochastic optimization and a parallel implementation of the proposed techniques. Comparative experimental results on three benchmark data sets demonstrate a superior performance of the proposed *FLR* classification schemes.

Key words: Parallel processing; GPU Computing; Stochastic Optimization; Particle swarm; PSO; Fuzzy Lattice Reasoning; Intervals' Number; Inclusion measure; Facial expression recognition.

* email: spap@staff.teicrete.gr
† email: vgkabs@teikav.edu.gr
‡ email: gpapak@teikav.edu.gr

1 INTRODUCTION

Pattern recognition is an important application domain, which is, typically, preceded by data preprocessing for feature extraction and pattern representation. Dealing with uncertainty and/or ambiguity is of high interest in pattern recognition applications. This work deals with uncertainty based on Intervals' Numbers (*IN*s) as explained below.

Intervals' Numbers, or *IN* s for short, have been studied lately. More specifically, it has been shown that the set $\mathbb{F}$ of *IN*s is a metric lattice with cardinality $\aleph_1$, where "$\aleph_1$" is the cardinality of the set $\mathbb{R}$ of real numbers; moreover, the space $\mathbb{F}$ is a cone in a linear space. In all, an *IN* is a mathematical object, which may be interpreted as either a possibility distribution or a probability distribution thus accommodating uncertainty [17].

Our approach is to represent human facial expressions [1], [14], by *IN*s induced from image-moments-based preprocessing techniques [20, 23]. In conclusion, decision-making is carried out by Fuzzy Lattice Reasoning (*FLR*) techniques in the space $(\mathbb{F}^{100}, \leq)$ of *IN*s based on an inclusion measure function.

An *FLR* scheme can be regarded as a specific methodology of the Lattice Computing (*LC*) paradigm. We remark that the term *lattice computing* was originally defined as "the collection of Computational Intelligence tools and techniques that either make use of lattice operators *inf* and *sup* for the construction of the computational algorithms or exploit Lattice Theory for language representation and reasoning" [2]. Later work extended the meaning of *LC* to denote "an evolving collection of tools and methodologies that process lattice ordered data including logic values, numbers, sets, symbols, graphs, etc" [7, 26]. Current trends in *LC* appear in [3, 6].

This work is a extension of a preliminary work presented lately [18]. Substantial differences include: First, the work in [18] delineates an agglomerative *FLR* learning scheme only for structure identification such that one *IN* is induced (unconditionally) per class; whereas, this work details two different *FLR* classification schemes including also parameter optimization as well as a parallel classifier implementation. Second, the work in [18] assumes one 100-dimensional features (moments) vector represented by one (non-trivial) *IN*, furthermore it employs seven random data partitions for training/testing; whereas, this work assumes higher dimensional *IN*s vectors, furthermore it employs a 10-fold data partition for training/testing. Third, the work in [18] engages only two classifiers, namely *(agglomerative) FLR* and *kNN*; whereas, this work engages two different *FLR* classifiers as well as four more classifiers, namely *kNN*, *Naive Bayes*, *Classification Tree* and a *neural network*; furthermore, all classifiers here are applied, in addition, on the *RADBOUD* as well as the *PAIN* benchmark data sets.

This paper is organized as follows. Section 2 presents the mathematical background. Section 3 describes two *FLR* classification schemes. Section 4 describes stochastic (*PSO*) optimization techniques. Section 5 shows a parallel (*GPU*) implementation of the proposed techniques. Section 6 demonstrates, comparatively, human facial expression classification experiments. Finally, section 7 concludes by summarizing our contribution.

2 MATHEMATICAL BACKGROUND

This section summarizes notions and notation presented elsewhere [7, 8, 9].

2.1 General Lattice Theory Preliminaries

Consider the following definition.

Definition 1 *An* inclusion measure *in a lattice* $(\mathbb{L}, \leq)$ *is a function* $\sigma : \mathbb{L} \times \mathbb{L} \to [0,1]$*, which satisfies the following conditions:*

C1. $\sigma(x,x) = 1$.

C2. $x \wedge y < x \Rightarrow \sigma(x,y) < 1$.

C3. $u \leq w \Rightarrow \sigma(x,u) \leq \sigma(x,w)$.

Any employment of an inclusion measure for decision-making is called *Fuzzy Lattice Reasoning (FLR)* [8]. Next, we focus on a complete lattice with least and greatest elements O and I, respectively. Consider the following theorem [7].

Theorem 1 *Let function* $v : \mathbb{L} \to [0, +\infty)$ *be a positive valuation* * *on a complete lattice* $(\mathbb{L}, \leq)$ *with least and greatest elements O and I, respectively, such that both* $v(O) = 0$ *and* $v(I) < +\infty$. *Let functions* sigma-meet $\sigma_{\wedge} : \mathbb{L} \times \mathbb{L} \to [0,1]$ *and* sigma-join $\sigma_{\vee} : \mathbb{L} \times \mathbb{L} \to [0,1]$ *be defined as follows; first,* $\sigma_{\wedge}(O,y) = 1$, $\sigma_{\wedge}(x,y) = \frac{v(x \wedge y)}{v(x)}$ *for* $x > O$*; second,* $\sigma_{\vee}(x,y) = 1$ *for* $x \vee y = O$, $\sigma_{\vee}(x,y) = \frac{v(y)}{v(x \vee y)}$ *for* $x \vee y > O$. *Then, both functions* $\sigma_{\wedge}(.,.)$ *and* $\sigma_{\vee}(.,.)$ *are inclusion measures.*

* *Positive valuation* in a lattice $(\mathbb{L}, \leq)$ is a real function $v : \mathbb{L} \to \mathbb{R}$ that satisfies both $v(x) + v(y) = v(x \wedge y) + v(x \vee y)$ and $x < y \Rightarrow v(x) < v(y)$.

2.2 Intervals' Numbers (*INs*)

Take the set of real numbers $\mathbb{R}$ and the usual order $\leq$. Then $(\mathbb{R}, \leq)$ is a totally ordered, non-complete lattice. Lattice $(\mathbb{R}, \leq)$ can be extended to a complete lattice by including both "$-\infty$" and "$+\infty$". Any strictly increasing function $v : \overline{\mathbb{R}} \to [0, +\infty)$ is a positive valuation on $(\overline{\mathbb{R}}, \leq)$.

Consider the lattice $(\mathbb{I}, \leq)$ of *intervals*, including the empty set, ordered by set inclusion ($\subseteq$), where $\mathbb{I} \doteq \{[a,b] : -\infty \leq a \leq b \leq \infty\} \cup \{\emptyset\}$ and "$\emptyset$" is the empty set. The corresponding supremum operation ($\dot{\cup}$) is defined as $[a,b] \,\dot{\cup}\, [c,d] = [a \wedge c, b \vee d]$. Consider a *length function* definition next.

Definition 2 *A* length function $v_1 : \mathbb{I} \to [0, +\infty)$ *is defined as*

$$v_1([a,b]) = v(\theta(a)) + v(b), \tag{1}$$

where $v : \overline{\mathbb{R}} \to [0, +\infty)$ *is a strictly increasing function and* $\theta : \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ *is a strictly decreasing function.*

For example, consider the parametric functions

$$v(x; \lambda, \mu) = \frac{1}{1 + e^{\lambda \cdot (\mu - x)}} \text{ and } \theta(x; \mu) = 2\mu - x, \tag{2}$$

where $\lambda > 0$. Note that the sigmoid function $v(a; -\lambda, \mu)$ is strictly increasing, whereas its anti-symmetric function $v(b; \lambda, \mu)$ is strictly decreasing; the two sigmoid functions intersect at μ. By substituting (2) into (1) it follows the *length function*

$$v_1([a,b]) = \frac{1}{1 + e^{-\lambda(\mu - a)}} + \frac{1}{1 + e^{\lambda(\mu - b)}} = v(a; -\lambda, \mu) + v(b; \lambda, \mu) \tag{3}$$

Another lattice of interest is the lattice of *Intervals' Numbers*, or *IN*s for short, defined next.

Definition 3 *An Intervals' Number (IN) is a function* $F : [0,1] \to \mathbb{I}$ *which satisfies*

$$\begin{gathered} h_1 \geq h_2 \Rightarrow F_{h_1} \subseteq F_{h_2}, \\ \forall X \subseteq [0,1] : \bigcap_{h \in X} F_h = F_{\vee X}. \end{gathered}$$

We denote the class of all *IN*s by $\mathbb{F}$ and we equip it with an order ($\leq$) such that for every pair $F, G \in \mathbb{F}$ we define $F \leq G \Leftrightarrow (\forall h \in [0,1] : F(h) \subseteq G(h))$. It turns out that $(\mathbb{F}, \leq)$ is a complete lattice, namely *lattice of INs*, which (lattice) is isomorphic to the lattice of fuzzy intervals [7, 10].

2.3 Inclusion Measures

We present two inclusion measures on $(\mathbb{I}, \subseteq)$.

Proposition 1 *Let $v_1 : \mathbb{I} \to [0, +\infty)$ be a length function on $(\mathbb{I}, \subseteq)$ satisfying both $v_1(O) = 0$ and $v_1(I) < +\infty$. Then, functions $\sigma_\cap : \mathbb{I} \times \mathbb{I} \to [0,1]$ and $\sigma_{\dot\cup} : \mathbb{I} \times \mathbb{I} \to [0,1]$ defined, respectively, by*

$$\sigma_\cap([a,b],[c,d]) = \begin{cases} 1, & \text{if} \quad [a,b] = O \\ \dfrac{v_1([a,b] \cap c,d])}{v_1([a,b])}, & \text{if} \quad [a,b] \supset O \end{cases}, \text{ and} \tag{4}$$

$$\sigma_{\dot\cup}([a,b],[c,d]) = \begin{cases} 1, & \text{if} \quad [a,b] \dot\cup [c,d] = O \\ \dfrac{v_1([c,d])}{v_1([a,b] \dot\cup [c,d])}, & \text{if} \quad [a,b] \dot\cup [c,d] \supset O \end{cases}, \tag{5}$$

are inclusion measures on $(\mathbb{I}, \subseteq)$.

Next, we extend the aforementioned two inclusion measures to $(\mathbb{F}, \leq)$.

Proposition 2 *Let a length function $v_1 : \mathbb{I} \to [0, +\infty)$ satisfy both $v_1(O) = 0$ and $v_1(I) < +\infty$. Moreover, let the functions $\sigma_\cap$ and $\sigma_{\dot\cup}$ be as in Proposition 1. Then functions $\sigma_\curlywedge : \mathbb{F} \times \mathbb{F} \to [0,1]$ and $\sigma_\curlyvee : \mathbb{F} \times \mathbb{F} \to [0,1]$ defined, respectively, by $\sigma_\curlywedge(F,G) = \int\limits_0^1 \sigma_\cap(F_h, G_h)dh$ and $\sigma_\curlyvee(F,G) = \int\limits_0^1 \sigma_{\dot\cup}(F_h, G_h)dh$ are inclusion measures.*

By "convex combination" in the following we mean a linear combination with positive coefficients $\lambda_1, \ldots \lambda_N$, such that $\lambda_1 + \cdots + \lambda_N = 1$.

Theorem 2 *Let function $\sigma_i : \mathbb{L}_i \times \mathbb{L}_i \to [0,1]$ be an inclusion measure in lattice $(\mathbb{L}_i, \leq_i)$ for $i \in \{1, \ldots, N\}$. Let $(\mathbb{L}, \leq)$ be the Cartesian product lattice $(\mathbb{L}, \leq) = (\mathbb{L}_1 \times \cdots \times \mathbb{L}_N, \leq_1 \times \cdots \times \leq_N)$. Then, function $\sigma : \mathbb{L} \times \mathbb{L} \to [0,1]$ given by the* convex combination *$\sigma(\boldsymbol{x},\boldsymbol{y}) = \lambda_1 \sigma_1(x_1,y_1) + \cdots + \lambda_N \sigma_N(x_N,y_N)$, where $\boldsymbol{x} = (x_1, \ldots, x_N)$ and $\boldsymbol{y} = (y_1, \ldots, y_N)$, is an inclusion measure.*

Proof 1 *We will prove conditions **C1** - **C3** of Definition 1. Let $\boldsymbol{x},\boldsymbol{y},\boldsymbol{z} \in \mathbb{L}$, where $\boldsymbol{x} = (x_1, \ldots, x_N), \boldsymbol{y} = (y_1, \ldots, y_N)$ and $\boldsymbol{z} = (z_1, \ldots, z_N)$.*

C1: $\sigma(\boldsymbol{x},\boldsymbol{x}) = \lambda_1 \sigma_1(x_1,x_1) + \cdots + \lambda_N \sigma_N(x_N,x_N) = \lambda_1 + \cdots + \lambda_N = 1.$

C2: $\boldsymbol{x} \not\leq \boldsymbol{y} \Rightarrow (x_1, \ldots, x_N) \not\leq (y_1, \ldots, y_N) \Rightarrow \exists i \in \{1, \ldots, N\} : x_i \not\leq y_i \Rightarrow$ $\sigma_i(x_i,y_i) < 1$. *Therefore,* $\sigma(\boldsymbol{x},\boldsymbol{y}) = \lambda_1 \sigma_1(x_1,y_1) + \cdots + \lambda_i \sigma_i(x_i,y_i) + \cdots + \lambda_N \sigma_N(x_N,y_N) < \lambda_1 + \cdots + \lambda_i + \cdots + \lambda_N = 1.$

C3: $\boldsymbol{y} \leq \boldsymbol{z} \Rightarrow (y_1,\ldots,y_N) \leq (z_1,\ldots,z_N) \Rightarrow y_1 \leq z_1,\ldots,y_N \leq z_N \Rightarrow \sigma_1(x_1,y_1) \leq \sigma_1(x_1,z_1),\ldots,\sigma_N(x_N,y_N) \leq \sigma_N(x_N,z_N) \Rightarrow$
$\lambda_1\sigma_1(x_1,y_1)+\cdots+\lambda_N\sigma_N(x_N,y_N) \leq \lambda_1\sigma_1(x_1,z_1)+\cdots+\lambda_N\sigma_N(x_N,z_N)$
$\Rightarrow \sigma(\boldsymbol{x},\boldsymbol{y}) \leq \sigma(\boldsymbol{x},\boldsymbol{z})$.

In view of Theorem 2, the extension of an inclusion measure from one to many dimensions is straightforward. More specifically, consider a "D-dimensional" *IN* $\mathbf{F} = (F_1,\ldots,F_D)$ including D "1-dimensional" *IN*s $F_1,\ldots,F_D$. Based on Theorem 2, an inclusion measure $\sigma_{\cup}(\mathbf{F},\mathbf{G}) : \mathbb{F}^D \times \mathbb{F}^D \to [0,1]$ is given by

$$\sigma_{\cup}(\mathbf{F},\mathbf{G}) = \frac{1}{D}\sum_{i=1}^{D}\sigma_{\cup}(F_i,G_i) \tag{6}$$

3 TWO FUZZY LATTICE REASONING (FLR) CLASSIFIERS

We employ the inclusion measure, given above, to the challenging problem of human facial expressions recognition [1], [14].

Given a vector of features, induced from an image, an *IN* meta representation is computed. In the context of this work, two different meta representations, namely "A" and "B", were employed exclusively by two *FLR* classifiers, namely *FLR*-A and *FLR*-B, respectively. In both cases the input to an *FLR* classifier is a set of M gray scale images, representing various facial expressions. A row vector $\mathbf{f_m} \in \mathbb{R}^D$, namely *feature vector,* is extracted by preprocessing image I_m, $m = 1,2,...,M$. Moreover I_m express a particular facial expression, labeled L_m. Image I_m can be represented as $I_m = [\mathbf{f}_m, L_m]$, and the set of M images as $\mathbf{I}_M \in \mathbb{R}^{M\times(D+1)}$. In this work a feature vector includes moments of one of six different families, namely, *Zernike*, *Pseudo-Zernike*, *Fourier-Mellin*, *Legendre*, *Tchebichef*, and *Krawtchouk*. For each image, a feature vector regarding a specific moment family was extracted.

3.1 The *FLR*-A Classifier

Let (labeled) data in $\mathbb{F}_1^N$ be partitioned, first, in n_c classes and, second, in 2 disjoint data sets for *training* and *testing*, respectively. Classifier *FLR*-A learning is carried out by Algorithm 1 (for structure identification); whereas, classifier *FLR*-A generalization is carried out by Algorithm 2. We remark that the function $\ell : \mathbb{F}^N \to \mathbb{L}$ assigns a class label to an *IN*. Assuming that class c includes M_c images, for each specific feature $f_{m,d}$, $m = 1,2,...,M_c$ an interval number $F_m \in \mathbb{F}$ is build from $\{x_{1,d}, x_{2,d}, ..., x_{M_c,d}\}$ values over all M_c images. Note that the set of images for learning is partitioned into n_c groups each including images of the same class. Next, for each class and each feature $d,\ d = 1, ..., 100$, an *IN* is computed. Finally, a class is represented as a $D = 100$ dimensional *IN* and its label.

3.2 The *FLR*-B Classifier

Each feature $x_{m,d} \in \mathbb{R}$ is represented by a “trivial” (singleton) *IN*, $F_{m,d} \in \mathbb{F}$, thus creating an 100 dimensional vector $\mathbf{F}_m$ of trivial *IN*s per image. Image I_m is represented by the row vector $[\mathbf{F}_m, L_m] = [F_{m,1}, ..., F_{m,d}, ..., F_{m,D}, L_m]$,$F_{m,d} \in \mathbb{F}$. A set of N images, can be represented as $\mathbf{I}_N = [\mathbf{F}_j, L_j]$, where the $j-th$ row represents the $j-th$ image as a row vector of D columns of *IN*s. The last $D+1$ column is the label of the image. According to that representation an *FLR* model is created by Algorithm 3. The output of which, is a set $\mathbf{R}$ with cardinality N_r. Each element of $\mathbf{R}$ is a class of images encoded as a D dimensional *IN* and a label which denotes a particular facial expression. The elements of $\mathbf{R}$ are named “classes”, “rules”, “granules” , “hyper-boxes” etc. The input to Algorithm 3, is a subset $\mathbf{I}_{N_\ell}$ of $N_\ell < N$ images considered for learning. Algorithm 2 describes the generalization phase of classifier *FLR*-B.

4 PARTICLE SWARM OPTIMIZATION

The structure of both classifiers *FLR*-A and *FLR*-B depends on the values of parameters μ, λ, T_s. Optimal values, in terms of generalization performance (pursued here by cross-validation) is necessary, to ensure a high quality classifier. The optimization task is carried out using a Particle Swarm Optimization, or *PSO* for short, which is a stochastic, derivative free, non-linear optimization technique [11]. A *PSO* includes a population of “particles”, each consisting of a position vector $\mathbf{p} \in \mathbb{R}^q$ that encodes the parameters of the objective function $Q(\mathbf{p}) :\ \mathbb{R}^q \rightarrow \mathbb{R}$ being optimized. At the first time of the evolution, i.e $t_0 = 0$, the position of each particle is randomly initialized, while its quality is computed as the value of the objective function for the given position. Next, particle’s are moved to a new promising direction, by updating their positions using both local (particle’s best) and global (population best) information. The adaptation low is one of the main criteria that differentiates a *PSO* implementation from another one. The basic adaptation low is given by Equation (7):

$$\begin{array}{rclcl}
\mathbf{V}_{\mathbf{p},t+1} & = & \alpha(t) \cdot \mathbf{V}_{\mathbf{p},t} & + & c_1 \cdot u(0,1) \cdot (\mathbf{p}_{best,t} - \mathbf{p}_t) \\
 & & & + & c_2 \cdot u(0,1) \cdot (\mathbf{g}_{best,t} - \mathbf{p}_t) \\
\mathbf{p}_{t+1} & = & \mathbf{p}_t & + & \beta(t) \cdot \mathbf{V}_{p,t+1}
\end{array} \tag{7}$$

$$\underbrace{\boxed{\lambda_{p,0}}\boxed{\mu_{p,0}}}_{input-0}\Vert\boxed{\cdots}\Vert\underbrace{\boxed{\lambda_{p,D-1}}\boxed{\mu_{p,D-1}}}_{input-(D-1)}\boxed{T_{p,s}} \longrightarrow \quad FLR_p \rightarrow \quad Q_p \in \mathbb{R}$$

FIGURE 1
FLR parameter encoding. The position vector of particle $p = 1, 2, ..., P$ encodes parameters λ_i, μ_i, $i = 0, 1, ..., D-1$ for each dimension and a threshold size T_s. Decoding parameter values of particle p, an FLR_p model can be built by Algorithm 3. The calculation of quality Q_p of FLR_p is accelerated by *GPU*.

where real numbers $\alpha(t), \beta(t)$, namely *inertia* and *restriction parameter*, respectively, may be either constant or time decreasing; $c_1, c_2 \in (0, 1]$ are constant; P_t and $p_{best,t}$ are the current and the best attained position of particle $\mathbf{p}$; $\mathbf{g}_{best,t}$ is the best position achieved by the population until time t; $u(0, 1)$ is a random number generated uniformly in the interval $(0, 1)$. At the end of evolution, i.e. at time step t_e, the $\mathbf{g}_{best,t_e}$ encodes the final solution, succeeded.

The position vector and the quality function are dealt with as follows. Since an *FLR* scheme includes two parameters (μ, λ) per input, and one parameter T_s, representing the maximum allowed hyper-box size, the *FLR* parameters can be encoded as particle's position vector as illustrated in Figure 1. The swarm evolution process is summarized in Algorithm 4.

Given a subset $\mathbf{I}_{N_\ell}$ of $N_\ell < N$ images considered for learning, the creation of *FLR* structure for a specific particle is given by Algorithm 3. For cross-validation, the learning set is subdivided into two disjoint subsets $\mathbf{I}_{N_T}$ and $\mathbf{I}_{N_v}$, namely training and validation set, respectively. The *FLR* structure is created on the images of training set. Then the success classification rates $Q_{p,T}$, $Q_{p,V}$ on the training set and the validation set, respectively, are calculated. The quality of particle is computed as the weighted sum: $Q_p = w \cdot Q_{p,T} + (1-w) \cdot Q_{p,V}$. The predefined parameter $w \in (0, 1)$ is a *relaxation factor* that balances the fitting and generalization performance of *FLR*. It has to be stressed that without cross validation, the generalization performance of *FLR* is remarkably poor.

Both *FLR* training and generalization process is based on the calculation of inclusion measure between multi-dimensional *IN*s. Computationally, a $D-$dimensional *IN*, $\mathbf{F}$ could be stored in computer memory as a three dimensional matrix $\mathtt{F}[D \times H \times 2]$, where a specific interval $[a, b]_{i,h}$ in the input i at level h, is stored as $[\mathrm{a}, \mathrm{b}]_{\mathrm{i,h}} = [\mathtt{F}_{\mathtt{i,h,0}}, \mathtt{F}_{\mathtt{i,h,1}}]$, $i \in [0, D)$, $, h \in [0, H)$, $k = 0, 1$. Then the inclusion measure between two D - dimensional *IN*s $\mathbf{F}, \mathbf{G}$ is computed by:

$$\sigma_{\cup}^{D}(\mathbf{F},\mathbf{G}) = \frac{1}{D \cdot H} \sum_{i=0}^{D-1} \sum_{h=0}^{H-1} K_{i,h}(\mathtt{F},\mathtt{G}), \tag{8}$$

where

$$K_{i,h}(\mathtt{F},\mathtt{G}) = \frac{v(\mathtt{G}_{i,h,0}; -\lambda_i, \mu_i) + v(\mathtt{G}_{i,h,1}; \lambda_i, \mu_i)}{v(min(\mathtt{F}_{i,h,0}, \mathtt{G}_{i,h,0}); -\lambda_i, \mu_i) + v(max(\mathtt{F}_{i,h,1}, \mathtt{G}_{i,h,1}); \lambda_i, \mu_i)} \tag{9}$$

In Eq.(9) $v(x;\lambda,\mu)$ is the length function given by Equation (3) where two, per dimension, adjustable parameters $\lambda_i, \mu_i \in (0,1]$ $i \in [0,D)$, are assigned. Moreover, the operators join ($\vee$) and meet ($\wedge$) are implemented as $x \vee y = max(x,y) \in \mathbb{R}$ and $x \wedge y = min(x,y) \in \mathbb{R}$ for any $x,y \in \mathbb{R}$, respectively.

The computation of $K_{i,h}$ terms are independent from one another, since the values involved in calculations are stored in different memory locations. Even for parameters λ_i, μ_i, which are jointly used in calculations for a specific input, no data dependencies/anti-dependencies exist since they are accessed only for reading. Typically, read access conflicts are transparently resolved by hardware, although some performance issues might be raised. However, the parallel computation of inclusion measure in Equation (9) is straightforward. We accelerated the calculation by Graphics Processing Units (*GPU*s) as detailed in the following section.

5 PARALLEL COMPUTING IMPLEMENTATION

A graphics card includes one or more Graphics Processing Units (*GPU*s). Each *GPU* is a tightly connected parallel machine, that consists of a number of physical multicore processors and has its own memory, accessible by the cores (the basic physical processing units). For example, *Geforce GTX 260*, includes one *GPU* of 24 eight-core processors, yielding $24 \times 8 = 192$ physical processing units with 1 GB of own memory, whereas *Geforce GTX 660 Ti* has 2GB of memory and includes 7 multi processors of 192 cores each, yielding $7 \times 192 = 1344$ computing cores.

The cores are transparent to the *end-user* (i.e. the programmer). That is, the end-user can not directly access them. Instead, he is able, through suitable Programming Interface (API), to dynamically create and handle a practically large number ($\approx 10^{11}$) of *logical processing units*, named *threads*. All threads concurrently execute the same piece of code written in a specific programming function, named *kernel function*, following the Single Instruction Multiple Thread (*S.I.M.T.*) parallel execution model [4, 16, 24]. The embedded *GPU* software transparently assigns threads to cores, carrying out the execution. The end-user can virtually layout the threads into up to three-dimensional blocks. The blocks, in turn, can also be organized into up to a three-dimensional grid. *GPU*, at runtime, assigns a unique triplet of integers, to each group identifying it in the grid, as well as a unique triplet of integers to each thread, identifying it in its group. Therefore, every thread has both a local (within group) and a global (within grid) identity that separates it from the other threads, permitting the kernel function, properly designed, to access/process different data, thus implementing the *S.I.M.T.* model.

Another concept of interest is the *synchronization barriers*. A synchronization barrier is an execution point of "appointment" among threads. Given a pool of threads, associated with a synchronization barrier, the *early arrived* to the barrier threads are blocked until all threads reach that point of execution. Then, all threads are simultaneously dismissed, continuing their execution. Two kinds of synchronization are supported: Internal synchronization, invoked by *GPU* to the threads of a block and external synchronization, invoked by host process, to all threads. Furthermore, a synchronization barrier is a tool which allows the resolution of memory write access conflicts, when several threads request to access same memory location, simultaneously.

Although there are several technical issues that significantly affect the performance of *GPU*, we will not refer to, since they are out of the scope of this paper. However, it has to be mentioned that host process, which leads the execution, has not direct access to *GPU* memory when the threads execute. Also *GPU* threads have no direct access to host memory. As a consequent the data being processed by threads must be prior transferred to *GPU* memory. Respectively, the results produced by *GPU* threads must be transferred back to host memory to be accessible to the end user. Also, the resolution of potential memory access conflicts is programmer's responsibility. The activity diagram of a "typical *GPU* work-flow" is illustrated in Figure 2.

5.1 Thread layout

To implement the *GPU* calculation of inclusion measure in the form of equation (9), the layout of *GPU* threads must be decided. It should be noted that no unique layout for a given problem exist. The layout is usually decided such that (a) to satisfy technical constrains imposed by the hardware, (b) to meet performance issues, and (c) to facilitate the programming effort. For the problem in hand $H \cdot D = 32 \times 100 = 3200$ independent instances must

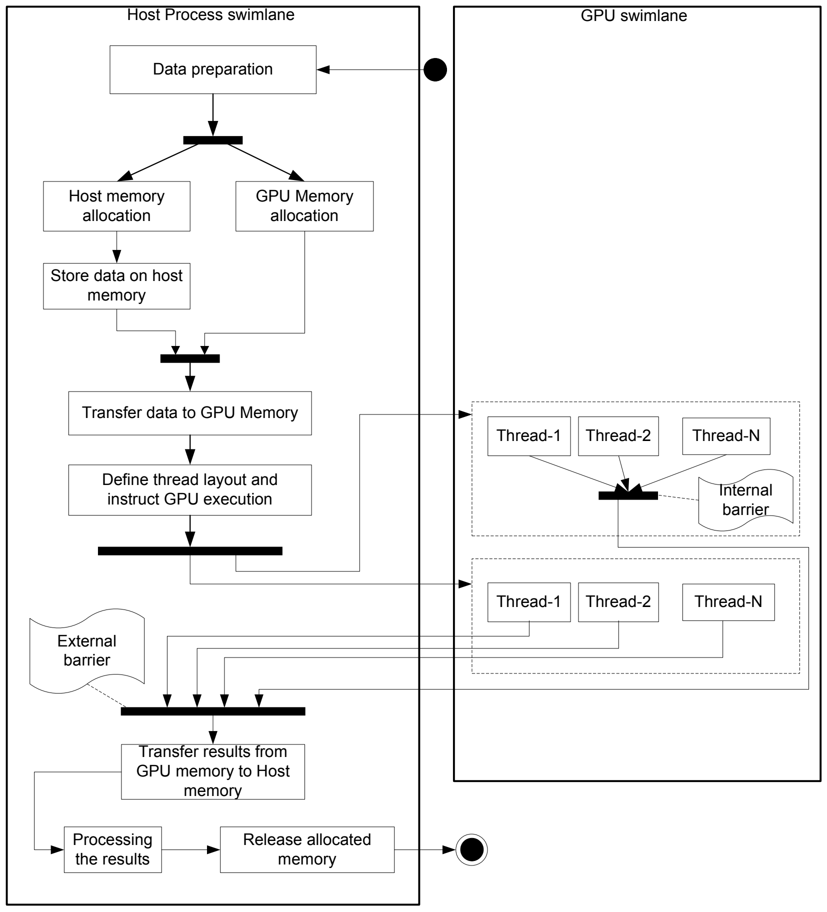

FIGURE 2
A typical work-flow of *GPU* execution

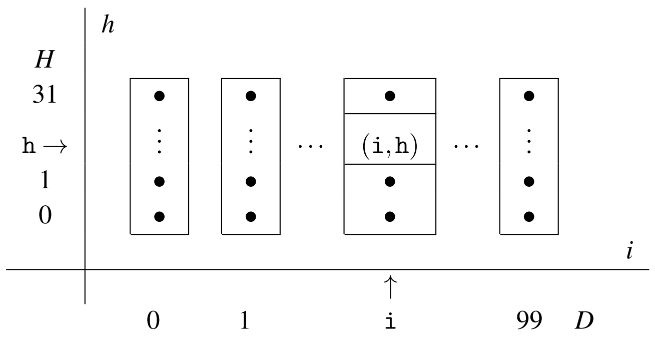

FIGURE 3
Thread layout consisting of 100 one dimensional blocks of 32 threads. The space of instances is tiled with 100 tiles of size 1×32. Each tile is assigned to a block.

be concurrently calculated. The graphics card we use, supports up to 1024 threads per block. As a result, a layout of one block including 3200 threads, for example, is not feasible. A common performance issue relates to the *warp size,* which is hardware depended and relates to the number of threads a physical processor concurrently executes. It is a preferable practice to design the layout such that the number of threads per block be a multiple of warp size, since all memory access are coalesced into multiples of the warp size. Since the warp size of the card we used is 32 and considering that the number of intervals equals $H = 32$, we selected blocks of 32 threads, keeping in mind that each thread of a block computes a specific level $h = [0, H)$. Regarding the number of blocks, we selected one block per dimension. Thus, the selected thread layout is 100 one dimensional blocks of 32 threads. This layout is programatically convenient since, at run time, the global identity of a block directly corresponds to dimension index i, while the local identity of a thread within its block corresponds to index h. The selected layout is illustrated in Figure 3. Note that an alternative thread layout could be four blocks of 32×25 threads instead of 100 blocks of 1×32 threads. The last layout produced slightly better results possibly due to better processor occupancy.

5.2 Kernel function

Having determined an appropriate thread layout, the “kernel function” (the piece of code the threads concurrently executes) has to be designed. We designed the kernel function assuming the *S.I.M.T.* execution model. On Instruction of the execution, all the threads of the layout concurrently executed the same code. The attached unique identity (both local and global) stated thread's ability to process different data and/or follow different execution paths. A potential design of kernel function, given that the required data are stored in *GPU* memory, is given by Algorithm 5. Following the work flow in Figure 2, the data are transferred to *GPU* memory and then the host process invokes the execution of threads according to Figure 3. All threads concurrently execute the code of the function, according to the *S.I.M.T.* execution model, producing partial results. Next, reduction of partial results, that is the calculation of final sum from the partial terms, was applied as in [5, 22] to produce the final sum. Reduction was applied in block level since internal synchronization barriers are by design effective to the threads of the same blocks. The final sum was calculated by the host process after external synchronization, by summing up the internally reduced sums of blocks.

6 COMPUTATIONAL EXPERIMENTS

We considered the problem of human facial expression recognition using three benchmark data sets: *JAFFE* [13], *RADBOUD* [12] and *PAIN*. For example, the *JAFFE* benchmark data set (Figure 4) includes images of facial expressions such Angry (30), Disgust (29), Fear (32), Happy (31), Neutral (30), Sadness (31), Surprise (30), where a number within parentheses, indicates the number of available images per facial expression.

6.1 Data Preprocessing and Feature Extraction

Irrelevant facial expression content was removed using the Viola-Jones face detector [25] followed by an ellipse-based face masking. Next, feature extraction was carried out using orthogonal moments. In particular, from each image, we induced 100 moments regarding *Zernike*, *Pseudo-Zernike*, *Fourier-Mellin*, *Legendre*, *Tchebichef* and *Krawtchouk* moments [15, 21], respectively. However, we based our experimentation to *Zernike* family only, since we obtained slightly better results compared to other ones.

6.2 Computational Experiments

We partitioned, the data in every class in two mutually disjoint sets: One set for learning (including around 90% of the data) and another one for testing (including the remaining 10% data). Then we clockwise rotated the data, making ten shuffles according to the 10-fold-out cross validation method. For

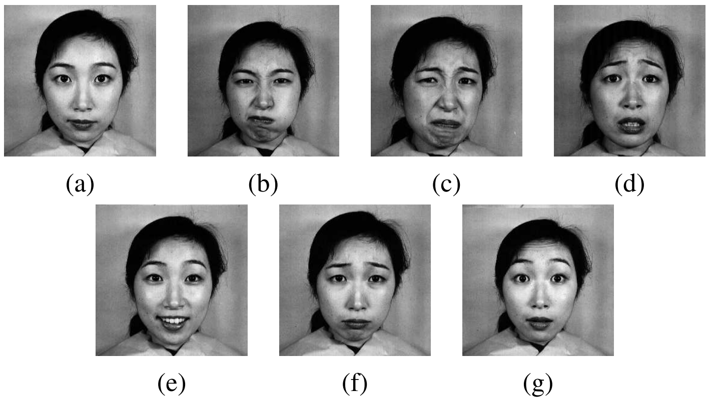
FIGURE 4
Seven different facial expressions from the *JAFFE* benchmark including (a) “neutral”, (b) “angry”, (c) “disgust”, (d) “fear”, (e) “happy”, (f) “sadness”, (g) “surprise”.

each shuffle the learning set itself was further split into two disjoint subsets, namely training subset and validation subset. Learning set was used to built the *FLR*-A classifiers while “unseen” testing set was used after learning process to evaluate the generalization performance of the classifier. The same learning set was also used to train/built the alternative classifiers for a fair comparison. The comparison placed by testing success classification rate. All the alternative classifiers were built by feature representation of the images, whereas both the *FLR*-A and *FLR*-B classifiers used the *IN*s meta representation as described above. Moreover, some empirical experimentation on the structural parameters of alternative classifiers was carried out, towards acquired relatively admissible results. The recorded success rates on three data sets, namely *JAFEE*, *RADBOUD* and *PAIN*, are reported in Table 6.2, while an illustrative rule base part for the *zernike* moment family of *JAFFEE* data set is given in Figure 5. We point out that the complete rule base cannot be included, for lack of space, since we have 100 inputs for seven classes. From Table 6.2 it is clear that the *FLR*-B, compared to six alternative classifiers, achieves the best testing classification rates in all the three benchmark classification problems.

6.3 Some Implementation Details

The speed up $s = \frac{t_s}{t_p}$, where t_s is the time required for the serial execution on *CPU* (single core) and t_p the time required for parallel implementation,

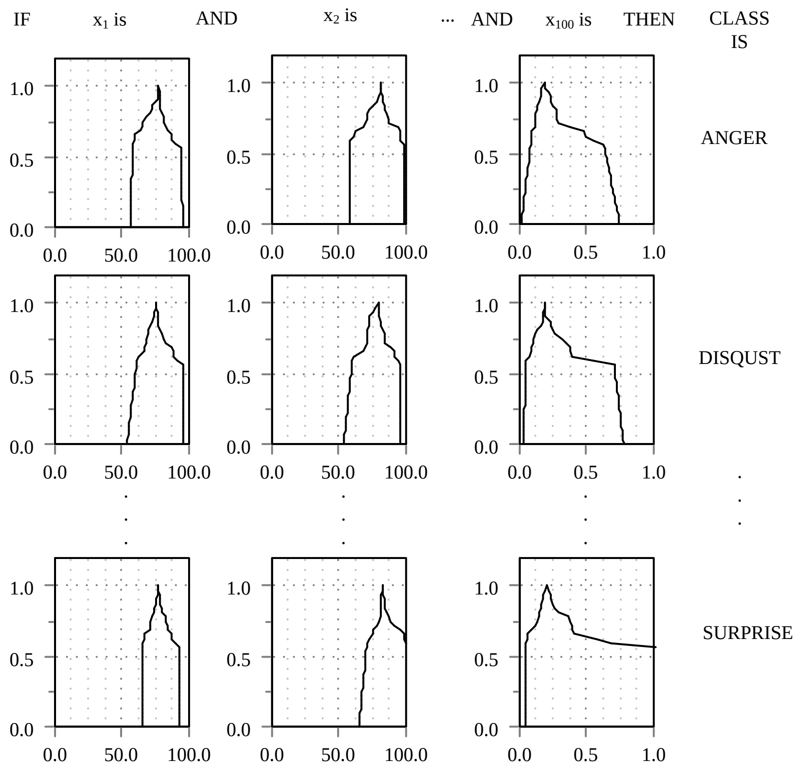

FIGURE 5
An illustrative rule base example of the FLR-A model for the JAFEE benchmark and Zernike moment family. Each rule represents a particular facial expression in the form $\mathbf{R}_c$: **IF** x_1 *is* $F_{c,1}$ *and* x_2 *is* $F_{c,2}$ *and* $\cdots$ *and* $x_{c,100}$ *is* $F_{c,100}$ **THEN** CLASS (facial expression) is L_c. x_i denotes a specific feature (moment), $F_{c,i}$ denotes the Interval number of rule c, $c = 1,...,7$ in dimension i and L_c denotes the label of rule c. For the specific example $i = 1,...,100$ and $L_c \in \{$ *ANGER,DISQUST,FEAR,HAPPINESS,NEUTRAL,SADNESS,SURPRISE* $\}$

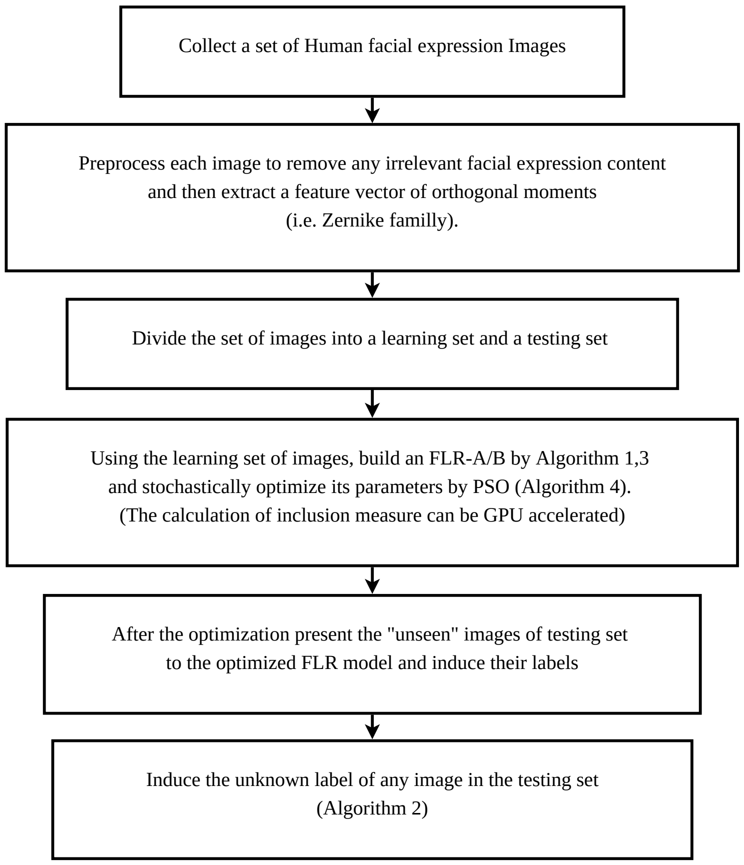
Collect a set of Human facial expression Images
Preprocess each image to remove any irrelevant facial expression content and then extract a feature vector of orthogonal moments (i.e. Zernike familly).
Divide the set of images into a learning set and a testing set
Using the learning set of images, build an FLR-A/B by Algorithm 1,3 and stochastically optimize its parameters by PSO (Algorithm 4). (The calculation of inclusion measure can be GPU accelerated)
After the optimization present the "unseen" images of testing set to the optimized FLR model and induce their labels
Induce the unknown label of any image in the testing set (Algorithm 2)

FIGURE 6
The workflow of the proposed approach for facial expression classifier

TABLE 1
Classification rate on *JAFEE*, *RADBOUD* and *PAIN* face recognition Benchmarks using the *Zernike* family of moments and two meta representations by *IN*s.

JAFEE Benchmark				
Classifier	min	max	ave	std
FLR-A	74.07	83.33	79.62	3.20
***FLR*-B**	**77.27**	**95.45**	**85.92**	**6.52**
kNN (k=1)	68.18	86.36	76.45	7.12
Naive Bayes	22.73	54.54	34.39	12.12
Classification Tree	27.27	54.55	41.57	9.54
Neural Network	13.63	63.63	30.12	14.54
RADBOUD Benchmark				
Classifier	min	max	ave	std
FLR-A	37.03	51.85	43.12	5.70
***FLR*-B**	**51.85**	**59.25**	**56.16**	**3.44**
kNN (k=1)	37.03	51.85	42.12	5.40
Naive Bayes	37.03	59.25	45.37	7.72
Classification Tree	24.07	38.88	29.54	5.84
Neural Network	18.51	51.85	29.26	11.41
PAIN Benchmark				
Classifier	min	max	ave	std
FLR-A	11.11	33.33	24.68	7.40
***FLR*-B**	**22.22**	**77.77**	**54.42**	**21.84**
kNN (k=1)	0.00	44.44	16.67	12.00
Naive Bayes	22.22	44.44	28.89	7.77
Classification Tree	11.11	44.44	31.11	11.48
Neural Network	11.11	55.56	24.44	13.66

is calculated with respect to the number of Dimensions (inputs) *D*, for the *RADBOUD* benchmark. In order to evaluate the performance of the parallel implementation, we artificially altered the number of inputs from ten to one thousand either by removing inputs (if less than 100) or by adding inputs (if more than 100). The results are graphically presented in Figure 7. In Figure 7(a) we monitor both *GPU* and *CPU* time in the number of dimensions. As it is observed, the serial (*CPU*) execution time linearly increases in the number of dimensions, while the *GPU* time is constant (about 0.31 sec). The constant *GPU* time is attributed as it is *startup time*, that is the time required for data transfers, threads creation etc while the time spent for the execution is practically negligible. The speedup of paralellization, is depicted in Figure 7(b). It is also linear in the number of dimensions. That is even for one thousand dimensions the performance gain in the "size" of problem increases constantly. Moreover, Figure 7(a) shows that when the number of inputs is around 50 then the speedup is $s \approx 1$. That is, $t_s \approx t_p$. For number of inputs less that 50 the serial time was less than the parallel one and hence there is not speedup gain. For around 100 inputs, which was our real case, the speedup was $s = 2.74$.

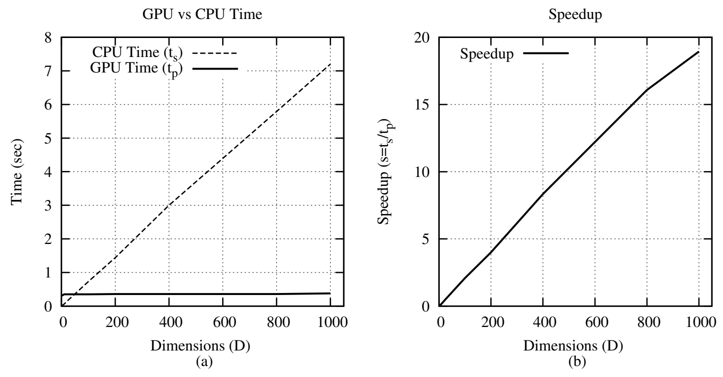

FIGURE 7
(a) The *CPU* (dashed line) and *GPU* (solid line) respectively, in the number of dimensions exetution Time (solid line) in the number of dimensions. (b) The speedup as a measure of the parallelization performance. The diagrams show that for "small size" problems no gain is monitored by *GPU*, while for "large scale" ones the speedup is substantial.

6.4 Discussion

The classification results presented in Table 6.2 demonstrate a superior average performance for the *FLR*-B classifier compared with alternative classifiers. Given the standard deviation values in Table 6.2, we can claim that the better average performance of the *FLR*-B classifier is statistically significant. In other words, the *FLR*-B classifier consistently results in a better classification performance in all the three benchmark human facial expression recognition problems considered above. The *workflow* of the proposed facial expression classifier is illustrated in Figure 6.

7 CONCLUSION

In this study We have employed the emerging framework of Fuzzy Lattice Reasoning *FLR*, to build classifiers able to process information granules represented as interval numbers in the form of rules instead of simple points defined in R^N. *FLR* classifiers are built by using a novel inclusion measure defined in the set of lattice ordered interval numbers, thus providing the advantage of unified data representation of disparate data types. Critical points of our study are summarized in the following

- The optimization of *FLR* parameters is carried out by stochastic particle swarm (*PSO*) optimization technique.
- A parallel computation of the inclusion measure in graphics cards (*GPU*) is given, showing that solving classification problems of hight computational complexity with *FLR* can be significantly accelerated.
- The proposed classification *FLR* schemes *FLR*-A and *FLR*-B, were successfully applied on three benchmark human facial expression recognition problems demonstrating a superior generalization performance compared to four other established classifiers.

Future work includes a full scale parallel implementation of an *FLR* classifier including *GPU* implementation of *PSO*, as in [19].

8 ACKNOWLEDGEMENT

This research has been co-funded by the European Union (Social Fund) and Greek national resources under the framework of the “Archimedes III: Funding of Research Groups in TEI of Athens” project of the “Education & Lifelong Learning” Operational Programme.

APPENDIX

Algorithm 1 : Classifier *FLR-A* learning (for structure identification)

```
Let $n_c$ be the total number of classes.
for $j = 1$ to $j = n_c$ do
Induce labeled INs $\overrightarrow{\mathbb{W}}_j \in \mathbb{F}_1^N$ from the training data in class $c_j$.
```

Algorithm 2 : Classifier *FLR-A/B* generalization

```
Assume a set $\bigcup_{j \in \{1,...L\}} \{c_j\}$ of labelled classes $c_j = \{\overrightarrow{\mathbb{W}}_j\}$.
for $i = 1$ to $i = n_{tst}$ do
Consider the next testing datum $(\overrightarrow{\mathbb{S}}_i, \ell(\overrightarrow{\mathbb{S}}_i)) \in \mathbb{F}_1^N \times \mathbb{L}$.
end for
Let $J = \underset{j \in \{1,...,L\}}{argmax} \{\sigma_{\cup}(\overrightarrow{\mathbb{S}}_i, \overrightarrow{\mathbb{W}}_j)\}$.
If $J = \ell(\overrightarrow{\mathbb{S}}_i)$ add on to the number of correct classifications.
```

Algorithm 3 : *FLR-B* learning algorithm for a given parameter set μ, λ, T_s and a learning set $\mathbf{I}_\ell = \mathbf{I}_{N_T} \cup \mathbf{I}_{N_V}$ of images regarded as the union of two disjoint subsets $\mathbf{I}_{N_T}$ (training set) and $\mathbf{I}_{N_V}$ (validation set).

```
:BEGIN
  1. For given $\mu, \lambda, T_s$ values: set $j \in [1, N_T)$ with $j = 0$ and the cardinality $N_r$ of $\mathbf{R}$ with $N_r = 0$.
  2. Insert datum $[\mathbf{F}_j, L_j] \in \mathbf{I}_{N_T}$ into set $\mathbf{R}$ and set $j = j+1, N_r = N_r + 1$.
  3. IF $j = N_T$ THEN go to step 8.
  4. Compute $r_{max} = \underset{L_r = L_j}{\text{argmax}}\{\sigma_{r,j}(\mathbf{H}_r, \mathbf{F}_j)\}$, where $\mathbf{H}_r \in \mathbf{R}$, $r = 1, 2, ..., N_r$.
  5. Calculate the size $S(\mathbf{H}_r^*)$, where $\mathbf{H}_r^* = \mathbf{H}_r \cup \mathbf{F}_j$.
  6. IF S($\mathbf{H}_r^*$) $\leq T_s$ THEN replace $\mathbf{H}_r$ with $\mathbf{H}_r^*$, set $j = j+1$ and go to step 3.
  7. ELSE IF S($\mathbf{H}_r^*$) $> T_s$ THEN insert $\mathbf{F}_j$ into $\mathbf{R}$, set $j = j+1$, $N_r = N_r + 1$ and go to step 3.
  8. set $j = 0$, $Q_T = 0$.
  9. IF $j = N_T$ THEN go to step 13.
 10. Calculate $r_{max} = \{\sigma_{r,j}(\mathbf{H}_r, \mathbf{F}_j)\}.\ r = 0, ..., N_r$.
 11. IF $L_{r_{max}} = L_j$ THEN $Q_T = Q_T + 1,\ j = j+1$, go to step 9.
 12. ELSE IF $L_{r_{max}} \neq L_j$ THEN set $j = j+1$ and go to step 9.
 13. set $j = 0,\ Q_V = 0$
 14. IF $j = N_v$ THEN go to step 18.
 15. Calculate $r_{max} = \text{argmax}\{\sigma_{r,j}(\mathbf{H}_r, \mathbf{F}_j)\}.\ r = 0, ..., N_r$, $\mathbf{F}_j \in \mathbf{I}_{N_v}$
 16. IF $L_{r_{max}} = L_j$ THEN $Q_V = Q_V + 1,\ j = j+1$, go to step 14.
 17. ELSE IF $L_{r_{max}} \neq L_j$ THEN set $j = j+1$ and go to step 14.
 18. set $w = 0.4$, $Q = w \cdot Q_T + (1-w) \cdot Q_V$
:END
```

Algorithm 4 : Particle swarm optimization algorithm

```
:BEGIN
  1. Define the number $P$ of particles and the number $T$ of time steps for
     the evolution.
  2. Set the time $t = 0,\ t \in [0,T]$ and initialize the positions of each parti-
     cle with uniformly distributed values; within $(0,5]$ for $\mu,\lambda$ and within
     $(0,1]$ for Threshold of rule size. Moreover, evaluate the particles by
     computing the objective function according to their positions.
  3. Update positions
  4. Evaluate the particles by computing the objective function according to
     their new positions.
  5. IF Termination criterion, i.e $t = T$, is satisfied then go to :END
  6. go to step 3
:END
```

Algorithm 5 : Kernel function that calculates the inclusion measure in the form of Equation (9).

```
:BEGIN
  1. $i = Get\_MyBlock\_Identity$; $h = Get\_MyLocal\_Identity$;
  2. $R[i,h] = K(F,G;i,h)$
  3. barrier()
  4. Reduction on R
:END
```

REFERENCES

[1] V Gomathi, K Ramar, and AS Jeeyakumar. (2009). Human facial expression recognition using manfis model. *World Academy of Science, Engineering and Technology*, 50:338–342.

[2] M. Grana. (2009). Lattice computing and natural computing. *Neurocomputing*, 72(10-12):2065–2066.

[3] M. Grana and A.I. Gonzalez-Acuna. (2013). Learning parsimonious dendritic classifiers. *Neurocomputing*, 109:3–8.

[4] A. Habermaier and A. Knapp. (2012). On the correctness of the simt execution model of gpus. *Programming Languages and Systems*, pages 316–335.

[5] D. Horn. (2005). Stream reduction operations for GPGPU applications. *Gpu gems*, 2:573–589.

[6] V. G. Kaburlasos. (2011). Information engineering applications based on lattices. *Information Sciences*, 181(10):1771–1773.

[7] V. G. Kaburlasos and A. Kehagias. (in press). Fuzzy inference system (FIS) extensions based on lattice theory. *IEEE Transactions on Fuzzy Systems*.

[8] V. G. Kaburlasos and T. Pachidis. (in press). A lattice-computing ensemble for reasoning based on formal fusion of disparate data types, and an industrial dispensing application. *Information Fusion*.

[9] V. G. Kaburlasos, S. E. Papadakis, and A. Amanatiadis. (2012). Binary image 2d shape learning and recognition based on lattice computing (lc) techniques. *Journal of Mathematical Imaging and Vision*, 42(2-3):118–133.

[10] A. Kehagias. (2011). Some remarks on the lattice of fuzzy intervals. *Information Sciences*, 181(10):1863–1873.

[11] J. Kennedy and R. Eberhart. (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948.

[12] O. Langner, R. Dotsch, G. Bijlstra, D.H.J. Wigboldus, S.T. Hawk, and A. van Knippenberg. (2010). Presentation and validation of the radboud faces database. *Cognition & Emotion*, 24:1377–1388.

[13] M. J. Lyons, S. Akamatsu, M. Kamachi, and J. Gyoba. (1998). Coding facial expressions with gabor wavelets. In *Proc. IEEE Intl. Conference on Automatic Face and Gesture Recognition*, pages 200–205, Nara, Japan.

[14] B.W. Miners and O.A. Basir. (2005). Dynamic facial expression recognition using fuzzy hidden markov models. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 2, pages 1417–1422.

[15] R. Mukundan and K.R. Ramakrishnan. (1998). *Moment functions in image analysis*. World Scientific, Singapore.

[16] J. Nickolls, I. Buck, M. Garland, and K. Skadron. (2008). Scalable parallel programming with cuda. *Queue- GPU Computing*, 6(2):40–53.

[17] S. E. Papadakis and V. G. Kaburlasos. (2010). Piecewise-linear approximation of non-linear models based on probabilistically/possibilistically interpreted intervals' numbers (ins). *Information Sciences*, 180(24):5060–5076.

[18] S. E. Papadakis, V. G. Kaburlasos, and G. A. Papakostas. (2012). Human facial expression recognition by fuzzy lattice reasoning (flr). In *Proc. 10th Intl. FLINS Conf. on Uncertainty Modeling in Knowledge Engineering and Decision Making (FLINS'12)*, pages 633–638, Istanbul, Turkey.

[19] S.E. Papadakis and A.G. Bakrtzis. (2011). A GPU accelerated PSO with application to Economic Dispatch problem. In *Intelligent System Application to Power Systems (ISAP), 2011 16th International Conference on*, pages 1–6.

[20] G. A. Papakostas, E. G. Karakasis, and D. E. Koulouriotis. (2010). Novel moment invariants for improved classification performance in computer vision applications. *Pattern Recognition*, 43(1):58–68.

[21] G. A. Papakostas, D. E. Koulouriotis, and E. G. Karakasis. (2009). A unified methodology for efficient computation of discrete orthogonal image moments. *Information Sciences*, 179(20):3619–3633.

[22] D. Roger, U. Assarsson, N. Holzschuch, *et al.* (2007). Efficient stream reduction on the GPU. In *Workshop on General Purpose Processing on Graphics Processing Units*.

[23] D. G. Sim, H. K. Kim, and R. H. Park. (2004). Invariant texture retrieval using modified zernike moments. *Image & Vis. Comput.*, 22(4):331–342.

[24] Talia D. Skillicorn D. B. (1998). Models and languages for parallel computation. *ACM Computing Surveys (CSUR)*, 30:123–169.

[25] P. Viola and M. J. Jones. (2004). Robust real-time face detection. *Intl. Journal of Computer Vision*, 57(2):137–154.

[26] Y. Xu, D. Ruan, K. Qin, and J. Liu. (2003). *Lattice-Valued Logic*. Springer, Heidelberg, Germany.