X Congresso Brasileiro de Inteligência Computacional (CBIC'2011), 8 a 11 de Novembro de 2011, Fortaleza, Ceará
© Sociedade Brasileira de Inteligência Computacional (SBIC)

COPROCESSADOR FUZZY PARA CLASSIFICAÇÃO DE IMAGENS MULTIESPECTRAIS OTIMIZADO POR PROGRAMAÇÃO EVOLUCIONÁRIA

Rodrigo Gomes de Souza & Wellington Pinheiro dos Santos
Núcleo de Engenharia Biomédica, Universidade Federal de Pernambuco
rodrigog2@gmail.com, wellington.santos@ieee.org

Resumo – O presente artigo apresenta o desenvolvimento de uma proposta de arquitetura de *hardware* de um coprocessador dedicado à classificação de imagens multiespectrais usando regras *fuzzy* definidas através de conjuntos de pertinência montados a partir de funções gaussianas. Nesta arquitetura, a computação dessas funções é realizada a partir da aproximação por funções equivalentes usando circuitos digitais programáveis especificamente projetados para aplicações neuro-*fuzzy*. Esse tipo de aplicação requer um método capaz de atingir um conjunto ótimo de parâmetros que irão permitir que o classificador opere dentro de uma determinada faixa de erro desejável. Para tanto, foi utilizada uma técnica de otimização bioinspirada conhecida como FEP (*Fast Evolutionary Programming*). Para validar o modelo proposto foram utilizadas imagens multiespectrais sintéticas de *pixels*. O estudo focou o desempenho do classificador otimizado pelo algoritmo FEP, sedo analisados tanto o número de iterações quanto os níveis de aptidão atingidos.

Palavras-chave – Sistemas neuro-*fuzzy*, imagens multiespectrais, classificação de imagens, arquiteturas de sistemas digitais, programação evolucionária.

Abstract – This paper proposes the development of a hardware architecture of a fuzzy co-processor for classification of multispectral images whose membership rules are defined by Gaussian functions. In this architecture, the computation of such functions is accomplished by using equivalent functions generated by approximations using a programmable digital circuit specially designed for use in neuro-fuzzy applications. This type of application requires a method capable of achieving an optimal set of parameters that will allow the classifier to operate with the desired levels of error. In order to optimize the architecture proposed, we used a bioinspired optimization technique known as Fast Evolutionary Programming - FEP. In order to validate the proposed model, we used synthetic multispectral images composed by random pixels. Several classification tests were conducted to study the performance of the classifier optimized by FEP algorithm.

Keywords – Neuro-fuzzy images, multispectral images, image classification, architectures of digital systems, evolutionary programming.

1. INTRODUÇÃO

Este artigo propõe o desenvolvimento de um classificador de imagens multiespectrais, *pixel* a *pixel*, através de uma abordagem *hardware-software* híbrida, baseando-se em [1–3]. O projeto em *hardware*, baseado no classificador desenvolvido em [3], é responsável pela realização da classificação dos *pixels*, enquanto que o projeto em *software* é responsável por encontrar o conjunto de parâmetros para sintetizar e programar o classificador em *hardware*, para que este produza respostas com a menor taxa de erros possível. Apesar de baseado no modelo proposto em [1, 3], o módulo em *hardware* não fará uso da abordagem de *lookup-tables*. Ao invés disso, utilizará circuitos digitais que realizam a aproximação das funções gaussianas definidas em [2]. Tais circuitos de aproximação são completamente programáveis através de um conjunto de parâmetros encontrado através de uma busca heurística realizada pelo módulo em *software*. Essa busca pelo melhor conjunto de parâmetros faz uso de técnicas de programação evolucionária através do algoritmo *Fast Evolutionary Programming*, FEP [4, 5].

Este artigo está organizado como segue: xxxxxx.

2. ABORDAGEM PROPOSTA

2.1 Classificação de Imagens

A análise de imagens digitais geradas por sensores remotos oferece uma grande quantidade de aplicações para os mais variados fins. Ao visualizar tais imagens, muitas vezes o ser humano é capaz de realizar abstrações e identificar padrões de maneira rápida e precisa, porém a tentativa de automatização de tais tarefas pelo uso de um computador a revela como um processo complexo e trabalhoso. Em muitas aplicações de reconhecimento de padrões e classificação de imagens, principalmente se originadas por sensoriamento remoto, é desejável construir um sistema que faça uso de *hardware* desenvolvido especificamente para o problema ao invés de um *hardware* de propósito geral. Isso se dá pelo fato que em *hardware* de uso específico é possível controlar o consumo de energia ou usar paralelismo em tarefas semelhantes - através da replicação dos módulos envolvidos.

O desenvolvimento de tal *hardware*, além de aspectos sobre o consumo de energia, deve preocupar-se também com o tempo de resposta na classificação e do uso de memória. A utilização de imagens de alta resolução agrava ambos os requisitos e, dependendo da arquitetura sugerida, pode inviabilizar o projeto [2, 3].

2.2 Classificador com Hardware Dedicado

A classificação de imagens e o reconhecimento de padrões são técnicas capazes de resolver com eficácia um grande número de problemas computacionais que fatalmente não teriam solução quando aplicadas técnicas tradicionais de computação. Porém, o emprego de técnicas que fazem uso de conceitos de computação inteligente normalmente demanda altos níveis de processamento e/ou alocação de memória e tal implicação ainda é intensificada caso as técnicas envolvidas utilizem funções não lineares, as quais normalmente fazem uso de operações com números de ponto flutuante [6, 7]. Neste cenário, seria necessária uma complexa arquitetura capaz de realizar tais operações e isso contribuiria bastante para um aumento considerável no custo final do projeto.

Diante de tais requisitos, é sugerido o desenvolvimento do projeto de uma arquitetura dedicada para solução do problema específico. Tal solução deve explorar recursos como paralelismo, aproximação de funções não lineares e o uso de técnicas de computação inteligente que busquem minimizar a taxa de erros de classificação. Dessa forma, a solução sugerida será capaz de atender a exigentes requisitos de desempenho e alocação de memória.

2.3 Lógica Fuzzy e Classificação

A lógica *fuzzy* pode ser vista como um complemento à teoria clássica dos conjuntos especialmente sobre o conceito de pertinência. Um conjunto clássico, por definição, separa os indivíduos de um universo de discurso em dois grupos: membros - aqueles que pertencem ao conjunto em questão, - e não membros - aqueles que não pertencem a tal conjunto. Não existe ambiguidade ou dúvidas quanto a essa divisão. Porém, em muitas das categorias ou classes que usamos no dia a dia, por exemplo, ao definir a classe dos adultos baixos, altos e de estatura média, o uso de conjuntos clássicos pode não modelar bem a separação de adultos dentre tais classes. Isso ocorre porque tais classes, definidas através linguagem natural, apresentam ambigüidade e seus limites são vagos e subjetivos, assim a transição entre os membros não ocorre de forma brusca e sim de maneira gradual. Com isso, o uso de conjunto *fuzzy* insere ambigüidade ao mesmo tempo em que remove a fronteira entre membros e não membros.

Para ser completamente definido, um conjunto *fuzzy* requer, além de seus membros, uma função de pertinência capaz de informar o nível de compatibilidade de um indivíduo às características do conjunto *fuzzy* em discussão. Grau de pertinência é o nome dado a esse nível de compatibilidade. A definição matemática de um conjunto *fuzzy* é dada, então, pelo agrupamento de pares formados pelos indivíduos e seus respectivos níveis de pertinência para tal conjunto, os quais normalmente estão compreendidos no intervalo fechado $[0, 1]$ [8–10].

O emprego de lógica *fuzzy* na classificação *pixel* a *pixel* de imagens multiespectrais é bastante indicado, pois seu novo conceito de pertinência de um elemento a um conjunto faz-se bastante útil no processo de separação de elementos entre classes cujos limites não são claramente definidos. Além disso, a utilização em conjunto das funções de pertinência permite transparência na definição das regras de decisão de classificação. Isso permite a combinação de diversas outras características e até mesmo uma personalização na forma de analisar o problema. É possível, por exemplo, criar uma classe de rejeição e uma regra para tal classe baseando-se nas saídas do sistema. Contudo, é importante alertar que a escolha do uso de lógica *fuzzy* para o classificador proposto exige o desenvolvimento de um meio de obtenção dos parâmetros responsáveis pela definição das funções de pertinência para cada uma das classes em questão.

2.4 Atendendo aos Requisitos: Solução Híbrida *Hardware-Software*

Sistemas baseados em lógica *fuzzy* com capacidade de aprendizagem e arquitetura neural, conhecidos como neuro-*fuzzy*, demandam grande esforço computacional, já que esse tipo de técnica usualmente possui pouco conhecimento estatístico prévio e muito frequentemente lida com tarefas de elevado grau multidimensional [11]. Em computação neuro-*fuzzy*, a informação é representada e processada através de funções de pertinência. A computação de tais funções, através de funções gaussianas (ou mesmo triangulares) é um dos principais fatores responsáveis pela queda do desempenho de tais sistemas [2, 3].

Assim, em sistemas que fazem uso de lógica *fuzzy*, surge uma questão a ser resolvida a respeito da arquitetura do classificador: o uso de uma álgebra de operações com números de ponto flutuante no cálculo de funções gaussianas necessário ao cálculo das funções de pertinência. O desenvolvimento de um *hardware* dedicado com tal recurso aumentaria drasticamente o tempo e o custo final do projeto. Como em sistemas que usam lógica *fuzzy* não é necessário assumir nenhuma distribuição específica de classes, a solução adotada em alguns trabalhos foi a utilização de *lookup-tables*, que são tabelas de pesquisa direta nas quais o cálculo das funções de pertinência fora feito previamente [1–3]. Nessa abordagem, cada um dos possíveis valores de entrada já se encontrava associado aos seus respectivos valores correspondentes à função de pertinência (*Membership Function*, MF). Essa abordagem resolve os problemas de dependência de operações com ponto flutuante e de tempo de resposta (pois pode ser implementada de forma paralela e suas funções de pertinência respondem rapidamente devido ao uso das *lookup-tables*), porém seu uso implica no surgimento de um novo problema: o excessivo uso de memória para alocação das tabelas. Tal questão é agravada com o aumento da quantidade de bandas e/ou de classes, pois será necessária a alocação em memória de mais tabelas, o que pode tornar proibitivo o desenvolvimento do projeto.

O desenvolvimento de um classificador neuro-*fuzzy*, com funções de pertinência que não necessitem nem alocar grandes porções de memória - principalmente com um grande número de bandas e classes - nem desenvolver em *hardware* funções de

© Sociedade Brasileira de Inteligência Computacional (SBIC)

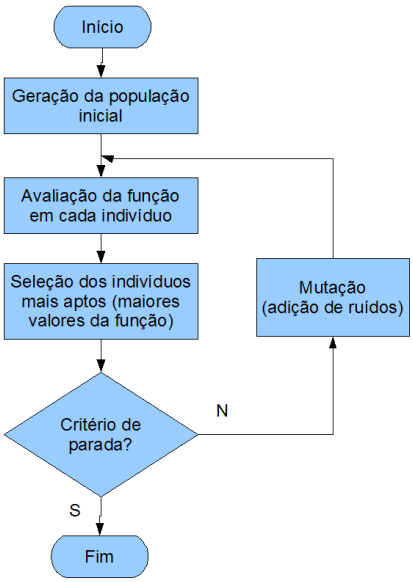

Figura 1: Fluxograma geral dos algoritmos EPs

ponto flutuante, tornar-se-ia possível através do uso de um circuito digital capaz de realizar o cálculo aproximado da função gaussiana. Tais circuitos devem ser programáveis a ponto de ser possível ajustar valores como média, desvio padrão e valor de pico, além de oferecer uma pequena taxa de erro na aproximação com poucas iterações. Em [12], um circuito digital com tais características foi projetado pelos seus autores. Trata-se de um circuito digital capaz de aproximar a função gaussiana por meio de um algoritmo que faz uso de um método de interpolação recursiva, desenvolvido pelo próprio autor. O algoritmo gera um conjunto de segmentos de reta capazes de aproximar o gráfico de qualquer função gaussiana.

2.5 Programação Evolucionária

A Programação Evolucionária (*Evolutionary Programming*, EP) é uma das principais famílias de técnicas da Computação Evolucionária. A EP foi inicialmente desenvolvida como uma abordagem de Inteligência Artificial, mas que passou a ser largamente empregada na resolução de problemas numéricos, dentre eles os problemas de busca e otimização [4, 13, 14].

Os algoritmos de EP são muito semelhantes aos algoritmos genéticos, tendo como principais diferenças a ausência da representação na forma de cromossomos e de operadores de cruzamento: nos EPs os indivíduos são representados por vetores de pesos reais ao invés de cromossomos; os indivíduos filhos são gerados a partir da mutação dos pais, simplesmente, sem cruzamento, podendo parte dos indivíduos filhos serem gerados a partir de clonagem; a mutação consiste na adição de algum tipo de ruído aos pesos dos indivíduos; já a seleção é feita a partir dos melhores indivíduos da união das populações de pais e filhos [4].

Em problemas de busca e otimização, os EPs são definidos pelo seguinte algoritmo generalizado, ilustrado pela figura 1:

1. Geração da população inicial de indivíduos, na forma de vetores de pesos, onde cada indivíduo é um candidato à solução;
2. Repetir até que o critério de parada seja atingido:
   (a) Exposição dos indivíduos ao ambiente, ou seja, avaliação da função objetivo em todos os indivíduos da população;
   (b) Busca do indivíduo mais apto, ou seja, do indivíduo associado ao maior valor da função objetivo;
   (c) Geração dos novos indivíduos por mutação dos indivíduos mais aptos escolhidos dentre os indivíduos anteriores.

Assim, o problema de otimização de uma função objetivo $f : S \to \mathbb{R}$, onde $S \subseteq \mathbb{R}^n$, sendo n a dimensionalidade do problema, considerando um conjunto inicial de m indivíduos candidatos à solução e representados pelos vetores $\mathbf{x}_i$, seria resolvido heuristicamente pela atualização dos vetores segundo expressões semelhantes à que segue:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{r}_i(t), \tag{1}$$

para $1 \leq i \leq m$, onde $\mathbf{r}_i(t)$ é um vetor aleatório n-dimensional cujas coordenadas $r_{i,j}(t)$, para $1 \leq j \leq n$, são tipicamente números aleatórios com distribuição gaussiana, podendo obedecer a outras distribuições probabilísticas, como a de Cauchy, ou mesmo ser obtidos por processos iterativos próprios envolvendo números aleatórios de distribuições quaisquer [4]. As iterações continuam até que se atinja uma determinada quantidade máxima de iterações ou que um dos seguintes critérios seja atingido:

$$||\mathbf{x}^* - \tilde{\mathbf{x}}(t)|| < \epsilon, \tag{2}$$

$$|f(\mathbf{x}^*) - f(\tilde{\mathbf{x}}(t))| < \epsilon, \tag{3}$$

para

$$f(\tilde{\mathbf{x}}(t)) = \max_{1 \leq j \leq m} f(\mathbf{x}_j(t)), \tag{4}$$

onde $\mathbf{x}^*$ é o ponto de máximo da função objetivo f e $f(\mathbf{x}^*)$ é o valor máximo de f, enquanto $\tilde{\mathbf{x}}(t)$ e $f(\tilde{\mathbf{x}}(t))$ são suas respectivas estimativas na t-ésima geração; ϵ é a tolerância da solução. Os critérios das expressões 2 e 3 são critérios mutuamente excludentes, não devendo, portanto, ser utilizados juntos.

O algoritmo CEP (*Classical Evolutionary Programming*) é baseado na estratégia de mutação gaussiana e funciona da seguinte maneira [4, 5]:

1. Geração de uma população inicial de m indivíduos n-dimensionais, representados pelo par $(\mathbf{x}_i, \mathbf{s}_i)$, $\forall i \in \{1, 2, \ldots, m\}$, onde $\mathbf{x}_i$ são variáveis objetivas, ou seja, candidados à solução que otimiza a função objetivo $f : S \to \mathbb{R}$, onde $S \subseteq \mathbb{R}^n$ e n é a dimensionalidade do problema, enquanto $\mathbf{s}_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,n})^T$ são os desvios padrões para as mutações gaussianas, também chamados de parâmetros estratégicos em algoritmos evolucionários auto-adaptativos;

2. Repetir até que um determinado critério de parada seja satisfeito:

   (a) Avaliação da função objetivo nos indivíduos pais, $(\mathbf{x}_i, \mathbf{s}_i)$, $\forall i \in \{1, 2, \ldots, m\}$, gerando $f(\mathbf{x}_i)$;

   (b) Geração de m indivíduos filhos por mutação, $(\mathbf{x}'_i, \mathbf{s}'_i)$, $\forall i \in \{1, 2, \ldots, m\}$, de forma que:

   $$s'_{i,j} = s_{i,j} \exp(\tau_a G(0,1) + \tau_b G_j(0,1)), \tag{5}$$

   $$x'_{i,j} = x_{i,j} + s'_{i,j} G_j(0,1), \tag{6}$$

   $\forall i \in \{1, 2, \ldots, m\}$ e $\forall j \in \{1, 2, \ldots, n\}$, onde $G(0,1)$ é uma variável aleatória gaussiana de média 0 e variância 1 gerada para um i fixo, enquanto $G_j(0,1)$ é gerada para cada j; já os parâmetros de controle são definidos como $\tau_a = 1/\sqrt{2n}$ e $\tau_b = 1/\sqrt{2\sqrt{n}}$;

   (c) Avaliação da função objetivo nos indivíduos filhos, $(\mathbf{x}'_i, \mathbf{s}'_i)$, $\forall i \in \{1, 2, \ldots, m\}$, gerando $f(\mathbf{x}'_i)$;

   (d) Seleção dos m indivíduos mais aptos dentre a população de $2m$ indivíduos composta da junção das populações de pais e de filhos, para compor a nova geração de pais: em problemas de minimização isso significa selecionar os m indivíduos $(\mathbf{x}_i, \mathbf{s}_i)$ ou $(\mathbf{x}'_i, \mathbf{s}'_i)$ com menores avaliações da função objetivo, ou seja, com os m menores valores de $f(\mathbf{x}_i)$ ou $f(\mathbf{x}'_i)$. Idem para problemas de maximização.

O algoritmo FEP (*Fast Evolutionary Programming*) é uma modificação do algoritmo CEP proposta por Yao *et al.* (1999) [4] e se diferencia do CEP pelo uso da mutação de Cauchy, definida como segue [4, 5]:

$$s'_{i,j} = s_{i,j} \exp(\tau_a G(0,1) + \tau_b G_j(0,1)), \tag{7}$$

$$x'_{i,j} = x_{i,j} + s'_{i,j} \delta_j, \tag{8}$$

$\forall i \in \{1, 2, \ldots, m\}$ e $\forall j \in \{1, 2, \ldots, n\}$, onde δ_j é uma variável aleatória de Cauchy padrão gerada para cada j. A função densidade de probabilidade de Cauchy padrão é dada por:

$$f_C(x) = \frac{1}{\pi(1 + x^2)}. \tag{9}$$

Os algoritmos EPs variam basicamente de acordo com a estratégia de mutação. Várias estratégias de mutação têm sido propostas na última década. A mutação gaussiana é a mutação clássica em EPs (CEP), mas não é muito boa para otimização de funções multimodais [4, 5]. A mutação de Cauchy foi proposta no algoritmo FEP e pode gerar melhores soluções do que o CEP em funções multimodais, mas é menos eficiente do que o CEP em problemas unimodais [5]. A mutação baseada na distribuição de Lévy, do algoritmo LEP, foi proposta como um algoritmo mais flexível do que CEP e FEP, e é equivalente ao FEP quando o parâmetro de escala $\beta = 1$, e se reduz ao CEP quando $\beta = 2$, mas na prática é difícil determinar o melhor valor de β para um determinado problema [5].

Os algoritmos EPs são relativamente simples de implementar. No entanto, em problemas de busca e otimização de funções, resultam em muitas iterações antes de atingir resultados aceitáveis para determinadas classes de funções, além de exigir uma população inicial relativamente grande, o que resulta, aliado a uma quantidade razoável de iterações, um grande número de avaliações da função antes de se atingir uma solução aproximada razoável, o que implica em alto custo computacional caso a função seja de avaliação computacionalmente custosa ou matematicamente complexa e se encaixe na classe de funções para a qual o algoritmo não é ótimo [4].

2.6 Módulo em Software

O classificador nebuloso cujas MF são definidas em termos de funções gaussianas exige um meio de encontrar quais são os pares (média, desvio padrão) de cada banda em cada classe. Em outras palavras, tal classificador requer um sistema que indique quais parâmetros definem cada uma das classes a serem usadas na classificação. Para solução dessa questão, foi sugerido o desenvolvimento, em software, de uma ferramenta baseada em programação evolucionária. Tal abordagem modelou o problema da busca pelos parâmetros do classificador em um problema de otimização no qual a solução seria encontrar o conjunto de parâmetros que permitissem ao classificador operar com os níveis de erro desejados. Nessa ferramenta, o classificador foi definido como um vetor de parâmetros que contém todos os pares média-desvio de cada banda em cada classe.

O desenvolvimento dessa solução para tal problema de otimização foi baseada no método FEP. Nessa abordagem, a aplicação desenvolvida gera conjuntos aleatórios de parâmetros e testa sua aptidão em classificar um conjunto de *pixels* cujas classes são previamente conhecidas.

2.7 Arquitetura Proposta

Quando o mapeamento algoritmo-*hardware* é realizado de forma adequada e são escolhidas as técnicas corretas, dificilmente o desempenho de um *hardware* dedicado na execução desse algoritmo será inferior ao de uma solução que use *hardware* de propósito geral. Isso pode ser evidenciado pelo fato que, em uma arquitetura dedicada, é possível fazer uso de unidades funcionais especialmente desenvolvidas e otimizadas para uma tarefa específica, além de existir a possibilidade de implantar paralelismo através da múltipla alocação dessas diversas unidades funcionais [3, 12].

2.7.1 Unidades Funcionais

UFG - Unidade de Função Gaussiana: É unidade fundamental da arquitetura. Baseia-se no circuito digital sugerido por [12], para representar uma função gaussiana. Portanto, uma classe é definida através da alocação de uma instância da UFG para cada banda. Essa unidade é responsável por calcular a pertinência local de um *pixel* para uma banda em dada classe. Para realizar essa função, a UFG utiliza três entradas de um *byte* cada, das quais duas são usadas para efetuar programação do módulo, chamadas M e D, que representam o centro e a largura da MF, correspondendo às entradas c_x e m, respectivamente, do modelo de [12]. Outra entrada, chamada de B, é usada para receber o valor do *pixel* para a banda definida localmente, e corresponde a entrada x no modelo descrito por [12]. A entrada q do modelo descrito por [12] não terá seu valor alterado durante a operação do classificador, o qual corresponde ao grau de interpolação e, portanto, receberá sempre o valor três, indicado como o mais adequado pelo autor. Além dessas entradas de dados, ainda existem dois bits de controle, os quais determinarão a operação a ser realizada: programação do módulo ou cálculo do valor da função gaussiana. A UFG terá uma única saída, chamada de G, responsável por retornar o valor do cálculo da função gaussiana.

UFP - Unidade de Função de Pertinência: É a unidade principal da arquitetura proposta. É responsável por definir o grau de pertinência do *pixel* recebido na entrada para cada uma das bandas, ou seja, é um encapsulamento de várias UFG. O diagrama na Figura 2 exibe como é feita a conexão entre os quatro módulos UFG. Note que o módulo UFP possui quatro saídas (P1 a P4) as quais correspondem às saídas das UFG que são responsáveis por informar o nível de pertinência do *pixel* recebido na entrada para cada uma das quatro bandas.

MINU - Unidade de Mínimo: O módulo MINU é responsável por informar o quanto o *pixel*, recebido em sua entrada, é semelhante à classe definida em através da intersecção do grau de pertinência de todas as bandas. Como em lógica *fuzzy* a intersecção é equivalente a operação de mínimo, a tarefa desse módulo é informar qual dos valores recebidos em suas quatro entradas é o menor. Como a sua função sugere, a MINU é uma das unidades mais simples da arquitetura. É formada por três unidades do tipo MIN2 que, juntas, são capazes de informar qual é o menor valor dentre os quatro recebidos nas entradas da MINU. Cada módulo MIN2, ao receber um valor em cada uma de suas duas entradas, tem a simples tarefa de informar qual das entradas é a menor.

UFD - Unidade de Função Discriminante: Sua função é realizar a interconexão entre o módulo UFP e o MINU. Dessa forma, na saída da UFD será informado qual o nível de pertinência do *pixel* para a classe localmente definida.

MAXU - Unidade de Máximo: A MAXU recebe o grau de pertinência encontrado para todas as classes e retorna o maior grau de pertinência (saída P) e o identificador da classe vencedora (saída I). A Unidade de Máximo é baseada no módulo MAX2, o qual é bastante semelhante ao MIN2, discutido anteriormente, diferenciando-se pela operação de cálculo do máximo valor recebido (ao invés do mínimo) e pela introdução de dois novos sinais. Assim, além dos valores a serem comparados, cada módulo MAX2 recebe um identificador de classe para cada uma de suas duas entradas. É através desse identificador que cada MAX2 pode informar qual UFP (qual das duas classes) retornou aquele valor, além do maior de pertinência. Como a arquitetura é sugerida para um classificador de quatro classes, então os identificadores precisam de apenas dois bits para identificar todas as classes envolvidas.

Diagrama da arquitetura: Finalizada a descrição uma a uma de todas as unidades funcionais da arquitetura, torna-se possível descrever com poucas palavras o funcionamento desta. O diagrama da arquitetura é ilustrado na Figura 3 e deve ser usado para possibilitar uma melhor compreensão da mesma.

© Sociedade Brasileira de Inteligência Computacional (SBIC)

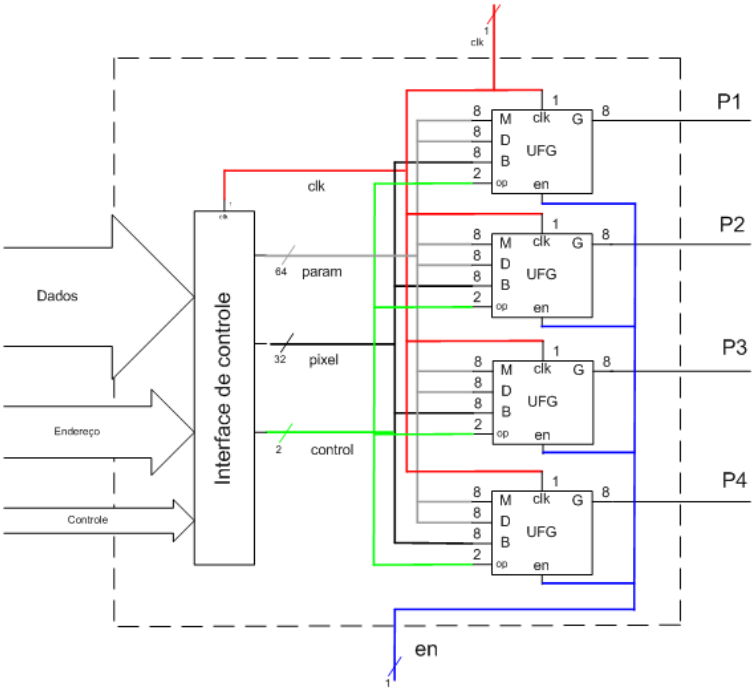

Figura 2: Unidade de Função de Pertinência

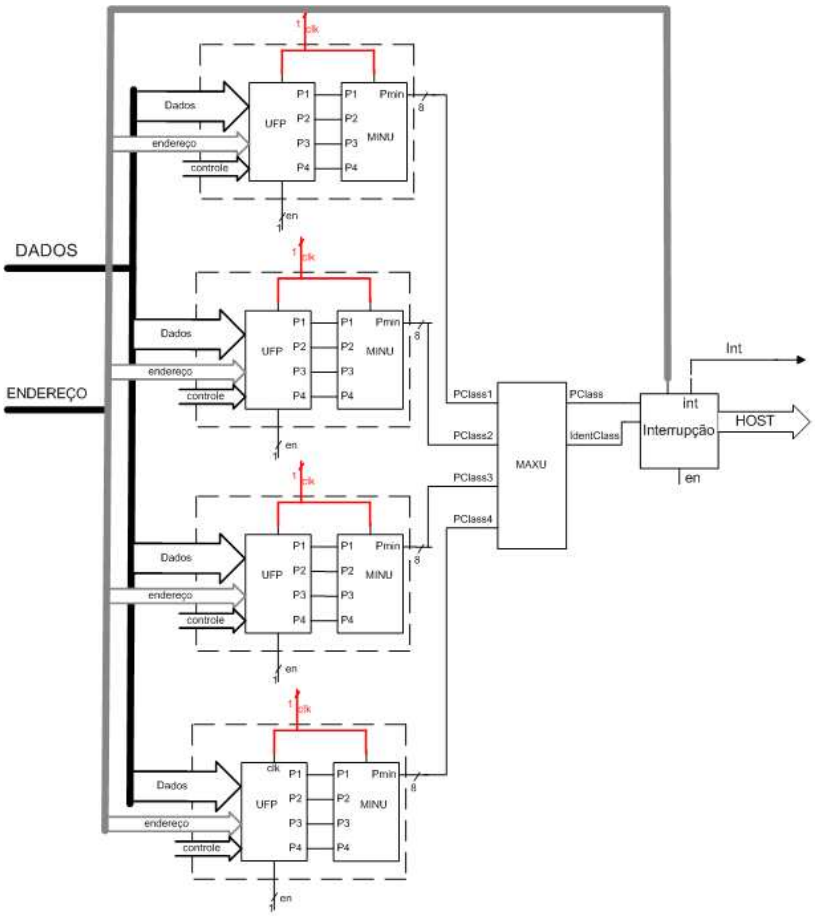

Figura 3: Diagrama completo da arquitetura

© Sociedade Brasileira de Inteligência Computacional (SBIC)

Tabela 1: Parâmetros fixados na simulação

Parâmetro	Valor
Tamanho da população	50
Número de Oponentes na etapa de confronto	10
Aptidão requerida como critério de parada	0,975
Número de classes	4
Número de bandas	4
Número de pixels sintéticos gerados por classe	150

Tabela 2: Estatísticas do experimento

	Aptidão	#Iterações
Média	0,92	133,12
Desvio	0,09	76,09
Moda	1,00	200
Mediana	0,96	200

3. RESULTADOS

Os resultados foram obtidos a partir de simulações do algoritmo FEP na busca pelo conjunto de parâmetros que permita ao classificador nebuloso obter os índices de erro e de convergência desejados. Na realização das 50 simulações do experimento foi utilizado o mesmo conjunto de pixels sintéticos. Nas simulações foi utilizado o conjunto de parâmetros que definem o comportamento do algoritmo FEP e a arquitetura do classificador testado, exibido na Tabela 1. No experimento foram avaliados classificadores para quatro classes com quatro bandas cada. Para cada um dessas classes foram gerados 150 pixels sintéticos.

A Tabela 2 exibe as medidas estatísticas calculadas a partir dos resultados obtidos no experimento.

Os gráficos da Figura 4 e da Figura 5 ilustram, respectivamente, o número de iterações usadas e a aptidão alcançada em cada uma das 50 simulações.

O gráfico da Figura 4 ilustra uma certa irregularidade, também detectada através do valor alto do desvio padrão (ver Tabela 2), com relação ao número de iterações usadas. Uma das possíveis causas dessa irregularidade são as distorções existentes no conjunto sintético de *pixels* utilizado. Essa má formação ocorre quando os valores de desvio padrão, escolhidos aleatoriamente para a geração sintética desses *pixels* têm valores elevados. Isso implicaria no aumento da chance de serem gerados, para uma mesma classe, *pixels* cujos valores sejam consideravelmente diferentes em uma mesma banda.

Nos índices de aptidão é perceptível uma maior regularidade dos valores obtidos. Isso também pode ser constatado através do baixo valor do desvio padrão. Os valores de mediana e moda para tais distribuições mostram que, mesmo nas simulações que alcançaram o limite de 200 iterações, o grau de aptidão requerido está próximo de ser alcançado.

4. CONCLUSÃO

O presente artigo apresentou uma proposta de arquitetura para classificação de imagens multiespectrais que explora o paralelismo e a capacidade de programação do classificador, ou seja: o treinamento é feito em *software*, por meio da otimização da taxa de acerto, dado um determinado conjunto de treinamento. A otimização foi baseada no algoritmo FEP, um algoritmo de programação evolucionária baseado em uma melhoria do algoritmo CEP. O resultado do processo de treinamento é um conjunto de parâmetros de configuração do classificador em *hardware* proposto, baseado na implementação de um sistema neuro-*fuzzy* com funções de pertinência baseadas na composição (operações de intersecção) de funções de pertinência menores, formadas

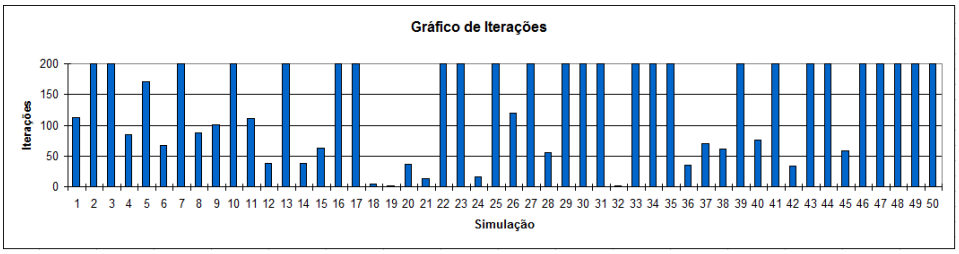

Figura 4: Número de iterações por simulação

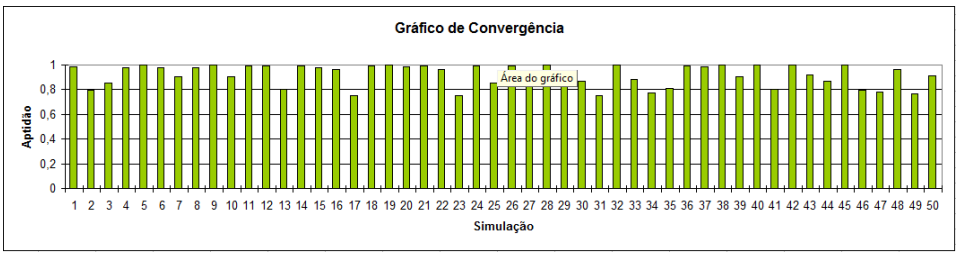

Figura 5: Taxa de aptidão por simulação

por funções gaussianas. Foram feitas simulações usando imagens multiespectrais sintéticas compostas de *pixels* com níveis de cinza aleatórios.

Os resultados mostraram que é possível gerar arquiteturas otimizadas para classificação de imagens multiespectrais, o que pode ter um impacto importante a médio prazo na análise em tempo real de imagens de sensoriamento remoto e de imagens médicas.

Como trabalhos futuros, a arquitetura proposta será estudada levando em conta os tempos de propagação e o consumo de energia elétrica. Nessa etapa serão utilizados também métodos numéricos de otimização, para gerar uma versão do classificador proposto com uma relação entre o consumo e a área de circuito otimizada.

REFERÊNCIAS

[1] W. P. Santos, M. S. C. Araï¿½jo Filho, A. J. B. Barros Neto, C. K. R. Silva, P. V. Silva, J. D. Silva and A. C. Frery. "Uma Nova Tï¿½cnica de Classificaï¿½ï¿½o de Imagens Baseada em Lï¿½gica Nebulosa Nï¿½o Iterativa". In *XI Simpï¿½sio Brasileiro de Sensoriamento Remoto*, pp. 1091–1098, Belo Horizonte, Brasil, 2003. Sociedade Brasileira de Sensoriamento Remoto.

[2] P. V. Silva, W. P. Santos, M. E. Lima and A. C. Frery. "A Fuzzy VLSI Architecture for Multispectral Image Classification". In *XI Simpï¿½sio Brasileiro de Sensoriamento Remoto*, pp. 1107–1114, Belo Horizonte, Brasil, 2003. Sociedade Brasileira de Sensoriamento Remoto.

[3] A. C. Frery, W. P. Santos, P. V. Silva, M. S. C. Araï¿½jo Filho, C. K. R. Silva, A. J. Barros Neto and M. E. Lima. "A fuzzy architecture for classification of multi- and hyperspectral images". In *Nonlinear Signal and Image Processing*, pp. 1–5, Grado-Trieste, Italy, 2003. IEEE - Eurasip. CD-ROM.

[4] X. Yao, Y. Liu and G. Lin. "Evolutionary Programming Made Faster". *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.

[5] H. Dong, J. He, H. Huang and W. Hou. "Evolutionary Programming using a mixed mutation strategy". *Information Sciences*, vol. 177, pp. 312–327, 2007.

[6] R. Duda, P. Hart and D. G. Stork. *Pattern Classification*. John Wiley and Sons, 2001.

[7] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1972.

[8] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

[9] J. Abonyi and J. A. Roubos. "Structure Identifications of Fuzzy Classifiers". In *The 5th Online World Conference on Soft Computing in Industrial Applications*, 2000.

[10] J. A. Roubos, M. Setnes and J. Abonyi. "Learning Fuzzy Classification Rules from Labeled Data". *International Journal of Information Sciences*, 2001.

[11] F. M. Guedes. "Sistemas com aprendizado utilizando lï¿½gica fuzzy", 2007.

[12] K. Basterretxea, J. M. Tarela and I. del Campo. "Digital Gaussian membership function circuit for neuro-fuzzy hardware". *Electronics Letters*, vol. 42, no. 1, pp. 44–46, 2006.

[13] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2007.

[14] R. Eberhart and Y. Shi. *Computational Intelligence: concepts to implementations*. Morgan Kaufmann, 2007.