UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"

Pós-Graduação em Ciência da Computação

Carlos Roberto Dutra Caldas Júnior

Implementação em *hardware* de um sistema inteligente para detecção de plantas daninhas em plantações de soja utilizando máquinas de vetores de suporte e redes neurais artificiais

UNESP

2012

Carlos Roberto Dutra Caldas Júnior

Implementação em *hardware* de um sistema inteligente para detecção de plantas daninhas em plantações de soja utilizando máquinas de vetores de suporte e redes neurais artificiais

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação – Área de Concentração em Sistemas de Computação, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

BANCA EXAMINADORA

Prof. Dr. Norian Marranghello
Professor Doutor
UNESP – São José do Rio Preto
Orientador

Prof. Dr. Adilson Gonzaga
Professor Doutor
USP – São Carlos

Prof. Dr. Rodrigo Capobianco Guido
Professor Doutor
UNESP – São José do Rio Preto

São José do Rio Preto, 2 de Agosto de 2012

Carlos Roberto Dutra Caldas Júnior

Implementação em *hardware* de um sistema inteligente para detecção de plantas daninhas em plantações de soja utilizando máquinas de vetores de suporte e redes neurais artificiais

Orientador: Prof. Dr. Norian Marranghello

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação – Área de Concentração em Sistemas de Computação, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

UNESP

2012

Caldas Júnior, Carlos Roberto Dutra.
Implementação em hardware e um sistema inteligente para a detecção de plantas daninhas em plantações de soja utilizando máquinas de vetores de suporte e redes neurais artificiais / Carlos Roberto Dutra Caldas Júnior. - São José do Rio Preto : [s.n.], 2012.
80 f. : il. ; 30 cm.

Orientador: Norian Marranghello
Dissertação (mestrado) - Universidade Estadual Paulista, Instituto de Biociências, Letras e Ciências Exatas

1. Processamento de imagens. 2. Sistemas inteligentes. 3. Arquitetura reconfigurável. I. Marranghello, Norian. II. Universidade Estadual Paulista, Instituto de Biociências, Letras e Ciências Exatas. IV. Título.

CDU – 004.932

Ficha catalográfica elaborada pela Biblioteca do IBILCE
Campus de São José do Rio Preto - UNESP

AGRADECIMENTOS

Agradeço, primeiramente, a toda minha família, que formou meu caráter, minha personalidade, me proporcionou educação e sempre esteve ao meu lado quando precisei.

Ao professor Norian, pela oportunidade de fazer esse mestrado e pela paciência e pelos diversos auxílios e orientações que me ajudaram a crescer no decorrer dos últimos anos.

Aos meus amigos Cleriston, Carol, Claudia, Leandro, Thiago, Eder, Adelaide que me mostraram o significado da palavra amizade e me deram uma força inacreditável para conseguir superar todas as dificuldades pelas quais passei e terminar a pós-graduação.

Aos meus colegas de laboratório Thais, Roberta, Drica, Rafael e Otávio que foram as pessoas que mais me aguentaram, tarefa nada fácil, ao longo dos últimos anos.

A tantos amigos que passaram pela minha vida nos últimos anos e que poderiam facilmente ter seus nomes aqui, mas o espaço não é tão grande quanto eles.

Àqueles que rezaram por mim, já que, como muitos sabem, não sou muito religioso e que, com certeza, a fé dessas pessoas compensou a minha.

Por fim, agradeço ao CNPq pelo apoio financeiro fundamental para o desenvolvimento desse trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ASIC - *Applicantion Specific Integrated Circuit*

AM - *Aprendizado de Máquina*

CI - Circuito Integrado

CPLD - *Complex Programmable Logic Device*

DUV - *Design Under Verification*

EDA - *Eletronic Design Automation*

FIFO - *First-in First-out*

FPGA - *Field Programmable Gate Array*

HDL - *Hardware Description Language*

IBILCE - Instituto de Biociências Letras e Ciências Exatas

IEEE - *Institute of Electrical and Electronics Engineers*

IPH - Intersecção por Histograma

LUT - *Lookup Table*

pixel - picture element

PLD - *Programmable Logic Device*

PAL - *Programmable Array Logic*

PLA - *Programmable Logic Array*

RAM - *Random Access Memory*

RNA - *Redes Neurais Artificiais*

RTL - *Register Transfer Level*

SPLD - *Simple Programmable Logic Device*

SVM - *Support Vector Machine*

ULA - Unidade Lógico-aritmética

VHDL - *Very High Speed Integrated Circuits Hardware Description Language*

RESUMO

A presença de sistemas automatizados é cada vez mais comum para as pessoas. Seus exemplos vão desde máquinas de lavar, que executam praticamente todo o processo de lavagem e secagem de roupas, até linhas de produção em fábricas dos mais diversos produtos. Esses são exemplos de aplicações que exigem pouca interferência humana no processo, já que as etapas realizadas pelos sistemas são bem definidas e iterativas. Porém, outros tipos de processos podem exigir capacidade de discernimento daquele – ou daquilo – que os executam. Para automatizar esse tipo de processo uma das alternativas é o uso de técnicas de inteligência artificial. Esse trabalho visa realizar uma análise comparativa entre técnicas de inteligência artificial, quais sejam Redes Neurais Artificiais e Máquinas de Vetores de Suporte. Com essa análise espera-se estabelecer qual técnica é mais vantajosa para implementação em *hardware* de sistemas inteligentes, por meio do uso das principais métricas de projeto de circuitos digitais: tamanho do circuito gerado, consumo de energia e desempenho. Para tanto, foram realizados diversos testes com técnicas de pré-processamento e extração de características das imagens para determinar requisitos necessários para o funcionamento do sistema. A partir desses requisitos foram implementadas diversas arquiteturas de sistemas inteligentes para obter-se o classificador mais adequado para resolver o problema. Por fim, o classificador escolhido foi implementado em FPGA na forma de um módulo, o qual se integrará a um sistema maior, para interpretação de imagens digitais para detecção de ervas daninhas em plantações de soja.

Palavras-chave: processamento digital de imagens, sistemas inteligentes, arquiteturas reconfiguráveis, máquinas de vetores de suporte, redes neurais artificiais.

ABSTRACT

Automated systems have become common for people. Examples range from washing machines, which perform almost the entire cloth washing and drying process, to the production of many products. These are examples of applications that require modest human interference, since the steps taken by the systems are well defined and iterative. However, other processes may require a capacity of judgment of the natural or artificial system performing them. An alternative to automate this kind of process is the use of artificial intelligence techniques. This study aims at a comparative analysis of artificial intelligence techniques, namely Artificial Neural Networks and Support Vector Machines. With this analysis we hope to establish which technique is more advantageous for hardware implementation of an intelligent system, through the use of key metrics for digital circuit design: circuit size, power consumption and performance. Therefore, several tests were performed with image preprocessing and feature extraction techniques to determine requirements for system operation. From these requirements, various architectures for intelligent systems were implemented to obtain the most appropriate classifier to solve the problem. Finally, the chosen classifier was implemented in FPGA as a module to fit into a larger system for digital image interpretation for the detection of weeds in crops of soybeans.

Keywords: digital image processing, intelligent systems, reconfigurable architecture, support vector machines, artificial neural network.

Capítulo 1 – Introdução

1.1 Considerações iniciais

A constante evolução tecnológica traz consigo a possibilidade do desenvolvimento de aplicações que, poucos anos antes, não eram viáveis. Uma das áreas com maior perspectiva de progresso é a de sistemas inteligentes. Os primeiros registros de interesse na área datam de 1943, em artigos nos quais McCulloch e Pitts sugeriram a construção de uma máquina baseada no cérebro humano. Ainda na década de 1940, Hebb propôs uma lei que modela o aprendizado para as sinapses dos neurônios. Em 1957, Rosenblatt desenvolveu o *perceptron*, uma Rede Neural Artificial (RNAs) de duas camadas para o reconhecimento de caracteres. O desenvolvimento da área gerou expectativas sobre o surgimento de máquinas complexas demais para a época, que acabaram não se concretizando e ocasionando um corte nos investimentos. A situação só voltou a melhorar a partir da década de 1970 quando Werbos apresentou as bases para o algoritmo de treinamento *backpropagation* [Tatibana e Kaetsu, 1996]. Em 1995, Vapnik baseou-se na teoria do aprendizado estatístico para criar as Máquinas de Vetores de Suporte (*Support Vector*

Machines – SVMs), classificadas por alguns autores como outra categoria de RNAs [Haykin, 1999].

As aplicações na área de sistemas inteligentes são inúmeras e, dentre elas, merecem destaque aquelas que resultam em melhorias socioeconômicas e de sustentabilidade. A agricultura de precisão surge nesse contexto como um conjunto de técnicas para melhorar a produtividade na agricultura enquanto preserva o meio ambiente. Uma dessas técnicas passa pelo uso de insumos agrícolas de forma controlada, sem que áreas que não necessitem de tais insumos sejam submetidas a eles. Uma maneira de se obter esse resultado de forma satisfatória é pelo processamento rápido de imagens da lavoura, por um sistema inteligente no momento da aplicação do insumo, visando averiguar a necessidade do uso do mesmo.

No IBILCE – Instituto de Biociências Letras e Ciências Exatas – já existem pesquisas voltadas para o desenvolvimento de um módulo de pré-processamento de imagens digitalizadas, deixando-as adequadas para identificação de elementos como plantas daninhas [Mertes, Marranghello e Pereira, 2008]. O trabalho de identificação pode ser automatizado com o uso de sistemas inteligentes. Dentre os principais métodos para implementação de sistemas inteligentes estão as redes neurais artificiais e as máquinas de vetores de suporte. Estas técnicas, também chamadas coletivamente de técnicas de aprendizado de máquina (AM), possuem características que as tornam vantajosas ou não, dependendo da aplicação escolhida.

1.2 Objetivos

O objetivo geral deste trabalho foi o desenvolvimento de um sistema inteligente capaz de discernir, dentre imagens digitalizadas de plantações de soja, aquelas em que plantas daninhas estão presentes. O sistema necessita ser capaz de realizar esse processamento em tempo real, para que fosse possível a aplicação de herbicidas apenas nos locais em que fosse necessário. Para tanto, o sistema deve receber como entrada imagens pré-processadas que passarão por uma RNA ou SVM implementadas em *hardware* visando o ganho de desempenho. Além disso, é feita

uma comparação para averiguar quais as vantagens e desvantagens de cada técnica de AM para a aplicação em questão. No decorrer do trabalho foi feita uma análise para determinar se os algoritmos de pré-processamento aplicados às imagens seriam suficientes para prepará-las para o processamento pela RNA e SVM ou se seria necessária a extração de características presentes nas imagens para reduzir quantidade de informações e processá-las.

1.3 Metodologia

Para o desenvolvimento do trabalho, primeiramente, foi realizado um levantamento teórico a respeito dos assuntos abordados: processamento digital de imagens, desenvolvimento de aplicações em *hardware* e técnicas de aprendizado de máquina. Então, foi feita a elaboração de um banco de imagens de plantações de soja a serem utilizadas, buscando aquelas que representam melhor o problema a ser solucionado. Para a implementação dos algoritmos de pré-processamento, extração de características e das técnicas de aprendizado de máquina, foi utilizada a linguagem C. Foram implementadas diferentes configurações de SVMs e RNAs para determinar qual obteria o melhor desempenho contra seu custo de implementação em *hardware*. Com os classificadores obtidos, iniciou-se a etapa de desenvolvimento do *hardware* e sua verificação para garantia de que este executa as tarefas da maneira desejada. Por fim, foram realizadas medições do classificador implementado para determinar seu desempenho, consumo de energia e custo.

1.4 Organização da dissertação

Esta dissertação está organizada da seguinte forma. No Capítulo 2 são apresentados os conceitos sobre processamento digital de imagens e desenvolvimento de aplicações em *hardware* necessários para abstrair o problema para o uso em AM. No Capítulo 3 é feita uma introdução aos principais conceitos de AM, bem como são apresentados os métodos de SVMs e RNAs. No Capítulo 4, é detalhado o processo de implementação em *software* dos filtros, SVMs e RNAs utilizados, bem como são analisados os resultados obtidos nos testes. No Capítulo 5 é

detalhada a implementação em *hardware* do classificador escolhido, bem como o processo de verificação funcional do circuito e a análise das métricas obtidas nos testes executados. Os testes foram separados em cada capítulo, em vez de agrupados, para justificar as decisões de projeto que foram todas Por fim, no Capítulo 6, são expostas as conclusões e as sugestões para trabalhos futuros.

Capítulo 2 – Fundamentação teórica: PDI e *hardware*

2.1 Considerações iniciais

Ao falarmos de aprendizado de máquina, estamos fundamentalmente ligados à aplicação com a qual trabalharemos. Portanto, é necessário um conhecimento prévio do campo onde as técnicas de aprendizado de máquina serão utilizadas. Neste capítulo será feita uma introdução a conceitos de processamento digital de imagens e projeto em *hardware* de sistemas digitais, áreas nas quais as técnicas de aprendizado de máquina alvos deste estudo serão aplicadas.

2.2 Processamento digital de imagens

A visão é um dos mais importantes, senão o mais importante, sentido humano. É com base nesse sentido que tomamos a maior parte de nossas decisões. Essas decisões passam desde uma simples mudança de direção, enquanto

caminhamos, para desviar de um obstáculo, à escolha de um parceiro que nos agrade visualmente. Devido à importância desse sentido, é da natureza humana querer registrar aquilo que visualizamos. Essa necessidade é observada desde o princípio da humanidade, com o registro de desenhos em pedras e cavernas de civilizações primitivas, passando pela pintura de retratos do período renascentista até a invenção das câmeras fotográficas que data do século XIX.

Processamento digital de imagens surge no âmbito da necessidade de promover alterações em uma imagem, como um conjunto de métodos que visam melhorá-la para uma melhor interpretação humana ou de um sistema computacional. Outro objetivo é a identificação e extração de características de imagens para serem comparadas ou adicionadas a uma base de conhecimento. Segundo Gonzales e Woods [Gonzales e Woods, 2002], uma das primeiras aplicações de processamento digital de imagens ocorreu no início da década de 1920 na indústria do jornalismo impresso. Imagens eram codificadas por um equipamento especializado e transmitidas por um cabo submarino entre Londres e Nova Iorque. O tempo de envio foi reduzido de mais de uma semana para cerca de três horas, mas um dos problemas encontrados era a qualidade da recomposição da imagem no destino.

A partir da década de 1960, com o surgimento dos circuitos integrados (CI), dos sistemas operacionais e da melhora na capacidade de armazenamento de dados, cuja conseqüência foi uma grande evolução dos computadores digitais, o processamento digital de imagens expandiu-se muito. Na época, foram utilizados pelo programa espacial americano métodos para reduzir as distorções inerentes das câmeras de televisão a bordo. A evolução dessas técnicas levou o processamento de imagens para diversas áreas como a indústria farmacêutica para detecção de pastilhas quebradas durante a produção [Oprea et al, 2008], a medicina para, dentre outras aplicações, a análise topológica da pele [Suprijanto Ayu et al, 2009] e a indústria automobilística, na detecção da distância entre veículos [Yuhui, 2010].

2.2.1 Etapas do processamento digital de imagens

Quando se desenvolve uma aplicação baseada no processamento digital de imagens, existe uma série de etapas fundamentais para se partir do domínio do

problema e desenvolver uma aplicação para resolvê-lo [Bastos, 2008]. Na Figura 2.1 estão ilustradas estas etapas e em seguida está descrito o que cada uma representa.

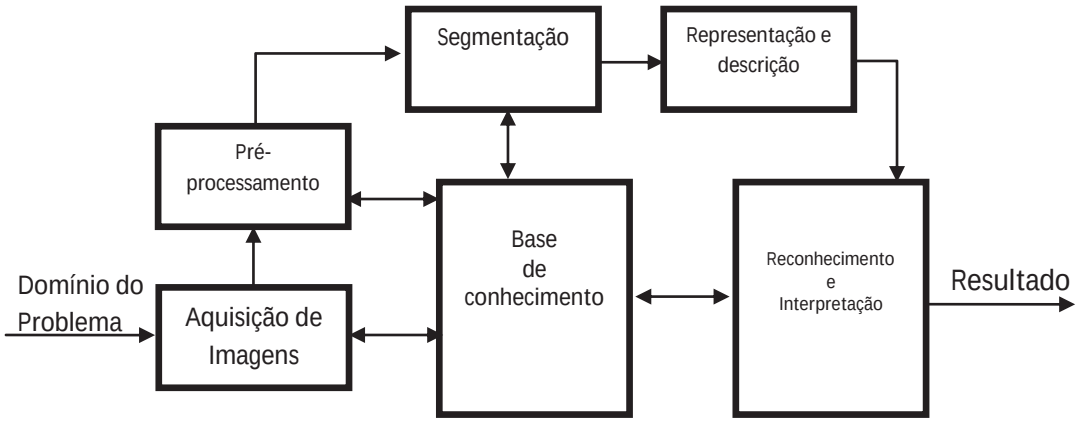


Figura 2.1: Passos fundamentais em visão computacional[Bastos, 2008].

- Aquisição de imagens: a primeira etapa para o processamento de uma imagem é sua aquisição. Esse processo consiste em obter uma representação visual digital o mais fiel possível de um objeto ou cenário real e que, ao mesmo tempo possibilite o seu processamento. Isso pode ser feito através de uma câmera de vídeo, uma máquina fotográfica digital ou um digitalizador de imagens.

- Pré-processamento: corresponde a uma primeira aplicação de métodos de processamento digital de imagens visando à redução de ruídos, à correção de distorções e ao realce de características que possam ser importantes para a aplicação fim. Caso esta aplicação seja a própria interpretação humana, a imagem pré-processada já poderá ser submetida a uma comparação com uma base de conhecimento, que neste caso seria o conhecimento da própria pessoa que a analisará.

- Segmentação: nesta etapa a imagem é analisada para agrupar objetos com características semelhantes ou que sejam significativos para a aplicação em questão. Esta etapa também pode simplesmente realizar a separação dos elementos que constituem a imagem. As técnicas de segmentação de imagens

normalmente são baseadas na identificação de regiões ou fronteiras entre os objetos.

- Representação e descrição: esta etapa visa encontrar maneiras de descrever ou representar os objetos identificados durante a segmentação da imagem. Dessa forma é possível realizar uma quantização desses elementos. Isso abre espaço para a comparação entre imagens ou para a classificação da imagem processada seguindo as características identificadas.

- Reconhecimento e interpretação: consiste em classificar os objetos descritos na etapa anterior de forma que sejam encontrados padrões. É a etapa onde é mais comum a utilização de técnicas de aprendizado de máquina.

A partir das etapas descritas, é possível dividir os métodos de processamento digital de imagens em três classes: processamento de baixo nível, processamento de nível médio e processamento de alto nível. O processamento de baixo nível é aquele realizado no pré-processamento da imagem. Processamento de nível médio é a etapa de segmentação e separação dos objetos da imagem. Por fim, o processamento de alto nível é aquele em que é realizado o reconhecimento e a interpretação da imagem. Não existe um consenso para a classificação dos métodos de representação e descrição, podendo ser adotados como de nível médio ou alto. Alguns métodos de cada classe serão apresentados no decorrer desta seção.

2.2.2 Imagem

Segundo Albuquerque [Albuquerque e Albuquerque, 2000], uma imagem é um suporte para troca de informações. Matematicamente, uma imagem é definida com uma função bidimensional *f(x,y)*, onde x e y são as coordenadas de um *pixel* (*picture element)* da imagem. O valor de *f(x,y)* corresponde ao nível de cinza ou à intensidade do brilho naquele *pixel*. O processo de assimilação de uma imagem real captada por uma máquina fotográfica digital ou por algum dispositivo digitalizador, como um *scanner*, é chamado de discretização. Nesse processo os *pixels* são organizados na forma matricial, e a eles é associado um número que representa o nível de cor ou intensidade do brilho. A ordem da matriz formada pelos *pixels*

amostrados é chamada de resolução espacial da imagem e a quantidade de níveis de cores ou de intensidade do brilho de resolução de intensidade. Quanto maior forem as resoluções espaciais e de intensidade da imagem, maior detalhamento será visto nela e um espaço maior será requerido para seu armazenamento [Gonzales e Woods, 2002].

A organização matricial de uma imagem pode ser vista na Figura 2.2. É possível notar que a orientação dos eixos é diferente da utilizada normalmente para representar o plano cartesiano. A origem é localizada na parte superior esquerda e o eixo das abscissas fica na vertical, orientado de cima para baixo, enquanto o eixo das ordenadas fica na horizontal, orientado da esquerda para direita, como se fosse o plano cartesiano girado 90º no sentido horário. São dispostas M linhas e N colunas cujos valores inteiros nos intervalos [0,M-1] e [0,N-1], respectivamente. Isso totaliza *m x n* pontos, cada qual com um valor de *f(x,y)* associado [Gonzales e Woods, 2002].

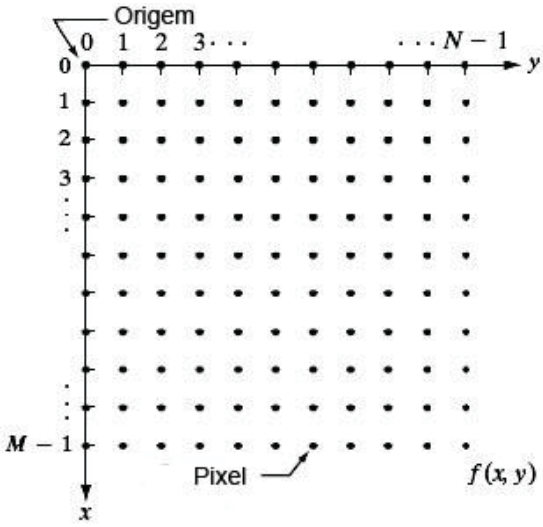


Figura 2.2: Organização matricial de uma imagem [Gonzales e Woods, 2002].

2.2.3 Domínio espacial e pré-processamento de imagens

Uma imagem representada por uma matriz obtida pelo método de discretização apresentado na subseção anterior é dita uma imagem no domínio

espacial. Outra maneira de representá-la seria no domínio de frequências, em que cada coordenada *(x,y)* contém o valor correspondente ao comprimento de onda refletido naquele ponto amostrado. As técnicas de processamento digital de imagens que trabalham no domínio espacial realizam manipulações com os *pixels* propriamente ditos, enquanto a principal abordagem ao se trabalhar no domínio de frequências passa pela aplicação da transformada de Fourier.

Operações no domínio espacial, como indicado anteriormente, são realizadas diretamente sobre os pixels e o seu processo pode ser descrito pela Equação 2.1 onde *f(x,y)* corresponde a uma imagem de entrada, *g(x,y)* é a imagem processada e *T* é um operador em *f* definido sobre alguma vizinhança do *pixel (x,y)*.

$$g(x,y) = T[f(x,y)] \qquad \text{(Equação 2.1)}$$

Uma operação para *T* pode ser a média dos valores *(x,y)* de k imagens para promover a redução de ruídos, mas a principal abordagem utilizada é a definição de uma vizinhança do *pixel (x,y)*, estabelecendo uma máscara quadrada ou retangular cuja posição central, no caso de ordem ímpar, ou a primeira posição, no caso de ordem par, fica sobre o próprio *pixel (x,y)*. Essa máscara é deslocada do primeiro até o último *pixel* da imagem realizando uma operação definida apenas com os pixels cobertos por ela. Alguns exemplos desse tipo de operação, muito usada no pré-processamento da imagem, são as filtragens por média e por mediana. Esses métodos muito comuns são utilizados para redução de ruídos presentes na imagem, como pode ser observado na Figura 2.3, onde uma imagem original (2.3a) foi alterada para apresentar muitos ruídos (2.3b) e tratada com o filtro da média (2.3c) e pelo filtro da mediana (2.3d).

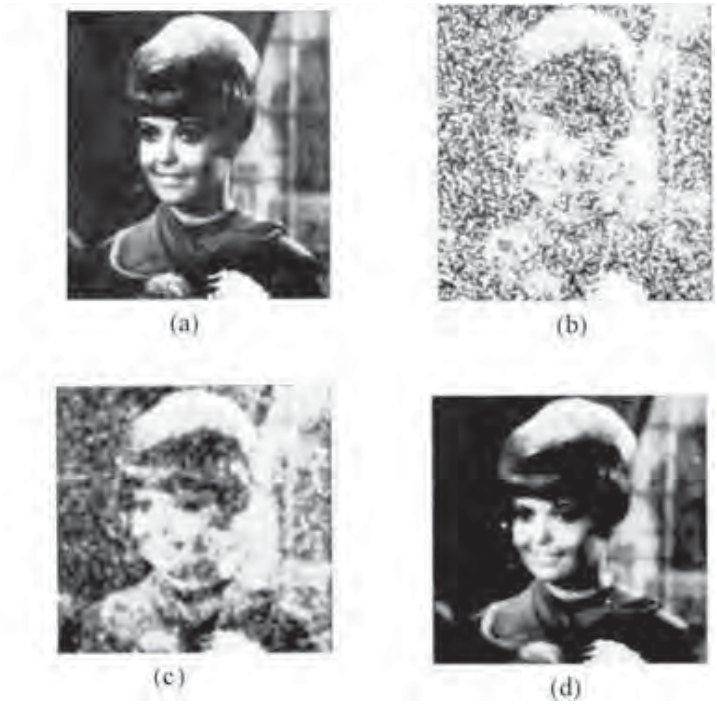


Figura 2.3: (a) Imagem original (b) Imagem com ruídos (c) Filtragem pela média (d) Filtragem pela mediana.

A filtragem pela técnica da média utiliza uma máscara onde o *pixel* localizado sob o centro da máscara, caso ela possua um número ímpar de linhas e colunas, ou o *pixel* sob a primeira posição da máscara, tem seu valor substituído pela média dos valores do *pixel* contidos na máscara. Esta operação é repetida para todos os *pixels* da imagem em que a máscara pode ser aplicada. De maneira similar ao filtro da média, a filtragem por mediana também percorre a imagem utilizando uma máscara. A diferença é que, neste caso, o *pixel* será substituído pelo valor do elemento central de um vetor que contém os *pixels* da máscara ordenados por método de ordenação qualquer. Ou seja, o novo valor será a mediana dos valores dos *pixels* sob a máscara. Em ambas as técnicas é observada uma redução do ruído presente na imagem com a desvantagem de ocorrer uma perda na definição das fronteiras dos objetos da imagem. Portanto, quando a aplicação, na qual se fizer uso das imagens pré-processadas depender de uma definição das fronteiras entre os objetos, o uso de filtragem por média ou mediana deve ser feito com cuidado.

Outra maneira com a qual se pode utilizar um operador de domínio espacial *T* é a operação ponto a ponto, em que a operação utiliza um único *pixel*. Nesse caso, o operador *T* é chamado de operador de transformação do nível de cinza (ou

intensidade do brilho). Esse tipo de operador é utilizado para promover alterações no contraste da imagem. Para se conseguir uma imagem mais clara, basta somar um determinado valor ao nível de cinza de cada *pixel* da imagem. O processo para escurecer a imagem é exatamente o oposto, subtraindo-se um valor do nível de cinza dos *pixels* a imagem. Ainda existe a possibilidade de mais de um operador deste tipo ser utilizado, somando-se um valor para *pixels* com nível de cinza acima de um limiar e subtraindo-se, caso contrário. Com isso é possível aumentar o contraste entre os objetos presentes na imagem.

2.2.4 Segmentação

A etapa de segmentação é utilizada em análise de imagems quando há necessidade de identificar diferentes objetos contidos em uma cena [Gonzales e Woods, 2002]. Os algoritmos de segmentação costumam basear-se em duas propriedades de imagens quando em níveis de cinza, quais sejam, descontinuidade e similaridade. A propriedade de descontinuidade propicia um critério para separar os objetos da imagem baseando-se em mudanças significativas do nível de cinza entre *pixels* vizinhos. Esse critério é muito útil em algoritmos para detecção de pontos isolados, linhas e bordas ou fronteiras dos componentes da imagem. As abordagens da propriedade de similaridade são feitas na segmentação da imagem em regiões consideradas similares. Isso é feito pela definição um conjunto de critérios, como o tipo de textura ou formato do objeto.

Uma das dificuldades do processo de segmentação de uma imagem é a escolha de qual método utilizar para atingir o objetivo desejado. Isto ocorre porque esta etapa é muito dependente da aplicação alvo. Existem diversos métodos para executar operações com objetivos semelhantes, mas que apresentam resultados distintos dependendo da imagem de entrada. A maioria das técnicas de segmentação, especialmente quando buscamos detectar descontinuidades, trabalha com a convolução de uma máscara com a imagem. A convolução de uma máscara é um processo que se resume a três operações: deslocar a máscara até o local desejado, multiplicar os coeficientes presentes na máscara pelos valores dos *pixels* correspondentes na imagem, somar os resultados obtidos nas multiplicações. O resultado da soma é armazenado na imagem de saída de acordo com a seguinte regra.

Se a máscara aplicada for de ordem par, o resultado é armazenado no lugar sob a primeira posição da máscara. Se a máscara possuir ordem ímpar, o resultado é armazenado no lugar sob a posição central da máscara.

A detecção de pontos isolados em uma imagem é uma das técnicas de segmentação na qual se utiliza a convolução de uma máscara conforme exemplificado a seguir. Na Figura 2.4 temos uma máscara para detecção de pontos isolados. O processo desse método é simples. Após a convolução da máscara, os valores dos *pixels* da imagem resultante são comparados a um limiar L estabelecido previamente. Se o valor do *pixel* da imagem resultante for inferior ao limiar, o *pixel* é transformado em preto. Caso contrário, ele é transformado em branco.

−1	−1	−1
−1	8	−1
−1	−1	−1

Figura 2.4: Máscara para detecção de pontos isolados [Gonzales e Woods, 2002].

A detecção de linhas é um processo um pouco mais complexo. Apesar de suas técnicas utilizarem o mesmo princípio da convolução de máscaras, são necessárias máscaras distintas dependendo da orientação da linha que se deseja detectar. Na Figura 2.5 é possível observar quatro máscaras para detecção de linhas, cada uma com uma orientação.

−1	−1	−1	−1	−1	2	−1	2	−1	2	−1	−1
2	2	2	−1	2	−1	−1	2	−1	−1	2	−1
−1	−1	−1	2	−1	−1	−1	2	−1	−1	−1	2

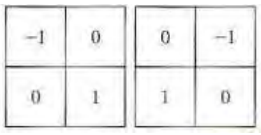
(a) (b) (c) (d)

Figura 2.5: Máscaras para detecção de linhas
(a) Horizontal (b) +45º (c) Vertical (d) -45º [Gonzales e Woods, 2002].

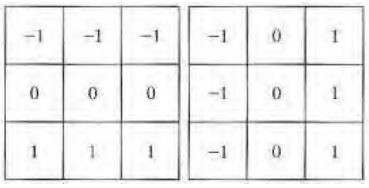
Apesar da importância da detecção de pontos e linhas em uma imagem, nenhuma delas é tão importante quanto a detecção de bordas. Por meio da detecção de bordas é possível extrair características fundamentais dos objetos que compõem a imagem, como, por exemplo, área, perímetro e forma do objeto [Gonzales e Woods, 2002].

Para identificar a borda de um objeto, o fator mais importante é a variação dos níveis de cinza em uma mesma região. Uma borda é o limite entre duas regiões com propriedades relativamente distintas de nível de cinza. É possível dizer que um *pixel* faz parte de uma borda se ele está no limite de uma mudança abrupta de nível de cinza e conectado com outros *pixels* com a mesma característica em uma determinada direção.

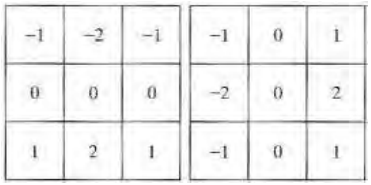
Existem diversas técnicas para detecção de bordas em uma imagem. Estas técnicas envolvem os chamados operadores gradientes. Os operadores gradientes buscam, por meio da sua convolução com a imagem, realçar as descontinuidades dos níveis de cinza da imagem. Os operadores de gradiente básicos para realçar bordas são os de Roberts, Prewitt e Sobel, apresentados na Figura 2.6.



(a)



(b)



(c)

Figura 2.6: Operadores de gradiente (a) Roberts (b) Prewitt (c) Sobel [Gonzales e Woods, 2002]

2.3 Dispositivos lógico-programáveis

Com a corrida espacial iniciada na década de 1960, houve um avanço tecnológico em diversas áreas. Uma das que mais evoluíram por este motivo foi a área de sistemas digitais. Até então, esses sistemas eram projetados e construídos a partir de componentes discretos, na maioria das vezes baseados em transistores e resistores. A invenção dos CIs nos laboratórios da *Texas Instruments* mudou este panorama, possibilitando a integração dos componentes do circuito em uma única pastilha de silício, reduzindo o tamanho e o consumo de energia. Contudo, os CIs apresentavam a desvantagem de executar apenas as funcionalidades predefinidas pelo fabricante. Por este motivo, para criar lógicas definidas pelos próprios usuários, era necessária a junção de vários CIs. Isso levava ao aumento do tamanho do circuito e do consumo de energia, reduzindo as vantagens originais dos CIs. Esta foi a motivação para o surgimento de dispositivos cuja funcionalidade pudesse ser definida pelo usuário. Na Figura 2.7 ilustram-se as opções disponíveis para o desenvolvimento de sistemas digitais.

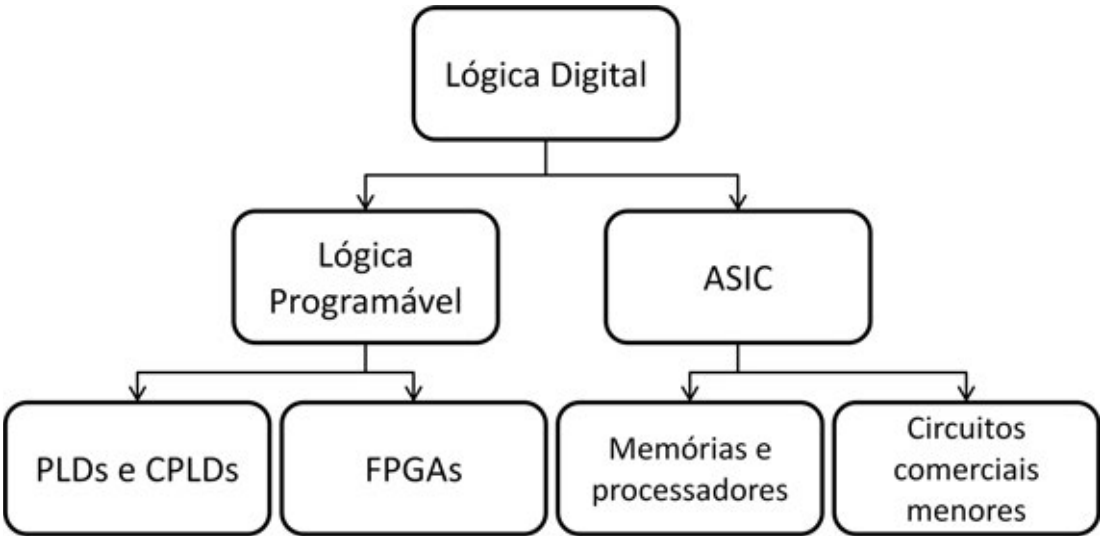


Figura 2.7: Opções para desenvolvimento de sistemas digitais.

Os dispositivos ASIC (*Applicantion Specific Integrated Circuit*) são implementações de sistemas digitais com o desempenho mais avançado e com menor consumo de energia. Esses sistemas, uma vez projetados, podem ser programados pelos usuários, mas só executarão as funcionalidades incluídas em seu projeto, sem a

possibilidade de atualizá-los. O custo e a complexidade de desenvolvimento para estes dispositivos é bastante elevado, mas são compensados quando produzidos em larga escala [Weste e Harris, 2004].

Já os dispositivos de lógica programável, cujos principais representantes são os PLD (*Programmable Logic Device*), CPLD (*Complex Programmable Logic Device*) e FPGA (*Field Programmable Gate Array*) funcionam como circuitos personalizáveis e permitem ao usuário implementar a funcionalidade do circuito. Além disso, o custo de desenvolvimento é muito inferior ao de um ASIC, com a mesma funcionalidade, se produzido em escala menor. A desvantagem destes dispositivos é não apresentarem um desempenho ótimo, se comparados com circuitos de aplicação específica.

Algo em comum ao desenvolvimento de sistemas digitais é o aumento da complexidade de projeto de circuitos a cada ano. Para evitar que estes circuitos precisem ser projetados desenhando cada porta lógica, surgiram as ferramentas EDA (*Eletronic Design Automation*). Uma das principais tarefas dessas ferramentas é realizar o mapeamento de uma linguagem de descrição de *hardware* (*Hardware Description Language* – HDL) para uma tecnologia de fabricação ou um dispositivo programável, como um FPGA, economizando muito tempo de projeto. No restante desta seção serão abordados os dispositivos de lógica programável e a HDL Verilog.

2.3.1 PLDs e CPLDs

Os PLDs, como o próprio nome sugere, são dispositivos cuja arquitetura permite que sejam programados de maneira personalizada. Sua estrutura é composta por portas lógicas e chaves de interconexão que podem ser ligadas de maneira a formar qualquer tipo de lógica booleana. O primeiro dispositivo deste tipo a ser lançado no mercado foi o PLA (*Programmable Logic Array*). Ele consiste em dois arranjos distintos de portas lógicas com as conexões de entrada programáveis, um com portas ‘E’ e outro com portas ‘OU’, como pode ser visto na Figura 2.8. As entradas do arranjo ‘OU’ podem ser ligadas de forma a produzir a soma lógica de quaisquer saídas do arranjo ‘E’. Para isso, basta configurar as chaves programáveis definindo quais entradas devem passar por uma porta ‘E’ qualquer e, da mesma forma, definir quais portas ‘E’ devem ter suas saídas somadas. Os PLAs são

adequados para implementar circuitos combinacionais cuja lógica componha uma soma de produtos.

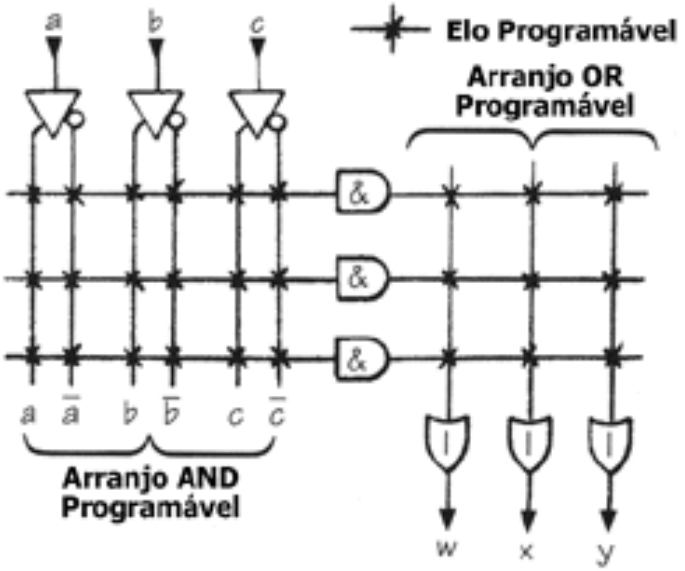


Figura 2.8: Esquema de um PLA [Leal, 1998].

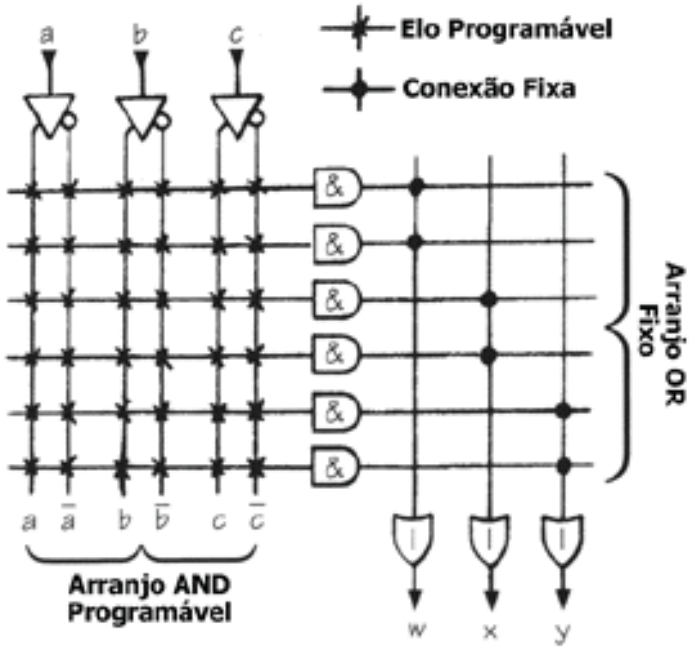


Figura 2.9: Esquema de um PAL [Leal, 1998].

Outro tipo de PLD é o PAL (*Programmable Array Logic*), que surgiu para suprir algumas deficiências no desempenho dos PLA. Para tanto, os PALs possuem um único arranjo de portas ‘E’ programáveis. As portas ‘OU’ dos PALs possuem interconexões fixas, como está ilustrado na figura 2.9. Além disso, alguns PALs

possuem *flip-flops* ligados às saídas, permitindo o desenvolvimento de circuitos com lógicas sequenciais.

Os circuitos implementados nos dispositivos do tipo PLA e PAL são limitados pelo baixo número de entradas e saídas destes dispositivos. Para possibilitar a implementação de circuitos com um número maior de entradas e saídas, foi desenvolvido um dispositivo mais sofisticado, conhecido por CPLD. Este dispositivo é constituído por vários blocos mais simples, chamados de SPLD (*Simple Programmable Logic Device*), similares aos circuitos do tipo PLA ou PAL. Todos os blocos são interconectados através de chaves programáveis e conectados a blocos de entrada e saída, os quais estão ligados aos pinos de entrada e saída do dispositivo. A estrutura de um CPLD é apresentada pela Figura 2.10.

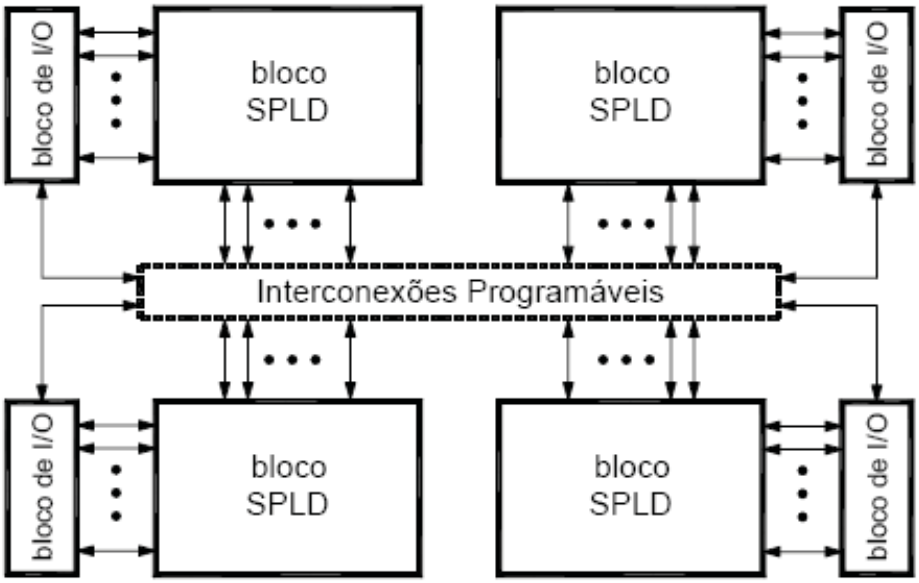


Figura 2.10: Estrutura de um CPLD.

2.3.2 FPGAs

Utilizado para aplicações cujo mercado demanda um menor volume de produção, os FPGAs apresentam como vantagem um baixo custo de desenvolvimento quando comparado ao ASIC. Além disso, eles trazem a possibilidade da realização de modificações no *hardware* devido a sua propriedade de reconfiguração. Isso facilita um desenvolvimento iterativo com certa facilidade para correção de erros e adição de novas funcionalidades de *hardware* [Silva, 2006].

A arquitetura de um FPGA é composta basicamente por três elementos, os blocos lógicos, as chaves de interconexão e os blocos de entrada e saída. Sua organização é feita da seguinte maneira: os blocos lógicos são dispostos em uma rede de interconexão composta pelas chaves de interconexão, que são programáveis. Toda essa estrutura é cercada pelos blocos de entrada e saída [Martins et al, 2003]. Na figura 2.11 encontra-se um esquema da arquitetura interna de um FPGA.

A arquitetura dos blocos lógicos pode mudar de acordo com a família ou o fabricante. Em geral são compostos por pontos de entrada conectados às LUTs (*Lookup Table*), multiplexadores – para direcionar o fluxo interno de sinais – e registradores. As LUTs contêm funções combinacionais e os registradores normalmente são *flip-flops* que estão ligados às saídas ou realimentam as entradas [Martins et al, 2003].

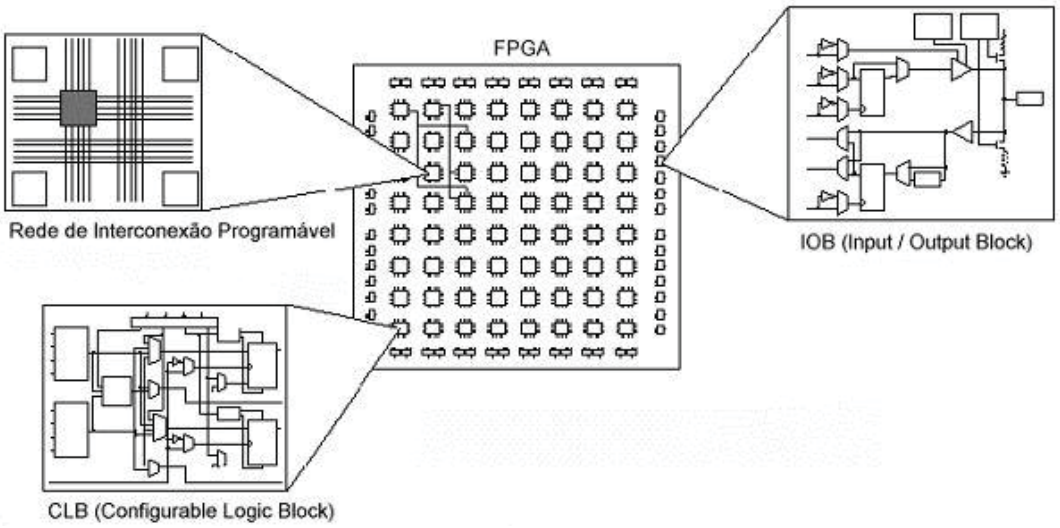


Figura 2.11: Arquitetura interna de um FPGA [Martins et al, 2003].

As chaves de interconexão são programáveis e produzem circuitos de complexidade arbitrária. Elas ligam as entradas e saídas dos blocos lógicos aos blocos de entrada e saída. São responsáveis pelo roteamento entre as linhas e colunas dos blocos lógicos [Martins et al, 2003].

Os elementos reconfiguráveis de um FPGA não são implementados de uma mesma maneira. Eles podem ser como uma simples LUT de 3 bits ou uma unidade lógico-aritmética (ULA). Essa diferença de complexidade é chamada de

granularidade. Dispositivos mais simples, com grão menor, são recomendados para aplicações que exigem um trabalho bit a bit. Já os dispositivos mais complexos, aqueles com grão maior, são mais adequadas para as aplicações que envolvem uma computação mais complexa, como manipulação de imagens [Martins et al, 2003].

Para o desenvolvimento das aplicações em FPGA são utilizadas ferramentas automatizadas de projeto eletrônico, chamadas de ferramentas EDA. O objetivo do uso dessas ferramentas é a simplificação e aceleração do ciclo de desenvolvimento. Por meio do uso de linguagens de descrição de *hardware* é possível descrever e simular como os circuitos funcionam. Entre as linguagens de descrição de *hardware* mais usadas estão a VHDL (*Very High Speed Integrated Circuits Hardware Description Language*) e a Verilog [Silva, 2006]. A seguir está um breve histórico de Verilog, uma vez que foi a linguagem adotada para implementação do projeto.

2.3.3 *Verilog*

A história da linguagem *Verilog* tem início na década de 1980 quando a empresa *Gateway Design Automation* desenvolveu um simulador de lógica chamado *Verilog-XT* e com ele a linguagem. Em 1989, outra empresa, *Cadence Design Systems*, comprou a *Gateway* e com ela os direitos sobre a linguagem e o simulador. Em 1990 a linguagem foi colocada em domínio público com o objetivo de torná-la um padrão não proprietário. Em 1995 a linguagem passou pelos procedimentos de padronização do IEEE (*Institute of Electrical and Electronics Engineers*) recebendo o número 1364-1995 e em 2001 foi publicada uma revisão com o número 1364-2001. Hoje, a linguagem *Verilog* é mantida pela organização sem fins lucrativos *Accellera* [Rosa, 2008].

Em *Verilog* existem três modelos de descrição: comportamental, RTL (*Register Transfer Level*) e estrutural. O modelo de descrição comportamental é utilizado para descrições em nível de sistema, bem como na elaboração de ambientes de verificação para simular projetos em desenvolvimento. O modelo em RTL visa descrever o circuito em nível de transferência de dados entre os registradores, abstraindo detalhes que tornariam inviáveis o projeto de circuitos de grande complexidade. Esses detalhes são inferidos pelas ferramentas EDA utilizadas durante o projeto. Elas, dentre outras tarefas, fazem a conversão do código em RTL para o

nível estrutural, que é equivalente à descrição do circuito em nível de portas lógicas [Rosa, 2008], e posteriormente para os níveis eletrônicos e geométricos, seguindo métricas contidas em bibliotecas específicas de cada tecnologia de fabricação.

2.4 Considerações finais

A partir das informações levantadas neste capítulo é possível observar que as técnicas de processamento digital de imagens presentes na literatura são soluções eficientes para realizar melhorias em imagens para utilização em diversas aplicações. Isso facilita a identificação e extração de informações relevantes para o uso em sistemas como os de aprendizado de máquina. Além disso, as diversas opções de implementação em *hardware* fornecem a flexibilidade necessária para o desenvolvimento destes sistemas. Portanto, o desenvolvimento de tais sistemas fica condicionado à escolha de uma boa abordagem para o uso das técnicas de processamento digital de imagens e do *hardware* escolhido para sua implementação.

Capítulo 3 – Fundamentação teórica: AM

3.1 Considerações iniciais

Reconhecer e interpretar informações é algo tão natural para os seres humanos que o fazemos sem ao menos notar que estamos fazendo. Contudo, modelar essas tarefas de forma que possamos descrevê-las e implementá-las em sistemas computacionais não é trivial. Uma das abordagens para este problema é o uso de aprendizado de máquina. As técnicas de AM utilizam um princípio de inferência denominado de indução com o objetivo de obter conclusões genéricas a partir de um conjunto reduzido de exemplos. Existem dois tipos de aprendizado de máquina, o supervisionado e o não-supervisionado [Lorena e De Carvalho, 2007].

O aprendizado supervisionado é caracterizado pela presença de um professor externo que fornece para cada conjunto de dados de entrada, a saída esperada. Com base nos dados recebidos o algoritmo extrai uma representação do conhecimento com o objetivo de produzir as saídas corretas para dados de entrada não apresentados anteriormente. O aprendizado não-supervisionado não conta com a figura do professor e seu funcionamento é dado pelo agrupamento dos dados de entrada com

base em alguma medida. Enquanto o aprendizado supervisionado é usado na classificação de dados, o aprendizado não supervisionado é aplicado na procura por padrões ainda não conhecidos.

Neste capítulo abordamos AM supervisionado com enfoque em SVM e RNA. O conjunto de treinamento utilizado no aprendizado supervisionado é composto por pares (x_i,y_i) onde x_i representa uma amostra do domínio de dados de entrada e y_i o seu rótulo, ou classe na qual a amostra deve ser classificada. A partir deste conjunto, deve-se obter um classificador ou função classificadora que rotule precisamente novos dados de entrada. O processo para obter esta função é chamado de treinamento.

Cada dado ou caso de entrada para um método de AM pode ser representado por um vetor composto pelas características do que se deseja classificar. Estas características podem ser dos tipos nominal ou contínuo. No primeiro caso, o atributo é uma categoria ou denominação, como a cor de uma determinada folha. Já um atributo contínuo expressa uma característica em que se pode definir uma ordem linear dos valores possíveis, como a largura das folhas citadas anteriormente.

Na Figura 3.1 está ilustrado o processo geral da aplicação das técnicas de AM. A partir de um conjunto de treinamento com n elementos de m atributos de classe y_i, uma técnica de AM deve obter então uma função classificadora que consegue distribuir rótulos para um novo conjunto de dados. Esta função deve ser robusta o suficiente para lidar com possíveis ruídos no conjunto de treinamento e também possuir uma boa generalização para conseguir classificar dados diferentes do conjunto de treinamento, embora do mesmo domínio.

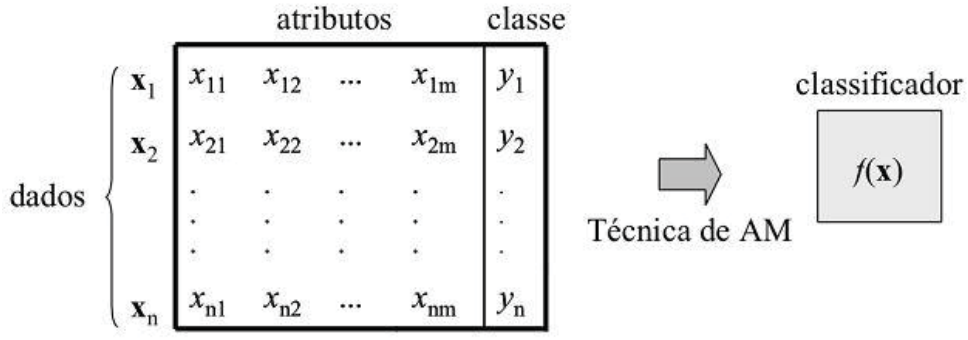


Figura 3.1: Obtenção de um classificador [Lorena e De Carvalho, 2007].

A definição de uma função classificadora precisa levar em conta essencialmente dois parâmetros para obtenção de bons resultados. O primeiro deles é o seu desempenho em relação ao conjunto de treinamento. O segundo é sua complexidade. Na Figura 3.2 está ilustrado o desempenho de três funções classificadoras para o mesmo conjunto de dados. Na Figura 3.2a está uma função que classifica corretamente todos os elementos do conjunto de treinamento, incluindo ruídos. Este é um exemplo de uma função muito ajustada ao conjunto de treinamento, o que a torna pouco útil para outros conjuntos de dados. Já, na Figura 3.2c, a função comete erros até mesmo para casos simples, o que significa que é uma função pouco ajustada ao conjunto de treinamento. Por fim, na Figura 3.2b temos uma função que trata bem quase todos os casos, ignorando os ruídos presentes no conjunto de treinamento e tornando-a uma boa função classificadora.

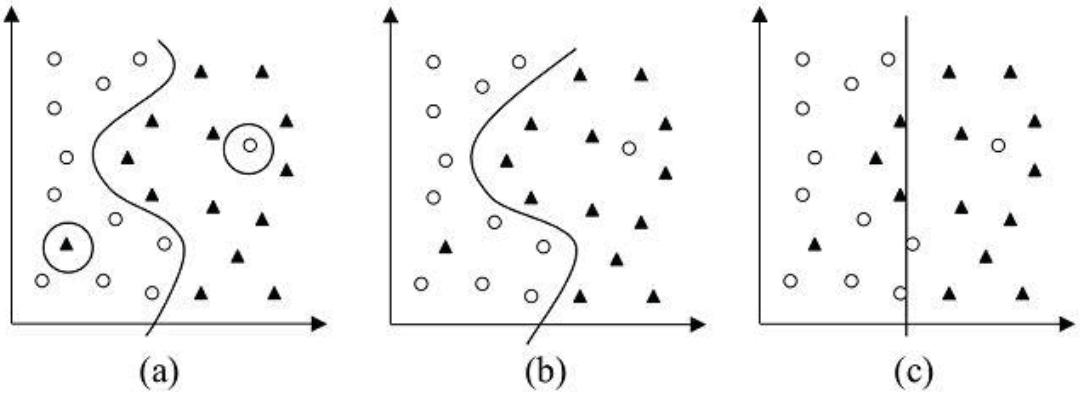


Figura 3.2: Funções classificadoras (a) muito ajustada (b) bem ajustada (c) sub-ajustada [Lorena e De Carvalho, 2007].

3.2 Máquinas de vetores de suporte

Uma das técnicas de aprendizado de máquina utilizadas na obtenção de classificadores de padrões são as SVMs. Diferente de uma RNA, as SVMs não são fundamentadas biologicamente [Hernandez e Strum, 2009]. As SVMs são embasadas pela teoria de aprendizado estatístico e vêm ganhando espaço pelos resultados comparáveis ou superiores às RNA, em alguns casos [Lorena e De Carvalho, 2007]. Este crescimento é evidenciado pelos diversos trabalhos recentes na área, como o de Zhi-ping e Ding [Zhi-ping e Ding, 2010] que utiliza lógica nebulosa combinada com

SVMs para detectar intrusos em uma rede de computadores. Tang [Tang et al, 2004] propõe um modelo de aprendizado combinando SVMs com computação granular para melhorar a classificação de dados médicos e biológicos.

As SVMs são divididas em duas categorias de acordo com a capacidade de classificação dos dados do domínio do problema. Quando esses dados são linearmente separáveis, são utilizadas SVMs ditas lineares. Caso contrário, é feita uma extensão das SVMs lineares para obtenção de fronteiras de separação não lineares, com as ditas SVMs não lineares.

3.2.1 SVMs lineares

Uma SVM linear define fronteiras lineares a partir de dados linearmente separáveis. A sua formulação é dada por um conjunto de treinamento *T* de *n* dados x_i ∈ X e seus respectivos rótulos y_i ∈ Y em que X constitui o espaço dos dados e Y = {-1, +1}. *T* é linearmente separável se os dados das classes -1 e +1 podem ser separados por um hiperplano [Lorena e De Carvalho, 2007]. Um hiperplano é definido por uma função como a da Equação 3.1, na qual w · x é o produto escalar entre os vetores *x* e *w*, *w* ∈ X é o vetor normal ao hiperplano e b é a distância entre o hiperplano e a origem, com b real.

$$f(x) = w \cdot x + b = 0 \qquad \text{(Equação 3.1)}$$

Por meio da Equação 3.1 o espaço dos dados pode ser dividido em duas partes, com $w \cdot x + b > 0$ e $w \cdot x + b < 0$. A partir desta divisão é possível determinar uma função sinal que classifica os dados como mostra a Equação 3.2.

$$g(x) = sgn\, f(x) = \begin{cases} -1, se\ w \cdot x + b < 0 \\ +1, se\ w \cdot x + b > 0 \end{cases} \qquad \text{(Equação 3.2)}$$

A partir de $f(x)$ infinitos hiperplanos equivalentes podem ser definidos, por meio da multiplicação de *w* e *b* por uma mesma constante. Contudo, o hiperplano canônico em relação ao conjunto de treinamento *T* é definido de forma que os elementos de *T* mais próximos do hiperplano satisfaçam a Equação 3.3.

$$|w \cdot x_i + b| = 1 \qquad \text{(Equação 3.3)}$$

Para rotular os dados de entrada segundo Y, como assumido no início desta subseção, a partir da Equação 3.3 os dados de entrada cujos valores para equação do hiperplano canônico forem positivos serão rotulados como $+1$, caso contrário serão rotulados como -1. A Equação 3.4 ilustra esta atribuição e a Equação 3.5 apresenta a mesma ideia de forma resumida.

$$\begin{cases} w \cdot x_i + b \geq +1 \; se \; y_i = +1 \\ w \cdot x_i + b \leq -1 \; se \; y_i = -1 \end{cases} \quad \text{(Equação 3.4)}$$

$$y_i(w \cdot x_i + b) - 1 > 0, \forall (x_i, y_i) \in T \quad \text{(Equação 3.5)}$$

Se considerarmos um ponto x_1 pertencente ao hiperplano H_1: $w \cdot x_i + b = +1$ e outro ponto x_2 pertencente ao hiperplano H_2: $w \cdot x_i + b = -1$, é possível calcular a distância entre os dois hiperplanos H_1 e H_2 projetando $(x_1 - x_2)$ na direção de w. Este valor é dado por $\frac{2}{\|w\|}$, portanto, a distância entre os hiperplanos H_1 e H_2 é de $\frac{1}{\|w\|}$ [Lorena e De Carvalho, 2007]. Para conseguir, então, a maior separação possível dos dados em relação ao hiperplano canônico é preciso minimizar $||w||$. Algebricamente, este problema de otimização pode ser formulado como sendo o de minimização da norma ao quadrado do vetor de peso w ortogonal ao hiperplano canônico [Gonzalez e Chau, 2006]. Ou seja:

$$\text{Minimize } w \text{ e } b \text{ em } \frac{1}{2}||w||^2 \quad \text{(Equação 3.6)}$$

$$\text{Sob as restrições: } y_i \, (w \cdot x_i + b) - 1 > 0, \forall i = 1, \ldots, n \quad \text{(Equação 3.7)}$$

A função a ser minimizada é convexa, o que permite que seja solucionada com a introdução de uma função lagrangeana. Restrições são englobadas à função objetivo junto a parâmetros denominados multiplicadores de Lagrange α_i, como é mostrado na Equação 3.8.

$$L(w, b, \alpha) = \frac{1}{2} \left|\left|w\right|\right|^2 - \sum_{i=1}^{n} \alpha_i (y_i (w \cdot x_i + b) - 1) \quad \text{(Equação 3.8)}$$

Para minimizar a função lagrangeana é preciso maximizar α_i e minimizar w e b. Isto é feito pelo cálculo do gradiente da lagrangeana com relação a w e b. O resultado é dado pelas Equações 3.9 e 3.10.

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{(Equação 3.9)}$$

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \quad \text{(Equação 3.10)}$$

Substituindo essas equações na Equação 3.8 temos um novo problema de otimização, desta vez de maximização. Esse problema é exposto pela Equação 3.11. Esta formulação é chamada de problema dual, enquanto a exposta na Equação 3.6 é denominada problema primal.

$$\text{Maximize } \alpha \text{ em } \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \quad \text{(Equação 3.11)}$$

$$\text{Com as restrições: } \begin{cases} \alpha_i \geq 0, \forall i = 1, \ldots, n \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \end{cases} \quad \text{(Equação 3.12)}$$

O uso de um problema dual é interessante por possuir restrições mais simples e representar o problema em termos de produtos internos, o que é útil para o caso de SVMs não lineares. Além disso, é possível notar que o problema é formulado utilizando apenas o conjunto de dados do treinamento e seus rótulos. Ao obtermos a solução α^* do problema dual, podemos encontrar w^* e b^*, soluções do problema primal. O primeiro é encontrado por meio da Equação 3.10, enquanto o segundo é obtido pela Equação 3.13.

$$\alpha_i^* \, (y_i \, (w^* \cdot x_i + b^*) - 1) = 0, \forall i = 1, \ldots, n \quad \text{(Equação 3.13)}$$

Desta equação temos que α_i^* poderá ser diferente de zero apenas quando os dados utilizados se encontrarem sobre os hiperplanos H_1 e H_2. Em outros casos as condições são satisfeitas apenas com $\alpha_i^* = 0$. Estes dados não são utilizados no cálculo de w^*. Já os dados em que $\alpha_i^* > 0$ são denominados vetores de suporte e são os dados significativos para geração da função de classificação. Então, temos que a partir do conjunto de vetores de suporte (SV), α^* e b^* podemos obter a Equação 3.14, que representa o classificador.

$$g(x) = sgn(f(x)) = sgn\left(\sum_{x_i \in SV} y_i \, \alpha^* x_i \cdot x + b^*\right) \quad \text{(Equação 3.14)}$$

3.2.2 SVMs não lineares

Há muitos casos em que os dados de um problema não são linearmente separáveis, ou sua separação é insatisfatória. Nesses casos, as SVMs lidam com o

problema fazendo um mapeamento do conjunto de treinamento, que pertence ao espaço de entradas, para um espaço de características. A Figura 3.3 ilustra o problema e a solução via mapeamento, onde em (a) vemos o conjunto de dados cuja separação linear não é possível, em (b) temos um exemplo de como seria uma fronteira não linear para este conjunto e em (c) o conjunto é mapeado para o espaço de características, linearmente separáveis.

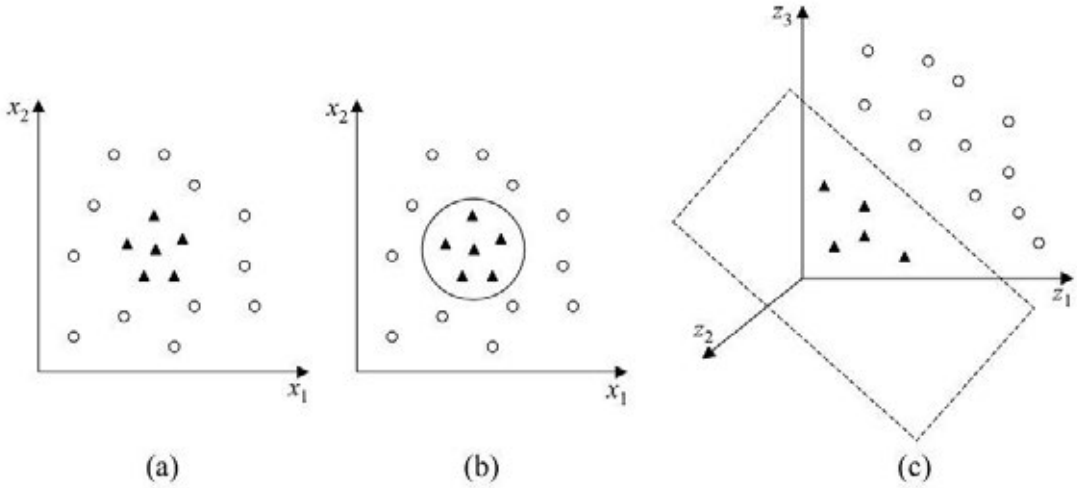


Figura 3.3: (a) Conjunto não separável linearmente (b) Fronteiras não lineares (c) Conjunto mapeado para espaço de características [Lorena e De Carvalho, 2007].

Um mapeamento do espaço de entradas para o espaço de características pode ser da seguinte forma: $\Phi: X \rightarrow \Im$, onde X representa o espaço de entradas e $\Im$ o espaço de características. Uma escolha adequada de Φ permite que o conjunto mapeado possa ser separável. Para isso, duas restrições devem ser respeitadas: a primeira exige que a transformação seja não linear; enquanto a segunda espera que o espaço de característica possua uma dimensão suficientemente alta.

Então, os passos para obtenção de uma função de classificação utilizando uma SVM não linear passa primeiramente pelo mapeamento dos dados de entrada. Em seguida uma SVM linear é aplicada ao conjunto mapeado para que seja encontrado o hiperplano com maior margem de separação, garantindo uma boa generalização. Para realizar o mapeamento Φ é aplicado ao problema dual da subseção anterior, como ilustra a Equação 3.15. Analogamente, é possível encontrar o classificador, como apresentado na Equação 3.16.

Maximize α em $\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\, \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j))$ (Equação 3.15)

$$g\,(x) = sgn\,(f\,(x)) = sgn\left(\sum_{x_i \in SV} y_i\, \alpha^* \Phi(x_i) \cdot \Phi(x) + b^*\right) \quad \text{(Equação 3.16)}$$

Como a dimensão do espaço de características pode ser muito elevada, espera-se que o custo computacional seja igualmente elevado. Porém, é possível notar que a única operação necessária no espaço de características é a multiplicação escalar entre dois dados do mesmo espaço. Para simplificar estas operações são utilizadas funções chamadas *Kernels*, as quais recebem vetores do espaço de entradas e entregam o produto escalar entre eles. É possível, portanto, utilizar *Kernels* sem o conhecimento da função de mapeamento entre os espaços, pois esta seria gerada implicitamente. Esta prática simplifica os cálculos eliminando a necessidade de representar espaços abstratos. Alguns dos *Kernels* mais utilizados são os polinomiais, gaussianos e sigmoidais.

3.3 Redes Neurais Artificiais

As RNAs constituem um método computacional que modela matematicamente um neurônio biológico. Esta modelagem é feita pela elaboração de um sistema com circuitos que simulam o cérebro humano. Com isso, busca-se a capacidade de aprendizado a partir de erros [Haykin, 1999].

Um neurônio biológico, como mostrado na Figura 3.4, possui quatro elementos principais: corpo, dendritos, axônio e terminais sinápticos. O corpo do neurônio, constituído pelo seu núcleo e citoplasma, é o responsável pelo processamento dos impulsos recebidos e pela geração dos impulsos de saída. Os dendritos são ramificações que partem do corpo do neurônio e têm a finalidade de receber impulsos vindos de outros neurônios. O axônio, por sua vez, é um grande prolongamento também partindo do corpo do neurônio e é responsável por levar os impulsos de saída até os terminais sinápticos. Estes, por fim, fazem conexão com os dendritos dos outros neurônios para transmitir os impulsos de saída.

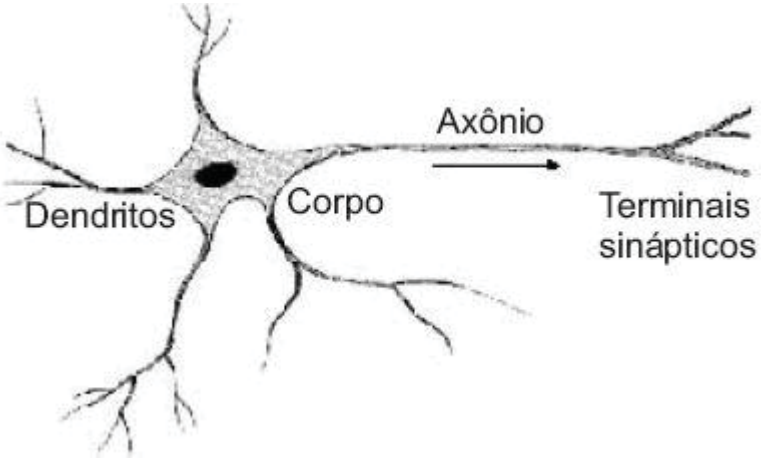


Figura 3.4: Neurônio biológico.

Um modelo artificial de um neurônio é representado na Figura 3.5. As entradas x_1 a x_i e seus respectivos pesos w_1 a w_i representam os dendritos. A junção aditiva e a função de ativação representam o corpo do neurônio e têm a função de processar os "impulsos" de entrada. A saída y_k , por sua vez, faz o papel do axônio, conduzindo o valor processado para a saída.

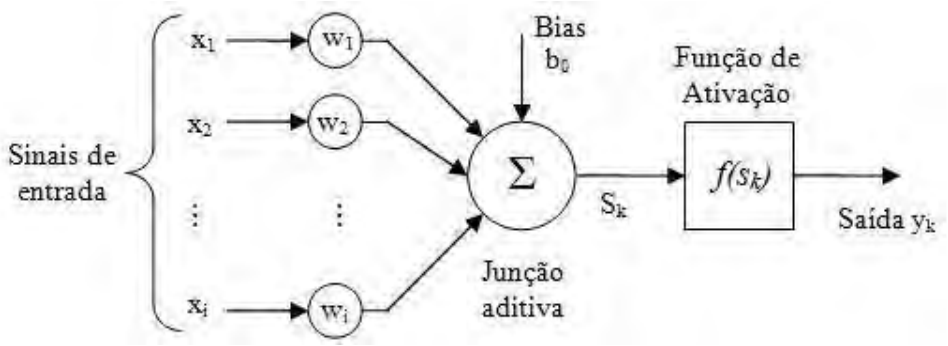


Figura 3.5: Modelo de neurônio artificial [Haykin, 1999].

O funcionamento do neurônio artificial se dá pela soma das entradas x_i ponderadas pelos pesos w_i , resultando em S_k. A este valor é aplicada uma função de ativação que visa limitar os valores possíveis para a saída y_k. Ainda existe um *bias* b_0 aplicado externamente cuja função é aumentar ou diminuir a entrada da função de ativação para ajustar o valor S_k e obter o resultado esperado da função de ativação. Desta forma, o comportamento do neurônio artificial pode ser descrito pelas Equações 3.17 e 3.18.

$$S_k = \sum_{i=0}^{n} x_i \, w_i \; + \; b_0 \qquad \text{(Equação 3.17)}$$

$$y_k = f(S_k) \qquad \text{(Equação 3.18)}$$

A função de ativação limita o domínio da saída y_k, para obter valores conhecidos. Alguns dos principais tipos de função de ativação são: função linear, função rampa, função limiar ou de degrau e função sigmoidal. A função linear opera pela multiplicação de um coeficiente linear real pelo valor de S_k. A função rampa limita os valores de y_k ao intervalo de -1 e $+1$. Já a função limiar realiza uma binarização da saída por meio da definição de um limiar. Por fim, a função de ativação sigmoidal é uma das mais utilizadas obtida por meio da aplicação de uma função exponencial, obtendo como resultado uma saída contínua com variações suaves [Haykin, 1999].

3.3.1 Arquiteturas de RNAs

As arquiteturas de RNAs podem ser organizadas segundo três características. A primeira delas é o número de camadas da rede, isto é, o número de níveis de neurônios presentes. A segunda é o tipo de conexão, as quais são divididas pela presença ou não de ciclos na rede. Por fim, as arquiteturas podem ser separadas pelo tipo de conectividade, com redes completamente conectadas (cada neurônio é conectado com todos os outros) e parcialmente conectadas (cada neurônio é conectado a alguns dos outros neurônios).

As arquiteturas de camada única são caracterizadas pela presença de neurônios apenas na camada de saída, além de uma camada de entrada sem a presença de neurônios. Dessa forma, só há computação na camada de saída. A Figura 3.6 ilustra o caso.

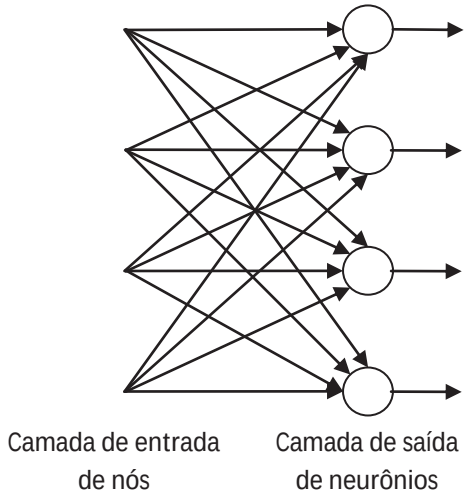


Figura 3.6: Arquitetura de camada única, acíclica e completamente conectada.

Já quando existem duas ou mais camadas de neurônios, a arquitetura é chamada de arquitetura de múltiplas camadas. Estas camadas a mais são escondidas e realizam um primeiro processamento das entradas. Um exemplo de arquiteturas de múltiplas camadas é ilustrado na Figura 3.7.

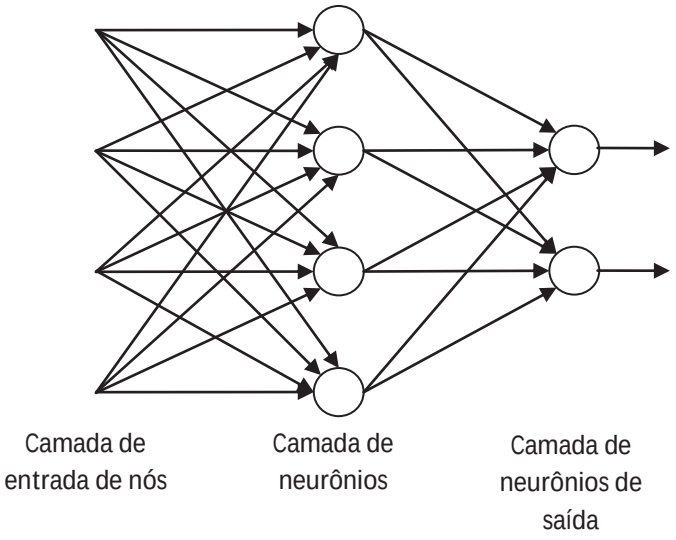


Figura 3.7: Arquitetura de múltiplas camadas, acíclica e completamente conectada.

A organização pelo tipo de conexão divide as redes em arquiteturas acíclicas e cíclicas. Uma arquitetura acíclica é aquela em que a saída de uma camada de neurônios não pode ser utilizada como entrada pela própria camada ou por uma camada anterior. Um exemplo deste caso é a rede da Figura 3.7. Caso esta condição

não seja satisfeita, a rede então é dita de arquitetura cíclica, como é ilustrado na Figura 3.8.

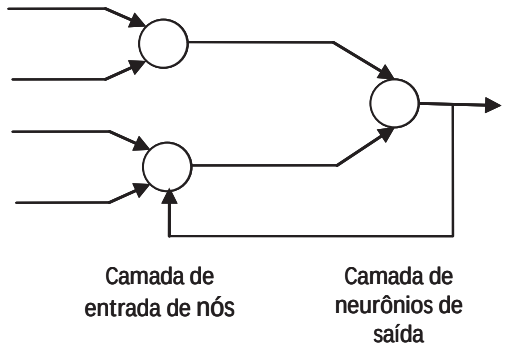


Figura 3.8: Múltiplas camadas, cíclica e completamente conectada.

Por fim, uma RNA pode ser classificada quanto à sua conectividade. Caso cada neurônio de cada camada esteja ligado a todos os neurônios da camada subsequente, a arquitetura é classificada como completamente conectada. Caso essa condição não seja satisfeita, a arquitetura é dita parcialmente conectada. A primeira situação é ilustrada nas Figuras 3.6 e 3.7, enquanto a segunda situação é representada na Figura 3.9.

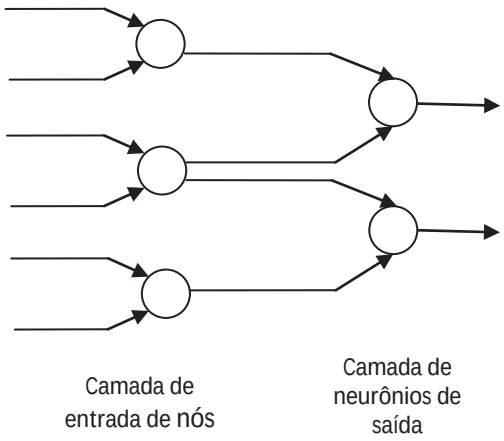


Figura 3.9: Múltiplas camadas, acíclica e parcialmente conectada.

3.3.2 Regra de Hebb e Regra Delta

A regra de Hebb é uma teoria para o fortalecimento do aprendizado de um neurônio artificial. Em sua formulação é dito que se um neurônio j recebe repetidamente estímulos de um neurônio i, o primeiro passa a ficar mais sensível a estímulos do segundo. Na prática, isso significa que o peso da entrada de j oriunda de i é ajustado de acordo com os estímulos recebidos. Esse ajuste é feito segundo a Equação 3.19.

$$\Delta w_{ij} = \lambda a_i a_j \qquad \text{(Equação 3.19)}$$

Na Equação 3.19, λ é uma constante de proporcionalidade que representa a taxa de aprendizado e a_i e a_j representam os níveis de ativação dos respectivos neurônios.

Uma extensão da regra de Hebb é a chamada regra Delta. Assim como a regra de Hebb, ela visa ajustar os pesos das entradas dos neurônios. Para tanto, padrões são apresentados a rede de forma que ela passa a produzir classificações utilizando os pesos aleatórios iniciais. As classificações produzidas são comparadas com as esperadas. A cada iteração os pesos são ajustados para que a diferença entre a saída da rede e a esperada seja menor. O ajuste de pesos da regra Delta é representado pela Equação 3.20.

$$\Delta w_{ij} = (y_i - a_j) a_i \qquad \text{(Equação 3.20)}$$

A regra Delta é aplicada nos casos de aprendizado supervisionado seguindo alguns passos. Primeiramente os pesos são iniciados com valores aleatórios. Para cada par do treinamento (dados e seu rótulo) é calculada a resposta obtida pela rede. Em seguida, é calculado o erro entre os resultados esperados e os obtidos. O processo é repetido até que o erro seja satisfatoriamente pequeno.

3.3.3 Algoritmo de retropropagação

Assim como quando o método de SVMs é utilizado, em RNAs existe o problema de trabalhar com conjuntos de dados não separáveis linearmente. Uma

maneira de solucionar isso é pela utilização de várias camadas escondidas de neurônios, o que permite a implementação de superfícies de decisão mais complexas. Esta abordagem traz uma dificuldade maior para o treinamento das RNAs. O uso de um algoritmo de retropropagação para o treinamento das redes é uma das soluções mais utilizadas para esse problema.

Durante o processo de treinamento de um algoritmo de retropropagação, os dados são introduzidos na camada de entrada e a rede trabalha normalmente propagando os resultados até a saída. Na saída, o resultado é comparado com o rótulo esperado para o problema e se houver uma discrepância acima de um tolerado, um sinal de erro é propagado de volta pela rede até a entrada para promover o reajuste dos pesos. O algoritmo de retropropagação trabalha uma variação da regra Delta, chamada de regra Delta generalizada. A principal diferença entre as duas regras está na necessidade do uso de funções de ativação contínuas e suaves, como a função sigmoidal. A regra Delta comum utiliza uma função de ativação limiar.

3.4 Considerações finais

Com base no que foi visto nesse capítulo, as técnicas de aprendizado de máquina buscam maneiras de determinar classificadores para agrupar dados para identificação de padrões rotulá-los em classes já estabelecidas. A principal dificuldade no trabalho com estes métodos está no trabalho com conjuntos de dados que não podem ser separados linearmente. Cada método soluciona este problema com uma abordagem própria que pode ser favorável ou não dependendo dos dados com que se está trabalhando.

Capítulo 4 – Desenvolvimento em *software*

4.1 Considerações iniciais

Ao se trabalhar com um problema que envolve múltiplas disciplinas é necessário que parâmetros sejam estabelecidos para que a troca de informações de áreas diferentes do conhecimento funcione. Por exemplo, neste trabalho são utilizados conceitos de agronomia, processamento digital de imagens, sistemas inteligentes e desenvolvimento de sistemas em *software* e *hardware*. Parâmetros como as condições em que a plantação precisa estar para que o sistema funcione, tipos de processamento a serem feitos nas imagens, configurações em que se pretende que as técnicas de sistemas inteligentes sejam implementadas e as limitações que o *hardware* impõe sobre o sistema precisam estar bem definidos para que o sistema atinja seu objetivo. Neste capítulo é detalhada cada etapa necessária para se chegar ao classificador escolhido para ser implementado em *hardware* e os requisitos para que ele funcione.

4.2 Testes preliminares

A princípio, esperava-se que o sistema fosse capaz de receber as imagens adquiridas e determinar se havia ou não plantas daninhas na mesma. Para isso, foram realizados testes com uma configuração de SVM para verificar se o mesmo seria capaz de processar todos os *pixels* de uma imagem com apenas um pré-processamento, ou seria necessário realizar a extração das características dessas imagens para reduzir a quantidade de informações a serem processadas. Esses testes foram realizados com um banco de imagens de baixa qualidade, o único disponível na época, então ele precisou ser trabalhado e selecionado antes de ser utilizado. Nessa seção são descritos os testes feitos e os resultados obtidos com esse protótipo do sistema.

4.2.1 Seleção das imagens

A cultura da soja, como a maioria das culturas, depende de boa concentração de chuvas para que o solo mantenha um grande nível de umidade. Por esse motivo, as plantações são cultivadas no período mais chuvoso no país, que tem início em novembro. Nesse período é possível encontrar plantações jovens, ainda em desenvolvimento, que são as que precisam de ajuda para se sobressaírem contra as plantas daninhas na competição pela água e nutrientes. Uma vez adulta, a planta da soja apresenta folhas grandes e com grande densidade, impedindo que outras plantas menores recebam a luz solar e dispensando a necessidade do uso de herbicidas para o controle destas. A Figura 4.1 ilustra uma plantação de soja adulta.

Figura 4.1: Plantação de soja adulta.

Então, para o desenvolvimento do trabalho, é interessante o uso de imagens de plantações jovens, nas quais é possível identificar tanto as plantas de soja, como as suas competidoras. O único banco de imagens disponível no início do desenvolvimento do trabalho era o que já vinha sendo utilizado no projeto para o desenvolvimento dos filtros de pré-processamento de Mertes [Mertes, Marranghello e Pereira, 2008]. Esse banco de imagens era composto por 49 imagens de resolução 360x240 *pixels* de uma plantação com alto grau de infestação de plantas daninhas e com a plantação de soja bastante degradada. A Figura 4.2 é uma das imagens do banco na qual é possível notar o grau de infestação de plantas daninhas (folhas menores).

Figura 4.2: Imagem do banco de imagens de soja.

Se as imagens do banco fossem utilizadas no estado em que se encontravam, não haveria amostras de imagens sem plantas daninhas para o processamento pelos sistemas inteligentes. Para solucionar esse problema, as imagens do banco foram cortadas em 36 imagens de resolução 60x40 *pixels*. A redução do tamanho das imagens também ajuda no desenvolvimento da versão em *hardware* do sistema, uma vez que a quantidade de memória disponível é bastante limitada em relação a um computador de uso geral. Dessa forma, foram obtidas 1764 imagens a partir das originais. Essas imagens menores, porém, podem apresentar pouca ou mesmo nenhuma informação interessante para o sistema, como no caso daquelas em que apenas o solo está presente. Portanto, dentre essas imagens, foram selecionadas 230 imagens mais significativas para o problema, das quais 30 são destinadas ao

treinamento da SVM e as 200 restantes utilizadas para avaliação do desempenho do sistema.

4.2.2 Filtros de pré-processamento

Devido à natureza das imagens utilizadas no trabalho, é esperado que haja diversos fatores que influenciem na qualidade delas. Esses fatores podem ser desde a luz do sol causando reflexo em algumas imagens e sombras em outras, como tremulações ou partículas do ambiente sobre as folhas das plantações que dificultem a identificação das folhas. Portanto, qualquer tentativa de minimizar esses efeitos é bem-vinda.

No trabalho, os efeitos que tentamos minimizar são justamente os citados. Na Figura 4.3 é possível notar claramente a influência deles nas imagens. Em (a) temos uma imagem com grande quantidade de ruídos, o que resulta em uma dificuldade em identificar as folhas. Já em (b), o que se nota é que parte da plantação está coberta por uma sombra, dificultando a interpretação de que tipos de folhas estão presentes. Por fim, em (c), o que se nota é um grande reflexo devido ao sol intenso, alterando completamente a tonalidade da folha de soja e dificultando o estabelecimento de um padrão para ela.

(a)

(b)

(c)

Figura 4.3: Efeitos encontrados em imagens da plantação.

Para tentar reduzir os efeitos apresentados, foram implementados três filtros de pré-processamento para o espaço RGB de 24 *bits*. O primeiro deles é um filtro de média,

que percorre a imagem com uma máscara e calcula a média dos valores da vizinhança de cada *pixel*, substituindo o valor do original. Esse filtro visa reduzir os ruídos encontrados nas imagens. Também com esse objetivo, foi implementado um filtro de mediana, que calcula a mediana dos vizinhos de cada *pixel* e substitui o original. Por último, foi implementado um filtro de controle de luminosidade [Alexandre, 2012], que tenta clarear as regiões da imagem onde estão presentes sombras e reduzir o brilho das regiões onde o reflexo é alto.

4.2.3 Análise dos filtros de pré-processamento

A aplicação de cada um dos filtros implementados às imagens visou à melhora das características das mesmas a fim de facilitar o processamento do sistema inteligente. Porém, ao aplicarmos os filtros, além dos benefícios para imagem para os quais são projetados, eles trazem alguns efeitos colaterais. Nesta seção estão expostos por meio de imagens os ganhos e perdas dos algoritmos implementados quando aplicados ao banco de imagens.

Os filtros de média e mediana, por apresentarem o mesmo objetivo, acabam compartilhando do mesmo efeito colateral. Na tentativa de suavizar os ruídos presentes nas imagens, as bordas dos objetos contidos nelas acabam sendo prejudicadas. Isso acaba prejudicando, em alguns casos, a distinção entre os objetos, principalmente aqueles que têm características semelhantes como as folhas. Para não sofrer tanto com esse efeito, os filtros implementados utilizaram uma máscara de 3 *x* 3, consequentemente, diminuindo o efeito da suavização. Na Figura 4.4 podemos ver os efeitos da aplicação dos filtros. A coluna (a) é composta pelas imagens originais. Na coluna (b) estão as imagens filtradas pela mediana. Por fim, na coluna (c), estão as imagens filtradas pela média.

(a) (b) (c)

Figura 4.4: Imagens processadas pela média e mediana.

Pelos resultados obtidos é possível observar um desempenho superior da filtragem pela mediana. Além das folhas perderem menos a característica do contorno, já que a imagem fica menos borrada, os ruídos são menos evidentes.

O desempenho esperado do filtro de controle de luminosidade é de que ele seja capaz de clarear as áreas escuras e escurecer as áreas claras sem que haja uma descaracterização dos objetos da imagem, ou seja, as folhas. A Figura 4.5 ilustra a comparação entre as imagens originais (a) e as processadas pelo filtro (b) quando se busca a redução do efeito das sombras e reflexo presente nas imagens.

(a) (b)

Figura 4.5: Filtro de controle de luminosidade.

A partir das imagens apresentadas, é possível notar que o filtro não obteve um desempenho satisfatório. Apesar das regiões escuras da imagem serem de fato clareadas, o aspecto das folhas não foi preservado, deixando em algumas regiões a imagem totalmente verde e impossibilitando a distinção entre as folhas. Já as regiões com maior brilho pouco foram alteradas, não justificando a utilização deste filtro de pré-processamento.

4.2.4 Implementação da SVM de testes

Como o desenvolvimento da SVM visa uma futura implementação em *hardware*, é fundamental que alguns aspectos inerentes desses tipos de sistemas sejam levados em consideração mesmo no prévio desenvolvimento em *software*. A questão fundamental nesse caso é a limitação no uso de memória a qual um FPGA, por exemplo, está sujeito. Na placa que temos disponível para implementação existe

uma memória de 310KB. Caso haja necessidade de mais espaço, os blocos lógicos devem ser utilizados para essa finalidade.

Com isso em mente, a arquitetura da SVM desenvolvida precisa contemplar dois fatores contraditórios. O primeiro é aceitar como entrada uma imagem inteira, de resolução 60x40, com 24*bits* de informação em cada *pixel*. O segundo é limitar a quantidade total de memória utilizada a 310KB. Para tanto, foi utilizada a arquitetura ilustrada na Figura 4.6. Nela é possível notar que existem três camadas de elementos de processamento. A primeira é camada de entrada, composta por 2400 elementos cuja finalidade é receber os valores de cada um dos *pixels* da imagem. Cada elemento de entrada está ligado diretamente a todos os 30 elementos da segunda camada, sem atribuição de pesos. Os elementos de processamento da segunda camada, por sua vez, estão todos conectados ao único elemento da terceira camada, sendo que cada conexão é ajustada por um peso. O elemento da terceira camada é responsável por apontar a presença ou não de plantas daninhas na imagem.

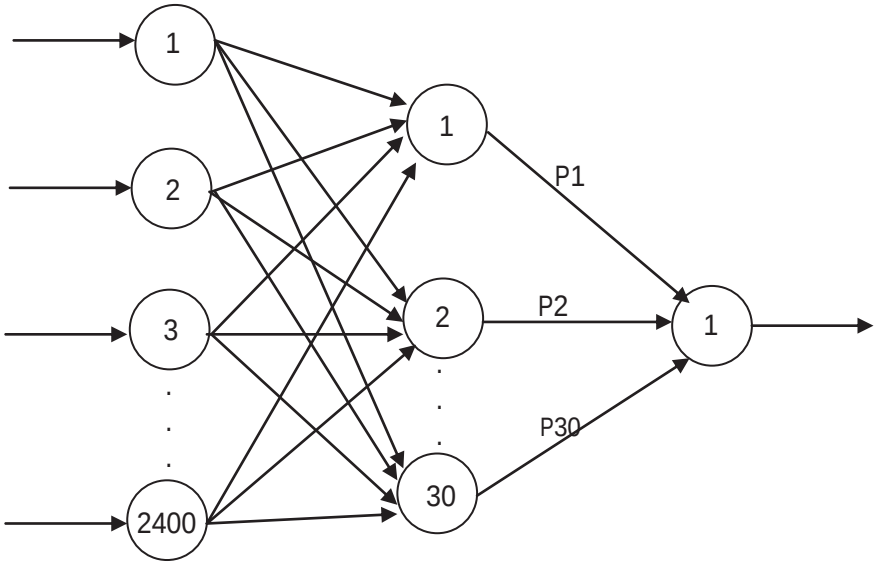


Figura 4.6: Arquitetura da SVM implementada.

Para o funcionamento da SVM é necessário o armazenamento de algumas informações. A primeira delas é a imagem que está sendo processada, armazenada na primeira camada. Na segunda camada são armazenadas as imagens utilizadas no treinamento como vetores de suporte, ou seja, 30 imagens. Na terceira camada são

armazenados os pesos das conexões entre a segunda e terceira camadas. Portanto, é necessário o armazenamento de 31 imagens de 7200 *bytes* cada uma, mais os valores dos 30 pesos. As imagens armazenadas na segunda camada são utilizadas para o cálculo dos valores de saída dos seus elementos de processamento utilizando o *kernel* Gaussiano, indicado na Equação 4.1. A saída da camada três é calculada pela soma ponderada dos valores da camada dois com os respectivos pesos.

$$s = e^{-dist(v1,v2)} \qquad \text{(Equação 4.1)}$$

Na Equação 4.1, s corresponde à saída de cada neurônio da camada dois e dist(*v1,v2*) é a distância vetorial entre *v1* – saída da primeira camada – e *v2* – imagem armazenada no respectivo neurônio da camada.

Para o treinamento da rede, primeiramente as 30 imagens de treinamento são carregadas na segunda camada. Em seguida, cada uma delas é processada pela rede até a segunda camada. Os valores de saída da segunda camada são multiplicados pelos respectivos pesos, incógnitas até então. Com isso, são obtidas 30 equações com 30 variáveis, igualadas ao valor esperado de saída da rede para cada imagem do treinamento. Com a solução desse sistema de equações os valores dos 30 pesos são encontrados e atribuídos a camada três para o processamento das demais imagens.

Com a rede treinada, as 200 imagens do banco foram processadas. Do total, 123 imagens tiveram resultados de acordo com o previsto, ou seja, 61,5% das imagens tiveram a presença ou não de plantas daninhas identificadas corretamente. O valor de aproveitamento é considerado baixo. Dentre os principais motivos para o resultado podem estar a forma de treinamento utilizada, a escolha das imagens para o treinamento ou as próprias imagens do banco serem inadequadas. Mas, a provável razão para essa taxa é o pequeno número de padrões de treinamentos utilizados na segunda camada devido às limitações da implementação em *hardware*. Uma vez que são utilizadas 2400 entradas com 2^{24} valores possíveis para cada uma, o número de combinações de entrada é muito elevado. Portanto, a maneira mais eficaz de melhorar o aproveitamento passa pela extração de características das imagens

resumindo os dados de entrada para elevar o número máximo de padrões possíveis no treinamento.

4.3 Implementações definitivas

Como a SVM não apresentou um desempenho satisfatório utilizando como entrada uma imagem completa, optou-se pela extração de características das imagens. Nesse caso, o problema passou a ser quais características podem ser utilizadas para identificar as plantas daninhas em meio a plantação. Características como cor e formato das folhas não poderiam ser utilizadas, já que as cores das folhas de soja e das plantas daninhas são semelhantes e o formato das folhas dificilmente poderia ser identificado, pois, além de serem em grande quantidade, as folhas são sobrepostas umas às outras. Optou-se então pela extração de informações sobre a textura das imagens e suas energias. Além disso, foi possível contar com um novo banco de imagens, já que se iniciou um novo período de plantio.

Para realização dos testes, foram utilizadas uma nova arquitetura para a SVM com três *kernels* distintos e algumas configurações de RNAs com duas funções de ativação. Foram realizados 20 mil testes com cada configuração e variando os parâmetros na extração da textura e da energia das imagens, com o objetivo de encontrar um classificador mais robusto. Nessa seção serão detalhadas as implementações realizadas, bem como o novo banco de imagens.

4.3.1 Novo banco de imagens

Com o início do período das chuvas, novas lavouras de soja foram cultivadas, possibilitando a composição de um banco de imagens mais adequado para o desenvolvimento do sistema. A plantação onde as imagens foram capturadas possui cerca de 70 hectares de soja e fica na região de Vitória Brasil – SP. Foram feitos dois tipos diferentes de plantio na propriedade, o que ajudou na variação das condições das imagens coletadas. O primeiro tipo é o plantio direto em que não existe uma preparação prévia do solo, então os restos de lavouras anteriores formam uma camada protetora para o solo. O outro tipo é o plantio indireto, em que o solo é preparado e, portanto, a terra é revolvida e os restos das plantações anteriores acabam

misturados. Na Figura 4.7a temos uma imagem do novo banco com plantio direto, enquanto na Figura 4.7b é ilustrado o plantio indireto.

(a) (b)

Figura 4.7: Imagens do novo banco.

O novo banco de imagens possui a seguinte composição: 154 imagens de plantio indireto, 49 imagens de plantio direto, totalizando 203 imagens. Desse montante, em 83 imagens é possível identificar a presença de plantas daninhas enquanto outras 120 estão limpas. Foi mantido um padrão na aquisição das imagens, mantendo a câmera posicionada diretamente para o solo, a cerca de 1m de altura, e com o *flash* sempre ligado, para tentar minimizar possíveis sombras. Além disso, foram capturadas com diferentes níveis e luminosidade, proporcionando maior variedade de situações. As imagens capturadas foram convertidas para o formato BMP de 24 bits sem compressão, para facilitar o processamento, e possuem resolução de 2048x1536 *pixels*.

4.3.2 Extração de características

Como foi citado anteriormente, foi definida a extração de informações da textura e energia da imagem para serem utilizadas pelo sistema inteligente. Existem diversos métodos para extrair características de texturas em imagens. Nesse caso, optou-se por determinar a dimensão fractal da imagem. A dimensão fractal tem como característica informar o grau de autossimilaridade da imagem, ou seja, o quanto pequenos padrões se repetem dentro de uma determinada região. Com isso, é

possível determinar se a imagem possui uma textura rugosa (dimensão fractal elevada) ou lisa (dimensão fractal baixa). Como a soja possui folhas largas, espera-se uma textura mais lisa se comparada a plantas daninhas, que possuem folhas menores e aglomeradas. Além disso, o banco de imagens é composto por fotografias de lavouras com plantio direto, os restos de outras plantações representam um ruído cujo sistema inteligente precisa ser capaz de lidar. Já a energia da imagem diz respeito a uma condensação do valor bruto dos *pixels* da imagem. Dessa forma, é possível identificar padrões em diferentes regiões da imagem comparando o valor dos *pixels* de uma região da imagem com os obtidos na imagem toda. Com isso, se espera respostas diferentes para a presença de solo, resto de outras plantações, folhas de soja e folhas de plantas daninhas presentes na imagem e que essas respostas sirvam para alimentar os sistemas inteligentes.

Para determinar a dimensão fractal de uma imagem existem variadas técnicas. A escolhida para ser utilizada foi de *box counting*, já que esta técnica apresenta resultados similares a outras mais robustas e possui tempo de processamento inferior. Os valores de dimensão fractal obtidos para uma imagem – plano cartesiano 2D – variam entre 2 e 3. O cálculo do valor usando esta técnica é feito linha por linha e coluna por coluna. Usando como exemplo o cálculo para uma linha, primeiramente monta-se um gráfico com o valor de cada *pixel* pela sua posição na linha. Esse gráfico então é normalizado para a menor potência de 2 em que caibam os valores dos *pixels* e o tamanho de cada linha. Supondo que o valor do lado normalizado do gráfico seja 1024, considera-se que a primeira caixa (*box*) tenha lado 1024 por 1024. A partir daí é construída uma tabela em que x é o tamanho do lado de cada caixa dentro do gráfico e y a quantidade de caixas dentro do gráfico que são cobertas pelos valores dos *pixels* da linha. A cada iteração o lado da caixa é dividido por 2 e a contagem de caixas cobertas refeita. Para o exemplo, x teria 10 valores partindo de 1024, 512, 256, até chegar em 1. Essa tabela, então, é convertida para base logarítmica a partir da qual um novo gráfico de pontos $\log_2(x)$ por $\log_2(y)$ é montado. Esses pontos, quando interpolados pelo método dos mínimos quadrados, geram uma equação de reta. O coeficiente linear da equação obtida é a dimensão fractal daquela linha. O cálculo é repetido para cada linha e coluna, a média dos valores obtidos somados a 1 é o valor da dimensão fractal para a imagem.

Para realizar a classificação pelo sistema inteligente, apenas um valor para a textura da imagem é insuficiente. As imagens do banco elaborado possuem vários conjuntos de informações, como solo, plantas de soja, plantas daninhas e plantas secas, no caso do plantio direto. Portanto, é desejável que as informações de textura sejam analisadas separadamente para cada caso. O extrator de textura implementado é capaz de dividir a imagem em 4, 16 ou 64 partes e fornecer a dimensão fractal individual de cada uma delas. Com isso, é possível ter informações específicas de cada parte da imagem, bem como realizar testes para verificar com qual configuração os sistemas inteligentes obtêm melhores resultados.

A mesma ideia foi utilizada para extração dos valores de energia da imagem, com a divisão da mesma em 4, 16 ou 64 quadrantes. O cálculo do valor da energia para cada quadrante é bastante simples. Primeiramente, todos os *pixels* da imagem são normalizados em função do valor do *pixel* de maior valor. Então, é calculada a energia da imagem toda, que nada mais é que o somatório dos valores de cada *pixel* normalizados elevados ao quadrado. Por fim, a imagem é dividida de acordo com a quantidade de quadrantes desejada e valor da energia de cada quadrante é calculada. O valor final para cada quadrante é a sua energia dividida pela energia total da imagem.

4.3.3 Implementação das SVMs e RNAs

Com os conjuntos de entradas definidos, foi possível desenvolver as arquiteturas das SVMs e RNAs a serem implementadas. Como a extração das características das imagens reduziu consideravelmente a quantidade de dados que seriam processados, foi possível, no caso da SVM, aumentar a quantidade de vetores de suporte utilizados na segunda camada. Foram utilizados então 100 vetores de suporte, o que consumiu 100 imagens de treinamento, sendo 50 com a presença de plantas daninhas e 50 limpas. A variação de parâmetros, então, ficou por conta da quantidade de entradas utilizadas, com vetores de características de tamanho 4, 16 e 64. Na Figura 4.8 vemos um esquema da arquitetura utilizada.

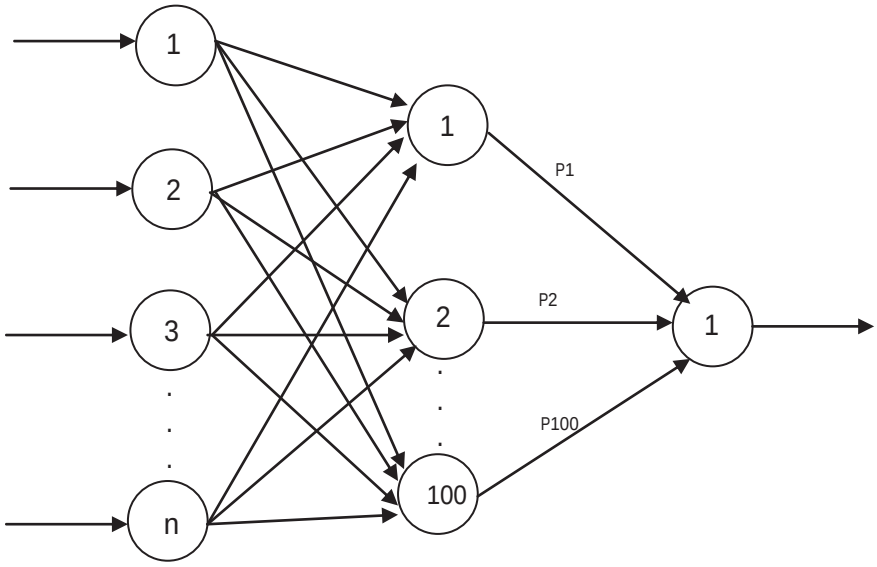


Figura 4.8: Arquitetura definitiva das SVMs implementadas.

Além disso, foram utilizados outros dois *kernels* além do Gaussiano implementado nos primeiros testes. O *kernel* Gaussiano, apresentado anteriormente, foi escolhido para os testes em função de ser um dos mais utilizados e bastante robusto. Outro *kernel* escolhido foi o linear, utilizado por ser de implementação simples e de execução rápida. O *kernel* linear está indicado pela Equação 4.2. Por fim, o outro *kernel* utilizado foi o de Intersecção por histograma [Barla, Odone e Ferri, 2003], pois tem apresentado bons resultados em aplicações relacionadas ao processamento de imagens e por sua implementação em *hardware* ser mais simples. Isso se deve à ausência de operações de potenciação, divisão e multiplicação, como é possível ver pela Equação 4.3.

$$S = \left(\overrightarrow{v1} \cdot \overrightarrow{v2}\right) + c \qquad \text{(Equação 4.2)}$$

$$S = \sum_{i=1}^{n} \min\left(\overrightarrow{v1}_i, \overrightarrow{v2}_i\right) \qquad \text{(Equação 4.3)}$$

Portanto, foram implementadas versões da SVM para receber as informações de textura e energia das imagens para 4, 16 e 64 valores, com os três tipos de *kernel* citados. Já as RNAs foram implementadas em três arquiteturas distintas, mas com os mesmos padrões de entradas. Foi utilizada uma rede *perceptron* de múltiplas camadas com treinamento feito por *retropropagação*. As três arquiteturas foram

elaboradas seguindo um padrão para o número de neurônios na segunda camada em função do número de entradas, sendo a quantidade definida pela raiz do número de entradas multiplicado pelo número de saídas, como ilustra a Figura 4.9.

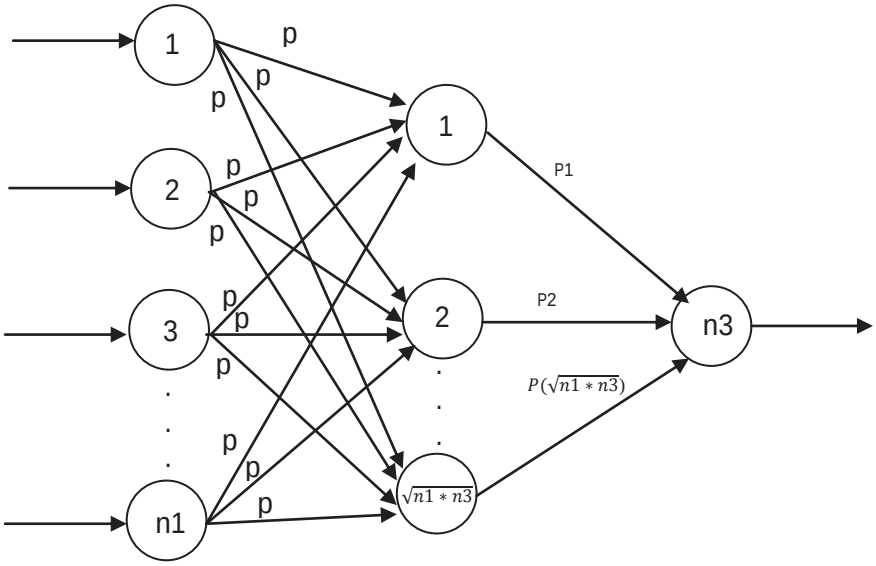


Figura 4.9: Arquitetura das RNAs implementadas.

Cada uma das arquiteturas foi implementada com duas versões de funções de ativação. As funções escolhidas foram a sigmoide e tangente hiperbólica, devido à compatibilidade com o treinamento de retropropagação. A função sigmoide é mostrada na Equação 4.4, enquanto a função tangente hiperbólica é representada pela Equação 4.5.

$$s = \frac{1}{1+e^{-x}} \quad \text{(Equação 4.4)}$$

$$stanh = \frac{e^{output} - e^{-output}}{e^{output} + e^{-output}} \quad \text{(Equação 4.5)}$$

Dessa forma, para realização dos testes, foram implementadas nove versões de SVMs, cada uma podendo ser alimentada utilizando as texturas ou energias extraídas e seis versões de RNAs, que suportam entradas nos mesmos formatos das SVMs.

4.4 Testes realizados

As aplicações implementadas até então estabeleceram uma linha para quais seriam os testes realizados. Bastava definir os critérios para esses testes e como verificar qual classificador seria mais adequado para a implementação em *hardware*. Uma das variáveis faltando era a definição de qual conjunto de imagens seria utilizado para o treinamento e qual seria utilizado para testes. Segundo Tatibana e Kaetsu [Tatibana e Kaetsu, 1996], normalmente utiliza-se entre 50% e 90% do conjunto de dados para treinamento e o restante para testes. Já que o conjunto de imagens não era muito grande, foi utilizado aproximadamente 50% – 100 imagens – dele para o treinamento. A escolha das imagens a serem utilizadas no treinamento foi feita de forma aleatória. Foram realizados 20 mil treinamentos, com a mesma quantidade de imagens de cada classe, seguidos de testes para cada configuração de SVMs e RNAs implementadas, exceto em alguns casos. Os resultados obtidos serão apresentados nessa seção.

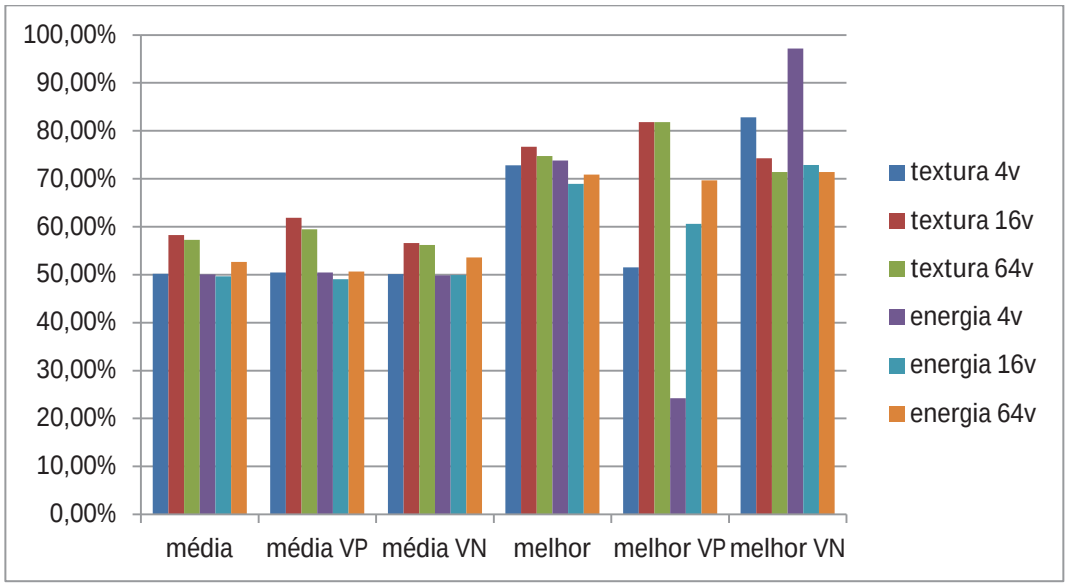


Figura 4.10: SVM com *kernel* Gaussiano.

Como citado anteriormente, para cada caso de testes, foram realizados 20 mil treinamentos seguidos de testes para avaliar os resultados do treinamento. Os casos de testes estão organizados segundo o tipo de técnica – SVM ou RNA – e o *kernel* ou função de ativação utilizado. O gráfico na Figura 4.10 ilustra os resultados obtidos

utilizando SVM com *kernel* Gaussiano. É possível notar que estão agrupados seis tipos de casos, cada um representado por uma cor. Cada caso representa um tipo de extração feita, de energia ou textura, com 4, 16 ou 64 valores de entrada. Na parte inferior do gráfico estão as medições realizadas. Média representa a média de acertos entre os 20 mil treinamentos e testes. Média VP e Média VN representam a média de resultados verdadeiro-positivos – existem plantas daninhas na imagem e o sistema diz que elas existem – e verdadeiro-negativos – não existem plantas daninhas na imagem e o sistema diz que não existem. Melhor representa o percentual de acerto do melhor resultado dentre os 20 mil testes. Melhor VP e Melhor VN são, respectivamente, os acertos de verdadeiro-positivo e verdadeiro-negativo do melhor resultado dentre os 20 mil testes.

Analisando os resultados obtidos com o *kernel* gaussiano, nota-se um acerto médio baixo, mesmo sendo um *kernel* robusto. Isso significa que caso o conjunto de imagens aumente, pode não ser muito fácil encontrar um conjunto de treinamento que obtenha um bom resultado. Os melhores resultados obtidos com esse *kernel* foram utilizando 16 e 64 valores de textura, chegando a quase 80% de acerto no melhor caso com 16 entradas. O destaque negativo ficou por conta dos valores de energia e os de menor quantidade de entradas onde, mesmo nos melhores casos, houve discrepância muito grande entre os verdadeiro-positivos e os verdadeiro-negativos.

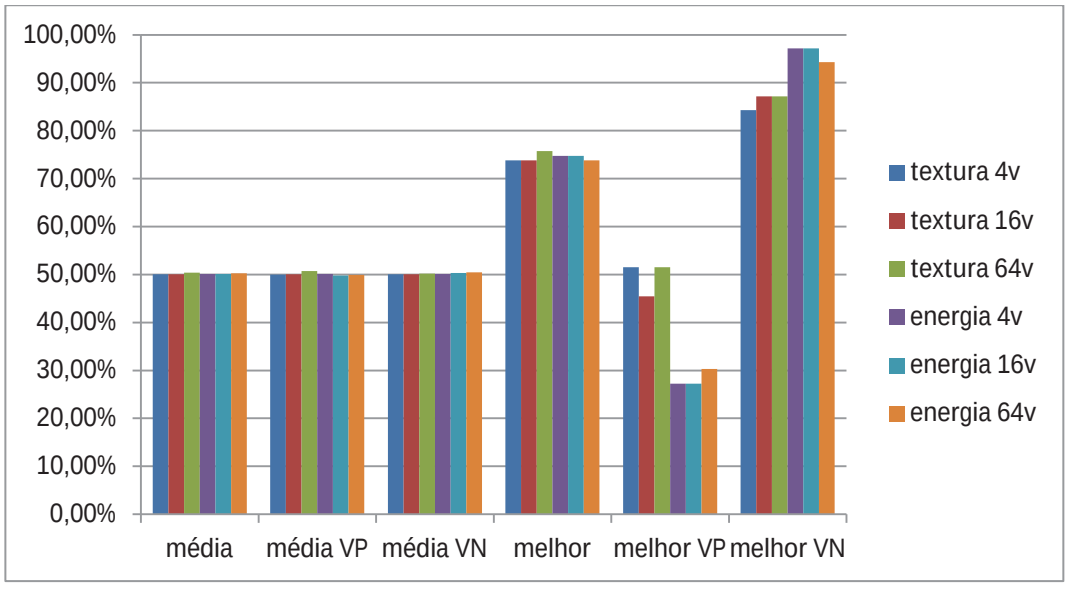


Figura 4.11: SVM com *kernel* Linear.

O gráfico na Figura 4.11 segue exatamente o mesmo padrão da figura anterior e das demais que serão apresentadas, dessa vez agrupando os casos testados com a SVM de *kernel* linear. Esse *kernel* é o de processamento mais rápido, porém não conseguiu apresentar bons resultados para o problema proposto. Nota-se que, em média, ele tem um acerto muito próximo a 50%, mesmo na média de verdadeiro-positivos e verdadeiro-negativos. Nos testes de melhor resultado os acertos em torno de 75% para todos os tipos de entrada podem enganar. É possível observar que os acertos de verdadeiro-negativo são bem maiores que os de verdadeiro-positivo, indicando que os classificadores têm uma tendência de apontar a inexistência de plantas daninhas nas imagens. Como a quantidade de imagens desse tipo é maior, o acerto é maior, mas enganoso. Vale notar que os testes que utilizaram entradas com extração de energia apresentaram verdadeiro-positivos menores ainda, outro destaque negativo.

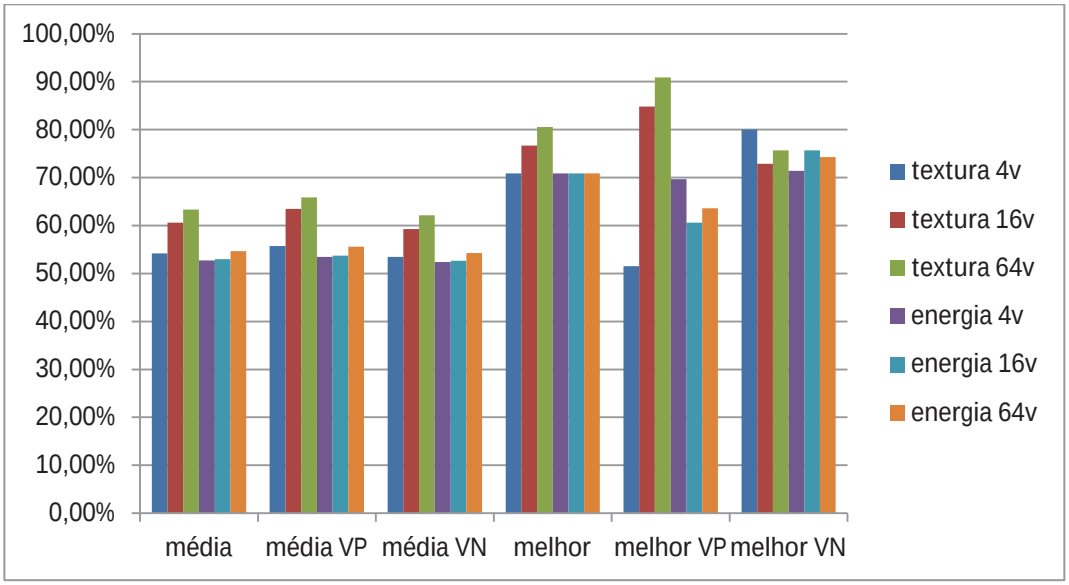


Figura 4.12: SVM com *kernel* Intersecção por histograma.

No gráfico da Figura 4.12 estão agrupados os resultados dos testes utilizando a SVM com *kernel* de intersecção por histograma. Esse *kernel* foi o que conseguiu os melhores resultados, chegando mais de 60% de média de acerto para os testes com 16 e 64 valores de textura. Além disso, nota-se um equilíbrio maior entre os valores de verdadeiro-positivos e verdadeiro-negativos, tanto na média, como no melhor

resultado. Isso indica que, mesmo com um conjunto de testes com muito mais imagens de uma classe que de outra, os resultados serão pouco influenciáveis por esse fator. Porém, o ponto de destaque foi o percentual de acertos no melhor resultado, já que, além de conseguir mais de 80% de acerto, a taxa de verdadeiro-positivos foi de mais de 90%. Isso significa que em mais de 90% dos casos o classificador obtido a partir do conjunto de treinamento sorteado indica que há plantas daninhas quando elas estão presentes e ainda indica corretamente a ausência das mesmas quando elas estão ausentes em 75%. Esse classificador apresenta resultados promissores, indicando ser possível a identificação de plantas daninhas em plantações de soja por um sistema inteligente. Além disso, como indicado anteriormente, devido a ausência de operações mais complexas, esse *kernel* necessita de menos recursos ou circuitos para ser implementado em *hardware*, tornando-o um excelente candidato.

Os gráficos mostrados nas Figuras 4.13 e 4.14 agrupam os resultados obtidos pelos testes realizados com as RNAs implementadas. O treinamento para as RNAs seguiu o mesmo processo utilizado nas SVMs, com seleção aleatória do conjunto utilizado no treinamento. Em alguns dos casos não foi possível realizar a seleção do conjunto 20 mil vezes, pois o treinamento chegou a durar 10 dias. Nesses casos foi escolhido apenas um conjunto de treinamento, baseado em situações de luminosidades e outras condições da imagem para inserir um maior conjunto de padrões no treinamento. Como é possível observar, nenhum dos testes apresentou resultados aproveitáveis. Os classificadores consideraram que todas as imagens pertenciam a uma mesma classe, fato que pode ser observado pelos dados de VP e VN, tanto na média, como no melhor resultado. Os motivos para que as RNAs não atinjam os mesmos resultados das SVMs podem ser diversos. Entre eles está a pouca variação entre os valores de entrada, utilizando textura ou energia, com valores semelhantes mesmo para imagens de classes diferentes. Outro fator que pode influenciar é a escolha de parâmetros do treinamento como a escolha inicial dos pesos da rede e o salto a cada iteração do algoritmo de retropropagação. Há ainda a possibilidade de teste de outras arquiteturas para RNA, em busca de alguma que seja adequada mais especificamente para o problema em questão. Como qualquer uma dessas alternativas demanda muito tempo e o foco do trabalho foi a implementação

de técnicas generalizadas, optou-se por seguir a implementação em *hardware* de apenas um classificador obtido por meio de máquinas de vetores de suporte.

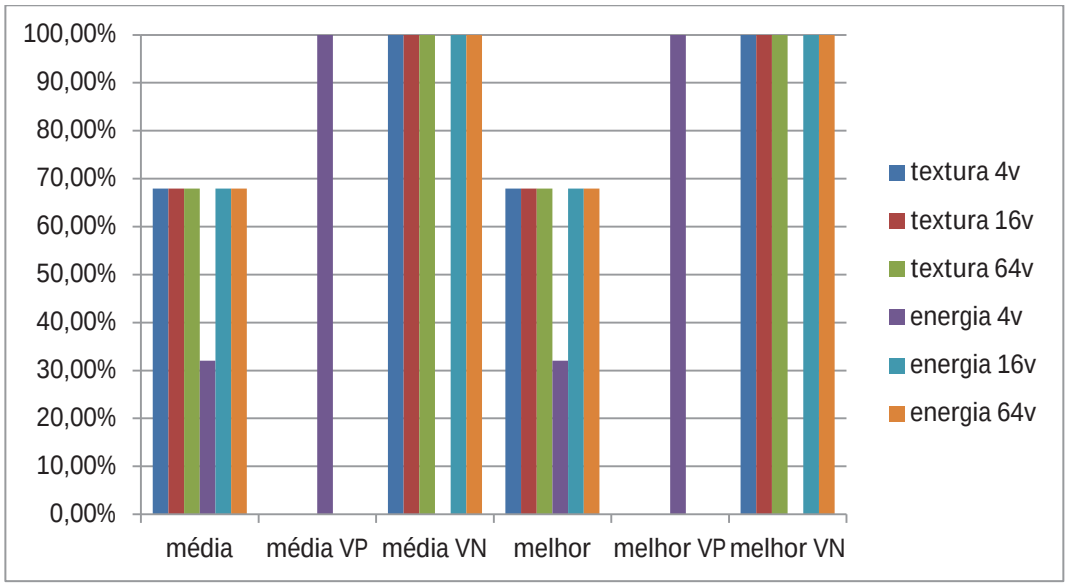


Figura 4.13: RNA com função de ativação Sigmoide.

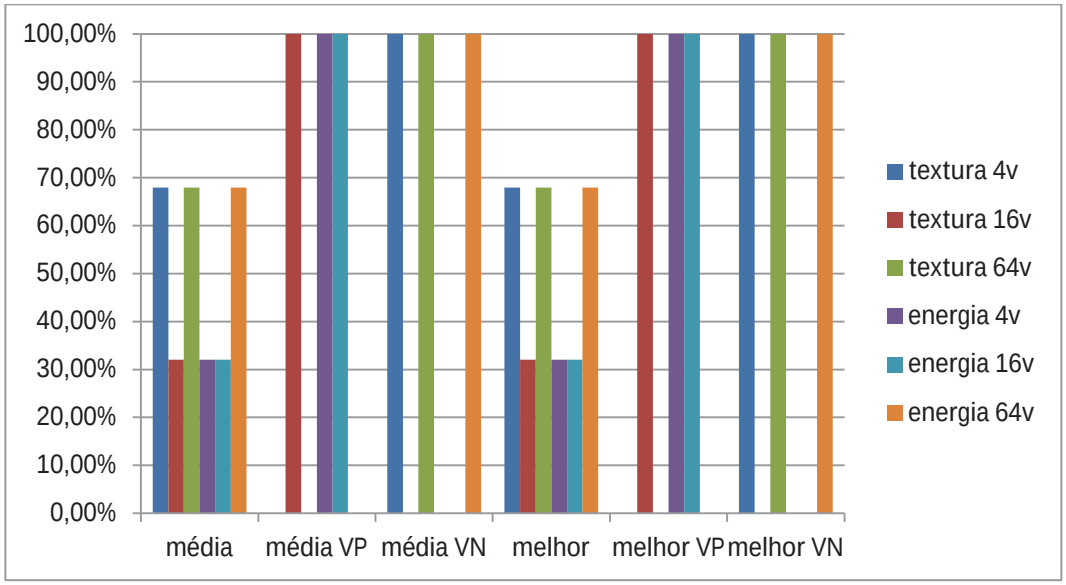


Figura 4.14: RNA com função de ativação Tangente hiperbólica.

Para auxiliar na escolha do classificador a ser implementado, foram realizados testes de desempenho. Os testes foram executados apenas para os classificadores SVM, já que os classificadores RNA não convergiram. Para obter os valores, foram utilizadas funções da biblioteca *time.h*. Os resultados são muito

influenciados por outros processos que estejam rodando no sistema. Portanto, foram executados com o mínimo possível de processos rodando em paralelo. O gráfico na Figura 4.15 ilustra os resultados obtidos.

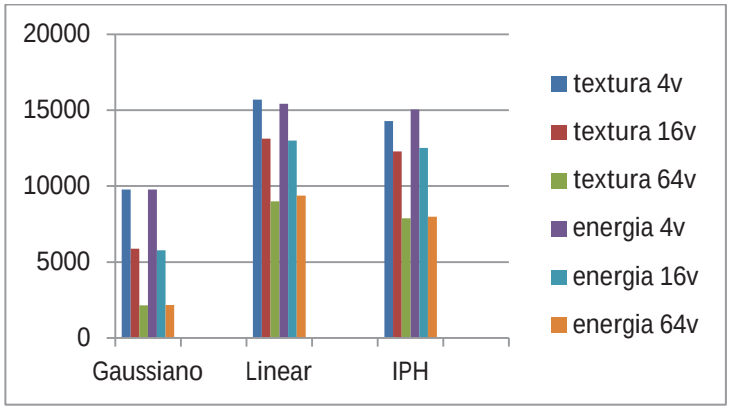


Figura 4.15: Desempenho dos classificadores.

Os testes seguiram os mesmos padrões dos realizados para avaliar a taxa de acertos. Nos resultados, medidos em imagens processadas por segundo, é possível observador pouca diferença de desempenho entre entradas usando textura e energia. O *kernel* do tipo Gaussiano apresentou um desempenho bem inferior aos demais, apesar de ainda processar uma grande quantidade de imagens por segundo. O *kernel* de Intersecção por Histograma mostrou um bom desempenho o que, junto com sua taxa de acertos, colabora para que seja o mais adequado para a implementação em *hardware*.

4.5 Considerações finais

Para poder estabelecer parâmetros para o desenvolvimento do sistema, diversos testes foram realizados para determinar o desempenho de técnicas de processamento de imagens e dos sistemas inteligentes. Foram definidas as condições para a captura das imagens das plantações que alimentam o sistema e como suas informações foram extraídas. Além disso, diversos testes foram realizados para definir qual classificador seria utilizado na sequência do trabalho, levando em conta sua taxa de acertos e adequação para implementação em *hardware* e o desempenho de cada classificador. Dessa forma, foi escolhido o classificador obtido por meio da SVM com *kernel* de intersecção por histograma utilizando como entrada a textura das imagens divididas em 64 quadrantes.

Capítulo 5 – Implementação em *hardware*

5.1 Considerações iniciais

Quando se deseja implementar um sistema em *hardware* é preciso sempre ter em mente qual será o objetivo desse sistema. Sistemas dos quais não se espera grande desempenho e sim a manutenção os custos de produção baixos encontram nos microcontroladores a melhor escolha. Quando o desempenho é uma prioridade, o uso de circuitos específicos acaba sendo a saída. Nesse caso, as opções passam pelo desenvolvimento de um circuito integrado ou a utilização de um FPGA. No primeiro caso, o desempenho é o ponto forte, mas os custos de produção são proibitivos para a produção de poucas unidades. No segundo caso, o desempenho é inferior, mas o custo de uma placa com um FPGA embarcado é muito inferior se comparado ao de um circuito integrado. Além disso, o desenvolvimento e testes levam um tempo consideravelmente inferior. Por esses motivos optou-se pela implementação em FPGA do classificador para detecção de plantas daninhas. Nesse capítulo será detalhada a implementação do circuito, bem como sua verificação e testes de desempenho.

5.2 Implementação do circuito

Para a implementação do classificador proposto nesse trabalho foi utilizada uma placa de desenvolvimento com um *chip* FPGA embarcado. A placa utilizada é a *Nios Development Board* da Altera [Altera, 2005] com o FPGA EP2S60F672C5ES, da família *Stratix II*. A Figura 5.1 ilustra os recursos presentes na placa.

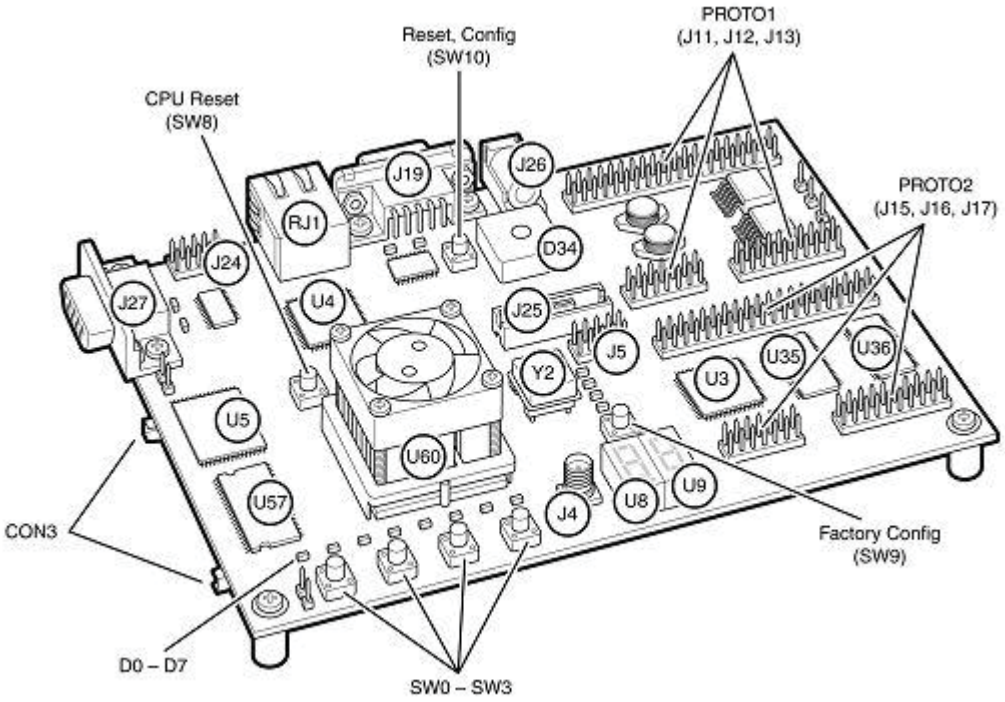


Figura 5.1: Representação da *Nios Development Board* [Altera, 2005].

Dentre os recursos que valem destaque estão as quatro chaves SW0-SW3, os conjuntos de pinos J11, J12, J13, J15, J16 e J17, os oito LEDs (*Light-emitting Diode*) D0-D7 e o *clock* interno Y2 de 50Mhz, que podem ser programados pelo usuário como entradas e saídas de dados para o FGPA. O mesmo ainda conta com 48352 LUTs adaptáveis, que podem ser programadas para trabalharem com as funções lógicas desejadas, e aproximadamente 310KB de memória RAM (*Random Access Memory*).

A programação do FPGA é feita utilizando uma HDL, os principais exemplos dessas linguagens para o desenvolvimento de sistemas digitais são: VHDL, *verilog* e

system verilog. A linguagem escolhida para a implementação do classificador foi a última, pois, além de ser mais recente e com mais recursos, possui nível de abstração elevado, facilitando a adaptação dos códigos já desenvolvidos em C, bem como sua verificação funcional. Apesar disso, alguns ajustes no classificador ainda foram necessários para a implementação.

A primeira diferença na implementação do classificador em *software* para o em *hardware* está nas operações de ponto flutuante. Tanto os valores das texturas extraídas das imagens, como os pesos internos utilizados pelo classificador são do tipo real. Para lidar com isso é necessária a implementação de uma unidade para tratar as operações de ponto flutuante, ou converter os valores utilizados para inteiros. Optou-se pela segunda solução por ser de implementação mais simples e sem aumento na quantidade de lógica utilizada do FPGA. Portanto, os pesos extraídos pelo treinamento da SVM foram multiplicados por 100000, enquanto os valores das texturas foram multiplicados por 10000.

Outra questão que precisa ser levada em conta é a maneira como deve ser feita a interface para entrada e saída de dados. A Figura 5.2 ilustra as portas de entrada e saída que foram utilizadas para o circuito. Notam-se as portas CLK e RST que são as utilizadas para entrada dos sinais de *clock* e *reset*, fundamentais em qualquer circuito. Foi inserida também uma porta *en*, que serve para habilitar ou desabilitar o funcionamento do circuito, possibilitando que um controlador diga quando o módulo da SVM deve trabalhar ou aguardar enquanto outros circuitos processam informações. A última porta a ser destacada aqui é a de entrada de dados *in*, que possui 32 *bits* e tem a função de receber os 64 valores das texturas individualmente. Optou-se pelo recebimento dos dados a partir de uma memória, já que dessa forma outros filtros podem realizar o pré-processamento da imagem e armazenar os dados em uma memória para então serem utilizados pela SVM. Dessa forma, as portas de saídas do módulo implementado contam com sinal de leitura/escrita *wr_en* para controlar se os dados serão escritos ou lidos na memória e um barramento de 6 vias para saída do endereço do qual o valor de entrada deve ser lido ou no qual o de saída deve ser escrito. Além disso, existem os sinais *ready*, indicando o fim do processamento da imagem atual, e *result*, que possuiu valor 1

quando existe uma planta daninha na imagem, 0 na ausência e alta impedância enquanto o processamento não estiver concluído.

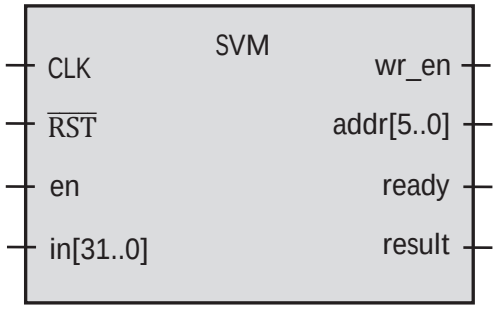


Figura 5.2: Interface de entrada e saída.

O paradigma de programação utilizado pelas HDLs representa outro ponto que diverge em relação a linguagens como C. O paradigma da última é o estruturado, onde a codificação dita uma sequência de operações que devem ser executadas em ordem para se chegar ao resultado esperado. Já nas HDLs é preciso ter o cuidado de perceber que o que está sendo codificado é a representação em alto nível de um circuito. Portanto, estruturas como laços de repetição e de desvio de fluxo do programa não funcionam da mesma forma ou mesmo não podem ser utilizadas. Para converter um algoritmo pensado para uma linguagem estruturada em uma descrição de *hardware* as estruturas utilizadas devem ser apropriadas. Para o problema da implementação do classificador foi necessária a utilização de uma máquina de estados. Essa estrutura serve para determinar os possíveis estados em que o circuito descrito pode estar e a sequência em que eles podem ocorrer. É a maneira mais simples de converter um algoritmo estruturado em uma descrição de *hardware*. Em cada estado, as ações a ele incumbidas são executadas, garantindo a ordem de execução das operações, tal qual em uma linguagem estruturada. Existem basicamente dois tipos de máquinas de estados, a de Moore e a de Mealy. Na primeira, as saídas do circuito dependem apenas do estado atual da máquina, enquanto na segunda as saídas são determinadas em função das entradas e do estado atual. Para implementar o classificador foi escolhido a do tipo Moore, cujo diagrama está representado na Figura 5.3.

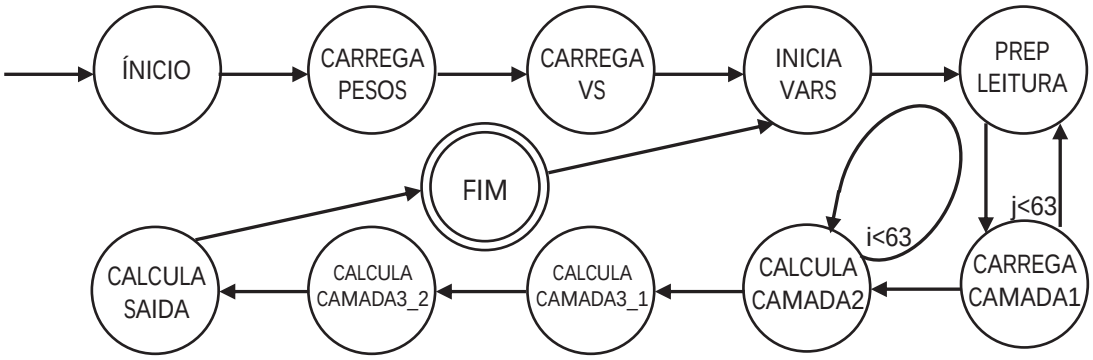


Figura 5.3: Máquina de estados da SVM.

Na máquina de estados apresentada na Figura 5.3, tem-se um estado INÍCIO que é o ponto de partida do circuito com a configuração para qual o retorna-se quando o circuito é reiniciado. Os estados CARREGA PESOS e CARREGA VS servem para definir os pesos e os vetores de suporte a serem utilizados. Nesse caso, os valores foram extraídos após o treinamento que obteve os melhores resultados quando testado em *software*. No estado INICIA VARS, as variáveis internas que precisam de valores iniciais específicos são iniciadas. Em PREP LEITURA, são atribuídos endereços de saída de 0 a 63 para a leitura dos valores da textura de uma memória. Em CARREGA CAMADA1, os valores fornecidos pela memória são carregados. No estado CALCULA CAMADA2 é feita a aplicação do *kernel* para cada um dos 64 valores do vetor de entrada, gerando 100 saídas para a camada 3. Em CALCULA CAMADA3_1 os valores obtidos na camada 2 são somados enquanto ponderados pelos pesos positivos. Em CALCULA CAMADA3_2 o processo é repetido para os pesos negativos. No estado CALCULA SAÍDA, o valor bruto é atribuído a uma classe, definindo a presença ou não de plantas daninhas. Por fim, no estado FIM, o resultado é jogado na saída do módulo para leitura externa, junto ao sinal avisando o fim do processamento. O processo se repete para o processamento de outra imagem a partir da inicialização das variáveis, já que os pesos e vetores de suporte não sofrem alterações. O principal ganho na implementação dessa forma se comparado com a implementação em *software* está no cálculo das camadas 2 e 3, já que todos os valores são calculados paralelamente, enquanto em *software* no máximo alguns valores seriam calculados dessa maneira.

O classificador implementado consegue operar a um *clock* máximo de 27,7 Mhz, obtido por meio da ferramenta de análise de tempo embutida no *software* de desenvolvimento *Quartus II* da Altera. Como para processar uma imagem é necessário percorrer 196 estados da máquina de estados na primeira execução ou 193 estados da segunda execução em diante e cada estado leva um pulso de *clock,* é possível processar mais de 14 mil imagens por segundo, um ganho de aproximadamente 75% em relação a sua versão em *software*.

O circuito implementado utilizou 20705 LUTs adaptativas e 3013 registradores, o que significa 45% da lógica do FPGA utilizado, segundo o relatório fornecido pelo *Quartus II*. Como menos da metade da lógica disponível no FPGA foi utilizada, sobra espaço para a implementação de outros módulos em conjunto, já que os filtros de pré-processamento já desenvolvidos [Nardo, 2012] [Alexandre, 2012], em geral, utilizam pouca lógica.

Quanto ao consumo de energia, o módulo dissipa apenas 1,195W, segundo estimativas das ferramentas de projeto. Para efeitos de comparação, a potência total dissipada por um processador com arquitetura *Ivy Bridge* da Intel é de 17W [ExtremeTech, 2012], nos modelos mais simples, sem contar os outros elementos, como placa mãe, e memória. Portanto, o desempenho do módulo implementado supera sem problemas os requisitos para processar as imagens em tempo real, consumindo pouca energia.

5.3 Verificação funcional

Verificação é um processo usado para identificar se a ideia original de um projeto foi preservada após sua execução. O processo de verificação está presente no dia-a-dia das pessoas, como quando se prova um prato enquanto ele é preparado para garantir que seu tempero está apropriado. A verificação de um projeto pode ser feita por meio de *testbenches*. O termo *testbench* refere-se a um código de simulação usado para criar estímulos de entrada para um projeto sob verificação e observar sua resposta. O desafio da verificação é determinar um padrão de estímulos e o que é esperado como saída de um projeto que funciona adequadamente [Bergeron, 2003].

Ao examinar livros sobre linguagens como VHDL e Verilog, constata-se que a maior parte deles é dedicada a explicar a sintaxe e semântica das linguagens. Além disso, apenas uma pequena porção dos livros é dedicada aos *testbenches*. Partindo desse cenário parece que a verificação de projetos não é considerada de grande importância, porém isso não é verdade. Hoje, mais de 70% dos esforços de desenvolvimento de projetos de *hardware* são dedicados à verificação. Equipes de projetistas possuem engenheiros dedicados unicamente à verificação [Bergeron, 2003].

O fato da verificação demandar tantos esforços, a escassez de tempo para o projeto de *hardware* qualificado e de engenheiros de verificação, e a quantidade de código que precisa ser produzida torna a verificação um elemento crítico. Para aumentar o problema, a verificação costuma ser deixada para o final do projeto, quando os prazos de entrega estão no limite [Bergeron, 2003]. Técnicas para a execução da verificação em paralelo com o desenvolvimento do projeto e ferramentas que auxiliam na elaboração de *testbenchs* [Pessoa, 2007] diminuem o tempo total gasto com a verificação.

A metodologia proposta por Silva [Silva, 2007] e aperfeiçoada por Oliveira [Oliveira, 2010], propõe uma sequência de etapas para elaboração e um ambiente de verificação que minimize as chances de erros passarem despercebidos. Esse ambiente está esquematizado na Figura 5.4.

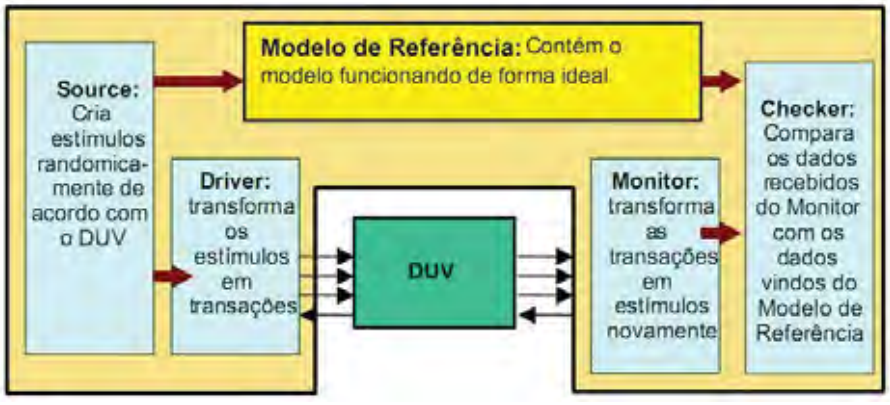


Figura 5.4: Ambiente de verificação funcional.

As funções de cada um dos módulos do ambiente podem ser resumidas da seguinte forma:

- *Source*: é o responsável pela geração de estímulos para simulação. Os estímulos são enviados para o Modelo de Referência e o *Driver* por meio das FIFOs (*First-in First-out*), responsáveis por controlar o sequenciamento e o sincronismo das transações. Os estímulos criados devem exercitar todas as funcionalidades especificadas para saber se as respostas do *Design Under Verification* (DUV) estão de acordo com o esperado.

- *Driver*: sua função é converter os estímulos enviados pelo *source* em sinais que serão submetidos ao DUV. Além disso, ele executa o protocolo de comunicação com o DUV.

- *Monitor*: sua função é a contrária do *driver*, ou seja, converter os sinais de saída do DUV em transações que serão repassadas ao *checker*. De maneira similar ao *driver*, ele também executa um protocolo de comunicação com o DUV.

- *Checker*: Sua função é comparar as respostas obtidas com os estímulos enviados ao Modelo de Referência com as respostas do DUV. Essa comparação é feita de maneira automática e, caso seja encontrada alguma diferença, ele se manifesta por meio de uma mensagem de erro.

- DUV: é o projeto a ser verificado. É implementado em nível RTL, utilizando uma HDL. No caso desse trabalho, o DUV é o classificador implementado em *system verilog*.

- Modelo de Referência: é a implementação ideal do sistema, que pode ser feita em alguma linguagem de alto nível. O que se espera desse módulo é que produza as respostas corretas a partir dos estímulos gerados pelo *source*. Essas respostas são enviadas para o *checker* para posterior verificação.

O ambiente criado para a verificação funcional do classificador SVM implementado é semelhante ao descrito. Analogamente, há uma estrutura para geração dos estímulos, de maneira aleatória ou a partir de valores de textura extraídos das imagens reais. Os valores gerados são enviados simultaneamente para um módulo implementado e para um modelo de referência implementado também em *system verilog*, mas em nível de descrição comportamental, como em uma

linguagem de paradigma estruturado. Após o processamento dos dados, as saídas são comparadas para verificar a compatibilidade. Caso haja divergência, a simulação é interrompida com um sinal de erro, senão, a simulação prossegue pelo número de iterações que for determinado.

Foram realizados três tipos de testes. O primeiro teste consistiu em utilizar as mesmas imagens que foram nos testes em *software* e verificar se o classificador em *hardware* se comportava da mesma forma. No segundo testes foram escolhidos valores arbitrários para os sinais de entrada para testar situações específicas ou absurdas para o sistema. Por exemplo, testar se o *reset* faz o circuito retornar para o estado inicial a partir de qualquer estado da máquina de estados ou então a entrada de valores que não são esperados, como números negativos ou fora dos limites possíveis para as informações de textura extraídas, e também para verificar se todos os estados da máquina de estado são percorridos. No total foram executados 42 vetores de testes para verificar essas situações. O último tipo de teste foi o aleatório, que visa cobrir o máximo de situações possíveis. Para isso foram gerados 10 milhões de vetores de entrada nos intervalos esperados possíveis para os valores da textura extraída. Os testes executados atingiram seu objetivo para a verificação, uma vez que foram apontados erros que teriam passado despercebidos. Após as devidas correções realizadas, todos os testes foram executados sem que nenhuma diferença entre as saídas do circuito implementado e as obtidas pelo modelo de referência fossem encontradas.

5.4 Considerações finais

Desenvolver um projeto em *hardware* é uma tarefa cheia de nuances. Além de escolher a melhor alternativa para o problema em questão, existem diversos cuidados necessários para garantir que ele atinja seus objetivos. Apesar disso, os ganhos em desempenho e consumo que um circuito específico para um problema proporciona compensam as dificuldades, ainda mais em aplicações onde esses ganhos são necessários.

Capítulo 6 – Conclusões

6.1 Conclusões

O trabalho desenvolvido e exposto nesta dissertação visou abordar as principais técnicas de aprendizado de máquina para o desenvolvimento de um sistema inteligente eficiente na detecção de plantas daninhas em plantações de soja. Para isso foram estabelecidos requisitos para a aquisição, pré-processamento e extração de características de imagens de plantações de modo que o domínio do problema pudesse ser explorado por uma solução em *hardware*. Com os testes realizados foi observada a vantagem da extração da textura das imagens, expressa em sua dimensão fractal, em relação à extração da energia delas. Além disso, foi nítido o ganho em eficácia conseguido ao utilizar o *kernel* de intersecção por histograma na implementação do SVM. Apesar disso, não foi possível realizar comparações mais profundas com a implementação da RNA, já que em nenhum dos testes o treinamento conseguiu convergir satisfatoriamente para um conjunto de pesos ideal.

A implementação no FPGA se mostrou uma escolha adequada dado o ganho de desempenho de cerca de 75% obtido com o processamento paralelo realizado pela

SVM. Além disso o circuito do classificador se mostrou de complexidade não muito elevada o que colaborou com um baixo consumo de energia.

6.2 Trabalhos futuros

Esse trabalho deixa em aberto algumas questões que podem ser exploradas em trabalhos futuros. A mais evidente delas é determinar com exatidão o motivo pelo qual a RNA não conseguiu lidar com o problema nas alternativas testadas. Para isso é possível buscar outras maneiras de extrair informações das imagens de maneira a ajudar na separação das classes pela RNA. Além disso, o estudo de outras arquiteturas e formas de treinamento que sejam mais específicas para o problema pode gerar resultados melhores. Por fim, a expansão dos testes realizados com um conjunto de imagens maior e a solução do mesmo problema em outros tipos de plantio pode tornar esse sistema mais robusto.

Referências bibliográficas

ALBUQUERQUE, M. P.; ALBUQUERQUE, M. P. Processamento de Imagens: Métodos e Análise. **Revista de Ciência e Tecnologia**, v. 1, n. 1, p. 10-20, 2000.

ALEXANDRE, J. M. **Implementação de filtros para processamento de imagens digitalizadas com vistas a análises foliares em plantações**. 2012. 104f. Trabalho de conclusão de curso – Universidade Estadual Paulista, São José do Rio Preto, SP, 2012.

ALTERA. 2005. **NIOS Development Board Reference Manual, Stratix II Edition**. Altera. Disponível em <www.altera.com/literature/manual/mnl_nios2_board_stratixII_2s60.pdf>. Acesso em: Novembro de 2011.

BARLA, A.; ODONE, F.; VERRI, A. **Histogram Intersection Kernel For Image Classification**. Image Processing. ICIP 2003. Proceedings. 2003 International Conference on, 2003. v. 2 p.III - 513-16.

BASTOS, V.P. **Técnicas de Segmentação de Imagens Para Recuperação de Informações Visuais**. Universidade Católica de Pelotas, Pelotas-RS, 2008. Disponível em <http://paginas.ucpel.tche.br/~vbastos/index.htm>. Acesso em: Novembro de 2010.

BERGERON, J. **Writing Testbenches – Functional Verification of HDL Models**. 2º Edição, Boston: Kluwer Academic Publishers, 2003. 475 p.

EXTREMETECH. 2012. **Intel unleashes dual-core Ivy Bridge chips, updated ultrabooks incoming**. Disponível em: <http://www.extremetech.com/computing/130154-intel-dual-core-ivy-bridge>. Acesso em: Julho de 2012.

GONZALEZ, J. A. Q.; CHAU, W. J. **Uma metodologia de projetos para circuitos com reconfiguração dinâmica de harware aplicada a support vector machines**.

2006. 152f. Tese (Doutorado em Engenharia) – Escola Politécnica, Universidade de São Paulo. São Paulo.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 2ª Ed. Nova Jersey: Prentice Hall, 2002. 793 p.

HAYKIN, S. S. **Neural Networks: A Comprehensive Foundation**. 2ª Ed. Prentice Hall, 1999. 842 p.

HAYKIN, S. S. **Redes Neurais, Princípios e prática**. 2ª Ed. [S.l.]: Bookman, 1999.

HERNANDEZ, R. A.; STRUM, M. **MP-SMO: um algoritmo para a implementação VLSI do treinamento de máquinas de vetores de suporte**. 2009. 113f. Dissertação (Mestrado em Engenharia) – Escola Politécnica, da Universidade de São Paulo. São Paulo.

LEAL, R. P. **Sistemas Digitais**. Material de aula. Universidade de Pernambuco. Departamento de Engenharia Elétrica, 1998. Disponível em: <http://www.fortunecity.com/bally/donegal/89/sistaula4_98_2.htm>. Acesso em: Dezembro de 2010.

LORENA, A. C; DE CARVALHO, A. C. P. L. F. 2007. Uma introdução às Support Vector Machines. **Revista de Informática Teórica e Aplicada**. v. XIV. n. 2, p. 43-67, 2007.

MARTINS, C. A. P. S.; ORDONEZ, E. D. M.; CORRÊA, J. B. T.; CARVALHO, M. B. **Computação Reconfigurável: conceitos, tendências e aplicações**. XXII Jornada de Atualização em Informática (JAI) - Livro Texto. 22ª Ed. Campinas: Sociedade Brasileira de Computação, 2003, v. XXII, p. 339-388.

MERTES, J. G.; MARRANGHELLO, N.; PEREIRA, A. S. **Implementation of Filters for Image Pre-processing for Leaf Analyses in Plantations**. International Conference on Computational Science (ICCS), 2008, Kraków. LNCS – Computational Science – ICCS 2008 – Proceedings – Part II. Heidelberg : Springer-Verlag, 2008 v. 5102. p. 153-162.

NARDO, L. A. S. **Implementação de filtros para processamento de imagens digitalizadas para análises foliares em plantações**. 2012. 104f. Trabalho de

conclusão de curso – Universidade Estadual Paulista, São José do Rio Preto, SP, 2012.

OLIVEIRA, H. F. A. **BVM: Reformulação da metodologia de verificação funcional VeriSC**. 2010. 139f. Dissertação – Universidade Federal de Campina Grande, Campina Grande, PB, 2010.

OPREA, S.; LITA, I.; JURIANU, M.; VISAN, D.A; CIOC, I.B. 2008. **Digital image processing applied in drugs industry for detection of broken aspirin tablets. Electronics Technology**. ISSE '08. 31st International Spring Seminar. p. 121-124.

PESSOA, I. M. **Geração semi-automática de Testbenches para circuitos integrados**. 2007. 67 f. Dissertação– Universidade Federal de Campina Grande, Campina Grande, PB, 2007.

ROSA, C. A. 2008. **Verilog.** IC-Brazil National Program. Disponível em: <http://icbrazil.carlos-rosa.com/verilog> Acesso em: Novembro de 2010.

SILVA, D. D. C. **Desenvolvimento de um IP Core de pré-processamento digital de sinal de voz para aplicação em sistemas embutidos**. 2006. 108f. Dissertação – Universidade Federal de Campina Grande. Campina Grande.

SILVA, K. R. G. **Uma metodologia de Verificação Funcional para circuitos digitais**. 2007. 133f. Tese – Universidade Federal de Campina Grande, Campina Grande, PB, 2007.

SUPRIJANTO AYU, D.; NADHIRA, V.; DARIJANTO, S.T. **Development of image processing for digital dermatoscopy**. Instrumentation, Communications, Information Technology, and Biomedical Engineering (ICICI-BME), 2009 International Conference. p. 1-5.

TANG, Y.; JIN, B.; SUN, Y.; ZHANG, Y. **Granular support vector machines for medical binary classification problems**. Computational Intelligence in Bioinformatics and Computational Biology, 2004. CIBCB '04. Proceedings of the 2004 IEEE Symposium. p. 73-78.

TATIBANA, C. Y.; KAETSU, D. Y. 1996. **Redes Neurais**. Universidade Estadual de Maringá. Departamento de Informática. Disponível em: <http://www.din.uem.br/ia/neurais/>. Acesso em: Julho de 2011.

WESTE, N.; HARRIS, D. **CMOS VLSI Design: A Circuits and Systems Perspective**. 3 Ed. Addison-Wesley, 2004. 800 p.

YUHUI, Y. Y. Z. **Technique of Measuring Leading Vehicle Distance Based on Digital Image Processing Theory**. Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference. p. 674-677.

ZHI-PING, L. L.; DING, G. W. **Fuzzy Multi-class Support Vector Machine Based on Binary Tree in Network Intrusion Detection**. Electrical and Control Engineering (ICECE), 2010 International Conference. p. 1043-1046.